Working With External Materials

The External Material Model

A general external material model defines a relationship between a number of *input quantities* and a number of *output quantities*, a relationship that may also depend on *model parameters* and stored *states*. From the physics interfaces' point of view, an external material model is a black box, which may implement any relationship between provided input and required output quantities.

The external material model is implemented as a function with a C calling convention, compiled and linked to create a dynamically linked library that can be called from a material feature in COMSOL Multiphysics at runtime. In each solver iteration, current values of the input quantities and model parameters are passed together with previous values of model states as arguments to the external function. The external function is, in return, expected to write new values of model outputs, their partial derivatives with respect to model inputs, as well as updated state values to other preallocated function arguments.

The following sections provide an overview of the external materials framework, as well as details on the built-in interface types and instruction for compiling and linking a shared library.

As a security precaution, running code contained in external libraries is by default not allowed in a new COMSOL installation. Therefore, in order to use external material models, you must open the **Preferences** dialog box, go to the **Security** page, and select **Allow external processes and libraries**.

Using External Materials in Physics Interfaces

In order for a physics interface to make use of external materials, it must contain one or more features that provide input quantity definitions and make use of returned output quantities in equations and postprocessing expressions. Such features are currently available in the Solid Mechanics; Magnetic Fields; and Magnetic Fields, No Currents interfaces.

- Using external materials in Solid Mechanics requires a Structural Mechanics Module or MEMS Module license.
- AC/DC interfaces supporting external materials are only available with an AC/DC Module license, except for Magnetic Fields in 2D.

EXTERNAL MATERIAL INTERFACE TYPES

Given that a physics feature requires certain output quantities and defines certain input quantities, there are many possible ways to set up a call to an external function, including declarations of material properties and states. The required interface specification, or *interface type*, is contained in an *external material socket*, which does a number of different things:

- It defines the call syntax for the external material functions; that is, it defines the function name, return type, and the number of arguments with their data types and sizes.
- It defines a mapping between input and output quantity tensor components in the COMSOL Multiphysics variable namespace and positions in external function arguments.
- It defines a mapping between function argument positions and partial derivatives of outputs with respect to inputs.

- It decides which input quantities should be evaluated at the last converged step—rather than at the last iteration—and sets up states for these variables.
- It declares material model state variables and maps them to positions in the function arguments.
- It declares required material properties and maps them to the function arguments.
- It specifies error handling procedures, including error messages for various exit conditions.

For an external shared library to be compatible with COMSOL Multiphysics, it must follow the conventions specified by some socket. Conversely, when using an existing external library in a multiphysics model, it is important to choose the right **Interface type**. Failure to do so will result in the external library being called with unexpected arguments, leading to arbitrary behavior.

It is not necessary for a socket to define all output quantities required by a physics feature. The socket and external library may compute only some quantities, while others are left to be specified by property groups under the External Material feature. For example, the built-in General stress-strain relation socket only returns second Piola-Kirchhoff stress as output, while the physics features using this interface type also may require the material density. The density must instead be specified as an expression in the Basic property group.

It is also possible to set the **Interface type** to **None** to select no socket and instead set up the relation between input and output quantities as expressions directly in an external material feature. This behavior is intended mostly for testing and debugging of multiphysics models.

Sockets and associated material property groups are identified by an ID tag that must be unique among all COMSOL installations between which files will be shared. In order to avoid conflicts between built-in sockets and future user-developed sockets, a full Java-style naming convention has been adopted. Therefore, for example, the built-in General H(B) relation socket has the unique ID `com.comsol.generalHBRelation` and the associated material property group has tag and name `comcomsolgeneralHBRelation`—that is, a direct concatenation of the ID.

SOLVER ITERATIONS AND STATE UPDATES

This section describes quantities as being evaluated either at the *current* step or at the *previous converged* step. This is related to the iteration pattern of the solvers. The external material functionality is primarily intended for modeling materials with some kind of memory or path dependence (for example, inelastic solid materials or materials exhibiting electromagnetic hysteresis). This means that simulations must be performed in steps over an interval of time or some parameter (which can often be interpreted as a pseudo-time).

Whether a time-dependent or parametric solver is used, taking a step forward requires solving a nonlinear problem, which is typically an iterative procedure. In order for the convergence to be efficient, the modified Newton solver used needs correct Jacobian information (the Jacobian is sometimes called the *tangential stiffness matrix*), which must be provided by the external material functions in the form of partial derivatives of output argument components with respect to input argument components. When the nonlinear iteration converges, the solution for that time step or parameter step is stored, and the solver moves on to the next step.

The external material functions are typically called in each iteration in the inner nonlinear loop. Arguments that are evaluated at the *current* step are reevaluated at each inner iteration, while arguments defined at the previous converged step retain the value they had when the previous nonlinear iteration converged; that is, the value that was last stored. State variable arguments are passed to the external material function with the same previous converged values in each inner nonlinear iteration. The external material function is expected to overwrite this previous value with a new value corresponding to the current iteration, but this value will only be stored and the states updated when the inner nonlinear iteration converges.

Built-in Material Function Interface Types

There are currently four different built-in external material function interface types, or sockets, which have many traits in common:

- General stress-strain relation
- Inelastic residual strain
- General H(B) relation
- General B(H) relation

All of these allow the user to specify the number of states, `nStates`, to be defined at each integration point. These state values are passed to the external functions in an argument called `states`. Note that the `states` vector is both input and output: when the function is called, it contains the previous step converged values of the states; on return, it must contain the state values to be stored if the solver decides to proceed to the next step.

The user is also required to specify an array of material model parameters, called `par`. The number of parameters `nPar` is specified implicitly as the length of the material property array, which must be defined as a property in a property group under the calling **External Material** feature or set in the **Material Contents** table in the External Material node settings.

The functions may be written so as to accept a varying number of parameters and states, or to require fixed numbers. In any case, the number of parameters and states passed should be checked, and unexpected argument lengths should return with an error condition as given in the following table:

TABLE 9-15: ACCEPTED RETURN VALUES FOR THE GENERAL STRESS-STRAIN RELATION

RETURN VALUE	MEANING
0	Normal exit
1	Error: “Wrong number of parameters”
2	Error: “Wrong number of states”
anything else	Unspecified error

GENERAL STRESS-STRAIN RELATION

The **General stress-strain relation** socket implements a stress-strain relation computing a second Piola-Kirchhoff stress tensor given the current Green-Lagrange strain together with a material property vector and a vector of stored states. The expected external material function signature is:

```
int eval(double *e,           // Green-Lagrange strain, input
         double *s,           // Second Pioloa-Kirchhoff stress, output
         double *D,           // Jacobian of stress with respect to strain,output
         int *nPar,           // Number of material model parameters, input
         double *par,         // Material model parameters, input
         int *nStates,        // Number of states, input
         double *states) { }  // States, input/output
```

The `e` and `s` tensors are given in Voigt order—that is., the components in `e` are $\{e_{xx}, e_{yy}, e_{zz}, e_{yz}, e_{xz}, e_{xy}\}$ and similarly for `s`. The Jacobian `D` is a 6-by-6 matrix of partial derivatives of components of `s` (rows) with respect to components of `e` (columns); the matrix is stored in row-major order.

INELASTIC RESIDUAL STRAIN

The **Inelastic Residual Strain** socket implements an update procedure for an additive inelastic contribution to the total Green-Lagrange strain. Total stress and strain at the previous converged step, current total strain, current temperature, a reference temperature, a material property vector, and a vector of stored states are passed as inputs.

```
int eval(double *sOld,         // Second Pioloa-Kirchhoff stress at previous step, input
         double *eOld,         // Green-Lagrange strain at previous step, input
```

```
          double *e,           // Green-Lagrange strain at current step, input
          double *T,           // Temperature, input
          double *Tref,        // Strain reference temperature, input
          double *eInel,       // Inelastic Green-Lagrange strain state, input/output
          double *Jac,         // Jacobian of inelastic strain, output
          int *nPar,           // Number of material model parameters, input
          double *par,         // Material model parameters, input
          int *nStates,        // Number of extra states, input
          double *states) { }  // Extra states, input/output
```

The `sOld`, `eOld`, `e`, and `eInel` tensors are given in Voigt order (that is, the components in `e` are {e_{xx}, e_{yy}, e_{zz}, e_{yz}, e_{xz}, e_{xy}}) and similarly for the other tensors. The Jacobian `Jac` is a 6-by-6 matrix of partial derivatives of components of `eInel` (rows) with respect to components of `e` (columns); the matrix is stored in row-major order. Note that the primary output quantity `eInel` is declared as states, meaning that the argument on entry contains the previous converged step values. The temperature arguments `T` and `Tref` are standard model inputs, which are specified in the physics feature calling the external material where the Inelastic residual strain socket is selected.

GENERAL H(B) RELATION

The **General H(B) relation** socket implements a generalization of an HB curve. It computes an updated magnetic field corresponding to an updated magnetic flux density, given the magnetic field and magnetic flux density at the previous converged step. A material property vector and a vector of stored states are passed as additional input. Typical implementations will use extra states to model hysteresis.

```
int eval(double *oldB,         // Magnetic flux density at previous step, input
         double *B,            // Magnetic flux density, input
         double *H,            // Magnetic field state, input/output
         double *Jac,          // Jacobian of H with respect to B, output
         int *nPar,            // Number of material model parameters, input
         double *par,          // Material model parameters, input
         int *nStates,         // Number of extra states, input
         double *states) { }   // Extra states, input/output
```

The magnetic flux densities `oldB` and `B` and the magnetic field `H` are passed as arrays of length 3. The Jacobian `Jac` is a 3-by-3 matrix of partial derivatives of components of H (rows) with respect to components of B (columns); the matrix is stored in row-major order. Note that the primary output quantity H is declared as states, meaning that the argument on entry contains the previous converged step values.

GENERAL B(H) RELATION

The **General B(H) relation** socket implements a generalization of a BH curve. It computes an updated magnetic flux density corresponding to an updated magnetic field, given the magnetic field and magnetic flux density at the previous converged step. A material property vector and a vector of stored states are passed as additional input. Typical implementations will use extra states to model hysteresis.

```
int eval(double *oldH,         // Magnetic field at previous step, input
         double *H,            // Magnetic field, input
         double *B,            // Magnetic flux density state, input/output
         double *Jac,          // Jacobian of Bwith respect to H, output
         int *nPar,            // Number of material model parameters, input
         double *par,          // Material model parameters, input
         int *nStates,         // Number of extra states, input
         double *states) { }   // Extra states, input/output
```

The magnetic fields `oldH` and `H` and the magnetic flux density `B` are passed as arrays of length 3. The Jacobian `Jac` is 3-by-3 matrix of partial derivatives of components of `B` (rows) with respect to components of `H` (columns); the matrix is stored in row-major order. Note that the primary output quantity `B` is declared as states, meaning that the argument on entry contains the previous converged step values..

For examples showing how to implement and use external materials, see

- http://www.comsol.com/model/external-material-examples-structural-mechanics-32331
- http://www.comsol.com/model/external-materials-general-hb-bh-relation-32321

How to Compile and Link an External Material Model

To export functions from the DLL when using Microsoft Visual Studio to compile your library, you must declare the functions as `__declspec(dllexport)`. Therefore, to write a source code that works across platforms, use the following `#define` pattern:

```
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

EXPORT <return_type> eval(<arguments>) { }
```

COMPILING AND LINKING

To compile the function into a library, place it in a file (here called `ext.c` as an example) and proceed as follows depending on the platform:

See http://www.comsol.com/system-requirements for information about supported compiler versions.

- 64-bit Windows with Microsoft Visual Studio 2010:
  - Start Microsoft Visual Studio > Visual Studio Tools > Visual Studio x64 Win64 Command Prompt (2010) from the Windows Start Menu.
  - `cd` to the directory that contains `ext.c.`
  - `cl /MT /c ext.c`
  - `link /OUT:ext.dll /DLL ext.obj`
- 64-bit Linux with Intel Compiler:
  - cd to the directory that contains `ext.c.`
  - `icc -fPIC -c ext.c`
  - `icc -shared -fPIC -Wl,-z -Wl,defs -o ext.so ext.o -ldl`
- 64-bit Mac OS X with Intel Compiler:
  - cd to the directory that contains `ext.c.`
  - `icc -fPIC -c ext.c`
  - `icc -dynamiclib -fPIC -o ext.dylib ext.o`

For other compilers, refer to the compiler’s documentation for instructions on how to compile and create a shared library.

Known Issues for External Materials

Being new functionality, the external materials framework suffers from a few known issues which may affect usability in some cases:

- There is no unit support: Inputs are passed to the external function in the model's base unit system and outputs are interpreted in the same system.
- Shared libraries containing external material functions are loaded the first time they are used and never reloaded. To avoid having to repeatedly restart COMSOL Multiphysics while developing an external material function, you can simply change the name of the linked library each time you make a change to it.
- The way the external functions are called is not fault tolerant in any way. This means that reading or writing outside allocated memory in an external function will typically make the COMSOL Desktop crash. Take care.

External Material

The **External Material** node ( ) sets up an interface between a physics feature and functions in an external shared library. In addition, it contains most of the functionality of a standard **Material** node, letting you add arbitrary material properties and property groups. Some aspects of a material can be handled by an external library, while others are defined internally in property groups.

The **External Material** node is only available under the **Global Definitions>Materials** node, not under **Materials** inside components. To use property groups under an **External Material** as domain material on geometric entities in a component, use a Material Link node.

This section focuses on using a material model from an existing external library in your COMSOL model. General information about the external materials framework and how to create your own custom library can be found under Working With External Materials.

EXTERNAL MATERIAL MODEL

Enter a **Library** path and name (the complete network path), or click **Browse** to locate a library to import. Depending on the platform, the library can be a DLL, .so, or .dylib file.

Select the appropriate **Interface type** matching your library. The default is **None**; other options may vary between COMSOL installations. The following types are preinstalled:

- **General stress-strain relation**
- **Inelastic residual strain**
- **General H(B) relation**
- **General B(H) relation**

If required by the chosen interface type, set the **Number of states** that must be stored at each evaluation point and provide a **State name** which will be used as a basis for creating state variables. Actual state variables are created by adding the `<matname>.state.` namespace as a prefix and appending a state index to the given name. For example, if for a material with name `extmat1` you request two states with **State name** `p`, the state values can be accessed during and after a solution as `extmat1.state.p1` and `extmat1.state.p2`.

Also, if allowed for the chosen interface type, select the **Pass arguments as complex** check box to use `complex` rather than `double` as the base type in all numeric array arguments to the external functions.

The **Required input quantities**, **Output quantities**, and **Model states** tables provide an overview of the interface, including which quantities are passed to and from calling physics features, which states are declared, and which component variables are defined by the material feature. Note that all variables are defined in the material's namespace. To directly access, for example, the first axial component of a second Piola-Kirchhoff stress output outside the scope of an **External Material** node with the name `extmat1`, use the variable name `extmat1.output.S11`.

Components of the **Required input quantities** are normally defined by a physics feature calling the external material. When using an external material in equation-based modeling, you can set the inputs up manually by defining the required input components (in the material's namespace) as global variables. For example, if external material `extmat1` requires the temperature T as input, define a global variable called `extmat1.input.T`.

Use the **Init** column in the **Model states** table to specify initial values for all internal states used by the external function. This includes both specific states required by the chosen **Interface type** and numbered states added by this feature.

MATERIAL PROPERTIES

The **Material Properties** section of an external material is identical to the same section in a common **Material** feature. See The Settings Window for Material for more information.

MATERIAL CONTENTS

The **Material Contents** section of an external material is very similar to the same section in a common **Material** feature (see The Settings Window for Material for more information). The only real difference is that the table shows not only defined output properties and properties required by some physics feature, but also properties required as input to the external material functions. Depending on the selected **Interface type**, these may appear either as individually named properties or as a single generic, arbitrary-length property array. In the latter case, any particular external library typically requires a specific set of properties in a certain order to be specified as an array inside curly brackets. For example, two required parameters can be specified as `{2e11,0.33}`.

Properties required as input to the external are stored in a property group subnode with the same name as the selected **Interface type**.

APPEARANCE

The **Appearance** section of an external material is identical to the same section in a common **Material** feature. See The Settings Window for Material for more information.