ResInsight Python API - rips

Release 2020.10

Nov 22, 2021

Contents

The ResInsight Python API allows you to interact with a running ResInsight instance from Python 3. This enables you to:

- Start ResInsight from Python.
- Communicate with a running ResInsight instance.
- Load a ResInsight project file.
- Load data files such as Eclipse EGRID files and summary files.
- Extract data to Python for further processing and automation.
- Export snapshots of graphics.
- And lots of other things...

CHAPTER 1

Documentation Sites

Please refer to https://resinsight.org for documentation on the graphical user interface, features and capabilites of **ResInsight**.

- resinsight.org - Documentation for latest stable release
- api.resinsight.org - Documentation of Python API
- beta.resinsight.org - Latest documentation (not yet released)

CHAPTER 2

Contents

2.1 Installation and Configuration

The ResInsight Python API is compatible with Python 3.

2.1.1 Use rips as bundled with ResInsight executable

If you download the ResInsight binaries, the rips package is bundeled with ResInsight and can be used directly from the user interface of ResInsight.

1. Download ResInsight
2. Make sure that Python 3 is installed
3. Make sure the dependencies of rips are installed python -m pip install grpcio protobuf
4. Make sure python executable is available from the ResInsight session - python executable is added to environment variables - full path to python executable is defined in Preferences->Scripting->Python Executable Location

2.1.2 Install rips using package system

As admin user, the necessary Python client package is available for install via the Python PIP package system:

```
python -m pip install rips
```

or as a regular user:

```
python -m pip install --user rips
```

On some systems the *python -m pip* command can be simplified to *pip*.

2.1.3 Usage from within ResInsight

Add your script folder to Scripts From the context menu of a Python script, select Execute Text output is reported in Process Monitor

2.1.4 Troubleshooting

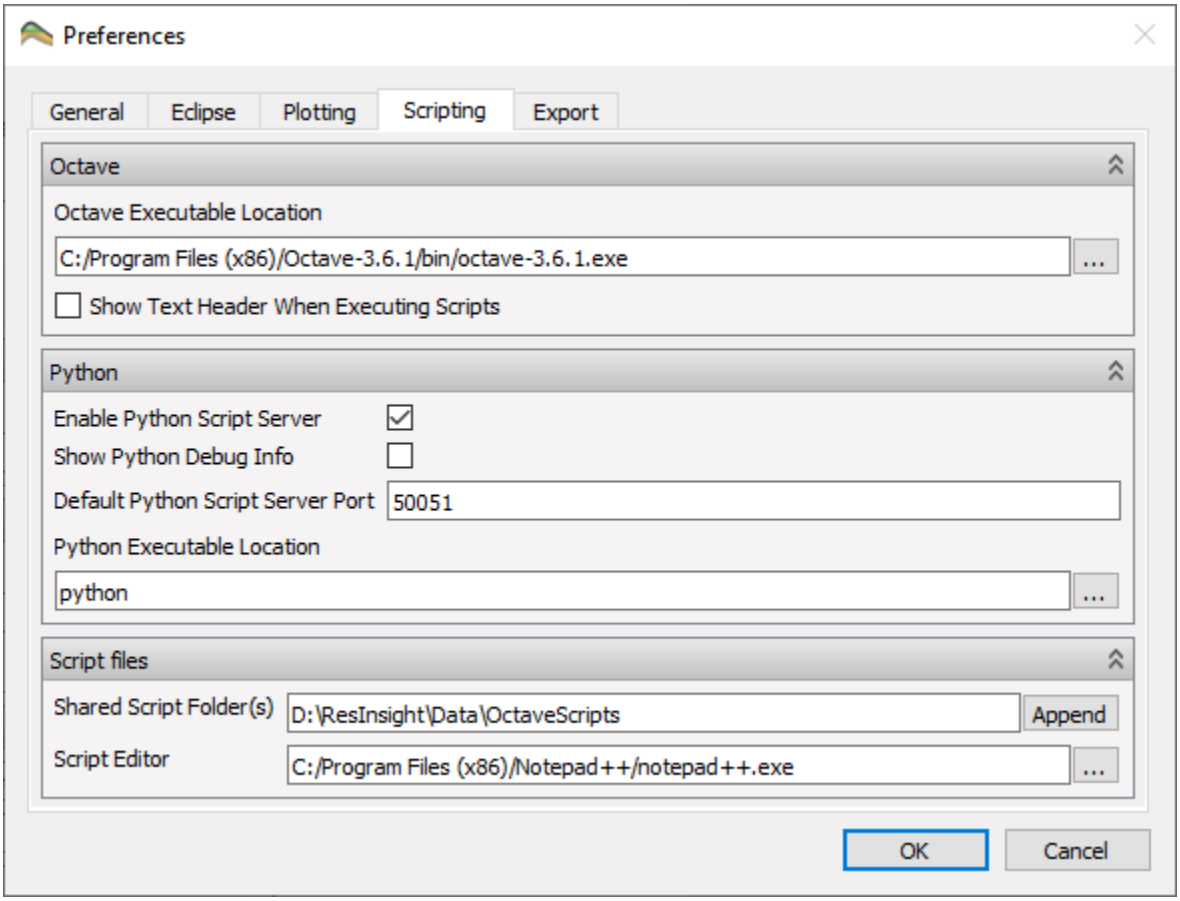
"ModuleNotFoundError: No module named 'grpc' Make sure grpc is installed using

```
python -m pip install grpcio
```

"ModuleNotFoundError: No module named 'google.protobuf' Make sure protobuf is installed using

```
python -m pip install protobuf
```

To configure the **ResInsight Python Script Server**, check *Enable Python Script Server* and verify Python settings in the *Scripting* tab of the ResInsight *Preference* dialog.



The availability of the ResInsight Python Script Server can be confirmed by ResInsight *About* dialog. If unavailable, please consult ResInsight Build Instructions on resinsight.org.

2.2 Main Classes

2.2.1 Case class

Overview

Access to case information. Inherits *PdmObjectBase* and all its members.

Can be accessed with the case() and cases() methods on rips.Project

```
# Import module
import rips
# Connect to running instance
resinsight = rips.Instance.find()
# Get a list of cases
cases = resinsight.project.cases()
```

In practise each object is of the sub-classes rips.EclipseCase and rips.GeoMechCase. See *Project Tree Classes* for a description of them.

API Documentation

class rips.**Case**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

The ResInsight base class for Cases

file_path
Case File Name

Type str

id
Case ID

Type int

name
Case Name

Type str

name_setting
One of [FULL_CASE_NAME, SHORT_CASE_NAME, CUSTOM_NAME]

Type str

active_cell_centers(*porosity_model='MATRIX_MODEL'*)
Get a cell centers for all active cells. Synchronous, so returns a list.

Parameters **porosity_model** (*str*) – string enum. See available()

Returns A list of Vec3d

active_cell_centers_async(*porosity_model='MATRIX_MODEL'*)
Get a cell centers for all active cells. Async, so returns an iterator

Parameters **porosity_model** (*str*) – string enum. See available()

Returns An iterator to a chunk object containing an array of Vec3d values. Loop through the chunks and then the values within the chunk to get all values.

active_cell_corners(*porosity_model='MATRIX_MODEL'*)
Get a cell corners for all active cells. Synchronous, so returns a list.

Arguments: porosity_model(str): string enum. See available()

CellCorner class description:

Parameter	Description	Type
c0		Vec3d
c1		Vec3d
c2		Vec3d
c3		Vec3d
c4		Vec3d
c5		Vec3d
c6		Vec3d
c7		Vec3d

active_cell_corners_async(*porosity_model='MATRIX_MODEL'*)
Get a cell corners for all active cells. Async, so returns an iterator

Parameters **porosity_model** (*str*) – string enum. See available()

Returns An iterator to a chunk object containing an array of CellCorners (which is eight Vec3d values). Loop through the chunks and then the values within the chunk to get all values.

active_cell_property(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all active cells. Sync, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns A list containing double values Loop through the chunks and then the values within the chunk to get all values.

active_cell_property_async(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all active cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

add_new_object(*class_definition*, *child_field=''*)

Create and add an object to the specified child field :param class_definition[class]: Class definition of the object to create :param child_field[str]: The keyword for the field to create a new object in. If empty, the first child array field is used.

Returns The created PdmObject inside the child_field

address()

Get the unique address of the PdmObject

Returns A 64-bit unsigned integer address

ancestor(*class_definition*)

Find the first ancestor that matches the provided class_keyword :param class_definition[class]: A class definition matching the type of class wanted

available_nnc_properties()

Get a list of available NNC properties

NNCConnection class description:

```
Parameter               | Description                                  | Type
------------------------| -------------------------------------------- | ----
↪-
cell_grid_index1        | Reservoir Cell Index to cell 1               |␣
↪int32
cell_grid_index2        | Reservoir Cell Index to cell 2               |␣
↪int32
```

(continues on next page)

(continued from previous page)

```
cell1                   | Reservoir Cell IJK to cell 1                  |␣
↪Vec3i
cell2                   | Reservoir Cell IJK to cell 1                  |␣
↪Vec3i
```

available_properties (*property_type*, *porosity_model='MATRIX_MODEL'*)

Get a list of available properties

For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string corresponding to property_type enum.
- **porosity_model** (*str*) – 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

cell_count (*porosity_model='MATRIX_MODEL'*)

Get a cell count object containing number of active cells and total number of cells

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns active_cell_count: number of active cells reservoir_cell_count: total number of reservoir cells

Return type Cell Count object with the following integer attributes

CellCount class description:

```
Parameter               | Description               | Type
----------------------- | ------------------------- | -----
active_cell_count       | Number of active cells    | Integer
reservoir_cell_count    | Total number of cells     | Integer
```

cell_info_for_active_cells (*porosity_model='MATRIX_MODEL'*)

Get list of cell info objects for current case

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns List of **CellInfo** objects

CellInfo class description:

```
Parameter                 | Description                                 |␣
↪Type
------------------------- | ------------------------------------------- | --
↪---
grid_index                | Index to grid                               |␣
↪Integer
parent_grid_index         | Index to parent grid                        |␣
↪Integer
coarsening_box_index      | Index to coarsening box                     |␣
↪Integer
local_ijk                 | Cell index in IJK directions of local grid  |␣
↪Vec3i
parent_ijk                | Cell index in IJK directions of parent grid |␣
↪Vec3i
```

Vec3i class description:

Parameter	Description	Type
i	I grid index	Integer
j	J grid index	Integer
k	K grid index	Integer

cell_info_for_active_cells_async(*porosity_model='MATRIX_MODEL'*)

Get Stream of cell info objects for current case

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns Stream of **CellInfo** objects

See `rips.case.cell_info_for_active_cells()` for detalis on the **CellInfo** class.

channel()

Private method

children(*child_field*, *class_definition*)

Get a list of all direct project tree children inside the provided child_field :param child_field[str]: A field name

Returns A list of PdmObjects inside the child_field

coarsening_info()

Get a coarsening information for all grids in the case.

Returns A list of CoarseningInfo objects with two Vec3i min and max objects for each entry.

copy_from(*object*)

Copy attribute values from object to self

create_lgr_for_completion(*time_step*, *well_path_names*, *refinement_i*, *refinement_j*, *refinement_k*, *split_type*)

Create a local grid refinement for the completions on the given well paths

Parameters:

Parameter	Description	Type
time_steps	Time step index	Integer
well_path_names	List of well path names	List of Strings
refinement_i	Refinment in x-direction	Integer
refinement_j	Refinment in y-direction	Integer
refinement_k	Refinment in z-direction	Integer
split_type	Defines how to split LGRS	String enum

Enum split_type:

Option	Description
"LGR_PER_CELL"	One LGR for each completed cell
"LGR_PER_COMPLETION"	One LGR for each completion (fracture, perforation, ...)
"LGR_PER_WELL"	One LGR for each well

create_multiple_fractures(*template_id*, *well_path_names*, *min_dist_from_well_td*, *max_fractures_per_well*, *top_layer*, *base_layer*, *spacing*, *action*)

Create Multiple Fractures in one go

Parameters:

Parameter	Description	Type
template_id	Id of the template	Integer
well_path_names	List of well path names	List of Strings
min_dist_from_well_td	Minimum distance from well TD	Double
max_fractures_per_well	Max number of fractures per well	Integer
top_layer	Top grid k-level for fractures	Integer
base_layer	Base grid k-level for fractures	Integer
spacing	Spacing between fractures	Double
action	'APPEND_FRACTURES' or 'REPLACE_FRACTURES'	String enum

create_saturation_pressure_plots()
Create saturation pressure plots for the current case

create_view()
Create a new view in the current case

Returns rips.generated.generated_classes.View

create_well_bore_stability_plot(*well_path*, *time_step*, *parameters=None*)
Create a new well bore stability plot

Parameters

- **well_path** (*str*) – well path name
- **time_step** (*int*) – time step

Returns rips.generated.generated_classes.WellBoreStabilityPlot

days_since_start()
Get a list of decimal values representing days since the start of the simulation

descendants(*class_definition*)
Get a list of all project tree descendants matching the class keyword :param class_definition[class]: A class definition matching the type of class wanted

Returns A list of PdmObjects matching the class_definition

export_flow_characteristics(*time_steps*, *injectors*, *producers*, *file_name*, *minimum_communication=0.0*, *aquifer_cell_threshold=0.1*)
Export Flow Characteristics data to text file in CSV format

Parameters:

Parameter	Description	Type
time_steps	Time step indices	List of Integer
injectors	Injector names	List of Strings
producers	Producer names	List of Strings
file_name	Export file name	Integer

(continues on next page)

(continued from previous page)

```
minimum_communication      | Minimum Communication, defaults to 0.0        |␣
↪Integer
aquifer_cell_threshold     | Aquifer Cell Threshold, defaults to 0.1       |␣
↪Integer
```

export_msw(*well_path*)

Export Eclipse Multi-segment-well model to file

Parameters **well_path** (*str*) – Well path name

export_property(*time_step*, *property_name*, *eclipse_keyword=<class 'property'>*, *undefined_value=0.0*, *export_file=<class 'property'>*)

Export an Eclipse property

Parameters

- **time_step** (*int*) – time step index
- **property_name** (*str*) – property to export
- **eclipse_keyword** (*str*) – Keyword used in export header. Defaults: value of property
- **undefined_value** (*double*) – Value to use for undefined values. Defaults to 0.0
- **export_file** (*str*) – File name for export. Defaults to the value of property parameter

export_snapshots_of_all_views(*prefix=''*, *export_folder=''*)

Export snapshots for all views in the case

Parameters

- **prefix** (*str*) – Exported file name prefix
- **export_folder** (*str*) – The path to export to. By default will use the global export folder

export_well_path_completions(*time_step*, *well_path_names*, *file_split*, *compdat_export='TRANSMISSIBILITIES'*, *include_perforations=True*, *include_fishbones=True*, *fishbones_exclude_main_bore=True*, *combination_mode='INDIVIDUALLY'*, *export_welspec=True*, *export_comments=True*, *custom_file_name=''*)

Export well path completions for the current case to file

Parameters:

```
Parameter                  | Description                                    ␣
↪ | Type
---------------------------| -----------------------------------------------
↪ | -----
time_step                  | Time step to export for                        ␣
↪ | Integer
well_path_names            | List of well path names                        ␣
↪ | List
file_split                 | Controls how export data is split into files   ␣
↪ | String enum
compdat_export             | Compdat export type                            ␣
↪ | String enum
include_perforations       | Export perforations?                           ␣
↪ | bool
```

(continues on next page)

(continued from previous page)

```
include_fishbones           | Export fishbones?                                 ␣
↪ | bool
fishbones_exclude_main_bore | Exclude main bore when exporting fishbones?       ␣
↪ | bool
combination_mode            | Settings for multiple completions in same cell  ␣
↪ | String Enum
export_welspec              | Export WELSPEC keyword                            ␣
↪ | bool
export_comments             | Export completion data source as comment        ␣
↪ | bool
custom_file_name            | Custom filename when file_split is "UNIFIED_FILE
↪"| String
```

Enum file_split:

```
Option                              | Description
----------------------------------- | ------------
"UNIFIED_FILE"                      | A single file with all combined␣
↪transmissibilities
"SPLIT_ON_WELL"                     | One file for each well with combined␣
↪transmissibilities
"SPLIT_ON_WELL_AND_COMPLETION_TYPE" | One file for each completion type for␣
↪each well
```

Enum compdat_export:

```
Option                                        | Description
--------------------------------------------- | ------------
"TRANSMISSIBILITIES"                          | Direct export of␣
↪transmissibilities
"WPIMULT_AND_DEFAULT_CONNECTION_FACTORS"      | Include WPIMULT in addition to␣
↪transmissibilities
```

Enum combination_mode:

```
Option               | Description
-------------------- | ------------
"INDIVIDUALLY"       | Exports the different completion types into separate␣
↪sections
"COMBINED"           | Export one combined transmissibility for each cell
```

grid(*index*)

Get Grid of a given index

Parameters **index** (*int*) – The grid index

Returns rips.grid.Grid

grid_property(*property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all grid cells. Synchronous, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for

- **grid_index** (*int*) – index to the grid we're getting values for
- **porosity_model** (*str*) – string enum

Returns A list of double values

grid_property_async(*property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all grid cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **gridIndex** (*int*) – index to the grid we're getting values for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

grids()

Get a list of all rips Grid objects in the case

Returns List of `rips.grid.Grid`

has_warnings()

import_formation_names(*formation_files=None*)

Import formation names into project and apply it to the current case

Parameters **formation_files** (*list*) – list of files to import

nnc_connections()

Get the NNC connection. Synchronous, so returns a list.

Returns A list of NNCConnection objects.

nnc_connections_async()

Get the NNC connections. Async, so returns an iterator.

Returns An iterator to a chunk object containing an array NNCConnection objects. Loop through the chunks and then the connection within the chunk to get all connections.

nnc_connections_dynamic_values(*property_name*, *time_step*)

Get the dynamic NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_dynamic_values_async(*property_name*, *time_step*)

Get the dynamic NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_generated_values(*property_name*, *time_step*)

Get the generated NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_generated_values_async(*property_name*, *time_step*)
Get the generated NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_static_values(*property_name*)
Get the static NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_static_values_async(*property_name*)
Get the static NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

pb2_object()
Private method

print_object_info()
Print the structure and data content of the PdmObject

replace(*new_grid_file*)
Replace the current case grid with a new grid loaded from file

Parameters **new_egrid_file** (*str*) – Path to EGRID file

reservoir_boundingbox()
Get the reservoir bounding box

Returns BoundingBox

BoundingBox class description:

Type	Name
int	min_x
int	max_x
int	min_y
int	max_y
int	min_z
int	max_z

reservoir_depth_range()
Get the reservoir depth range

Returns A tuple with two members. The first is the minimum depth, the second is the maximum depth

selected_cell_property(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)
Get a cell property for all selected cells. Sync, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns A list containing double values Loop through the chunks and then the values within the chunk to get all values.

selected_cell_property_async(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all selected cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

selected_cells()

Get the selected cells. Synchronous, so returns a list.

Returns A list of Cells.

selected_cells_async()

Get the selected cells. Async, so returns an iterator.

Returns An iterator to a chunk object containing an array of cells. Loop through the chunks and then the cells within the chunk to get all cells.

set_active_cell_property(*values*, *property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set a cell property for all active cells. For argument details, see *Result Definition*

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

set_active_cell_property_async(*values_iterator*, *property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set cell property for all active cells Async. Takes an iterator to the input values. For argument details, see *Result Definition*

Parameters

- **values_iterator** (*iterator*) – an iterator to the properties to be set
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property

- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

set_grid_property(*values*, *property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Set a cell property for all grid cells. For argument details, see *Result Definition*

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **grid_index** (*int*) – index to the grid we're setting values for
- **porosity_model** (*str*) – string enum

set_nnc_connections_values(*values*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set nnc connection values for all connections..

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum. See available()

set_value(*snake_keyword*, *value*)

Set the value associated with the provided keyword and updates ResInsight

Parameters

- **keyword** (*str*) – A string containing the parameter keyword
- **value** (*varying*) – A value matching the type of the parameter. See keyword documentation and/or print_object_info() to find the correct data type.

set_visible(*visible*)

Set the visibility of the object in the ResInsight project tree

simulation_wells()

Get a list of all simulation wells for a case

Returns `rips.generated.generated_classes.SimulationWell`

time_steps()

Get a list containing all time steps

The time steps are defined by the class **TimeStepDate**

TimeStepDate class description:

```
Type      | Name
--------- | ----------
int       | year
int       | month
int       | day
int       | hour
```

(continues on next page)

(continued from previous page)

```
int        | minute
int        | second
```

update()
Sync all fields from the Python Object to ResInsight

view(*view_id*)
Get a particular view belonging to a case by providing view id

Parameters **view_id** (*int*) – view id

Returns rips.generated.generated_classes.View

visible()
Get the visibility of the object in the ResInsight project tree

warnings()

Detailed documentation *rips.Case*

Examples

See *All Cases*, *Load Case* and *Selected Cases*.

Result Definition

When working with grid case results, the following two arguments are used in many functions to identify a result

Result Definition enums:

```
property_type            |        | porosity_model
(str enum)               |        | (str enum)
------------------------ | ------ | --------------
DYNAMIC_NATIVE           |        | MATRIX_MODEL
STATIC_NATIVE            |        | FRACTURE_MODEL
SOURSIMRL                |        |
GENERATED                |        |
INPUT_PROPERTY           |        |
FORMATION_NAMES          |        |
FLOW_DIAGNOSTICS         |        |
INJECTION_FLOODING       |        |
```

2.2.2 Grid class

Overview

Access to grid geometry and size information. Can be accessed with the grid() and grids() methods on rips.Case.

API Documentation

class rips.**Grid**(*index*, *case*, *channel*)
Bases: object

Grid Information. Created by methods in Case rips.case.grid() rips.case.grids()

cell_centers()
The cell center for all cells in given grid

Returns class with double attributes x, y, x giving cell centers

Return type List of Vec3d

cell_centers_async()
The cells center for all cells in given grid async.

Returns class with double attributes x, y, x giving cell centers

Return type Iterator to a list of Vec3d

cell_corners()
The cell corners for all cells in given grid

Returns a class with Vec3d for each corner (c0, c1.., c7)

Return type list of CellCorners

cell_corners_async()
The cell corners for all cells in given grid, async.

Returns a class with Vec3d for each corner (c0, c1.., c7)

Return type iterator to a list of CellCorners

dimensions()
The dimensions in i, j, k direction

Returns class with integer attributes i, j, k giving extent in all three dimensions.

Return type Vec3i

Detailed documentation *rips.Grid*

Examples

See *Grid Information*.

2.2.3 Instance - the main starting point

Overview

The basic access class to ResInsight. Use to find or launch a running instance of ResInsight

```
import rips
# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Get the ResInsight project
project = resinsight.project
```

API Documentation

class rips.**Instance**(*port=50051*, *launched=False*)
Bases: object

The ResInsight Instance class. Use to launch or find existing ResInsight instances

launched
Tells us whether the application was launched as a new process. If the application was launched we may need to close it when exiting the script.

Type bool

commands
Command executor. Set when creating an instance.

Type Commands

project
Current project in ResInsight. Set when creating an instance and updated when opening/closing projects.

Type *Project*

client_version_string()
Get a full version string, i.e. 2019.04.01

exit()
Tell ResInsight instance to quit

static find(*start_port=50051*, *end_port=50071*)
Search for an existing Instance of ResInsight by testing ports.

By default we search from port 50051 to 50071 or if the environment variable RESINSIGHT_GRPC_PORT is set we search RESINSIGHT_GRPC_PORT to RESINSIGHT_GRPC_PORT+20

Parameters

- **start_port** (*int*) – start searching from this port
- **end_port** (*int*) – search up to but not including this port

is_console()
Returns true if the connected ResInsight instance is a console app

is_gui()
Returns true if the connected ResInsight instance is a GUI app

static launch(*resinsight_executable=''*, *console=False*, *launch_port=-1*, *command_line_parameters=None*)
Launch a new Instance of ResInsight. This requires the environment variable RESINSIGHT_EXECUTABLE to be set or the parameter resinsight_executable to be provided. The RESINSIGHT_GRPC_PORT environment variable can be set to an alternative port number.

Parameters

- **resinsight_executable** (*str*) – Path to a valid ResInsight executable. If set will take precedence over what is provided in the RESINSIGHT_EXECUTABLE environment variable.
- **console** (*bool*) – If True, launch as console application, without GUI.
- **launch_port** (*int*) – If -1 will use the default port 50051 or RESINSIGHT_GRPC_PORT if anything else, ResInsight will try to launch with this port
- **command_line_parameters** (*list*) – Additional parameters as string entries in the list.

Returns an instance object if it worked. None if not.

Return type *Instance*

major_version()
Get an integer with the major version number

minor_version()
Get an integer with the minor version number

patch_version()
Get an integer with the patch version number

set_export_folder(*export_type*, *path*, *create_folder=False*)
Set the export folder used for all export functions

Parameters:

Parameter	Description	Type
export_type	String specifying what to export	String
path	Path to folder	String
create_folder	Create folder if it doesn't exist?	Boolean

Enum export_type:

Option	Description
"COMPLETIONS"	
"SNAPSHOTS"	
"PROPERTIES"	
"STATISTICS"	

set_main_window_size(*width*, *height*)
Set the main window size in pixels

Parameters:

Parameter	Description	Type
width	Width in pixels	Integer
height	Height in pixels	Integer

set_plot_window_size(*width*, *height*)
Set the plot window size in pixels

Parameters:

Parameter	Description	Type
width	Width in pixels	Integer
height	Height in pixels	Integer

set_start_dir(*path*)
Set current start directory

Parameters **path** (*str*) – path to directory

version_string()
Get a full version string, i.e. 2019.04.01

Detailed documentation *rips.Instance*

Examples

See *Instance Example* and *Launch With Commandline Options*.

2.2.4 PdmObjectBase class

Overview

The base class of all data classes in ResInsight. Not used directly but all attributes and methods are available in the other ResInsight data classes.

For any object based on PdmObjectBase you can access ancestors and descendants using the methods ancestors(), children() and descendants() using a class name as the argument

```
import rips
# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Get a list of all plot windows
plot_windows = resinsight.project.descendants(rips.PlotWindow)
```

See *Project Tree Classes* for all classes based on PdmObjectBase.

API Documentation

Detailed documentation *rips.PdmObjectBase*

2.2.5 Project class

Overview

The ResInsight project. Inherits *PdmObjectBase* and all its members. Always available as an object member "project" on the rips.Instance

```
# Import the module
import rips
# Connect to a ResInsight instance
resinsight = rips.Instance.find()
# Get the project
project = resinsight.project
```

API Documentation

class rips.**Project**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

The ResInsight Project

add_new_object(*class_definition*, *child_field=''*)

Create and add an object to the specified child field :param class_definition[class]: Class definition of the object to create :param child_field[str]: The keyword for the field to create a new object in. If empty, the first child array field is used.

Returns The created PdmObject inside the child_field

address()

Get the unique address of the PdmObject

Returns A 64-bit unsigned integer address

ancestor(*class_definition*)
Find the first ancestor that matches the provided class_keyword :param class_definition[class]: A class definition matching the type of class wanted

case(*case_id*)
Get a specific grid case from the provided case Id

Parameters **id** (*int*) – case id

Returns rips.generated.resinsight_classes.Case

cases()
Get a list of all grid cases in the project

Returns A list of rips.generated.generated_classes.Case

channel()
Private method

children(*child_field*, *class_definition*)
Get a list of all direct project tree children inside the provided child_field :param child_field[str]: A field name

Returns A list of PdmObjects inside the child_field

close()
Close the current project (and open new blank project)

copy_from(*object*)
Copy attribute values from object to self

create()

create_grid_case_group(*case_paths*)
Create a Grid Case Group from a list of cases

Parameters **case_paths** (*list*) – list of file path strings

Returns rips.generated.resinsight_classes.GridCaseGroup

descendants(*class_definition*)
Get a list of all project tree descendants matching the class keyword :param class_definition[class]: A class definition matching the type of class wanted

Returns A list of PdmObjects matching the class_definition

export_multi_case_snapshots(*grid_list_file*)
Export snapshots for a set of cases

Parameters **grid_list_file** (*str*) – Path to a file containing a list of grids to export snapshot for

export_snapshots(*snapshot_type='ALL'*, *prefix=''*, *plot_format='PNG'*)
Export all snapshots of a given type

Parameters

- **snapshot_type** (*str*) – Enum string ('ALL', 'VIEWS' or 'PLOTS')
- **prefix** (*str*) – Exported file name prefix
- **plot_format** (*str*) – Enum string, 'PNG' or 'PDF'

export_well_paths(*well_paths=None*, *md_step_size=5.0*)
Export a set of well paths

Parameters

- **well_paths** (*list*) – List of strings of well paths. If none, export all.
- **md_step_size** (*double*) – resolution of the exported well path

grid_case_group(*group_id*)
Get a particular grid case group belonging to a project

Parameters **groupId** (*int*) – group id

Returns rips.generated.generated_classes.GridCaseGroup

grid_case_groups()
Get a list of all grid case groups in the project

Returns List of rips.generated.generated_classes.GridCaseGroup

has_warnings()

import_formation_names(*formation_files=None*)
Import formation names into project

Parameters **formation_files** (*list*) – list of files to import

import_summary_case(*file_name=''*)
Import Summary Case

Parameters **file_name** (*str*) –

Returns FileSummaryCase

import_well_log_files(*well_log_files=None*, *well_log_folder=''*)
Import well log files into project

Parameters

- **well_log_files** (*list*) – List of file paths to import
- **well_log_folder** (*str*) – A folder path containing files to import

Returns A list of well path names (strings) that had logs imported

import_well_paths(*well_path_files=None*, *well_path_folder=''*)
Import well paths into project

Parameters

- **well_path_files** (*list*) – List of file paths to import
- **well_path_folder** (*str*) – A folder path containing files to import

Returns List of rips.generated.generated_classes.WellPath

load_case(*path*, *grid_only=False*)
Load a new grid case from the given file path

Parameters **path** (*str*) – file path to case

Returns rips.generated.generated_classes.Case

open(*path*)
Open a new project from the given path

Parameters **path** (*str*) – path to project file

pb2_object()
Private method

plot(*view_id*)
Get a particular plot by providing view id

Parameters **view_id** (*int*) – view id

Returns rips.generated.generated_classes.Plot

plots()
Get a list of all plots belonging to a project

Returns List of rips.generated.generated_classes.Plot

print_object_info()
Print the structure and data content of the PdmObject

replace_source_cases(*grid_list_file*, *case_group_id=0*)
Replace all source grid cases within a case group

Parameters

- **grid_list_file** (*str*) – path to file containing a list of cases
- **case_group_id** (*int*) – id of the case group to replace

save(*path=''*)
Save the project to the existing project file, or to a new file

Parameters **path** (*str*) – File path to the file to save the project to. If empty, saves to the active project file

scale_fracture_template(*template_id*, *half_length*, *height*, *d_factor*, *conductivity*)
Scale fracture template parameters

Parameters

- **template_id** (*int*) – ID of fracture template
- **half_length** (*double*) – Half Length scale factor
- **height** (*double*) – Height scale factor
- **d_factor** (*double*) – D-factor scale factor
- **conductivity** (*double*) – Conductivity scale factor

selected_cases()
Get a list of all grid cases selected in the project tree

Returns A list of rips.generated.generated_classes.Case

set_fracture_containment(*template_id*, *top_layer*, *base_layer*)
Set fracture template containment parameters

Parameters

- **template_id** (*int*) – ID of fracture template
- **top_layer** (*int*) – Top layer containment
- **base_layer** (*int*) – Base layer containment

set_value(*snake_keyword*, *value*)
Set the value associated with the provided keyword and updates ResInsight

Parameters

- **keyword** (*str*) – A string containing the parameter keyword

- **value** (*varying*) – A value matching the type of the parameter. See keyword documentation and/or print_object_info() to find the correct data type.

set_visible(*visible*)
Set the visibility of the object in the ResInsight project tree

summary_case(*case_id=-1*)
Find Summary Case

Parameters **case_id** (*int*) –

Returns FileSummaryCase

summary_cases()
Get a list of all summary cases in the Project

Returns: A list of `rips.generated.resinsight_classes.SummaryCase`

surface_folder(*folder_name=''*)
Get Surface Folder

Parameters **folder_name** (*str*) –

Returns SurfaceCollection

update()
Sync all fields from the Python Object to ResInsight

view(*view_id*)
Get a particular view belonging to a case by providing view id

Parameters **view_id** (*int*) – view id

Returns `rips.generated.generated_classes.View`

views()
Get a list of views belonging to a project

visible()
Get the visibility of the object in the ResInsight project tree

warnings()

well_path_by_name(*well_path_name*)
Get a specific well path by name from the project

Returns `rips.generated.generated_classes.WellPath`

well_paths()
Get a list of all well paths in the project

Returns List of `rips.generated.generated_classes.WellPath`

Detailed documentation *rips.Project*

Examples

See *Open Project*

2.2.6 View class

Overview

The base class of all views in ResInsight. Inherits *PdmObjectBase* and all its members

API Documentation

class rips.**View**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.ViewWindow

background_color
Background

Type str

current_time_step
Current Time Step

Type int

disable_lighting
Disable Results Lighting

Type str

grid_z_scale
Z Scale

Type float

id
View ID

Type int

perspective_projection
Perspective Projection

Type str

show_grid_box
Show Grid Box

Type str

show_z_scale
Show Z Scale Label

Type str

add_new_object(*class_definition*, *child_field=''*)
Create and add an object to the specified child field :param class_definition[class]: Class definition of the object to create :param child_field[str]: The keyword for the field to create a new object in. If empty, the first child array field is used.

Returns The created PdmObject inside the child_field

address()
Get the unique address of the PdmObject

Returns A 64-bit unsigned integer address

ancestor(*class_definition*)
Find the first ancestor that matches the provided class_keyword :param class_definition[class]: A class definition matching the type of class wanted

apply_cell_result(*result_type*, *result_variable*)
Apply a regular cell result

Parameters

- **result_type** (*str*) –

  String representing the result category. The valid values are::

  - DYNAMIC_NATIVE
  - STATIC_NATIVE
  - SOURSIMRL
  - GENERATED
  - INPUT_PROPERTY
  - FORMATION_NAMES
  - FLOW_DIAGNOSTICS
  - INJECTION_FLOODING

- **result_variable** (*str*) – String representing the result variable.

apply_flow_diagnostics_cell_result(*result_variable='TOF'*, *selection_mode='FLOW_TR_BY_SELECTION'*, *injectors=None*, *producers=None*)
Apply a flow diagnostics cell result

Parameters:

Parameter	Description	Type
result_variable	String representing the result value	String
selection_mode	String specifying which tracers to select	String
injectors	List of injector names, used by 'FLOW_TR_BY_SELECTION'	String List
producers	List of injector names, used by 'FLOW_TR_BY_SELECTION'	String List

Enum compdat_export:

Option	Description
"TOF"	Time of flight
"Fraction"	Fraction
"MaxFractionTracer"	Max Fraction Tracer
"Communication"	Communication

case()
Get the case the view belongs to

channel()
Private method

children(*child_field*, *class_definition*)
Get a list of all direct project tree children inside the provided child_field :param child_field[str]: A field name

Returns A list of PdmObjects inside the child_field

clone()
Clone the current view

copy_from(*object*)
Copy attribute values from object to self

descendants(*class_definition*)
Get a list of all project tree descendants matching the class keyword :param class_definition[class]: A class definition matching the type of class wanted

Returns A list of PdmObjects matching the class_definition

export_property(*undefined_value=0.0*)
Export the current Eclipse property from the view

Parameters **undefined_value** (*double*) – Value to use for undefined values. Defaults to 0.0

export_sim_well_fracture_completions(*time_step*, *simulation_well_names*, *file_split*, *compdat_export*)
Export fracture completions for simulation wells

Parameters:

```
Parameter                   | Description                                                 ␣
↪   | Type
----------------------------| ------------------------------------------------
↪- | -----
time_step                   | Time step to export for                                     ␣
↪   | Integer
simulation_well_names       | List of simulation well names                               ␣
↪   | List
file_split                  | Controls how export data is split into files                ␣
↪   | String enum
compdat_export              | Compdat export type                                         ␣
↪   | String enum
```

Enum file_split:

```
Option                              | Description
----------------------------------- | ------------
"UNIFIED_FILE" <b>Default Option</b>| A single file with all␣
↪transmissibilities
"SPLIT_ON_WELL"                     | One file for each well␣
↪transmissibilities
"SPLIT_ON_WELL_AND_COMPLETION_TYPE" | One file for each completion type for␣
↪each well
```

Enum compdat_export:

```
Option                                      | Description
--------------------------------------------| ------------
```

(continues on next page)

(continued from previous page)

```
"TRANSMISSIBILITIES"<b>Default Option</b>| Direct export of␣
↪transmissibilities
"WPIMULT_AND_DEFAULT_CONNECTION_FACTORS" | Include export of WPIMULT
```

export_snapshot(*prefix=''*, *export_folder=''*)
Export snapshot for the current view

Parameters

- **prefix** (*str*) – Exported file name prefix
- **export_folder** (*str*) – The path to export to. By default will use the global export folder

export_visible_cells(*export_keyword='FLUXNUM'*, *visible_active_cells_value=1*, *hidden_active_cells_value=0*, *inactive_cells_value=0*)
Export special properties for all visible cells.

Parameters

- **export_keyword** (*string*) – The keyword to export.
- **Choices** – 'FLUXNUM' or 'MULTNUM'. Default: 'FLUXNUM'
- **visible_active_cells_value** (*int*) – Value to export forvisible active cells. Default: 1
- **hidden_active_cells_value** (*int*) – Value to export for hidden active cells. Default: 0
- **inactive_cells_value** (*int*) – Value to export for inactive cells. Default: 0

has_warnings()

pb2_object()
Private method

print_object_info()
Print the structure and data content of the PdmObject

set_time_step(*time_step*)
Set the time step for current view

set_value(*snake_keyword*, *value*)
Set the value associated with the provided keyword and updates ResInsight

Parameters

- **keyword** (*str*) – A string containing the parameter keyword
- **value** (*varying*) – A value matching the type of the parameter. See keyword documentation and/or print_object_info() to find the correct data type.

set_visible(*visible*)
Set the visibility of the object in the ResInsight project tree

update()
Sync all fields from the Python Object to ResInsight

visible()
Get the visibility of the object in the ResInsight project tree

warnings()

Detailed documentation *rips.View*

Examples

See *View Example* and *Set Cell Result*.

2.3 Project Tree Classes

ResInsight provides access to a number of other objects in the Project Tree. These all inherit the *PdmObjectBase class* class.

You can look for objects of a specific type by using the **descendants** method of **rips.project**

```
import rips
# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Example code
print("ResInsight version: " + resinsight.version_string())
# Get a list of all Eclipsev view in the project
views = resinsight.project.descendants(rips.EclipseView)
```

2.3.1 rips Package

Classes

Case([pb2_object, channel])	The ResInsight base class for Cases
CellColors([pb2_object, channel])	Eclipse Cell Colors class
CellFilterCollection([pb2_object, channel])	rips.**active**
CheckableNamedObject([pb2_object, channel])	rips.**is_checked**
CommandRouter([pb2_object, channel])	The CommandRouter is used to call code independent to the project
DataContainerFloat([pb2_object, channel])	rips.**values**
DataContainerString([pb2_object, channel])	rips.**values**
DataContainerTime([pb2_object, channel])	rips.**values**
DepthTrackPlot([pb2_object, channel])	rips.**auto_scale_depth_enabled**
EclipseCase([pb2_object, channel])	The Regular Eclipse Results Case
EclipseContourMap([pb2_object, channel])	A contour map for Eclipse cases
EclipseResult([pb2_object, channel])	An eclipse result definition
EclipseView([pb2_object, channel])	The Eclipse 3d Reservoir View

Continued on next page

Table 1 – continued from previous page

ElasticProperties([pb2_object, channel])	rips.**file_path**
ElasticPropertyScaling([pb2_object, channel])	rips.**facies**
ElasticPropertyScalingCollection([...])	
EnsembleStatisticsSurface([pb2_object, channel])	
EnsembleSurface([pb2_object, channel])	
EnsembleWellLogs([pb2_object, channel])	
FaciesInitialPressureConfig([pb2_object, ...])	rips.**facies_name**
FaciesProperties([pb2_object, channel])	rips.**color_legend**
FileSummaryCase([pb2_object, channel])	A Summary Case based on SMSPEC files
FileWellPath([pb2_object, channel])	Well Paths Loaded From File
FractureTemplate([pb2_object, channel])	rips.**azimuth_angle**
FractureTemplateCollection([pb2_object, channel])	
GeoMechCase([pb2_object, channel])	The Abaqus Based GeoMech Case
GeoMechContourMap([pb2_object, channel])	A contour map for GeoMech cases
GeoMechPart([pb2_object, channel])	rips.**part_id**
GeoMechPartCollection([pb2_object, channel])	
GeoMechView([pb2_object, channel])	The Geomechanical 3d View
Grid(index, case, channel)	Grid Information.
GridCaseGroup([pb2_object, channel])	A statistics case group
GridCaseSurface([pb2_object, channel])	rips.**slice_index**
GridSummaryCase([pb2_object, channel])	A Summary Case based on extracting grid data.
Instance([port, launched])	The ResInsight Instance class.
ModeledWellPath([pb2_object, channel])	A Well Path created interactively in ResInsight
MudWeightWindowParameters([pb2_object, channel])	
NamedObject([pb2_object, channel])	rips.**name**
NonNetLayers([pb2_object, channel])	rips.**cutoff**
PdmObjectBase(pb2_object, channel)	The ResInsight base class for the Project Data Model
Plot([pb2_object, channel])	The Abstract Base Class for all Plot Objects
PlotCurve([pb2_object, channel])	

Continued on next page

Table 1 – continued from previous page

PlotWindow([pb2_object, channel])	The Abstract base class for all MDI Windows in the Plot Window
PressureTable([pb2_object, channel])	rips.**pressure_date**
PressureTableItem([pb2_object, channel])	rips.**depth**
Project([pb2_object, channel])	The ResInsight Project
ResampleData([pb2_object, channel])	rips.**time_steps**
Reservoir([pb2_object, channel])	Abtract base class for Eclipse Cases
SimulationWell([pb2_object, channel])	An Eclipse Simulation Well
StimPlanFractureTemplate([pb2_object, channel])	
StimPlanModel([pb2_object, channel])	rips.**anchor_position**
StimPlanModelCollection([pb2_object, channel])	
StimPlanModelPlot([pb2_object, channel])	A fracture model plot
StimPlanModelPlotCollection([pb2_object, ...])	
StimPlanModelTemplate([pb2_object, channel])	rips.**default_permeability**
StimPlanModelTemplateCollection([...])	
SummaryCase([pb2_object, channel])	The Base Class for all Summary Cases
SummaryCaseSubCollection([pb2_object, channel])	rips.**id**
SummaryPlot([pb2_object, channel])	A Summary Plot
SummaryPlotCollection([pb2_object, channel])	
Surface([pb2_object, channel])	
SurfaceCollection([pb2_object, channel])	rips.**surface_user_decription**
SurfaceInterface([pb2_object, channel])	rips.**depth_offset**
View([pb2_object, channel])	rips.**background_color**
ViewWindow([pb2_object, channel])	The Base Class for all Views and Plots in ResInsight
WbsParameters([pb2_object, channel])	rips.**df_source**
WellBoreStabilityPlot([pb2_object, channel])	A GeoMechanical Well Bore Stabilit Plot
WellLogExtractionCurve([pb2_object, channel])	

Continued on next page

Table 1 – continued from previous page

WellLogPlot([pb2_object, channel])	A Well Log Plot With a shared Depth Axis and Multiple Tracks
WellLogPlotCollection([pb2_object, channel])	
WellLogPlotCurve([pb2_object, channel])	
WellLogPlotTrack([pb2_object, channel])	
WellPath([pb2_object, channel])	A ResInsight Well Path
WellPathCollection([pb2_object, channel])	Collection of Well Paths
WellPathGeometry([pb2_object, channel])	Class containing the geometry of a modeled Well Path
WellPathTarget([pb2_object, channel])	Class containing the Well Target definition

Case

class rips.**Case**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

The ResInsight base class for Cases

file_path

Case File Name

Type str

id

Case ID

Type int

name

Case Name

Type str

name_setting

One of [FULL_CASE_NAME, SHORT_CASE_NAME, CUSTOM_NAME]

Type str

Methods Summary

active_cell_centers([porosity_model])	Get a cell centers for all active cells.
active_cell_centers_async([porosity_model])	Get a cell centers for all active cells.
active_cell_corners([porosity_model])	Get a cell corners for all active cells.
active_cell_corners_async([porosity_model])	Get a cell corners for all active cells.
active_cell_property(property_type, ...[, ...])	Get a cell property for all active cells.
active_cell_property_async(property_type, ...)	Get a cell property for all active cells.
available_nnc_properties()	Get a list of available NNC properties
available_properties(property_type[, ...])	Get a list of available properties
cell_count([porosity_model])	Get a cell count object containing number of active cells and total number of cells
cell_info_for_active_cells([porosity_model])	Get list of cell info objects for current case
cell_info_for_active_cells_async([...])	Get Stream of cell info objects for current case
coarsening_info()	Get a coarsening information for all grids in the case.

Continued on next page

Table 2 – continued from previous page

create_lgr_for_completion(time_step, ...)	Create a local grid refinement for the completions on the given well paths
create_multiple_fractures(template_id, ...)	Create Multiple Fractures in one go
create_saturation_pressure_plots()	Create saturation pressure plots for the current case
create_view()	Create a new view in the current case
create_well_bore_stability_plot(well_path, ...)	Create a new well bore stability plot
days_since_start()	Get a list of decimal values representing days since the start of the simulation
export_flow_characteristics(time_steps, ...)	Export Flow Characteristics data to text file in CSV format
export_msw(well_path)	Export Eclipse Multi-segment-well model to file
export_property(time_step, property_name[, ...])	Export an Eclipse property
export_snapshots_of_all_views([prefix, ...])	Export snapshots for all views in the case
export_well_path_completions(time_step, ...)	Export well path completions for the current case to file
grid(index)	Get Grid of a given index
grid_property(property_type, property_name, ...)	Get a cell property for all grid cells.
grid_property_async(property_type, ...[, ...])	Get a cell property for all grid cells.
grids()	Get a list of all rips Grid objects in the case
import_formation_names([formation_files])	Import formation names into project and apply it to the current case
nnc_connections()	Get the NNC connection.
nnc_connections_async()	Get the NNC connections.
nnc_connections_dynamic_values(...)	Get the dynamic NNC values.
nnc_connections_dynamic_values_async(...)	Get the dynamic NNC values.
nnc_connections_generated_values(...)	Get the generated NNC values.
nnc_connections_generated_values_async(...)	Get the generated NNC values.
nnc_connections_static_values(property_name)	Get the static NNC values.
nnc_connections_static_values_async(...)	Get the static NNC values.
replace(new_grid_file)	Replace the current case grid with a new grid loaded from file
reservoir_boundingbox()	Get the reservoir bounding box
reservoir_depth_range()	Get the reservoir depth range
selected_cell_property(property_type, ...[, ...])	Get a cell property for all selected cells.
selected_cell_property_async(property_type, ...)	Get a cell property for all selected cells.
selected_cells()	Get the selected cells.
selected_cells_async()	Get the selected cells.
set_active_cell_property(values, ...[, ...])	Set a cell property for all active cells.
set_active_cell_property_async(...[, ...])	Set cell property for all active cells Async.
set_grid_property(values, property_type, ...)	Set a cell property for all grid cells.
set_nnc_connections_values(values, ...[, ...])	Set nnc connection values for all connections..
simulation_wells()	Get a list of all simulation wells for a case

Continued on next page

Table 2 – continued from previous page

time_steps()	Get a list containing all time steps
view(view_id)	Get a particular view belonging to a case by providing view id

Methods Documentation

active_cell_centers(*porosity_model='MATRIX_MODEL'*)
Get a cell centers for all active cells. Synchronous, so returns a list.

Parameters **porosity_model** (*str*) – string enum. See available()

Returns A list of Vec3d

active_cell_centers_async(*porosity_model='MATRIX_MODEL'*)
Get a cell centers for all active cells. Async, so returns an iterator

Parameters **porosity_model** (*str*) – string enum. See available()

Returns An iterator to a chunk object containing an array of Vec3d values. Loop through the chunks and then the values within the chunk to get all values.

active_cell_corners(*porosity_model='MATRIX_MODEL'*)
Get a cell corners for all active cells. Synchronous, so returns a list.

Arguments: porosity_model(str): string enum. See available()

CellCorner class description:

Parameter	Description	Type
c0		Vec3d
c1		Vec3d
c2		Vec3d
c3		Vec3d
c4		Vec3d
c5		Vec3d
c6		Vec3d
c7		Vec3d

active_cell_corners_async(*porosity_model='MATRIX_MODEL'*)
Get a cell corners for all active cells. Async, so returns an iterator

Parameters **porosity_model** (*str*) – string enum. See available()

Returns An iterator to a chunk object containing an array of CellCorners (which is eight Vec3d values). Loop through the chunks and then the values within the chunk to get all values.

active_cell_property(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)
Get a cell property for all active cells. Sync, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns A list containing double values Loop through the chunks and then the values within the chunk to get all values.

active_cell_property_async(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all active cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

available_nnc_properties()

Get a list of available NNC properties

NNCConnection class description:

```
Parameter               | Description                                 |␣
↪Type
------------------------| ------------------------------------------- | ---
↪--
cell_grid_index1        | Reservoir Cell Index to cell 1              |␣
↪int32
cell_grid_index2        | Reservoir Cell Index to cell 2              |␣
↪int32
cell1                   | Reservoir Cell IJK to cell 1                |␣
↪Vec3i
cell2                   | Reservoir Cell IJK to cell 1                |␣
↪Vec3i
```

available_properties(*property_type*, *porosity_model='MATRIX_MODEL'*)

Get a list of available properties

For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string corresponding to property_type enum.
- **porosity_model** (*str*) – 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

cell_count(*porosity_model='MATRIX_MODEL'*)

Get a cell count object containing number of active cells and total number of cells

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns active_cell_count: number of active cells reservoir_cell_count: total number of reservoir cells

Return type Cell Count object with the following integer attributes

CellCount class description:

Parameter	Description	Type
active_cell_count	Number of active cells	Integer
reservoir_cell_count	Total number of cells	Integer

cell_info_for_active_cells(*porosity_model='MATRIX_MODEL'*)

Get list of cell info objects for current case

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns List of **CellInfo** objects

CellInfo class description:

Parameter	Description	Type
grid_index	Index to grid	Integer
parent_grid_index	Index to parent grid	Integer
coarsening_box_index	Index to coarsening box	Integer
local_ijk	Cell index in IJK directions of local grid	Vec3i
parent_ijk	Cell index in IJK directions of parent grid	Vec3i

Vec3i class description:

Parameter	Description	Type
i	I grid index	Integer
j	J grid index	Integer
k	K grid index	Integer

cell_info_for_active_cells_async(*porosity_model='MATRIX_MODEL'*)

Get Stream of cell info objects for current case

Parameters **porosity_model** (*str*) – String representing an enum. must be 'MATRIX_MODEL' or 'FRACTURE_MODEL'.

Returns Stream of **CellInfo** objects

See `rips.case.cell_info_for_active_cells()` for detalis on the **CellInfo** class.

coarsening_info()

Get a coarsening information for all grids in the case.

Returns A list of CoarseningInfo objects with two Vec3i min and max objects for each entry.

create_lgr_for_completion(*time_step*, *well_path_names*, *refinement_i*, *refinement_j*, *refinement_k*, *split_type*)

Create a local grid refinement for the completions on the given well paths

Parameters:

Parameter	Description	Type
time_steps	Time step index	Integer
well_path_names	List of well path names	List of Strings
refinement_i	Refinment in x-direction	Integer
refinement_j	Refinment in y-direction	Integer
refinement_k	Refinment in z-direction	Integer
split_type	Defines how to split LGRS	String enum

Enum split_type:

Option	Description
"LGR_PER_CELL"	One LGR for each completed cell
"LGR_PER_COMPLETION"	One LGR for each completion (fracture, perforation, ...)
"LGR_PER_WELL"	One LGR for each well

create_multiple_fractures(*template_id*, *well_path_names*, *min_dist_from_well_td*, *max_fractures_per_well*, *top_layer*, *base_layer*, *spacing*, *action*)

Create Multiple Fractures in one go

Parameters:

Parameter	Description	Type
template_id	Id of the template	Integer
well_path_names	List of well path names	List of Strings
min_dist_from_well_td	Minimum distance from well TD	Double
max_fractures_per_well	Max number of fractures per well	Integer
top_layer	Top grid k-level for fractures	Integer
base_layer	Base grid k-level for fractures	Integer
spacing	Spacing between fractures	Double
action	'APPEND_FRACTURES' or 'REPLACE_FRACTURES'	String enum

create_saturation_pressure_plots()

Create saturation pressure plots for the current case

create_view()

Create a new view in the current case

Returns rips.generated.generated_classes.View

create_well_bore_stability_plot(*well_path*, *time_step*, *parameters=None*)

Create a new well bore stability plot

Parameters

- **well_path** (*str*) – well path name
- **time_step** (*int*) – time step

Returns rips.generated.generated_classes.WellBoreStabilityPlot

days_since_start()

Get a list of decimal values representing days since the start of the simulation

export_flow_characteristics(*time_steps*, *injectors*, *producers*, *file_name*, *minimum_communication=0.0*, *aquifer_cell_threshold=0.1*)

Export Flow Characteristics data to text file in CSV format

Parameters:

Parameter	Description	Type
time_steps	Time step indices	List of Integer
injectors	Injector names	List of Strings
producers	Producer names	List of Strings
file_name	Export file name	Integer
minimum_communication	Minimum Communication, defaults to 0.0	Integer
aquifer_cell_threshold	Aquifer Cell Threshold, defaults to 0.1	Integer

export_msw(*well_path*)

Export Eclipse Multi-segment-well model to file

Parameters **well_path** (*str*) – Well path name

export_property(*time_step*, *property_name*, *eclipse_keyword=<class 'property'>*, *undefined_value=0.0*, *export_file=<class 'property'>*)

Export an Eclipse property

Parameters

- **time_step** (*int*) – time step index
- **property_name** (*str*) – property to export
- **eclipse_keyword** (*str*) – Keyword used in export header. Defaults: value of property
- **undefined_value** (*double*) – Value to use for undefined values. Defaults to 0.0
- **export_file** (*str*) – File name for export. Defaults to the value of property parameter

export_snapshots_of_all_views(*prefix=''*, *export_folder=''*)

Export snapshots for all views in the case

Parameters

- **prefix** (*str*) – Exported file name prefix
- **export_folder** (*str*) – The path to export to. By default will use the global export folder

export_well_path_completions(*time_step*, *well_path_names*, *file_split*, *compdat_export='TRANSMISSIBILITIES'*, *include_perforations=True*, *include_fishbones=True*, *fishbones_exclude_main_bore=True*, *combination_mode='INDIVIDUALLY'*, *export_welspec=True*, *export_comments=True*, *custom_file_name=''*)

Export well path completions for the current case to file

Parameters:

```
Parameter                   | Description                                                 ␣
↪   | Type
----------------------------| ------------------------------------------------------------
↪- | -----
time_step                   | Time step to export for                                     ␣
↪   | Integer
well_path_names             | List of well path names                                     ␣
↪   | List
file_split                  | Controls how export data is split into files                ␣
↪   | String enum
compdat_export              | Compdat export type                                         ␣
↪   | String enum
include_perforations        | Export perforations?                                        ␣
↪   | bool
include_fishbones           | Export fishbones?                                           ␣
↪   | bool
fishbones_exclude_main_bore | Exclude main bore when exporting fishbones?                  ␣
↪   | bool
combination_mode            | Settings for multiple completions in same cell              ␣
↪   | String Enum
export_welspec              | Export WELSPEC keyword                                      ␣
↪   | bool
export_comments             | Export completion data source as comment                    ␣
↪   | bool
custom_file_name            | Custom filename when file_split is "UNIFIED_
↪FILE"| String
```

Enum file_split:

```
Option                              | Description
----------------------------------- | ------------
"UNIFIED_FILE"                      | A single file with all combined␣
↪transmissibilities
"SPLIT_ON_WELL"                     | One file for each well with combined␣
↪transmissibilities
"SPLIT_ON_WELL_AND_COMPLETION_TYPE" | One file for each completion type for␣
↪each well
```

Enum compdat_export:

```
Option                                        | Description
--------------------------------------------- | ------------
"TRANSMISSIBILITIES"                          | Direct export of␣
↪transmissibilities
"WPIMULT_AND_DEFAULT_CONNECTION_FACTORS"      | Include WPIMULT in addition to␣
↪transmissibilities
```

Enum combination_mode:

```
Option              | Description
------------------- | ------------
"INDIVIDUALLY"      | Exports the different completion types into separate␣
↪sections
"COMBINED"          | Export one combined transmissibility for each cell
```

grid(*index*)

Get Grid of a given index

Parameters **index** (*int*) – The grid index

Returns rips.grid.Grid

grid_property(*property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all grid cells. Synchronous, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **grid_index** (*int*) – index to the grid we're getting values for
- **porosity_model** (*str*) – string enum

Returns A list of double values

grid_property_async(*property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all grid cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **gridIndex** (*int*) – index to the grid we're getting values for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

grids()

Get a list of all rips Grid objects in the case

Returns List of rips.grid.Grid

import_formation_names(*formation_files=None*)

Import formation names into project and apply it to the current case

Parameters **formation_files** (*list*) – list of files to import

nnc_connections()

Get the NNC connection. Synchronous, so returns a list.

Returns A list of NNCConnection objects.

nnc_connections_async()

Get the NNC connections. Async, so returns an iterator.

Returns An iterator to a chunk object containing an array NNCConnection objects. Loop through the chunks and then the connection within the chunk to get all connections.

nnc_connections_dynamic_values(*property_name*, *time_step*)

Get the dynamic NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_dynamic_values_async(*property_name*, *time_step*)
Get the dynamic NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_generated_values(*property_name*, *time_step*)
Get the generated NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_generated_values_async(*property_name*, *time_step*)
Get the generated NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_static_values(*property_name*)
Get the static NNC values.

Returns A list of doubles. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

nnc_connections_static_values_async(*property_name*)
Get the static NNC values. Async, so returns an iterator.

Returns An iterator to a chunk object containing an list of doubles. Loop through the chunks and then the values within the chunk to get values for all the connections. The order of the list matches the list from nnc_connections, i.e. the nth object of nnc_connections() refers to nth value in this list.

replace(*new_grid_file*)
Replace the current case grid with a new grid loaded from file

Parameters **new_egrid_file** (*str*) – Path to EGRID file

reservoir_boundingbox()
Get the reservoir bounding box

Returns BoundingBox

BoundingBox class description:

Type	Name
int	min_x
int	max_x
int	min_y
int	max_y
int	min_z
int	max_z

reservoir_depth_range()
Get the reservoir depth range

Returns A tuple with two members. The first is the minimum depth, the second is the maximum depth

selected_cell_property(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all selected cells. Sync, so returns a list. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns A list containing double values Loop through the chunks and then the values within the chunk to get all values.

selected_cell_property_async(*property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Get a cell property for all selected cells. Async, so returns an iterator. For argument details, see *Result Definition*

Parameters

- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

Returns An iterator to a chunk object containing an array of double values Loop through the chunks and then the values within the chunk to get all values.

selected_cells()

Get the selected cells. Synchronous, so returns a list.

Returns A list of Cells.

selected_cells_async()

Get the selected cells. Async, so returns an iterator.

Returns An iterator to a chunk object containing an array of cells. Loop through the chunks and then the cells within the chunk to get all cells.

set_active_cell_property(*values*, *property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set a cell property for all active cells. For argument details, see *Result Definition*

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

set_active_cell_property_async(*values_iterator*, *property_type*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set cell property for all active cells Async. Takes an iterator to the input values. For argument details, see *Result Definition*

Parameters

- **values_iterator** (*iterator*) – an iterator to the properties to be set
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum

set_grid_property(*values*, *property_type*, *property_name*, *time_step*, *grid_index=0*, *porosity_model='MATRIX_MODEL'*)

Set a cell property for all grid cells. For argument details, see *Result Definition*

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_type** (*str*) – string enum
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **grid_index** (*int*) – index to the grid we're setting values for
- **porosity_model** (*str*) – string enum

set_nnc_connections_values(*values*, *property_name*, *time_step*, *porosity_model='MATRIX_MODEL'*)

Set nnc connection values for all connections..

Parameters

- **values** (*list*) – a list of double precision floating point numbers
- **property_name** (*str*) – name of an Eclipse property
- **time_step** (*int*) – the time step for which to get the property for
- **porosity_model** (*str*) – string enum. See available()

simulation_wells()

Get a list of all simulation wells for a case

Returns rips.generated.generated_classes.SimulationWell

time_steps()

Get a list containing all time steps

The time steps are defined by the class **TimeStepDate**

TimeStepDate class description:

```
Type      | Name
--------- | ----------
int       | year
int       | month
int       | day
int       | hour
```

(continues on next page)

(continued from previous page)

```
int       | minute
int       | second
```

view(*view_id*)

Get a particular view belonging to a case by providing view id

Parameters **view_id** (*int*) – view id

Returns rips.generated.generated_classes.View

CellColors

class rips.**CellColors**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.EclipseResult

Eclipse Cell Colors class

CellFilterCollection

class rips.**CellFilterCollection**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

active

Active

Type str

CheckableNamedObject

class rips.**CheckableNamedObject**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.NamedObject

is_checked

Active

Type str

CommandRouter

class rips.**CommandRouter**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

The CommandRouter is used to call code independent to the project

Methods Summary

extract_surfaces([grid_model_filename, ...])	Extract Layer Surface

Methods Documentation

extract_surfaces(*grid_model_filename=''*, *layers=[]*, *minimum_i=-1*, *maximum_i=-1*, *minimum_j=-1*, *maximum_j=-1*)

Extract Layer Surface

Parameters

- **grid_model_filename** (*str*) –
- **layers** (*List of int*) –
- **minimum_i** (*int*) –
- **maximum_i** (*int*) –
- **minimum_j** (*int*) –
- **maximum_j** (*int*) –

Returns:

DataContainerFloat

class rips.**DataContainerFloat**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

values

Float Values

Type List of float

DataContainerString

class rips.**DataContainerString**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

values

String Values

Type List of str

DataContainerTime

class rips.**DataContainerTime**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

values

Time Values

Type List of time

DepthTrackPlot

class rips.**DepthTrackPlot**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.PlotWindow

auto_scale_depth_enabled
Auto Scale

Type str

axis_title_font_size
One of [XX_Small, X_Small, Small, Medium, Large, X_Large, XX_Large]

Type str

axis_value_font_size
One of [XX_Small, X_Small, Small, Medium, Large, X_Large, XX_Large]

Type str

depth_type
One of [MEASURED_DEPTH, TRUE_VERTICAL_DEPTH, PSEUDO_LENGTH, CONNECTION_NUMBER, TRUE_VERTICAL_DEPTH_RKB]

Type str

depth_unit
One of [UNIT_METER, UNIT_FEET, UNIT_NONE]

Type str

maximum_depth
Max

Type float

minimum_depth
Min

Type float

show_depth_grid_lines
One of [GRID_X_NONE, GRID_X_MAJOR, GRID_X_MAJOR_AND_MINOR]

Type str

sub_title_font_size
One of [XX_Small, X_Small, Small, Medium, Large, X_Large, XX_Large]

Type str

EclipseCase

class rips.**EclipseCase**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.Reservoir

The Regular Eclipse Results Case

EclipseContourMap

class rips.**EclipseContourMap**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.EclipseView

A contour map for Eclipse cases

Methods Summary

export_to_text([export_file_name, ...])	Export snapshot for the current view

Methods Documentation

export_to_text(*export_file_name=''*, *export_local_coordinates=False*, *undefined_value_label='NaN'*, *exclude_undefined_values=False*)

Export snapshot for the current view

Parameters

- **export_file_name** (*str*) – The file location to store results in.
- **export_local_coordinates** (*bool*) – Should we export local coordinates, or UTM.
- **undefined_value_label** (*str*) – Replace undefined values with this label.
- **exclude_undefined_values** (*bool*) – Skip undefined values.

EclipseResult

class rips.**EclipseResult**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

An eclipse result definition

flow_tracer_selection_mode

One of [FLOW_TR_INJ_AND_PROD, FLOW_TR_PRODUCERS, FLOW_TR_INJECTORS, FLOW_TR_BY_SELECTION]

Type str

phase_selection

One of [PHASE_ALL, PHASE_OIL, PHASE_GAS, PHASE_WAT]

Type str

porosity_model_type

One of [MATRIX_MODEL, FRACTURE_MODEL]

Type str

result_type

One of [DYNAMIC_NATIVE, STATIC_NATIVE, SOURSIMRL, GENERATED, INPUT_PROPERTY, FORMATION_NAMES, ALLAN_DIAGRAMS, FLOW_DIAGNOSTICS, INJECTION_FLOODING]

Type str

result_variable

Variable

Type str

selected_injector_tracers

Injector Tracers

Type List of str

selected_producer_tracers

Producer Tracers

Type List of str

selected_souring_tracers
Tracers

Type List of str

show_only_visible_categories_in_legend
Show Only Visible Categories In Legend

Type str

EclipseView

class rips.**EclipseView**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.View

The Eclipse 3d Reservoir View

Methods Summary

cell_result()	Cell Result
cell_result_data()	Current Eclipse Cell Result
set_cell_result_data(values)	Set Current Eclipse Cell Result

Methods Documentation

cell_result()
Cell Result

Returns CellColors

cell_result_data()
Current Eclipse Cell Result

Returns str

set_cell_result_data(*values*)
Set Current Eclipse Cell Result

Parameters **values** (*str*) – data

ElasticProperties

class rips.**ElasticProperties**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

file_path
File Path

Type str

properties_table
Properties Table

Type str

show_scaled_properties
Show Scaled Properties

Type str

Methods Summary

add_property_scaling([formation, facies, ...])	Add Elastic Property Scaling
property_scaling_collection()	PropertyScalingCollection

Methods Documentation

add_property_scaling(*formation=''*, *facies=''*, *property=''*, *scale=0*)
Add Elastic Property Scaling

Parameters

- **formation** (*str*) – Formation
- **facies** (*str*) – Facies
- **property** (*str*) – Property
- **scale** (*float*) – Scale

Returns ElasticPropertyScaling

property_scaling_collection()
PropertyScalingCollection

Returns ElasticPropertyScalingCollection

ElasticPropertyScaling

class rips.**ElasticPropertyScaling**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.CheckableNamedObject

facies
Facies

Type str

formation
Formation

Type str

property
One of [UNDEFINED, FACIES, LAYERS, POROSITY, PERMEABILITY_X, PERMEABILITY_Z, INITIAL_PRESSURE, PRESSURE, STRESS, INITIAL_STRESS, STRESS_GRADIENT, YOUNGS_MODULUS, POISSONS_RATIO, K_IC, PROPPANT_EMBEDMENT, BIOT_COEFFICIENT, K0, FLUID_LOSS_COEFFICIENT, SPURT_LOSS, TEMPERATURE, RELATIVE_PERMEABILITY_FACTOR, PORO_ELASTIC_CONSTANT, THERMAL_EXPANSION_COEFFICIENT, IMMOBILE_FLUID_SATURATION, NET_TO_GROSS, POROSITY_UNSCALED, EQLNUM, PRESSURE_GRADIENT, FORMATIONS]

Type str

scale
Scale

Type float

ElasticPropertyScalingCollection

class rips.**ElasticPropertyScalingCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

elastic_property_scalings()	Elastic Property Scalings

Methods Documentation

elastic_property_scalings()
Elastic Property Scalings

Returns List of ElasticPropertyScaling

EnsembleStatisticsSurface

class rips.**EnsembleStatisticsSurface**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SurfaceInterface

EnsembleSurface

class rips.**EnsembleSurface**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SurfaceCollection

EnsembleWellLogs

class rips.**EnsembleWellLogs**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.NamedObject

FaciesInitialPressureConfig

class rips.**FaciesInitialPressureConfig**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

facies_name
Facies

Type str

facies_value
Value

Type int

fraction
? Pressure Fraction

Type float

FaciesProperties

class rips.**FaciesProperties**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

color_legend
Colors

Type str

file_path
File Path

Type str

properties_table
Properties Table

Type str

Methods Summary

facies_definition()	**returns** EclipseResult

Methods Documentation

facies_definition()

Returns EclipseResult

FileSummaryCase

class rips.**FileSummaryCase**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SummaryCase

A Summary Case based on SMSPEC files

include_restart_files
Include Restart Files

Type str

FileWellPath

class rips.**FileWellPath**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.WellPath

Well Paths Loaded From File

FractureTemplate

class rips.**FractureTemplate**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

azimuth_angle
Azimuth Angle

Type float

orientation
One of [Azimuth, Longitudinal, Transverse]

Type str

FractureTemplateCollection

class rips.**FractureTemplateCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

append_fracture_template([file_path])	Create a new StimPlan Fracture Template

Methods Documentation

append_fracture_template(*file_path=''*)
Create a new StimPlan Fracture Template

Parameters **file_path** (*str*) – File Path to StimPlan Countour File

Returns StimPlanFractureTemplate

GeoMechCase

class rips.**GeoMechCase**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.Case

The Abaqus Based GeoMech Case

Methods Summary

views()	All GeoMech Views in the Case

Methods Documentation

views()
All GeoMech Views in the Case

Returns List of GeoMechView

GeoMechContourMap

class rips.**GeoMechContourMap**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.GeoMechView

A contour map for GeoMech cases

Methods Summary

export_to_text([export_file_name, ...])	Export snapshot for the current view

Methods Documentation

export_to_text(*export_file_name=''*, *export_local_coordinates=False*, *undefined_value_label='NaN'*, *exclude_undefined_values=False*)
Export snapshot for the current view

Parameters

- **export_file_name** (*str*) – The file location to store results in.
- **export_local_coordinates** (*bool*) – Should we export local coordinates, or UTM.
- **undefined_value_label** (*str*) – Replace undefined values with this label.
- **exclude_undefined_values** (*bool*) – Skip undefined values.

GeoMechPart

class rips.**GeoMechPart**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.CheckableNamedObject

part_id
Part Id

Type int

GeoMechPartCollection

class rips.**GeoMechPartCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

parts()	Parts

Methods Documentation

parts()
Parts

Returns List of GeoMechPart

GeoMechView

class rips.**GeoMechView**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.View

The Geomechanical 3d View

Grid

class rips.**Grid**(*index*, *case*, *channel*)
Bases: object

Grid Information. Created by methods in Case rips.case.grid() rips.case.grids()

Methods Summary

cell_centers()	The cell center for all cells in given grid
cell_centers_async()	The cells center for all cells in given grid async.
cell_corners()	The cell corners for all cells in given grid
cell_corners_async()	The cell corners for all cells in given grid, async.
dimensions()	The dimensions in i, j, k direction

Methods Documentation

cell_centers()
The cell center for all cells in given grid

Returns class with double attributes x, y, x giving cell centers

Return type List of Vec3d

cell_centers_async()
The cells center for all cells in given grid async.

Returns class with double attributes x, y, x giving cell centers

Return type Iterator to a list of Vec3d

cell_corners()
The cell corners for all cells in given grid

Returns a class with Vec3d for each corner (c0, c1.., c7)

Return type list of CellCorners

cell_corners_async()
The cell corners for all cells in given grid, async.

Returns a class with Vec3d for each corner (c0, c1.., c7)

Return type iterator to a list of CellCorners

dimensions()
The dimensions in i, j, k direction

Returns class with integer attributes i, j, k giving extent in all three dimensions.

Return type Vec3i

GridCaseGroup

class rips.**GridCaseGroup**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

A statistics case group

group_id

Case Group ID

Type int

user_description

Name

Type str

Methods Summary

compute_statistics([case_ids])	Compute statistics for the given case ids
create_statistics_case()	Create a Statistics case in the Grid Case Group
statistics_cases()	Get a list of all statistics cases in the Grid Case Group
view(view_id)	Get a particular view belonging to a case group by providing view id
views()	Get a list of views belonging to a grid case group

Methods Documentation

compute_statistics(*case_ids=None*)

Compute statistics for the given case ids

Parameters **case_ids** (*list of integers*) – List of case ids. If this is None all cases in group are included

create_statistics_case()

Create a Statistics case in the Grid Case Group

Returns rips.generated.generated_classes.EclipseCase

statistics_cases()

Get a list of all statistics cases in the Grid Case Group

Returns List of rips.generated.generated_classes.EclipseCase

view(*view_id*)

Get a particular view belonging to a case group by providing view id

Parameters **id** (*int*) – view id

Returns List of rips.generated.generated_classes.EclipseView

views()

Get a list of views belonging to a grid case group

Returns List of rips.generated.generated_classes.EclipseView

GridCaseSurface

class rips.**GridCaseSurface**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SurfaceInterface

slice_index
Slice Index (K)

Type int

source_case
Source Case

Type str

watertight
Watertight Surface (fill gaps)

Type str

GridSummaryCase

class rips.**GridSummaryCase**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SummaryCase

A Summary Case based on extracting grid data.

Instance

class rips.**Instance**(*port=50051*, *launched=False*)
Bases: object

The ResInsight Instance class. Use to launch or find existing ResInsight instances

launched
Tells us whether the application was launched as a new process. If the application was launched we may need to close it when exiting the script.

Type bool

commands
Command executor. Set when creating an instance.

Type Commands

project
Current project in ResInsight. Set when creating an instance and updated when opening/closing projects.

Type *Project*

Methods Summary

client_version_string()	Get a full version string, i.e.
exit()	Tell ResInsight instance to quit
find([start_port, end_port])	Search for an existing Instance of ResInsight by testing ports.

Continued on next page

Table 15 – continued from previous page

is_console()	Returns true if the connected ResInsight instance is a console app
is_gui()	Returns true if the connected ResInsight instance is a GUI app
launch([resinsight_executable, console, ...])	Launch a new Instance of ResInsight.
major_version()	Get an integer with the major version number
minor_version()	Get an integer with the minor version number
patch_version()	Get an integer with the patch version number
set_export_folder(export_type, path[, ...])	Set the export folder used for all export functions
set_main_window_size(width, height)	Set the main window size in pixels
set_plot_window_size(width, height)	Set the plot window size in pixels
set_start_dir(path)	Set current start directory
version_string()	Get a full version string, i.e.

Methods Documentation

client_version_string()
Get a full version string, i.e. 2019.04.01

exit()
Tell ResInsight instance to quit

static find(*start_port=50051*, *end_port=50071*)
Search for an existing Instance of ResInsight by testing ports.

By default we search from port 50051 to 50071 or if the environment variable RESINSIGHT_GRPC_PORT is set we search RESINSIGHT_GRPC_PORT to RESINSIGHT_GRPC_PORT+20

Parameters

- **start_port** (*int*) – start searching from this port
- **end_port** (*int*) – search up to but not including this port

is_console()
Returns true if the connected ResInsight instance is a console app

is_gui()
Returns true if the connected ResInsight instance is a GUI app

static launch(*resinsight_executable=''*, *console=False*, *launch_port=-1*, *command_line_parameters=None*)
Launch a new Instance of ResInsight. This requires the environment variable RESINSIGHT_EXECUTABLE to be set or the parameter resinsight_executable to be provided. The RESINSIGHT_GRPC_PORT environment variable can be set to an alternative port number.

Parameters

- **resinsight_executable** (*str*) – Path to a valid ResInsight executable. If set will take precedence over what is provided in the RESINSIGHT_EXECUTABLE environment variable.
- **console** (*bool*) – If True, launch as console application, without GUI.
- **launch_port** (*int*) – If -1 will use the default port 50051 or RESINSIGHT_GRPC_PORT if anything else, ResInsight will try to launch with this port
- **command_line_parameters** (*list*) – Additional parameters as string entries in the list.

Returns an instance object if it worked. None if not.

Return type *Instance*

major_version()
Get an integer with the major version number

minor_version()
Get an integer with the minor version number

patch_version()
Get an integer with the patch version number

set_export_folder(*export_type*, *path*, *create_folder=False*)
Set the export folder used for all export functions

Parameters:

Parameter	Description	Type
export_type	String specifying what to export	String
path	Path to folder	String
create_folder	Create folder if it doesn't exist?	Boolean

Enum export_type:

Option	Description
"COMPLETIONS"	
"SNAPSHOTS"	
"PROPERTIES"	
"STATISTICS"	

set_main_window_size(*width*, *height*)
Set the main window size in pixels

Parameters:

Parameter	Description	Type
width	Width in pixels	Integer
height	Height in pixels	Integer

set_plot_window_size(*width*, *height*)
Set the plot window size in pixels

Parameters:

Parameter	Description	Type
width	Width in pixels	Integer
height	Height in pixels	Integer

set_start_dir(*path*)
Set current start directory

Parameters **path** (*str*) – path to directory

version_string()
Get a full version string, i.e. 2019.04.01

ModeledWellPath

class rips.**ModeledWellPath**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.WellPath

A Well Path created interactively in ResInsight

Methods Summary

append_lateral([tie_in_depth, lateral_name])	Append Well Path Lateral
well_path_geometry()	Trajectory

Methods Documentation

append_lateral(*tie_in_depth=0*, *lateral_name=''*)
Append Well Path Lateral

Parameters

- **tie_in_depth** (*float*) – Measured Depth on the Parent Well Path
- **lateral_name** (*str*) – Lateral Name

Returns ModeledWellPath

well_path_geometry()
Trajectory

Returns WellPathGeometry

MudWeightWindowParameters

class rips.**MudWeightWindowParameters**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

NamedObject

class rips.**NamedObject**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

name
Name

Type str

NonNetLayers

class rips.**NonNetLayers**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

cutoff
Cutoff

Type float

facies
Facies

Type str

Methods Summary

facies_definition()	**returns** EclipseResult

Methods Documentation

facies_definition()

Returns EclipseResult

PdmObjectBase

class rips.**PdmObjectBase**(*pb2_object*, *channel*)
Bases: object

The ResInsight base class for the Project Data Model

Methods Summary

add_new_object(class_definition[, child_field])	Create and add an object to the specified child field :param class_definition[class]: Class definition of the object to create :param child_field[str]: The keyword for the field to create a new object in.
address()	Get the unique address of the PdmObject
ancestor(class_definition)	Find the first ancestor that matches the provided class_keyword :param class_definition[class]: A class definition matching the type of class wanted
channel()	Private method
children(child_field, class_definition)	Get a list of all direct project tree children inside the provided child_field :param child_field[str]: A field name
copy_from(object)	Copy attribute values from object to self
descendants(class_definition)	Get a list of all project tree descendants matching the class keyword :param class_definition[class]: A class definition matching the type of class wanted
has_warnings()	
pb2_object()	Private method
print_object_info()	Print the structure and data content of the PdmObject
set_value(snake_keyword, value)	Set the value associated with the provided keyword and updates ResInsight
set_visible(visible)	Set the visibility of the object in the ResInsight project tree
update()	Sync all fields from the Python Object to ResInsight

Continued on next page

Table 18 – continued from previous page

visible()	Get the visibility of the object in the ResInsight project tree
warnings()	

Methods Documentation

add_new_object(*class_definition*, *child_field=''*)
Create and add an object to the specified child field :param class_definition[class]: Class definition of the object to create :param child_field[str]: The keyword for the field to create a new object in. If empty, the first child array field is used.

Returns The created PdmObject inside the child_field

address()
Get the unique address of the PdmObject

Returns A 64-bit unsigned integer address

ancestor(*class_definition*)
Find the first ancestor that matches the provided class_keyword :param class_definition[class]: A class definition matching the type of class wanted

channel()
Private method

children(*child_field*, *class_definition*)
Get a list of all direct project tree children inside the provided child_field :param child_field[str]: A field name

Returns A list of PdmObjects inside the child_field

copy_from(*object*)
Copy attribute values from object to self

descendants(*class_definition*)
Get a list of all project tree descendants matching the class keyword :param class_definition[class]: A class definition matching the type of class wanted

Returns A list of PdmObjects matching the class_definition

has_warnings()

pb2_object()
Private method

print_object_info()
Print the structure and data content of the PdmObject

set_value(*snake_keyword*, *value*)
Set the value associated with the provided keyword and updates ResInsight

Parameters

- **keyword** (*str*) – A string containing the parameter keyword
- **value** (*varying*) – A value matching the type of the parameter. See keyword documentation and/or print_object_info() to find the correct data type.

set_visible(*visible*)
Set the visibility of the object in the ResInsight project tree

update()
Sync all fields from the Python Object to ResInsight

visible()
Get the visibility of the object in the ResInsight project tree

warnings()

Plot

class rips.**Plot**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.PlotWindow

The Abstract Base Class for all Plot Objects

PlotCurve

class rips.**PlotCurve**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

PlotWindow

class rips.**PlotWindow**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.ViewWindow

The Abstract base class for all MDI Windows in the Plot Window

id
View ID

Type int

Methods Summary

export_snapshot([export_folder, ...])	Export snapshot for the current plot

Methods Documentation

export_snapshot(*export_folder=''*, *file_prefix=''*, *output_format='PNG'*)
Export snapshot for the current plot

Parameters

- **export_folder** (*str*) – The path to export to. By default will use the global export folder
- **prefix** (*str*) – Exported file name prefix
- **output_format** (*str*) – Enum string. Can be 'PNG' or 'PDF'.

PressureTable

class rips.**PressureTable**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

pressure_date
Pressure Date

Type str

Methods Summary

items()	Pressure Table Items

Methods Documentation

items()
Pressure Table Items

Returns List of PressureTableItem

PressureTableItem

class rips.**PressureTableItem**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

depth
Depth TVDMSL [m]

Type float

initial_pressure
Initial Pressure [Bar]

Type float

pressure
Pressure [Bar]

Type float

Project

class rips.**Project**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

The ResInsight Project

Methods Summary

case(case_id)	Get a specific grid case from the provided case Id
cases()	Get a list of all grid cases in the project

Continued on next page

Table 21 – continued from previous page

close()	Close the current project (and open new blank project)
create()	
create_grid_case_group(case_paths)	Create a Grid Case Group from a list of cases
export_multi_case_snapshots(grid_list_file)	Export snapshots for a set of cases
export_snapshots([snapshot_type, prefix, ...])	Export all snapshots of a given type
export_well_paths([well_paths, md_step_size])	Export a set of well paths
grid_case_group(group_id)	Get a particular grid case group belonging to a project
grid_case_groups()	Get a list of all grid case groups in the project
import_formation_names([formation_files])	Import formation names into project
import_summary_case([file_name])	Import Summary Case
import_well_log_files([well_log_files, ...])	Import well log files into project
import_well_paths([well_path_files, ...])	Import well paths into project
load_case(path[, grid_only])	Load a new grid case from the given file path
open(path)	Open a new project from the given path
plot(view_id)	Get a particular plot by providing view id
plots()	Get a list of all plots belonging to a project
replace_source_cases(grid_list_file[, ...])	Replace all source grid cases within a case group
save([path])	Save the project to the existing project file, or to a new file
scale_fracture_template(template_id, ...)	Scale fracture template parameters
selected_cases()	Get a list of all grid cases selected in the project tree
set_fracture_containment(template_id, ...)	Set fracture template containment parameters
summary_case([case_id])	Find Summary Case
summary_cases()	Get a list of all summary cases in the Project
surface_folder([folder_name])	Get Surface Folder
view(view_id)	Get a particular view belonging to a case by providing view id
views()	Get a list of views belonging to a project
well_path_by_name(well_path_name)	Get a specific well path by name from the project
well_paths()	Get a list of all well paths in the project

Methods Documentation

case(*case_id*)

Get a specific grid case from the provided case Id

Parameters **id** (*int*) – case id

Returns `rips.generated.resinsight_classes.Case`

cases()

Get a list of all grid cases in the project

Returns A list of `rips.generated.generated_classes.Case`

close()

Close the current project (and open new blank project)

create()

create_grid_case_group(*case_paths*)
Create a Grid Case Group from a list of cases

Parameters **case_paths** (*list*) – list of file path strings

Returns rips.generated.resinsight_classes.GridCaseGroup

export_multi_case_snapshots(*grid_list_file*)
Export snapshots for a set of cases

Parameters **grid_list_file** (*str*) – Path to a file containing a list of grids to export snapshot for

export_snapshots(*snapshot_type='ALL'*, *prefix=''*, *plot_format='PNG'*)
Export all snapshots of a given type

Parameters

- **snapshot_type** (*str*) – Enum string ('ALL', 'VIEWS' or 'PLOTS')
- **prefix** (*str*) – Exported file name prefix
- **plot_format** (*str*) – Enum string, 'PNG' or 'PDF'

export_well_paths(*well_paths=None*, *md_step_size=5.0*)
Export a set of well paths

Parameters

- **well_paths** (*list*) – List of strings of well paths. If none, export all.
- **md_step_size** (*double*) – resolution of the exported well path

grid_case_group(*group_id*)
Get a particular grid case group belonging to a project

Parameters **groupId** (*int*) – group id

Returns rips.generated.generated_classes.GridCaseGroup

grid_case_groups()
Get a list of all grid case groups in the project

Returns List of rips.generated.generated_classes.GridCaseGroup

import_formation_names(*formation_files=None*)
Import formation names into project

Parameters **formation_files** (*list*) – list of files to import

import_summary_case(*file_name=''*)
Import Summary Case

Parameters **file_name** (*str*) –

Returns FileSummaryCase

import_well_log_files(*well_log_files=None*, *well_log_folder=''*)
Import well log files into project

Parameters

- **well_log_files** (*list*) – List of file paths to import
- **well_log_folder** (*str*) – A folder path containing files to import

Returns A list of well path names (strings) that had logs imported

import_well_paths(*well_path_files=None*, *well_path_folder=''*)
Import well paths into project

Parameters

- **well_path_files** (*list*) – List of file paths to import
- **well_path_folder** (*str*) – A folder path containing files to import

Returns List of `rips.generated.generated_classes.WellPath`

load_case(*path*, *grid_only=False*)
Load a new grid case from the given file path

Parameters **path** (*str*) – file path to case

Returns `rips.generated.generated_classes.Case`

open(*path*)
Open a new project from the given path

Parameters **path** (*str*) – path to project file

plot(*view_id*)
Get a particular plot by providing view id

Parameters **view_id** (*int*) – view id

Returns `rips.generated.generated_classes.Plot`

plots()
Get a list of all plots belonging to a project

Returns List of `rips.generated.generated_classes.Plot`

replace_source_cases(*grid_list_file*, *case_group_id=0*)
Replace all source grid cases within a case group

Parameters

- **grid_list_file** (*str*) – path to file containing a list of cases
- **case_group_id** (*int*) – id of the case group to replace

save(*path=''*)
Save the project to the existing project file, or to a new file

Parameters **path** (*str*) – File path to the file to save the project to. If empty, saves to the active project file

scale_fracture_template(*template_id*, *half_length*, *height*, *d_factor*, *conductivity*)
Scale fracture template parameters

Parameters

- **template_id** (*int*) – ID of fracture template
- **half_length** (*double*) – Half Length scale factor
- **height** (*double*) – Height scale factor
- **d_factor** (*double*) – D-factor scale factor
- **conductivity** (*double*) – Conductivity scale factor

selected_cases()
Get a list of all grid cases selected in the project tree

Returns A list of `rips.generated.generated_classes.Case`

set_fracture_containment(*template_id*, *top_layer*, *base_layer*)
Set fracture template containment parameters

Parameters

- **template_id** (*int*) – ID of fracture template
- **top_layer** (*int*) – Top layer containment
- **base_layer** (*int*) – Base layer containment

summary_case(*case_id=-1*)
Find Summary Case

Parameters **case_id** (*int*) –

Returns FileSummaryCase

summary_cases()
Get a list of all summary cases in the Project

Returns: A list of `rips.generated.resinsight_classes.SummaryCase`

surface_folder(*folder_name=''*)
Get Surface Folder

Parameters **folder_name** (*str*) –

Returns SurfaceCollection

view(*view_id*)
Get a particular view belonging to a case by providing view id

Parameters **view_id** (*int*) – view id

Returns `rips.generated.generated_classes.View`

views()
Get a list of views belonging to a project

well_path_by_name(*well_path_name*)
Get a specific well path by name from the project

Returns `rips.generated.generated_classes.WellPath`

well_paths()
Get a list of all well paths in the project

Returns List of `rips.generated.generated_classes.WellPath`

ResampleData

class rips.**ResampleData**(*pb2_object=None*, *channel=None*)
Bases: `rips.pdmobject.PdmObjectBase`

time_steps
Time Steps

Type List of time

values
Values

Type List of float

Reservoir

class rips.**Reservoir**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.Case

Abtract base class for Eclipse Cases

Methods Summary

views()	All Eclipse Views in the case

Methods Documentation

views()
All Eclipse Views in the case

Returns List of EclipseView

SimulationWell

class rips.**SimulationWell**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

An Eclipse Simulation Well

name
Name

Type str

Methods Summary

case()	
cells(timestep)	Get reservoir cells the simulation well is defined for
status(timestep)	Get simulation well status

Methods Documentation

case()

cells(*timestep*)
Get reservoir cells the simulation well is defined for

SimulationWellCellInfo class description:

```
Parameter   | Description                                                    ␣
↪   | Type
----------- | ---------------------------------------------------------
↪-- | -----
ijk         | Cell IJK location                                              ␣
↪   | Vec3i
```

(continues on next page)

(continued from previous page)

```
grid_index   | Grid index                                                    ␣
↪    | int
is_open      | True if connection to is open at the specified time␣
↪step   | bool
branch_id    |                                                               ␣
↪    | int
segment_id   |                                                               ␣
↪    | int
```

Parameters **timestep** (*int*) – Time step index

Returns List of SimulationWellCellInfo

status (*timestep*)

Get simulation well status

SimulationWellStatus class description:

```
Parameter    | Description                                                   ␣
↪        | Type
------------ | ------------------------------------------------------------
↪------- | -----
well_type    | Well type as string                                           ␣
↪        | string
is_open      | True if simulation well is open at the specified time␣
↪step     | bool
```

Parameters **timestep** (*int*) – Time step index

StimPlanFractureTemplate

class rips.**StimPlanFractureTemplate**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.FractureTemplate

StimPlanModel

class rips.**StimPlanModel**(*pb2_object=None*, *channel=None*)

Bases: rips.generated.generated_classes.CheckableNamedObject

anchor_position

Anchor Position

Type str

auto_compute_barrier

Auto Compute Barrier

Type str

azimuth_angle

Azimuth Angle

Type float

barrier

Barrier

Type str

barrier_dip
Barrier Dip

Type float

barrier_fault_name
Barrier Fault

Type str

barrier_text_annotation
Barrier Text Annotation

Type str

bounding_box_horizontal
Bounding Box Horizontal

Type float

bounding_box_vertical
Bounding Box Vertical

Type float

distance_to_barrier
Distance To Barrier [m]

Type float

eclipse_case
Dynamic Case

Type str

extraction_depth_bottom
Depth

Type float

extraction_depth_top
Depth

Type float

extraction_offset_bottom
Bottom Offset

Type float

extraction_offset_top
Top Offset

Type float

extraction_type
One of [TVT, TST]

Type str

formation_dip
Formation Dip

Type float

fracture_orientation
One of [Longitudinal, Transverse, Azimuth]

Type str

initial_pressure_eclipse_case
Initial Pressure Case

Type str

measured_depth
Measured Depth

Type float

perforation_interval
Perforation Interval

Type str

perforation_length
Perforation Length [m]

Type float

poro_elastic_constant
Poro-Elastic Constant

Type float

relative_permeability_factor
Relative Permeability Factor

Type float

show_all_faults
Show All Faults

Type str

show_only_barrier_fault
Show Only Barrier Fault

Type str

static_eclipse_case
Static Case

Type str

thermal_expansion_coefficient
Thermal Expansion Coefficient [1/C]

Type float

thickness_direction
Thickness Direction

Type str

thickness_direction_well_path
Thickness Direction Well Path

Type str

time_step
Time Step

Type int

use_detailed_fluid_loss
Use Detailed Fluid Loss

Type str

well_penetration_layer
Well Penetration Layer

Type int

Methods Summary

export_to_file([directory_path])	Export StimPlan Model Plot to File

Methods Documentation

export_to_file(*directory_path=''*)
Export StimPlan Model Plot to File

Parameters **directory_path** (*str*) – Directory Path

Returns StimPlanModel

StimPlanModelCollection

class rips.**StimPlanModelCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.CheckableNamedObject

Methods Summary

append_stim_plan_model([well_path, ...])	Create a new StimPlan Model
stim_plan_models()	**returns** List of StimPlanModel

Methods Documentation

append_stim_plan_model(*well_path=''*, *measured_depth=0*, *stim_plan_model_template=''*)
Create a new StimPlan Model

Parameters

- **well_path** (*WellPathBase*) – Well Path
- **measured_depth** (*float*) –
- **stim_plan_model_template** (StimPlanModelTemplate) – StimPlan Model Template

Returns StimPlanModel

stim_plan_models()

Returns List of StimPlanModel

StimPlanModelPlot

class rips.**StimPlanModelPlot**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.DepthTrackPlot

A fracture model plot

eclipse_case
Case

Type str

stim_plan_model
StimPlan Model

Type str

time_step
Time Step

Type int

StimPlanModelPlotCollection

class rips.**StimPlanModelPlotCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

append_stim_plan_model_plot([stim_plan_model])	Create a new StimPlan Model
stim_plan_model_plots()	**returns** List of StimPlanModelPlot

Methods Documentation

append_stim_plan_model_plot(*stim_plan_model=''*)
Create a new StimPlan Model

Parameters **stim_plan_model** (StimPlanModel) – StimPlan Model

Returns StimPlanModelPlot

stim_plan_model_plots()

Returns List of StimPlanModelPlot

StimPlanModelTemplate

class rips.**StimPlanModelTemplate**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.NamedObject

default_permeability
Default Permeability

Type float

default_porosity
Default Porosity

Type float

dynamic_eclipse_case
Dynamic Case

Type str

id
ID

Type int

initial_pressure_eclipse_case
Initial Pressure Case

Type str

overburden_facies
Overburden Facies

Type str

overburden_fluid_density
Overburden Fluid Density [g/cm^3]

Type float

overburden_formation
Overburden Formation

Type str

overburden_height
Overburden Height

Type float

overburden_permeability
Overburden Permeability

Type float

overburden_porosity
Overburden Porosity

Type float

reference_temperature
Temperature [C]

Type float

reference_temperature_depth
Temperature Depth [m]

Type float

reference_temperature_gradient
Temperature Gradient [C/m]

Type float

static_eclipse_case
Static Case

Type str

stress_depth
Stress Depth

Type float

time_step
Time Step

Type int

underburden_facies
Underburden Facies

Type str

underburden_fluid_density
Underburden Fluid Density [g/cm^3]

Type float

underburden_formation
Underburden Formation

Type str

underburden_height
Underburden Height

Type float

underburden_permeability
Underburden Permeability

Type float

underburden_porosity
Underburden Porosity

Type float

vertical_stress
Vertical Stress

Type float

vertical_stress_gradient
Vertical Stress Gradient

Type float

Methods Summary

elastic_properties()	Elastic Properties
facies_initial_pressure_configs()	Facies Initial Pressure Configs
facies_properties()	Facies Properties
non_net_layers()	Non-Net Layers
pressure_table()	Pressure Table

Methods Documentation

elastic_properties()
Elastic Properties

Returns ElasticProperties

facies_initial_pressure_configs()
Facies Initial Pressure Configs

Returns List of FaciesInitialPressureConfig

facies_properties()
Facies Properties

Returns FaciesProperties

non_net_layers()
Non-Net Layers

Returns NonNetLayers

pressure_table()
Pressure Table

Returns PressureTable

StimPlanModelTemplateCollection

class rips.**StimPlanModelTemplateCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

append_stim_plan_model_template([...])	Create a new StimPlan Model Template
stim_plan_model_templates()	StimPlan Model Templates

Methods Documentation

append_stim_plan_model_template(*eclipse_case=''*, *time_step=0*, *facies_properties_file_path=''*, *elastic_properties_file_path=''*)
Create a new StimPlan Model Template

Parameters

- **eclipse_case** (*RimReservoir*) – Eclipse Case
- **time_step** (*int*) – Time Step
- **facies_properties_file_path** (*str*) – Facies Properties File Path
- **elastic_properties_file_path** (*str*) – Elastic Properties File Path

Returns StimPlanModelTemplate

stim_plan_model_templates()
StimPlan Model Templates

Returns List of StimPlanModelTemplate

SummaryCase

class rips.**SummaryCase**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

The Base Class for all Summary Cases

auto_shorty_name
Use Auto Display Name

Type str

id
Case ID

Type int

name_setting
One of [FULL_CASE_NAME, SHORT_CASE_NAME, CUSTOM_NAME]

Type str

short_name
Display Name

Type str

summary_header_filename
Summary Header File

Type str

Methods Summary

available_addresses()	Arguments:
available_time_steps()	Arguments:
resample_values([address, resampling_period])	**param address** Formatted address specifying the summary vector
summary_vector_values([address])	Get all values for a summary vector

Methods Documentation

available_addresses()
Arguments:

Returns DataContainerString

available_time_steps()
Arguments:

Returns DataContainerTime

resample_values(*address=''*, *resampling_period=''*)

Parameters

- **address** (*str*) – Formatted address specifying the summary vector
- **resampling_period** (*str*) – Resampling Period

Returns ResampleData

summary_vector_values(*address=''*)
Get all values for a summary vector

Parameters **address** (*str*) – Formatted address specifying the summary vector

Returns DataContainerFloat

SummaryCaseSubCollection

class rips.**SummaryCaseSubCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

id
Ensemble ID

Type int

is_ensemble
Is Ensemble

Type str

name_count
Name

Type str

summary_collection_name
Name

Type str

SummaryPlot

class rips.**SummaryPlot**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.Plot

A Summary Plot

is_using_auto_name
Auto Title

Type str

normalize_curve_y_values
Normalize all curves

Type str

plot_description
Name

Type str

SummaryPlotCollection

class rips.**SummaryPlotCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

new_summary_plot([summary_cases, ensemble, ...])	Create a new Summary Plot

Methods Documentation

new_summary_plot(*summary_cases=[]*, *ensemble=''*, *address=''*)
Create a new Summary Plot

Parameters

- **summary_cases** (*List of SummaryCase*) – Summary Cases
- **ensemble** (SummaryCaseSubCollection) – Ensemble
- **address** (*str*) – Formatted address string specifying the plot options

Returns SummaryPlot

Surface

class rips.**Surface**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.SurfaceInterface

SurfaceCollection

class rips.**SurfaceCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

surface_user_decription
Name

Type str

Methods Summary

add_folder([folder_name])	Add a new surface folder
import_surface([file_name])	Import a new surface from file
new_surface([case, k_index])	Create a new surface
sub_collections()	Surfaces
surfaces_field()	Surfaces

Methods Documentation

add_folder(*folder_name='Surfaces'*)
Add a new surface folder

Parameters **folder_name** (*str*) – New surface folder name

Returns SurfaceCollection

import_surface(*file_name=''*)
Import a new surface from file

Parameters **file_name** (*str*) – Filename to import surface from

Returns Surface

new_surface(*case=''*, *k_index=-1*)
Create a new surface

Parameters

- **case** (Case) –
- **k_index** (*int*) –

Returns GridCaseSurface

sub_collections()
Surfaces

Returns List of SurfaceCollection

surfaces_field()
Surfaces

Returns List of SurfaceInterface

SurfaceInterface

class rips.**SurfaceInterface**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

depth_offset
Depth Offset

Type float

surface_user_decription
Name

Type str

Methods Summary

export_to_file([file_name])	Export a surface to file

Methods Documentation

export_to_file(*file_name=''*)
Export a surface to file

Parameters **file_name** (*str*) – Filename to export surface to

Returns DataContainerString

View

class rips.**View**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.ViewWindow

background_color
Background

Type str

current_time_step
Current Time Step

Type int

disable_lighting
Disable Results Lighting

Type str

grid_z_scale
Z Scale

Type float

id
View ID

Type int

perspective_projection
Perspective Projection

Type str

show_grid_box
Show Grid Box

Type str

show_z_scale
Show Z Scale Label

Type str

Methods Summary

apply_cell_result(result_type, result_variable)	Apply a regular cell result
apply_flow_diagnostics_cell_result([...])	Apply a flow diagnostics cell result
clone()	Clone the current view
export_property([undefined_value])	Export the current Eclipse property from the view
export_sim_well_fracture_completions(...)	Export fracture completions for simulation wells
export_visible_cells([export_keyword, ...])	Export special properties for all visible cells.
set_time_step(time_step)	Set the time step for current view

Methods Documentation

apply_cell_result(*result_type*, *result_variable*)
Apply a regular cell result

Parameters

- **result_type** (*str*) –

String representing the result category. The valid values are::

- DYNAMIC_NATIVE
- STATIC_NATIVE
- SOURSIMRL
- GENERATED
- INPUT_PROPERTY
- FORMATION_NAMES
- FLOW_DIAGNOSTICS
- INJECTION_FLOODING

- **result_variable** (*str*) – String representing the result variable.

apply_flow_diagnostics_cell_result(*result_variable='TOF'*, *selection_mode='FLOW_TR_BY_SELECTION'*, *injectors=None*, *producers=None*)

Apply a flow diagnostics cell result

Parameters:

```
Parameter           | Description                                                 ␣
↪| Type
------------------- | ------------------------------------------------------------␣
↪| -----
result_variable     | String representing the result value                        ␣
↪| String
selection_mode      | String specifying which tracers to select                   ␣
↪| String
injectors           | List of injector names, used by 'FLOW_TR_BY_SELECTION'      ␣
↪| String List
producers           | List of injector names, used by 'FLOW_TR_BY_SELECTION'      ␣
↪| String List
```

Enum compdat_export:

```
Option                  | Description
------------------------| ------------
"TOF"                   | Time of flight
"Fraction"              | Fraction
"MaxFractionTracer"     | Max Fraction Tracer
"Communication"         | Communication
```

clone()

Clone the current view

export_property(*undefined_value=0.0*)

Export the current Eclipse property from the view

Parameters **undefined_value** (*double*) – Value to use for undefined values. Defaults to 0.0

export_sim_well_fracture_completions(*time_step*, *simulation_well_names*, *file_split*, *compdat_export*)

Export fracture completions for simulation wells

Parameters:

```
Parameter                  | Description                                    ␣
↪   | Type
---------------------------| ---------------------------------------------
↪- | -----
time_step                  | Time step to export for                        ␣
↪   | Integer
simulation_well_names      | List of simulation well names                  ␣
↪   | List
file_split                 | Controls how export data is split into files   ␣
↪   | String enum
compdat_export             | Compdat export type                            ␣
↪   | String enum
```

Enum file_split:

```
Option                             | Description
---------------------------------- | ------------
"UNIFIED_FILE" <b>Default Option</b>| A single file with all␣
↪transmissibilities
"SPLIT_ON_WELL"                    | One file for each well␣
↪transmissibilities
"SPLIT_ON_WELL_AND_COMPLETION_TYPE" | One file for each completion type for␣
↪each well
```

Enum compdat_export:

```
Option                                    | Description
------------------------------------------| ------------
"TRANSMISSIBILITIES"<b>Default Option</b>| Direct export of␣
↪transmissibilities
"WPIMULT_AND_DEFAULT_CONNECTION_FACTORS" | Include export of WPIMULT
```

export_visible_cells(*export_keyword='FLUXNUM'*, *visible_active_cells_value=1*, *hidden_active_cells_value=0*, *inactive_cells_value=0*)

Export special properties for all visible cells.

Parameters

- **export_keyword** (*string*) – The keyword to export.
- **Choices** – 'FLUXNUM' or 'MULTNUM'. Default: 'FLUXNUM'
- **visible_active_cells_value** (*int*) – Value to export forvisible active cells. Default: 1
- **hidden_active_cells_value** (*int*) – Value to export for hidden active cells. Default: 0
- **inactive_cells_value** (*int*) – Value to export for inactive cells. Default: 0

set_time_step(*time_step*)

Set the time step for current view

ViewWindow

class rips.**ViewWindow**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

The Base Class for all Views and Plots in ResInsight

Methods Summary

case()	Get the case the view belongs to
export_snapshot([prefix, export_folder])	Export snapshot for the current view

Methods Documentation

case()
Get the case the view belongs to

export_snapshot(*prefix=''*, *export_folder=''*)
Export snapshot for the current view

Parameters

- **prefix** (*str*) – Exported file name prefix
- **export_folder** (*str*) – The path to export to. By default will use the global export folder

WbsParameters

class rips.**WbsParameters**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

df_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

fg_multiplier
SH Multiplier for FG in Shale

Type float

fg_shale_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

k0_fg_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

k0_sh_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

obg0_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

poission_ratio_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

pore_pressure_non_reservoir_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

pore_pressure_reservoir_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

ucs_source
One of [GRID, LAS_FILE, ELEMENT_PROPERTY_TABLE, USER_DEFINED, HYDROSTATIC, DERIVED_FROM_K0FG, PROPORTIONAL_TO_SH, UNDEFINED]

Type str

user_df
User Defined DF

Type float

user_k0_fg
User Defined K0_FG

Type float

user_k0_sh
User Defined K0_SH

Type float

user_poisson_ratio
User Defined Poisson Ratio

Type float

user_pp_non_reservoir
Multiplier of hydrostatic PP

Type float

user_ucs
User Defined UCS [bar]

Type float

water_density
Density of Sea Water [g/cm^3]

Type float

WellBoreStabilityPlot

class rips.**WellBoreStabilityPlot**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.WellLogPlot

A GeoMechanical Well Bore Stabilit Plot

Methods Summary

parameters()	Well Bore Stability Parameters

Methods Documentation

parameters()
Well Bore Stability Parameters

Returns WbsParameters

WellLogExtractionCurve

class rips.**WellLogExtractionCurve**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.WellLogPlotCurve

WellLogPlot

class rips.**WellLogPlot**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.DepthTrackPlot

A Well Log Plot With a shared Depth Axis and Multiple Tracks

Methods Summary

export_data_as_ascii(export_folder[, ...])	Export LAS file(s) for the current plot
export_data_as_las(export_folder[, ...])	Export LAS file(s) for the current plot
new_well_log_track([title, case, well_path])	Create a new well log track

Methods Documentation

export_data_as_ascii(*export_folder*, *file_prefix=''*, *capitalize_file_names=False*)
Export LAS file(s) for the current plot

Parameters

- **export_folder** (*str*) – The path to export to. By default will use the global export folder
- **file_prefix** (*str*) – Exported file name prefix
- **capitalize_file_names** (*bool*) – Make all file names upper case

Returns A list of files exported

export_data_as_las(*export_folder*, *file_prefix=''*, *export_tvdrkb=False*, *capitalize_file_names=False*, *resample_interval=0.0*, *convert_to_standard_units=False*)
Export LAS file(s) for the current plot

Parameters

- **export_folder** (*str*) – The path to export to. By default will use the global export folder
- **file_prefix** (*str*) – Exported file name prefix
- **export_tvdrkb** (*bool*) – Export in TVD-RKB format
- **capitalize_file_names** (*bool*) – Make all file names upper case
- **resample_interval** (*double*) – if > 0.0 the files will be resampled

Returns A list of files exported

new_well_log_track(*title=''*, *case=''*, *well_path=''*)
Create a new well log track

Parameters

- **title** (*str*) – Title
- **case** (*RimReservoir*) – Case
- **well_path** (*WellPathBase*) – Well Path

Returns WellLogPlotTrack

WellLogPlotCollection

class rips.**WellLogPlotCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Methods Summary

new_well_log_plot([case, well_path, ...])	Create a new well log plot
well_log_plots()	**returns** List of WellLogPlot

Methods Documentation

new_well_log_plot(*case=''*, *well_path=''*, *property_type=''*, *property_name=''*, *time_step=0*)
Create a new well log plot

Parameters

- **case** (*RimReservoir*) – Case
- **well_path** (*WellPathBase*) – Well Path
- **property_type** (*str*) – Property Type
- **property_name** (*str*) – Property Name
- **time_step** (*int*) – Time Step

Returns WellLogPlot

well_log_plots()

Returns List of WellLogPlot

WellLogPlotCurve

class rips.**WellLogPlotCurve**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.PlotCurve

WellLogPlotTrack

class rips.**WellLogPlotTrack**(*pb2_object=None*, *channel=None*)
Bases: rips.generated.generated_classes.Plot

Methods Summary

add_extraction_curve([case, well_path, ...])	Create a well log extraction curve

Methods Documentation

add_extraction_curve(*case=''*, *well_path=''*, *property_type=''*, *property_name=''*, *time_step=0*)
Create a well log extraction curve

Parameters

- **case** (*RimReservoir*) – Case
- **well_path** (*WellPathBase*) – Well Path
- **property_type** (*str*) – Property Type
- **property_name** (*str*) – Property Name
- **time_step** (*int*) – Time Step

Returns WellLogExtractionCurve

WellPath

class rips.**WellPath**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

A ResInsight Well Path

name
Name

Type str

Methods Summary

add_fracture([measured_depth, ...])	Add StimPlan Fracture
append_perforation_interval([start_md, ...])	Append Perforation Interval

Methods Documentation

add_fracture(*measured_depth=0*, *stim_plan_fracture_template=''*)
Add StimPlan Fracture

Parameters

- **measured_depth** (*float*) –
- **stim_plan_fracture_template** (StimPlanFractureTemplate) – StimPlan Fracture Template

Returns WellPathFracture

append_perforation_interval(*start_md=0*, *end_md=0*, *diameter=0*, *skin_factor=0*)
Append Perforation Interval

Parameters

- **start_md** (*float*) – Start Measured Depth
- **end_md** (*float*) – End Measured Depth
- **diameter** (*float*) – Diameter
- **skin_factor** (*float*) – Skin Factor

Returns Perforation

WellPathCollection

class rips.**WellPathCollection**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Collection of Well Paths

Methods Summary

well_paths()	Well Paths

Methods Documentation

well_paths()
Well Paths

Returns List of WellPath

WellPathGeometry

class rips.**WellPathGeometry**(*pb2_object=None*, *channel=None*)
Bases: rips.pdmobject.PdmObjectBase

Class containing the geometry of a modeled Well Path

air_gap
Air Gap

Type float

attached_to_parent_well
Attached to Parent Well

Type str

link_reference_point_updates
Link Reference Point

Type str

md_at_first_target
MD at First Target

Type float

reference_point
UTM Reference Point

Type str

show_spheres
Spheres

Type str

use_auto_generated_target_at_sea_level
Generate Target at Sea Level

Type str

Methods Summary

append_well_target([coordinate, absolute])	Create and Add New Well Target
auto_generated_target()	Auto Generated Target
well_path_targets()	Well Targets

Methods Documentation

append_well_target(*coordinate=[0, 0, 0]*, *absolute=False*)
Create and Add New Well Target

Parameters

- **coordinate** (*class cvf::Vector3<double>*) – Coordinate
- **absolute** (*bool*) – Relative or Absolute Coordinate

Returns WellPathTarget

auto_generated_target()
Auto Generated Target

Returns WellPathTarget

well_path_targets()
Well Targets

Returns List of WellPathTarget

WellPathTarget

class rips.**WellPathTarget**(*pb2_object=None*, *channel=None*)

Bases: rips.pdmobject.PdmObjectBase

Class containing the Well Target definition

azimuth
Azi(deg)

Type float

dogleg1
DL in

Type float

dogleg2
DL out

Type float

inclination
Inc(deg)

Type float

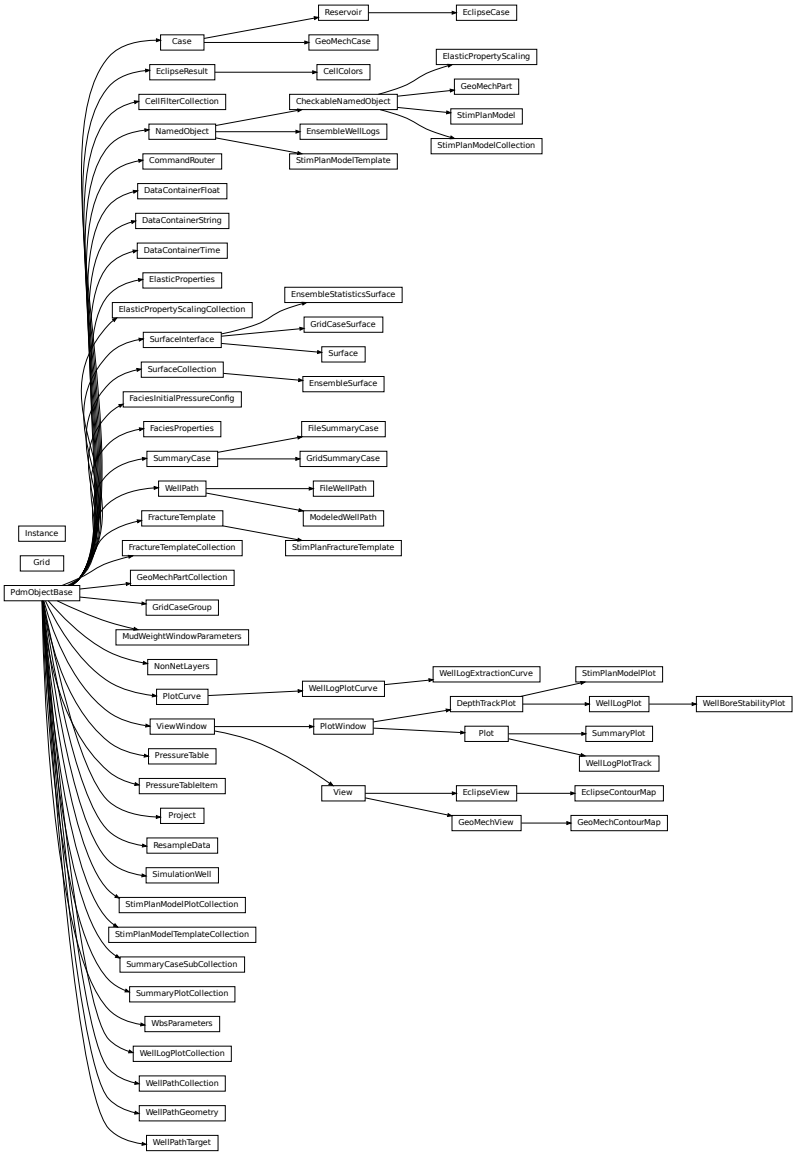
target_measured_depth
MD

Type float

target_point
Relative Coord

Type str

Class Inheritance Diagram



2.4 Python Examples

This pages is created based on the content in the **PythonExamples** folder located inside the **rips** module, made available online for convenience.

2.4.1 All Cases

```
############################################################################
# This example will connect to ResInsight, retrieve a list of cases and print info
#
############################################################################

# Import the ResInsight Processing Server Module
import rips

# Connect to ResInsight
resinsight = rips.Instance.find()
if resinsight is not None:
    # Get a list of all cases
    cases = resinsight.project.cases()

    print("Got " + str(len(cases)) + " cases: ")
    for case in cases:
        print("Case id: " + str(case.id))
        print("Case name: " + case.name)
        print("Case type: " + case.__class__.__name__)
        print("Case file name: " + case.file_path)
        print("Case reservoir bounding box:", case.reservoir_boundingbox())

        timesteps = case.time_steps()
        for t in timesteps:
            print("Year: " + str(t.year))
            print("Month: " + str(t.month))

        if isinstance(case, rips.EclipseCase):
            print("Getting coarsening info for case: ", case.name, case.id)
            coarsening_info = case.coarsening_info()
            if coarsening_info:
                print("Coarsening information:")

            for c in coarsening_info:
                print(
                    "[{}, {}, {}] - [{}, {}, {}]".format(
                        c.min.x, c.min.y, c.min.z, c.max.x, c.max.y, c.max.z
                    )
                )
```

2.4.2 All Simulation Wells

```
############################################################################
# This example will connect to ResInsight, retrieve a list of
# simulation wells and print info
############################################################################

# Import the ResInsight Processing Server Module
import rips

# Connect to ResInsight
resinsight = rips.Instance.find()
if resinsight is not None:
```

(continues on next page)

(continued from previous page)

```
    # Get a list of all wells
    cases = resinsight.project.cases()

    for case in cases:
        print("Case id: " + str(case.id))
        print("Case name: " + case.name)

        timesteps = case.time_steps()
        sim_wells = case.simulation_wells()
        for sim_well in sim_wells:
            print("Simulation well: " + sim_well.name)

            for (tidx, timestep) in enumerate(timesteps):
                status = sim_well.status(tidx)
                cells = sim_well.cells(tidx)
                print(
                    "timestep: "
                    + str(tidx)
                    + " type: "
                    + status.well_type
                    + " open: "
                    + str(status.is_open)
                    + " cells:"
                    + str(len(cells))
                )
```

2.4.3 All Wells

```
############################################################################################
# This example will connect to ResInsight, retrieve a list of wells and print info
#
############################################################################################

# Import the ResInsight Processing Server Module
import rips

# Connect to ResInsight
resinsight = rips.Instance.find()
if resinsight is not None:
    # Get a list of all wells
    wells = resinsight.project.well_paths()

    print("Got " + str(len(wells)) + " wells: ")
    for well in wells:
        print("Well name: " + well.name)
```

2.4.4 Alter Wbs Plot

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
```

(continues on next page)

(continued from previous page)

```
# Connect to ResInsight instance
resinsight = rips.Instance.find()

# Get the project
project = resinsight.project

# Find all the well bore stability plots in the project
wbsplots = project.descendants(rips.WellBoreStabilityPlot)

# Chose a sensible output folder
dirname = tempfile.gettempdir()

# Loop through all Well Bore Stability plots
for wbsplot in wbsplots:
    # Set depth type a parameter and export snapshot
    wbsplot.depth_type = "TRUE_VERTICAL_DEPTH_RKB"

    # Example of setting parameters for existing plots
    params = wbsplot.parameters()
    params.user_poisson_ratio = 0.12345
    params.update()
    wbsplot.update()
    wbsplot.export_snapshot(export_folder=dirname)
```

2.4.5 Case Grid Group

```
import os
import rips

resinsight = rips.Instance.find()

case_paths = []
case_paths.append(
    "C:/Users/lindk/source/repos/ResInsight/TestModels/Case_with_10_timesteps/Real0/
↪BRUGGE_0000.EGRID"
)
case_paths.append(
    "C:/Users/lindk/source/repos/ResInsight/TestModels/Case_with_10_timesteps/Real10/
↪BRUGGE_0010.EGRID"
)
for case_path in case_paths:
    assert os.path.exists(
        case_path
    ), "You need to set valid case paths for this script to work"

case_group = resinsight.project.create_grid_case_group(case_paths=case_paths)

case_group.print_object_info()

# stat_cases = caseGroup.statistics_cases()
# case_ids = []
# for stat_case in stat_cases:
#     stat_case.set_dynamic_properties_to_calculate(["SWAT"])
#     case_ids.append(stat_case.id)
```

(continues on next page)

(continued from previous page)

```
case_group.compute_statistics()

view = case_group.views()[0]
cell_result = view.cell_result()
cell_result.set_result_variable("PRESSURE_DEV")
```

2.4.6 Case Info Streaming Example

```
############################################################################
# This example will get the cell info for the active cells for the first case
############################################################################

# Import the ResInsight Processing Server Module
import rips

# Connect to ResInsight
resinsight = rips.Instance.find()

# Get the first case. This will fail if you haven't loaded any cases
case = resinsight.project.cases()[0]

# Get the cell count object
cell_counts = case.cell_count()
print("Number of active cells: " + str(cell_counts.active_cell_count))
print("Total number of reservoir cells: " + str(cell_counts.reservoir_cell_count))

# Get information for all active cells
active_cell_infos = case.cell_info_for_active_cells()

# A simple check on the size of the cell info
assert cell_counts.active_cell_count == len(active_cell_infos)

# Print information for the first active cell
print("First active cell: ")
print(active_cell_infos[0])
```

2.4.7 Cell Result Data

```
#####################################################################
# This script retrieves cell result data and alters it
#####################################################################
import rips

resinsight = rips.Instance.find()

view = resinsight.project.views()[0]
results = view.cell_result_data()
print("Number of result values: ", len(results))

newresults = []
for i in range(0, len(results)):
    newresults.append(results[i] * -1.0)
view.set_cell_result_data(newresults)
```

2.4.8 Command Example

```
########################################################################
# This example will show setting time step, window size and export snapshots and␣
↪properties
########################################################################
import os
import tempfile
import rips

# Load instance
resinsight = rips.Instance.find()

# Set window sizes
resinsight.set_main_window_size(width=800, height=500)
resinsight.set_plot_window_size(width=1000, height=1000)


# Retrieve first case
case = resinsight.project.cases()[0]

# Get a view
view1 = case.views()[0]

# Clone the view
view2 = view1.clone()

# Set the time step for view1 only
view1.set_time_step(time_step=2)

# Set cell result to SOIL
view1.apply_cell_result(result_type="DYNAMIC_NATIVE", result_variable="SOIL")


# Create a temporary directory which will disappear at the end of this script
# If you want to keep the files, provide a good path name instead of tmpdirname
with tempfile.TemporaryDirectory(prefix="rips") as tmpdirname:
    print("Temporary folder: ", tmpdirname)

    # Set export folder for snapshots and properties
    resinsight.set_export_folder(export_type="SNAPSHOTS", path=tmpdirname)
    resinsight.set_export_folder(export_type="PROPERTIES", path=tmpdirname)

    # Export all snapshots
    resinsight.project.export_snapshots()

    assert len(os.listdir(tmpdirname)) > 0

    # Export properties in the view
    view1.export_property()

    # Check that the exported file exists
    expected_file_name = case.name + "-" + str("3D_View") + "-" + "T2" + "-SOIL"
    full_path = tmpdirname + "/" + expected_file_name

    # Print contents of temporary folder
    print(os.listdir(tmpdirname))
```

(continues on next page)

(continued from previous page)

```
    assert os.path.exists(full_path)
```

2.4.9 Create And Export Stim Plan Model

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
from os.path import expanduser
from pathlib import Path

# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Example code
project = resinsight.project

# Look for input files in the home directory of the user
home_dir = expanduser("~")
elastic_properties_file_path = (Path(home_dir) / "elastic_properties.csv").as_posix()
print("Elastic properties file path:", elastic_properties_file_path)

facies_properties_file_path = (Path(home_dir) / "facies_id.roff").as_posix()
print("Facies properties file path:", facies_properties_file_path)

# Find a case
cases = resinsight.project.cases()
case = cases[1]

# Use the last time step
time_steps = case.time_steps()
time_step = time_steps[len(time_steps) - 1]


# Create stim plan model template
fmt_collection = project.descendants(rips.StimPlanModelTemplateCollection)[0]
stim_plan_model_template = fmt_collection.append_stim_plan_model_template(
    eclipse_case=case,
    time_step=time_step,
    elastic_properties_file_path=elastic_properties_file_path,
    facies_properties_file_path=facies_properties_file_path,
)
stim_plan_model_template.overburden_formation = "Garn"
stim_plan_model_template.overburden_facies = "Shale"
stim_plan_model_template.underburden_formation = "Garn"
stim_plan_model_template.underburden_facies = "Shale"
stim_plan_model_template.overburden_height = 68
stim_plan_model_template.update()
print("Overburden: ", stim_plan_model_template.overburden_formation)


# Set eclipse result for facies definition
eclipse_result = stim_plan_model_template.facies_properties().facies_definition()
eclipse_result.result_type = "INPUT_PROPERTY"
eclipse_result.result_variable = "OPERNUM_1"
eclipse_result.update()
```

(continues on next page)

(continued from previous page)

```
# Set eclipse result for non-net layers
non_net_layers = stim_plan_model_template.non_net_layers()
non_net_layers_result = non_net_layers.facies_definition()
non_net_layers_result.result_type = "STATIC_NATIVE"
non_net_layers_result.result_variable = "NTG"
non_net_layers_result.update()
non_net_layers.formation = "Not"
non_net_layers.facies = "Shale"
non_net_layers.update()


# Add some scaling factors
elastic_properties = stim_plan_model_template.elastic_properties()
elastic_properties.add_property_scaling(
    formation="Garn", facies="Calcite", property="YOUNGS_MODULUS", scale=1.44
)


well_name = "B-2 H"

# Find a well
well_path = project.well_path_by_name(well_name)
print("well path:", well_path)
stim_plan_model_collection = project.descendants(rips.StimPlanModelCollection)[0]


export_folder = tempfile.gettempdir()

stim_plan_models = []

# Create and export a StimPlan model for each depth
measured_depths = [3200.0, 3400.0, 3600.0]
for measured_depth in measured_depths:

    # Create stim plan model at a give measured depth
    stim_plan_model = stim_plan_model_collection.append_stim_plan_model(
        well_path=well_path,
        measured_depth=measured_depth,
        stim_plan_model_template=stim_plan_model_template,
    )
    stim_plan_models.append(stim_plan_model)

    # Make the well name safer to use as a directory path
    well_name_part = well_name.replace(" ", "_")
    directory_path = Path(export_folder) / "{}_{}".format(
        well_name_part, int(measured_depth)
    )

    # Create the folder
    directory_path.mkdir(parents=True, exist_ok=True)

    print("Exporting fracture model to: ", directory_path)
    stim_plan_model.export_to_file(directory_path=directory_path.as_posix())

    # Create a fracture mode plot
    stim_plan_model_plot_collection = project.descendants(
```

(continues on next page)

(continued from previous page)

```
        rips.StimPlanModelPlotCollection
    )[0]
    stim_plan_model_plot = stim_plan_model_plot_collection.append_stim_plan_model_
↪plot(
        stim_plan_model=stim_plan_model
    )

    print("Exporting fracture model plot to: ", directory_path)
    stim_plan_model_plot.export_snapshot(export_folder=directory_path.as_posix())


print("Setting measured depth and perforation length.")
stim_plan_models[0].measured_depth = 3300.0
stim_plan_models[0].perforation_length = 123.445
stim_plan_models[0].update()
```

2.4.10 Create Wbs Plot

```
import os
import grpc

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resInsight = rips.Instance.find()

# Get all GeoMech cases
cases = resInsight.project.descendants(rips.GeoMechCase)

# Get all well paths
well_paths = resInsight.project.well_paths()

# Ensure there's at least one well path
if len(well_paths) < 1:
    print("No well paths in project")
    exit(1)

# Create a set of WbsParameters
params = rips.WbsParameters()
params.user_poisson_ratio = 0.23456
params.user_ucs = 123

# Loop through all cases
for case in cases:
    assert isinstance(case, rips.GeoMechCase)
    min_res_depth, max_res_depth = case.reservoir_depth_range()

    # Find a good output path
    case_path = case.file_path
    folder_name = os.path.dirname(case_path)

    # Import formation names
    case.import_formation_names(
        formation_files=[
```

(continues on next page)

(continued from previous page)

```
            "D:/Projects/ResInsight-regression-test/ModelData/norne/Norne_ATW2013.lyr"
        ]
    )

    # create a folder to hold the snapshots
    dirname = os.path.join(folder_name, "snapshots")
    print("Exporting to: " + dirname)

    for well_path in well_paths[0:4]:  # Loop through the first five well paths
        # Create plot with parameters
        wbsplot = case.create_well_bore_stability_plot(
            well_path=well_path.name, time_step=0, parameters=params
        )
```

2.4.11 Error Handling

```
########################################################################
# This example demonstrates the use of ResInsight exceptions
# for proper error handling
########################################################################

import rips
import grpc
import tempfile

resinsight = rips.Instance.find()

case = None

# Try loading a non-existing case. We should get a grpc.RpcError exception from the␣
↪server
try:
    case = resinsight.project.load_case("Nonsense")
except grpc.RpcError as e:
    print(
        "Expected Server Exception Received while loading case: ", e.code(), e.
↪details()
    )

# Try loading well paths from a non-existing folder.  We should get a grpc.RpcError␣
↪exception from the server
try:
    well_path_files = resinsight.project.import_well_paths(
        well_path_folder="NONSENSE/NONSENSE"
    )
except grpc.RpcError as e:
    print(
        "Expected Server Exception Received while loading wellpaths: ",
        e.code(),
        e.details(),
    )

# Try loading well paths from an existing but empty folder. We should get a warning.
try:
    with tempfile.TemporaryDirectory() as tmpdirname:
```

(continues on next page)

(continued from previous page)

```
        well_path_files = resinsight.project.import_well_paths(
            well_path_folder=tmpdirname
        )
        assert len(well_path_files) == 0
        assert resinsight.project.has_warnings()
        print("Should get warnings below")
        for warning in resinsight.project.warnings():
            print(warning)
except grpc.RpcError as e:
    print("Unexpected Server Exception caught!!!", e)

case = resinsight.project.case(case_id=0)
if case is not None:
    results = case.active_cell_property("STATIC_NATIVE", "PORO", 0)
    active_cell_count = len(results)

    # Send the results back to ResInsight inside try / except construct
    try:
        case.set_active_cell_property(results, "GENERATED", "POROAPPENDED", 0)
        print("Everything went well as expected")
    except:  # Match any exception, but it should not happen
        print("Ooops!")

    # Add another value, so this is outside the bounds of the active cell result␣
↪storage
    results.append(1.0)

    # This time we should get a grpc.RpcError exception, which is a server side error.
    try:
        case.set_active_cell_property(results, "GENERATED", "POROAPPENDED", 0)
        print("Everything went well??")
    except grpc.RpcError as e:
        print("Expected Server Exception Received: ", e)
    except IndexError:
        print("Got index out of bounds error. This shouldn't happen here")

    # With a chunk size exactly matching the active cell count the server will not
    # be able to see any error as it will successfully close the stream after␣
↪receiving
    # the correct number of values, even if the python client has more chunks to send
    case.chunk_size = active_cell_count

    try:
        case.set_active_cell_property(results, "GENERATED", "POROAPPENDED", 0)
        print("Everything went well??")
    except grpc.RpcError as e:
        print("Got unexpected server exception", e, "This should not happen now")
    except IndexError:
        print("Got expected index out of bounds error on client side")
```

2.4.12 Export Contour Maps

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
```

(continues on next page)

(continued from previous page)

```
import pathlib

# Connect to ResInsight instance
resInsight = rips.Instance.find()

# Data will be written to temp
tmpdir = pathlib.Path(tempfile.gettempdir())

# Find all eclipse contour maps of the project
contour_maps = resInsight.project.descendants(rips.EclipseContourMap)
print("Number of eclipse contour maps:", len(contour_maps))

# Export the contour maps to a text file
for (index, contour_map) in enumerate(contour_maps):
    filename = "eclipse_contour_map" + str(index) + ".txt"
    filepath = tmpdir / filename
    print("Exporting to:", filepath)
    contour_map.export_to_text(str(filepath))

# The contour maps is also available for a Case
cases = resInsight.project.cases()
for case in cases:
    contour_maps = case.descendants(rips.GeoMechContourMap)
    # Export the contour maps to a text file
    for (index, contour_map) in enumerate(contour_maps):
        filename = "geomech_contour_map" + str(index) + ".txt"
        filepath = tmpdir / filename
        print("Exporting to:", filepath)
        contour_map.export_to_text(str(filepath))
```

2.4.13 Export Plots

```
# Import the tempfile module
import tempfile

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resInsight = rips.Instance.find()

# Get a list of all plots
plots = resInsight.project.plots()

export_folder = tempfile.mkdtemp()

print("Exporting to: " + export_folder)

for plot in plots:
    plot.export_snapshot(export_folder=export_folder)
    plot.export_snapshot(export_folder=export_folder, output_format="PDF")
    if isinstance(plot, rips.WellLogPlot):
        plot.export_data_as_las(export_folder=export_folder)
        plot.export_data_as_ascii(export_folder=export_folder)
```

2.4.14 Export Snapshots

```
########################################################################
# This script will export snapshots for two properties in every loaded case
# And put them in a snapshots folder in the same folder as the case grid
########################################################################
import os
import rips

# Load instance
resinsight = rips.Instance.find()
cases = resinsight.project.cases()

# Set main window size
resinsight.set_main_window_size(width=800, height=500)

n = 5  # every n-th time_step for snapshot
property_list = ["SOIL", "PRESSURE"]  # list of parameter for snapshot

print("Looping through cases")
for case in cases:
    print("Case name: ", case.name)
    print("Case id: ", case.id)
    # Get grid path and its folder name
    case_path = case.file_path
    folder_name = os.path.dirname(case_path)

    # create a folder to hold the snapshots
    dirname = os.path.join(folder_name, "snapshots")

    if os.path.exists(dirname) is False:
        os.mkdir(dirname)

    print("Exporting to folder: " + dirname)
    resinsight.set_export_folder(export_type="SNAPSHOTS", path=dirname)

    time_steps = case.time_steps()
    print("Number of time_steps: " + str(len(time_steps)))

    for view in case.views():
        for property in property_list:
            view.apply_cell_result(
                result_type="DYNAMIC_NATIVE", result_variable=property
            )
        for time_step in range(0, len(time_steps), 10):
            view.set_time_step(time_step=time_step)
            view.export_snapshot()
```

2.4.15 Export Well Path Completions

```
########################################################################
# This script will export completions for a well path for all cases in the project
#
########################################################################
```

(continues on next page)

(continued from previous page)

```
import os
import rips

# Load instance
resinsight = rips.Instance.find()
cases = resinsight.project.cases()

for case in cases:
    print("Case name: ", case.name)
    print("Case id: ", case.id)

    case.export_well_path_completions(
        time_step=0,
        well_path_names=["Well-1"],
        file_split="UNIFIED_FILE",
        include_perforations=True,
        custom_file_name="d:/scratch/well_path_export/myfile.myext",
    )
```

2.4.16 Generate Ensemble Of Well Logs

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
from os.path import expanduser
from pathlib import Path

# Connect to ResInsight instance
resinsight = rips.Instance.find()


home_dir = expanduser("~")


properties = [
    ("STATIC_NATIVE", "INDEX_K", 0),
    ("STATIC_NATIVE", "PORO", 0),
    ("STATIC_NATIVE", "PERMX", 0),
    ("DYNAMIC_NATIVE", "PRESSURE", 0),
]

export_folder = tempfile.mkdtemp()

directory_path = "resprojects/webviz-subsurface-testdata/reek_history_match/"


case_file_paths = []
num_realizations = 9
num_iterations = 4


for realization in range(0, num_realizations):
    for iteration in range(0, num_iterations):
        realization_dir = "realization-" + str(realization)
        iteration_dir = "iter-" + str(iteration)
```

(continues on next page)

(continued from previous page)

```
        egrid_name = "eclipse/model/5_R001_REEK-" + str(realization) + ".EGRID"
        path = Path(
            home_dir, directory_path, realization_dir, iteration_dir, egrid_name
        )
        case_file_paths.append(path)

for path in case_file_paths:
    # Load a case
    path_name = path.as_posix()
    grid_only = True
    case = resinsight.project.load_case(path_name, grid_only)

    # Load some wells
    well_paths = resinsight.project.import_well_paths(
        well_path_files=[
            Path(home_dir, directory_path, "wellpaths", "Well-1.dev").as_posix(),
            Path(home_dir, directory_path, "wellpaths", "Well-2.dev").as_posix(),
        ]
    )

    if resinsight.project.has_warnings():
        for warning in resinsight.project.warnings():
            print(warning)

    well_log_plot_collection = resinsight.project.descendants(
        rips.WellLogPlotCollection
    )[0]

    for well_path in well_paths:
        print(
            "Generating las file for well: " + well_path.name + " in case: " + path_
↪name
        )

        well_log_plot = well_log_plot_collection.new_well_log_plot(case, well_path)

        # Create a track for each property
        for (prop_type, prop_name, time_step) in properties:
            track = well_log_plot.new_well_log_track(
                "Track: " + prop_name, case, well_path
            )

            c = track.add_extraction_curve(
                case, well_path, prop_type, prop_name, time_step
            )

        parent_path = path.parent
        export_folder_path = Path(parent_path, "lasexport")
        export_folder_path.mkdir(parents=True, exist_ok=True)

        export_folder = export_folder_path.as_posix()
        well_log_plot.export_data_as_las(export_folder=export_folder)

    resinsight.project.close()
```

2.4.17 Generate Ensemble Surface

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
from os.path import expanduser
from pathlib import Path

# Connect to ResInsight instance
resinsight = rips.Instance.find()


home_dir = expanduser("~")

export_folder = tempfile.mkdtemp()

directory_path = "resprojects/webviz-subsurface-testdata/reek_history_match/"
# directory_path = "e:/gitroot/webviz-subsurface-testdata/reek_history_match"


case_file_paths = []
num_realizations = 9
num_iterations = 4


for realization in range(0, num_realizations):
    for iteration in range(0, num_iterations):
        realization_dir = "realization-" + str(realization)
        iteration_dir = "iter-" + str(iteration)
        egrid_name = "eclipse/model/5_R001_REEK-" + str(realization) + ".EGRID"
        path = Path(
            home_dir, directory_path, realization_dir, iteration_dir, egrid_name
        )
        case_file_paths.append(path)

k_indexes = [4, 10]

for path in case_file_paths:
    # Load a case
    path_name = path.as_posix()
    case = resinsight.project.load_case(path_name)

    if resinsight.project.has_warnings():
        for warning in resinsight.project.warnings():
            print(warning)

    surface_collection = resinsight.project.descendants(rips.SurfaceCollection)[0]

    for k_index in k_indexes:
        print("Generating surface K layer " + str(k_index) + " for case " + path_name)

        surface = surface_collection.new_surface(case, k_index)
        print("Surface: ", surface)

        parent_path = path.parent
        export_folder_path = Path(parent_path, "surfaceexport")
        export_folder_path.mkdir(parents=True, exist_ok=True)
```

(continues on next page)

(continued from previous page)

```
        export_file = Path(export_folder_path, "surf_" + str(k_index) + ".ts")
        print("Exporting to " + export_file.as_posix())
        surface.export_to_file(export_file.as_posix())

    # Close project to avoid aggregated memory usage
    # Can be replaced when case.close() is implemented
    resinsight.project.close()
```

2.4.18 Generate Ensemble Surface Optimized

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
from os.path import expanduser
from pathlib import Path

# Connect to ResInsight instance
resinsight = rips.Instance.find()


home_dir = expanduser("~")

export_folder = tempfile.mkdtemp()

directory_path = "resprojects/webviz-subsurface-testdata/reek_history_match/"
# directory_path = "e:/gitroot/webviz-subsurface-testdata/reek_history_match"


case_file_paths = []
num_realizations = 9
num_iterations = 4


for realization in range(0, num_realizations):
    for iteration in range(0, num_iterations):
        realization_dir = "realization-" + str(realization)
        iteration_dir = "iter-" + str(iteration)
        egrid_name = "eclipse/model/5_R001_REEK-" + str(realization) + ".EGRID"
        path = Path(
            home_dir, directory_path, realization_dir, iteration_dir, egrid_name
        )
        case_file_paths.append(path)

k_indexes = [4, 10]

command_router = resinsight.command_router

for path in case_file_paths:
    path_name = path.as_posix()

    command_router.extract_surfaces(path_name, k_indexes)
```

2.4.19 Grid Information

```
############################################################################################
# This example prints information about the grids of all cases in the current project
############################################################################################

import rips

resinsight = rips.Instance.find()

cases = resinsight.project.cases()
print("Number of cases found: ", len(cases))
for case in cases:
    print(case.name)
    grids = case.grids()
    print("Number of grids: ", len(grids))
    for grid in grids:
        print("Grid dimensions: ", grid.dimensions())
```

2.4.20 Import Fractures On Well

```
# Load ResInsight Processing Server Client Library
import rips
import tempfile
from os.path import expanduser
from pathlib import Path

# Connect to ResInsight instance
resinsight = rips.Instance.find()
project = resinsight.project

# Look for input files in the home directory of the user
home_dir = expanduser("~")
stim_plan_file_path = (Path(home_dir) / "contour.xml").as_posix()
print("StimPlan contour file path:", stim_plan_file_path)

# Find a case
cases = resinsight.project.cases()
case = cases[0]

# Create stim plan template
fmt_collection = project.descendants(rips.FractureTemplateCollection)[0]
fracture_template = fmt_collection.append_fracture_template(
    file_path=stim_plan_file_path
)

well_name = "B-2 H"

# Find a well
well_path = project.well_path_by_name(well_name)
print("well path:", well_path.name)

# Place fracture at given depths
measured_depths = [3200.0, 3400.0, 3600.0]
for measured_depth in measured_depths:
```

(continues on next page)

(continued from previous page)

```
    print("Placing fracture at {} depth (MD)".format(measured_depth))
    # Create stim plan  at a give measured depth
    fracture = well_path.add_fracture(
        measured_depth=measured_depth,
        stim_plan_fracture_template=fracture_template,
    )

# Update the orientation of the fracture
# Call update() to propagate changes from the Python object back to ResInsight
fracture_template.orientation = "Azimuth"
fracture_template.azimuth_angle = 60.0
fracture_template.update()
```

2.4.21 Import Well Paths And Logs

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resInsight = rips.Instance.find()

well_paths = resInsight.project.import_well_paths(
    well_path_folder="D:/Projects/ResInsight-regression-test/ModelData/norne/wellpaths
↪"
)
if resInsight.project.has_warnings():
    for warning in resInsight.project.warnings():
        print(warning)


for well_path in well_paths:
    print("Imported from folder: " + well_path.name)

well_paths = resInsight.project.import_well_paths(
    well_path_files=[
        "D:/Projects/ResInsight-regression-test/ModelData/Norne_WellPaths/E-3H.json",
        "D:/Projects/ResInsight-regression-test/ModelData/Norne_WellPaths/C-1H.json",
    ]
)
if resInsight.project.has_warnings():
    for warning in resInsight.project.warnings():
        print(warning)


for well_path in well_paths:
    print("Imported from individual files: " + well_path.name)


well_path_names = resInsight.project.import_well_log_files(
    well_log_folder="D:/Projects/ResInsight-regression-test/ModelData/Norne_PLT_LAS"
)
if resInsight.project.has_warnings():
    for warning in resInsight.project.warnings():
        print(warning)
```

(continues on next page)

(continued from previous page)

```
for well_path_name in well_path_names:
    print("Imported well log file for: " + well_path_name)
```

2.4.22 Input Prop Test Async

```
################################################################################
↪##
# This example generates a derived property in an asynchronous manner
# Meaning it does not wait for all the data for each stage to be read before␣
↪proceeding
################################################################################
↪##
import rips
import time

# Internal function for creating a result from a small chunk of poro and permx results
# The return value of the function is a generator for the results rather than the␣
↪result itself.
def create_result(poro_chunks, permx_chunks):
    # Loop through all the chunks of poro and permx in order
    for (poroChunk, permxChunk) in zip(poro_chunks, permx_chunks):
        resultChunk = []
        # Loop through all the values inside the chunks, in order
        for (poro, permx) in zip(poroChunk.values, permxChunk.values):
            resultChunk.append(poro * permx)
        # Return a generator object that behaves like a Python iterator
        yield resultChunk


resinsight = rips.Instance.find()
start = time.time()
case = resinsight.project.cases()[0]

# Get a generator for the poro results. The generator will provide a chunk each time␣
↪it is iterated
poro_chunks = case.active_cell_property_async("STATIC_NATIVE", "PORO", 0)
# Get a generator for the permx results. The generator will provide a chunk each time␣
↪it is iterated
permx_chunks = case.active_cell_property_async("STATIC_NATIVE", "PERMX", 0)

# Send back the result with the result provided by a generator object.
# Iterating the result generator will cause the script to read from the poro and␣
↪permx generators
# And return the result of each iteration
case.set_active_cell_property_async(
    create_result(poro_chunks, permx_chunks), "GENERATED", "POROPERMXAS", 0
)

end = time.time()
print("Time elapsed: ", end - start)
print("Transferred all results back")
view = case.views()[0].apply_cell_result("GENERATED", "POROPERMXAS")
```

2.4.23 Input Prop Test Sync

```
################################################################################
↪##
# This example generates a derived property in an synchronous manner
# Meaning it completes reading each result before calculating the derived result
# See InputPropTestAsync for how to do this asynchronously instead.
################################################################################
↪##
import rips
import time
import grpc

resinsight = rips.Instance.find()
start = time.time()
case = resinsight.project.cases()[0]

# Read poro result into list
poro_results = case.active_cell_property("STATIC_NATIVE", "PORO", 0)
# Read permx result into list
permx_results = case.active_cell_property("STATIC_NATIVE", "PERMX", 0)

# Generate output result
results = []
for (poro, permx) in zip(poro_results, permx_results):
    results.append(poro * permx)

try:
    # Send back output result
    case.set_active_cell_property(results, "GENERATED", "POROPERMXSY", 0)
except grpc.RpcError as e:
    print("Exception Received: ", e)


end = time.time()
print("Time elapsed: ", end - start)
print("Transferred all results back")

view = case.views()[0].apply_cell_result("GENERATED", "POROPERMXSY")
```

2.4.24 Instance Example

```
#####################################
# This example connects to ResInsight
#####################################
import rips

resinsight = rips.Instance.find()

if resinsight is None:
    print("ERROR: could not find ResInsight")
else:
    print("Successfully connected to ResInsight")
```

2.4.25 Launch Load Case Snapshot Exit

```
# Access to environment variables
import os

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.launch()

# This requires the TestModels to be installed with ResInsight (RESINSIGHT_BUNDLE_
↪TESTMODELS):
resinsight_exe_path = os.environ.get("RESINSIGHT_EXECUTABLE")

# Get the TestModels path from the executable path
resinsight_install_path = os.path.dirname(resinsight_exe_path)
test_models_path = os.path.join(resinsight_install_path, "TestModels")
path_name = os.path.join(
    test_models_path, "TEST10K_FLT_LGR_NNC/TEST10K_FLT_LGR_NNC.EGRID"
)

# Load an example case. Needs to be replaced with a valid path!
case = resinsight.project.load_case(path_name)

# Get a view
view1 = case.views()[0]

# Set the time step for view1 only
view1.set_time_step(time_step=2)

# Set cell result to SOIL
view1.apply_cell_result(result_type="DYNAMIC_NATIVE", result_variable="SOIL")

# Set export folder for snapshots and properties
resinsight.set_export_folder(export_type="SNAPSHOTS", path="e:/temp")
resinsight.set_export_folder(export_type="PROPERTIES", path="e:/temp")

# Export all snapshots
resinsight.project.export_snapshots()

# Export properties in the view
view1.export_property()

resinsight.exit()
```

2.4.26 Launch With Commandline Options

```
# Load ResInsight Processing Server Client Library
import rips

# Launch ResInsight with last project file and a Window size of 600x1000 pixels
resinsight = rips.Instance.launch(
    command_line_parameters=["--last", "--size", 600, 1000]
)
```

(continues on next page)

(continued from previous page)

```
# Get a list of all cases
cases = resinsight.project.cases()

print("Got " + str(len(cases)) + " cases: ")
for case in cases:
    print("Case name: " + case.name)
    print("Case grid path: " + case.file_path)
```

2.4.27 Load Case

```
# Access to environment variables and path tools
import os

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()

# This requires the TestModels to be installed with ResInsight (RESINSIGHT_BUNDLE_
↪TESTMODELS):
resinsight_exe_path = os.environ.get("RESINSIGHT_EXECUTABLE")

# Get the TestModels path from the executable path
resinsight_install_path = os.path.dirname(resinsight_exe_path)
test_models_path = os.path.join(resinsight_install_path, "TestModels")
path_name = os.path.join(
    test_models_path, "TEST10K_FLT_LGR_NNC/TEST10K_FLT_LGR_NNC.EGRID"
)
case = resinsight.project.load_case(path_name)
case.create_view()

# Print out lots of information from the case object
print("Case id: " + str(case.id))
print("Case name: " + case.name)
print("Case type: " + case.__class__.__name__)
print("Case file name: " + case.file_path)
print("Case reservoir bounding box:", case.reservoir_boundingbox())

timesteps = case.time_steps()
for t in timesteps:
    print("Year: " + str(t.year))
    print("Month: " + str(t.month))
```

2.4.28 Modeled Well Path

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()
```

(continues on next page)

(continued from previous page)

```
# Create a modeled well path and add well path targets
# The coordinates are based on the Norne case

well_path_coll = resinsight.project.descendants(rips.WellPathCollection)[0]
well_path = well_path_coll.add_new_object(rips.ModeledWellPath)
well_path.name = "Test Well-1"
well_path.update()

geometry = well_path.well_path_geometry()

reference_point = geometry.reference_point
reference_point[0] = 457196
reference_point[1] = 7322270
reference_point[2] = 2742
geometry.update()  # Commit updates back to ResInsight

# Create the first well target at the reference point
coord = [0, 0, 0]
geometry.append_well_target(coord)

# Append new well targets relative the the reference point
coord = [454.28, 250, -10]
target = geometry.append_well_target(coord)

coord = [1054.28, 250, -50]
target = geometry.append_well_target(coord)

well_path.append_perforation_interval(3300, 3350, 0.2, 0.76)
```

2.4.29 Modeled Well Path Lateral

```
# Load ResInsight Processing Server Client Library
import rips
import time

# Connect to ResInsight instance
resinsight = rips.Instance.find()

# Create a modeled well path and add well path targets
# The coordinates are based on the Norne case
# Add a lateral to the main well path

well_path_coll = resinsight.project.descendants(rips.WellPathCollection)[0]
well_path = well_path_coll.add_new_object(rips.ModeledWellPath)
well_path.name = "Test Well-1"
well_path.update()

geometry = well_path.well_path_geometry()

reference_point = geometry.reference_point
reference_point[0] = 457196
reference_point[1] = 7322270
reference_point[2] = 2742
geometry.update()  # Commit updates back to ResInsight
```

(continues on next page)

(continued from previous page)

```
# Create the first well target at the reference point
coord = [0, 0, 0]
geometry.append_well_target(coord)

# Append new well targets relative the the reference point
coord = [454.28, 250, -10]
target = geometry.append_well_target(coord)

coord = [1054.28, 250, -50]
target = geometry.append_well_target(coord)

# Create a lateral at specified location on parent well
measured_depth = 3600
lateral = well_path.append_lateral(measured_depth)
geometry = lateral.well_path_geometry()

coord = [770, 280, 50]
target = geometry.append_well_target(coord)

coord = [1054.28, -100, 50]
target = geometry.append_well_target(coord)

coord = [2054.28, -100, 45]
target = geometry.append_well_target(coord)


# Wait 2 second
print("Wait 2 seconds ...")
time.sleep(2)
print("Move reference point of parent well")

geometry = well_path.well_path_geometry()
reference_point = geometry.reference_point
reference_point[2] += 50
geometry.update()  # Commit updates back to ResInsight
```

2.4.30 New Summary Plot

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Example code
project = resinsight.project

summary_cases = project.descendants(rips.SummaryCase)
summary_plot_collection = project.descendants(rips.SummaryPlotCollection)[0]
if len(summary_cases) > 0:
    summary_plot = summary_plot_collection.new_summary_plot(
        summary_cases=summary_cases, address="FOP*"
    )
```

2.4.31 Open Project

```
# Access to environment variables and path tools
import os

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()

# This requires the TestModels to be installed with ResInsight (RESINSIGHT_BUNDLE_
↪TESTMODELS):
resinsight_exe_path = os.environ.get("RESINSIGHT_EXECUTABLE")

# Get the TestModels path from the executable path
resinsight_install_path = os.path.dirname(resinsight_exe_path)
test_models_path = os.path.join(resinsight_install_path, "TestModels")
path_name = os.path.join(test_models_path, "TEST10K_FLT_LGR_NNC/10KWithWellLog.rsp")

# Open a project
resinsight.project.open(path_name)
```

2.4.32 Replace Case

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Example code
print("ResInsight version: " + resinsight.version_string())

case = resinsight.project.case(case_id=0)
case.replace(
    new_grid_file="C:/Users/lindkvis/Projects/ResInsight/TestModels/Case_with_10_
↪timesteps/Real0/BRUGGE_0000.EGRID"
)
```

2.4.33 Save Project

```
# Access to environment variables and path tools
import os

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()

# This requires the TestModels to be installed with ResInsight (RESINSIGHT_BUNDLE_
↪TESTMODELS):
resinsight_exe_path = os.environ.get("RESINSIGHT_EXECUTABLE")
```

(continues on next page)

(continued from previous page)

```
# Get the TestModels path from the executable path
resinsight_install_path = os.path.dirname(resinsight_exe_path)
test_models_path = os.path.join(resinsight_install_path, "TestModels")
path_name = os.path.join(
    test_models_path, "TEST10K_FLT_LGR_NNC/TEST10K_FLT_LGR_NNC.EGRID"
)
case = resinsight.project.load_case(path_name)

# Save the project to file
home_dir = os.path.expanduser("~")
project_path = home_dir + "/new-project.rsp"
print("Saving project to: ", project_path)
resinsight.project.save(project_path)
```

2.4.34 Selected Cases

```
########################################################################
# This example returns the currently selected cases in ResInsight
# Because running this script in the GUI takes away the selection
# This script does not run successfully from within the ResInsight GUI
# And will need to be run from the command line separately from ResInsight
########################################################################

import rips

resinsight = rips.Instance.find()
if resinsight is not None:
    cases = resinsight.project.selected_cases()

    print("Got " + str(len(cases)) + " cases: ")
    for case in cases:
        print(case.name)
        for property in case.available_properties("DYNAMIC_NATIVE"):
            print(property)
```

2.4.35 Selected Cells

```
########################################################################
# This example prints center and corners for the currently selected cells
# in ResInsight
########################################################################

import rips

resinsight = rips.Instance.find()
if resinsight is not None:
    cases = resinsight.project.cases()

    print("Got " + str(len(cases)) + " cases: ")
    for case in cases:
        print(case.name)
```

(continues on next page)

(continued from previous page)

```
        cells = case.selected_cells()
        print("Found " + str(len(cells)) + " selected cells")

        time_step_info = case.time_steps()

        for (idx, cell) in enumerate(cells):
            print(
                "Selected cell: [{}, {}, {}] grid: {}".format(
                    cell.ijk.i + 1, cell.ijk.j + 1, cell.ijk.k + 1, cell.grid_index
                )
            )

            # Get the grid and dimensions
            grid = case.grids()[cell.grid_index]
            dimensions = grid.dimensions()

            # Map ijk to cell index
            cell_index = (
                dimensions.i * dimensions.j * cell.ijk.k
                + dimensions.i * cell.ijk.j
                + cell.ijk.i
            )

            # Print the cell center
            cell_centers = grid.cell_centers()
            cell_center = cell_centers[cell_index]
            print(
                "Cell center: [{}, {}, {}]".format(
                    cell_center.x, cell_center.y, cell_center.z
                )
            )

            # Print the cell corners
            cell_corners = grid.cell_corners()[cell_index]
            print("Cell corners:")
            print("c0:\n" + str(cell_corners.c0))
            print("c1:\n" + str(cell_corners.c1))
            print("c2:\n" + str(cell_corners.c2))
            print("c3:\n" + str(cell_corners.c3))
            print("c4:\n" + str(cell_corners.c4))
            print("c5:\n" + str(cell_corners.c5))
            print("c6:\n" + str(cell_corners.c6))
            print("c7:\n" + str(cell_corners.c7))

            for (tidx, timestep) in enumerate(time_step_info):
                # Read the full SOIL result for time step
                soil_results = case.selected_cell_property(
                    "DYNAMIC_NATIVE", "SOIL", tidx
                )
                print(
                    "SOIL: {} ({}.{}.{})".format(
                        soil_results[idx], timestep.year, timestep.month, timestep.day
                    )
                )
```

2.4.36 Set Cell Result

```
########################################################################
# This script applies a cell result to the first view in the project
########################################################################
import rips

resinsight = rips.Instance.find()

view = resinsight.project.views()[0]
view.apply_cell_result(result_type="STATIC_NATIVE", result_variable="DX")
```

2.4.37 Set Flow Diagnostics Result

```
########################################################################
# This script applies a flow diagnostics cell result to the first view in the project
########################################################################

# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()

view = resinsight.project.view(view_id=1)
# view.apply_flow_diagnostics_cell_result(result_variable='Fraction',
#                                         selection_mode='FLOW_TR_INJ_AND_PROD')

# Example of setting individual wells. Commented out because well names are case␣
↪specific.
view.apply_flow_diagnostics_cell_result(
    result_variable="Fraction",
    selection_mode="FLOW_TR_BY_SELECTION",
    injectors=["C-1H", "C-2H", "F-2H"],
    producers=["B-1AH", "B-3H", "D-1H"],
)
```

2.4.38 Set Grid Properties

```
########################################################################
# This script sets values for SOIL for all grid cells in the first case in the project
########################################################################
import rips

resinsight = rips.Instance.find()

case = resinsight.project.case(case_id=0)
total_cell_count = case.cell_count().reservoir_cell_count

values = []
for i in range(0, total_cell_count):
    values.append(i % 2 * 0.75)
```

(continues on next page)

(continued from previous page)

```
print("Applying values to full grid")
case.set_grid_property(values, "DYNAMIC_NATIVE", "SOIL", 0)
```

2.4.39 Soil Average Async

```
################################################################################
#####
# This example will asynchronously calculate the average value for SOIL for all time␣
steps
################################################################################
#####

import rips
import itertools
import time

resinsight = rips.Instance.find()

start = time.time()

# Get the case with case id 0
case = resinsight.project.case(case_id=0)

# Get a list of all time steps
timeSteps = case.time_steps()

averages = []
for i in range(0, len(timeSteps)):
    # Get the results from time step i asynchronously
    # It actually returns a generator object almost immediately
    result_chunks = case.active_cell_property_async("DYNAMIC_NATIVE", "SOIL", i)
    mysum = 0.0
    count = 0
    # Loop through and append the average. each time we loop resultChunks
    # We will trigger a read of the input data, meaning the script will start
    # Calculating averages before the whole resultValue for this time step has been␣
received
    for chunk in result_chunks:
        mysum += sum(chunk.values)
        count += len(chunk.values)

    averages.append(mysum / count)

end = time.time()
print("Time elapsed: ", end - start)
print(averages)
```

2.4.40 Soil Average Sync

```
################################################################################
#####
# This example will synchronously calculate the average value for SOIL for all time␣
steps
```

(continues on next page)

(continued from previous page)

```
############################################################################################
↪#####
import rips
import itertools
import time

resinsight = rips.Instance.find()

start = time.time()

# Get the case with case id 0
case = resinsight.project.case(case_id=0)

# Get a list of all time steps
time_steps = case.time_steps()

averages = []
for i in range(0, len(time_steps)):
    # Get a list of all the results for time step i
    results = case.active_cell_property("DYNAMIC_NATIVE", "SOIL", i)
    mysum = sum(results)
    averages.append(mysum / len(results))

end = time.time()
print("Time elapsed: ", end - start)
print(averages)
```

2.4.41 Soil Porv Async

```
############################################################################
# This example will create a derived result for each time step asynchronously
############################################################################

import rips
import time

# Internal function for creating a result from a small chunk of soil and porv results
# The return value of the function is a generator for the results rather than the␣
↪result itself.
def create_result(soil_chunks, porv_chunks):
    for (soil_chunk, porv_chunk) in zip(soil_chunks, porv_chunks):
        resultChunk = []
        number = 0
        for (soil_value, porv_value) in zip(soil_chunk.values, porv_chunk.values):
            resultChunk.append(soil_value * porv_value)
        # Return a Python generator
        yield resultChunk


resinsight = rips.Instance.find()
start = time.time()
case = resinsight.project.cases()[0]
timeStepInfo = case.time_steps()

# Get a generator for the porv results. The generator will provide a chunk each time␣
↪it is iterated
```

(continues on next page)

(continued from previous page)

```
porv_chunks = case.active_cell_property_async("STATIC_NATIVE", "PORV", 0)

# Read the static result into an array, so we don't have to transfer it for each␣
↪iteration
# Note we use the async method even if we synchronise here, because we need the␣
↪values chunked
# ... to match the soil chunks
porv_array = []
for porv_chunk in porv_chunks:
    porv_array.append(porv_chunk)

for i in range(0, len(timeStepInfo)):
    # Get a generator object for the SOIL property for time step i
    soil_chunks = case.active_cell_property_async("DYNAMIC_NATIVE", "SOIL", i)
    # Create the generator object for the SOIL * PORV derived result
    result_generator = create_result(soil_chunks, iter(porv_array))
    # Send back the result asynchronously with a generator object
    case.set_active_cell_property_async(
        result_generator, "GENERATED", "SOILPORVAsync", i
    )

end = time.time()
print("Time elapsed: ", end - start)

print("Transferred all results back")

view = case.views()[0].apply_cell_result("GENERATED", "SOILPORVAsync")
```

2.4.42 Soil Porv Sync

```
################################################################################
# This example will create a derived result for each time step synchronously
################################################################################

import rips
import time

resinsight = rips.Instance.find()
start = time.time()
case = resinsight.project.cases()[0]

# Read the full porv result
porv_results = case.active_cell_property("STATIC_NATIVE", "PORV", 0)
time_step_info = case.time_steps()

for i in range(0, len(time_step_info)):
    # Read the full SOIl result for time step i
    soil_results = case.active_cell_property("DYNAMIC_NATIVE", "SOIL", i)

    # Generate the result by looping through both lists in order
    results = []
    for (soil, porv) in zip(soil_results, porv_results):
        results.append(soil * porv)

    # Send back result
```

(continues on next page)

(continued from previous page)

```
    case.set_active_cell_property(results, "GENERATED", "SOILPORVSync", i)

end = time.time()
print("Time elapsed: ", end - start)

print("Transferred all results back")

view = case.views()[0].apply_cell_result("GENERATED", "SOILPORVSync")
```

2.4.43 Summary Cases

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()
# Example code

# Specific summary case with case_id = 1
summary_case = resinsight.project.summary_case(case_id=1)
summary_case.print_object_info()

# All summary cases
summary_cases = resinsight.project.summary_cases()
for summary_case in summary_cases:
    print("Summary case found: ", summary_case.short_name)
```

2.4.44 Summary Vectors

```
import rips
import time

resinsight = rips.Instance.find()

project = resinsight.project

# Use the following commented lines to import a file from disk
# filename = "path/to/file/1_R001_REEK-0.SMSPEC"
# summary_case = project.import_summary_case(filename)

# Assumes at least one summery case loaded with case_id 1
summary_case = project.summary_case(1)
if summary_case is None:
    print("No summary case found")
    exit()

vector_name = "FOPT"
summary_data = summary_case.summary_vector_values(vector_name)

print("Data for summary vector " + vector_name)
print(summary_data.values)
```

(continues on next page)

(continued from previous page)

```
time_steps = summary_case.available_time_steps()
print(time_steps.values)

summary_data_sampled = summary_case.resample_values("FOPT", "QUARTER")
print("\nResampled data")

for t, value in zip(summary_data_sampled.time_steps, summary_data_sampled.values):
    print(time.strftime("%a, %d %b %Y ", time.gmtime(t)) + " | " + str(value))
```

2.4.45 Surface Import

```
# Load ResInsight Processing Server Client Library
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()
print("ResInsight version: " + resinsight.version_string())

# Example code

# get the project
project = resinsight.project

# get the topmost surface folder from the project
surfacefolder = project.surface_folder()

# list of surface files to load
filenames = ["surface1.ts", "surface2.ts", "surface3.ts"]

# Load the files into the top level
for surffile in filenames:
    surface = surfacefolder.import_surface(surffile)
    if surface is None:
        print("Could not import the surface " + surffile)

# add a subfolder
subfolder = surfacefolder.add_folder("ExampleFolder")

# load the same surface multiple times using increasing depth offsets
# store them in the new subfolder we just created
for offset in range(0, 200, 20):
    surface = subfolder.import_surface("mysurface.ts")
    if surface:
        surface.depth_offset = offset
        surface.update()
    else:
        print("Could not import surface.")

# get an existing subfolder
existingfolder = project.surface_folder("ExistingFolder")
if existingfolder is None:
    print("Could not find the specified folder.")
```

2.4.46 View Example

```
############################################################
# This example will alter the views of all cases
# By setting the background color and toggle the grid box
# Also clones the first view
############################################################
import rips

# Connect to ResInsight instance
resinsight = rips.Instance.find()

# Check if connection worked
if resinsight is not None:
    # Get a list of all cases
    cases = resinsight.project.cases()
    for case in cases:
        # Get a list of all views
        views = case.views()
        for view in views:
            # Set some parameters for the view
            view.show_grid_box = not view.show_grid_box
            view.background_color = "#3388AA"
            # Update the view in ResInsight
            view.update()
        # Clone the first view
        new_view = views[0].clone()
        new_view.background_color = "#FFAA33"
        new_view.update()
        view.show_grid_box = False
        view.set_visible(False)
        view.update()
```

2.5 Command Interface Module

As the Python interface is growing release by release, we are investigating how to automate the building of reference documentation. This document is not complete, but will improve as the automation moves forward.

More details on the command file operations, see https://resinsight.org/scripting/commandfile/

2.5.1 clone_view

Parameter	Type	Description
view_id	int	View Id

2.5.2 close_project

2.5.3 compute_case_group_statistics

Parameter	Type	Description
case_group_id	int	Case Group ID
case_ids	int	Case IDs

2.5.4 create_grid_case_group

Parameter	Type	Description
case_paths	str	List of Paths to Case Files

2.5.5 create_lgr_for_completions

Parameter	Type	Description
case_id	int	Case ID
time_step	int	Time Step Index
well_path_names	str	Well Path Names
refinement_i	int	RefinementI
refinement_j	int	RefinementJ
refinement_k	int	RefinementK
split_type	str	SplitType

2.5.6 create_multi_plot

Parameter	Type	Description
plots	str	Plots

2.5.7 create_multiple_fractures

Parameter	Type	Description
case_id	int	Case ID
well_path_names	str	Well Path Names
min_dist_from_well_td	float	Min Distance From Well TD
max_fractures_per_well	int	Max Fractures per Well
template_id	int	Template ID
top_layer	int	Top Layer
base_layer	int	Base Layer
spacing	float	Spacing
action	str	Action

2.5.8 create_saturation_pressure_plots

Parameter	Type	Description
case_ids	int	Case IDs

2.5.9 create_statistics_case

Parameter	Type	Description
case_group_id	int	Case Group Id

2.5.10 create_view

Parameter	Type	Description
case_id	int	Case Id

2.5.11 create_well_bore_stability_plot

Parameter	Type	Description
case_id	int	GeoMech Case Id
well_path	str	Well Path
time_step	int	Time Step
wbs_parameters	str	WbsParameters

2.5.12 export_contour_map_to_text

Parameter	Type	Description
export_file_name	str	
export_local_coordinates	str	
undefined_value_label	str	
exclude_undefined_values	str	
view_id	int	View Id

2.5.13 export_flow_characteristics

Parameter	Type	Description
case_id	int	Case ID
time_steps	int	Selected Time Steps
injectors	str	Injectors
producers	str	Producers
file_name	str	Export File Name
minimum_communication	float	Minimum Communication
aquifer_cell_threshold	float	Aquifer Cell Threshold

2.5.14 export_lgr_for_completions

Parameter	Type	Description
case_id	int	Case ID
time_step	int	Time Step Index
well_path_names	str	Well Path Names
refinement_i	int	RefinementI
refinement_j	int	RefinementJ
refinement_k	int	RefinementK
split_type	str	SplitType

2.5.15 export_msw

Parameter	Type	Description
case_id	int	Case ID
well_path	str	Well Path Name
include_perforations	str	Include Perforations
include_fishbones	str	Include Fishbones
include_fractures	str	Include Fractures
file_split	str	File Split

2.5.16 export_multi_case_snapshots

Parameter	Type	Description
grid_list_file	str	Grid List File

2.5.17 export_property

Parameter	Type	Description
case_id	int	Case ID
time_step	int	Time Step Index
property	str	Property Name
eclipse_keyword	str	Eclipse Keyword
undefined_value	float	Undefined Value
export_file	str	Export FileName

2.5.18 export_property_in_views

Parameter	Type	Description
case_id	int	Case ID
view_ids	int	View IDs
view_names	str	View Names
undefined_value	float	Undefined Value

2.5.19 export_sim_well_fracture_completions

Parameter	Type	Description
case_id	int	Case ID
view_id	int	View ID
view_name	str	View Name
time_step	int	Time Step Index
simulation_well_names	str	Simulation Well Names
file_split	str	File Split
compdat_export	str	Compdat Export

2.5.20 export_snapshots

Parameter	Type	Description
type	str	Type
prefix	str	Prefix
case_id	int	Case Id
view_id	int	View Id
export_folder	str	Export Folder
plot_output_format	str	Output Format

2.5.21 export_visible_cells

Parameter	Type	Description
case_id	int	Case ID
view_id	int	View ID
view_name	str	View Name
export_keyword	str	Export Keyword
visible_active_cells_value	int	Visible Active Cells Value
hidden_active_cells_value	int	Hidden Active Cells Value
inactive_cells_value	int	Inactive Cells Value

2.5.22 export_well_log_plot_data

Parameter	Type	Description
export_format	str	
view_id	int	
export_folder	str	
file_prefix	str	
export_tvd_rkb	str	
capitalize_file_names	str	
resample_interval	float	
convert_curve_units	str	

2.5.23 export_well_path_completions

Parameter	Type	Description
case_id	int	Case ID
time_step	int	Time Step Index
well_path_names	str	Well Path Names
file_split	str	File Split
compdat_export	str	Compdat Export
combination_mode	str	Combination Mode
include_msw	str	Export Multi Segment Well Model
use_ntg_horizontally	str	Use NTG Horizontally
include_perforations	str	Include Perforations
include_fishbones	str	Include Fishbones
include_fractures	str	Include Fractures
exclude_main_bore_for_fishbones	str	Exclude Main Bore for Fishbones
perform_trans_scaling	str	Perform Transmissibility Scaling
trans_scaling_time_step	int	Transmissibility Scaling Pressure Time Step
trans_scaling_wbhp_from_summary	str	Transmissibility Scaling WBHP from summary
trans_scaling_wbhp	float	Transmissibility Scaling Constant WBHP Value
export_comments	str	Export Data Source as Comments
export_welspec	str	Export WELSPEC keyword
custom_file_name	str	Custom Filename

2.5.24 export_well_paths

Parameter	Type	Description
well_path_names	str	Well Path Names
md_step_size	float	MD Step Size

2.5.25 import_formation_names

Parameter	Type	Description
formation_files	str	
apply_to_case_id	int	

2.5.26 import_well_log_files

Parameter	Type	Description
well_log_folder	str	
well_log_files	str	

2.5.27 import_well_paths

Parameter	Type	Description
well_path_folder	str	
well_path_files	str	
import_grouped	str	

2.5.28 load_case

Parameter	Type	Description
path	str	Path to Case File
grid_only	str	Load Grid Data Only

2.5.29 open_project

Parameter	Type	Description
path	str	Path

2.5.30 replace_case

Parameter	Type	Description
case_id	int	Case ID
new_grid_file	str	New Grid File

2.5.31 replace_multiple_cases

2.5.32 replace_source_cases

Parameter	Type	Description
case_group_id	int	Case Group ID
grid_list_file	str	Grid List File

2.5.33 run_octave_script

Parameter	Type	Description
path	str	Path
case_ids	int	Case IDs

2.5.34 save_project

Parameter	Type	Description
file_path	str	

2.5.35 save_project_as

Parameter	Type	Description
file_path	str	

2.5.36 scale_fracture_template

Parameter	Type	Description
id	int	Id
half_length	float	HalfLengthScaleFactor
height	float	HeightScaleFactor
d_factor	float	DFactorScaleFactor
conductivity	float	ConductivityScaleFactor
width	float	WidthScaleFactor

2.5.37 set_export_folder

Parameter	Type	Description
type	str	Type
path	str	Path
create_folder	str	Create Folder

2.5.38 set_fracture_containment

Parameter	Type	Description
id	int	Id
top_layer	int	TopLayer
base_layer	int	BaseLayer

2.5.39 set_main_window_size

Parameter	Type	Description
height	int	Height
width	int	Width

2.5.40 set_plot_window_size

Parameter	Type	Description
height	int	Height
width	int	Width

2.5.41 set_start_dir

Parameter	Type	Description
path	str	Path

2.5.42 set_time_step

Parameter	Type	Description
case_id	int	Case ID
view_id	int	View ID
time_step	int	Time Step Index

2.5.43 stack_curves

Parameter	Type	Description
curves	str	

2.5.44 unstack_curves

Parameter	Type	Description
curves	str	

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W