Index

Module variables

Functions

Classes

pyMeasure.Code.DataHandlers.GeneralModels module

Module that contains general data models and functions for handling them

SHOW SOURCE ≡

Module variables

var DEFAULT_FILE_NAME

var METHOD_ALIASES

var NUMBER_MATCH_STRING

var StringTypes

var TESTS_DIRECTORY

Functions

def ascii_data_table_join(column_selector, table_1, table_2)

Given a column selector (name or zero based index) and two tables a data_table with extra columns is returned. The options from table 1 are inherited headers and footers are added, if the tables have a diffferent number of rows problems may occur

SHOW SOURCE ≡

def check_arg_type(arg, arg_type)

Checks argument and prints out a statement if arg is not type

SHOW SOURCE ≡

def collect_inline_comments(list_of_strings, begin_token=None, end_token=None)

Reads a list of strings and returns all of the inline comments in a list. Output form is ['comment',line_number,string_location] returns None if there are none or tokens are set to None

SHOW SOURCE ≡

def convert_all_rows(list_rows, column_types=None)

Converts all the rows (list of strings) in a list of rows using column types

SHOW SOURCE ≡

def convert_row(row_list_strings, column_types=None)

Converts a row list of strings to native python types using a column types list

SHOW SOURCE ≡

def ensure_string(input_object, list_delimiter='', end_if_list='')

Returns a string given an object. If the object is a string just returns it, if it is a list of strings, returns a collapsed version. If is another type of object returns str(object). If all else fails it returns an empty string

SHOW SOURCE ≡

def insert_inline_comment(list_of_strings, comment='', line_number=None, string_position=None, begin_token='(*', end_token='*)')

Inserts an inline comment in a list of strings, location is determined by line_number and string_position

SHOW SOURCE ≡

def line_comment_string(comment, comment_begin=None, comment_end=None)

Creates a comment optionally wrapped with comment_begin and comment_end, meant for a single string comment

SHOW SOURCE ≡

def line_list_comment_string(comment_list, comment_begin=None, comment_end=None, block=False)

Creates a string with each line wrapped in comment_begin and comment_end, by repeatedly calling line_comment_string, or the full string wrapped with block_comment_begin and block_comment_end if block is set to True. Meant to deal with a list of comment strings

SHOW SOURCE ≡

def list_list_to_string(list_lists, data_delimiter=None, row_formatter_string=None, line_begin=None, line_end=None)

Repeatedly calls list to string on each element of a list and string adds the result . ie coverts a list of lists to a string. If line end is None the value defaults to " ", for no seperator use ''

SHOW SOURCE ≡

def list_to_string(row_list, data_delimiter=None, row_formatter_string=None, begin=None, end=None)

Given a list of values returns a string, if row_formatter is specifed it uses it as a template, else uses data delimiter. Inserts data_delimiter between each list element. An optional begin and end wrap the resultant string. (i.e ['1','2','3']-> 'begin+'1'+','+'2'+','+'3'+'end') end defaults to to have nothing at the end use ''

SHOW SOURCE ≡

def parse_lines(string_list, **options)

Default behavior returns a two dimensional list given a list of strings that represent a table.

SHOW SOURCE ≡

def print_comparison(var_1, var_2)

If var_1==var_2 prints True, else Prints false and a string representation of the 2 vars

SHOW SOURCE ≡

def read_schema(file_path, format=None)

Reads in a schema and returns it as a python dictionary, the default format is a single string

SHOW SOURCE ≡

def show_structure_script()

Shows a table elements by substituting the names Explicitly :return: None

SHOW SOURCE ≡

def split_all_rows(row_list, delimiter=None, escape_character=None)

Splits all rows in a list of rows and returns a 2d list

SHOW SOURCE ≡

def split_row(row_string, delimiter=None, escape_character=None, strip=True)

Splits a row given a delimiter, and ignores any delimiters after an escape character returns a list. If the string is unsplit returns a list of length 1

SHOW SOURCE ≡

def string_list_collapse(list_of_strings, string_delimiter='\n')

Makes a list of strings a single string

SHOW SOURCE ≡

def strip_all_line_tokens(string_list, begin_token=None, end_token=None)

Strips all line tokens from a list of strings, meant to reverse the action of line_list_comment_string with block=false

SHOW SOURCE ≡

def strip_begin_end_tokens(string_list, begin_token=None, end_token=None)

Strips out tokens at the begining and ending of a list of strings. Meant to reverse the action of "begin_data_token", etc. This does not work with the end_token's because of where the is.

SHOW SOURCE ≡

def strip_inline_comments(list_of_strings, begin_token='(*', end_token='*)')

Removes inline comments from a list of strings and returns the list of strings

SHOW SOURCE ≡

def strip_line_tokens(string, begin_token=None, end_token=None)

Strips a begin and end token if present from an inputted string, meant to remove line_comments

SHOW SOURCE ≡

def strip_tokens(string_list, *remove_tokens)

Strips all tokens in the list remove_tokens from a list of strings Returns the list with less elements if the tokens contained " ". Now newline characters are returned in the list elements. Meant to reverse the action of adding tokens to a list of strings

SHOW SOURCE ≡

def structure_metadata(header_string, metadata_fact_delimiter=';', metadata_key_value_delimiter='=', comment_character='#')

Strucutre Metadata returns a metadata string and returns a metadata dictionary

SHOW SOURCE ≡

def test_AsciiDataTable()

SHOW SOURCE ≡

def test_AsciiDataTable_equality()

SHOW SOURCE ≡

def test_AsciiDataTable_get_column_and_getitem()

Tests the get_column and **getitem** methods

SHOW SOURCE ≡

def test_add_column()

Tests the add_column method of AsciiDataTable

SHOW SOURCE ≡

def test_add_index()

SHOW SOURCE ≡

def test_add_method()

Tests the **add** method

SHOW SOURCE ≡

def test_add_row()

SHOW SOURCE ≡

def test_change_unit_prefix()

Tests the change_unit_prefix method of the AsciiDataTable Class

SHOW SOURCE ≡

def test_copy_method()

Tests the copy() method

SHOW SOURCE ≡

def test_get_item()

Tests the **add** method

SHOW SOURCE ≡

def test_inline_comments()

SHOW SOURCE ≡

def test_open_existing_AsciiDataTable()

SHOW SOURCE ≡

def test_read_schema()

Tests the read_schema function

SHOW SOURCE ≡

def test_save_schema()

tests the save schema method of the Ascii Data Table

SHOW SOURCE ≡

Classes

class AsciiDataTable

An AsciiDatable is a generalized model of a data table with optional header, column names,rectangular array of data, and footer

SHOW SOURCE ≡

Ancestors (in MRO)

- AsciiDataTable

Instance variables

var elements

var options

Methods

def __init__(self, file_path=None, **options)

Initializes the AsciiDataTable class

SHOW SOURCE ≡

def add_column(self, column_name=None, column_type=None, column_data=None, format_string=None)

Adds a column with column_name, and column_type. If column data is supplied and it's length is the same as data(same number of rows) then it is added, else self.options['empty_character'] is added in each spot in the preceding rows

SHOW SOURCE ≡

def add_index(self)

Adds a column with name index and values that are 0 referenced indices, does nothing if there is already a column with name index, always inserts it at the 0 position

SHOW SOURCE ≡

def add_inline_comment(self, comment='', line_number=None, string_position=None)

Adds an inline in the specified location

SHOW SOURCE ≡

def add_row(self, row_data)

Adds a single row given row_data which can be an ordered list/tuple or a dictionary with column names as keys

SHOW SOURCE ≡

def build_string(self, **temp_options)

Builds a string representation of the data table based on self.options, or temp_options. Passing temp_options does not permanently change the model

SHOW SOURCE ≡

def change_unit_prefix(self, column_selector=None, old_prefix=None, new_prefix=None, unit='Hz')

Changes the prefix of the units of the column specified by column_selector (column name or index) example usage is self.change_unit_prefix(column_selector='Frequency',old_prefix=None,new_prefix='G',unit='Hz') to change a column from Hz to GHz. It updates the data values, column_descriptions, and column_units if they exist, see http://www.nist.gov/pml/wmd/metric/prefixes.cfm for possible prefixes

SHOW SOURCE ≡

def copy(self)

Creates a shallow copy of the data table

SHOW SOURCE ≡

def find_line(self, begin_token)

Finds the first line that has begin token in it

SHOW SOURCE ≡

def get_column(self, column_name=None, column_index=None)

Returns a column as a list given a column name or column index

SHOW SOURCE ≡

def get_column_names_string(self)

Returns the column names as a string using options

SHOW SOURCE ≡

def get_data_dictionary_list(self, use_row_formatter_string=True)

Returns a python list with a row dictionary of form {column_name:data_column}

SHOW SOURCE ≡

def get_data_string(self)

Returns the data as a string

SHOW SOURCE ≡

def get_footer_string(self)

Returns the footer using options in self.options. If block comment is specified, and the footer is a list it will block comment out the footer. If comment_begin and comment_end are specified it will use those to represent each line of the footer. If footer_begin_token and/or footer_end_token are specified it will wrap the footer in those.

SHOW SOURCE ≡

def get_header_string(self)

Returns the header using options in self.options. If block comment is specified, and the header is a list it will block comment out the header. If comment_begin and comment_end are specified it will use those to represent each line of the header. If header_begin_token and/or header_end_token are specified it will wrap the header in those.

SHOW SOURCE ≡

def get_options(self)

Prints the option list

SHOW SOURCE ≡

def get_options_by_element(self, element_name)

returns a dictionary of all the options that have to do with element. Element must be header,column_names,data, or footer

SHOW SOURCE ≡

def get_row(self, row_index=None)

Returns the row as a list specified by row_index

SHOW SOURCE ≡

def is_valid(self)

Returns True if ascii table conforms to its specification given by its own options

SHOW SOURCE ≡

def lines_defined(self)

If begin_line and end_line for all elements that are None are defined returns True

SHOW SOURCE ≡

def move_footer_to_header(self)

Moves the DataTable's footer to the header and updates the model

SHOW SOURCE ≡

def remove_column(self, column_name=None, column_index=None)

Removes the column specified by column_name or column_index and updates the model. The column is removed from column_names, data and if present column_types, column_descriptions and row formatter

SHOW SOURCE ≡

def remove_row(self, row_index)

Removes the row specified by row_index and updates the model. Note index is relative to the data attribute so to remove the first row use row_index=0 and the last data row is row_index=-1

SHOW SOURCE ≡

def save(self, path=None, **temp_options)

" Saves the file, to save in another ascii format specify elements in temp_options, the options specified do not permanently change the object's options. If path is supplied it saves the file to that path otherwise uses the object's attribute path to define the saving location

SHOW SOURCE ≡

def save_schema(self, path=None, format=None)

Saves the tables options as a text file or pickled dictionary (default). If no name is supplied, autonames it and saves

SHOW SOURCE ≡

def structure_metadata(self)

Function that should be overidden by whatever model the datatable has, it only responds with a self.metadata attribute in its base state derrived from self.options["metadata]

SHOW SOURCE ≡

def update_column_names(self)

Update column names adds the value x# for any column that exists in self.data that is not named

SHOW SOURCE ≡

def update_import_options(self, import_table)

Updates the options in the import table

SHOW SOURCE ≡

def update_index(self)

Updates the index column if it exits, otherwise exits quietly

SHOW SOURCE ≡

def update_model(self)

Updates the model after a change has been made. If you add anything to the attributes of the model, or change this updates the values. If the model has an index column it will make sure the numbers are correct. In addition, it will update the options dictionary to reflect added rows, changes in deliminators etc.

SHOW SOURCE ≡

class AsciiDataTableCollection

A collection of multiple AsciiDataTables. The class can be created from a file path with options or can be created without a file path as a empty container.

SHOW SOURCE ≡

Ancestors (in MRO)

- AsciiDataTableCollection

Methods

def __init__(self, file_path=None, table_names=None, **options)

SHOW SOURCE ≡

def build_string(self, **temp_options)

Builds the string for the table collection using the temp_options

SHOW SOURCE ≡

class DataDimensionError

An error associated with a mismatch in data dimensions

SHOW SOURCE ≡

Ancestors (in MRO)

- DataDimensionError
- exceptions.Exception
- exceptions.BaseException
- __builtin__.object

Class variables

var args

var message

class ECPVModel

ECPVmodel is a class that deals with Entity-Context-Property-Value models. An ECPV model is a model for providing a description of an entity. It can be thought of a virtual file system, in which its members have metadata describing them stored for analysis and manipulation

SHOW SOURCE ≡

Ancestors (in MRO)

- ECPVModel

class TypeConversionError

An error in the conversion of rows with provided types

SHOW SOURCE ≡

Ancestors (in MRO)

- TypeConversionError
- exceptions.Exception
- exceptions.BaseException
- __builtin__.object

Class variables

var args

var message