Writing Components

(For a lightweight introduction to writing a McStas component you may also have a look at http://ven2010.essworkshop.org/storage/Writing_a_component/Writing%20a%20component%20large.m4v from an old ESS workshop webpage)

First, let us discuss the overall structure of a McStas component, illustrated by the one of the simplest components, Slit.comp. The component realises an aperture or slit, in this case either a rectangular or circular opening in an absorbing material slab.

First we will have a look at the documentation header below. The various '"%-tags"' are parsed by mcdoc to define component online documentation like this http://mcstas.org/download/components/optics/Slit.html

The first part including the '"%I"' information tag signifies who wrote the component and where.

```
/*******************************************************************************
*
* McStas, neutron ray-tracing package
* Copyright 1997-2002, All rights reserved
* Risoe National Laboratory, Roskilde, Denmark
* Institut Laue Langevin, Grenoble, France
*
* Component: Slit
*
* %I
* Written by: Kim Lefmann and Henrik M. Roennow
* Date: June 16, 1997
* Version: $Revision: 4382 $
* Origin: Risoe
* Release: McStas 1.6
*
</pre>
 Next, some descriptive text is added, including the '''%D''' description tag.
<pre>
* Rectangular/circular slit with optional insignificance cut
*
* %D
* A simple rectangular or circular slit. You may either
* specify the radius (circular shape), or the rectangular bounds.
* No transmission around the slit is allowed.
* If cutting option is used, low-weight neutron rays are ABSORBED
*
* Example: Slit(xmin=-0.01, xmax=0.01, ymin=-0.01, ymax=0.01)
* Slit(radius=0.01, cut=1e-10)
*
</pre>
The '''%P''' tag indicates component input parameters, defining e.g. physical properties and geometry of the
component.
<pre>
* %P
* INPUT PARAMETERS
*
* radius: Radius of slit in the z=0 plane, centered at Origo (m)
* xmin: Lower x bound (m)
* xmax: Upper x bound (m)
* ymin: Lower y bound (m)
* ymax: Upper y bound (m)
* xwidth: Width of slit. Overrides xmin,xmax. (m)
* yheight:Height of slit. Overrides ymin,ymax. (m)
*
* Optional parameters:
* cut: Lower limit for allowed weight (1)
*
* %E
*******************************************************************************/
```

The optional '"%L"' tag can be used for attaching links to other information - and '"%E"' ends the commentary section.

Next follows some important lines of code that are mandatory for components:

```
 DEFINE COMPONENT Slit
```

Defines the component type (here Slit) - and must match the filename under which the component is stored, i.e. Slit.comp

```
 DEFINITION PARAMETERS ()
```

Definition parameters defines a list (here empty) of input parameters of "special" type, i.e. not basic c-types like double or int - can e.g. be used for strings (<code>string filename="blahblah"</code>), lists or pointers.

```
 SETTING PARAMETERS (xmin=-0.01, xmax=0.01, ymin=-0.01, ymax=0.01, radius=0,
 cut=0, xwidth=0, yheight=0)
```

Setting parameters are of basic c-type, defaulting to being doubles but can be cast as integers. All definition and setting parameters are prefixed in the generated c-code by the component instance name.

```
 OUTPUT PARAMETERS ()
```

Output parameters defines other c-variables than those listed above that should be "protected" i.e. prefixed by the component instance name, e.g. calculated values that are dependent on the instance parameters. The variables that can be protected here are those defined in the optional DECLARE block

```
DECLARE %{
%}
```

Declare is used to define internal calculation variables and internal component functions. Bear in mind to put these as output parameters if they are to be "private" to the component.

```
SHARE %{
%}
```

Share is very much like the Declare block, but is used for code that should be shared for components of the same type - and is only included once in the generated c-code.

```
INITIALIZE
%{
 if (xwidth > 0) { xmax=xwidth/2; xmin=-xmax; }
 if (yheight > 0) { ymax=yheight/2; ymin=-ymax; }
 if (xmin == 0 && xmax == 0 && ymin == 0 && ymax == 0 && radius == 0)
 { fprintf(stderr,"Slit: %s: Error: give geometry\n", NAME_CURRENT_COMP); exit(-1); }
%}
```

Initialize is used to calculate values for the declared variables, make checks of input variable consistency etc. This is where runtime-errors due to unmeaningful parameters should be placed.

```
TRACE
%{
 PROP_Z0;
 if (((radius == 0) && (x<xmin || x>xmax || y<ymin || y>ymax))
 || ((radius != 0) && (x*x + y*y > radius*radius)))
 ABSORB;
 else
 if (p < cut)
 ABSORB;
 else
 SCATTER;
%}
```

The Trace section is where the fun begins! 🙂 An incoming neutron (always with parameters x,y,z,vx,vy,vz,sx,sy,sz,t,p) is typically checked for intersection with the object - here after propagation by PROP_Z0. Other possibilities for intersection include e.g. the **cylinder_intersect** routine and similar, that calculate intersection times between the object and the neutron equation of motion (including gravitation if this is enabled).

After propagation something usually "happens" here, i.e. the neutron is scattered, changes direction, is absorbed or equivalent, and a new set of values for the neutron parameters are set accordingly.

```
SAVE %{
%}
```

Save is an optional section - typically used by monitors for calling the DETECTOR_OUT macros for storing histograms to disk.

```
FINALLY %{
%}
```

Finally is an optional section that allows to do e.g. cleanups at the end of execution.

```
MCDISPLAY
%{
 magnify("xy");
 if (radius == 0) {
 double xw, yh;
 xw = (xmax - xmin)/2.0;
 yh = (ymax - ymin)/2.0;
 multiline(3, xmin-xw, (double)ymax, 0.0,
 (double)xmin, (double)ymax, 0.0,
 (double)xmin, ymax+yh, 0.0);
 multiline(3, xmax+xw, (double)ymax, 0.0,
 (double)xmax, (double)ymax, 0.0,
 (double)xmax, ymax+yh, 0.0);
 multiline(3, xmin-xw, (double)ymin, 0.0,
 (double)xmin, (double)ymin, 0.0,
 (double)xmin, ymin-yh, 0.0);
 multiline(3, xmax+xw, (double)ymin, 0.0,
 (double)xmax, (double)ymin, 0.0,
 (double)xmax, ymin-yh, 0.0);
 } else {
 circle("xy",0,0,0,radius);
 }
%}
```

Mcdisplay is used to render the object on a graphics display, by means of simple linear outlines of the object.

```
END
```

The END keyword marks the end of the component code.

== Additional information ==

- See the McStas manual section 5.5 for instructions on how-to write components