```
SHADOWLibExtensions: methods definitions and parameters.
All the definitions are extracted from: Shadow3/ShadwoLibExtensions.py


 Possible choice for col are:
             1   X spatial coordinate [user's unit]
             2   Y spatial coordinate [user's unit]
             3   Z spatial coordinate [user's unit]
             4   Xp direction or divergence [rads]
             5   Yp direction or divergence [rads]
             6   Zp direction or divergence [rads]
             7   X component of the electromagnetic vector (s-polariz)
             8   Y component of the electromagnetic vector (s-polariz)
             9   Z component of the electromagnetic vector (s-polariz)
            10   Lost ray flag
            11   Energy [eV]
            12   Ray index
            13   Optical path length
            14   Phase (s-polarization) in rad
            15   Phase (p-polarization) in rad
            16   X component of the electromagnetic vector (p-polariz)
            17   Y component of the electromagnetic vector (p-polariz)
            18   Z component of the electromagnetic vector (p-polariz)
            19   Wavelength [A]
            20   R= SQRT(X^2+Y^2+Z^2)
            21   angle from Y axis
            22   the magnituse of the Electromagnetic vector
            23   |E|^2 (total intensity)
            24   total intensity for s-polarization
            25   total intensity for p-polarization
            26   K = 2 pi / lambda [A^-1]
            27   K = 2 pi / lambda * col4 [A^-1]
            28   K = 2 pi / lambda * col5 [A^-1]
            29   K = 2 pi / lambda * col6 [A^-1]
            30   S0-stokes = |Es|^2 + |Ep|^2
            31   S1-stokes = |Es|^2 - |Ep|^2
            32   S2-stokes = 2 |Es| |Ep| cos(phase_s-phase_p)
            33   S3-stokes = 2 |Es| |Ep| sin(phase_s-phase_p)
            34   Power = intensity(col 23) * energy (col 11)
            35   Angle-X with Y: |arcsin(X')|
            36   Angle-Z with Y: |arcsin(Z')|
            37   Angle-X with Y: |arcsin(X') - mean(arcsin(X'))|
            38   Angle-Z with Y: |arcsin(Z') - mean(arcsin(Z'))|


FROM CLASS BEAM:

 def duplicate(self):
      beam_copy = Beam()
      beam_copy.rays = copy.deepcopy(self.rays)
      return beam_copy

 def rotate(self,theta1,axis=1,rad=1):
    """

    :param theta1: the rotation angle in degrees (default=0)
    :param axis: The axis number (Shadow's column) for the rotation
                 (i.e, 1:x (default), 2:y, 3:z)
    :param file:
    :param rad: set this flag when theta1 is in radiants
    :return:
    """

```

```
def traceCompoundOE(self,compoundOE,from_oe=1,write_start_files=0,write_end_files=0,\
                    write_star_files=0, write_mirr_files=0):
      """
      traces a compound optical element

      Note that when using write_*_files keyword, the files are written by python, not
                  by SHADOW (so FWRITE is not changed), with the exception of write_
                  mirr_files. In this case the code changes in the oe copy
                  FWRITE=1 (mirror files only). This affects the returned list of oe's
                  after tracing.

      :param compoundOE: input object
      :param from_oe: index of the first oe (for tracing compoundOE after an existing
                    system) (default=1)
      :param write_start_files: 0=No (default), 1=Yes (all), 2: only first and last ones
      :param write_end_files:  0=No (default), 1=Yes (all), 2: only first and last ones
      :param write_star_files:  0=No (default), 1=Yes (all), 2: only first and last ones
      :param write_mirr_files:  0=No (default), 1=Yes (all), 2: only first and last ones
      :return: a new compoundOE with the list of the OE objects after tracing (the info
                    of end.xx files)
      """

def getshonecol(self,col, nolost=0):
    '''
    Extract a column from a shadow file (eg. begin.dat) or a Shadow.Beam instance.
    The column are numbered in the fortran convention, i.e. starting from 1.
    It returns a numpy.array filled with the values of the chosen column.

    Inumpy.ts:
       beam     : str instance with the name of the shadow file to be loaded. OR
                  Shadow.Beam initialized instance.
       col      : int for the chosen columns.

    Outputs:
       numpy.array 1-D with length numpy.INT.

    Error:
       if an error occurs an ArgsError is raised.
    '''

def getshcol(self,col,nolost=0):
      '''
      Extract multiple columns from a shadow file (eg.'begin.dat') or a Shadow.Beam
      instance.
      The column are numbered in the fortran convention, i.e. starting from 1.
      It returns a numpy.array filled with the values of the chosen column.

      Inputs:
         beam     : str instance with the name of the shadow file to be loaded. OR
                    Shadow.Beam initialized instance.
         col      : tuple or list instance of int with the number of columns chosen.

      Outputs:
         numpy.array 2-D with dimension R x numpy.INT. Where R is the total number of
         column chosen

      Error:
         if an error occurs an ArgsError is raised.
    '''
```

```
127 def histo1(self,col,xrange=None,nbins=50,nolost=0,ref=0,write=None,factor=1.0,\
128             calculate_widths=1,calculate_hew=0):
129       """
130       Calculate the histogram of a column, simply counting the rays, or weighting with
131       another column. It returns a dictionary which contains the histogram data.
132 
133       :param col: int for the chosen column.
134       :param xrange:  tuple or list of length 2 describing the interval of interest for
…  x,
135                       the data read from the chosen column. (default: None, thus using min
136                       and max of the array)
137       :param nbins:  number of bins of the histogram.
138       :param nolost:
139                0   All rays
140                1   Only good rays
141                2   Only lost rays
142       :param ref:
143                0, None, "no", "NO" or "No":   only count the rays
144                23, "Yes", "YES" or "yes":     weight with intensity (look at col=23 |E|^2
145                total intensity)
146                other value: use that column as weight
147       :param write:
148                None (default)   don't write any file
149                file_name   write the histogram into the file 'file_name'.
150       :param factor:  a scalar factor to multiply the selected column before
…  histogramming
151                      (e.g., for changing scale from cm to um then factor=1e4).
152       :param calculate_widths:
153       :return:         a python dictionary with the calculated histogram. The following
154                         keys are set:
155                         error, col, write, nolost, nbins, xrange, factor
156                         histogram, bins, histogram_sigma, bin_center, bin_left,
…  bin_right,
157                         intensity, fwhm, nrays, good_rays,
158 
159         Inputs:
160          beam     : str instance with the name of the shadow file to be loaded, or a
161                     Shadow.Beam initialized instance.
162          col      :
163 
164       Optional keywords:
165          xrange   :
166          nbins    :
167          nolost   :
168          ref      :
169          write    :
170          factor   :
171     """
172 
173 def histo2(self,col_h,col_v,nbins=25,ref=23, nbins_h=None, nbins_v=None, nolost=0,
174             xrange=None,yrange=None, calculate_widths=1):
175     """
176 
177     performs 2d histogram to prepare data for a plotxy plot
178 
179     It uses histogram2d for calculations
180 
181     Note that this Shadow.Beam.histo2 was previously called Shadow.Beam.plotxy
182 
183     :param col_h: the horizontal column
184     :param col_v: the vertical column
185     :param nbins: number of bins
186     :param ref      :
```

```
                0, None, "no", "NO" or "No":   only count the rays
                23, "Yes", "YES" or "yes":     weight with intensity (look at col=23
                     |E|^2 total intensity)
                other value: use that column as weight
    :param nbins_h: number of bins in H
    :param nbins_v: number of bins in V
    :param nolost: 0 or None: all rays, 1=good rays, 2=only losses
    :param xrange: range for H
    :param yrange: range for V
    :param calculate_widths: 0=No, 1=calculate FWHM (default), 2=Calculate FWHM and FW
                         at 25% and 75% if Maximum
    :return: a dictionary with all data needed for plot
    """

 FROM CLASS OE:


 def set_empty(self,T_INCIDENCE=0,T_REFLECTION=180.0,T_SOURCE=0.0,T_IMAGE=0.0,ALPHA=0.0):
    """
    Defines an empty optical element (useful as an arm, e.g. to rotate reference frames)
    By default, there is no change in the optical axis direction.

    :param T_INCIDENCE: incidence angle (default=0)
    :param T_REFLECTION: reflection angle (default=180)
    :param T_SOURCE: distance from previous o.e. (default=0)
    :param T_IMAGE: ddistance to next or (default=0)
    :param ALPHA: mirror oriantation angle (default=0)
    :return:
    """""

  def mirinfo(self, title=None):
    '''
    mimics SHADOW mirinfo postprocessor. Returns a text array.
    :return: a text array with the result
    '''

FROM CLASS CompoundOE:

  def info(self,file=''):
    """
    write a summary of the real distances, focal distances and orientation angles.
    :param file: set to a file name to dump tesult into ir
    :return: a text array
    """

  def syspositions(self):
    """
    return a dictionary with the positions of the source, o.e. and images

    :return: dic
            dic["source"]  numpy  array (icoor) with the three coordinates of source
            dic["source"]  numpy  array (n_oe,icoor) with the three coordinates of
                           optical elements for all elements
            dic["source"]  numpy  array (n_oe,icoor) with the three coordinates of image
                           positions for all elements
            dic["optical_axis_x"] numpy array with the x coordinates of the optical axis
            dic["optical_axis_y"] numpy array with the y coordinates of the optical axis
            dic["optical_axis_z"] numpy array with the z coordinates of the optical axis
    """
```

```
250
251   def get_oe_index(self,oe_index):
252       """
253       returns the pointer to the oe with index oe_index
254       :param oe_index:
255       :return:
256       """
257
258   def duplicate(self):
259       """
260       Makes a copy of the compound optical element
261       :return:
262       """
263
264  def add_drift_space_downstream(self,dd):
265       """
266       Adds empty space to the last element of the compound oe
267       :param dd: The distance
268       :return: None
269       """
270
271   def add_drift_space_upstream(self,dd):
272       """
273       Adds empty space before the first element of the compound oe
274       :param dd: The distance
275       :return: None
276       """
277
278   def append(self,item):
279     """
280     append an instance of Shadow.OW or Shadow.CompoundOE
281     :param item: an OE or CompoundOE to append
282     :return: the CompoundOE updated with a copy of item appended
283     """
284
285 def append_lens(self,p,q,surface_shape=1,convex_to_the_beam=1,diameter=None,
286                 cylinder_angle=None, prerefl_file=None, refraction_index=1.0,\
287                 attenuation_coefficient=0.0,\ radius=500e-2,interthickness=0.001,\
288                 use_ccc=0):
289       """
290       Adds and sets a lens (two interfaces) to the compound optical element
291
292       :param p: distance source-first lens interface
293       :param q: distance last lens interface to image plane
294       :param surface_shape: 1=sphere 4=paraboloid, 5=plane (other surfaces not yet
…  implamented)
295       :param convex_to_the_beam: convexity of the first interface exposed to the beam
…  0=No, 1=Yes
296                                  the second interface has opposite convexity
297       :param diameter: lens diameter. Set to None for infinite dimension
298       :param cylinder_angle: None=not cylindrical, 0=meridional 90=sagittal
299       :param prerefl_file:file name (from prerefl) to get the refraction index. If set
300                 then the keywords refraction_index and attenuation_coefficient are not
…  used.
301       :param refraction_index: n (real) #ignored if prerefl_file points to file.
302       :param attenuation_coefficient:mu (real); ignored if prerefl file points to file.
303       :param radius: lens radius (for pherical, or radius at the tip for paraboloid)
304       :param interthickness: lens thickness (distance between the two interfaces at the
305                               center of the lenses)
306       :param use_ccc 0=set shadow using surface shape (FMIRR=1,4,5), 1=set shadow using
307                               CCC coeffs (FMIRR=10)
308       :return: self
309       """
```

```
310
311  def append_crl(self,p0,q0, nlenses=30, slots_empty=0, radius=500e-2, thickness=625e-4,
312                 interthickness=0.001, surface_shape=1, convex_to_the_beam=1, \
313                 diameter=None, cylinder_angle=None, prerefl_file=None,\
314                 refraction_index=1.0, attenuation_coefficient=0.0, use_ccc=0):
315         """
316         Builds the stack of oe for a CRL.
317
318         Notes: if nlenses=0 sets a single "lens" with flat interfaces and no change of
319                 refraction index (like empty)
320
321                 The refraction index should be input either by i) prerefl_index or ii)
322                 refraction_index and
323                 attenuation_coefficient keywords. The first one is prioritary.
324
325                 slots_empty: if different from zero, adds a distance equal to
326                 thickness*slots_empty to q0. The
327                 intention is to simulate a lens that is off but the path should be
328                 considered.
329
330
331
332         :param p0:distance source-first lens interface
333         :param q0:distance last lens interface to image plane
334         :param nlenses: number of lenses
335         :param slot_empty: number of empty slots (default=0)
336         :param radius:lens radius (for pherical, or radius at the tip for paraboloid)
337         :param thickness: lens thickness (piling thickness)
338         :param interthickness:lens thickness (distance between the two interfaces at the
339                                 center of the lenses)
340         :param surface_shape:1=sphere 4=paraboloid, 5=plane (other surfaces not yet
341                               implamented)
342         :param convex_to_the_beam:convexity of the first interface exposed to the
343                               beam 0=No, 1=Yes the second interface has opposite convexity
344         :param diameter:lens diameter. Set to None for infinite dimension
345         :param cylinder_angle:None=not cylindrical, 0=meridional 90=sagittal
346         :param prerefl_file:file name (from prerefl) to get the refraction index. If set
347                 then the keywords refraction_index and attenuation_coefficient are not
…  used.
348         :param refraction_index: n (real) #ignored if prerefl_file points to file.
349         :param attenuation_coefficient:mu (real); ignored if prerefl file points to file.
350         :param use_ccc:0=set shadow using surface shape (FMIRR=1,4,5), 1=set shadow
351                           using CCC coeffs (FMIRR=10)
352         :return: self
353         """
354
355 def append_transfocator(self,p0,q0, nlenses=[4,8], slots_empty=0, radius=500e-2,
356             thickness=625e-4, interthickness=0.001, surface_shape=1, \
357             convex_to_the_beam=1, diameter=None, cylinder_angle=None, prerefl_file=None,
…  \
358             refraction_index=1.0, attenuation_coefficient=0.0,use_ccc=0):
359         """
360         Builds the stack of oe for a TRANSFOCATOR. A transfocator is a stack of CRLs.
361         append_transfocator is therefore very similar to append_crl, but now arguments
362         are lists instead of scalar. However, if the value of a particular keyword is an
363         scalar and a list is expected, then it is automatically replicated "nslots"
364         times, where nslots=len(nlenses)
365
366         Notes: if nlenses=0 sets a single "lens" with flat interfaces and no change of
367                 refraction index (like empty)
368
369                 The refraction index should be input either by
370                     i) prerefl_index or
```

```
                    ii) refraction_index and attenuation_coefficient keywords.
                    The first one is prioritary.

                slots_empty: if different from zero, adds a distance equal to
                thickness*slots_empty to q0. The intention is to simulate a lens that is
                off but the path should be considered.

                Note that all arrays must be "list". If you are using numpy arrays,
                convert them:  array.tolist()



        :param p0 (list):distance previous continuation plane to first lens for each CRL
                (usually [p,0,0,...]
        :param q0 (scalar):distance last lens in each CRLto continuation plane
        :param nlenses (list): number of lenses
        :param slots_empty (list): number of empty slots
        :param radius (list):lens radius (for pherical, or radius at the tip for
                            paraboloid)
        :param thickness (list): lens thickness (piling thickness)
        :param interthickness (list):lens thickness (distance between the two interfaces
                                    at the center of the lenses)
        :param surface_shape (list):1=sphere 4=paraboloid, 5=plane (other surfaces not
                                    yet implamented)
        :param convex_to_the_beam (list):convexity of the first interface exposed to the
                                    beam 0=No, 1=Yes
                                  the second interface has opposite convexity
        :param diameter (list):lens diameter. Set to None for infinite dimension
        :param cylinder_angle (list):None=not cylindrical, 0=meridional 90=sagittal
        :param prerefl_file (list):file name (from prerefl) to get the refraction index.
If set
                then the keywords refraction_index and attenuation_coefficient are not
used.
        :param refraction_index (list): n (real) #ignored if prerefl_file points to file.
        :param attenuation_coefficient (list):mu (real); ignored if prerefl file points
to file.
        :param use_ccc (scalar):0=set shadow using surface shape (FMIRR=1,4,5), 1=set
                                shadow using CCC coeffs (FMIRR=10)
        :return: self
        """
def append_kb(self, p0, q0, grazing_angles_mrad=[3.0,3.0],separation=100.0,\
             mirror_orientation_angle=0,focal_positions=[0,0],shape=[2,2],\
          dimensions1=[0,0],dimensions2=[0,0], reflectivity_kind=[0,0],\
          reflectivity_files=["",""],surface_error_files=["",""]):
      """
      Appends a KB (Kirkpatrick-Baez) system
      First mirror is focusing in vertical plane, second mirror focusses in horizontal
plane.

      Note that SHADOW rotates the image plane because the second mirror has mirror
      orientation angle 90 ded


      :param p0: distance from previous source plane (continuation plane) to center of
                    KB (!!!)
      :param q0: distance from center of KB (!!!) to image plane (continuation plane)
      :param grazing_angles_mrad: grazing angle in mrad for both mirrors. Default:
                                  grazing_angles_mrad=[3.0,3.0]
      :param separation: separation between the mirrors from center of first mirror (VFM)
                        to center of second one (HFM).
                                ()Default=100). T
                                he continuation plane is set in the middle.
      :param mirror_orientation_angle: set the mirror orientation angle with respect to
```

```
                                the previous o.e. for the first mirror of the  KB
      :param focal_positions: the focal positions [p_focal,q_focal] measured from the
                              center of KB. If set to [0,0] then uses p0 and q0
      :param shape: the shape code for the surfaces 1=spherica, 2=elliptical.
                    Default: shape=[2,2]
      :param dimensions1: the dimensions [width,length] for the first mirror.
                          Default: [0,0] meaning infinite dimensions.
      :param dimensions2: the dimensions [width,length] for the second  mirror.
                          Default: [0,0] meaning infinite dimensions.
      :param reflectivity_kind: flag for reflectivity: 0=ideal reflector, 1=mirror,
                              2=multilayer. Default=[0,0]
                                 If reflectivity is set to mirror or multilayer,
                                 the corresponding file must be entered in the
                                 reflectivity_files keyword.
      :param reflectivity_files: the reflectivity files, if needed. If mirror, the file
                              must have been created by prerefl. If multilayer,
                              the file must come from pre_mlayer.
      :param surface_error_files: Set to file names containing the surface error mesh.
                                 Default: surface_error_files=["",""] which means that no
                                 surface error is considered.
      :return: self
      """

def append_monochromator_double_crystal(self, p0, q0, photon_energy_ev=14000,\
                              separation=0.0, dimensions1=[0,0],dimensions2=[0,0],\
          reflectivity_file=""):
      """
      Appends a double crystal monochromator (with plane crystals)

      :param p0: distance from previous source plane (continuation plane) to center of
                    doble crystal monochrtomator
      :param q0: distance from center of double crystal monochromator to image plane
                (continuation plane)
      :param set_photon_energy: photon energy in eV to set the monochromator
      :param separation: separation between the crystals (Default=0). The continuation
                         plane is set in the middle.
      :param dimensions1: the dimensions [width,length] for the first mirror.
                          Default: [0,0] meaning infinite dimensions.
      :param dimensions2: the dimensions [width,length] for the second  mirror.
                          Default: [0,0] meaning infinite dimensions.
      :param reflectivity_files: the reflectivity files as created by bragg
      :return: self
      """

FROM CLASS Source:

def to_dictionary(self):
    """
    returns a python dictionary of the Shadow.Source instance
    :return: a dictionary
    """

def duplicate(self):
        """
        makes a copy of the source
        :return: new instance of Shadow.Source()
        """

 #Gaussian source
def set_divergence_gauss(self, sigmaxp, sigmazp):
    """
    sets Gaussian source in divergence space
    :param sigmaxp: SIGDIX for SHADOW
```

```
493     :param sigmazp: SIGDIZ for SHADOW
494     :return: self
495     """
496
497 def set_spatial_gauss(self,sigmax, sigmaz):
498     """
499     sets Gaussian source in real space
500     :param sigmax: SIGMAX for SHADOW.
501     :param sigmaz: SIGMAZ for SHADOW.
502     :return: self
503     """
504
505 def set_gauss(self,sigmax,sigmaz,sigmaxp,sigmazp):
506     """
507     Sets a Gaussian source in both real and divergence spaces
508     :param sigmax: SIGMAX for SHADOW.
509     :param sigmaz: SIGMAZ for SHADOW.
510     :param sigmaxp: SIGDIX for SHADOW.
511     :param sigmazp: SIGDIZ for SHADOW.
512     :return: self
513     """
514
515 def set_energy_monochromatic(self,emin):
516     """
517     Sets a single energy line for the source (monochromatic)
518     :param emin: the energy in eV
519     :return: self
520     """
521
522 def set_energy_box(self,emin,emax):
523     """
524     Sets a box energy distribution for the source (monochromatic)
525     :param emin: minimum energy in eV
526     :param emax: maximum energy in eV
527     :return: self
528     """
529
530 def set_pencil(self):
531     """
532     Sets a pencil beam (zero size, zero divergence)
533     :return:
534     """
535
536
537 def apply_gaussian_undulator(self, undulator_length_in_m=1.0,user_unit_to_m=1e2, \
538                              verbose=1, und_e0=None):
539     """
540     Convolves the already defined Gaussian source (for the electrons) with the
541     photon emission for an undulator.
542
543     :param undulator_length_in_m:
544     :param user_unit_to_m:
545     :param verbose: set to 0 for silent output
546     :param und_e0: the setting photon energy in eV, if undefined (None) reads from SHADOW
547                    PH1 variable
548     :return: self
549     """
550
551 def sourcinfo(self,title=None):
552     '''
553     mimics SHADOW sourcinfo postprocessor. Returns a text array.
554     :return: a text string
555     '''
```