Insertion of Module polarizer_sm to the module list

```
### neededModulesSET
###      (generated from AvailableSET)
### 0 name of module
### 1 selection of {active disabled simulated unused}
### 2 list of names of control variable lists for that module
###   (or "" if no control vars)
### 3 if not "", name of callback procedure
###   to set initial module constants
### 4 special procedure used to set control variables
###   (generateEntries is always possible)
### 5 if not "", name of help item for that module

proc makeModuleSets {} {
  global AvailableSET
  # 0 name of module categorie
  # 1 list of submodules; may be empty
  # 2 help item; may be a list if different submodules have different help
   set AvailableSET {
     {source {source_cws source_spss source_lpss} source}
     {guide {} guide}
     {spacewindow {spacewindow spacewindow_multiple}
       {spacewindow spacewindo_multiple}}
     {chopper {chopper_disc chopper_fermistr} {chopper chopper_fermistr}}
     {velselect {} velselect}
     {collimator_soller {} collimator}
     {monochr_analyser {ma_flat ma_focus ma_focus_dat} monochr_analyser}
      {polariser {polariser_he3 polariser_sm} {polariser_he3 polariser_sm}}
     {flipper_coil {} flipper_coil}
     {precessionfield {} precessionfield}
     {sample {sample_elasticisotr sample_inelast sample_powder
       sample_reflectom sample_sans} {sample_elasticisotr sample_inelast
        sample_powder sample_reflectom sample_sans}
     }
…
```

Definition of parameters for polarizer_sm

```
gSet polariser_smESET {
  {pfile pareditablefile polariser_SM.par
    {"parameter\nfile" "" "" P} w pol 1}
  {ufile pareditablefile mirr3+.dat {"Up-reflectivity\nfile" "reflectivity
data file for Up neutrons" "" U}}
  {dfile pareditablefile mirr1a.dat {"Down-reflectivit\nfile" "reflectivity
data file for Down neutrons" "" D}}
  {"" header}
  {x float 100 {"position\nX [cm]" "x center position of the rectangular
geometry polariser" "" a}}
  {y float 0   {"position\nY [cm]" "y center position of the rectangular
geometry polariser" "" b}}
  {z float 0   {"position\nZ [cm]" "z center position of the rectangular
geometry polariser" "" c}}

…
```

another parameter set definition, now for module frame

```
### Frame
###
gSet frameESET {
  {Transformation header}
  {seq radio RTM {sequence "sequence of Rotation, Translation, and Mirroring" "" S}
    {RTM RMT TRM TMR MTR MRT} {1 2 3 4 5 6}}
  {Rotation header}
  {rotz float 0 {"rot. angle [deg]\naround z axis" "rotation angle around Z FIRST rotation
[deg]\nNOTE: 90deg means X+  rotated on Y+" "" H} 1}
  {roty float 0 {"rot. angle [deg]\naround y axis" "rotation angle around Y SECOND rotation
[deg]\nNOTE: 90deg means Z+ rotated on X+" "" V} 1}
  {rotx float 0 {"rot. angle [deg]\naround x axis" "rotation angle around X (beam axis) THIRD
rotation [deg]\nNOTE: 90deg means Y+  rotated on Z+" "" A} 1}
  {Translation header}
  {tx float 0 {"x [cm]" "x component of translation vector [cm]" "" x} 1}
  {ty float 0 {"y [cm]" "y component of translation vector [cm]" "" y} 1}
  {tz float 0 {"z [cm]" "z component of translation vector [cm]" "" z} 1}
  {Mirror header}
  {mx radio no {"mirror at\nyz plane" "mirror x axis (plane yz)" "" i} {no yes} {0 1}}
  {my radio no {"mirror at\nxz plane" "mirror y axis (plane xz)" "" j} {no yes} {0 1}}
  {mz radio no {"mirror at\nxy plane" "mirror z axis (plane xy)" "" k} {no yes} {0 1}}
}

…
```

Definition of parameters in *.pol files

```
gSet polESET {
  {dx float 60 {"dimension\nX [cm]" "length of the polariser" "" } gt0}
  {dy float 10 {"dimension\nY [cm]" "width of the polariser" "" } gt0}
  {dz float 10 {"dimension\nZ [cm]" "height of the polariser" "" } gt0}
  {nc int 9    {"number of\nchannels" "number of channels in vertical direction" "" } ge1}
  {dw float 0.05 {"wall\nwidth [cm" "width of the wall between the channels" "" } gt0}
  {cp float 0 {"cutoff\nprobability" "minimal probability weight transmitted" "" } ge0}
  {gx float 1 {"guide field\nX [Oe]" "x component of the guide field" "" }}
  {gy float 0 {"guide field\nY [Oe]" "y component of the guide field" "" }}
  {gz float 0 {"guide field\nZ [Oe]" "z component of the guide field" "" }}
  {ax float 1 {"analysis dir.\nX [-]" "x direction vector component of the quantization
direction" "" }}
  {ay float 0 {"analysis dir.\nY [-]" "y direction vector component of the quantization
direction" "" }}
  {az float 0 {"analysis dir.\nZ [-]" "z direction vector component of the quantization
direction" "" }}
}
…
```

necessary changes of edit/save routines for *.pol files

```
proc editFile {var param ext app} {
  # Edit parameters of a module, which are separated in
  #  a parameter file with extension ext.
  #  $var$app is the name of the parameter file.
  #  param is a parameter for editSave.
  # This GUI generator relies on serialize${ee}File
  #  (where $ee is capitalized $ext) to read a file
  #  and editSave to store the results.
  …
  switch $ext {
    chp - crs - ine - ref - san - pow - pol - iso {set special 1}
    default {
      …
    }
  }
  …
  if $special {
    switch $ext {
     iso {serializeIsoFile $f r $var $app}
     pol {serializePolFile $f r $var $app}
     chp {serializeChpFile $f r $var $app}
     crs {serializeCrsFile $f r $var $app}
     ine {serializeIneFile $f r $var $app}
     ref {serializeRefFile $f r $var $app}
     san - pow {serializeSamFile $f r $var $app $ext}
    }
    generateEntries $w.v ${ext}ESET {} $app
  } elseif {$f != "0"} {
    while {[gets $f line] >= 0} {$w.v.text insert end "$line\n"}
  }
…

proc editSave {var param ext app {saveAs 0}} {
…
  catch {
    switch $ext {
     chp {serializeChpFile $f w $var $app}
     crs {serializeCrsFile $f w $var $app}
     san - pow {serializeSamFile $f w $var $app $ext}
     ine {serializeIneFile $f w $var $app}
     ref {serializeRefFile $f w $var $app}
     iso {serializeIsoFile $f w $var $app}
     pol {serializePolFile $f w $var $app}
     default {puts $f [$w.v.text get 1.0 end]}
    }
  }
…
```

Read/write *.pol-Files

```
proc serializePolFile {f mode var app} {
  set nlist {dx dy dz nc dw cp gx gy gz ax ay az}
  foreach l $nlist {
    upvar #0 $l$app $l
  }
  if {$mode == "r"} {
    foreach l $nlist { catch {unset $l}}
    if {$f == "0"} return
    readNumItems $f $nlist $app
  } else {
    puts $f "$dx $dy $dz\n$nc $dw $cp\n$gx $gy $gz\n$ax $ay $az"
  }
}
```

Necessary changes, if exe file for module xyz isn't xyz.exe
(here e.g. source_cws and source.exe with parameters)

```
### compose the VITESS command pipe string
###
proc generateVitessCommand {} {
  …
  for {set i 1} {$i <= $maxModule} {incr i} {
    …
    switch $var {
       source_cws  {set com "source$sys -S1"}
       source_spss {set com "source$sys -S2"}
       source_lpss {set com "source$sys -S3"}
      ma_flat       {set com "monochr_analyser$sys -O1"}
      ma_focus      {set com "monochr_analyser$sys -O2"}
…
```

Changes in tools.tcl for *.pol files

```
proc fileDialog {operation {ext ""} {ifile Untitled}} {
  #   Type names          Extension(s)  Mac File Type(s)
  #---------------------------------------------------------
  set types {
    {"All files"           {*}}
    {"X,Y ASCII files"          {.dat}}
    {"2 D Intensity files" {.out}}
    {"chopper files"            {.chp .par .dat}}
    {"crystal description"      {.crs .par .dat}}
    {"powder sample description" {.pow .par .dat}}
    {"sans sample description"   {.san .par .dat}}
    {"sample reflecometer description"   {.ref .dat}}
    {"inelastic sample description" {.ine .par .dat}}
    {"elastic isotropic sample description" {.iso .par .dat}}
   {"polarizer sm description" {.pol .par .dat}}
    {"GUI settings"             {.gui}}
    {"Batch command files"      {.bat}}
    {"Tcl files"                {.tcl}}
  }
```

Structure of example.c

```
/*********************************************************/
/*  VITESS module example                                */
/*********************************************************/
#include "init.h"

int ParA;

/* own initialization of the monochromator/analyser module */
void OwnInit(int argc, char **argv)
{
  char *arg;
  int Option;
  fprintf (LogFilePtr," \n") ;
  print_module_name("example") ;

  argv++;                  /* jump over program name */
  while ((arg = *argv++)) {
    arg++;                 /* jump over - char  */
    switch (*arg++) {
    case 'A':
       sscanf(arg, "%d", &ParA); break;
       /* ... */
    }
  }
}

void OwnCleanup()
{
  /* do some final action, like logging. */
  fprintf(LogFilePtr,"\nsome logging");
}


int main(int argc, char **argv)
{
  Neutron output;
  int i;

  Init(argc, argv);
  OwnInit(argc, argv);

  DECLARE_SOFTABORT

  while(ReadNeutrons()!= 0)
    for(i=0; i<NumNeutGot; i++)
      {
      CHECK
      /* ... */
      WriteNeutron( &output);
      }

 soft_exit:

  OwnCleanup();
  Cleanup();

  return 0;
}
```

Structure of general.h

```
#ifndef GENERAL_H
#define GENERAL_H

#include <stdio.h>
#ifdef _MSC_VER
#include <fcntl.h>
#include <io.h>
#endif
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

#define BUFFER_SIZE     10000
#define CHAR_BUF_LENGTH 200

typedef double VectorType[3];
typedef double DoublePair[2];

typedef struct {
  double    Time;
  double        Wavelength;
  double        Probability;
  VectorType    Position;
  VectorType    Vector;
  VectorType    Spin;
} Neutron;


#endif
```

Structure of init.h

```
#ifndef INIT_H
#define INIT_H

#include "general.h"

extern long BufferSize;  /* size of the neutron input and ouput buffer */
extern Neutron *InputNeutrons;  /* input neutron Buffer */

extern Neutron *OutputNeutrons; /* output neutron buffer */
extern long OutNeutPtr;         /* points to the next free position in
OutputNeutrons */

extern long NumNeutGot;       /* number of neutrons read in the current
batch */
extern long NumNeutRead;      /* number of neutrons read in total */
extern long NumNeutWritten;   /* number of neutrons written in total */

extern FILE *InputFilePtr;    /* stream from which the neutrons are read */
extern FILE *OutputFilePtr;   /* stream to which the neutrons are written */
extern FILE *LogFilePtr;      /* stream to which things are logged*/
extern char *InputFileName;    /* file to read neutrons */
extern char *OutputFileName;  /* file to write neutrons */
extern char *LogFileName;     /* log filename  */

extern double Temp;           /* Temperature of the simulation */

extern long idum;             /* random number specific */

void OutputBufferFlush();
void Init(int argc, char **argv);
void Cleanup();
int ReadNeutrons();
void CopyNeutron(Neutron *source, Neutron *dest);
void WriteNeutron(Neutron *OutNeutron);
long parseline(char *Buffer, char *ptr[]);
long ParseFileLine(FILE *In, char *ptr[]);
long LinesInFile(FILE *In);
void ReadSourceData(double* p_pCurrent, double* p_pTimeMeas);
void WriteSourceData(double p_dCurrent, double p_dTimeMeas);
void print_module_name(char name[]);

#ifdef _MSC_VER
# define CHECK /* test semaphore and go my_exit if finished */
# define DECLARE_SOFTABORT /* windows ... */
#else
# include <signal.h>
  static int finish;
  void abort_handler(int sig);
# define CHECK if (finish) {goto soft_exit;}
# define DECLARE_SOFTABORT signal(SIGTERM, abort_handler);
#endif

#endif
```