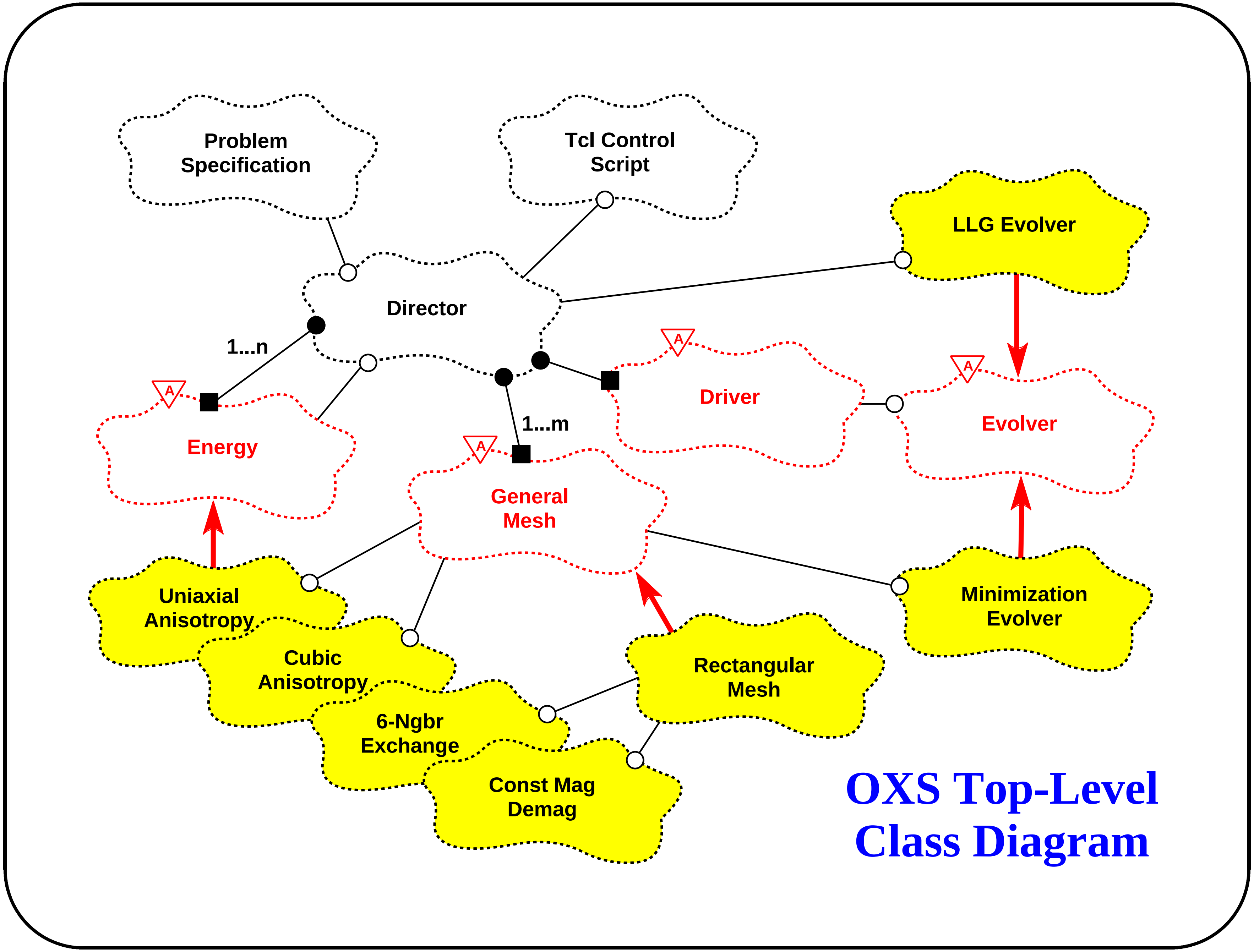
The
OOMMF
eXtensible
Solver


Problem Specification
Tcl Control Script
LLG Evolver
Director
1...n
1...m
A
Energy
Driver
Evolver
General Mesh
Uniaxial Anisotropy
Cubic Anisotropy
6-Ngbr Exchange
Const Mag Demag
Rectangular Mesh
Minimization Evolver
OXS Top-Level Class Diagram

Sample MIF 2.0 File

```
# MIF 2.0
set pi [expr 4*atan(1.0)]
set mu0 [expr 4*$pi*1e-7]
proc Skip { args } {}

Specify Oxs_RectangularRegion:World {
  xrange {0 310e-9}
  yrange {0 310e-9}
  zrange {0 40e-9}
}

Specify Oxs_CubicMesh {
  cellsize 10e-9
  region Oxs_RectangularRegion:World
}
```

```
proc UpDownSpin { x y z xmin ymin zmin
                        xmax ymax zmax } {
    if { $x < 0.55*$xmin + 0.45*$xmax } {
        return "0 1 0"
    } elseif { $x > 0.45*$xmin + 0.55*$xmax } {
        return "0 -1 0"
    } else {
        return "0 0 1"
    }
}

Specify Oxs_TSFVectorField:mupdown {
    mesh Oxs_CubicMesh
    script  UpDownSpin
    norm  1
}
```

```
Specify Oxs_UCExchange:NiFe {
  Ms 8e5
  A  13e-12
  mesh Oxs_CubicMesh
}

Specify Oxs_UZeeman "
  Ms 8e5
  Hscale [expr 0.001/$mu0]
  Hrange {
      0  0  0   50  0  0   2
  }
"
```

```
Specify Oxs_CubicDemag {
  mesh Oxs_CubicMesh
}

Specify Oxs_EulerEvolve {
  alpha 0.5
  start_dm 0.01
}

Specify Oxs_UniformFixedScalarField:Ms {
  value 8e5
}
```

```
Specify Oxs_BaseDriver {
 evolver Oxs_EulerEvolve
 min_timestep 1e-15
 max_timestep 10e-9
 stopping_dm_dt 1e5
 mesh Oxs_CubicMesh
 Ms Oxs_UniformFixedScalarField:Ms
 m0 Oxs_TSFVectorField:mupdown
 number_of_stages 0
 stage_iteration_limit 200
 total_iteration_limit 0
}
```

Adding an Energy Term to OXS in

THREE EASY STEPS

Adding an Energy Term to OXS: Step 1

1. Copy sample header (*.h) and C++ source (*.cc) files from oommf/app/oxs/ext to oommf/app/oxs/local

2. Make desired edits

3. Run pimake

NB: Modify no files from OOMMF distribution!

Sample Energy Header File

```
/* FILE: uuanisotropy.h            -*-Mode: c++-*-
 *
 * Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
 *
 */
#ifndef _OXS_UUANISOTROPY
#define _OXS_UUANISOTROPY

#include "nb.h"
#include "threevector.h"
#include "energy.h"
#include "depkey.h"
#include "key.h"
#include "simstate.h"
#include "mesh.h"
#include "meshvalue.h"
/* End includes */

class Oxs_UUAnisotropy:public Oxs_Energy {
private:
  REAL8m K1;
  REAL8m Ms;
  ThreeVector axis;
public:
  virtual const char* ClassName() const; // ClassName() is
  /// automatically generated by the OXS_EXT_REGISTER macro.
  virtual BOOL Init();
  Oxs_UUAnisotropy(const char* name,  // Child instance id
Oxs_Director* newdtr, // App director
                Tcl_Interp* safe_interp, // Safe interpreter
                const char* argstr);  // MIF input block parameters

  virtual ~Oxs_UUAnisotropy() {}

  virtual void GetEnergyAndField(const Oxs_SimState& state,
                                 Oxs_MeshValue<REAL8m>& energy,
                                 Oxs_MeshValue<ThreeVector>& field
                                 ) const;
};
#endif // _OXS_UUANISOTROPY
```

Sample Energy C++ Source

```
/* FILE: uuanisotropy.cc            -*-Mode: c++-*-
 *
 * Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
 *
 */

#include "oc.h"
#include "nb.h"
#include "threevector.h"
#include "director.h"
#include "simstate.h"
#include "ext.h"
#include "depkey.h"
#include "key.h"
#include "mesh.h"
#include "meshvalue.h"
#include "cubicmesh.h"
#include "uuanisotropy.h"
#include "energy.h" // Needed to make MSVC++ 5 happy

// Oxs_Ext registration support
OXS_EXT_REGISTER(Oxs_UUAnisotropy);

/* End includes */


// Constructor
Oxs_UUAnisotropy::Oxs_UUAnisotropy(
  const char* name,     // Child instance id
  Oxs_Director* newdtr, // App director
  Tcl_Interp* safe_interp, // Safe interpreter
  const char* argstr)   // MIF input block parameters
  : Oxs_Energy(name,newdtr,safe_interp,argstr)
{
  // Process arguments
  CheckInitValueParamCount("K1",1);
  K1=Nb_Atof((*FindInitValue("K1"))[0].c_str());
  DeleteInitValue("K1");

  CheckInitValueParamCount("Ms",1);
  Ms=Nb_Atof((*FindInitValue("Ms"))[0].c_str());
```

```
  DeleteInitValue("Ms");

  CheckInitValueParamCount("axis",3);
  axis.x=Nb_Atof((*FindInitValue("axis"))[0].c_str());
  axis.y=Nb_Atof((*FindInitValue("axis"))[1].c_str());
  axis.z=Nb_Atof((*FindInitValue("axis"))[2].c_str());
  DeleteInitValue("axis");

  VerifyAllInitArgsUsed();

  if(Ms==0.0) {
    K1=0.0; Ms=1.0; // Safety
  }

  REAL8m magsq=axis.MagSq();
  if(magsq==0.0) {
    string msg="Invalid MIF input block detected for object "
      + string(InstanceName())
      + ": Specified anisotropy axis is (0,0,0)";
    throw Oxs_Ext::Error(msg.c_str());
  }
  if(magsq!=1.0) axis *= 1.0/sqrt(magsq);
}

BOOL Oxs_UUAnisotropy::Init()
{ return 1; }

void Oxs_UUAnisotropy::GetEnergyAndField
(const Oxs_SimState& state,
 Oxs_MeshValue<REAL8m>& energy,
 Oxs_MeshValue<ThreeVector>& field
 ) const
{
  UINT4m size = state.mesh->Size();

  REAL8m energy_mult = -K1;
  REAL8m field_mult = 2*K1/(Ms*MU0);
  for(UINT4m i=0;i<size;++i) {
    REAL8m temp = axis*state.spin[i];
    energy[i] = energy_mult*temp*temp;
    field[i] = (field_mult*temp)*axis;
  }

}
```

Modified Energy Header File

```
/* FILE: myanisotropy.h            -*-Mode: c++-*-
 *
 * My Cubic Anisotropy, derived from Oxs_Energy class.
 *
 */
#ifndef _MY_ANISOTROPY
#define _MY_ANISOTROPY

#include "nb.h"
#include "threevector.h"
#include "energy.h"
#include "depkey.h"
#include "key.h"
#include "simstate.h"
#include "mesh.h"
#include "meshvalue.h"
/* End includes */

class My_Anisotropy:public Oxs_Energy {
private:
  REAL8m K1;
  REAL8m Ms;
  ThreeVector axis1;
  ThreeVector axis2;
public:
  virtual const char* ClassName() const; // ClassName() is
  /// automatically generated by the OXS_EXT_REGISTER macro.
  virtual BOOL Init();
  My_Anisotropy(const char* name,  // Child instance id
Oxs_Director* newdtr, // App director
                Tcl_Interp* safe_interp, // Safe interpreter
                const char* argstr);  // MIF input block parameters

  virtual ~My_Anisotropy() {}

  virtual void GetEnergyAndField(const Oxs_SimState& state,
                                 Oxs_MeshValue<REAL8m>& energy,
                                 Oxs_MeshValue<ThreeVector>& field
                                 ) const;
};
#endif // _MY_ANISOTROPY
```

Modified Energy C++ Source

```
/* FILE: myanisotropy.cc            -*-Mode: c++-*-
 *
 * My Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
 *
 */

#include "oc.h"
#include "nb.h"
#include "threevector.h"
#include "director.h"
#include "simstate.h"
#include "ext.h"
#include "depkey.h"
#include "key.h"
#include "mesh.h"
#include "meshvalue.h"
#include "cubicmesh.h"
#include "myanisotropy.h"
#include "energy.h" // Needed to make MSVC++ 5 happy

// Oxs_Ext registration support
OXS_EXT_REGISTER(My_Anisotropy);

/* End includes */


// Constructor
My_Anisotropy::My_Anisotropy(
  const char* name,     // Child instance id
  Oxs_Director* newdtr, // App director
  Tcl_Interp* safe_interp, // Safe interpreter
  const char* argstr)   // MIF input block parameters
  : Oxs_Energy(name,newdtr,safe_interp,argstr)
{
  // Process arguments
  CheckInitValueParamCount("K1",1);
  K1=Nb_Atof((*FindInitValue("K1"))[0].c_str());
  DeleteInitValue("K1");

  CheckInitValueParamCount("Ms",1);
  Ms=Nb_Atof((*FindInitValue("Ms"))[0].c_str());
```

```
DeleteInitValue("Ms");

CheckInitValueParamCount("axis1",3);
axis1.x=Nb_Atof((*FindInitValue("axis1"))[0].c_str());
axis1.y=Nb_Atof((*FindInitValue("axis1"))[1].c_str());
axis1.z=Nb_Atof((*FindInitValue("axis1"))[2].c_str());
DeleteInitValue("axis1");

CheckInitValueParamCount("axis2",3);
axis2.x=Nb_Atof((*FindInitValue("axis2"))[0].c_str());
axis2.y=Nb_Atof((*FindInitValue("axis2"))[1].c_str());
axis2.z=Nb_Atof((*FindInitValue("axis2"))[2].c_str());
DeleteInitValue("axis2");

VerifyAllInitArgsUsed();

if(Ms==0.0) {
  K1=0.0; Ms=1.0; // Safety
}

REAL8m magsq=axis1.MagSq();
if(magsq==0.0) {
  string msg="Invalid MIF input block detected for object "
    + string(InstanceName())
    + ": Specified anisotropy axis 1 is (0,0,0)";
  throw Oxs_Ext::Error(msg.c_str());
}
if(magsq!=1.0) axis1 *= 1.0/sqrt(magsq);

magsq=axis2.MagSq();
if(magsq==0.0) {
  string msg="Invalid MIF input block detected for object "
    + string(InstanceName())
    + ": Specified anisotropy axis 2 is (0,0,0)";
  throw Oxs_Ext::Error(msg.c_str());
}
if(magsq!=1.0) axis2 *= 1.0/sqrt(magsq);

if(fabs(axis1*axis2)>1e-12) {
  string msg="Invalid MIF input block detected for object "
    + string(InstanceName())
    + ": Specified anisotropy axes aren't perpendicular";
  throw Oxs_Ext::Error(msg.c_str());
}
```

```
}

BOOL My_Anisotropy::Init()
{ return 1; }

void My_Anisotropy::GetEnergyAndField
(const Oxs_SimState& state,
 Oxs_MeshValue<REAL8m>& energy,
 Oxs_MeshValue<ThreeVector>& field
 ) const
{
  UINT4m size = state.mesh->Size();
  ThreeVector axis3 = axis1 ^ axis2;
  axis3.SetMag(1.0); // Just to be safe

  REAL8m field_mult = -2*K1/(Ms*MU0);
  for(UINT4m i=0;i<size;++i) {
    REAL8m a = axis1*state.spin[i];
    REAL8m b = axis2*state.spin[i];
    REAL8m c = axis3*state.spin[i];
    energy[i] = K1 * (a*a*b*b+a*a*c*c+b*b*c*c);
    field[i]  = (a*(b*b+c*c))*axis1;
    field[i] += (b*(a*a+c*c))*axis2;
    field[i] += (c*(a*a+b*b))*axis3;
    field[i] *= field_mult;
  }

}
```

Header File Diffs

```
-/* FILE: uuanisotropy.h            -*-Mode: c++-*-
+/* FILE: myanisotropy.h            -*-Mode: c++-*-
  *
- * Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
+ * My Cubic Anisotropy, derived from Oxs_Energy class.
  *
  */

-#ifndef _OXS_UUANISOTROPY
-#define _OXS_UUANISOTROPY
+#ifndef _MY_ANISOTROPY
+#define _MY_ANISOTROPY

 #include "nb.h"
 #include "threevector.h"
 #include "energy.h"
 #include "depkey.h"
 #include "key.h"
 #include "simstate.h"
```

```
 #include "mesh.h"
 #include "meshvalue.h"

 /* End includes */

-class Oxs_UUAnisotropy:public Oxs_Energy {
+class My_Anisotropy:public Oxs_Energy {
 private:
   REAL8m K1;
   REAL8m Ms;
-  ThreeVector axis;
+  ThreeVector axis1;
+  ThreeVector axis2;
 public:
   virtual const char* ClassName() const; // ClassName() is
   /// automatically generated by the OXS_EXT_REGISTER macro.
   virtual BOOL Init();
-  Oxs_UUAnisotropy(const char* name,  // Child instance id
+  My_Anisotropy(const char* name,  // Child instance id
  Oxs_Director* newdtr, // App director
                 Tcl_Interp* safe_interp, // Safe interpreter
```

```
                     const char* argstr);  // MIF input block parameters

-  virtual ~Oxs_UUAnisotropy() {}
+  virtual ~My_Anisotropy() {}

   virtual void GetEnergyAndField(const Oxs_SimState& state,
                                  Oxs_MeshValue<REAL8m>& energy,
                                  Oxs_MeshValue<ThreeVector>& field
                                  ) const;
 };


-#endif // _OXS_UUANISOTROPY
+#endif // _MY_ANISOTROPY
```

C++ Source Diffs

```
-/* FILE: uuanisotropy.cc            -*-Mode: c++-*-
+/* FILE: myanisotropy.cc            -*-Mode: c++-*-
  *
- * Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
+ * My Uniform Uniaxial Anisotropy, derived from Oxs_Energy class.
  *
  */

 #include "oc.h"
 #include "nb.h"
 #include "threevector.h"
 #include "director.h"
 #include "simstate.h"
 #include "ext.h"
 #include "depkey.h"
 #include "key.h"
 #include "mesh.h"
 #include "meshvalue.h"
 #include "cubicmesh.h"
```

```
-#include "uuanisotropy.h"
+#include "myanisotropy.h"
 #include "energy.h" // Needed to make MSVC++ 5 happy

 // Oxs_Ext registration support
-OXS_EXT_REGISTER(Oxs_UUAnisotropy);
+OXS_EXT_REGISTER(My_Anisotropy);

 /* End includes */


 // Constructor
-Oxs_UUAnisotropy::Oxs_UUAnisotropy(
+My_Anisotropy::My_Anisotropy(
   const char* name,      // Child instance id
   Oxs_Director* newdtr, // App director
   Tcl_Interp* safe_interp, // Safe interpreter
   const char* argstr)    // MIF input block parameters
   : Oxs_Energy(name,newdtr,safe_interp,argstr)
 {
   // Process arguments
```

```diff
   CheckInitValueParamCount("K1",1);
   K1=Nb_Atof((*FindInitValue("K1"))[0].c_str());
   DeleteInitValue("K1");

   CheckInitValueParamCount("Ms",1);
   Ms=Nb_Atof((*FindInitValue("Ms"))[0].c_str());
   DeleteInitValue("Ms");

-  CheckInitValueParamCount("axis",3);
-  axis.x=Nb_Atof((*FindInitValue("axis"))[0].c_str());
-  axis.y=Nb_Atof((*FindInitValue("axis"))[1].c_str());
-  axis.z=Nb_Atof((*FindInitValue("axis"))[2].c_str());
-  DeleteInitValue("axis");
+  CheckInitValueParamCount("axis1",3);
+  axis1.x=Nb_Atof((*FindInitValue("axis1"))[0].c_str());
+  axis1.y=Nb_Atof((*FindInitValue("axis1"))[1].c_str());
+  axis1.z=Nb_Atof((*FindInitValue("axis1"))[2].c_str());
+  DeleteInitValue("axis1");
+
+  CheckInitValueParamCount("axis2",3);
+  axis2.x=Nb_Atof((*FindInitValue("axis2"))[0].c_str());
```

```
+   axis2.y=Nb_Atof((*FindInitValue("axis2"))[1].c_str());
+   axis2.z=Nb_Atof((*FindInitValue("axis2"))[2].c_str());
+   DeleteInitValue("axis2");

    VerifyAllInitArgsUsed();

    if(Ms==0.0) {
      K1=0.0; Ms=1.0; // Safety
    }

-   REAL8m magsq=axis.MagSq();
+   REAL8m magsq=axis1.MagSq();
    if(magsq==0.0) {
      string msg="Invalid MIF input block detected for object "
        + string(InstanceName())
-       + ": Specified anisotropy axis is (0,0,0)";
+       + ": Specified anisotropy axis 1 is (0,0,0)";
      throw Oxs_Ext::Error(msg.c_str());
    }
```

```
-  if(magsq!=1.0) axis *= 1.0/sqrt(magsq);
+  if(magsq!=1.0) axis1 *= 1.0/sqrt(magsq);
+
+  magsq=axis2.MagSq();
+  if(magsq==0.0) {
+    string msg="Invalid MIF input block detected for object "
+      + string(InstanceName())
+      + ": Specified anisotropy axis 2 is (0,0,0)";
+    throw Oxs_Ext::Error(msg.c_str());
+  }
+  if(magsq!=1.0) axis2 *= 1.0/sqrt(magsq);
+
+  if(fabs(axis1*axis2)>1e-12) {
+    string msg="Invalid MIF input block detected for object "
+      + string(InstanceName())
+      + ": Specified anisotropy axes aren't perpendicular";
+    throw Oxs_Ext::Error(msg.c_str());
+  }
+
 }
```

```
-BOOL Oxs_UUAnisotropy::Init()
+BOOL My_Anisotropy::Init()
 { return 1; }

-void Oxs_UUAnisotropy::GetEnergyAndField
+void My_Anisotropy::GetEnergyAndField
 (const Oxs_SimState& state,
  Oxs_MeshValue<REAL8m>& energy,
  Oxs_MeshValue<ThreeVector>& field
  ) const
 {
   UINT4m size = state.mesh->Size();
+  ThreeVector axis3 = axis1 ^ axis2;
+  axis3.SetMag(1.0); // Just to be safe

-  REAL8m energy_mult = -K1;
-  REAL8m field_mult = 2*K1/(Ms*MU0);
+  REAL8m field_mult = -2*K1/(Ms*MU0);
```

```
   for(UINT4m i=0;i<size;++i) {
-    REAL8m temp = axis*state.spin[i];
-    energy[i] = energy_mult*temp*temp;
-    field[i] = (field_mult*temp)*axis;
+    REAL8m a = axis1*state.spin[i];
+    REAL8m b = axis2*state.spin[i];
+    REAL8m c = axis3*state.spin[i];
+    energy[i] = K1 * (a*a*b*b+a*a*c*c+b*b*c*c);
+    field[i]  = (a*(b*b+c*c))*axis1;
+    field[i] += (b*(a*a+c*c))*axis2;
+    field[i] += (c*(a*a+b*b))*axis3;
+    field[i] *= field_mult;
   }

 }
```

Adding an Energy Term to OXS: Step 2

1. Add a Specify block for your new energy term to your MIF file.

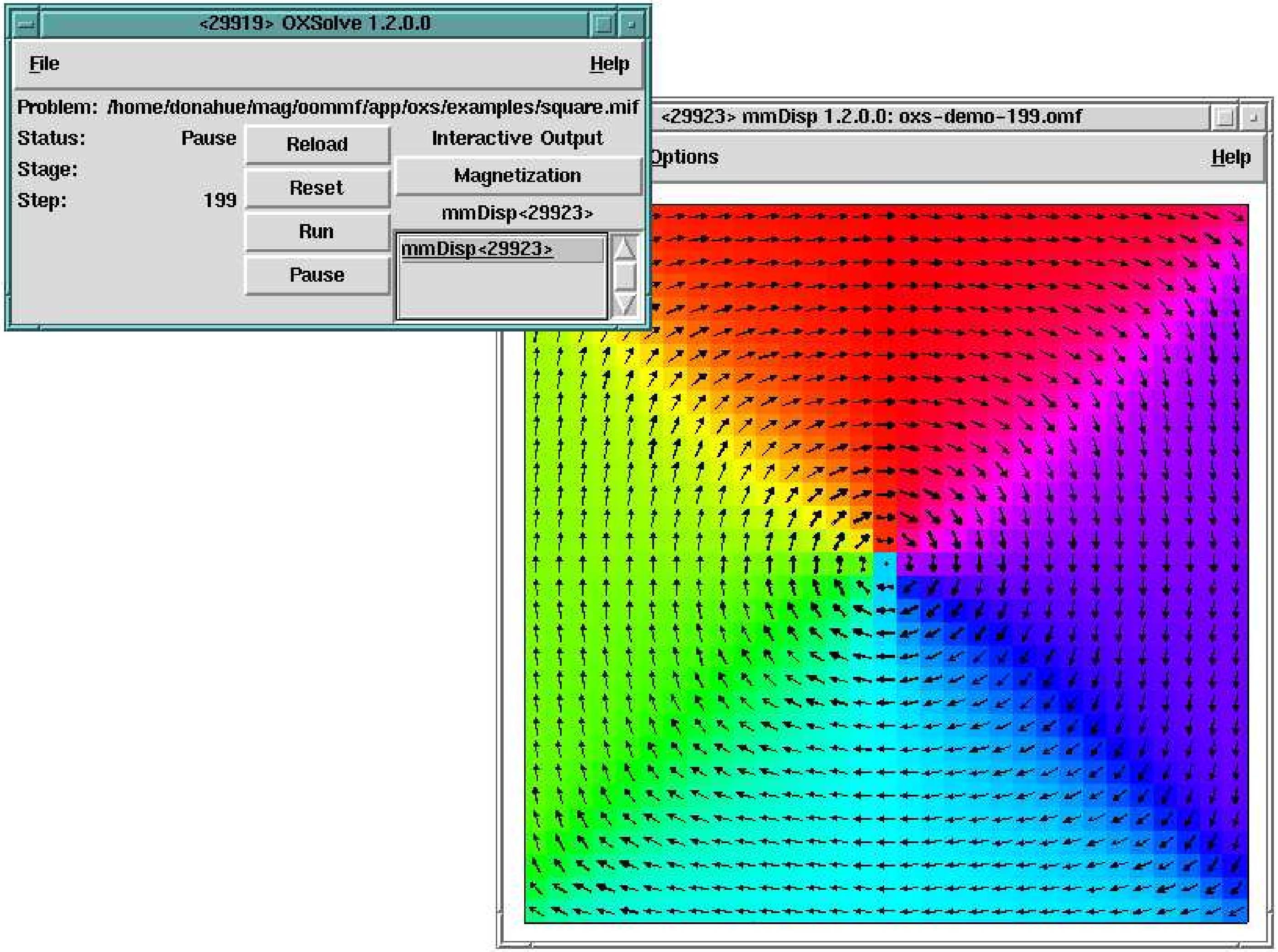
MIF File Diffs

```
+Specify My_Anisotropy {
+  Ms 8e5
+  K1 530e3
+  axis1 {1  1 0}
+  axis2 {1 -1 0}
+}
```

Adding an Energy Term to OXS: Step 3

(There is no step 3.)

Sample OXS Output



Sample OXS Output with My_Anisotropy