- Accessing data from matlab

Accessing data from matlab

Last edited by **Diego Torres** 2 weeks ago
New page Page history Edit

Accessing data from MATLAB is done through a single primary function. This primary function accesses relevant data.

Structures

A method's name is stored in a 1x1 MATLAB struct. This struct contains all of functionality offered by the HiGRID Database System. The name of this structure is called 'DB'. To access different fields from 'DB' the dot operator is used in an example shown below

- DB.getLocalWindTable

Primary Data Retrieval Function

A single MATLAB function is used to interface a developer of HiGRID and the Database. This 'getData' function allows specific commands and arguments to grab relevant data. The structure of this function is as shown

- getData(*func_name*,{*string_arg1*,*string_arg2*,...})

In the example below the getData function is used as follows,

getData(DB.getLocationStreamTable,{'tehachapi'}) In this example the getData function retrieves the excel Location_STREAM_hour for different locations. In this case the table data is retrieved for the location 'tehachapi'.

A log of different commands, with the corresponding function signatures and their use are kept here.

Database Structure Commands

'`func_name`', '`func_params`', '`description`'

- '**getLocationStreamTable**','STRING location','returns the data of the table associated with the excel files STREAM_hour from the specified location',

- '**getLocationStreamPowerPerTurbineBlock**','STRING location,STRING turbineBlocks','returns the data of the table associated with the excel files STREAM_hour from the specified location and turbine block count',

- '**getSolarThermalProfileWNETScaledTableFromTime**','STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time',

- '**getSolarThermalProfileGasScaledTableFromTime**','STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time',

- '**getSolarThermalProfileNoAuxWNETScaledTableFromTime**','STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time',

- '**getSolarThermalProfileWNETScaledFromCityAndTime**','STRING city,STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time and the city',

- '**getSolarThermalProfileGasScaledFromCityAndTime**','STRING city,STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time and the city',

- '**getSolarThermalProfileNoAuxWNETScaledFromCityAndTime**','STRING city,STRING time','returns the data of the table associated with the excel Central_Solar_Thermal_Profiles from the specified time and the city',

- '**getReserveStorageHistoricalTable**','NO ARGS','return the data from the table associated with the Res_Storage excel files',

- '**getSCERateMatrixTableFromGeneralService**','STRING gs_sheetname','returns the data from the table associated with SCE_RateMatrix2 excel files from the specified general sheet',

- '**getSCERateMatrixTableFromNonTOU**','NO ARGS','returns the data from the table associated with the SCE_RateMatrix2 excel files from non TOU',

- '**getLocalWindTable**','NO ARGS','returns the data from the table associated with the localwind_hour_lowcutin2 excel file',

- '**getLocalWindFromTurbine**','STRING turbine_count','returns the data from the table associated with the localwind_hour_lowcutin2 excel file from the specified turbine count',
- '**getLocalWindTableFromTurbineAndTime**','STRING turbine_begin, STRING turbine_end, STRING time_begin, STRING time_end','returns the data from table associated with localwind_hour_lowcutin2 excel file from the subtable specified by the range arguments',
- '**getTwoAxisTrackingProfileTable**','NO ARGS','returns the data from the table associated with 2ax_norm excel file',
- '**getTwoAxisTrackingProfileFromCity**','STRING city','returns the data from the table associated with 2ax_norm excel file from the specified city',
- '**getAzimuthProfileTable**','NO ARGS','returns the data from the table associated with the azitrack_norm excel file',
- '**getAzimuthProfileFromCity**','STRING city','returns the data from the table associated with the azitrack_norm excel file from the specified city',
- '**getFixedNormProfileTable**','NO ARGS','returns the data from the the table associated with the fixed_norm excel file',
- '**getFixedNormProfileFromCity**','STRING city','returns the data from the the table associated with the fixed_norm excel file from the specified city',
- '**getFERC_PK_LF_BASE_REN_TotTableFromYear**','STRING year','returns the data from the table associated with the FERC_PK_LF_BASE_HYDRO_REN_Tot_year_small excel file from specified year',
- '**getFERC_PK_LF_Fleet_HourlyOperationTableFromYear**','STRING year','returns the data from the table associated with the FERC_year__PK_LF_fleet_hourly_operation excel file from specified year',
- '**getFERC_PK_LF_BASE_REN_TotHeaderFromYear**','STRING year','returns the header from the table associated with the FERC_PK_LF_BASE_HYDRO_REN_Tot_year_small excel file from specified year',
- '**getFERC_PK_LF_Fleet_HourlyOperationHeaderFromYear**','STRING year','returns the header from the table associated with the FERC_year__PK_LF_fleet_hourly_operation excel file from specified year',
- '**getUCIEndUseModelTotalsTable**','NO ARGS','returns the data from the table associated with the uci_end_use_model_totals excel file',
- '**getUCIEndUseModelTotalsHeader**','NO ARGS','returns the header from the table associated with the uci_end_use_model_totals excel file',
- '**getUCIEndUseModelTotalsEETable**','NO ARGS','returns the data from the table associated with the uci_end_use_model_totals_ee excel file',
- '**getUCIEndUseModelTotalsEEHeader**','NO ARGS','returns the header from the table associated with the uci_end_use_model_totals_ee excel file',
- '**getCAISOInstateDemandAndSPScaledTableFromYear**','STRING year','return the data from the table associated with the CAISO_Year_instate_demand_&*SP*_scaled excel file from specified year',
- '**getCAISOInstateDemandAndSPRegScaledTableFromYear**','STRING year','return the data from the table associated with the CAISO_2005_instate_demand_SP_Reg_scaled excel file from specified year',
- '**getCAISOInstateDemandAndSPRegScaledPUMPTableFromYear**','STRING year','return the data from the table associated with the CAISO_2005_instate_demand_SP_Reg_scale-PUMP excel from the specified year',
- '**getCAISOIOULoadAndSPRURDTableFromYear**','STRING year','return data from the table associated with the CAISO_2005_IOU__Load_SP_RU_RD excel file from the specified year',
- '**getCAISOIOULoadAndSPTableFromYear**','STRING year','return the data from the table associated with the CAISO_2005_IOU_Load_&_SP excel from the specified year',
- '**getCAISOStateGenAndSPPUMPRemovedTableFromYear**','STRING year','return the data from the table associated with the CAISO_2005_state_gen_&_SP_pump_removed UNSURE OF ORIGIN excel from the specified year',
- '**getCAISOLoadTableFromYear**','STRING year','return the data from the table associated with the caiso_2010_load excel from the specified year',
- '**getCAISOInstateDemandAndSPScaledHeaderFromYear**','STRING year','return the header from the table associated with the CAISO_Year_instate_demand_&*SP*_scaled excel file from specified year',
- '**getCAISOInstateDemandAndSPRegScaledHeaderFromYear**','STRING year','return the header from the table associated with the CAISO_2005_instate_demand_SP_Reg_scaled excel file from specified year',

- **'getCAISOInstateDemandAndSPRegScaledPUMPHeaderFromYear'**,'STRING year','return the header from the table associated with the CAISO_2005_instate_demand_SP_Reg_scale-PUMP excel from the specified year',
- **'getCAISOIOULoadAndSPRURDHeaderFromYear'**,'STRING year','return header from the table associated with the CAISO_2005_IOU__Load_SP_RU_RD excel file from the specified year',
- **'getCAISOIOULoadAndSPHeaderFromYear'**,'STRING year','return the header from the table associated with the CAISO_2005_IOU_Load_&_SP excel from the specified year',
- **'getCAISOStateGenAndSPPUMPRemovedHeaderFromYear'**,'STRING year','return the header from the table associated with the CAISO_2005_state_gen_&_SP_pump_removed UNSURE OF ORIGIN excel from the specified year',
- **'getCAISOLoadHeaderFromYear'**,'STRING year','return the header from the table associated with the caiso_2010_load excel from the specified year',
- **'getUCICampusDataTableFromYear'**,'STRING year','return the data from the table associated with the UCI_campus_data_2005 excel file from the specified year',
- **'getUCICampusDataHeaderFromYear'**,'STRING year','return the header from the table associated with the UCI_campus_data_2005 excel file from the specified year',
- **'getOtherDemandSPAndRegTableFromYear'**,'STRING year','return the data from the table associated with the Other_demand_2005_SP_Reg excel file from the specified year',
- **'getOtherDemandSPAndRegHeaderFromYear'**,'STRING year','return the header from the table associated with the Other_demand_2005_SP_Reg excel file from the specified year',
- **'getDryBulbTemperaturesColumnFromYear'**,'STRING year, STRING city','return the data from column associated with the 2005_santa_ana_full_noaa_weather_v2 excel file from the specified year and city',
- **'getWetBulbTemperatureColumnFromYear'**,'STRING year, STRING city','return the data from column associated with the 2005_santa_ana_full_noaa_weather_v2 excel file from the specified year and city',
- **'getDewPointTemperatureColumnFromYear'**,'STRING year, STRING city','return the data from column associated with the 2005_santa_ana_full_noaa_weather_v2 excel file from the specified year and city',
- **'getCEUSProfileTemperaturesTable'**,'NO ARGS','return the data from the table associated with the NOAA_temperatures_for_CEUS_code excel file',
- **'getCEUSProfileTemperaturesHeader'**,'NO ARGS','return the data from the table associated with the NOAA_temperatures_for_CEUS_code excel file',
- **'getCEUSProfileTemperatureColumnFromSector'**,'NO ARGS','return the data from the table associated with the NOAA_temperatures_for_CEUS_code excel file',
- **'getCostModuleRawDataTextTable'**,'STRING extension','return the data,raw data, and text data from the table associated with the cost_module excel files from the specified cost_module version provided by the extension',
- **'getH2DemandTableFromYear'**,'STRING year','return data from the table from the associated H2 demand_CA_AllVMT excel file from the specified year',
- **'getH2DemandGasolineProfileFromYear'**,'STRING year','return column data from the associated H2 demand_CA_AllVMT excel file from the specified year',
- **'getH2DemandUCIProfileFromYear'**,'STRING year','return column data from the associated H2 demand_CA_AllVMT excel file from the specified year',
- **'getH2ElectrolyzerMassFlowRate'**,'NO ARGS','return the header from the table from the associated h2_elec_table excel file',
- **'getH2ElectrolyzerWetBulbTemperatures'**,'NO ARGS','return the WetBulb temperatures from the table associated h2_elec_table excel file',
- **'getH2ElectrolyzerPower'**,'NO ARGS','return the electrolyzer power from the table associated with the h2_elec_table excel file excel file',
- **'getExcelTableFromMySQL'**,'STRING filename','return the header + table data from the specified filename',
- **'LoadWorkSpaceFromMySQL'**,'STRING filename','return the directory path from the specified mat file',
- **'LoadWorkSpaceQuery'**,'STRING filename','return data to workspace from the specified mat file'
-

- **Higrid database system**

Higrid database system

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

HiGRID Database System

The HiGRID Database System is used to access a pool of centralized information necessary to the operation of HiGRID for increased easy of use, demand, and organization. This platform incorporates Python, SQL, and MATLAB code to create a seamless database environment for MATLAB users. A MySQLdb stores all information on excel sheets used.

Table Of Contents

The Pipeline Outline

This data retrieval process has 3 core components(HiGRID, Python, MySQL). The flow of the 3 components are HiGRID to Python to MySQL to Python to HiGRID.

HiGRID to Python

GetData Functionality

HiGRID uses information supplied by the user in the form of the getData function. The getData function processes the parameters passed into the getData function so that they can be understood by MATLAB and Python. MATLAB performs a system call on using the function name and the function arguments to form a systemCommand string to execute the Python code. This called using the following line.

output = evalc('system(systemCommand)')

The output is the output stream of the python code executed.

Python to MySQL to Python

Once the Python code is executed by the system call, the MySQLdb is queried for the desired data. The sql commmand that is sent to the database depends on the data requested. The resulting data is then formatted into the output stream to be properly interpreted by MATLAB. The generic output stream will look like

*-start

-table

-data

-end*

MySQLWorkbench

The start and end terms are wrapped around the data to indicated that the information in the output stream was requested. The second term, which can be table, column, multitable, multicolumn, raw, numpytable, numpytabletranspose indicate how the data should be parsed and packaged to the desired MATLAB datatype. The third term, data, is the actual data that has been formatted by python and needs to be converted.

Python to MATLAB

After the python code has executed, the output stream is evaluated within MATLAB and segments of data are processed. After the data has been interpreted it will either be returned or placed directly in the MATLAB workspace.

higrid database diagram

This pipeline is designed to include additional data and scripts if need be.

Additional Notes

- Latency between database and MATLAB
    - There is considerable latency when using a database to store the information that is required. While data retrieval time increases roughly 1 additional second, if the number of requests to the server increases then a large chunk of the time is spent reconnecting to the database. Optimal solutions to solutions to this problem would be to preload every file needed for HiGRID, so that the information doesn't need to be retrieved every time the software is run. A separate MATLAB script to handle a large chunk of the database calls would alleviate some of latency.
    - While designing additional tables for the 'higrid' database, it is important to take note of the data type that is used for field. The size of the datatype increases the time spent sending the data to HiGRID. If a larger data type can be reduced, without sacrificing the accuracy of the data, change the data type of the field.

Clone repository

- Accessing data from matlab
- Higrid database system
- Mysql database
- Python: adding new database content
- Generic_sqlcommand
- Matlab to python
- Python to matlab
- Python to mysql to python

More Pages
×

New Wiki Page

- [Mysql database]

Mysql database

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

The database that will be used to store the xls and mat files is based in a MySQL database which a very reliable and secure database management system. MySQL's load balancing optimization software provides sufficient speeds for the demands of HiGRID developers.

There are two databases used for HiGRID, *higrid* and *matfilesdb*. The higrid database contains the sheet data for each Excel file that is used. While the matfilesdb database contains the variable data for each mat file that is used. The separation of both databases is for user organization.

higrid

Inserting A New Table To insert a new table into the database, MySQL Workbench provides an import tool, which converts CSV files or JSON files into standard tables. The list of available tables are based on the available Excel Files.

Steps

The Table Data Import Wizard Tool is found in the Navigator Pane, Schemas Section.

Select the CSV or JSON that you would like to convert

Choose the appropriate data types for the each field

In addition there exists a directory table in the *higrid* database, *matfiles* and *matdirectory*. These tables contain the pathways of the files, located within the networked file system. In our case, the Grab Bag.

To add a new pathway

*"insert into matdirectory (`directory`,matpathway) values (*int,absolute_path*); "* The *int* used must be unique.

To add a new file

insert into matfiles (filename,`directory`) values (*filename*,*int*); The *int* is associated with matdirectory table path

matfilesdb

The matfilesdb database is used to store the matfiles data tables. This database is useful for remote access to mat files.

To insert the matfile variable table data use the the file autoLoadMatfile.py.

To use the autoLoadMatfile the variable *rootDir* need to be changed. The rootDir variable is the folder location of the matfiles that will be uploaded to the server. Database configuration values can be changed to move tables into a different database.

To run the auto loader make sure that the specified imports are installed. In this example I am using the Anaconda Prompt. I am providing the absolute directory to the folder of the autoLoaderMatfile.py

All mat files within the folder will be uploaded to the database. If an existing table with the same name is present, then the script will overwrite the existing table.

Common Errors For the Import Data Wizard Tool For CSV Files

The name of each field in the CSV must be smaller the 64 characters. The number of columns in the CSV must be relatively small. No null values that are assigned as the first record in the table. Inconsistencies in the type of each record. Common CSV errors like, too many commas, the field may contain a commas,

Common Errors For the autoLoadMatfile

The name of each field in the CSV must be smaller the 64 characters. The number of columns in the CSV must be relatively small. In the autoLoader, the script handles large numbers of columns by transposing the table.

New Wiki Page

- **Python: adding new database content**

Python: adding new database content

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

Python files are used to access additional tables in the database. While there exists , a generic function for pulling tables from the higrid database, there exist unique conditions in which that data needs to be further inspected and processed before being delivered to MATLAB. In these cases, it is up to developers to incorporate new python files to handle the new data. These type of python files will be referred to as PyMatSQL Interface.

How to Construct a new PyMatSQL Interface

To construct a PyMatSQL Interface create a new class and import the Generic_SQLCommand module Add methods to the class. These methods will be referenced later in MATLAB. The example below is for a new library table that has just been added into higrid.

The Generic_SQLCommand object is the most important part of creating your own PyMatSQL Interface. This object contains many useful functions for querying data, getting SQL commands, and formatting data for the output stream.

gs.getSQLCommand - *A convenient way to access generic SQL commands that are repeatedly needed throughout the coding process. These commands are used to query data from the database and follow the MySQL semantics. *

gs.makePullAndExecute(*command, format_specifier*) - *Allows you to pass the command specified to the MySQL database, execute the statement, and pull the desired contents. *

gs.partitionStartEnd(*data_pulled, format_specifier*) - _Takes the data retrieved, specifies the format, and partitions that the data in the output stream so that it can be recognized and interpreted by MATLAB.

In the new library example, 2 variations of the gs.getSQLCommand are used, getSQLCommandGetTable and getSQLCommandGetColumn. These different commands return the SQL command for a table and a specific column from that table.

These 3 basic functions allow the complete process of data retrieval and packaging from the database. Additional functionality can easily be incorporated into Generic_SQLCommand object.

However depending on the data that is returned from the query further processing may need to be done.

Naming Conventions of New Classes To keep with simplicity, the names of the classes should reflect the name of the tables that will be processed by the database. The subsequent names of the methods within the class should also be named with respect to the specific form and type of information related to the class. For standard form, start each method off with 'get' followed by the name of the data, followed then by the form of the data 'table','column','row'. While naming of the methods and classes does not have to follow these set of standards it is a good idea to keep a naming convention for the sake of comprehending the use of the function without explicitly saying it. The name of the class and methods does not effect the end results.

With the new class created, two mechanisms need to be able to recognize that the new files and its methods are created. These two mechanisms are FunctionAndObjectData.py and autoCompleteStructure.m.

The FunctionAndObjectData's job is to take the information that was passed from the myParse.py and convert use the function name and function arguements to create an instance of the PyMatSQL Interface class and call the function that is requested.

The FunctionAndObjectData file has four main parts. The import module, *call_function*, *functionDic *dictionary and *switcher *dictionary section.

Import Module Section

The import module section imports the new file as shown below. Add the new file created like so,

*from file_name import **

Call_Function Section

The call_function methods are named according the name of the file that it is used for. Each of these functions handle creating their associated object and calling a function from that object. To add a new file to the list of call_function methods, use the following naming convention call_filename Inside of each function a different object is created. Create the object from the newly created PyMatSQL interface file.

Use the new object with the getattr function. This results in the value of the attribute from the object.

getattr(object, *FUNC_NAME*)

***functionDic *dictionary Section**

This dictionary associates each function name with the function class. Used for mapping. Add a new mapping to the dictionary.

method_name : file_name

***switcher *dictionary Section**

This dictionary is used to map the filename to the appropriate call_function. Used for mapping. Add a new mapping to the dictionary.

file_name : call_func_name

These sections of the *FunctionAndObjectData* file, need to be updated for each new set of data retrieval functionality.

Updating the autoCompleteStructure

The autoComplete Structure is an assistive tool used in MATLAB for helping navigate the various functionalities that offered by auto-completing. Updating this tool is optional but recommended for easy of use for developers of HiGRID. Show below is the struct of the autoCompleteStructure.

To add a new auto-complete entry use the the following template

DB(1).function_name = 'functionname';

To have the autocomplete struct in the matlab workspace, run the autoCompleteStructure.m script.

With both the FunctionAndObjectData and the autoCompleteStructure files updated the your new PyMatSQL Interface has now been implemented for use in MATLAB.

Owen's Theory of Being Wrong: On So Many Levels

So over here we have the documentation for the part about documentation. This program uses Python 2.7 and sadly not Python 3.6+. I know, I know. It sucks, but bare with me here. Now if you turn it off and turn it on again it should work. It will still uses python 2.7, but at least it will work

New Wiki Page

Page slug Tip: You can specify the full path for the new file. We will automatically create any missing directories.

Create page

- **Generic_sqlcommand**

Generic_sqlcommand

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

The Generic_SQLCommand.py file is very useful in simplifying the addition of new tables, as most of the functionality is already prebuilt. There are three main sections to this file; the *makePullAndExecute*, *partitionStartEnd*, and the *getSQLCommand *functions. These are the three primary functions that are used in every python file for HiGRID data retrieval.

Contents

getSQLCommand Section

These set of functions are simple functions for retrieving commonly used segments of SQL for the database. Although these segments are very simple and rudimentary for those that are inexperienced with the SQL it provides a convenient way to query information.

List of available functions here

PartitionStartEnd Section

These set of functions are used to format the data into the output stream so that MATLAB can partition the output stream into the appropriate data sections. Provides a container in the form of text, as well as an identifier tag to associate the send data with the method for interpreting that data.

*List of available cases * here

MakePullAndExecute Section

This function takes the SQL string, queries the data, and returns it to the caller. The format must also be specified. This format converts the queried data into a readable parse-able string or tuple. There are different advantages to this being speed, and efficiency, MATLAB type.

Additional Functionality

The MySQLDB function allows you change the database that is accessed. The default database that is accessed is the *higrid* database. Shown below is the function

- setMySQLdbVariables(*ext_ip*, *port*, *password*, *database_name*,*user_name*)

List of GetSQLCommands

- `getSQLCommandGetTable`*(STRING table_name)* : returns command to query all of the data in the table specified
- `getSQLCommandGetColumn`*(STRING column_name,STRING table_name)* : returns command to query entire column from the specified field from the table specified
- `getSQLCommandGetColumnsAndRows`*(STRING column_names_list,STRING row_begin,STRING* *row_end,STRING table_name) * : returns command to a list table composed of the columns specified by the by the column_names_list, which is a list of column names separated by commas. row_begin and row_end indicate the interval of rows from the specified table.
- `getSQLCommandGetSubTable`*(STRING column_begin,STRING column_end, STRING row_begin,row_end,STRING table_name) * : returns command to query a all of the data specified by the intervals column_begin and column_end which are the names of columns, and the intervals row_begin and row_end which are the indices of the table specified.
- `getSQLCommandColumnNames`*(STRING table_name)* : returns command to query the column names in the table specified

List of PartitionStartEnd Cases

Each case is specified by the format.

- `column` : format single column from a table of one type through. Separate entries using ':'
- `table` : format table . Separate entries of same row using ':'. Separate new rows with 'NL'
- `multitable` : format table for different MATLAB datatypes, including raw, string, and numerical
- `rawtable` : format table to include all information, including headers into a cell array
- `value` : format a single value
- `mattable` : format a matfiletable more efficiently by reducing tuple to string
- `mattabletransform` : format a matfiletable more efficiently by reducing tuple to string and labeling for post-process transpose in MATLAB

New Wiki Page

Matlab to python

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

MATLAB

HiGRID uses information supplied by the user in the form of the getData function. The getData function processes the parameters passed into the getData function so that they can be understood by MATLAB and Python. MATLAB performs a system call on using the function name and the function arguments to form a systemCommand string to execute the Python code. This called using the following lines.

systemCommand = [PYTHON_MAIN_NAME, ' ',FUNCTION_STRING]; *System command string will be sent to the system for execution. It is formed by concatenating the function name and the function arguments.*

output = evalc('system(systemCommand)')* The system is able to call this using the evalc function and the output stream of the executed python code is stored in the output.*

The output is the output stream of the python code executed.

PYTHON

After python has received the function name and the function arguments the information is passed to the myparser.py file. This is where the function name and function arguments are realized as unique information. The unique information is sent to the FunctionAndObjectData.py where function name is then associated with the appropriate function call. The function call is supplied with the function arguments. This is done through mapping specific function names to their associated class names. The myparser.py and the FunctionAndObjectData.py are both responsible for transforming the information passed from MATLAB into a direct and specified function call.

Clone repository

More Pages
×

New Wiki Page

Page slug Tip: You can specify the full path for the new file. We will automatically create any missing directories.
Create page

Python to matlab

Last edited by **Diego Torres** 3 weeks ago
New page Page history Edit

After the python code has executed, the output stream is evaluated within MATLAB and segments of data are processed. After the data has been interpreted it will either be returned or placed directly in the MATLAB workspace.

The communication from Python to MATLAB is facilitated through the output stream. The stream is then interpreted as a list as shown below

C = strsplit(output,'\n');

By iterating through the list, data can be found and appropriately processed by recognizing the format specifiers and partition flags like in the example shown below

start table 1,2,3:4,5,6 end

Once the specifiers and flags are recognized then the action needed to process that specific kind of data is done. Different types of data include, double arrays, cell arrays, single values, or a collection of the former.

As shown in the example above the data is a string, and needs to be parsed in order to convert the string into useful data. Different characters sequences tell the different MATLAB parser scripts how to associate the data into vectors. Once the data has been formatted it is then returned to the user either as a directly returned value or a value placed into the MATLAB workspace.

Clone repository

- Accessing data from matlab
- Higrid database system
- Mysql database
- Python: adding new database content
- Generic_sqlcommand
- Matlab to python
- Python to matlab
- Python to mysql to python

More Pages
×

New Wiki Page

Page slug Tip: You can specify the full path for the new file. We will automatically create any missing directories.
Create page

Python to mysql to python

Last edited by **Diego Torres** 3 weeks ago

Once the Python code is executed by the system call, the MySQLdb is queried for the desired data. The sql commmand that is sent to the database depends on the data requested. The resulting data is then formatted into the output stream to be properly interpreted by MATLAB. The generic output stream will look like

start table data end

The start and end terms are wrapped around the data to indicated that the information in the output stream was requested. The second term, which can be table, column, multitable, multicolumn, raw, numpytable, numpytabletranspose indicate how the data should be parsed and packaged to the desired MATLAB datatype. The third term, data, is the actual data that has been formatted by python and needs to be converted.

Python

The associated python functions are executed from the associated python class. Each Python class contains different functions to interact with the database. The class allows interactions between tables that are related with respect to their usage in HiGRID. Depending on class different functionality is offered. With some classes you can the access the entire raw table,the entire table, a subset of the table, a column, or a single value.

The typical structure of retrieving data is fairly simple. In this example 'gs' is a Generic_SQLCommand object. This object provides some of the basic tools for interacting with any generic MySQL database. Some of these tools are shown below

gs.getSQLCommand - *A convenient way to access generic SQL commands that are repeatedly needed throughout the coding process. These commands are used to query data from the database and follow the MySQL semantics. *

gs.makePullAndExecute(command, format_specifier) - *Allows you to pass the command specified to the MySQL database, execute the statement, and pull the desired contents. *

gs.partitionStartEnd(data_pulled, format_specifier) - *Takes the data retrieved, specifies the format, and partitions that the data in the output stream so that it can be recognized and interpreted by MATLAB.*

These 3 basic functions allow the complete process of data retrieval and packaging from the database. Additional functionality can easily be incorporated into Generic_SQLCommand object.

After the desired code is executed the control of the process returns to MATLAB

Clone repository

- Accessing data from matlab
- Higrid database system
- Mysql database
- Python: adding new database content
- Generic_sqlcommand
- Matlab to python
- Python to matlab
- Python to mysql to python

More Pages
×

New Wiki Page

Page slug [ ] Tip: You can specify the full path for the new file. We will automatically create any missing directories.
Create page