# Attributes

Attributes are inherited from parent nodes in most cases. This means that

<data pixel\_size="50nm">

<measurementset wavelength="193nm" NA="0.3">

<image defocus="100nm">100nm.tiff</image>

</measurementset>

</data>

This will result in one image that will have the attributes pixel\_size, wavelength, NA, and defocus.

Attributes can also have units specified. The unit is expanded into a multiplication by a scaling factor. The following units are available:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| k = 103 | m = 10-3 | u = 10-6 | n = 10-9 | p = 10-12 | f = 10-15 | % = 0.01 |
| km = 103 | mm = 10-3 | um = 10-6 | nm = 10-9 | pm = 10-12 | fm = 10-15 |  |
| deg = | rad = 1 | px = #pixel\_size | | pi = | f0 = 1/(#pixel\_size\*#size) | |

Sometimes the attribute parser will have difficulties parsing a particular line. In that case the value can start with “=” and the attribute will be evaluated as a matlab expression meaning text needs to be wrapped in single quotes. When the = is present units are also not substituted:

<image sigma="1.00">='IMO113715-141017-0027-0021-3.0um.tif'</image>

The problem with this line was that the “um” would get converted into a unit, which would cause the file to load incorrectly. To fix this the = was used.

# Variables

There are 3 kinds of variables available to be used in scripts.

1. @param – parameters – defined using <var> or specified during loading
2. #setting – attributes specified in a parent node using an attribute or <setting>
3. $file – variables loaded from the file specified by the attribute file

These variables can be used anywhere and can be used in mathematical expressions such as

exp(@alpha\*#wavelength/#defocus={5nm})

Variables can also have defaults specified. This means that anywhere you use a variable you can write

#defocus={value} or #defocus=value

Braces {} should be used if there is any ambiguity about where the value ends. Variables should not be nested in defaults.

A var tag can also contain image tags (and any image processing tags). In that case the value of the tag will be the stack of images returned.

# Scope

@parameters are scoped by id. Each tag is assigned a unique id and any parameters that exist or are defined within that tag are scoped to only exist within that tag. This means that if you, for example, have two measurementset tags and you use a var tag inside the first. That var will not be accessible from the second tag. You can force the tags to share @ variables by giving each measurementset the same id. The root automatically has the id global, so if you give any tag the id global it will share its variables with all other tags.

# Reserved Words

Certain words have special meaning and you should avoid using them for your own attributes because you could potentially run into conflicts. Nothing technically prevents you from using them for other than their intended purpose.

name, id, size, pixel\_size, index, value, image, x, y, offset\_x, offset\_y, NA, type, file, path, radius

There are more, but these are some of the most important.

# Caching

measurementset can have the cache variable set. This is the name of a file to cache the results of the measurementset in. The file will by default be put in the directory specified by path. This can significantly speed up loading, but be careful with parameterized xml files. You can use the function DataHash to hash the parameters and have the cache file depend on the parameter values. An additional parameter nocache can be specified to disable caching. Since this parameter can be inherited this is a way to disable caching using if statements.

|  |
| --- |
| **Caching** |
| <measurementset cache="cache.mat" nocache="0">  ...  </measurementset> |
| <measurementset cache="sprintf('%s', DataHash(struct('d',#defocus\_step,'o',#offset)))">  ...  </measurementset> |

# Generating Objects

|  |
| --- |
| **Creating Objects** |
| <object name="" type="speckle" size="" pixel\_size="" rms="" correlation\_length="" option="real" seed="" slope\_1="0" slope\_2="-inf" oversample="1" /> |
| <object name="" type="gaussian" size="" pixel\_size="" sigma="" integrated="" offset\_x="0" offset\_y="0" /> |
| <object name="" type="gaussian" size="" pixel\_size="" sigma\_x="" sigma\_y="" peak="" /> |
| <object name="" imag="" real="" /> |
| <object name="" amplitude="" phase="" /> |
| <object name="" field="" /> |
| <object name="" type="frequency" size="" pixel\_size="" f="" amplitude="1" phase="0" option="sin" /> |
| <object name="" type="planewave" size="" pixel\_size="" theta="0deg" /> |
| <object name="" type="aberration" pixel\_size="" NA="" wavelength="" field="@..." astigmatism\_90="1" … /> |

There are several ways to generate a mask. The result is an NxN matrix of dimension size. This mask is available through parameters using the name or to any children as object. Object tags do not pass any attributes on to their children, so the parameter struct returned will not have this information. They will pass the object field on to their children.

The frequency option generates a single frequency. phase is the phase of the frequency. Option can have values sin, cos, exp, or rect.

All objects that generate an object can provide an oversample option. This must be a positive integer. It causes each pixel in the final image to be split into the specified number of sub pixels for the function to be evaluated on. The final is then averaged over those subpixels to get back the final pixel.

## Generating a Speckle Field

It is possible to generate a random mask with a random, rough field. The resulting field will be real if option="real" or complex. There is no imaginary option. The seed is a positive integer used as the seed for the random number generator. If it is provided the mask will be the same each time the file is loaded, if not specified the mask will be different each time the file is loaded. The following attributes are required: type="speckle", size, rms, and correlation\_length.

## Generating a Gaussian Object

It is possible to generate a Gaussian object. The Gaussian can have separate sigma for the x and y direction. The sigma can be specified either using sigma\_x and sigma\_y or just sigma. The height of the Gaussian can be specified either using the peak value (peak) or the integrated volume (integrated). In addition to these parameters you can specify the x and y offsets (offset\_x, offset\_y). By default the Gaussian is placed at the center of the image. For images with an even size the peak will be at ceil(#size/2).

## Applying Aberrations

You can use the object type aberration to apply an aberration to the object. This uses the same code as the zernike tag, so any aberrations available there can be used. You must specify the NA and wavelength since that defines the circle on which the aberrations are defined.

# Propagating Objects

Once a field is defined you can image it under different conditions. Inside the object create a propagate tag to image the field under those conditions. Settings are inherited down from all enclosing environments, but specified fields can be overwritten. Alternatively, the object to propagate can be specified explicitly using a variable using the object attribute.

|  |
| --- |
| **Propagating Objects** |
| <object **…**>  <propagate …/>  …  </object> |
| <propagate object="" /> |

If the defocus attribute is an array, the object is imaged to each focus resulting in multiple images. This is only supported for the defocus tag. Propagating requires illumination and pupil. Illumination can be incoherent to do incoherent imaging.

# Saving Variables

You can save most variables to a file using a save structure. This includes masks and images. Illuminations can’t be saved to a file. Remember that for a variable to be saved it must be in scope. A variable defined in a parent is in scope, but if you have an <object> in a <measurementset> the <save> must be in the measurement set as well.

|  |
| --- |
| **Saving Variables** |
| <save file="mask.2dm">  <value name="msk">@mask</value>  <value name="pitch">1000</value>  </save> |
| <save file="image.png">  <value name="img">@img</value>  <value name="min">0</value>  <value name="max">1</value>  </save> |

If the filename is an image then the file will be saved as an image. The first value that is a 2D matrix (that could be an <object> or an image from <propagate> or <image>) will be saved. Additional fields “min” and “max” can be provided that result in scaling of the image. If the file type is not an image it is saved as a “.mat” file with “-v6”. Each value is stored as a field with the name given and the value provided.

# Conditionals

It is possible to evaluate some statements based on variables. This can make certain types of parameters much easier to implement.

|  |
| --- |
| **If Statement** |
| <if true="@phase\_mask={0}">  <object name="mask" phase="@phase\_roughness" />  </if> |
| <if false="@phase\_mask={0}">  <object name="mask" amplitude="@phase\_roughness" />  </if> |

The if statement can have either the true or false option set. No other parameters are passed on to children.

|  |
| --- |
| **For Statement** |
| <for sigma="[0.1,0.3,0.5]">  <propagate />  </for> |

The for statement can have any number of attributes that are arrays. It will repeat the contents for each value of the attribute. The example above will propagate the object using the three different values of sigma. You can have multiple attributes, but they must have the same length.

# Illumination / Pupil / Shape

Illuminations can be defined in the script directly using a composition of shapes.

|  |  |
| --- | --- |
| **Illumination / Pupil / Shape** | |
| <illumination name="">  …  </illumination> | <pupil name="">  <zernike tilt="" coma\_x="" … />  …  </pupil> |
| <circle radius="" x="" y="" value="=@(fx,fy)..." mode="intersect" /> | |
| <arc radius="" x="" y="" value="=@(fx,fy)..." mode="intersect" start="" end="" /> | |

Illumination and pupil use similar syntax. In each case you compose using shapes. The space on which the shapes are defined is the -1 to 1 with the pupil having a radius of NA. The illumination or pupil is a function @(fx,fy) defined on the frequency space. It is defined on the space of sin(theta). So the pupil would be a circle of radius NA in this space. They can be composed of any number of shapes. Shapes available are circle and arc. For the arc, start and end angles are defined relative to the positive x-axis and are in radians.

Shapes are composed together in order depending on their modes:

1. union (default) – takes an OR of the previous value and the shape, this will result in a binary output
2. add – add the shape to the previous shapes, this can be used with functions
3. subtract – subtract the shape from previous shapes
4. intersect – keep only the intersection of the current shape and the total of the previous shapes
5. multiply – this will multiply everything inside the shape by value, or the function value evaluated at each point.
6. xor – treats the value and the existing value as a binary value ( >0 ) and applies an xor between them.
7. set – sets the value of the shape to the set value within the valid range – so within the arc or circle.

Illuminations are functions, so they treat variables a bit differently. Any variable (parameter, setting, or file variable) are evaluated from the context where the function is used. This means that you can use settings in the illumination that are defined where the illumination is used:

<illumination name="monopole" sigma="@sigma">

<circle radius="#sigma\*#NA" /> **🡨 This would ordinarily cause an error**

</illumination>

<measurementset wavelength="193nm" NA="@NA=0.4" illumination="@monopole">

<propagate object="@mask" defocus="100nm" />

</measurementset>

<pupil name="zernike\_pupil">

<circle radius="#NA" ↓ **This defines a function**

value="=@(fx,fy)exp(1i\*pi\*#defocus/#wavelength\*(fx.^2+fy.^2))" />

<circle radius="0.3\*#NA" value="0.5\*exp(1i\*pi/2)" mode="multiply" />

</pupil>

If a binary pupil is not sufficient you can provide a value. This can be either a single value or a function evaluated inside the shape. When using this only add and multiply are valid operations.

Aberrations can be added using the zernike tag. In addition to defocus the following are also available: 'piston', 'tilt', 'tip', 'astigmatism\_45', 'astigmatism\_90', 'coma\_y', 'coma\_x', 'trefoil1', 'trefoil2', 'astigmatism\_secondary\_45', 'astigmatism\_secondary\_90', 'primary\_spherical'

# Adding Noise

It is possible to add noise to images or variables in a measurementset. This is done by adding one or more noise tags into a measurement set. The noise tags are applied to all images and parameters in the measurementset in the order that the noise tags appear.

<measurementset wavelength="193nm" NA="0.6">

<image file="10nm.jpg" defocus="10nm" />

<noise type="poisson" var="images" />

<noise type="gaussian" var="images" std="0.1" />

</measurementset>

|  |
| --- |
| **Noise** |
| <noise type="gaussian" var="" seed="" std="" variance="" mean="" /> |
| <noise type="poisson" var="" seed="" /> |
| <noise type="salt-pepper" var="" seed="" density="" /> |
| <noise type="speckle" var="" seed="" std="" variance="" /> |

var is the variable to add noise to. If var="images" then the noise is added to the images, otherwise it is added to the parameter for each image. If var is not specified it defaults to images. seed is the random number seed. If set to a positive integer then the noise added will be the same during each loading of the file. Otherwise the noise will be different each time the file is loaded.

For Gaussian and speckle noise the use of variance or standard deviation is optional. The noise generated is the same as that produced by **imnoise**. The image processing toolbox is needed for poisson, salt-pepper, and gaussian noise with a non-uniform variance.

# Modifying Images and Params

Sometimes, particularly when loading data from files, you may want to modify the images after they have been loaded.

|  |
| --- |
| **Set Param** |
| <setparam name="" value="">  ...  </setparam> |

setparam will, for each image inside it, set the parameter of the given name to the value specified. The value is loaded from the context of the image, so it can reference settings specified to the image.

A few special settings are available within value. You may use #index, #image. #index is an integer specifying which image is being modified, and #image is the corresponding image.

No attributes of assignparam are inherited by children.

|  |
| --- |
| **Image Crop** |
| <imagecrop size="128" offset\_x="10nm" offset\_y="10nm" subpixel="on">  ...  </imagecrop> |

It is possible to crop images to a smaller size. All images inside the imagecrop tag must be the same size, but they are then cropped to size. The area is, by default, the center of the image. If offsets are specified the area is moved. The offset is specified in distance, not pixels (so use px as the unit if you want it in pixels), and performs a subpixel shift using fourier transforms when necessary if subpixel="on". subpixel can also have values of x, y, or off to restrict subpixel shifting to only one direction or to disable it.

The offset values are taken from the image param, so they can be specified in a child and can reference settings present in the param.

|  |
| --- |
| **Resample – Crop in Fourier Space** |
| <resample power="0">  ...  </resample> |

If an image is highly band-limited then it may be a waste of computation to process the entire image. It is then possible to resample the image. A negative power will downsample by that many powers of two. So a power="-2" would result in an image with ¼ the number of pixels but the same dimension. Similarly positive powers effectively pad in fourier space. A power="2" would result in an image with 4x the number of pixels and the same dimension – so the pixel has been shrunk by a factor of 4.

|  |
| --- |
| **Image Pad** |
| <imagepad size="" value="border" offset\_x="" offset\_y="">  ...  </imagepad> |

This will pad the image out to the specified size. The image will be centered at offset\_x and offset\_y. The padding will have the value specified by value. If value is border then the border value is based on the average of the border of the image. offset\_x, offset\_y, and value are evaluated from the context of the image and have access to #index and #image.

|  |
| --- |
| **Remove Outliers** |
| <removeoutliers mode="cutoff" cutoff="0.2" inpaintmethod="">  ...  </removeoutliers>  <removeoutliers mode="outliers" outliers="5%" inpaintmethod=""> ... </removeoutliers> |

Remove Outliers takes any pixels that are larger than mean \* (1+cutoff) or less than mean\*(1-cutoff) and removes them. It then fills them back in using inpaint\_nans. The inpaintmethod attribute is passed to inpain\_nans and determines what method to use. I recommend not specifying it, the loader will select the best. Alternatively you can use an outlier detection method. The outlier method will not remove outliers if it doesn’t find anything way out there, so it may not give the desired results.

|  |
| --- |
| **Normalize Background** |
| <normalizebackground mode="polynomial" bgorders="-1" mask="[]">  ...  </normalizebackground>  <normalizebackground mode="dividemean"></normalizebackground>  <normalizebackground mode="min"></normalizebackground>  <normalizebackground mode="max"></normalizebackground>  <normalizebackground mode="range"></normalizebackground> |

This will normalize the background of all the contained images. If the mode is dividemean then each pixel in the image is divided by the mean of all copies of that pixel in the stack. If the mode is polynomial then a polynomial of order bgorders is fit to each image and the image is divided by that. Only positive orders will cause anything to be done. Note that for order 0 you are normalizing the mean to 1.

An optional mask can be provided. This is a binary mask that says which pixels to use when fitting the background. The extra variables #image and #index are available, so you can create a mask such as mask="#image > 1" to only fit the background on part of the image. The value of the mask can be either a binary image or a function @(x,y) that will return the binary image. The function will be called with coordinates ranging [-1, 1]. You can also use a shape, pupil, or illumination.

min sets the minimum to 0. max sets the maximum to 1. range performs both the max and min operations.

|  |
| --- |
| **Filter** |
| <filter type="bandreject" fmin="0" fmax="inf" pixel\_size="" wh\_ratio="1">  ...  </filter>  <filter type="median" neighborhood="2">  ...  </filter> |

This will filter any contained images. Options for type are bandreject and bandpass. The minimum and maximum frequencies are in spatial frequencies, which is why pixel\_size is required. Note that f0 is a unit available. This will correspond to one pixel in frequency space. The wh\_ratio allows you to filter out an oval in frequency space. A number larger than one will make a wide ellipse. A number smaller than one will make a tall ellipse.

filter can also do median filtering using type set to median. The size of the neighborhood can then be provided. If it is a single scalar then a square neighborhood is used. Alternatively a 2 element vector can be provided. This feature uses medfilt2.

# Importing Settings and Images

|  |
| --- |
| **Import Settings** |
| <data import="@settings">  ...  </data> |

You can add the attribute import to any tag. The value of import should be a struct. All the fields of that struct are then merged into the settings of the field before any of the other fields or settings and parameters in that node are read.

|  |
| --- |
| **Import Images** |
| <import params="@params" images="@images" /> |

You can import images from a variable using the import tag. This is useful for scripts that don’t want to load or generate the images each time the file is loaded but do want to take advantage of the scripting capabilities of the file format.

# Including XML Files

|  |
| --- |
| **Include** |
| <include file="../algorithm.xml" /> |
| <include>XML DATA...</include> |

Sometimes it may be desirable to split data into several xml files. By using the include tag you can load a separate xml file. The settings, parameters, and images are merged. Be careful about variables being overwritten. It is probably best to put the includes at the top or bottom of the file.

The second method requires the content to be parsed (by the xml parser) as a string. This means if you just put xml there it will not work. It is intended to load xml out of variables.

# Removing Images

|  |
| --- |
| **Select** |
| <select index="(-1:1)+round(#N/2)">  ...  </select> |

Sometimes when you are including another file you only want to keep some of the images. The select tag lets you do that. It has a special parameters index which is a list of index values to keep. The special values #N and #images are available giving the number of images and the 3D matrix of all the images, respectively.