SolTrace API

Generated by Doxygen 1.9.3

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 pysoltrace.PySolTrace.Stage.Element Class Reference

Public Member Functions

- def **__init__** (self, parent_stage, int element_id)
- int Create (self)
- def surface_spherical (self, radius)
- def surface_parabolic (self, focal_len_x, focal_len_y)
- def surface_flat (self)
- def surface_hypellip (self, vertex_curv, kappa)
- def surface_conical (self, theta)
- def surface_cylindrical (self, radius)
- def surface_toroid (self, rad_annulus, rad_ring)
- def surface_zernicke (self, file_path)
- def surface_polynomialrev (self, file_path)
- def surface_cubicspline (self, file_path)
- def surface_finiteelement (self, file_path)
- def surface_vshot (self, file_path)
- def aperture_circle (self, diameter)
- def aperture_hexagon (self, diameter)
- def aperture_triangle (self, diameter)
- def aperture_rectangle (self, length_x, length_y)
- def aperture_annulus (self, r_inner, r_outer, theta)
- def aperture_singleax_curve (self, x1, x2, L)
- def aperture_irr_triangle (self, x1, y1, x2, y2, x3, y3)
- def aperture_quadrilateral (self, x1, y1, x2, y2, x3, y3, x4, y4)

Public Attributes

- **stage_id**

  Identifying integer associated with the containing stage.
- **id**

  Identifying integer associated with element.
- **enabled**

  Flag indicating whether the element is included in the model.
- **position**

  Element location in stage coordinates.
- **aim**

  Element coordinate system aim point in stage coordinates.
- **zrot**

  [deg] Rotation of coordinate system around z-axis
- **aperture**

  Charater indicating aperture type.
- **aperture_params**

  Up to 8 coefficients defining aperture – values depend on selection for 'aperture'.
- **surface**

  Character indicating surface type.
- **surface_params**

  Up to 8 coefficients defining surface – values depend on selection for 'surface'.
- **surface_file**

  Name for surface file, if using compatible type.
- interaction

  Flag indicating optical interaction type.
- **optic**

  Reference to Optics instance associated with this element.

2.1.1 Detailed Description

```
 *Element* is a subclass of PySolTrace.Stage, and represents a set of properties and
geometric settings related to a single optical element in SolTrace. Elements are associ-
ated with a single stage, and are stored in the respective stage's Stage.elements[] list.

Attributes
----------
p_dll
    Reference for API DLL, managed by PySolTrace
p_data
    Memory location for soltrace context, managed by PySolTrace
stage_id : int
    Identifying integer associated with the containing stage
id : int
    Identifying integer associated with element
enabled : bool
    Flag indicating whether the element is included in the model
position : Point
    Element location in stage coordinates
aim : Point
    Element coordinate system aim point in stage coordinates
zrot : float
    [deg] Rotation of coordinate system around z-axis
```

```
interaction : int
    Flag indicating optical interaction type. {1:refraction, 2:reflection}
optic : Optics
    Reference to *Optics* instance associated with this element
aperture : char
    Charater indicating aperture type. One of:
    {'c':circle, 'h':hexagon, 't':triangle, 'r':rectangle, 'a':annulus,
    'l':single-axis curvature, 'i':irregular triangle, 'q':quadrilateral}
aperture_params : [float,]
    Up to 8 coefficients defining aperture -- values depend on selection for 'aperture'
surface : char
    Character indicating surface type. One of:
    {'s':spherical, 'p':parabolic, 'f':flat plane, 'o':hyperboloid/ellipsoid,
    'c':conical, 't':cylindrical, 'd':toroid, 'm':Zernicke monomial,
    'r':Polynomial revolution, 'i':cubic spline interpolation,
    'e':finite element data, 'v':VSHOT data}
surface_params : [float,]
    Up to 8 coefficients defining surface -- values depend on selection for 'surface'
surface_file : string
    Name for surface file, if using compatible type. File extension:
    *.mon --> 'm' / Zernicke monomial
    *.sht --> 'v' / VSHOT data
    *.ply --> 'r' / Polynomial revolution
    *.csi --> 'i' / Cubic spline interpolation
    *.fed --> 'e' / Finite element data

Methods
----------
Create
    Calls methods to instantiate and construct element in the SolTrace context
surface_xxxx
    Family of methods that compute surface coefficients
```

2.1.2 Member Function Documentation

2.1.2.1 aperture_annulus()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_annulus (
              self,
              r_inner,
              r_outer,
              theta )
```

```
Set up the aperture as annular, where aperture is the annulus between to specified radii
and within an angular slice 'theta' which is centered around the x-axis.

Parameters
----------
r_inner
    Inner radius of annular region
r_outer
    Outer radius of annular region
theta : deg
    Slice of the circle contained, centered around x-axis
```

2.1.2.2 aperture_circle()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_circle (
              self,
              diameter )

Set up the aperture as circular with 'diameter'.

Parameters
----------
diameter
    Diameter of the circle
```

2.1.2.3 aperture_hexagon()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_hexagon (
              self,
              diameter )

Set up the aperture as a hexagon centered at x=0,y=0. The hexagon is circumscribed
by a circle of 'diameter'.

Parameters
----------
diameter
    Diameter of the circumscribing circle.
```

2.1.2.4 aperture_irr_triangle()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_irr_triangle (
              self,
              x1,
              y1,
              x2,
              y2,
              x3,
              y3 )

Set up the aperture as a triangle given by three (x,y) coordinate pairs.

Parameters
----------
x1
    x-coordinate, point 1
y1
    y-coordinate, point 1
x2
    x-coordinate, point 2
y2
    y-coordinate, point 2
x3
    x-coordinate, point 3
y3
    y-coordinate, point 3
```

2.1.2.5 aperture_quadrilateral()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_quadrilateral (
              self,
              x1,
              y1,
              x2,
              y2,
              x3,
              y3,
              x4,
              y4 )
```

Set up the aperture as a quadrilateral given by four (x,y) coordinate pairs.

```
Parameters
----------
x1
    x-coordinate, point 1
y1
    y-coordinate, point 1
x2
    x-coordinate, point 2
y2
    y-coordinate, point 2
x3
    x-coordinate, point 3
y3
    y-coordinate, point 3
x4
    x-coordinate, point 4
y4
    y-coordinate, point 4
```

2.1.2.6 aperture_rectangle()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_rectangle (
              self,
              length_x,
              length_y )
```

Set up the aperture as a rectangle with the centroid at x=0,y=0.

```
Parameters
----------
length_x
    Width in x-coordinate direction
length_y
    Height in y-coordinate direction
```

2.1.2.7 aperture_singleax_curve()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_singleax_curve (
              self,
              x1,
              x2,
              L )
```

Set up the aperture as revolved around a single axis. Revolved window is between two coordinates x1->x1, both non-negative and with x2 > x1. The aperture has length 'L' in the y-direction.

This aperture is often used with a cylindrical surface. In this case, both x1 and x2 should be zero, and the cylinder height specified with 'L'.

```
Parameters
----------
x1
    inner coordinate of revolved section
x2
    outer coordinate of revolved section
L
    length of revolved section along axis of revolution
```

2.1.2.8 aperture_triangle()

```
def pysoltrace.PySolTrace.Stage.Element.aperture_triangle (
              self,
              diameter )
```

Set up the aperture as a equilateral triangle with centroid at x=0,y=0. The triangle is circumscribed by a circle of 'diameter'.

```
Parameters
----------
diameter
    Diameter of the circumscribing circle.
```

2.1.2.9 Create()

```
int pysoltrace.PySolTrace.Stage.Element.Create (
              self )
```

Create Element instance in the SolTrace context.

```
Returns
----------
int
    1 if successful, 0 otherwise
```

2.1.2.10 surface_conical()

```
def pysoltrace.PySolTrace.Stage.Element.surface_conical (
              self,
              theta )
```

Set up the surface described by cone with half-angle theta.

```
Parameters
----------
theta : deg
    half-angle of cone
```

2.1.2.11 surface_cubicspline()

```
def pysoltrace.PySolTrace.Stage.Element.surface_cubicspline (
              self,
              file_path )
```

Set up the surface from a file as a rotationally symmetric cubic spline. Accepts *csi file extension.
File format should be two tab-separated columns:

```
    N
    r1      Z1
    r2      Z2
    r3      Z3
    ...
    rN      ZN
    dZ/dr1  dZ/drN

Parameters
----------
file_path
    Path to the file containing the data.
```

2.1.2.12 surface_cylindrical()

```
def pysoltrace.PySolTrace.Stage.Element.surface_cylindrical (
              self,
              radius )
```

Set up the surface as cylindrical.

```
Parameters
----------
radius
    Radius of the cylinder
```

2.1.2.13 surface_finiteelement()

```
def pysoltrace.PySolTrace.Stage.Element.surface_finiteelement (
              self,
              file_path )
```

```
Set up the surface from a file using finite element data specifying the vertices of the elements in
x,y,z coordinates.

Accepts the *.fed file extension. File format should be 3 tab-separated
columns:
    N
    x1      y1      z1
    x2      y2      z2
    x3      y3      z3
    ...
    xN      yN      zN

Parameters
----------
file_path
    Path to the file containing the data.
```

2.1.2.14 surface_flat()

```
def pysoltrace.PySolTrace.Stage.Element.surface_flat (
              self )
```

```
Set up the surface as flat
```

2.1.2.15 surface_hypellip()

```
def pysoltrace.PySolTrace.Stage.Element.surface_hypellip (
              self,
              vertex_curv,
              kappa )
```

```
Set up the surface described by equation:
    Z(x,y) = ( vertex_curv*(x^2 + y^2) ) /
               (1 + sqrt(1-kappa*vertex_curv^2*(x^2 + y^2)))
Parameters
----------
vertex_curv
    Curvature parameter
kappa
    Form parameter. Value of parameter determines geometry as follows:
    kappa < 0 --> tall hyperboloid
    kappa 0..1 --> ellipsoid
    kappa > 1 --> stout ellipsoid
```

2.1.2.16 surface_parabolic()

```
def pysoltrace.PySolTrace.Stage.Element.surface_parabolic (
              self,
              focal_len_x,
              focal_len_y )
```

```
Set up the surface as parabolic

Parameters
==========
focal_len_x : float
    Focal length of the surface in the x-direction. If infinite, use float('inf')
focal_len_y : float
    Focal length of the surface in the y-direction. If infinite, use float('inf')
```

2.1.2.17 surface_polynomialrev()

```
def pysoltrace.PySolTrace.Stage.Element.surface_polynomialrev (
              self,
              file_path )
```

```
Set up the surface from a file as a rotationally symmetric polynomial, where the surface is described by
the equation:
Z(r) = sum_i=0^N  C_i * r^i,  where r=sqrt(x^2 + y^2)

Accepts *ply file extension specifying equation coefficients.
File format should be a single data column:
    N
    C0
    C1
    C2
    ...
    C,N

Parameters
----------
file_path
    Path to the file containing the data.
```

2.1.2.18 surface_spherical()

```
def pysoltrace.PySolTrace.Stage.Element.surface_spherical (
              self,
              radius )
```

```
Set up the surface as spherical type

Parameters
==========
radius : float
    Radius of the spherical surface
```

2.1.2.19 surface_toroid()

```
def pysoltrace.PySolTrace.Stage.Element.surface_toroid (
              self,
              rad_annulus,
              rad_ring )
```

```
Set up the surface as a toroid "donut".

Parameters
----------
rad_annulus
    Radius of the 'tube', the distance between the min and max radii of the torus
rad_ring
    The radius of the centerpoint of the annular tube
```

2.1.2.20 surface_vshot()

```
def pysoltrace.PySolTrace.Stage.Element.surface_vshot (
              self,
              file_path )
```

```
Set up the surface from a file using VSHOT data specifying matrix coefficients generated by a VSHOT test.

Accepts the *.sht file extension. File format should be:
    First line - file name (skipped)
    Radius      Focal length        Target-dist
    0           order               num points
    rmsslope    rmsscale
    b00
    b10
    b11
    b20
    b21
    b22
    ...
    bDD    || where 'D' is order
    a1      b1      c1      d1      e1
    a2      b2      c2      d2      e2
    a3      b3      c3      d3      e3
    ...
    aN      bN      cN      dN      eN  || where 'N' is num points

Parameters
----------
file_path
    Path to the file containing the data.
```

2.1.2.21 surface_zernicke()

```
def pysoltrace.PySolTrace.Stage.Element.surface_zernicke (
              self,
              file_path )
```

```
Set up the surface from a file as a Zernicke surface, where the surface is described by the equation:
Z(x,y) = sum_i=0^N
            sum_j=0^i  Bi,j * x^j * y^(i-j)

Accepts *mon file extension specifying the Zernicke coefficients.
File format should be a single data column:
    N
    B0,0
    B1,0
    B1,1
    B2,1
    B2,2
    B2,3
    ...
    BN,N

Parameters
----------
file_path
    Path to the file containing the data.
```

2.1.3 Member Data Documentation

2.1.3.1 interaction

```
pysoltrace.PySolTrace.Stage.Element.interaction
```

Flag indicating optical interaction type.

{1:refraction, 2:reflection}

The documentation for this class was generated from the following file:

- pysoltrace.py

2.2 pysoltrace.PySolTrace.Optics.Face Class Reference

Public Member Functions

- def **__init__** (self)

Public Attributes

- dist_type

  Distribution type for surface interactions.
- **refraction_real**

  Real component of the refraction index.
- **reflectivity**

  [0..1] Surface reflectivity
- **transmissivity**

  [0..1] Surface transmissivity
- **slope_error**

  [mrad] Surface RMS slope error, half-angle
- **spec_error**

  [mrad] Surface specularity error, half-angle
- **usereflтable**

  Flag specifying use of user reflectivity table to modify reflectivity as a function of incidence angle.
- refltable

  [mrad,0..1] 2D list containing pairs of [angle,reflectivity] values.

2.2.1 Detailed Description

```
Subclass of Optics, contains properties associated with one of the optical faces.

Attributes
----------
dist_type : char
    Distribution type for surface interactions. One of:
    {'g':Gaussian, 'p':Pillbox, 'd':Diffuse }
refraction_real : float
    Real component of the refraction index
reflectivity : float
    [0..1] Surface reflectivity
transmissivity : float
    [0..1] Surface transmissivity
slope_error : float
    [mrad] Surface RMS slope error, half-angle
spec_error : float
    [mrad] Surface specularity error, half-angle
usereflтable : bool
    Flag specifying use of user reflectivity table to modify reflectivity as a function of incidence angle
refltable : [[float,float],]
    [mrad,0..1] 2D list containing pairs of [angle,reflectivity] values.
```

2.2.2 Member Data Documentation

2.2.2.1 dist_type

```
pysoltrace.PySolTrace.Optics.Face.dist_type
```

Distribution type for surface interactions.

One of: {'g':Gaussian, 'p':Pillbox, 'd':Diffuse }

2.2.2.2 refltable

`pysoltrace.PySolTrace.Optics.Face.refltable`

[mrad,0..1] 2D list containing pairs of [angle,reflectivity] values.

The documentation for this class was generated from the following file:

- pysoltrace.py

2.3 pysoltrace.PySolTrace.Optics Class Reference

Classes

- class Face

Public Member Functions

- def **__init__** (self, p_dll, int p_data, int id)
- int Create (self)

Public Attributes

- **name**

  Unique name for the optical property set.
- **id**

  Identifying integer associated with the property set.
- **front**

  properties associated with the front of the optical surface
- **back**

  properties associated with the back of the optical surface

2.3.1 Detailed Description

```
 *Optics* is a subclass of PySolTrace, and represents an optical property set.
A PySolTrace instance may have multiple Optics member instances, which are stored in
the PySolTrace.optics list.

Optics contains a subclass *Face*, which collects properties associated with the front
or back face of an optical surface.

Attributes
----------
p_dll
    Reference for API DLL, managed by PySolTrace
p_data
    Memory location for soltrace context, managed by PySolTrace
name : str
    Unique name for the optical property set
id : int
    Identifying integer associated with the property set
front : Face
    properties associated with the front of the optical surface
back : Face
    properties associated with the back of the optical surface

Methods
----------
Create
    Calls methods to instantiate and construct optical surface in the SolTrace context.
```

2.3.2 Member Function Documentation

2.3.2.1 Create()

```
int pysoltrace.PySolTrace.Optics.Create (
              self )


Create Optics instance in the SolTrace context.

Returns
----------
int
    1 if successful, 0 otherwise
```

The documentation for this class was generated from the following file:

- pysoltrace.py

2.4 pysoltrace.Point Class Reference

Public Member Functions

- def __init__ (self, x=0, y=0, z=0)
- def **__str__** (self)
- def __add__ (self, obj)
- def __sub__ (self, obj)
- def __mul__ (self, obj)
- def __floordiv__ (self, obj)
- def __truediv__ (self, obj)
- def radius (self)
- def unitize (self, bool inplace=False)

Public Attributes

- **x**

  (float) x-coordinate
- **y**

  (float) y-coordinate
- **z**

  (float) z-coordinate

2.4.1 Detailed Description

```
A simple class to manage points in Cartesian coordinates.
```

2.4.2 Constructor & Destructor Documentation

2.4.2.1 __init__()

```
def pysoltrace.Point.__init__ (
              self,
              x = 0,
              y = 0,
              z = 0 )
```

```
Parameters
==========
x : float
    x-coordinate
y : float
    y-coordinate
z : float
    z-coordinate
```

2.4.3 Member Function Documentation

2.4.3.1 __add__()

```
def pysoltrace.Point.__add__ (
              self,
              obj )
```

Add to the current point coordinate values.

```
Parameters
==========
obj : variant
    If obj = (Point), adds component-wise to the current point
    If obj = (float), adds obj to each component
Returns
=======
Point
    Reference to this point
```

2.4.3.2 __floordiv__()

```
def pysoltrace.Point.__floordiv__ (
              self,
              obj )
```

Divides the current point coordinate values, taking the floor of the result.

```
Parameters
==========
obj : variant
    If obj = (Point), divides current point component-wise
    If obj = (float), divides components of current point by obj
Returns
=======
Point
    Reference to this point
```

2.4.3.3 __mul__()

```
def pysoltrace.Point.__mul__ (
              self,
              obj )

Multiplies the current point coordinate values.

Parameters
==========
obj : variant
    If obj = (Point), multiplies the current point component-wise by obj
    If obj = (float), multiplies each component of the current point by obj
Returns
=======
Point
    Reference to this point
```

2.4.3.4 __sub__()

```
def pysoltrace.Point.__sub__ (
              self,
              obj )

Subtract from the current point coordinate values.

Parameters
==========
obj : variant
    If obj = (Point), subtracts component-wise from the current point
    If obj = (float), subtracts obj from each component
Returns
=======
Point
    Reference to this point
```

2.4.3.5 __truediv__()

```
def pysoltrace.Point.__truediv__ (
              self,
              obj )

Divides the current point coordinate values/

Parameters
==========
obj : variant
    If obj = (Point), divides current point component-wise
    If obj = (float), divides components of current point by obj
Returns
=======
Point
    Reference to this point
```

2.4.3.6 radius()

```
def pysoltrace.Point.radius (
              self )


Computes distance between current point and the origin (0,0,0)

Returns
=======
float
    Calculated radius
```

2.4.3.7 unitize()

```
def pysoltrace.Point.unitize (
              self,
             bool  inplace = False )


Converts current point into a unit vector with magnitude 1

Parameters
==========
inplace : bool = False
    Specifies whether the current point is converted to a unit vector in place, or whether the
    current point remains unchanged and a unitized copy of the vector is returned.
```

The documentation for this class was generated from the following file:

- pysoltrace.py

2.5 pysoltrace.PySolTrace Class Reference

Classes

- class Optics
- class Stage
- class Sun

Public Member Functions

- def **__init__** (self)
- def clear_context (self)
- int get_num_optics (self)
- def add_optic (self, str optic_name)
- int delete_optic (self, int optic_id)
- def add_sun (self)
- int get_num_stages (self)
- def add_stage (self)
- int delete_stage (self, int stage_id)
- def run (self, int seed=-1, as_power_tower=False)
- int get_num_intersections (self)
- def get_intersect_locations (self)
- def get_intersect_cosines (self)
- def get_intersect_elementmap (self)
- def get_intersect_stagemap (self)
- def get_intersect_raynumbers (self)
- def get_sun_stats (self)
- def get_ray_dataframe (self)
- bool validate (self)
- def util_calc_euler_angles (self, origin, aimpoint, zrot)
- def util_transform_to_local (self, posref, cosref, origin, rreftoloc)
- def util_transform_to_ref (self, posloc, cosloc, origin, rloctoref)
- def util_matrix_vector_mult (self, m, v)
- def util_calc_transforms (self, euler)
- def util_matrix_transpose (self, m)
- def util_rotation_arbitrary (self, theta, axis, axloc, pt)
- def **util_calc_unitvect** (self, vect)
- float util_calc_zrot_azel (self, vect)
- def write_soltrace_input_file (self, str path)

Public Attributes

- **optics**

  List of Optics instances.
- **stages**

  List of Stage instances.
- **num_ray_hits**

  Minimum number of simulation ray hits.
- **max_rays_traced**

  Maximum number of ray hits in a simulation.
- **is_sunshape**

  Flag indicating whether sunshape should be included.
- **is_surface_errors**

  Flag indicating whether surface errors should be included.
- **sun**

  Object containing Sun class data.

2.5.1 Detailed Description

```
A class to access PySolTrace (SolTrace's Python API)

Attributes
----------
pdll : ctypes.CDLL
    loaded SolTrace library of exported functions
p_data
    Memory location for soltrace context
optics : [PySolTrace.Optics,]
    List of Optics instances
stages : [PySolTrace.Stage,]
    List of Stage instances
num_ray_hits : int
    Minimum number of simulation ray hits
max_rays_traced : int
    Maximum number of ray hits in a simulation
is_sunshape : bool
    Flag indicating whether sunshape should be included
is_surface_errors : bool
    Flag indicating whether surface errors should be included

Methods
-------
clear_context
    Frees SolTrace instance from memory at p_data
get_num_optics
    Get number of optical elements in the SolTrace context
add_optics
    Instantiates a new PySolTrace.Optics object
delete_optic
    Delete Optics instance
add_sun
    Adds Sun instance
get_num_stages
    Get number of Stages
add_stage
    Adds Stage instance
validate
    Detect common errors in simulation setup
run
    Runs SolTrace simulation
get_num_intersections
    Get the number of simulation ray intersections
get_ray_dataframe
    Get all simulation ray data in pandas dataframe
get_intersect_locations
    Get ray intersection locations
get_intersect_cosines
    Get ray intersection cosines / direction vectors
get_intersect_elementmap
    Get ray intersection associated element
get_intersect_stagemap
    Get ray intersection associated stage
get_intersect_raynumbers
    Get ray intersection number
get_sun_stats
    Get information on sun box
util_calc_euler_angles
    Calculate Euler angles for a position, aimpoint, and rotation
util_transform_to_local
    Transform a coordinate system from reference to a local system
util_transform_to_ref
    Transform a coordinate system from local to reference system
util_matrix_vector_mult
    Calculate product of a square matrix and a vector
util_calc_transforms
```

```
    Calculate matrix transforms
util_matrix_transpose
    Compute the transpose of a matrix
```

2.5.2 Member Function Documentation

2.5.2.1 add_optic()

```
def pysoltrace.PySolTrace.add_optic (
              self,
             str optic_name )
```

```
Instantiates a new PySolTrace.Optics object, adding it to the optics list, and
creating the optics in the SolTrace context. This method does not set optics
properties, which instead is done using the Optics.Create method.

Parameters
----------
optic_name : string
    Unique name for this Optics instance.

Returns
----------
Optics
    Reference to the Optics object that was just created.
```

2.5.2.2 add_stage()

```
def pysoltrace.PySolTrace.add_stage (
              self )
```

```
Adds a new Stage instance to the PySolTrace.stages list and creates the stage in
the SolTrace context. The Stage ID is automatically generated based on the number
of current stages.

Returns
----------
PySolTrace.Stage
    Reference to the newly created Stage object.
```

2.5.2.3 add_sun()

```
def pysoltrace.PySolTrace.add_sun (
              self )
```

```
Instantiates a PySolTrace.Sun object and associates it with the PySolTrace.sun member.
This does not create or modify the Sun data in the SolTrace context.

Returns
----------
PySolTrace.Sun
    Reference to newly created Sun instance.
```

2.5.2.4 clear_context()

```
def pysoltrace.PySolTrace.clear_context (
              self )
```

```
Frees SolTrace instance from memory

Returns
-------
bool
    True if successful, False otherwise
```

2.5.2.5 delete_optic()

```
int pysoltrace.PySolTrace.delete_optic (
              self,
             int optic_id )
```

```
Delete Optics instance. The optics object is removed from the PySolTrace.optics list and
from the SolTrace context.

Parameters
----------
optic_id : int
    ID associated with the optics to be deleted

Returns
----------
int
    1 if successful, 0 otherwise
```

2.5.2.6 delete_stage()

```
int pysoltrace.PySolTrace.delete_stage (
              self,
             int stage_id )
```

```
Delete Stage instance. The stage object is removed from the PySolTrace.stages list and
from the SolTrace context.

Parameters
----------
stage_id : int
    ID associated with the stage to be deleted

Returns
----------
int
    1 if successful, 0 otherwise
```

2.5.2.7 get_intersect_cosines()

```
def pysoltrace.PySolTrace.get_intersect_cosines (
              self )
```

```
[Post simulation] Get the ray intersection cosines (direction vectors) in global coordinates.

Returns
----------
[[cx,cy,cz],]
    2D array of ray vectors associated with the intersection at the same index.
```

2.5.2.8 get_intersect_elementmap()

```
def pysoltrace.PySolTrace.get_intersect_elementmap (
              self )
```

```
[Post simulation] Get the elements associated with the ray intersection. The element numbers are
interpreted as follows:
    0     | Ray did not hit an element in this stage
    >= +1 | Ray hit this element ID and was reflected / refracted
    <= -1 | Ray hit this element ID and was absorbed

Returns
----------
[int,]
    2D array of integers associated with the intersection at the same index.
```

2.5.2.9 get_intersect_locations()

```
def pysoltrace.PySolTrace.get_intersect_locations (
              self )
```

[Post simulation] Get the ray intersection locations in global coordinates.

```
Returns
----------
[[x,y,z],]
    2D array of ray positions
```

2.5.2.10 get_intersect_raynumbers()

```
def pysoltrace.PySolTrace.get_intersect_raynumbers (
              self )
```

[Post simulation] Get the ray intersection numbers.

```
Returns
----------
[int,]
    List of ray intersection numbers.
```

2.5.2.11 get_intersect_stagemap()

```
def pysoltrace.PySolTrace.get_intersect_stagemap (
              self )
```

[Post simulation] Get the stage associated with the ray intersection at the same positional index.

```
Returns
----------
[int,]
    List of stages associated with the intersection at the same index.
```

2.5.2.12 get_num_intersections()

```
int pysoltrace.PySolTrace.get_num_intersections (
              self )
```

[Post simulation] Get the number of ray intersections detected in the simulation.

```
Returns
----------
int
    Number of intersections
```

2.5.2.13 get_num_optics()

```
int pysoltrace.PySolTrace.get_num_optics (
              self )
```

Get number of optical elements in the SolTrace context

```
Returns
----------
int
    number of elements
```

2.5.2.14 get_num_stages()

```
int pysoltrace.PySolTrace.get_num_stages (
              self )
```

Retrieve the number of stages that have been created in the SolTrace context.

```
Returns
----------
int
    Number of stages.
```

2.5.2.15 get_ray_dataframe()

```
def pysoltrace.PySolTrace.get_ray_dataframe (
              self )
```

Get a pandas dataframe with all of the ray data from the simulation.

```
Returns
----------
Pandas.DataFrame
    with columns:
    loc_x   | Ray hit location, x-coordinate
    loc_y   | Ray hit location, y-coordinate
    loc_z   | Ray hit location, z-coordinate
    cos_x   | Ray directional vector, x-component
    cos_y   | Ray directional vector, y-component
    cos_z   | Ray directional vector, z-component
    element | Element associated with ray hit
    stage   | Stage associated with ray hit
    number  | Ray number
```

2.5.2.16 get_sun_stats()

```
def pysoltrace.PySolTrace.get_sun_stats (
              self )
```

Get information on the sun box.

```
Returns
----------
dict
    Keys in the return dictionary are:
    'xmin' --> Minimum x extent of the bounding box for hit testing
    'xmax' --> Maximum x extent of the bounding box for hit testing
    'ymin' --> Minimum y extent of the bounding box for hit testing
    'ymax' --> Maximum y extent of the bounding box for hit testing
    'nsunrays' --> Number of sun rays simulated
```

2.5.2.17 run()

```
def pysoltrace.PySolTrace.run (
              self,
             int  seed = -1,
              as_power_tower = False )
```

Run SolTrace simulation.

```
Parameters
----------
seed : int
    Seed for random number generator. [-1] for random seed.
as_power_tower : bool
    Flag indicating simulation should be processed as power
    tower / central receiver type, with corresponding efficiency adjustments.

Returns
----------
int
    Simulation return value
```

2.5.2.18 util_calc_euler_angles()

```
def pysoltrace.PySolTrace.util_calc_euler_angles (
              self,
              origin,
              aimpoint,
              zrot )
```

```
Calculate the Euler angles associated with a given origin, aimpoint, and z-axis rotation.

Parameters
----------
origin : [float,*3]
    Origin of the coordinate system
aimpoint : [float,*3]
    Aimpoint of the vector originating at the origin
zrot : float
    Rotation around the z-axis coordinate (degr)

Returns
----------
list
    Calculated Euler angles (rad)
```

2.5.2.19 util_calc_transforms()

```
def pysoltrace.PySolTrace.util_calc_transforms (
              self,
              euler )

Calculate matrix transforms

Parameters
----------
euler : [float,]*3
    Euler angles

Returns
----------
(dict) A dictionary containing the keys:
    rreftoloc : Transformation matrix from Reference to Local system
    rloctoref : Transformation matrix from Local to Reference system
```

2.5.2.20 util_calc_zrot_azel()

```
float pysoltrace.PySolTrace.util_calc_zrot_azel (
              self,
              vect )

Compute the z-rotation of a vector, assuming the vector's deviation from (0,0,1)
has been realized using azimuth-elevation transforms.

Parameters
----------
vect : (list OR Point)
    i,j,k components of a vector

Returns
----------
float
    Computed z-rotation (degrees)
```

2.5.2.21 util_matrix_transpose()

```
def pysoltrace.PySolTrace.util_matrix_transpose (
              self,
              m )
```

```
Calculate matrix transpose

Parameters
----------
m : [[float,]*3]*3
    Square matrix 3x3 to be transposed.

Returns
----------
[[float,]*3]*3
    Square matrix 3x3, transpose of 'm'
```

2.5.2.22 util_matrix_vector_mult()

```
def pysoltrace.PySolTrace.util_matrix_vector_mult (
              self,
              m,
              v )
```

```
Perform multiplication of a 3x3 matrix and a length-3 vector, returning the result vector.

Parameters
----------
m : list
    m[3][3] - a 3x3 matrix
v : list
    v[3] - a list, length 3

Returns
----------
list
    m x v [3]
```

2.5.2.23 util_rotation_arbitrary()

```
def pysoltrace.PySolTrace.util_rotation_arbitrary (
              self,
              theta,
              axis,
              axloc,
              pt )
```

```
Rotation of a point 'pt' about an arbitrary axis with direction 'axis' centered at point 'axloc'.
The point is rotated through 'theta' radians.

Parameters
----------
theta : float
    Angle of rotation (rad)
axis : Point()
    Vector (x=i,y=j,z=k) indicating direction of axis for rotation
axloc : Point()
    Location of the axis origin
pt : Point()
    Location of the point that is to be rotated

Returns
-----------
Point
    Point after rotation
```

2.5.2.24 util_transform_to_local()

```
def pysoltrace.PySolTrace.util_transform_to_local (
              self,
              posref,
              cosref,
              origin,
              rreftoloc )

Perform coordinate transformation from reference system to local system.

Parameters
----------
PosRef : [float,]*3
    X,Y,Z coordinates of ray point in reference system
CosRef : [float,]*3
    Direction cosines of ray in reference system
Origin : [float,]*3
    X,Y,Z coordinates of origin of local system as measured in reference system
RRefToLoc
    Rotation matrices required for coordinate transform from reference to local

Returns
----------
(dict)  Keys in return dictionary include:
    posloc : ([float,]*3) X,Y,Z coordinates of ray point in local system
    cosloc : ([float,]*3) Direction cosines of ray in local system
```

2.5.2.25 util_transform_to_ref()

```
def pysoltrace.PySolTrace.util_transform_to_ref (
              self,
              posloc,
              cosloc,
              origin,
              rloctoref )
```

```
Perform coordinate transformation from local system to reference system.

Parameters
----------
PosLoc : [float,]*3
    X,Y,Z coordinates of ray point in local system
CosLoc : [float,]*3
    Direction cosines of ray in local system
Origin : [float,]*3
    X,Y,Z coordinates of origin of local system as measured in reference system
RLocToRef
    Rotation matrices required for coordinate transform from local to reference
    -- inverse of reference to local transformation

Returns
----------
dict
    Keys in return dictionary include:
    posref : ([float,]*3) X,Y,Z coordinates of ray point in reference system
    cosref : ([float,]*3) Direction cosines of ray in reference system
```

2.5.2.26 validate()

```
bool pysoltrace.PySolTrace.validate (
              self )

Validate that the current SolTrace context has been configured correctly, and
that commonly missed inputs are specified.

Returns
----------
bool
    False if error is identified, True otherwise
```

2.5.2.27 write_soltrace_input_file()

```
def pysoltrace.PySolTrace.write_soltrace_input_file (
              self,
             str path )

Write a SolTrace input file (.stinput) based on the currently created API objects. This file is written
using the objects and data in the PySolTrace instance, not necessarily on what has been created in the
coretrace 'context' data space. The 'context' may not match the PySolTrace instance if not all 'Create()'
methods have been called.

Parameters
==========
path : str
    Path and file name to be used to write the resulting .stinput file

Returns
=======
None
```

The documentation for this class was generated from the following file:

- pysoltrace.py

2.6 pysoltrace.PySolTrace.Stage Class Reference

Classes

- class Element

Public Member Functions

- def __init__ (self, pdll, int p_data, int id)
- int Create (self)
- int get_num_elements (self)
- int add_element (self)

Public Attributes

- **id**

  Identifying integer associated with the stage.
- **position**

  Stage location in global coordinates.
- **aim**

  Coordinate system aim point in global coordinates.
- **zrot**

  [deg] Rotation of coordinate system around z-axis
- **is_virtual**

  Flag indicating virtual stage.
- **is_multihit**

  Flag indicating that rays can have multiple interactions within a single stage.
- **is_tracethrough**

  Flag indicating the stage is in trace-through mode.
- **name**

  Descriptive name for this stage.
- **elements**

  list of all elements in the stage

2.6.1 Detailed Description

```
 *Stage* is a subclass of PySolTrace, and represents a grouping of elements.
A PySolTrace instance may have multiple Stage member instances, which are stored in
the PySolTrace.stages list.

Stage contains a subclass *Element*, which collects properties and geometry associated
with individual geometric elements.

Attributes
----------
p_dll
    Reference for API DLL, managed by PySolTrace
p_data
```

```
    Memory location for soltrace context, managed by PySolTrace
id : int
    Identifying integer associated with the stage
position : Point
    Stage location in global coordinates
aim : Point
    Coordinate system aim point in global coordinates
zrot : float
    [deg] Rotation of coordinate system around z-axis
is_virtual : bool
    Flag indicating virtual stage
is_multihit : bool
    Flag indicating that rays can have multiple interactions within a single stage.
is_tracethrough : bool
    Flag indicating the stage is in trace-through mode
name : str
    Descriptive name for this stage
elements : [Stage.Element,]
    list of all elements in the stage

Methods
----------
Create
    Calls methods to instantiate and construct a stage in the context.
get_num_elements
    Returns number of elements associated with this stage (from context).
add_elements
    Creates new element in Stage.element[] list, and adds to context
```

2.6.2 Constructor & Destructor Documentation

2.6.2.1 __init__()

```
def pysoltrace.PySolTrace.Stage.__init__ (
              self,
              pdll,
             int p_data,
             int id )
```

2.6.3 Member Function Documentation

2.6.3.1 add_element()

```
int pysoltrace.PySolTrace.Stage.add_element (
              self )
```

```
Add one element to the stage. This method appends an Element object to the
stage's Stage.elements list and adds the element to the SolTrace context.
To update element properties and settings, call the Element.Create method
on each element.

Returns
----------
PySolTrace.Stage.Element
    Reference to the newly created element
```

2.6.3.2 Create()

```
int pysoltrace.PySolTrace.Stage.Create (
              self )
```

```
Create Stage instance in the SolTrace context.
Note: This does not create any associated Elements, which must have their Create method called separately.

Returns
----------
int
    1 if successful, 0 otherwise
```

2.6.3.3 get_num_elements()

```
int pysoltrace.PySolTrace.Stage.get_num_elements (
              self )
```

```
Retrieve the number of elements associated with this stage that are currently created
in the SolTrace context.

Returns
----------
int
    Number of elements
```

The documentation for this class was generated from the following file:

- pysoltrace.py

2.7 pysoltrace.PySolTrace.Sun Class Reference

Public Member Functions

- def **__init__** (self, pdll, p_data)
- def Create (self)

Public Attributes

- **point_source**

  Flag indicating whether the sun is modeled as a point source at a finite distance.
- shape

  Sun shape model.
- sigma

  [mrad] Half-width or std.
- **position**

  Location of the sun/sun vector in global coordinates.
- user_intensity_table

  [mrad, 0..1] 2D list containing pairs of angle deviation from sun vector and irradiation intensity.

2.7.1 Detailed Description

```
 *Sun* is a subclass of PySolTrace, and represents a sun property set.
A PySolTrace instance may have a single Sun member instance, which is stored as the
PySolTrace.sun member.

Attributes
----------
p_dll
    Reference for API DLL, managed by PySolTrace
p_data
    Memory location for soltrace context, managed by PySolTrace
point_source : bool
    Flag indicating whether the sun is modeled as a point source at a finite distance.
shape : char
    Sun shape model. One of: {'p':Pillbox, 'g':Gaussian, 'd':data table, 'f':gray diffuse}
sigma : float
    [mrad] Half-width or std. dev. of the error distribution
position : Point
    Location of the sun/sun vector in global coordinates
user_intensity_table : [[float,float],]
    [mrad, 0..1] 2D list containing pairs of
    angle deviation from sun vector and irradiation intensity.
    A typical table will have angles spanning 0->~5mrad, and inten-
    sities starting at 1 and decreasing to zero. The table must
    contain at least 2 entries.
Methods
----------
Create
    Calls methods to instantiate and construct optical surface in the SolTrace context.
```

2.7.2 Member Function Documentation

2.7.2.1 Create()

```
def pysoltrace.PySolTrace.Sun.Create (
              self )
```

```
Create Sun instance in the SolTrace context.

Returns
----------
int
    1 if successful, 0 otherwise
```

2.7.3 Member Data Documentation

2.7.3.1 shape

```
pysoltrace.PySolTrace.Sun.shape
```

Sun shape model.

One of: {'p':Pillbox, 'g':Gaussian, 'd':data table, 'f':gray diffuse}

2.7.3.2 sigma

```
pysoltrace.PySolTrace.Sun.sigma
```

[mrad] Half-width or std.

dev. of the error distribution

2.7.3.3 user_intensity_table

```
pysoltrace.PySolTrace.Sun.user_intensity_table
```

[mrad, 0..1] 2D list containing pairs of angle deviation from sun vector and irradiation intensity.

A typical table will have angles spanning 0->∼5mrad, and inten- sities starting at 1 and decreasing to zero. The table must contain at least 2 entries.

The documentation for this class was generated from the following file:

- pysoltrace.py

Index