Version 2.0

User's Manual

Last Updated: July 7, 2014

Contributors
Steve Maas (steve.maas@utah.edu)
Dave Rawlins (rawlins@sci.utah.edu)
Dr. Jeffrey Weiss (jeff.weiss@utah.edu)
Dr. Gerard Ateshian (ateshian@columbia.edu)

Contact address
Musculoskeletal Research Laboratories, University of Utah
72 S. Central Campus Drive, Room 2646
Salt Lake City, Utah

Website
MRL: http://mrl.sci.utah.edu
FEBio: http://mrl.sci.utah.edu/software/febio

Forum
http://mrl.sci.utah.edu/forums/

Development of the FEBio project is supported in part by a grant from the U.S. National Institutes of Health (R01GM083925).

Table of Contents

Chapter 1 Introduction

1.1. Overview of FEBio

FEBio is a nonlinear finite element solver that is specifically designed for biomechanical applications. It offers modeling scenarios, constitutive models and boundary conditions that are relevant to many research areas in biomechanics, thus offering a powerful tool for solving 3D problems in computational biomechanics. The software is open-source and the source code, as well as pre-compiled executables for Windows, OS-X and Linux platforms are available for download at http://mrl.sci.utah.edu/software. This chapter presents a brief overview of the available features of FEBio.

FEBio can solve different kinds of physics. It can solve problems in structural mechanics, heat transfer, biphasic and multiphasic physics. Both a (quasi-) static and dynamic (or transient) analysis can be performed in each of the different physics modules. For instance, in the structural mechanics module, the (quasi-) static response of the system is sought in a quasi-static analysis and the effects of inertia are ignored. In a dynamic analysis, the inertial effects are included in the governing equations to calculate the time dependent response of the system. In the biphasic module, a coupled solid-fluid problem is solved. In a transient biphasic analysis the time dependent response of both the solid and the fluid phase is determined. For the steady-state analysis the final relaxed state is recovered. Similarly, for multiphasic and heat transfer problems, both the time dependent transient response as well as the steady-state response can be determined.

Many nonlinear constitutive models are available, allowing the user to model the often complicated biological tissue behavior. Several isotropic constitutive models are supported such as Neo-Hookean, Mooney-Rivlin, Ogden, Arruda-Boyce and Veronda-Westmann. All these models have a nonlinear stress-strain response and are objective for large deformations. In addition to the isotropic models there are several transversely isotropic and orthotropic constitutive models available. These models exhibit anisotropic behavior in a single or multiple preferred directions and are useful for representing biological tissues such as tendons, muscles, cartilage and other tissues that contain fibers. FEBio also contains a *rigid body* constitutive model. This model can be used to represent materials or structures whose deformation is negligible compared to that of other materials in the overall model. Several constitutive models are available for representing the solid phase of biphasic and multiphasic materials, which are materials that contain both a solid phase and a fluid phase. For incompressible materials FEBio employs special algorithms for enforcing the incompressibility constraint. A three-field formulation is used for tri-linear hexahedral and wedge elements. This algorithm allows the user to capture the accurate response of highly incompressible materials.

FEBio can now also solve first-order computational homogenization problems. In such problems, the response of the macro-model is determined by the averaged local response of a representative volume element (RVE). The deformation of the macro-model, and more specifically the local deformation gradient, is applied to a RVE model which in turns determines the stress (and tangent) of the macro-model.

FEBio supports a wide range of boundary conditions to model interactions between materials that are relevant to problems in biomechanics. These include prescribed displacements, fluid pressures or temperatures, depending on the physics model. Also nodal forces or fluxes can be prescribed, as well as several types of surface boundary loads. Deformable models can be connected to rigid bodies. With this feature, the user can model prescribed rotations and torques for rigid segments, thereby allowing the coupling of rigid body mechanics with deformable continuum mechanics. FEBio provides the ability to represent frictionless and frictional contact between rigid and/or deformable materials using sliding interfaces. A sliding surface is defined between two surfaces that are allowed to separate and slide across each other but are not allowed to penetrate. Variations of the sliding interface, such as tied interfaces, tied-sliding (tension-compression) and rigid walls, are available as well. As of version 1.2 it is also possible to model the fluid flow across two contacting biphasic materials. Finally, the user may specify a body force to model the effects such as, gravity, base acceleration or centripetal acceleration.

FEBio has a large library of element formulations. These include linear tetrahedral, hexahedral and pentahedral (wedge) elements, as well as quadratic tetrahedral and hexahedral elements. FEBio also supports triangular and bi-linear quadratic director-based shell elements.

FEBio is a nonlinear implicit FE solver and does not have mesh generation capabilities. The finite element mesh, as well as all constitutive parameters and loading is defined in an input file, the format of which is described in detail in this document. This input file needs to be generated by preprocessing software. The preferred preprocessor for FEBio is called *PreView*. PreView can convert some other formats to the FEBio input specification. For instance, NIKE3D [1] and Abaqus input files can be imported in PreView and can be exported from PreView as a FEBio input file. See the PreView User's Manual for more information.

1.2. About this document

This document is part of a set of three manuals that accompany FEBio: the User's Manual, describing how to use FEBio (this manual), a Developer's Manual for users who wish to modify or add features to the code, and a Theory Manual, which describes the theory behind the FEBio algorithms.

This document discusses how to use FEBio and describes the input file format in detail. Chapter 2 describes how to run FEBio and explains the various command line options. It also discusses the different files that are required and created by FEBio. Chapter 3 describes the format of the FEBio input file. An XML-based format is used, organizing the data in a convenient hierarchical structure. Chapter 4 gives a detailed overview of the available constitutive models. Chapter 5 discusses the restart capabilities of FEBio. The restart feature allows the user to interrupt a run and continue it at a later time, optionally making changes to the problem data. Chapter 6 describes the multi-step analysis feature, which allows the user to split up the entire analysis into several steps. Chapter 7 explains how to setup and run a parameter optimization problem using FEBio's optimization module.

Although this document describes some of the theoretical aspects of FEBio, a complete theoretical development can be found in the *FEBio Theory Manual*. Developers who are

interested in modifying or extending the FEBio code will find the *FEBio Developer's Manual* very useful.

1.3. Units in FEBio

FEBio does not assume a specific unit system. It is up to the user to enter numbers that are defined in consistent units. For example, when entering material parameters in SI units, the user must enter all loads, contact parameters, and other boundary conditions in SI units as well. The units of all the parameters are given when they are defined in this manual. We use a generic designation of units for all the parameters using the following symbols.

Symbol	***Name***	***SI unit***
L	Length	meter (m)
M	Mass	kilogram (kg)
t	Time	second (s)
T	Temperature	Kelvin (K)
n	Amount of substance	mole (mol)
F	Force	Newton ($N = kg{\cdot}m/s^2$)
P	Pressure, stress	Pascal ($Pa = N/m^2$)
Q	Electric charge	Coulomb (C = s.A)

Units are given using the bracket notation. For instance, the unit for density is [$\mathbf{M/L^3}$] and the unit for permeability is [$\mathbf{L^4/F{\cdot}t}$]**.** When using SI units, this corresponds to units of kg/m^3 for density and $m^4/N.s$ for permeability, respectively. Unitless parameters are designated by empty brackets (**[ ]**). The units for angles are either [**deg**] for degrees or [**rad**] for radians.

When adopting a consistent set of units, first choose a primary set of units, and then determine the remaining derived units. For example, in typical problems in solid mechanics, the primary set consists of three units. If you choose [**M**]=kg, [**L**]=m, and [**t**]=s, then [**F**]=N and [**P**]=Pa. Alternatively, if you choose [**L**]=mm, [**F**]=N and [**T**]=s as the primary set, then [**P**]=MPa (since $1\ N/mm^2 = 10^6\ N/m^2 = 1$ MPa) and [**M**]=tonne (tonne = $N{\cdot}s^2/mm$). The primary set of units must be independent. For instance, in the last example, you cannot choose [**P**] as a primary unit as it can be expressed in terms of [**F**] and [**L**] (i.e. [**P**]=[$\mathbf{F/L^2}$]).

Example:

Primary Units	
time	s
length	mm
force	N
amount of substance	nmol
charge	C
temperature	K
Derived Units	
stress	N/mm^2, MPa

permeability	$mm^4/N{\cdot}s$, $mm^2/MPa \cdot s$
diffusivity	mm^2/s
concentration	$nmol/mm^3$, mM
charge density	nEq/mm^3, mEq/L
voltage	mV
current density	A/mm^2
current	A

Chapter 2 Running FEBio

FEBio is a command line application which means it does not have its own Graphical User Interface (GUI) and must be run from a shell or command line. FEBio runs on several different computing platforms including Windows XP, Win7, Win8, Mac OSX and many versions of Linux. The command line input and output options are described in this chapter.

2.1. Running FEBio on Windows

There are several ways to run FEBio on Windows. The easiest way is by simply selecting the FEBio program from the Programs menu or by double-clicking the FEBio icon in the installation folder. However, this runs FEBio with the installation folder as the working folder, and unless the FEBio input files are in this folder, you will need to know the relative or absolute path to your input files. A more practical approach is to run FEBio from a command prompt. Before you can do this, you need to know two things: how to open a command prompt and how to add the FEBio installation folder to your PATH environment variable so that you can run FEBio from any folder on your system. The process is slightly different depending on whether you are using Windows XP or Windows 7, so we'll look at the two Windows versions separately.

2.1.1. Windows XP

First, we'll add the FEBio installation folder to the PATH variable. Open the *Control Panel* from the *Start* menu. Switch to *Classic View* and double-click the *System* icon. In the dialog box that appears, select the *Advanced* tab and click the button named *Environment variables*. Find the *path* variable and click the *Edit* button. Add the end of the PATH's value (don't delete the current value) type a semi-colon and then the absolute path to the FEBio installation folder (e.g. C:/Program Files/FEBio/). Then click the *OK*-button on all open dialog boxes.
To open a command prompt, click the *Run* menu item on the *Start* menu. In the dialog box that appears type *cmd* and press the *OK*-button. A command prompt window appears. You can now use the *cd* (change directory) command to navigate to the folder that contains the FEBio input files. To run FEBio, simply type *febio* (with or without additional arguments) and press *Enter*.

2.1.2. Windows 7

Let's first modify the *PATH* environment variable. Open the *Start* menu and type *system* in the search field. From the search results, select the *System* option under *Control Panel*. The *System* window will appear. Find the *Change Settings* option (on the lower, right side) and click it. The System Properties dialog box appears. Activate the *Advanced* tab and click the *Environment Variables* button. Find the *path* variable and click the *Edit* button. Add the end of the PATH's value (don't delete the current value) type a semi-colon and then the absolute path to the FEBio installation folder (e.g. C:/Program Files/FEBio/). Then click the *OK*-button on all open dialog boxes.

Next, open a command prompt as follows. Click the *Start* menu and type *cmd* in the search field. From the search results, select the *cmd* option under Programs. A command prompt window

appears. You can now use the *cd* (change directory) command to navigate to the folder that contains the FEBio input files. To run FEBio, simply type *febio* (with or without additional arguments) and press *Enter*.

2.1.3. Running FEBio from Explorer

A third method of running FEBio, which often is very convenient, is to run FEBio from Windows Explorer. To do this, first open Explorer and browse to the folder that contains your FEBio input files. Next, right-click on the input file and select *Open With*. Now select *Choose default program*. A dialog box appears with a list of programs to open the input file. If FEBio is not on this list yet, click the *Browse* button. Locate the FEBio executable (e.g. in C:/Program Files/FEBio2/bin), select it and press the *Open* button. Now select FEBio in the *Open With* dialog box and press Ok.

After you have done this once, the process simplifies. After you right-click the input file, FEBio should now show up in the *Open With* menu item and can be selected immediately without having to go through all the previous steps.

2.2. Running FEBio on Linux or MAC

Running FEBio on Linux or Mac is as easy as opening up a shell window and typing FEBio on the command line. However, you may need to define an alias to the folder that contains the FEBio executable if you want to run FEBio from any folder on your system. Since this depends on your shell, you need to consult your Linux documentation on how to do this. E.g. if you are using c-shell, you can define an alias as follows:

```
alias febio '/path/to/febio/executable/'
```

If you don't want to define this alias every time you open a shell window, you can place it in your shell start up file (e.g. *.cshrc* for c-shell).

2.3. The Command Line

FEBio is started from a shell window (or the *command prompt* in Windows). The command line is the same for all platforms:

```
febio [-o1 [name1] | -o2 [name2] | …]
```

Where `-o1`, `-o2` are options and *`name1, name2,`* … are filenames. The different options (of which most are optional) are given by the following list:

- `-i` : name of input file
- `-r` : restart file name
- `-g` : debug flag (does not require a file name)

- `-p`: plot file name
- `-a`: dump file name
- `-o`: log file name
- `-c`: data check only
- `-s`: material parameter optimization control file
- `-nosplash`: don't show the welcome screen
- `-cnf`: configuration filename
- `-noconfig`: don't use the configuration file

A more detailed description of these options follows.

-i

The `-i` option is used to specify the name of the input file. The input file is expected to follow the format specifications as described in Chapter 3.

Example:

```
> febio -i input.feb
```

This is the most common way to start a FEBio run. However, FEBio allows the omission of the `-i` when only a filename is given.

```
> febio input.feb
```

On Windows, this allows for starting FEBio by double-clicking on an input file (assuming you have chosen FEBio as the default program to open .feb files). Note that if additional options are specified on the command line the `-i` must be present.

-r

The `-r` option allows you to restart a previous analysis. The filename that must follow this option is a FEBio *restart input file* or a *dump file*. The restart input file and dump file are described in more detail in Chapter 5. The `-i` and `-r` options are mutually exclusive; only one of them may appear on the command line.

Example:

```
> febio -r file.feb
```

-g

The `–g` option runs FEBio in *debug mode*. See section 2.8.2 for more information on running FEBio in debug mode.

Example:

```
> febio -i input.feb –g
```

-p
The –p option allows the user to specify the name of the *plot file*. The plot file is a binary file that contains the main results of the analysis. FEBio usually provides a default name for this file; however, the user can override the default name using this option. See section 2.7 for more details on the output files generated by FEBio.

Example:
```
> febio -i input.feb –p out.plt
```

-a
It is possible to restart a previous run using the restart capability in FEBio. This is useful when a run terminates unexpectedly. If that happens, the user can restart the analysis from the last converged timestep. Before this feature can be used, the user must request the creation of a *dump file*. This file will store all the information that FEBio will need to restart the analysis. FEBio will usually provide a default name for the dump file, but the –a command line option allows the user to override the default name for the dump file. See section 2.8.3 and Chapter 5 for more details on how to use the restart feature.

Example:
```
> febio -i input.feb –a out.dmp
```

-o
The –o option allows the user to set the name of the *log file*. The log file will contain a record of the screen output that was generated during a run. FEBio usually provides a default name for this file (see section 2.7), but the user can override it with this command line option.

Example:
```
> febio -i input.feb –o out.log
```

-c
When the –c option is specified on the command line, FEBio will only read the input file and check it for possible errors. When the check is complete, FEBio will terminate. See section 2.8.4 for more details on this option.

Example:
```
> febio -i input.feb –c
```

-nosplash
When the –nosplash command is entered on the command line, FEBio will not print the welcome message to the screen. This is useful when calling FEBio from another application and when the user wishes to suppress any screen output from FEBio. Other options for suppressing output can be set in the control section of the FEBio input file (see section 3.4.1).

Example:
```
> febio -i input.feb –nosplash
```

-silent
When the –silence option is specified on the command line, FEBio will not generate any output to the screen. Unless explicitly instructed not to, FEBio will still create a log file which will have the convergence information.

Example:
```
> febio -i input.feb –silent
```

-cnf
-noconfig
As of version 1.2, FEBio uses a *configuration file* to store platform specific settings. Usually FEBio assumes that the location for this configuration file is the same as the executable. However, the user can specify a different location and filename using the `–cnf` command line option. If the user does not have a configuration file or does not wish to use one, this can be specified using the `–noconfig` option. More details on the configuration file can be found in section 2.5 and Appendix A.

Example:
```
> febio -i input.feb –cnf C:\path\to\febio.xml.
```

-s
This option instructs FEBio to run a material parameter optimization on the specified input file. The optimization module is described in detail in Chapter 7. The –s option is followed by the optimization control file which contains among other things the parameters that need to be optimized. Note that the restart feature does not work with the optimization module.

Example:
```
> febio –i file.feb –s control.feb
```

2.4. The FEBio Prompt

If you start FEBio without any command arguments, the FEBio prompt will appear (after the welcome message). It will look something like this:

```
febio>
```

You can now enter one of the following commands:

- **help**: prints an overview of available commands with a brief description of each command.
- **quit**: exit FEBio.
- **run**: run an FEBio input file. This command takes the same options as you can enter on the command line. For example, to run a file named *test.feb* from the FEBio prompt, type the following:

  ```
  run –i test.feb
  ```

- **version**: print version information.

You can also bring up the FEBio prompt during a run by pressing ctrl+c[*]. See section 2.8.1 for more details.

2.5. The Configuration File

As of version 1.2, FEBio uses a *configuration file* to store platform-specific settings, such as the default linear solver and the list of plugins that need to be loaded at startup. The configuration file uses an xml format to store data and is detailed in Appendix A. For backward compatibility, it is still possible to run FEBio without the configuration file. In that case, the default settings prior to version 1.2 are used.

Example:

```
> febio –i myfile.feb -noconfig
```

The configuration file needs to be stored in the same location as the executable and named *febio.xml*. Alternatively, the location of the file can also be specified on the command line using the –cnf option.

Example:

```
> febio -i myfile.feb –cnf /home/my/folder/FEBio/febio.cnf
```

2.6. Using Multiple Processors

As of version 2.0, FEBio uses OpenMP to parallellize several of the finite element calculations, improving the performance considerably. Both the right-hand-side and the stiffness matrix evaluations for many types of problems have been parallelized. On a system with four processors, a speedup of 2-3 can be expected, depending on the size and type of model. Models with complex material behavior (such as EFD-type materials, biphasic, multiphasic materials, etc.) will benefit most from these parallelization efforts. In addition, FEBio implements the MKL version of the PARDISO linear solver, which is a parallel linear solver using OpenMP.

To use multiple processors set the environment variable OMP_NUM_THREADS to the number of desired threads. You should set the number of threads to be equal or less than the number of processors on your system (Setting it higher may actually decrease performance). For example, on a system with four processors you can set the environment as follows. On Linux using the Bash shell, execute:

```
> export OMP_NUM_THREADS=4
```

Using the c-shell, execute:

```
> setenv OMP_NUM_THREADS 4
```

[*] This feature does not work on some Linux platforms and may abruptly terminate the run.

Or at a Windows command prompt:

```
> set OMP_NUM_THREADS=4
```

On Windows, you can add this environment variable as well from the Control Panel. On Win7, open the Control Panel (*Start→Control Panel*). Open the System panel and click *Change Settings*. The *System Properties* dialog box should open up. Select the *Advanced Tab* and click the *Environment Variables* button. In the next dialog box, click the *New…* button under the *User variables*. Enter OMP_NUM_THREADS for the variable name and 4. Click OK on all open dialog boxes.

A note on repeatability
When using multiple processors, it can not always be guaranteed that all calculations are executed in the same order and, due to numerical round-off, the results of these calculations will not always be the same. In FEBio, this means that the same model run repeatedly on the same machine with multiple processors, may give slightly different convergence norms or even slightly different answers. In many cases, the differences should be small, but in some problems that are prone to ill-conditioning (e.g. contact) the discrepancies may be more significant. When running on one processor, the results of consecutive runs should always be identical.

2.7. FEBio Output

After running FEBio, two or three files are created: the *log file*, the *plot file* and optionally the *dump file*. The log file is a text file that contains the same output (and usually more) that was written to the screen. The *plot file* contains the results of the analysis. Since this is a binary file, the results must be analyzed using post processing software such as *PostView*. In some cases, the user may wish to request the creation of a *dump file*. This file contains temporary results of the run. If an analysis terminates unexpectedly or with an error, this file can be used to restart the analysis from the last converged time step. See Chapter 5 for more details. The names of these files can be specified with the command options `-p` (plot file), `-a` (dump file), `-o` (log file). If one or more of the file names following these flags are omitted, then the omitted file name(s) will be given a default name. The default file names are derived from the input file name. For example, if the input file name is *input.feb* the logfile will have the name *input.log*, the plot file is called *input.xplt* and the dump file is called *input.dmp*.

Note 1: The name of the log and plot file can also be specified in the FEBio input file. See section 3.15 for more information.

Note 2: When running an optimization problem the name of the log file is derived from the optimization control file. See Chapter 5 for more information on running optimization problems with FEBio.

2.8. Advanced Options

2.8.1. Interrupting a Run*

The user can pause the run by pressing *ctrl+c*. This will bring up the FEBio prompt, and the user can enter a command. The following commands are available.

- *cont*: continues the run. FEBio will continue the analysis where it left off.
- *conv*: force the current time step to converge. This is useful for example when a time step is having difficulty satisfying too tight of convergence criteria. The user can then manually force the convergence of the time step. However, if the convergence difficulties are due to instabilities, forcing a time step to converge could cause the solution to become unstable or even incorrect. Also be aware that even if the solution recovers on later timesteps, the manually converged step might be incorrect.
- *debug [on|off]*: entering *debug* will toggle debug mode. Adding *on* (*off*) will turn the debug mode on (resp. off). In debug mode, FEBio will store additional information to the log and plot file that could be useful in debugging the run. It is important to note that since FEBio will store all non-converged states to the plot file, this file may become very large in a short number of time steps. See section 2.8.2 for more details on debugging.
- *dtmin*: set the minimum time step size. This command overrides the minimum time step size that was specified in the input file.
- *fail*: stop the current iteration and retry. If the current time step is not converging and if the auto-time stepper is enabled, the fail command will stop the current time step and retry it with a smaller time step size. If the auto-time stepper is not enabled, the fail command will simply exit the application.
- *help*: list the available commands with a short explanation. Prints the information provided in this section of the manual.
- *plot*: dump current state to plot database and continue. This command is useful when you want to store the non-converged state at the current iteration. Note that this command only stores the state at the current iteration. If you turn on debug mode, all the iterations are stored to the plot file.
- *print*: print values of variables:
  - *nnz*: number of non-zeroes in stiffness matrix
  - *neq*: number of equations
  - *time*: the current time step
- *quit*: exit the application
- *restart*: toggles restart flag. When the restart flag is set, FEBio will create a dump file at the end of each converged time step. This dump file can then later be used to restart the analysis from the last converged time step. See Chapter 5 for more details on FEBio's restart feature.
- *time*: print elapsed time and an estimation of the remaining time.
- *version*: print version information

* This feature may not work properly on all systems, although it will always work on Windows systems.

Note that it may take a while before the FEBio prompt is displayed after the user requests a *ctrl+c* interruption. This may be because the program may be in the middle of a call to the linear solver or another time-consuming part of the analysis procedure that cannot be interrupted.

2.8.2. Debugging a Run

As stated in Section 2.3, FEBio can be run in debug-mode by specifying the `–g` option on the command line. When running in debug mode, FEBio performs additional checks and prints out more information to the screen and to the plot file. It will also store all non-converged geometry states to the plot file. These non-converged states can be very useful for determining the cause of non-convergence or slow convergence. Because of this additional work, the problem may run slightly slower. Note that debug mode can be turned on/off while running an analysis by first interrupting the run with *ctrl+c* and then using the *debug* command to toggle the debug mode on or off. It is important to note that since FEBio will store all non-converged states to the plot file, this file may become very large in a short number of time steps. An alternative approach is to use the *plot* command to write out select non-converged states.

2.8.3. Restarting a Run

When the creation of a restart file is requested, the analysis can be restarted from the last converged timestep. This is useful when the run terminated unexpectedly or when the user wishes to modify some parameters during the analysis. To request a restart file, simply set the appropriate option in the control section of the input file. This will generate a *dump* file which then can be used to restart the analysis. See Chapter 5 for more details.

To restart an analysis, use the *–r* command line option. This option requires a filename as a parameter, and this name can be either the name of a dump file or the name of a restart input file. The latter case is a text file that allows the user to redefine some parameters when restarting the run. The format of this file is described in Chapter 5.

2.8.4. Input File Checking

The *–c* option allows the user to stop FEBio after the initial data checking is done. This way, potential input errors can be spotted without running the actual problem.

Example:

```
> febio –i input.feb –c
```

Chapter 3 Free Format Input

This chapter describes the XML-based input format used by FEBio. Since this format follows standard XML conventions, the files can be viewed with any file viewer that supports XML files. Since the free format input file is a text file, it can be edited with any text editor.

An XML file is composed of a hierarchical list of *elements*. The first element is called the *root element*. Elements can have multiple *child elements*. All elements are enclosed by two *tags:* a tag defining the element and an *end tag*. A simple example of an XML file might look like this:

```
<root>
      <child>
            <subchild> ... </subchild>
      </child>
</root>
```

The *value* of an element is enclosed between the name and the end tag.

```
<element> here is the value </element>
```

Note that the XML format is case-sensitive.

XML elements can also have *attributes* in name/value pairs. The attribute value must always be quoted.

```
<element attr="value">...</element>
```

If an XML element has no value, an abbreviated syntax can be used. The following two lines are identical.

```
<element [attribute list]></element>
```

or

```
<element [attribute list]/>
```

Comments can be added as follows.

```
<!-- This is a comment -->
```

The first line in the document – the XML declaration – defines the XML version and the character encoding used in the document. An example can be:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

3.1. Free Format Overview

The free format organizes the FEBio input data into hierarchical XML elements. The root element is called *febio_spec*. This root element also defines the format version number (Note that FEBio and the input format specification follow different version numberings). This document describes version 2.0 of the FEBio specification* (see section 3.2 below for more details on the different input specification formats). The root element will therefore be defined as follows:

```
<febio_spec version="2.0">
      <!-- contents of file -->
</febio_spec>
```

The different sections introduced in this chapter are child elements of this root element. The following sections are currently defined:

- *Module*: defines the physics module for solving the model.
- *Control*: specifies control and solver parameters.
- *Globals*: Defines the global variables in the model
- *Material*: Specifies the materials used in the problem and the material parameters.
- *Geometry*: Defines the geometry of the problem, such as nodal coordinates and element connectivity.
- *Initial*: Defines initial conditions for dynamic problems, such as initial velocities, and for transient quasi-static problems.
- *Boundary*: Defines the boundary conditions that are applied on the geometry.
- *Loads*: Defines the loads applied to the model. This includes nodal loads, boundary loads and volume loads (or sources for heat transfer problems).
- *Contact*: This section defines all contact interfaces.
- *LoadData*: Defines the load curve data.
- *Ouput*: Defines additional data that is to be stored.
- *Step*: Defines different analysis steps, where in each analysis the boundary, loads, contact and initial conditions can be redefined.

The current format specification expects the different sections of the input file to be listed in the same order as given above. Not all sections are required. Empty sections can be omitted and some are optional. A minimal file must contain at least the Control, Material and Geometry sections. The rest of this chapter describes each of these sections in more detail.

3.2. Format Specification Versions

This document describes version 2.0 of the FEBio input specification. This format differs in several aspects from the previous 1.x versions of the input specification. This section describes the major changes between the different versions.

* FEBio continues to read some older formats, but they are considered to be obsolete.

- **Version 2.0**: The latest and recommended version of the FEBio input specification described in this document. This is the first major revision of the input file format and redefines many of the file sections: The *Elements* section uses a different organization. Elements are now grouped by material and element type. Multiple *Elements* sections can now be defined to create multiple parts. Surfaces can now be defined in the *Geometry* section and referenced by boundary conditions and contact definitions. A new *Contact* section contains all the contact definitions. A new *Discrete* section was defined that contains all the materials and definitions of the discrete elements (e.g. springs). The *Boundary* section is also redesigned.
- **Version 1.3**: This was an experimental version that redefined the *Geometry* section, but was later abandoned in favor of version 2.0. This version is **no longer supported**.
- **Version 1.2**: A *Loads* section was added and all surface and body loads are now defined in this section instead of the *Boundary* section. This version is **still supported** but considered obsolete.
- **Version 1.1**: Rigid body constraints are no longer defined in the rigid material definition but instead placed in a new *Constraints* section. This version is **no longer supported**.
- **Version 1.0**: The original input format specification. This version is **no longer supported**.

As of FEBio 2.0, only versions 1.2, 2.0 and up are supported. Version 1.2 is considered obsolete and it is highly recommended to convert older files to the new 2.0 specification for use with newer versions of FEBio. This can be done for instance using PreView.

3.3. Multiple Input Files

FEBio supports distributing the model definition across multiple input files. This can greatly facilitate defining large, complex models and allows the re-use of model input files without the need to create the entire model input file from scratch. When using multiple input files to define a model, you must create a master input file that will reference all the other input files. This master file will be used to run the model in FEBio.

All the main sections defined in section 3.1 support the *from* attribute which can be used to load the section from another input file. For example, to load the *Material* section from the file mat.feb, defining the *Material* section in the master input file as follows.

```
<Material from="mat.feb"/>
```

FEBio will now read the *Material* section from this child file. The child file must be a valid FEBio input file, meaning it must begin with the *febio_spec* root section, but does not have to be complete. For example, the file mat.feb only needs to define the Material section. However, the child file may contain other sections. In that case, only the section referenced in the master file will be read from the child file. For example, if the file *in.feb* contains both the *Material* and the *Geometry* section, the master file can read both these sections as follows.

```
<Material from="in.feb"/>
<Geometry from="in.feb"/>
```

To give a more concrete example, assume that the *Material*, *Geometry*, and *Boundary* sections are defined in the files *mat.feb*, *geom.feb*, and *bc.feb* respectively. The master input file could then look like the following.

```
<febio_spec version="2.0">
      <Control>
            <time_steps>10</time_steps>
            <step_size>0.1</step_size>
      </Control>
      <Material from="mat.feb"/>
      <Geometry from="geom.feb"/>
      <Boundary from="bc.feb"/>
</febio_spec>
```

Notice that the *Control* section is still defined in the master file. The master file can contain a combination of explicit section definitions and referenced sections using the *from* attribute. As mentioned above, the master file is used to run the model in FEBio. So, if the master file is called *model.feb* then the model is run as follows.

```
>febio –i model.feb
```

When FEBio parses the master file it will automatically parse the referenced child files it encounters in the master input file.

3.4. Module Section

The module section defines the type of analysis to perform with FEBio. This section must be defined as the first section in the input file. It takes on the following format:

```
<Module type="[type]"/>
```

where *type* can be any of the following values:

type	Description
`solid`	Structural mechanics analysis: quasi-static or dynamic
`biphasic`	Biphasic analysis: steady-state or transient
`solute`	Biphasic analysis including solute transport: steady-state or transient
`multiphasic`	Multiphasic analysis including solute transport and chemical reactions.
`heat`	Heat transfer analysis: steady-state or transient

For example:

```
<febio_spec version="2.0">
      <Module type="solid"/>
      <!-- rest of file -->
</febio_spec>
```

Note 1: In version 1.2 the Module section was optional. If omitted it was assumed that the *solid* module was used. In version 2.0 the Module section is required and must be the first section in the file.

Note 2: Previous versions of FEBio allowed you to run a poroelastic (now called biphasic) problem by simply defining a poroelastic material. This is no longer possible. You need to define the proper Module section to run a biphasic analysis. If you have a file that no longer works as of version 1.4 of FEBio, you'll need to insert the following Module section in the file as the first section of the file.

```
<febio_spec version="2.0">
      <Module type="biphasic"/>
      <!-- rest of the file unaltered -->
</febio_spec>
```

3.5. Control Section

The control section is defined by the *Control* element. This section defines all parameters that are used to control the evolution of the solution as well as parameters for the nonlinear solution procedure. These parameters are defined as child elements of the *Control* element. The parameters depend somewhat on the analysis as defined by the *Module* section. Many parameters are common to all types of analysis, so they are listed first.

3.5.1. Common Parameters

The following parameters are common for all analysis. If not specified they are assigned default values, which are found in the last column. An asterisk (*) after the name indicates a required parameter. The numbers behind the description refer to the comments following the table.

Parameter	**Description**	**Default**
`title`	Title of problem	(none)
`time_steps*`	Total number of time steps. (= *ntime*)(1)	(none)
`step_size*`	The initial time step size. (= *dt*) (1)	(none)
`dtol`	Convergence tolerance on displacements (2)	0.001
`etol`	Convergence tolerance on energy (2)	0.01
`rtol`	Convergence tolerance on residual (2)	0 (disabled)
`lstol`	Convergence tolerance on line search (3)	0.9
`time_stepper`	Enable the auto time stepper (4)	(off)
`max_refs`	Max number of stiffness reformations (5)	15
`max_ups`	Max number of BFGS stiffness updates (5)	10
`optimize_bw`	Optimize bandwidth of stiffness matrix (6)	0
`restart`	Generate restart flag (7)	0
`plot_level`	Sets the level of state dumps to the plot file (8)	PLOT_MAJOR_ITRS
`cmax`	Set the max condition number for the stiffness matrix (9)	1e+5
`analysis`	Sets the analysis type (10)	static
`print_level`	Sets the amount of output that is generated on screen (11)	PRINT_MINOR_ITRS
`linear_solver`	Set the linear solver (12)	skyline
`min_residual`	Sets minimal value for residual tolerance (13)	1e-20
`integration`	Set the integration rule for a particular element (14)	N/A

Comments:

1. The total running time of the analysis is determined by *ntime* * *dt*. Note that when the auto-time stepper is enabled (see below), the actual number of time steps and time step size may be different than specified in the input file. However, the total running time will always be determined by *ntime* * *dt.*

2. FEBio determines convergence of a time step based on three convergence criteria: displacement, residual and energy (that is, residual multiplied by displacement). Each of these criteria requires a tolerance value that will determine convergence when the relative change will drop below this value. For example, a displacement tolerance of ε means that the ratio of the displacement increment (i.e. the solution of the linearized FE equations, $\Delta\mathbf{U} = -\mathbf{K}_k^{-1}\mathbf{R}_k$) norm at the current iteration *k+1* to the norm of the total displacement ($\mathbf{U}_{k+1} = \mathbf{U}_k + \Delta\mathbf{U}$) must be less than ε:

$$\frac{\|\Delta\mathbf{U}\|}{\|\mathbf{U}_{k+1}\|} < \varepsilon$$

For the residual and energy norms, it is the ratio of the current residual norm (resp. energy norm) to the initial one that needs to be less than the specified convergence tolerance.

To disable a specific convergence criterion, set the corresponding tolerance to zero. For example, by default, the residual tolerance is zero, so that this convergence criterion is not used.

3. The *lstol* parameter controls the scaling of the vector direction obtained from the line search. A line search method is used to improve the convergence of the nonlinear (quasi-) Newton solution algorithm. After each BFGS or Newton iteration, this algorithm searches in the direction of the displacement increment for a solution that has less energy (that is, residual multiplied with the displacement increment) than the previous iteration. In many problems this will automatically be the case. However, in some problems that are very nonlinear (e.g. contact), the line search can improve convergence significantly. The line search can be disabled by setting the *lstol* parameter to zero, although this is not recommended.

4. If the *time_stepper* parameter is defined it will enable the auto time-stepper, which will adjust the new time step size based on convergence information from the previous time step. The following sub-elements may also be defined, although all are optional. Note that these are sub-elements of the *time_stepper* element and not of the *Control* element.

Parameter	**Description**	**Default**
`dtmin`	Minimum time step size	dt/3
`dtmax`	Maximum time step size	dt*3
`max_retries`	Maximum nr. of retries allowed per time step	5
`opt_iter`	Optimal number of iterations	10

The *dtmin* and *dtmax* values are used to constrain the range of possible time step values. The *opt_iter* defines the estimated optimal number of quasi-Newton iterations. If the actual number of iterations is less than or equal to this value the time step size is increased, otherwise it is decreased.

When a time step fails (e.g. due to a negative jacobian), FEBio will retry the time step with a smaller time step size. The *max_retries* parameter determines the maximum number of times a timestep may be retried before FEBio error terminates. The new time step size is determined by the ratio of the previous time step size and the *max_retries* parameter. For example, if the last time step size is 0.1 and *max_retries* is set to 5, then the time step size is adjusted by 0.02: The first retry will have a step size of 0.08; the next will be 0.06, and so on.

The user can specify a load curve for the *dtmax* parameter. This load curve is referred to as the *must-point* curve and serves two purposes. Firstly, it defines the value of the *dtmax* parameter as a function of time. This can be useful, for instance, to enforce smaller time steps during rapid loading or larger time steps when approaching steady-state in a transient analysis. Secondly, the time points of the *dtmax* loadcurve define so-called *must-points*. A must-point is a time point where FEBio must pass through. This is useful for synchronizing the auto-time stepper with the loading scenario. For instance, when loading starts at time 0.5, adding a must-point at this time will guarantee that the timestepper evaluates the model at that time. In conjunction with the `PLOT_MUST_POINT` value of the *plot_level* parameter, this option can also be used to only write results to the plotfile at the specified time points. Consider the following example.

```
<dtmax lc="1">0.1</dtmax>
...
<loadcurve id="1" type="step">
      <loadpoint>0,0</loadpoint>
      <loadpoint>0.5, 0.1</loadpoint>
      <loadpoint>1.0, 0.2</loadpoint>
</loadcurve>
```

This example defines load curve 1 as the *must-point* curve. This curve defines three points where FEBio will pass through (namely 0, 0.5 and 1.0). The values of each time point is the value of the maximum time-step size (*dtmax*). Since the curve is defined as a step-function, each value is valid up to the corresponding time-point. Thus, between time 0 and time 0.5, the maximum time step value is 0.1. Between 0.5 and 1.0 the maximum time step value is 0.2. If the *plot_level* parameter is set to `PLOT_MUST_POINTS`, then only the three defined time points will be stored to the plotfile.

5. The *max_ups* and *max_refs* parameters control the BFGS method that FEBio uses to solve the nonlinear FE equations. In this method the global stiffness matrix is only calculated at the beginning of each time step. For each iteration, a matrix update is then done. The maximum number of such updates is set with *max_ups*. When FEBio reaches this number, it reforms the global stiffness matrix (that is, it recalculates it) and factorizes it, essentially taking a "full Newton" iteration. Then FEBio continues with BFGS iterations. The *max_refs* parameter is used to set the maximum of such reformations FEBio can do, before it fails the timestep. In that case, FEBio will either terminate or, if the auto-time stepper is enabled, retry with a smaller time step size.

Note that when max_ups is set to 0, FEBio will use the Full-Newton method instead of the BFGS method. In other words, the stiffness matrix is reformed for every iteration. In this case it is recommended to increase the number of *max_refs* (to e.g. 50), since the default value might cause FEBio to terminate prematurely when convergence is slow.

6. The *optimize_bw* parameter enables bandwidth minimization for the global stiffness matrix. This can drastically decrease the memory requirements and running times when using the skyline solver. It is highly recommended when using the skyline solver.

   ```
   <optimize_bw>1</optimize_bw>
   ```

   When using a different linear solver (e.g., pardiso or SuperLU), the bandwidth optimization can still be performed if so desired. However, for these solvers there will be little or no effect since these solvers are not as sensitive to the bandwidth as the skyline solver.

7. The *restart* parameter can be used to generate a restart dump file. To activate it, specify a non-zero value. A filename may be specified as an option. If the filename is omitted, a default name will be provided. Note that this will only generate the binary dump file that is needed to restart the analysis. To override certain parameters before restarting, create a restart input file. FEBio does not generate this file automatically so the user needs to create that file manually. See chapter 4 on the format of the restart input file.

   ```
   <restart file="out.dmp">1</restart>
   ```

8. The *plot_level* allows the user to control exactly when the solution is to be saved to the plot file. The following values are allowed:

Value	**Description**
`PLOT_NEVER`	Don't save the solution
`PLOT_MAJOR_ITRS`	Save the solution after each converged timestep
`PLOT_MINOR_ITRS`	Save the solution for every quasi-Newton iteration
`PLOT_MUST_POINTS`	Only save the solution at the must points

   The `PLOT_MUST_POINTS` option must be used in conjunction of a *must-point* curve. See the comments on the *dtmax* parameter for more information on must-point curves. When the *plot_level* option is set to `PLOT_MUST_POINTS`, only the time-points defined in the must-point curve are stored to the plotfile.

9. When the condition number of the stiffness matrix becomes too large, inversions of the stiffness matrix become inaccurate. This will negatively affect the convergence of the quasi-Newton or Newton solution algorithm. FEBio monitors the condition number of the BFGS stiffness update and when it exceeds *cmax* it reforms the stiffness matrix.

   ```
   <cmax>1e5</cmax>
   ```

10. The *analysis* element sets the analysis type. Currently, FEBio defines three analysis types: (quasi-)static, steady-state, and dynamic. In a quasi-static analysis, inertial effects are ignored and an equilibrium solution is sought. Note that in this analysis mode it is still possible to simulate time dependant effects such as viscoelasticity. In a dynamic analysis the inertial effects are included.

Value	**Description**
`static`	(quasi-) static analysis
`steady-state`	steady-state response of a transient (quasi-static) biphasic, biphasic-solute, or triphasic analysis
`dynamic`	dynamic analysis.

11. The *print_level* allows the user to control exactly how much output is written to the screen. The following values are allowed:

Value	**Description**
`PRINT_NEVER`	Don't generate any output
`PRINT_PROGRESS`	Only print a progress bar
`PRINT_MAJOR_ITRS`	Only print the converged solution
`PRINT_MINOR_ITRS`	Print convergence information during equilibrium iterations
`PRINT_MINOR_ITRS_EXP`	Print additional convergence info during equilib. iterations

12. The linear solver is by default set in the configuration file. Since this allows FEBio to run the same file on different platforms (which may support different linear solvers), this is the recommended way to set the linear solver. However, if the need arises, the user can also override the default linear solver, by explicitly specifying the *linear_solver* parameter in the FEBio input file. The following linear solvers are implemented although they might not all be supported on all platforms. The only solver that is guaranteed to work on all platforms is the skyline solver.

Name	**Description**	**Sym**	**ASym**	**MT**
skyline	The default skyline solver	●		
pardiso	The preferred solver for most platforms	●	●	●
superlu	The single-threaded SuperLU solver	●	●	
superlu_mt	The multi-threader SuperLU solver	●	●	●
wsmp	The WSMP solver	●	●	●
lusolver	Full-matrix solver	●		

13. If no force is acting on the model, then convergence might be problematic due to numerical noise in the calculations. For example, this can happen in a displacement driven contact problem where one of the contacting bodies is moved before initial contact is made. When this happens, the residual norm will be very small. When it drops below the tolerance set by *min_residual*, FEBio will assume that there is no force acting on the system and will converge the time step.

14. You can override FEBio's default integration rule for specific element classes. For each element class, define a *rule* element in which you set the integration rule.

```
<integration>
      <rule elem="<elem>">VALUE</rule>
      <!-- repeat rules for other elements -->
</integration>
```

The *elem* attribute value defines for which element class you wish to override the default integration rule and can have any of the following values.

elem	**Description**
`hex8`	8-node hexahedral element
`tet4`	4-node linear tetrahedral element
`tet10`	10-node quadratic tetrahedral element
`tri3`	3-node linear triangles (e.g. for contact)
`tri6`	6-node quadratic triangles

The values of the rule elements depend on the *elem* attribute. The tables below show the available integration rules for the different element types. The values marked with an asterisk (*) are the default.

- For the *hex8* element, the following values are defined.

hex8	**Description**
GAUSS8*	Gaussian integration using 2x2x2 integration points.
POINT6	Alternative integration rule for bricks using 6 integration point

- For the *tet4* element, the following values are allowed.

tet4	**Description**
GAUSS4	Gaussian integration rule using 4 integration points.
GAUSS1*	Gaussian integration rule using one integration point.
UT4	Nodally integrated tetrahedron. (1)

Comments:
1. The UT4 is a special formulation for tetrahedral elements that uses a nodally averaged integration rule, as proposed by Gee et al [2]. This formulation requires additional parameters. To override the default values, use the following alternative syntax:

```
<rule elem="tet4" type="UT4">
      <alpha>0.05</alpha>
      <iso_stab>0</iso_stab>
</rule>
```

The *alpha* parameter defines the amount of "blending" between the regular tet-contribution and the nodally integrated contribution. The value must be between 0 and 1, where 0 means no contribution from the regular tet and 1 means no contribution from the nodally averaged tet. The *iso_stab* parameter is a flag that chooses between two slightly different formulations of the nodally integrated tet. When set to 0, the stabilization is applied to the entire virtual work, whereas when set to 1 the stabilization is applied only to the isochoric part. See the Theory Manual for a detailed description of this formulation.

- For the *tet10* element, the following integration rules are supported.

tet10	**description**
GAUSS4*	Gaussian integration rule using 4 integration points
GAUSS8	Gaussian integration rule using 8 integration points
LOBATTO11	Gauss-Lobatto integration rule using 11 integration points

- For the *tri3* element, the following integration rules are supported.

tri3	**description**
GAUSS1	Gaussian integration with one integration point
GAUSS3*	Guassian integration with three integration rules.

- For the *tri6* element, the following integration rules are supported.

tri6	**description**
GAUSS3*	Gaussian integration with 3 integration points
GAUSS6	Gaussian integration with 6 integration points (1)
GAUSS4	Gaussian integration with 4 integration points
GAUSS7	Gaussian integration with 7 integration points
LOBATTO7	Gauss-Lobatto integration with 7 integration points.

Comments:
1. This rule has only nonzero weights at the edge nodes, which effectively reduces this rule to a 3-node rule.

3.5.2. Parameters for *Biphasic* Analysis

A biphasic analysis is defined by using the *biphasic* type in Module section. Since a biphasic analysis couples a fluid problem to a solid mechanics problem, all control parameters above can be used in a biphasic analysis. In addition, the following parameters can be defined:

Parameter	Description	Default
ptol	Specify the fluid pressure convergence tolerance	0.01
symmetric_biphasic	Choose between a symmetric or nonsymmetric formulation	1

3.5.3. Parameters for *Solute* Analysis

When the type attribute of the Module section is set to *solute* or *triphasic*, an analysis is solved that includes solute transport. All parameters for a biphasic analysis can be used (including the ones for a structural mechanics analysis). In addition, the following parameters can be specified:

Parameter	Description	Default
ctol	Specify the concentration convergence tolerance	0.01

3.5.4. Parameters for *Heat* Analysis

A heat analysis uses the parameters defined in section 3.4.1. However, not all parameters have an effect. In particular, any parameter related to the auto-time stepping capability of FEBio or the nonlinear solution strategy is ignored.

3.6. Globals Section

The *Globals* section is used to define some global variables, such as global constants, solute data and solid bound molecule data.

3.6.1. Constants

Global constants currently include the universal gas constant R [**F**·**L**/**n**·**T**], absolute temperature θ [**T**], and Faraday constant F_c [**Q**/**n**]. These constants must be expressed in units consistent with the rest of the analysis:

```
<Globals>
      <Constants>
            <R>8.314e-6</R>
            <T>298</T>
            <Fc>96485e-9</Fc>
      </Constants>
</Globals>
```

3.6.2. Solutes

In biphasic-solute, triphasic and multiphasic analyses, a unique identifier must be associated with each solute in order to enforce consistent nodal degrees of freedom across boundaries of different materials. This unique identification is achieved by listing each solute species that appears in the entire finite element model and associating it with a unique *id*, *nam,e* charge number z^α, molar mass M^α, and density ρ_T^α:

```
<Globals>
      <Solutes>
            <solute id="1" name="Na">
                  <charge_number>1</charge_number>
                  <molar_mass>22.99</molar_mass>
                  <density>0.97<density>
            </solute>
            <solute id="2" name="Cl">
                  <charge_number>-1</charge_number>
                  <molar_mass>35.45</molar_mass>
                  <density>3.21<density>
            </solute>
            <solute id="3" name="Glc">
                  <charge_number>0</charge_number>
                  <molar_mass>180.16</molar_mass>
                  <density>1.54<density>
            </solute>
      </Solutes>
</Globals>
```

These solute identification numbers should be referenced in the *sol* attribute of solutes when defining a biphasic-solute (Section 4.5.2), triphasic (Section 4.6.2) or multiphasic material.

The molar mass and density of solutes are needed only when solutes are involved in chemical reactions. When not specified, default values for these properties are set to 1.

3.6.3. Solid-Bound Molecules

In multiphasic analyses with chemical reactions involving solid-bound molecules, a unique identifier must be associated with each such molecule in order to enforce consistent properties across the entire model. This unique identification is achieved by listing each solid-bound species that appears in the entire finite element model and associating it with a unique *id*, *name*, charge number, molar mass and density:

```
<Globals>
      <SolidBoundMolecules>
            <solid_bound id="1" name="CS">
                  <charge_number>-2</charge_number>
                  <molar_mass>463.37</molar_mass>
                  <density>1.5<density>
            </solid_bound >
      </SolidBoundMolecules >
</Globals>
```

The *id* number should be referenced in the *sbm* attribute of solid-bound molecules when included in the definition of a multiphasic material (Section 4.6). The charge number is used in the calculation of the fixed charge density contributed by this solid-bound molecule to the overall solid matrix fixed-charge density. The density is used in the calculation of the contribution of this molecule to the referential solid volume fraction. The density and molar mass are used in the calculation of the molar volume of this molecule in chemical reactions.

3.7. Material Section

The material section is defined by the *Material* element. This section defines all the materials and material parameters that are used in the model. A material is defined by the *material* child element. This element has two attributes: *id*, which specifies a number that is used to reference the material, and *type*, which specifies the type of the material. The *material* element can also have a third optional attribute called *name*, which can be used to identify the material by a text description. A material definition might look like this:

```
<material id="1" type="isotropic elastic">
```

Or, if the optional *name* attribute is present:

```
<material id="2" type="rigid body" name="femur">
```

The material name is required for parameter optimizations since the material parameters are resolved using the material's name.

The material parameters that have to be entered depend on the material type. A complete list of available materials is provided in Chapter 4.

3.8. Geometry Section

The geometry section contains all the geometry data, including nodal coordinates and element connectivity. It has the following sub-sections:

- *Nodes*: contains nodal coordinates.
- *Elements*: contains element connectivity.
- *ElementData*: contains additional element data.

3.8.1. Nodes Section

The nodes section contains all the nodal coordinates. Repeat the following XML-element for each node:

```
<node id="n">x,y,z</node>
```

The *id* attribute is a unique number that identifies the node. This *id* is used as a reference in the element connectivity section. The *id*s do not have to be in order, but the lowest id *must* be 1 and numbers cannot be skipped. (So if there is a node 4 and a node 6, there must also be a node 5 somewhere).

3.8.2. Elements Section

The *Elements* sections contain a list of the element connectivity data. Multiple *Elements* sections can be defined. The *Elements* section can have the following attributes.

Attribute	**Description**
mat	material identifier
type	element type
name	unique name that identifies this domain (optional)

Each *Elements* section contains multiple *elem* elements that define the element connectivities. Each *elem* tag has a *id* attribute that defines the element number. For example, the following *Elements* section defines a list of hexahedral elements:

```
<Elements type="hex8" mat="1" name="part1">
      <elem id="1">1,2,3,4,5,6,7,8</elem>
      <elem id="2">1,2,3,4,5,6,7,8</elem>
      <...>
</Elements>
```

FEBio classifies elements in two categories, namely *solids* and *shells*.

3.8.2.1. Solid Elements

The following solid element types are defined:

hex8	8-node trilinear hexahedral element

hex20	20-node quadratic elements
penta6	6-node linear pentahedral (wedge) element
tet4	4-node linear tetrahedral element
tet10	10-node quadratic tetrahedral element

The node numbering has to be defined as in the figure below.

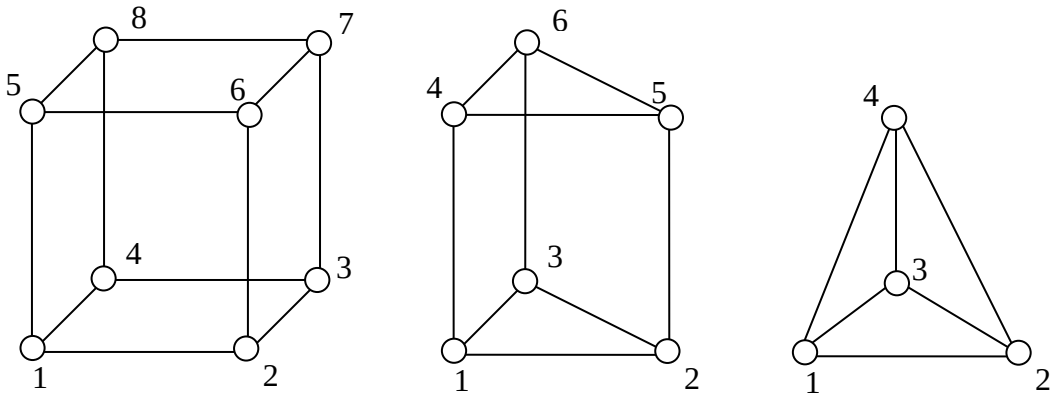


Figure 3-1. Node numbering for solid elements.

3.8.2.2. Shell Elements

FEBio currently supports a 4-noded quadrilateral and a 3-noded triangular shell element.

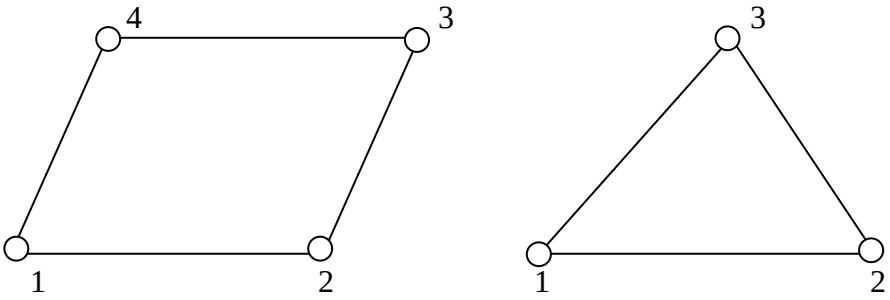


Figure 3-2. Node numbering for shell elements.

3.8.2.3. Surface Elements

In many cases the surface of some geometry (or part of it) is required. For example, pressure forces are applied to the surface. For this reason surface elements have to be defined. The following surface elements are available:

- *quad4*: 4-node quadrilateral element
- *quad8*: 8-node serendipity quadrilater element
- *tri3*: 3-node triangular element
- *tri6*: 6-node quadratic triangular element

The value for the surface element is the nodal connectivity:

```
<quad4 id="n">n1,n2,n3,n4</quad4>
<tri3 id="n">n1,n2,n3</tri3>
```

Surface elements cannot overlap element boundaries. That is, the surface element must belong to a specific element. Surface elements do not contribute to the total number of elements in the mesh. They are also not to be confused with shell elements.

3.8.3. Element Data Section

Additional element data can be specified in the *ElementData* section. The data in this section usually represents material data that can vary from element to element. The following element properties can be defined:

Property	**Description**
`fiber`	Specify a local fiber direction
`thickness`	Specify the shell element thickness
`MRVonMisesParameters`	Von Mises Fibers coefficients kf and tp

Note that defining a *fiber* direction here will override the fiber distribution specified in the material definition and only for those elements specified in the *ElementData* section. In other words, you can define a particular fiber distribution in the material section and then override the fiber direction for a subset of elements. See section 4.1.1 for more information on defining fiber directions for transversely isotropic materials.

The thickness parameter can only be specified for shell elements. The value of this parameter is the shell thickness at each of the shell nodes. For example, for a four-node shell:

```
<element id="n">
      <thickness>0.01,0.01,0.01,0.01</thickness>
</element>
```

To specify a fiber direction for a particular element, enter the following xml-element in the *ElementData* section:

```
<element id="n">
      <fiber>1,0,0</fiber>
</element>
```

The *id* attribute identifies the element to which this fiber direction is applied. The *fiber* element defines a vector in global coordinates. This vector does not have to be of unit length since it is automatically normalized by FEBio.

The VonMisesParameters option can only be used for an element made of Von Mises distributed fibers material.
Defining the Von Mises parameters (concentration factor kf and preferred fiber orientation tp) here will override the parameters specified in the *Material* section, but only for the elements specified in the *ElementData* section.
The syntax to specify the Von Mises parameters at element n in the *ElementData* section is :

```
<element id="n">
      <MRVonMisesParameters>7.0,2.1</VonMisesParameters>
</element>
```

Note that the parameters are first kf, then tp in radian.
The local coordinate system of the plane of the fibers is defined by the directions given in mat_axis. If mat_axis is defined for element n in the *ElementData* section, the corresponding directions will be used, otherwise the directions specified in mat_axis in the *Material* section will be taken.

3.8.4. Surface Section

The *Surface* section allows users to define surfaces. These surfaces can then later be used to define the boundary conditions and contact definitions. A surface definition is followed by a list of surface elements, following the format described in section 3.7.2.3. .

```
<Surface name="named_surface">
      <quad4 id="1">1,2,3,4</quad4>
      <...>
</Surface>
```

The *Surface* takes one required attribute, namely the *name*. This attribute sets the name of the surface. This name will be used later to refer back to this surface.

3.8.5. NodeSet Section

The *NodeSet* section allows users to define node sets. These node sets can then later be used in the definition of the fixed and prescribed boundary conditions. A node set is defined by the *NodeSet* tag. This tag takes one required attribute, *name*, which defines the name of the node set. The value of this tag is a list of nodes that defines the node set.

```
<NodeSet name="nodeset1">1:5,6,7,8:12:2</NodeSet>
```

The list of nodes is a comma separated list of entries, where an entry is either a node or a range of nodes. A range of nodes is defined using the following syntax.

```
n1:n2
```

where n1 is the start node and n2 is the end node, or

```
n1:n2:n3
```

where n1 is the start node, n2 is the end node and n3 is the increment. For example, 1:5, is the equivalent of 1,2,3,4,5 and 1:5:2 is the equivalent of 1,3,5.

Note that it is allowed to separate large node set definitions over multiple lines. For example,

```
<NodeSet name="nodeset1">
      1,2,3,4,5,
      6:120:3,
      1000,1001
</NodeSet>
```

3.9. Initial Section

The *Initial* section defines all initial conditions that may be applied to the analysis. Several initial conditions are available: nodal velocities, nodal effective pressure, and nodal effective concentrations.

3.9.1. Initial Nodal Velocities

Initial nodal velocities are needed in dynamic structural mechancis analyses (see Section 3.4.1). They can be prescribed using the *velocity* sub-element.

Example:

```
<Initial>
      <velocity>
            <node id="1"> 0.5, 1.0, 0</node>
            ...
      </velocity>
</Initial>
```

3.9.2. Initial Nodal Effective Fluid Pressure

Initial nodal effective fluid pressures are needed in biphasic-solute and triphasic analyses. They can be prescribed using the *fluid_pressure* sub-element.

Example:

```
<Initial>
      <fluid_pressure>
            <node id="1"> -2.477e-3</node>
            ...
      </fluid_pressure>
</Initial>
```

3.9.3. Initial Nodal Effective Concentration

Initial nodal effective solute concentrations are needed in biphasic-solute and triphasic analyses. They can be prescribed using the *concentration* sub-element. An optional *sol* attribute may be provided to identify the solute to which this initial condition applies, referencing the corresponding list in the Globals section (Section 3.5.2). If not specified, the default is 1.

Example:

```
<Initial>
      <concentration sol="1">
            <node id="1"> 1.0</node>
            ...
      </concentration>
</Initial>
```

3.10. Boundary Section

The *Boundary* section defines all fixed and prescribed boundary conditions that may be applied to the geometry.

3.10.1. Prescribed Nodal Degrees of Freedom

Nodal degrees of freedom (dof) can be prescribed using the *prescribe* sub-element. This element has two required attributes (*bc* and *lc*) and one optional attribute (*relative*).

```
<prescribe bc="x" lc="1" [type="type"] [set="nodeset"] [scale="scl"]>
      <node id="n">2.3</node>
      ...
</prescribe>
```

The *bc* attribute specifies the particular degree of freedom. The following values are allowed:

`x`: apply displacement in *x*-direction
`y`: apply displacement in *y*-direction
`z`: apply displacement in *z*-direction
`u`: apply rotation about x-direction
`v`: apply rotation about y-direction
`w`: apply rotation about *z*-direction
`p`: apply prescribed effective fluid pressure
`t`: apply prescribed temperature (heat transfer analysis)
`c`: apply prescribed effective solute concentration (biphasic analysis)
`c[`*n*`]`: apply prescribed effective solute concentration on solute *n*

For solutes, replace "*n*" with the solute id from the global solute table (Section 3.5.2); for example, "`c2`".

The loadcurve is specified with the *lc* attribute. The value of the *lc* attribute is the ID of the loadcurve that is defined in the *LoadData* section of the input file.

The optional *type* attribute in the *prescribe* tag allows users to choose between absolute (default) and relative boundary conditions. Absolute boundary conditions assign the specified value to the desired nodal degree of freedom. Relative boundary conditions are meaningful only in multi-step analyses. When a nodal degree of freedom is specified to be relative at a particular step, the value prescribed for that node is superposed over the value of that degree of freedom at the end of the preceding step.

The node list can be defined either explicitly by defining a *node* tag for each node. The *id* attribute in the *node* tag indicates to which node this prescribed dof is applied. The value of the *node* element (e.g. 2.3 in the example above) is the value for the prescribed displacement. Note that this value scales the value determined by the loadcurve.

Alternatively, the node list can be defined through the *set* attribute. In this case the name of the set refers to a previously defined node set. The optional attribute *scale* can now be used to define the scale value. This value scales the value determined by the load curve. For example,

```
<prescribe bc="x" lc="1" set="nodeset1" scale="1.0"/>
```

Here, *nodeset1* is the name of a node set defined in the Geometry section. See section 3.7.5 for more information on how to define node sets.

3.10.2. Fixed Nodal Degrees of Freedom

Degrees of freedom that are fixed (in other words, constrained, or are always zero) can be defined using the *fix* element:

```
<fix bc="x" [set="nodeset"]>
      <node id="n"/>
      ...
</fix>
```

Although the *prescribe* element with a value of zero for the *node* tags can also be used to fix a certain nodal degree of freedom, the user should use the *fix* element whenever possible, since this option causes the equation corresponding to the constrained degree of freedom to be removed from the linear system of equations. This results in fewer equations that need to be solved for and thus reduces the run time of the FE analysis.

The nodes can be defined explicitly by adding a list of *node* tags. The ID attribute of each node tag then defines the node to which to apply the constraint. For example,

```
<fix bc="x">
      <node id="1"/>
      ...
</fix>
```

Alternatively, the node list can be defined through the *set* attribute. In this case, the value of the set attribute is the name of a previously defined node set.

```
<fix bc="x" set="nodeset1"/>
```

Here, *nodeset1* refers to a node set that is defined in the Geometry section of the input file. See section 3.7.5 for more information on defining node sets.

3.11. Loads Section

The *Loads* section defines all nodal, surface and body loads that can be applied to the model.

3.11.1. Nodal Loads

Nodal loads are applied by the *nodal_load* element. When the loads are applied to displacement degrees of freedom, the forces always point in the same direction and do not deform with the geometry (i.e. they are non-follower forces). For other degrees of freedom they define a constant normal flux.

```
<nodal_load bc="x" lc="1">
      <node id="n">3.14</node>
      ...
</force>
```

The *id* attribute indicates to which node this prescribed dof is applied.

The *bc* attribute gives the degree of freedom. The following values are allowed:

`x`: apply force in *x*-direction
`y`: apply force in *y*-direction
`z`: apply force in *z*-direction
`p`: apply normal volumetric fluid flow rate
`c`*n*: apply normal molar solute flow rate
`t`: apply normal heat flux (heat transfer analysis)

For solutes, replace "*n*" with the solute id from the global solute table (Section 3.5.2); for example, "`c2`". An optional *loadcurve* can be specified with the *lc* attribute. If a loadcurve is not specified, the value will be automatically ramped from a value of 0 at time *t*=0 to the value specified in the xml file at the time corresponding to the end of the analysis.

The value of the *node* element (e.g. 3.14 in the example above) is the value for the nodal force. Note that if a *loadcurve* is specified, this value scales the value determined by the *loadcurve*.

3.11.2. Surface Loads

A surface load can be applied using the *surface_load* element. This element takes one attribute, namely *type*, which defines the type of surface load that will be applied. The following sections define the different surface loads that can be applied in FEBio.

All surface loads require the *surface* element which defines the surface to which the load is applied. A surface can be defined explicitly or through a reference to a named surface defined in the *Geometry* section.

If the surface is defined explicitly, the *surface* element is followed by a list of surface facets. For instance:

```
<surface>
      <quad4 id="1">1,2,3,4</quad>
      ...
</surface>
```

To define a surface implicitly by referencing to an existing surface defined in the *Geometry* section, use the *set* attribute.

```
<surface set="surface_name"/>
```

Here, the *surface_name* is the name of a surface that is defined in the *Geometry* section (see section 3.7.4). Note that in this case the *surface* element is defined as an empty element.

3.11.2.1. Pressure Load

Pressure forces are applied to the surface of the geometry and are defined by the *surface_load* element with the type attribute set to *pressure*:

```
<surface_load type="pressure">
      <pressure [lc="1"]>1.0</pressure>
      <surface>
            ...
      </surface>
</surface_load>
```

These pressure forces are also known as *follower forces*; they change direction as the body is deformed and, in this case, are always oriented along the local surface normal. The sign convention is so that a positive pressure will act opposite to the normal, so it will compress the material. The *pressure* element defines the pressure value [**P**]. The optional parameter *lc* defines a loadcurve for the pressure evolution. If *lc* is not defined a constant pressure is applied.

3.11.2.2. Traction Load

A traction load applies a traction to a surface. The direction of the traction remains unchanged as the mesh deforms.

```
<surface_load type="traction">
      <scale [lc="1"]>1.0</scale>
      <traction>0,0,1</traction>
      <surface [set="surface_name"]>
            ...
      </surface>
</surface_load>
```

The traction vector is determined by two quantities. The direction and magnitude is defined by the *traction* element. In addition, the magnitude can be scaled using the *scale* element. An

optional load curve can be defined for the *scale* element using the *lc* attribute. This allows the traction load to become time dependant. If the *lc* attribute is omitted a constant traction load is applied.

3.11.2.3. Mixture Normal Traction

This section applies to biphasic, biphasic-solute, triphasic and multiphasic analyses. In a mixture of intrinsically incompressible solid and fluid constituents, the formulation adopted in FEBio implies that the total traction is a natural boundary condition (*FEBio Theory Manual*). If this boundary condition is not explicitly prescribed, the code automatically assumes that it is equal to zero. Therefore, boundaries of mixtures are traction-free by default.

The *mixture traction* $\mathbf{t}$ is the traction vector corresponding to the mixture (or total) stress $\boldsymbol{\sigma}$; thus $\mathbf{t}=\boldsymbol{\sigma}\cdot\mathbf{n}$, where $\mathbf{n}$ is the outward unit normal to the boundary surface. Since $\boldsymbol{\sigma}=-p\mathbf{I}+\boldsymbol{\sigma}^e$, where p is the fluid pressure and $\boldsymbol{\sigma}^e$ is the *effective stress* resulting from strains in the solid matrix, it is also possible to represent the total traction as $\mathbf{t}=-p\mathbf{n}+\mathbf{t}^e$, where $\mathbf{t}^e=\boldsymbol{\sigma}^e\cdot\mathbf{n}$ is the *effective traction*. Currently, FEBio allows the user to specify only the normal component of the traction, either $t_n=\mathbf{t}\cdot\mathbf{n}$ (the normal component of the mixture traction) or $t_n^e=\mathbf{t}^e\cdot\mathbf{n}$ (the normal component of the effective traction):

A mixture normal traction is defined by the *surface_load* element using *normal_traction* for the type attribute.

```
<surface_load type="normal_traction">
      <traction [lc="1"]>1.0</traction>
      <effective>1</effective>
      <linear>0</linear>
      <surface [set="surface_name"]>
            ...
      </surface></surface_load>
```

The *traction* element defines the magnitude of the traction force. The optional attribute *lc* defines a loadcurve that controls the time dependency of the traction force magnitude. If omitted a constant traction is applied.

The *effective* element defines whether the traction is applied as an effective traction or a total mixture traction.

The *linear* element defines whether the traction remains normal to the deformed surface or the reference surface. If set to true the traction remains normal to the reference surface. When false it defines a follower force that remains normal to the deformed surface.

The *surface* element defines the surface to which the traction is applied. It consists of child elements defining the individual surface facets.

Unlike the mixture and effective traction, the fluid pressure p is a nodal variable (see Prescribed Nodal Degrees of Freedom). There may be common situations where the user must apply a combination of related fluid pressure and traction boundary conditions. For example, if a biphasic surface is subjected to a non-zero fluid pressure p_0, the corresponding boundary conditions are $p = p_0$ and $t_n = -p_0$ (or $t_n^e = 0$). In FEBio, both boundary conditions must be applied. For example:

```
<Boundary>
	<prescribed bc="p" lc="1">
		<node id="n1">1.0</node>
		<node id="n2">1.0</node>
		<node id="n3">1.0</node>
		<node id="n4">1.0</node>
		...
	</prescribed>
	<surface_load type="normal_traction" >
		<traction lc="1">-1.0</traction>
		<effective>0</effective>
		<linear>0</linear>
		<surface>
			<quad4 id="n">n1,n2,n3,n4</quad4>
			...
		</surface>
	</surface_load>
</Boundary>
<LoadData>
	<loadcurve id="1">
		<loadpoint>0,0</loadpoint>
		<loadpoint>1,2.0</loadpoint>
	</loadcurve>
</LoadData>
```

3.11.2.4. Fluid Flux

In a biphasic mixture of intrinsically incompressible solid and fluid constituents, the **u**-p formulation adopted in FEBio implies that the normal component of the relative fluid flux is a natural boundary condition. If this boundary condition is not explicitly prescribed, the code automatically assumes that it is equal to zero. Therefore, biphasic boundaries are impermeable by default. (To implement a free-draining boundary, the fluid pressure nodal degrees of freedom should be set to zero.)

The flux of fluid relative to the solid matrix is given by the vector $\mathbf{w}$. Since viscosity is not explicitly modeled in a biphasic material, the tangential component of $\mathbf{w}$ on a boundary surface may not be prescribed. Only the normal component of the relative fluid flux, $w_n = \mathbf{w} \cdot \mathbf{n}$, represents a natural boundary condition. To prescribe a value for w_n on a surface, use:

```
<surface_load type="fluidflux">
```

```
      <flux [lc="1"]>1.0</flux>
      <linear>0</linear>
      <mixture>1</mixture>
      <surface [set="surface_name"]>
            ...
      </surface>
</fluidflux>
```

The *flux* parameter defines the flux that will be applied to the surface. The optional parameter *lc* defines a loadcurve for the normal flux evolution. If omitted a constant fluid flux is applied.

When *linear* is set to zero (default) it means that the flux matches the prescribed value even if the surface on which it is applied changes in area as it deforms. Therefore, the net volumetric flow rate across the surface changes with changes in area. This type of boundary condition is useful if the fluid flux is known in the current configuration.

When *linear* is set to non-zero it means that the prescribed flux produces a volumetric flow rate based on the undeformed surface area in the reference configuration. Therefore, the flux in the current configuration does not match the prescribed value. This type of boundary condition is useful if the net volumetric flow rate across the surface is known. For example: Let Q be the known volumetric flow rate, let A_0 be the surface area in the reference configuration (a constant). Using *"linear"* means that the user prescribes Q/A_0 as the flux boundary condition. (However, regardless of the *type*, the fluid flux saved in the output file has a normal component equal to Q/A, where A=area in current configuration.)

Prescribing w_n on a free surface works only if the nodal displacements of the corresponding faces are also prescribed. If the nodal displacements are not known a priori, the proper boundary condition calls for prescribing the normal component of the mixture velocity, $v_n = \left(\mathbf{v}^s + \mathbf{w}\right)\cdot\mathbf{n}$. To prescribe the value of v_n on a surface, use

```
<surface_load type="fluidflux">
      <flux lc="1">1.0</flux>
      <linear>0</linear>
      <mixture>1</mixture>
      <surface [set="surface_name"]>
            ...
      </surface>
</surface_load>
```

For example, this boundary condition may be used when modeling a permeation problem through a biphasic material, when the upstream fluid velocity is prescribed, $v_n = v_0$. If the upstream face is free, the companion boundary condition would be to let $t_n^e = 0$ on that face as well.

3.11.2.5. Solute Flux

The molar flux of solute relative to the solid matrix is given by the vector $\mathbf{j}$. Since solute viscosity is not explicitly modeled in a biphasic-solute material, the tangential component of $\mathbf{j}$ on a boundary surface may not be prescribed. Only the normal component of the relative solute flux, $j_n = \mathbf{j}\cdot\mathbf{n}$, represents a natural boundary condition. To prescribe a value for j_n on a surface, use:

```
<surface_load type="soluteflux">
      <flux [lc="1"]>1.0</flux>
      <linear>0</linear>
      <solute_id>1</solute_id>
      <surface>
            ...
      </surface>
</ surface_load>
```

The optional parameter *solute_id* specifies to which solute this flux condition applies, referencing the corresponding list in the Globals section (Section 3.5.2). If *solute_id* is not defined, the default value is 1.

The *flux* element defines the flux magnitude. The optional parameter *lc* defines a loadcurve for the normal flux evolution. If omitted a constant flux is applied.

When *linear* is set to 0 (default) it means that the flux matches the prescribed value even if the surface on which it is applied changes in area as it deforms. Therefore, the net molar flow rate across the surface changes with changes in area. This type of boundary condition is useful if the solute molar flux is known in the current configuration.

When *linear* is set to non-zero it means that the prescribed flux produces a molar flow rate based on the undeformed surface area in the reference configuration. Therefore, the flux in the current configuration does not match the prescribed value. This type of boundary condition is useful if the net molar flow rate across the surface is known. For example: Let Q be the known molar flow rate (in units of moles per time [**n**/**t**]), let A_0 be the surface area in the reference configuration (a constant). Using *"linear"* means that the user prescribes Q/A_0 as the flux boundary condition (in units of moles per area per time [$\mathbf{n}/\mathbf{L}^2\cdot\mathbf{t}$]). However, regardless of the *type*, the solute molar flux saved in the output file has a normal component equal to Q/A, where A=area in current configuration.

3.11.2.6. Heat Flux

A heat flux can be defined for heat transfer analyses using the *surface_load* element with type set to *heatflux* element.

```
<surface_load type="heatflux">
      <flux [lc="1"]>1.0</flux>
      <surface [set="surface_name"]>
```

```
            ...
      </surface>
</surface_load>
```

The heat flux element takes one parameter, namely *flux* which defines the flux that will be applied. It has an optional *lc* attribute which defines the load curve for this parameter. If omitted, a constant heat flux will be applied.

3.11.2.7. Convective Heat Flux

A convective heat flux can be defined to a surface that is cooled (or heated) by exposure to an ambient temperature.

```
<surface_load type="convective_heatflux">
      <Ta [lc="1"]>1.0</Ta>
      <hc>1.0</hc>
      <surface [set="surface_name"]>
            ...
      </surface>
</surface_load>
```

The *hc* parameter defines the heat transfer coefficient. The ambient temperature is defined by the *Ta* parameter. It takes an optional load curve defined through the *lc* attribute.

3.11.3. Body Loads

A body load can be used to define a source term to the model. The following sections define the different type of body loads that can be applied.

3.11.3.1. Constant Body Force

The body force is defined as a 3D vector. Each component can be associated with a load curve to define a time dependent body force. Only the non-zero components need to be defined. This type of body force is spatially homogeneous, though it may vary with time when associated with a load curve:

```
<body_load type="const">
      <x lc="1">1.0</x>
      <y lc="2">1.0</y>
      <z lc="3">1.0</z>
</body_load>
```

The *lc* attribute defines the load curve to use for the corresponding component. The values of the components can be used to define scale factors for the load values.

3.11.3.2. Non-Constant Body Force

This type of body force may be spatially inhomogeneous. The spatial inhomogeneity may be specified using a formula with variables x, y, z. For example:

```
<body_load type="non-const">
      <x>x+y+z</x>
      <y>x-y-z</y>
      <z>x*x+z</z>
</body_load>
```

3.11.3.3. Centrifugal Force

A centrifugal body force may be used for bodies undergoing steady-state rotation with angular speed ω about a rotation axis directed along $\mathbf{n}$ and passing through the rotation center $\mathbf{c}$.

```
<body_load type="centrifugal">
      <angular_speed>1.0</angular_speed>
      <rotation_axis>0.707,0.707,0</rotation_axis>
      <rotation_center>0,0,0</rotation_center>
</body_load>
```

3.11.3.4. Heat source

A heat source can be defined using the *heat_source* type.

```
<body_load type="heat_source">
      <Q>1.0</Q>
</body_load>
```

3.12. Contact Section

The *Contact* section defines all the contact interfaces. Contact boundary conditions are defined with the *contact* sub-element. The *type* attribute specifies the type of contact interface that is defined. For example:

```
<contact type="sliding_with_gaps"> ... </contact>
```

The *type* can be one of the following options:

Type	Description
`sliding_with_gaps`, `facet-to-facet sliding`, `sliding2`, `sliding3`, `sliding-multiphasic`	A sliding interface that may separate (with biphasic contact for *sliding2*, biphasic-solute contact for *sliding3*, and multiphasic contact for *sliding-multiphasic*)
`sliding-tension-compression`	A sliding interface that may optionally sustain tension
`rigid_wall`	A sliding interface with rigid wall as master surface
`rigid`	A rigid interface
`rigid_joint`	A joint between two rigid bodies
`tied`, `tied-biphasic`	A tied interface (solid-solid, solid-rigid, solid-biphasic, rigid-biphasic) or tied-biphasic interface (biphasic-biphasic).

3.12.1. Sliding Interfaces

A sliding interface can be used to setup a non-penetration constraint between two surfaces. As of version 1.2, three different sliding contact algorithms are available. Although all three are based on the same contact enforcement method, they all differ slightly in their implementation and have been shown to give different performance for different contact scenarios. Each sliding contact implementation is identified by a different *type* attribute.

- ***sliding_with_gaps (SWG)***: This is FEBio's original implementation of sliding contact. It is based on Laursen's contact formulation [3] which poses the contact problem as a nonlinear constrained optimization problem. In FEBio, the Lagrange multipliers that enforce the contact constraints are computed either using a penalty method or the augmented Lagrangian method.
- ***facet-to-facet sliding (F2F)***: This implementation is identical to the *sliding_with_gaps* implementation but uses a more accurate integration rule: where the former method uses nodal integration, this method uses Gaussian quadrature to integrate the contact equations. This method has been demonstrated to give additional stability and often converges when the former method does not.
- ***sliding-tension-compression (STC)***: This sliding contact interface may be set to sustain tension to prevent contact surfaces from separating along the direction normal to the interface, while still allowing tangential sliding. This interface is useful for creating symmetry planes, e.g., for axisymmetric problems.

- ***sliding2 (S2)***: This method is similar to the *facet-to-facet sliding* but differs in the linearization of the contact forces, which results in a different contact stiffness matrix compared to the previous two methods. This method is described in detail in [4]. This method sometimes performs better than the previous two methods for problems that are dominated by compression. However, the formulation is inherently non-symmetric and therefore will require additional memory and running time. A symmetrized version of this implementation is available (see below), but the symmetric version does not converge as well as the non-symmetric version. This particular contact implementation also supports biphasic contact (see the next section).
- ***sliding3 (S3)***: This method is similar to *sliding2*. This contact implementation supports biphasic-solute contact (see the next section). When using biphasic-solute materials, the non-symmetric version must be used.
- ***sliding-multiphasic (SMP)***: This method is similar to *sliding3*. This contact implementation supports multiphasic contact (see the next section). When using multiphasic materials, the non-symmetric version must be used.

The following table lists the properties that are defined for sliding interfaces. It is important to note that the three different sliding implementations cannot be used interchangeably: not all features are available for each method. The third, fourth and fifth column indicate if a parameter is available for a particular implementation.

Parameter	**Description**	**SWG**	**F2F**	**STC**	**S2/S3 /SMP**	**Default**
`penalty`	normal penalty scale factor (1)	●	●	●	●	1.0
`auto_penalty`	auto-penalty calculation flag (2)	●	●	●	●	0
`two_pass`	two-pass flag (3)	●	●	●	●	0
`laugon`	augmented Lagrangian flag (4)	●	●	●	●	0
`tolerance`	aug. Lagrangian convergence tolerance (4)	●	●	●	●	1.0
`gaptol`	tolerance for gap value (4)	●	●	●	●	0.0 (off)
`minaug`	minimum number of augmentations (4)	●	●	●		0
`maxaug`	maximum number of augmentations (4)	●	●	●		10
`fric_coeff`	frictional coefficient (5)	●				0.0
`fric_penalty`	tangential penalty factor (5)	●				0.0
`ktmult`	tangential stiffness multiplier (5)	●				1.0
`seg_up`	maximum number of segment updates (6)	●		●		0 (off)
`symmetric_stiffness`	symmetric stiffness matrix flag (7)			●	●	0
`search_tol`	Projection search tolerance (8)	●	●	●	●	0.01
`search_radius`	search radius (9)		●	●	●	1.0
`tension`	tension flag (10)			●		0

The slave and master surfaces, which define the contact interface, are entered by specifying the *surface* element. The *type* attribute is used to specify whether it is a *slave* or *master* surface. The *surface* tag is followed by the definition of the surface elements. For example, a list of facets composing the master surface of the sliding interface could be written as:

```
<surface type="master">
     <quad4 id="n">n1,n2,n3,n4</quad4>
     ...
</surface>
```

The supported surface elements are described in section 3.7.2.3.

If surfaces are defined in the input file using the *Geometry/Surface* section, then the contact surface definition can be defined by referring to the surfaces defined in the *Geometry/Surface* section and this can be done using the *set* attribute. For example:

```
<surface type="master" set="Surface01"/>
```

It is assumed here that the surface named *Surface01* is defined in the *Geometry/Surface* section. See section 3.7.4 for more information on how to define surfaces.

Comments:

1. If the augmented Lagrangian flag is turned off (see comment 4), the penalty method is used to enforce the contact constraint. In this method, the contact traction is determined by the gap (i.e. penetration distance) multiplied by the user-defined *penalty* factor. In the augmented Lagrangian method, the *penalty* parameter is also used but has a slightly different meaning. In this case, it scales the lagrange multiplier increment. Due to the different meanings, the user might have to adjust the penalty factor when switching between penalty method and augmented Lagrangian method. In general the penalty method requires a larger penalty factor to reach the same gap than the augmented Lagrangian method. See comment 4 for more information on when to use which method.

2. Choosing a good initial penalty parameter can often be a difficult task, since this parameter depends on material properties as well as on mesh dimensions. For this reason, an algorithm has been implemented in FEBio that attempts to calculate a good initial value for the penalty factor ε_i for a particular node/integration piont *i* on the contact interface:

   $$\varepsilon_i = \frac{EA}{V}.$$

   Here, *A* is the area of the element the integration point belongs to, *V* is the element volume and *E* is a measure of the elasticity modulus, which is calculated from the elasticity tensor of the element. Although the meaning of *E* depends on the precise material formulation, in general one can regard it as a measure of the small strain Young's modulus of the material.

To use this feature, add the following element to the contact section:

```
<auto_penalty>1</auto_penalty>
```

When the auto-penalty flag is on, the value of the *penalty* parameter serves as a scale factor for the automatically-calculated penalty factor.

3. Each sliding interface consists of a master surface and a slave surface. The slave surface is the surface over which the contact equations are integrated and on which the contact tractions are calculated. The master surface is used to measure the gap function and to define the necessary kinematic quantities such as surface normals and tangents. This approach is usually referred to as the *single-pass* method. When using the single-pass algorithm, the results can be influenced by the choice of slave and master surfaces. It is best to use the most tessellated surface as the slave and the coarsest as the master surface. To resolve the bias issue, one can also use a *two-pass* algorithm for enforcement of the contact constraint. In this case, a single pass is performed first, after which the slave and master surfaces are swapped and another pass is performed. When using the two-pass method, the definition of master and slave surfaces is arbitrary. In most problems, the single pass is sufficient to enforce contact; with a judicious choice of slave-master pair and contact parameters, good results can be obtained. If however, the single pass does not give good answers, for example, when due to the geometry's curvature the gap cannot be small enough with a single pass, the two-pass method can be used, although at the expense of more calculations.

   If one of the contacting surfaces is rigid, a slightly different approach is recommended. In this case, it is best to pick the rigid surface as the master surface and to use a single pass algorithm. The reason is that the nodal degrees of freedom on the rigid surface are condensed to the rigid degrees of freedom and if the rigid surface is the slave surface, the reaction forces may not propagate correctly to the master surface. This is especially true if the rigid degrees of freedom are fixed.[*]

4. In the presence of a sliding interface (and other contact interfaces), FEBio needs to calculate the contact tractions that prevent the two participating surfaces from penetrating. In general these tractions can be found using the method of Lagrange multipliers. However, the direct calculation of these multipliers has several computational disadvantages and therefore FEBio approximates the multipliers using one of two alternative methods: the penalty method and the augmented Lagrangian method. In the former method, the multipliers are approximated by the gap (i.e. penetration distance) scaled by a suitably chosen penalty factor. In many cases, this method is sufficient to get good results. Since the correctness of a contact solution is directly determined by the amount of penetration at the converged state, the user has direct control over the quality of the solution. By increasing the penalty factor, the penetration is reduced. However, in some cases, especially in large compression problems, the penalty factor required to achieve an acceptable amount of penetration has to be so large that it causes numerical

[*] In future versions of FEBio rigid surfaces will be automatically picked to be the master.

instabilities in the non-linear solution algorithm due to ill-conditioning of the stiffness matrix. In these cases, the augmented Lagrangian method might be a better choice. In this method, the multipliers are determined iteratively where, in each iteration, the multiplier's increments are determined with a penalty-like method. The advantage of this method is twofold: due to the iterative nature, the method will work with a smaller penalty factor, and in the limit, the exact lagrange multipliers can be recovered.

To turn on the augmented Lagrangian method, simply add the following line to the contact section:

```
<laugon>1</laugon>
```

To turn off the augmented Lagrangian method, either set the value to 0 or remove the parameter altogether. The convergence tolerance is set as follows:

```
<tolerance>0.01</tolerance>
```

With this parameter set, the augmented Lagrangian method will iterate until the relative increment in the multipliers is less than the tolerance. For instance, setting the tolerance parameter to 0.01 implies that the augmented Lagrangian method will converge when there is less than a 1% change in the L2 norm of the augmented Lagrangian multiplier vector between successive augmentations. Alternatively, the user can also specify a tolerance for the gap value. In this case, the iterations will terminate when the gap norm, which is defined as the averaged L2 norm, ($\sqrt{\sum_i < g_i >^2} / N$, $<\bullet>$ the Macauley Bracket) is less than the user-specified value:

```
<gaptol>0.001</gaptol>
```

However, one must be careful when specifying a gap tolerance. First note that the gap tolerance is an absolute value (unlike the *tolerance* which is a relative value), so this parameter depends on the dimensions of the model. Also, there are cases when a gap tolerance simply cannot be reached due to the geometry of the model in which case the augmentations may never converge.

Note that both convergence parameters may be defined at once. In that case, FEBio will try to satisfy both convergence criteria. On the other hand, setting a value of zero will turn off the convergence criteria. For example, the default value for *gaptol* is zero, so that FEBio will not check the gap norm by default.

Finally, the *minaug* and *maxaug* can be used to set a minimum and maximum number of augmentations. When the *maxuag* value is reached, FEBio will move on to the next timestep, regardless of whether the force and gap tolerances have been met. When specifying a value for *minaug*, FEBio will perform at least minaug augmentations.

5. The *sliding_with_gaps* contact implementation is currently the only contact algorithm that supports friction. Three parameters control the frictional response: *fric_coeff* is the

material's friction coefficient and its value must be in the range from 0.0 to 1.0; *fric_penalty* is the penalty factor that regulates the tangential traction forces, much like the *penalty* parameter regulates the normal traction force component; the parameter *ktmult* is a scale factor for the tangential stiffness matrix. It is default to 1.0, but it is observed that reducing this value might sometimes improve convergence.

6. In a contact problem, FEBio calculates the projection of each slave node on the master surface. As a slave node slides across the master surface, the corresponding master segment can change. However, in some cases, switching segments is undesirable since it might cause instabilities in the solution process or a state in which the node oscillates continuously between two adjacent facets and thus prevents FEBio from meeting the displacement convergence tolerance. The parameter *seg_up* allows the user to control the number of segment updates FEBio will perform during each time step. For example, a value of 4 tells FEBio it can do the segment updates during the first four iterations. After that, slave nodes will not be allowed to switch to new master segments. The default value is 0, which means that FEBio will do a segment update each iteration of each timestep.

7. The *sliding2*, *sliding3* and *sliding-multiphasic* contact implementations for sliding contact are inherently non-symmetric formulations. Symmetrized versions of these algorithms do not perform as well as the nonsymmetric version so it is recommended to use the latter. The following line controls which version of the algorithm is used.

   ```
   <symmetric_stiffness>0</symmetric_stiffness>
   ```

   A value of 1 uses the symmetric version, where a value of 0 uses the non-symmetric version.

8. The *search_tol* parameter defines the search tolerance of the algorithm that projects slave nodes onto master facets. A node that falls outside an element, but whose distance to the closest element's edge is less than the search tolerance is still considered inside. This can alleviate convergence problems when nodes are projected onto edges of master elements and due to numerical error may be projected outside the master surface.

9. The *search_radius* parameter defines the search radius for the algorithm that projects slave points onto master facets. When the distance between the slave point and the master facet exceeds the search_radius, that projection is ignored. This can alleviate convergence problems when master surfaces have multiple folds and the projection produces multiple solutions, only one of which (the closest distance) is valid.

10. The *tension* flag determines whether the contact interface can sustain tension and compression (*tension*=1) or only compression (*tension*=0).

3.12.2. Biphasic Contact

The *sliding2* implementation for sliding interfaces can deal with biphasic contact surfaces (including biphasic-on-biphasic, biphasic-on-elastic, biphasic-on-rigid). It allows for the

possibility to track fluid flow across the contact interface. In other words, fluid can flow from one side of the contact interface to the other when both contact surfaces are biphasic. To use this feature, the user must define an additional contact parameter, namely:

```
<pressure_penalty>1.0</pressure_penalty>
```

In the same way that the *penalty* parameter controls the contact tractions, this parameter controls the penalty value that is used to calculate the Lagrange multipliers for the pressure constraint. If the *laugon* flag is set, the augmented Lagrangian method is used to enforce the pressure constraint. And if the *auto_penalty* flag is defined (which is the recommended approach), an initial guess for the pressure penalty is calculated automatically using the following formula:

$$\varepsilon_p = \frac{kA}{V},$$

where A is the element's area, V is the element's volume and k is a measure of the permeability which is defined as one third of the trace of the material's initial permeability tensor.

When either contact surface is biphasic, the surface outside the contact area(s) is automatically set to free-draining conditions (equivalent to setting the fluid pressure to zero).

When performing biphasic-solute-on-rigid contact, a two-pass analysis should not be used; the biphasic-solute surface should be the slave surface.

3.12.3. Biphasic-Solute and Multiphasic Contact

The *sliding3* implementation for sliding interfaces can deal with biphasic-solute contact surfaces (including biphasic-solute-on-biphasic-solute, biphasic-solute-on-biphasic, biphasic-solute-on-elastic, biphasic-solute-on-rigid) and the *sliding-multiphasic* contact interface can similarly deal with multiphasic contact surfaces. These contact interfaces allow for the possibility to track fluid and solute flow across the contact interface. In other words, fluid and solute can flow from one side of the contact interface to the other. To use this feature, the user must define additional contact parameters, namely:

```
<pressure_penalty>1.0</pressure_penalty>
<concentration_penalty>1.0</concentration_penalty>
<ambient_pressure>0</ambient_pressure>
<ambient_concentration>0</ambient_concentration> (for sliding3)
or
<ambient_concentration      sol=”id”>0</ambient_concentration>      (for
sliding-multiphasic)
```

In the same way that the *penalty* parameter controls the contact tractions, these penalty parameters control the penalty values that are used to calculate the Lagrange multipliers for the pressure and concentration constraints. If the *laugon* flag is set, the augmented Lagrangian method is used to enforce the pressure and concentration constraints. And if the *auto_penalty* flag is defined, an initial guess for the pressure and concentration penalty is calculated automatically using the following formulas:

$$\varepsilon_p = \frac{k \cdot A}{V}, \qquad \varepsilon_c = \frac{d \cdot A}{V},$$

where A is the element's area, V is the element's volume, k is a measure of the fluid permeability which is defined as one third of the trace of the material's initial permeability tensor, and d is a measure of the solute diffusivity which is defined as one third of the trace of the material's initial diffusivity tensor.

When either contact surface is biphasic-solute or multiphasic, the surface outside the contact area(s) is automatically set to ambient conditions (equivalent to setting the effective fluid pressure and effective solute concentration to the `<ambient_pressure>` and `<ambient_concentration>` values, respectively). Ambient conditions may also be associated with a load curve, for example:

```
<ambient_pressure lc="2">1.0</ambient_pressure>
<ambient_concentration lc="3">1.0</ambient_concentration>
```

When performing biphasic-solute-on-rigid or multiphasic-on-rigid contact, a two-pass analysis should not be used; the rigid surface should be the master surface.

3.12.4. Rigid Wall Interfaces

A rigid wall interface is similar to a sliding interface, except that the master surface is a rigid wall. The following properties are defined for rigid wall interfaces:

Parameter	Description	Default
`tolerance`	augmentation tolerance	0.01
`penalty`	penalty factor	1.0
`plane`	the plane equation for the rigid wall	N/A

The *plane* property defines the plane equation for the rigid wall. Its value is an array of four values: a,b,c,d_0. It also takes a loadcurve as an optional attribute to define the motion of the plane as a function of time. The loadcurve defines the offset *h* from the initial position in the direction of the plane normal:

$$ax + by + cz + d(t) = 0, \quad d(t) = d_0 + h(t)$$

So, for example, a rigid wall that initially lies in the xy-coordinate plane and moves in the z-direction would be specified as follows:

```
<plane lc="1">0,0,1,0</plane>
```

The slave surface is defined by specifying a *surface* element. The *surface* tag is followed by the definition of the surface elements:

```
<surface>
      <quad4 id="n">n1,n2,n3,n4</quad4>
      ...
</surface>
```

Triangular elements (tria3) may also be used instead of quadrilateral elements (quad4).

3.12.5. Tied Interfaces

A tied interface can be used to connect two non-conforming meshes. A tied interface requires the definition of both a slave and a master surface. It is assumed that the nodes of the slave surface are connected to the faces of the master surface. The following control parameters need to be defined:

`<penalty>`	penalty factor
`<tolerance>`	augmentation tolerance

The slave and master surfaces are defined similarly as for the sliding interfaces. The *type* attribute is used to specify whether it is a *slave* or *master* surface. The *surface* tag is followed by the definition of the surface elements:

```
<surface type="master">
      <quad4 id="n">n1,n2,n3,n4</quad4>
      ...
</surface>
```

Both quadrilateral surface elements (quad4) and triangular elements (tri3) may be used to define the surface.

3.12.6. Tied Biphasic Interfaces

A tied biphasic interface is similar to the tied interface. It may be used for tying any combination of solid, biphasic, and rigid materials. It enforces continuity of the fluid pressure across the interface when both materials are biphasic. The following control parameters need to be defined:

Parameter	**Description**	**Default**
`penalty`	normal penalty scale factor (1)	1.0
`auto_penalty`	auto-penalty calculation flag (2)	0
`two_pass`	two-pass flag (3)	0
`laugon`	augmented Lagrangian flag (4)	0
`tolerance`	aug. Lagrangian convergence tolerance (4)	1.0
`gaptol`	tolerance for gap value (4)	0.0 (off)
`symmetric_stiffness`	symmetric stiffness matrix flag (7)	0

search_tol	Projection search tolerance (8)	

3.12.7. Rigid Interfaces

A rigid interface defines a list of nodes that are attached to a rigid body. These nodes will move with the rigid body:

```
<contact type="rigid">
      <node id="n1" rb="1"></node>
      ...
      <node id="n2" rb="1"></node>
</contact>
```

The *id* attribute identifies the node and *rb* is the material id of the rigid body. The value of the node is ignored.

3.12.8. Rigid Joints

A rigid joint connects two rigid bodies at a point in space:

```
<contact type="rigid joint">
      <tolerance>0.1</tolerance>
      <penalty>2</penalty>
      <body1>1</body1>
      <body2>2</body2>
      <joint>0,0,0</joint>
</contact>
```

The *tolerance* element defines the augmentation tolerance. That is, when the relative change in the constraint forces (the Lagrange multipliers) are less than this value. The *body1* and *body2* elements are the material numbers of the two rigid bodies. The *joint* element defines the position of the joint in world coordinates at the start of the analysis. Note that this point does not have to be inside or on the surface of either of the two bodies.

3.13. Constraints Section

The Constraints section allows the user to enforce additional constraints to the model. Currently, only rigid body constraints can be enforced.

3.13.1. Rigid Body Constraints

Rigid bodies are initially unconstrained which means they can move in all three directions and can rotate about all three axes. To constrain the degrees of freedom of a rigid body you can use the *rigid_body* element:

```
<Constraints>
	<rigid_body>
		<!-- constraints go here -->
	</rigid_body>
</Constraints>
```

The following table lists the elements that can be defined in the *rigid_body* element:

Tag	Description
`fixed`	Degree of freedom is fixed
`prescribed`	Degree of freedom is prescribed by user
`force`	A force is applied in direction of degree of freedom

All these tags require the *bc* attribute which defines the degree of freedom that will be constrained.

Bc	Description
`X`	Constrain the *x* degree of freedom
`Y`	Constrain the *y* degree of freedom
`Z`	Constrain the *z* degree of freedom
`Rx`	Prevent the rigid body from rotating around the *x*-axis
`Ry`	Prevent the rigid body from rotating around the *y*-axis
`Rz`	Prevent the rigid body from rotating around the *z*-axis

If the tag is *prescribed* or *force* then the *lc* attribute can be used to specify a load curve defining the amplitude of the displacement or force. The value is then interpreted as a scale factor. For all other types the value is ignored. The syntax and interpretation is the same for the other translation and rotation codes.

When the type is *force*, the force is applied at the center of mass for translational degrees of freedom and torque is applied around the center of mass for rotational degrees of freedom. The center of mass of a rigid body is either specified in the material definition or calculated automatically by FEBio.

Example:

```
<rigid_body mat="1">
      <prescribed bc="x" lc"1">2.0</prescribed>
      <fixed bc="y"/>
      <fixed bc="z"/>
      <fixed bc="Rx"/>
      <fixed bc="Ry"/>
      <fixed bc="Rz"/>
</rigid_body>
```

In this example the rigid body that corresponds to material definition *1* has a prescribed displacement defined for the *x* degree of freedom and has all other degrees of freedom fixed.

A force (torque) can be applied at the center of mass by setting the *type* attribute to *force*. Note that specifying a force (torque) will automatically free the corresponding translational (rotational) degree of freedom. For example, applying a force in the *x*-direction while keeping the *y* and *z* directions fixed and the rotational degrees of freedom free, can be done as follows:

```
<rigid_body mat="1" >
      <force bc="x" lc="1">1.0</force>
      <fixed bc="y"/>
      <fixed bc="z"/>
 </rigid_body>
```

3.14. Discrete Section

This section defines discrete elements, such as springs.

3.14.1. Springs

In FEBio you can connect two nodes via a discrete spring. The spring will exert a force on the nodes, depending on the separation distance and a spring contant.

A spring is defined using the `spring` element.

```
<spring [type="<type>"]>
```

The type attribute defines the type of spring. It can be any of the following values.

Type	Description
`linear`	spring that has a linear force-displacement relation
`tension-only linear`	like linear spring but force is only applied in tension
`nonlinear`	user defines the force-displacement relation

If the type attribute is omitted, it is assumed that the spring is linear.

All spring types must define the `node` sub-element to define which two nodes are connected through the spring.

The `linear` and `tension-only linear` elements require the `E` sub-element that defines the spring constant.

The `nonlinear` spring requires the force sub-element which defines the loadcurve that will be used for the force-displacement relation.

For example, you can define a linear spring between nodes 2 and 4 and having a spring constant of 2.5 as follows.

```
<spring type="linear">
      <node>2,4</node>
      <E>2.5</E>
</spring>
```

To define a nonlinear spring between nodes 2 and 4 use the following.

```
<spring type="nonlinear">
      <node>2,4</node>
      <force lc="1">1.0</force>
</spring>
```

In the last example, the force-displacement is defined by loadcurve 1. The value of the `force` element is the force scale factor, which in this case is one.

3.15. LoadData Section

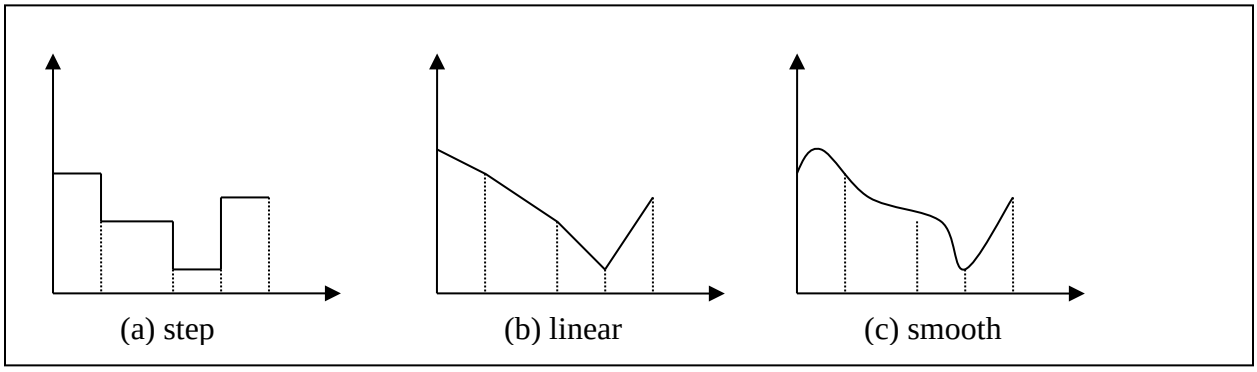
The *LoadData* section contains the loadcurve data. A loadcurve is defined by the *loadcurve* element. Each loadcurve is defined by repeating the *point* element for all data points:

```
<loadcurve id="1" [type="type" extend="extend"]>
      <point> 0, 0 </point>
      ...
      <point> 1, 1 </point>
</loadcurve>
```

The *id* attribute is the loadcurve number and is used in other sections of the input file as a means to reference this curve.

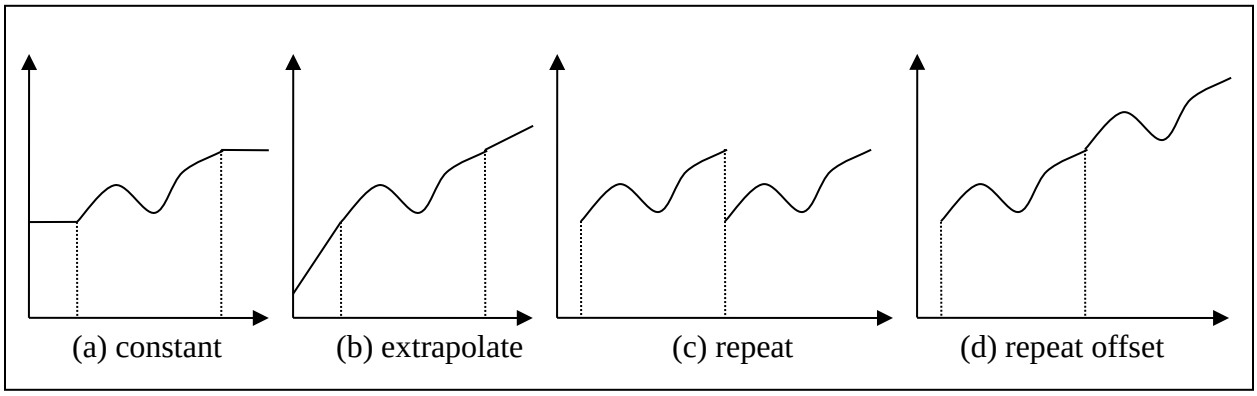
The optional attributes *type* and *extend* define how the value of the loadcurve is interpolated from the data points. The *type* defines the interpolation function and *extend* defines how the values of the loadcurve are determined outside of the interval defined by the first and last data point. The following tables list the possible values. The default values are marked with an asterisk (*).

Type	Description
`step`	Use a step interpolation function
`linear*`	Use a linear interpolation function
`smooth`	The values are interpolated using a cubic spline.



The different values for the *type* attribute of load curves.

Extend	Description
`constant`	The value of the curve is the value of the closest endpoint
`extrapolate*`	The value is extrapolated linearly from the endpoints
`repeat`	The curve is repeated
`repeat offset`	The curve is repeated but offset from the endpoints



The different values for the *extend* attribute of the load curve.

3.16. Output Section

FEBio usually splits the output in two files: the *logfile*, which contains the same information that was written to the screen during the analysis, and the *plotfile*, which contains the results. The contents of these output files can be customized in the *Output* section.

3.16.1. Logfile

The logfile records the same output that is printed to the screen. In addition, the user can request FEBio to output additional data to the logfile. This feature is called *data logging*. To use this feature, simply define the following element in the *Output* section of the input file:

```
<Output>
     <logfile [file="<log file>"]>
          <node_data       [attributes]>item list</node_data>
          <element_data    [attributes]>item list</element_data>
          <rigid_body_data [attributes]>item list</rigid_body_data>
     </logfile>
</Output>
```

The optional attribute *file* defines the name of the logfile. If omitted, a default name is used that is derived from the FEBio input file. See section 2.7 for details on default naming conventions for output files.

Additional data is stored to the logfile by adding one or more of the following elements:

- *node_data*: request nodal data
- *element_data*: request element data
- *rigid_body_data*: request rigid body data

Each of these data classes takes the following attributes:

- *data*: an expression defining the data that is to be stored
- *name:* a descriptive name for the data (optional; default = data expression)
- *file*: the name of the output file where the data is stored. (optional; default = logfile)
- *delim*: the delimiter used to separate data in multi-column format (optional; default = space)

The *data* attribute is the most important one and is mandatory. It contains a list of variable names, separated by a semi-colon. The available variable names depend on the data class and are defined below. For example, the data expression:

```
data="x;y;z"
```

will store the variables *x*, *y* and *z* in separate columns. See below for more examples.

The optional *name* attribute is a descriptive name for the data. It is used in the logfile to refer to this data and can be used to quickly find the data record in the logfile. If omitted, the data expression is used as the name.

The *file* attribute defines the name of the output file where the data is to be stored. This attribute is optional and when not specified the data will be stored in the logfile. Note that the filename given is a template. FEBio appends a number at the end of the filename to indicate to which timestep the data belongs. For instance, if you define a file name as follows:

```
file = "data.txt"
```

then the first file that is written will have the name *data000.txt*. After the first converged timestep a file with name *data001.txt* will be written and so on.

The optional *delim* attribute defines the delimiter that is used in multi-column format. As described above, data can be stored in multiple columns and the delimiter is used to separate the columns. The default is a single space.

The value of the data elements is a list of items for which the data is to be stored. For example, for the *node_data* element the value is a list of nodes, for the *element_data* element it is a list of FE elements and for the *rigid_body* element it is a list of rigid bodies. The value may be omitted in which case the data for all items will be stored. For instance, omitting the value for the *node_data* element will store the data for all nodes.

As stated above, the data is either stored in the logfile or in a separate file. In any case, a record is made in the logfile. When storing the data in the logfile, tThe following entry will be found in the logfile at the end of each converged timestep for each data element:

```
Data Record #<n>
=============================================
Step = <time step>
Time = <time value>
Data = <data name>
<actual data goes here>
```

The record number *n* corresponds to the *n*th data element in the input file. The *Step* value is the current time step. The *Time* value is the current solution time. The *Data* value is the name of the data element as provided by the *name* attribute (or the *data* attribute if *name* is omitted). The actual data immediately follows this record. If multiple column output is used, the columns are separated by the *delim* attribute of the data element.

When storing the data in a separate file, the format is slightly different:

```
Data Record #<n>
=============================================
Step = <time step>
Time = <time value>
Data = <data name>
File = <file name>
```

The *File* value is the name of the physical file. Note that this is the name to which the time step number is appended. In addition, the physical file that stores the data contains the following header:

```
*Title = <problem title>
*Step  = <time step>
*Time  = <time value>
*Data  = <data name>
<actual data goes here>
```

The problem title is as defined in the input file.

In either case, the actual data is a multi-column list, separated by the delimiter specified with the *delim* attribute (or a space when omitted). The first column always contains the item number. For example, the following data element:

```
<node_data
 data="x;y;z" name="nodal coordinates" delim=",">1:4:1</node_data>
```

will result in the following record in the logfile:

```
Data Record #1
Step = 1
Time = 0.1
Data = "nodal coordinates"
1,0.000,0.000,0.000
2,1.000,0.000,0.000
3,1.000,1.000,0.000
4,0.000,1.000,0.000
```

This data record is repeated for each converged time step. The following sections define the data variables that are available for each of the data classes.

3.16.1.1. Node_Data Class

The *node_data* class defines a set of nodal variables. The data is stored for each node that is listed in the item list of the *node_data* element or for all nodes if no list is defined. The following nodal variables are defined.

Node variables	Description
x	x-coordinate of current nodal position
y	y-coordinate of current nodal position
z	z-coordinate of current nodal position
ux	x-coordinate of nodal displacement
uy	y-coordinate of nodal displacement
uz	z-coordinate of nodal displacement

vx	x-coordinate of nodal velocity
vy	y-coordinate of nodal velocity
vz	z-coordinate of nodal velocity
ax	x-coordinate of nodal acceleration
ay	y-coordinate of nodal acceleration
az	z-coordinate of nodal acceleration
Rx	x-coordinate of nodal reaction force
Ry	y-coordinate of nodal reaction force
Rz	z-coordinate of nodal reaction force

For analyses using biphasic, biphasic-solute, and triphasic materials, the following additional variables can be defined.

Node variables	**Description**
p	effective fluid pressure
vx	x-component of solid velocity
vy	y-component of solid velocity
vz	z-component of solid velocity
c[*n*]	effective concentration of solute *n*, with *n* from 1 to 8.

For heat transfer problems the following nodal variables are available.

Node variables	**Description**
T	Nodal temperature

For example, to store the current nodal positions of all nodes, use the following *node_data* element:

```
<node_data data="x;y;z"></node_data>
```

You can store the total nodal displacement for nodes 1 through 100, and all even numbered nodes 200 through 400 as follows:

```
<node_data data="ux;uy;uz">1:100:1,200:400:2</node_data>
```

3.16.1.2. Element_Data Class

The *element_data* class defines a set of element variables. The data is stored for each element that is listed in the item list of the *element_data* element or for all nodes if no list is defined. The following element variables are defined. Note that the actual value is the average over the element's integration points values (if applicable).

Element variables	**Description**
x	x-coordinate of current element centroid position

y	y-coordinate of current element centroid position
z	z-coordinate of current element centroid position
sx	xx-component of the Cauchy stress
sy	yy-component of the Cauchy stress
sz	zz-component of the Cauchy stress
sxy	xy-component of the Cauchy stress
syz	yz-component of the Cauchy stress
sxz	xz-component of the Cauchy stress
s1	first eigenvalue of Cauchy stress tensor
s2	second eigenvalue of Cauchy stress tensor
s3	third eigenvalue of Cauchy stress tensor
Ex	xx-component of the Green-Lagrange strain
Ey	yy-component of the Green-Lagrange strain
Ez	zz-component of the Green-Lagrange strain
Exy	xy-component of the Green-Lagrange strain
Eyz	yz-component of the Green-Lagrange strain
Exz	xz-component of the Green-Lagrange strain
E1	first eigenvalue of Green-Lagrange strain tensor
E2	second eigenvalue of Green-Lagrange strain tensor
E3	third eigenvalue of Green-Lagrange strain tensor
Fxx	xx-component of the deformation gradient
Fyy	yy-component of the deformation gradient
Fzz	zz-component of the deformation gradient
Fxy	xy-component of the deformation gradient
Fyz	yz-component of the deformation gradient
Fxz	xz-component of the deformation gradient
Fyx	yx-component of the deformation gradient
Fzy	zy-component of the deformation gradient
Fzx	xz-component of the deformation gradient
J	relative volume (determinant of deformation gradient)

For analyses using biphasic, biphasic-solute, and triphasic materials, the following additional variables can be defined:

Element variables	**Description**
p	actual fluid pressure
c*n*	actual concentration of solute *n*
wx	x-component of fluid flux
wy	y-component of fluid flux
wz	z-component of fluid flux
j*n*x	x-component of flux of solute *n*
j*n*y	y-component of flux of solute *n*
j*n*z	z-component of flux of solute *n*

For example, to store the (average) Cauchy stress for all elements, define the following data element:

```
<element_data data="sxx;syy;szz;sxy;syz;sxz" name="element stresses">
</element_data>
```

3.16.1.3. Rigid_Body_Data Class

The *rigid_body_data* class defines a set of variables for each rigid body. The data is stored for each rigid body that is listed in the item list of the *rigid_body_data* element or for all rigid bodies if no list is defined. The following variables are defined. Note that the item referenced in the item list is the material number of the rigid body.

Rigid body variables	Description
x	x-coordinate of center of mass
y	y-coordinate of center of mass
z	z-coordinate of center of mass
qx	x-component of rotation quaternion
qy	y-component of rotation quaternion
qz	z-component of rotation quaternion
qw	w-component of rotation quaternion
Fx	x-component of the rigid body reaction force
Fy	y-component of the rigid body reaction force
Fz	z-component of the rigid body reaction force
Mx	x-component of the rigid body reaction torque
My	y-component of the rigid body reaction torque
Mz	z-component of the rigid body reaction torque

For example, to store the rigid body reaction force of rigid body 2 and 4 add the following data element. Note that the 2 and 4 refer to the rigid body material number as defined in the *Material* section of the input file:

```
<rigid_body_data data="Fx;Fy;Fz">2,4</rigid_body_data>
```

3.16.2. Plotfile

By default, all the results are stored in a binary database, referred to as the *plotfile*. The preferred storage format is the FEBio bindary database format (referred to as the *xplt* format)[*]. This section describes how to customize the data that is stored in the *xplt* format.

To define the contents of the plotfile you need to define the *plotfile* element in the Output section of the FEBio input file.

[*] As of FEBio version 2.0, the LSDYNA database is no longer supported. The FEBio database format is the only format that will be supported from now on.

```
<plotfile type="febio" [file="name.xplt"]/>
```

The *file* attribute is optional and allows the user to define the file name of the plotfile. If this attribute is omitted, FEBio will use a default file name for the plotfile.

By default, FEBio will store the most common data variables to the plot file. However, it is advised to always specify the specific contents of the plotfile. This can be done by adding *var* elements in the *plotfile* section.

```
<var type="<name>"/>
```

where *name* is the name of the variable. The following table lists the possible values for *name*.

Name	**Description**
`Acceleration`	Nodal accelerations
`anion concentration`	Anion concentration
`anion flux`	Anion flux
`cation concentration`	Cation concentration
`cation flux`	Cation flux
`contact gap`	Contact gap distance
`contact force`	Net contact force
`contact pressure`	Contact pressures
`contact traction`	Contact traction vectors
`current density`	Current density
`Damage`	Damage
`Displacement`	Nodal displacements
`effective anion concentration`	Effective anion concentration
`effective cation concentration`	Effective cation concentration
`effective fluid pressure`	Effective fluid pressure
`effective solute concentration`	Effective solute concentration
`electric potential`	Electric potential
`fixed charge density`	Fixed charge density
`fiber vector`	Material fiber vector
`fluid flux`	Fluid flux
`fluid force`	Net fluid force across contact interface
`fluid pressure`	Fluid pressure
`pressure gap`	Pressure gap across contact interface
`reaction forces`	Reaction force vectors
`receptor-ligand concentration`	Receptor-ligand concentration
`relative volume`	Relative volume
`shell thickness`	Shell thickness
`solute concentration`	Solute concentration
`solute flux`	Solute flux
`stress`	Cauchy stress
`temperature`	Nodal temperatures

`velocity`	Nodal velocities

The following example stores nodal displacements and element stresses.

Example:

```
<plotfile type="febio">
      <var type="displacement"/>
      <var type="stress"/>
</plotfile>
```

Chapter 4 Materials

The following sections describe the material parameters for each of the available constitutive models, along with a short description of each material. A more detailed theoretical description of the constitutive models can be found in the *FEBio Theory Manual*.

4.1. Elastic Solids

This section describes the elastic materials, which are materials defined by a hyperelastic strain-enerygy function. A distinction will be made between so-called *compressible* and *uncoupled* materials. The former describe materials that can undergo volumetric compression. The latter are used for modeling (near-) incompressible materials.

4.1.1. Specifying Fiber Orientation

Some of the anisotropic materials are modeled as an isotropic matrix in which fibers are embedded. For these materials, the user must specify several parameters related to the fibers, including an initial fiber orientation. This orientation can be specified in several ways. FEBio gives the option to automatically generate the fiber orientation, based on some user-specified parameters. However, the user can override this feature and specify the fiber directions for each element manually in the *ElementData* section. See section 3.7.3 for more details on how to do this.

4.1.1.1. Transversely Isotropic Materials

For transversely isotropic materials fiber orientation is specified with the *fiber* element. This element takes one attribute, namely *type*, which specifies the type of the fiber generator. The possible values are specified in the following table.

***type* Value**	**Description**
`local`	Use local element numbering (1)
`spherical`	Define the fiber orientation as a constant vector (2)
`vector`	specifies a spherical fiber distribution (3)
`cylindrical`	specifies a cylindrical fiber distribution (4)

If the *type* attribute is omitted, the fiber distribution will follow the local element nodes 1 and 2. This would be the same as setting the fiber attribute to *local* and setting the value to “1,2”.

Comments:

1. In this case, the fiber direction is determined by local element node numbering. The value is interpreted as the local node numbers of the nodes that define the direction of the fiber. The following example defines a local fiber axis by local element nodes 1 and 2. This option is very useful when the local fiber direction can be interpreted as following one of the mesh edges.

   ```
   <fiber type="local">1,2</fiber>
   ```

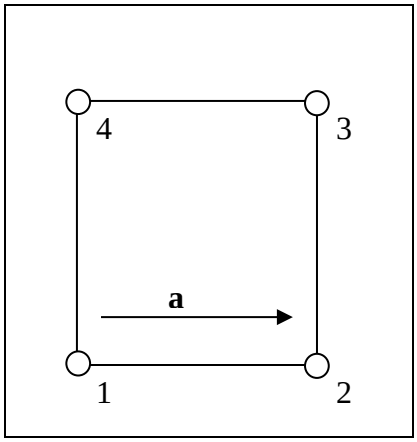


Figure 4-1. *local* fiber direction option.

2. The fiber orientation is determined by a point in space and the global location of each element integration point. The value is the location of the point. The following example defines a spherical fiber distribution centered at [0,0,1]:

```
<fiber type="spherical">0,0,1</fiber>
```

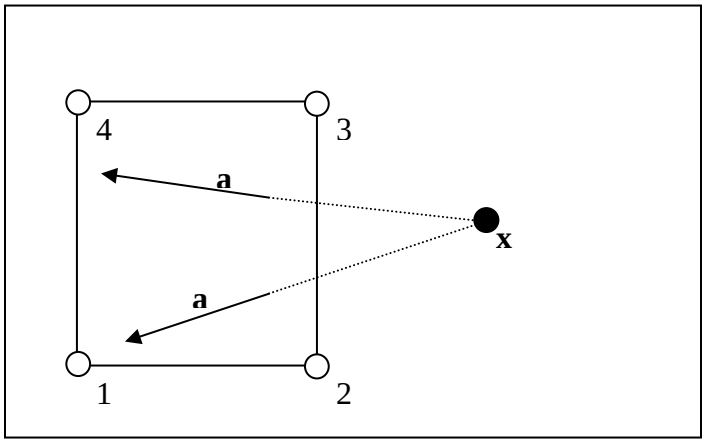


Figure 4-2. spherical fiber direction option.

3. The fiber orientation is specified by a vector. The value is the direction of the fiber. The following defines all element fiber directions in the direction of the vector [1,0,0]:

```
<fiber type="vector">1,0,0</fiber>
```

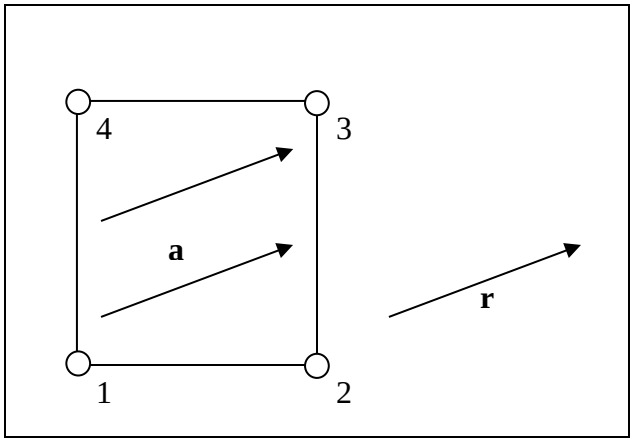


Figure 4-3. *vector* fiber direction option.

4. *cylindrical*: This type generates a fiber distribution that is cylindrical. The following subparameters must be defined.
    - *center:* defines the center of the cylinder
    - *axis:* defines the axis of the cylinder
    - *vector:* defines a vector that will be transported around the cylinder

```
<fiber type="cylindrical">
      <center>0,0,0</center>
      <axis>0,0,1</axis>
      <vector>0,1,0</vector>
</fiber>
```

4.1.1.2. Orthotropic Materials

For orthotropic materials, the user needs to specify two fiber directions **a** and **d**. From these FEBio will generate an orthonormal set of fiber vectors as follows:

$$\mathbf{e}_1 = \frac{\mathbf{a}}{\|\mathbf{a}\|}, \mathbf{e}_2 = \mathbf{e}_3 \times \mathbf{e}_1, \mathbf{e}_3 = \frac{\mathbf{a} \times \mathbf{d}}{\|\mathbf{a} \times \mathbf{d}\|}$$

The vectors **a** and **d** are defined using the *mat_axis* element. This element takes a *type* attribute, which can take on the following values:

Value	Description
`local`	Use local element numbering (1)
`vector`	Specify the vectors **a** and **d** directly. (2)

Comments:

1. When specifying *local* as the material axis type, the value is interpreted as a list of three local element node numbers. When specifying zero for all three, the default (1,2,4) is used.

```
<mat_axis type="local">0,0,0</mat_axis>
```

2. When using the *vector* type, you need to define the two generator vectors *a* and *d*. These are specified as child elements of the *mat_axis* element:

```
<mat_axis type="vector">
      <a>1,0,0</a>
      <d>0,1,0</d>
</mat_axis>
```

4.1.2. Uncoupled Materials

Uncoupled, nearly-incompressible hyperelastic materials are described by a strain energy function that features an additive decomposition of the hyperelastic strain energy into deviatoric and dilational parts [5]:

$$\Psi(\mathbf{C}) = \tilde{\Psi}(\tilde{\mathbf{C}}) + U(J),$$

where $\tilde{\mathbf{C}} = \tilde{\mathbf{F}} \cdot \tilde{\mathbf{F}}^T$ and $\tilde{\mathbf{F}} = J^{-1/3}\mathbf{F}$ is the deviatoric part of the deformation gradient. The resulting 2nd Piola-Kirchhoff stress is given by

$$\mathbf{S} = J^{-2/3}\,\mathrm{Dev}\left[\tilde{\mathbf{S}}\right] + pJ\mathbf{C}^{-1},$$

where

$$\tilde{\mathbf{S}} = 2\frac{\partial \tilde{\Psi}}{\partial \tilde{\mathbf{C}}},$$

and

$$p := \frac{dU}{dJ},$$

and $\mathrm{Dev}[\cdot]$ is the deviatoric operator in the material frame.

The corresponding Cauchy stress is given by

$$\sigma = \mathrm{dev}\left[\tilde{\sigma}\right] + p\mathbf{I},$$

where $\tilde{\sigma} = J^{-1}\tilde{\mathbf{F}} \cdot \tilde{\mathbf{S}} \cdot \tilde{\mathbf{F}}^T$ and $\mathrm{dev}[\cdot]$ is the deviatoric operator in the spatial frame.

For these materials, the entire bulk (volumetric) behavior is determined by the function $U(J)$, and *p* represents the entire hydrostatic stress. The function $U(J)$ is constructed to have a value of 0 for J=1 and to have a positive value for all other values of J>0.

All of these materials make use of the three-field element described by Simo and Taylor [5]. This element uses a trilinear interpolation of the displacement field and piecewise-constant interpolations for the pressure and volume ratio.

The uncoupled materials and the associated three-field element are very effective for representing nearly incompressible material behavior. Fully incompressible behavior can be obtained using an augmented Lagrangian method. To use the augmented Lagrangian method for enforcement of the incompressibility constraint to a user-defined tolerance, the user must define two additional material parameters:

Parameter	Description	Default
`<laugon>`	Turn augmented Lagrangian on for this material or off (1)	0 (off)
`<atol>`	Augmentation tolerance (2)	0.01

Comments:

1. A value of *1* (one) turns augmentation on, where a value of 0 (zero) turns it off.

2. The augmentation tolerance determines the convergence condition that is used for the augmented Lagrangian method: convergence is reached when the relative ratio of the lagrange multiplier norm of the previous augmentation $\|\lambda_k\|$ to the current one $\|\lambda_{k+1}\|$ is less than the specified value:

$$\left| \frac{\|\lambda_{k+1}\| - \|\lambda_k\|}{\|\lambda_{k+1}\|} \right| < \varepsilon .$$

Thus, a value of 0.01 implies that the change in norm between the previous augmentation loop and the current loop is less than 1%.

The augmented Lagrangian method for incompressibility enforcement is available for all materials that are based on an uncoupled hyperelastic strain energy function.

Example:

```
<material id="1" type="Mooney-Rivlin">
      <c1>5</c1>
      <c2>0.4</c2>
      <k>10000</k>
      <laugon>1</laugon>          ← turns on augmented Lagrangian iterations
      <atol>0.05</atol>           ← sets the augmentation tolerance
</material>
```

4.1.2.1. Arruda-Boyce

This material describes an incompressible Arruda-Boyce model [6]. The following material parameters are required:

mu	initial modulus	[**P**]
N	number of links in chain	[]
k	Bulk modulus	[**P**]

The uncoupled strain energy function for the Arruda-Boyce material is given by:

$$\Psi = \mu \sum_{i=1}^{5} \frac{C_i}{N^{i-1}} \left(\tilde{I}_1^i - 3^i \right) + U\left(J\right),$$

where, $C_1 = \frac{1}{2}, C_2 = \frac{1}{20}, C_3 = \frac{11}{1050}, C_4 = \frac{19}{7000}, C_5 = \frac{519}{673750}$ and I_1 the first invariant of the right Cauchy-Green tensor. The volumetric strain function *U* is defined as follows,

$$U\left(J\right) = \frac{1}{2} k \left(\ln J\right)^2$$

This material model was proposed by Arruda and Boyce [6] and is based on an eight-chain representation of the macromolecular network structure of polymer chains. The strain energy form represents a truncated Taylor series of the inverse Langevin function, which arises in the statistical treatment of non-Gaussian chains. The parameter *N* is related to the locking stretch λ_L, the stretch at which the chains reach their full extended state, by $\lambda_L = \sqrt{N}$.

Example:

```
<material id="1" type="Arruda-Boyce">
      <mu>0.09</mu>
      <N>26.5</N>
      <k>100</k>
</material>
```

4.1.2.2. Ellipsoidal Fiber Distribution

The material type for an ellipsoidal continuous fiber distribution in an uncoupled formulation is "*EFD uncoupled*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable compressible material that acts as a ground matrix, using a "*uncoupled solid mixture*" container as described in Section 4.1.2.14. . The following material parameters need to be defined:

<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The stress $\tilde{\sigma}$ for this material is given by [7-9]:

$$\tilde{\sigma} = \int_0^{2\pi}\int_0^{\pi} H\left(\tilde{I}_n - 1\right)\tilde{\sigma}_n\left(\mathbf{n}\right)\sin\varphi d\varphi d\theta .$$

$\tilde{I}_n = \tilde{\lambda}_n^2 = \mathbf{N}\cdot\tilde{\mathbf{C}}\cdot\mathbf{N}$ is the square of the fiber stretch $\tilde{\lambda}_n$, $\mathbf{N}$ is the unit vector along the fiber direction (in the reference configuration), which in spherical angles is directed along (θ,φ), $\mathbf{n} = \tilde{\mathbf{F}}\cdot\mathbf{N}/\tilde{\lambda}_n$, and $H(.)$ is the unit step function that enforces the tension-only contribution. The fiber stress is determined from a fiber strain energy function in the usual manner,

$$\tilde{\sigma}_n = \frac{2\tilde{I}_n}{J}\frac{\partial\tilde{\Psi}}{\partial\tilde{I}_n}\mathbf{n}\otimes\mathbf{n},$$

where in this material,

$$\tilde{\Psi}\left(\mathbf{n},\tilde{I}_n\right) = \xi\left(\mathbf{n}\right)\left(\tilde{I}_n - 1\right)^{\beta(\mathbf{n})}.$$

The materials parameters β and ξ are determined from:

$$\xi\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta\sin^2\varphi}{\xi_1^2} + \frac{\sin^2\theta\sin^2\varphi}{\xi_2^2} + \frac{\cos^2\varphi}{\xi_3^2}\right)^{-1/2}$$

$$\beta\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta\sin^2\varphi}{\beta_1^2} + \frac{\sin^2\theta\sin^2\varphi}{\beta_2^2} + \frac{\cos^2\varphi}{\beta_3^2}\right)^{-1/2}.$$

The orientation of the material axis can be defined as explained in detail in section 4.1.1.

Example:

```
<material id="1" type="uncoupled solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <k>1000</k>
      <solid type="Mooney-Rivlin">
            <c1>1</c1>
            <c2>0</c2>
      </solid>
      <solid type="EFD uncoupled">
            <ksi>10, 12, 15</ksi>
            <beta>2.5, 3, 3</beta>
      </solid>
</material>
```

4.1.2.3. Ellipsoidal Fiber Distribution Mooney-Rivlin

The material type for a Mooney-Rivlin material with an ellipsoidal continuous fiber distribution is "*EFD Mooney-Rivlin*". The following material parameters need to be defined:

<c1>	Mooney-Rivlin parameter c1	[**P**]
<c2>	Mooney-Rivlin parameter c2	[**P**]
<k>	bulk modulus	[**P**]
<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The stress $\tilde{\sigma}$ for this material is given by,

$$\tilde{\sigma} = \tilde{\sigma}_{MR} + \tilde{\sigma}_f .$$

Here, $\tilde{\sigma}_{MR}$ is the stress from the Mooney-Rivlin basis (Section 4.1.2.7.), and $\tilde{\sigma}_f$ is the stress contribution from the ellipsoidal fiber distribution (Section 4.1.2.2.). The orientation of the material axes can be defined as explained in detail in section 4.1.1.

Example:

```
<material id="1" type="EFD Mooney-Rivlin">
      <c1>1</c1>
      <c2>0</c2>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <k>20000</k>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
```

4.1.2.4. Ellipsoidal Fiber Distribution Veronda-Westmann

The material type for a Veronda-Westmann material with an ellipsoidal continuous fiber distribution is "*EFD Veronda-Westmann*". The following material parameters need to be defined:

<c1>	First VW coefficient	[**P**]
<c2>	Second VW coefficient	[ ]
<k>	Bulk modulus	[**P**]
<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The stress $\tilde{\sigma}$ for this material is given by,

$$\tilde{\sigma} = \tilde{\sigma}_{VW} + \tilde{\sigma}_f \, .$$

Here, $\tilde{\sigma}_{VW}$ is the stress from the Veronda-Westmann basis (Section 4.1.2.15.), and $\tilde{\sigma}_f$ is the stress contribution from the ellipsoidal fiber distribution (Section 4.1.2.3.).

Example:

```
<material id="1" type="EFD Veronda-Westmann">
      <c1>1</c1>
      <c2>0.5</c2>
      <k>1000</k>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
```

4.1.2.5. Fiber with Exponential-Power Law, Uncoupled Formulation

The material type for a single fiber with an exponential-power law, in an uncoupled strain energy formulation, is "*fiber-exp-pow-uncoupled*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable uncoupled material that acts as a ground matrix, using a "*uncoupled solid mixture*" container as described in Section 4.1.2.14. . The following material parameters need to be defined:

`<ksi>`	ξ, representing a measure of the fiber modulus	[**P**]
`<alpha>`	α, coefficient of exponential argument	[ ]
`<beta>`	β, power of exponential argument	[ ]
`<theta>`	θ, spherical angle for fiber orientation in local coordinate system	[**deg**]
`<phi>`	φ, spherical angle for fiber orientation in local coordinate system	[**deg**]

The fiber is oriented along

$$\mathbf{N} = \sin\varphi\cos\theta\,\mathbf{e}_1 + \sin\varphi\sin\theta\,\mathbf{e}_2 + \cos\varphi\,\mathbf{e}_3,$$

where $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ are orthonormal vectors representing the local element coordinate system (Section 4.1.1). The stress $\tilde{\sigma}$ for this fibrous material is given by

$$\tilde{\sigma} = H\left(\tilde{I}_n - 1\right)\frac{2\tilde{I}_n}{J}\frac{\partial\tilde{\Psi}}{\partial\tilde{I}_n}\mathbf{n}\otimes\mathbf{n},$$

where $\tilde{I}_n = \tilde{\lambda}_n^2 = \mathbf{N}\cdot\tilde{\mathbf{C}}\cdot\mathbf{N}$ is the square of the fiber stretch, $\mathbf{n} = \tilde{\mathbf{F}}\cdot\mathbf{N}/\tilde{\lambda}_n$, and $H(.)$ is the unit step function that enforces the tension-only contribution.. The fiber strain energy density is given by

$$\tilde{\Psi} = \frac{\xi}{\alpha\beta}\left(\exp\left[\alpha\left(\tilde{I}_n - 1\right)^{\beta}\right] - 1\right),$$

where $\xi > 0$, $\alpha \geq 0$, and $\beta \geq 2$.

Note: In the limit when $\alpha \to 0$, this expressions produces a power law,

$$\lim_{\alpha\to 0}\tilde{\Psi} = \frac{\xi}{\beta}\left(\tilde{I}_n - 1\right)^{\beta}.$$

Note: When $\beta > 2$, the fiber modulus is zero at the strain origin ($\tilde{I}_n = 1$). Therefore, use $\beta > 2$ when a smooth transition in the stress is desired from compression to tension.

Example:
Single fiber oriented along $\mathbf{e}_1$, embedded in a neo-Hookean ground matrix.

```
<material id="1" type="uncoupled solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="fiber-exp-pow-uncoupled">
            <ksi>5</ksi>
```

```
            <alpha>20</alpha>
            <beta>3</beta>
            <theta>0</theta>
            <phi>90</phi>
      </solid>
</material>
```

Example:
Two fibers in the plane orthogonal to $\mathbf{e}_1$, oriented at ±25 degrees relative to $\mathbf{e}_3$, embedded in a neo-Hookean ground matrix.

```
<material id="1" type="uncoupled solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type=”neo-Hookean”>
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="fiber-exp-pow">
            <ksi>5</ksi>
            <alpha>20</alpha>
            <beta>3</beta>
            <theta>90</theta>
            <phi>25</phi>
      </solid>
      <solid type="fiber-exp-pow">
            <ksi>5</ksi>
            <alpha>20</alpha>
            <beta>3</beta>
            <theta>-90</theta>
            <phi>25</phi>
      </solid>
</material>
```

4.1.2.6. Fung Orthotropic

The material type for orthotropic Fung elasticity [10, 11] is "*Fung orthotropic*". The following material parameters must be defined:

<mu1>	μ_1 modulus	[**P**]
<mu2>	μ_2 modulus	[**P**]
<mu3>	μ_3 modulus	[**P**]
<l11>	λ_{11} modulus	[**P**]
<l22>	λ_{22} modulus	[**P**]
<l33>	λ_{33} modulus	[**P**]
<l12>	λ_{12} modulus	[**P**]
<l23>	λ_{23} modulus	[**P**]
<l31>	λ_{31} modulus	[**P**]
<c>	c coefficient	[**P**]
<k>	bulk modulus	[**P**]

The hyperelastic strain energy function is given by [12],

$$\Psi = \frac{1}{2}c\left(e^{\tilde{Q}} - 1\right) + U\left(J\right), \tag{0.1}$$

where,

$$\tilde{Q} = c^{-1}\sum_{a=1}^{3}\left[2\mu_a\mathbf{M}_a : \tilde{\mathbf{E}}^2 + \sum_{b=1}^{3}\lambda_{ab}\left(\mathbf{M}_a : \tilde{\mathbf{E}}\right)\left(\mathbf{M}_b : \tilde{\mathbf{E}}\right)\right].$$

Here, $\tilde{\mathbf{E}} = \left(\tilde{\mathbf{C}} - \mathbf{I}\right)\big/2$ and $\mathbf{M}_a = \mathbf{V}_a \otimes \mathbf{V}_a$ where $\mathbf{V}_a$ defines the initial direction of material axis *a*. See Section 4.1.1.2. on how to define the material axes for orthotropic materials. The orthotropic Lamé parameters should produce a positive definite stiffness matrix.

Example:

```
<material id="3" type="Fung orthotropic">
      <mu1>1</mu1>
      <mu2>2</mu2>
      <mu3>3</mu3>
      <l11>3</l11>
      <l22>2</l22>
      <l33>1</l33>
      <l12>0.5</l12>
      <l23>1.0</l23>
      <l31>1.5</l31>
      <c>1</c>
      <K>100</K>
</material>
```

4.1.2.7. Mooney-Rivlin

The material type for uncoupled Mooney-Rivlin materials is *Mooney-Rivlin*. The following material parameters must be defined:

<c1>	Coefficient of first invariant term	[**P**]
<c2>	Coefficient of second invariant term	[**P**]
<k>	Bulk modulus	[**P**]

This material model is a hyperelastic Mooney-Rivlin type with uncoupled deviatoric and volumetric behavior. The strain-energy function is given by:

$$\Psi = C_1\left(\tilde{I}_1 - 3\right) + C_2\left(\tilde{I}_2 - 3\right) + \frac{1}{2}K\left(\ln J\right)^2 .$$

C_1 and C_2 are the Mooney-Rivlin material coefficients. The variables $\tilde{I}_1$ and $\tilde{I}_2$ are the first and second invariants of the deviatoric right Cauchy-Green deformation tensor $\tilde{\mathbf{C}}$. The coefficient K is a bulk modulus-like penalty parameter and J is the determinant of the deformation gradient tensor. When $C_2 = 0$, this model reduces to an uncoupled version of the neo-Hookean constitutive model.

This material model uses a three-field element formulation, interpolating displacements as linear field variables and pressure and volume ratio as piecewise constant on each element [5].

This material model is useful for modeling certain types of isotropic materials that exhibit some limited compressibility, i.e. $100 < (K/C_1) < 10000$.

Example:

```
<material id="2" type="Mooney-Rivlin">
      <c1>10.0</c1>
      <c2>20.0</c2>
      <k>1000</k>
</material>
```

4.1.2.8. Muscle Material

This material model implements the constitutive model developed by Silvia S. Blemker [13]. The material type for the muscle material is *muscle material*. The model is designed to simulate the passive and active material behavior of skeletal muscle. It defines the following parameters:

`<g1>`	along fiber shear modulus	[**P**]
`<g2>`	cross fiber shear modulus	[**P**]
`<p1>`	exponential stress coefficients	[**P**]
`<p2>`	fiber uncrimping factor	[ ]
`<Lofl>`	optimal fiber length	[ ]
`<smax>`	maximum isometric stress	[**P**]
`<lambda>`	fiber stretch for straightened fibers	[ ]
`<k>`	bulk modulus	[**P**]
`<active_contraction>`	activation level	

The main difference between this material formulation compared to other transversely hyperelastic materials is that it is formulated using a set of new invariants, originally due to Criscione [14], instead of the usual five invariants proposed by A.J.M. Spencer [15]. For this particular material, only two of the five Criscione invariants are used. The strain energy function is defined as follows:

$$\Psi\left(B_1, B_2, \lambda\right) = G_1\tilde{B}_1^2 + G_2\tilde{B}_2^2 + F_m\left(\tilde{\lambda}\right) + U\left(J\right).$$

The function F_m is the strain energy contribution of the muscle fibers. It is defined as follows:

$$\lambda\frac{\partial F_m}{\partial\lambda} = \sigma_{\max}\left(f_m^{passive}\left(\lambda\right) + \alpha\, f_m^{active}\left(\lambda\right)\right)\frac{\lambda}{\lambda_{ofl}},$$

where,

$$f_m^{passive}\left(\lambda\right) = \begin{cases} 0 & ,\lambda \le \lambda_{ofl} \\ P_1\left(e^{P_2\left(\lambda/\lambda_{ofl}-1\right)}-1\right) & ,\lambda_{ofl} < \lambda < \lambda^* \\ P_3\lambda/\lambda_{ofl} + P_4 & ,\lambda \ge \lambda^* \end{cases},$$

and,

$$f_m^{active}\left(\lambda\right) = \begin{cases} 9\left(\lambda/\lambda_{ofl} - 0.4\right)^2 & ,\lambda \le 0.6\lambda_{ofl} \\ 9\left(\lambda/\lambda_{ofl} - 1.6\right)^2 & ,\lambda \ge 1.4\lambda_{ofl} \\ 1 - 4\left(1 - \lambda/\lambda_{ofl}\right)^2 & ,0.6\lambda_{ofl} < \lambda < 1.4\lambda_{ofl} \end{cases},$$

The values P_3 and P_4 are determined by requiring C^0 and C^1 continuity at $\lambda = \lambda^*$.

The parameter α is the activation level and can be specified using the *active_contraction* element. You can specify a loadcurve using the *lc* attribute. The value is interpreted as a scale factor when a loadcurve is defined or as the constant activation level when no loadcurve is defined.

The muscle fiber direction is specified similarly to the transversely isotropic Mooney-Rivlin model.

Example:

```
<material id="1" type="muscle material">
      <g1>500</g1>
      <g2>500</g2>
      <p1>0.05</p1>
      <p2>6.6</p2>
      <smax>3e5</smax>
      <Lofl>1.07</Lofl>
      <lambda>1.4</lambda>
      <k>1e6</k>
      <fiber type="vector">1,0,0</fiber>
</material>
```

4.1.2.9. Ogden

This material describes an incompressible hyperelastic Ogden material [5]. The following material parameters must be defined:

<c[n]>	Coefficient of nth term, where n can range from 1 to 6	[**P**]
<m[n]>	Exponent of nth term, where n can range from 1 to 6	[ ]
<k>	Bulk modulus	[**P**]

The uncoupled hyperelastic strain energy function for this material is given in terms of the eigenvalues of the deformation tensor:

$$\Psi = \sum_{i=1}^{N} \frac{c_i}{m_i^2}\left(\tilde{\lambda}_1^{m_i} + \tilde{\lambda}_2^{m_i} + \tilde{\lambda}_3^{m_i} - 3\right) + U\left(J\right).$$

Here, $\tilde{\lambda}_i^2$ are the eigenvalues of $\tilde{\mathbf{C}}$, c_i and m_i are material coefficients and N ranges from 1 to 6. Note that you only have to include the material parameters for the terms you intend to use.

Example:

```
<material id="1" type="Ogden">
      <m1>2.4</m1>
      <c1>1</c1>
      <k>100</k>
</material>
```

4.1.2.10. Tendon Material

The material type for the tendon material is *tendon material*. The tendon material is similar to the muscle material. The only difference is the fiber function. For tendon material this is defined as:

$$\lambda \frac{\partial F_t}{\partial \lambda} = \sigma(\lambda),$$

where

$$\sigma(\lambda) = \begin{cases} 0, & \lambda \le 1 \\ L_1\left(e^{L_2(\lambda-1)} - 1\right) & 1 < \lambda < \lambda^* \\ L_3\lambda + L_4 & \lambda \ge \lambda^* \end{cases}.$$

The parameters L_3 and L_4 are determined by requiring C^0 and C^1 continuity at λ^*.

The material parameters for this material are listed below.

`<g1>`	along fiber shear modulus	[**P**]
`<g2>`	cross fiber shear modulus	[**P**]
`<l1>`	exponential stress coefficients	[**P**]
`<l2>`	fiber uncrimping factor	[ ]
`<lambda>`	fiber stretch for straightened fibers	[ ]
`<k>`	bulk modulus	[**P**]

The tendon fiber direction is specified similarly to the transversely isotropic Mooney-Rivlin model.

Example:

```
<material id="1" type="tendon material">
      <g1>5e4</g1>
      <g2>5e4</g2>
      <l1>2.7e6/l1>
      <l2>46.4</l2>
      <lambda>1.03</lambda>
      <k>1e7</k>
      <fiber type="vector">1,0,0</fiber>
</material>
```

4.1.2.11. Tension-Compression Nonlinear Orthotropic

The material type for the tension-compression nonlinear orthotropic material is "*TC nonlinear orthotropic*". The following material parameters are defined:

<c1>	First Mooney-Rivlin material parameter	[**P**]
<c2>	Second Mooney-Rivlin material parameter	[**P**]
<k>	bulk modulus	[**P**]
<beta>	the β parameter (see below)	[ ]
<ksi>	the ξ parameter (see below)	[**P**]
<mat_axis>	defines the material axes	

This material is based on the following uncoupled hyperelastic strain energy function [16]:

$$\Psi\left(\mathbf{C},\lambda_1,\lambda_2,\lambda_3\right)=\tilde{\Psi}_{iso}\left(\tilde{\mathbf{C}}\right)+\sum_{i=1}^{3}\tilde{\Psi}_i^{TC}\left(\tilde{\lambda}_i\right)+U\left(J\right).$$

The isotropic strain energy $\tilde{\Psi}_{iso}$ and the dilatational energy U are the same as for the Mooney-Rivlin material. The tension-compression term is defined as follows:

$$\tilde{\Psi}_i^{TC}\left(\tilde{\lambda}_i\right)=\begin{cases}\xi_i\left(\tilde{\lambda}_i-1\right)^{\beta_i}, & \tilde{\lambda}_i>1\\ 0, & \tilde{\lambda}_i\le 1\end{cases}\qquad \xi_i\ge 0 \quad \text{(no sum on i)}.$$

The $\tilde{\lambda}_i$ parameters are the deviatoric fiber stretches of the local material fibers:

$$\tilde{\lambda}_i=\left(\mathbf{a}_i^0\cdot\tilde{\mathbf{C}}\cdot\mathbf{a}_i^0\right)^{1/2}.$$

The local material fibers are defined (in the reference frame) as an orthonormal set of vectors $\mathbf{a}_i^0$. See Section 4.1.1 for more information. As with all uncoupled materials, this material uses the three-field element formulation.

A complete example for this material follows.

```
<material id="7" name="cartilage" type="TC nonlinear orthotropic">
      <c1>1.0</c1>
      <c2>0.0</c2>
      <k>100</k>
      <beta>4.3,4.3,4.3</beta>
      <ksi>4525, 4525, 4525</ksi>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
```

4.1.2.12. Transversely Isotropic Mooney-Rivlin

The material type for transversely isotropic Mooney-Rivlin materials is "*trans iso Mooney-Rivlin*". The following material parameters must be defined:

<c1>	Mooney-Rivlin coefficient 1	[**P**]
<c2>	Mooney-Rivlin coefficient 2	[**P**]
<c3>	Exponential stress coefficient	[**P**]
<c4>	Fiber uncrimping coefficient	[ ]
<c5>	Modulus of straightened fibers	[**P**]
<k>	Bulk modulus	[**P**]
<lam_max>	Fiber stretch for straightened fibers	[ ]
<fiber>	Fiber distribution option	

This constitutive model can be used to represent a material that has a single preferred fiber direction and was developed for application to biological soft tissues [17-19]. It can be used to model tissues such as tendons, ligaments and muscle. The elastic response of the tissue is assumed to arise from the resistance of the fiber family and an isotropic matrix. It is assumed that the uncoupled strain energy function can be written as follows:

$$\Psi = F_1\left(\tilde{I}_1, \tilde{I}_2\right) + F_2\left(\tilde{\lambda}\right) + \frac{K}{2}\left[\ln\left(J\right)\right]^2.$$

Here $\tilde{I}_1$ and $\tilde{I}_2$ are the first and second invariants of the deviatoric version of the right Cauchy Green deformation tensor $\tilde{\mathbf{C}}$ and $\tilde{\lambda}$ is the deviatoric part of the stretch along the fiber direction ($\tilde{\lambda}^2 = \mathbf{a}_0 \cdot \tilde{\mathbf{C}} \cdot \mathbf{a}_0$, where $\boldsymbol{a}_0$ is the initial fiber direction), and $J = \det\left(\boldsymbol{F}\right)$ is the Jacobian of the deformation (volume ratio). The function F_1 represents the material response of the isotropic ground substance matrix and is the same as the Mooney-Rivlin form specified above, while F_2 represents the contribution from the fiber family. The strain energy of the fiber family is as follows:

$$\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} = 0, \quad \tilde{\lambda} \le 1$$

$$\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} = C_3\left(e^{C_4\left(\tilde{\lambda}-1\right)} - 1\right), \quad 1 < \tilde{\lambda} < \lambda_m .$$

$$\tilde{\lambda}\frac{\partial F_2}{\partial \tilde{\lambda}} = C_5\tilde{\lambda} + C_6, \quad \tilde{\lambda} \ge \lambda_m$$

Here, C_1 and C_2 are the Mooney-Rivlin material coefficients, *lam_max* (λ_m) is the stretch at which the fibers are straightened, C_3 scales the exponential stresses, C_4 is the rate of uncrimping of the fibers, and C_5 is the modulus of the straightened fibers. C_6 is determined from the requirement that the stress is continuous at λ_m.

This material model uses a three-field element formulation, interpolating displacements as linear field variables and pressure and volume ratio as piecewise constant on each element [5].

The fiber orientation can be specified as explained in Section 4.1.1. Active stress along the fiber direction can be simulated using an active contraction model. To use this feature you need to define the *active_contraction* parameter. This parameter takes an optional attribute, *lc*, which defines the loadcurve. There are also several options:

<ca0>	Intracellular calcium concentration
<beta>	tension-sarcomere length relation constant
<l0>	No tension sarcomere length
<refl>	Unloaded sarcomere length

Example:
This example defines a transversely isotropic material with a Mooney-Rivlin basis. It defines a homogeneous fiber direction and uses the active fiber contraction feature.

```
<material id="3" type="trans iso Mooney-Rivlin">
      <c1>13.85</c1>
      <c2>0.0</c2>
      <c3>2.07</c3>
      <c4>61.44</c4>
      <c5>640.7</c5>
      <k>100.0</k>
      <lam_max>1.03</lam_max>
      <fiber type="vector">1,0,0</fiber>
      <active_contraction lc="1">
            <ca0>4.35</ca0>
            <beta>4.75</beta>
            <l0>1.58</l0>
            <refl>2.04</refl>
      </active_contraction>
</material>
```

4.1.2.13. Transversely Isotropic Veronda-Westmann

The material type for transversely isotropic Veronda-Westmann materials is "*trans iso Veronda-Westmann"*. The following material parameters must be defined:

<c1>	Veronda-Westmann coefficient 1	[**P**]
<c2>	Veronda-Westmann coefficient 2	[ ]
<c3>	Exponential stress coefficient	[**P**]
<c4>	Fiber uncrimping coefficient	[ ]
<c5>	Modulus of straightened fibers	[**P**]
<k>	Bulk modulus	[**P**]
<lam_max>	Fiber stretch for straightened fibers	[ ]
<fiber>	Fiber distribution option.	

This uncoupled hyperelastic material differs from the Transversely Isotropic Mooney-Rivlin model in that it uses the Veronda-Westmann model for the isotropic matrix. The interpretation of the material parameters, except C_1 and C_2 is identical to this material model.

The fiber distribution option is explained in Section 4.1.1. An active contraction model can also be defined for this material. See the transversely isotropic Mooney-Rivlin model for more details.

Example:
This example defines a transversely isotropic material model with a Veronda-Westmann basis. The fiber direction is implicitly implied as *local*.

```
<material id="3" type="trans iso Veronda-Westmann">
      <c1>13.85</c1>
      <c2>0.0</c2>
      <c3>2.07</c3>
      <c4>61.44</c4>
      <c5>640.7</c5>
      <lam_max>1.03</lam_max>
</material>
```

4.1.2.14. Uncoupled Solid Mixture

This material describes a mixture of quasi-incompressible elastic solids. It is a container for any combination of the materials described in Section 4.1.2.

<solid>	Container tag for compressible material	
<k>	Bulk modulus	[**P**]

The mixture may consist of any number of solids. The stress tensor for the solid mixture is the sum of the stresses for all the solids. If a bulk modulus is specified for the <solid> material, it will be added to the bulk modulus of the mixture.

Example:

```
<material id="1" type="uncoupled solid mixture">
      <k>1000</k>
      <mat_axis type="vector">
            <a>1,0,0</a>
            <d>0,1,0</d>
      </mat_axis>
      <solid type="Mooney-Rivlin">
            <c1>2.0</c1>
            <c2>0.0</c2>
      </solid>
      <solid type="EFD uncoupled">
            <ksi>10, 12, 15</ksi>
            <beta>2.5, 3, 3</beta>
      </solid>
</material>
```

4.1.2.15. Veronda-Westmann

The material type for incompressible Veronda-Westmann materials is *Veronda-Westmann* [20]. The following material parameters must be defined:

<c1>	First VW coefficient	[**P**]
<c2>	Second VW coefficient	[ ]
<k>	Bulk modulus	[**P**]

This model is similar to the Mooney-Rivlin model in that it also uses an uncoupled deviatoric dilatational strain energy:

$$\Psi = C_1\left[e^{\left(C_2\left(\tilde{I}_1-3\right)\right)}-1\right]-\frac{C_1C_2}{2}\left(\tilde{I}_2-3\right)+U\left(J\right).$$

The dilatational term is identical to the one used in the Mooney-Rivlin model. This model can be used to describe certain types of biological materials that display exponential stiffening with increasing strain. It has been used to describe the response of skin tissue [20].

Example:

```
<material id="2" type="Veronda-Westmann">
      <c1>1000.0</c1>
      <c2>2000.0</c2>
      <k>1000</k>
</material>
```

4.1.2.16. Mooney-Rivlin Von Mises Distributed Fibers
(Sclera and other thin soft tissues)

Authors: Cécile L.M. Gouget and Michaël J.A. Girard

The material type for a thin material where fiber orientation follows a von Mises distribution is "Mooney-Rivlin von Mises Fibers". The following parameters must be defined:

<c1>	Mooney-Rivlin coefficient 1	[**P**]
<c2>	Mooney-Rivlin coefficient 2	[**P**]
<c3>	Exponential stress coefficient	[**P**]
<c4>	Fiber uncrimping coefficient	[ ]
<c5>	Modulus of straightened fibers	[**P**]
<k>	Bulk modulus	[**P**]
<lam_max>	Fiber stretch for straightened fibers	[ ]
<tp>	Preferred fiber orientation in radian (angle within the plane of the fibers, defined with respect to the first direction given in mat_axis)	[rad]
<kf>	Fiber concentration factor	[ ]
<vmc>	Choice of von Mises distribution	
	= 1 semi-circular von Mises distribution	
	= 2 constrained von Mises mixture distribution	
<var_n>	Exponent (only for vmc = 2)	
<gipt>	Number of integration points (value is a multiple of 10)	
<mat_axis>	Reference frame of the plane of the fibers	

This constitutive model is designed for thin soft tissues. Fibers are multi-directional: they are distributed within the plane tangent to the tissue surface and they follow a unimodal distribution. It is a constitutive model that is suitable for ocular tissues (e.g. sclera and cornea) but can be used for other thin soft tissues. The proposed strain energy function is as follows:

$$\Psi = F_1\left(\tilde{I}_1,\tilde{I}_2\right)+\int_{\theta_P-\pi/2}^{\theta_P+\pi/2} P\left(\theta\right)F_2\left(\tilde{\lambda}\left[\theta\right]\right)d\theta+\frac{K}{2}\left[\ln\left(J\right)\right]^2,$$

where P is the 2D unimodal distribution function of the fibers which satisfies the normalization condition:

$$\int_{\theta_P-\pi/2}^{\theta_P+\pi/2} P\left(\theta\right)d\theta=1.$$

θ_P is the preferred fiber orientation relative to a local coordinate system (parameter tp). The functions F_1 and F_2 are described in the "Transversely Isotropic Mooney-Rivlin" model. The user can choose between 2 distribution functions using the parameter vmc.

1. Semi-circular von Mises distribution (vmc = 1) [21]
The semi-circular von Mises distribution is one of the simplest unimodal distribution in circular statistics. It can be expressed as

$$P(\theta)=\frac{1}{\pi I_0(k_f)}\exp\left(k_f\cos\left(2\left(\theta-\theta_p\right)\right)\right),$$

where I_0 is the modified Bessel function of the first kind (order 0), and k_f is the fiber concentration factor. k_f controls the amount of fibers that are concentrated along the orientation θ_P as illustrated in Figure 1.

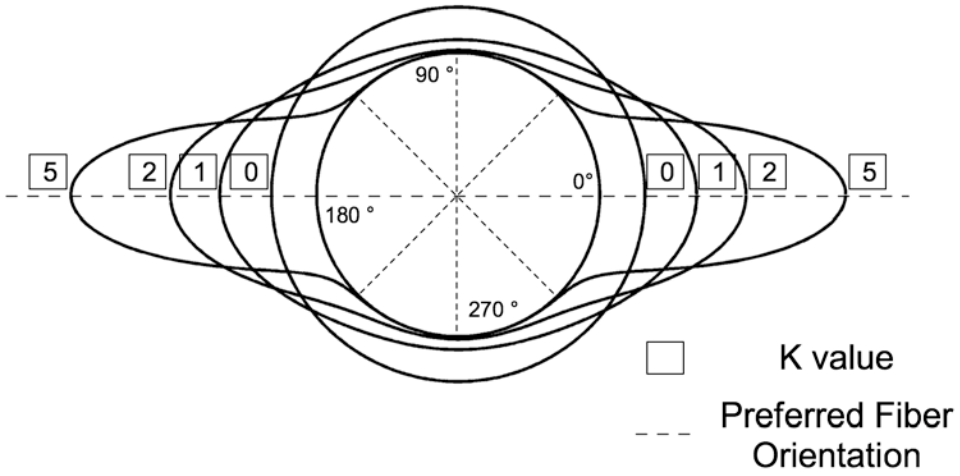


Figure 4-4. Polar representation of the semi-circular von-Mises distribution describing in-plane collagen fiber alignment. In this case, the preferred fiber orientation θ_P is equal to zero degrees. When the fiber concentration factor k is equal to zero, the collagen fibers have an isotropic distribution in a plane tangent to the scleral wall. As k increases, the collagen fibers align along the preferred fiber orientation θ_P. Note that the distributions were plotted on a circle of unit one to ease visualization.

2. Constrained von Mises Mixture Distribution (vmc = 2) [22]

The semi-circular von Mises distribution is ideal for its simplicity but in some instance it fails to accurately describe the isotropic subpopulation of fibers present in thin soft tissues. An improved mathematical description is proposed here as a weighted mixture of the semi-circular uniform distribution and the semi-circular von Mises distribution. It can be expressed as:

$$P(\theta)=\frac{1-\beta}{\pi}+\frac{\beta}{\pi I_0(k_f)}\exp\left(k_f\cos\left(2\left(\theta-\theta_p\right)\right)\right),$$

where β needs to be constrained for uniqueness and stability. β is expressed as:

$$\beta=\left(\frac{I_1(k_f)}{I_0(k_f)}\right)^n,$$

where I_1 is the modified Bessel function of the first kind (order 1), and n is an exponent to be determined experimentally (parameter var_n). n=2 has been found to be suitable for the sclera based on fiber distribution measurements using small angle light scattering.

Note about Numerical Integration

All numerical integrations are performed using a 10-point Gaussian quadrature rule. The number of Gaussian integration points can be increased for numerical stability. This is controlled through the parameter gipt. We recommend to use at least gipt = 20 (2X10 integration points). Note that the more integration points are used, the slower the model will run. By increasing the number of integration points, one should observe convergence in the numerical accuracy. The parameter gipt is required to be a multiple of 10.

Definition of the Fiber Plane

The user must specify two directions to define a local coordinate system of the plane in which the fibers lay. As explained in section 4.1.1.2., this can be done by defining two directions with the parameter mat_axis. The first direction (vector **a** of mat_axis) must be the reference direction used to define the angle tp. The second direction (vector **d** of mat_axis) can be any other direction in the plane of the fibers. Thus, a fiber at angle tp will be along the vector $\mathbf{v} = \cos(\text{tp})\,\mathbf{e_1} + \sin(\text{tp})\,\mathbf{e_2}$ where $\mathbf{e_1} = \mathbf{a}/\|\mathbf{a}\|$; $\mathbf{e_2} = \mathbf{e_3} \times \mathbf{a}/\|\mathbf{a}\|$; $\mathbf{e_3} = \mathbf{a} \times \mathbf{d}/\|\mathbf{a} \times \mathbf{d}\|$, as was defined in 4.1.1.2.

Note on parameters kf and tp

The parameters kf and tp can be specified either in the Material section or in the *ElementData* section with the tag MRVonMisesParameters. With this second option the user can define different fiber characteristics for each element separately, which can be useful if fiber orientations or concentrations vary spatially. See section 3.5 for more details.The parameter mat_axis can also be defined either in the Material section or in the *ElementData* section.

Example

This example defines a thin soft tissue material where fibers are distributed according to a constrained von Mises mixture distribution.

```
<material id="1" name="Material 1" type="Mooney-Rivlin von Mises
Fibers">
      <c1>10.0</c1>
      <c2>0.0</c2>
      <c3>50.0</c3>
      <c4>5.0</c4>
      <c5>1</c5>
      <lam_max>10</lam_max>
      <k>1000000.0</k>
      <kf>1</kf>
      <vmc>2</vmc>
      <var_n>2.0</var_n>
      <tp>0.0</tp>
      <gipt>40</gipt>
      <mat_axis type="local">4,1,3</mat_axis>
</material>
```

4.1.3. Compressible Materials

Unlike the materials in Section 4.1.2, these materials do not necessarily assume an additive decomposition of the bulk and deviatoric parts of the strain energy or stress. Further, these materials can only be used with the standard displacement-based finite element formulation, rather than the three-field element formulatoin. They should not be used for nearly-incompressible material behavior due to the potential for element locking.

4.1.3.1. Carter-Hayes

The material type for a Carter-Hayes material is *Carter-Hayes*. The following parameters must be defined:

`<E0>`	Young's modulus at reference density E_0	[**P**]
`<rho0>`	reference density ρ_0	[**M**/**L**^{3}]
`<gamma>`	exponent of solid-bound molecule density for calculation of Young's modulus γ	[ ]
`<v>`	Poisson's ratio ν	[ ]
`<sbm>`	index of solid bound molecule	[ ]

This model describes a compressible neo-Hookean material [23] whose Young's modulus is a power-law function of the referential apparent density ρ_r^σ of a solid-bound molecule. It is derived from the following hyperelastic strain-energy function:

$$\Psi_r = \frac{E_Y}{2(1+\nu)}\left[\frac{1}{2}(\mathrm{tr}\,\mathbf{C}-3)-\ln J+\frac{\nu}{1-2\nu}(\ln J)^2\right].$$

Here, $\mathbf{C}$ is the right Cauchy-Green deformation tensor and J is the determinant of the deformation gradient tensor.

Young's modulus depends on ρ_r^σ according to a power law [24, 25],

$$E_Y = E_0\left(\frac{\rho_r^\sigma}{\rho_0}\right)^\gamma .$$

This type of material references a solid-bound molecule that belongs to a multiphasic mixture. Therefore this material may only be used as the solid (or a component of the solid) in a multiphasic mixture (Section 4.6). The solid-bound molecule must be defined in the `<Globals>` section (Section 3.5.3) and must be included in the multiphasic mixture using a `<solid_bound>` tag. The parameter *sbm* must refer to the global index of that solid-bound molecule. The value of ρ_r^σ is specified within the `<solid_bound>` tag. If a chemical reaction is defined within that multiphasic mixture that alters the value of ρ_r^σ, lower and upper bounds may be specified for this referential density within the `<solid_bound>` tag to prevent E_Y from reducing to zero or achieving excessively elevated values.

Example:

```
<material id="1" name="solid matrix" type="multiphasic">
```

```
      <phi0>0</phi0>
      <solid_bound sbm="1">
            <rho0>1</rho0>
            <rhomin>0.1</rhomin>
            <rhomax>5</rhomax>
      </solid_bound>
      <fixed_charge_density>0</fixed_charge_density>
      <solid type="Carter-Hayes">
            <sbm>1</sbm>
            <E0>10000</E0>
            <rho0>1</rho0>
            <gamma>2.0</gamma>
            <v>0</v>
      </solid>
      <permeability type="perm-const-iso">
            <perm>1</perm>
      </permeability>
      <osmotic_coefficient type="osm-coef-const">
            <osmcoef>1</osmcoef>
      </osmotic_coefficient>
      <reaction name="solid remodeling" type="mass-action-forward">
            <Vbar>0</Vbar>
            <vP sbm="1">1</vP>
            <forward_rate type="Huiskes reaction rate">
                  <B>1</B>
                  <psi0>0.01</psi0>
            </forward_rate>
      </reaction>
</material>
```

4.1.3.2. Cell Growth

The material type for cell growth is "*cell growth*". The following material parameters need to be defined:

<phir>	intracellular solid volume fraction in reference (strain-free) configuration, φ_r^s	[ ]
<cr>	intracellular molar content of membrane-impermeant solute (moles per volume of the cell in the reference configuration), c_r	[**n**/**L**^{3}]
<ce>	extracellular osmolarity, c_e	[**n**/**L**^{3}]

The Cauchy stress for this material is

$$\boldsymbol{\sigma} = -\pi \mathbf{I} ,$$

where π is the osmotic pressure, given by

$$\pi = RT\left(\frac{c_r}{J - \varphi_r^s} - c_e \right),$$

where $J = \det \mathbf{F}$ is the relative volume. The values of the universal gas constant R and absolute temperature T must be specified as global constants.

Cell growth may be modeled by simply increasing the mass of the intracellular solid matrix and membrane-impermeant solute. This is achieved by allowing the parameters φ_r^s and c_r to increase over time as a result of growth, by associating them with user-defined load curves. Since cell growth is often accompanied by cell division, and since daughter cells typically achieve the same solid and solute content as their parent, it may be convenient to assume that φ_r^s and c_r increase proportionally, though this is not an obligatory relationship. To ensure that the initial configuration is a stress-free reference configuration, let $c_r = \left(1 - \varphi_r^s\right) c_e$ in the initial state prior to growth.

Example (using units of mm, N, s, nmol, K):

```
<Globals>
      <Constants>
            <T>298</T>
            <R>8.314e-06</R>
            <Fc>0</Fc>
      </Constants>
</Globals>
<Material>
      <material id="1" name="Cell" type="cell growth">
            <phir lc="1">1</phir>
            <cr lc="2">1</cr>
            <ce>300</ce>
      </material>
</Material>
<LoadData>
      <loadcurve id="1" type="smooth">
            <loadpoint>0,0.3</loadpoint>
```

```
            <loadpoint>1,0.6</loadpoint>
      </loadcurve>
      <loadcurve id="2" type="smooth">
            <loadpoint>0,210</loadpoint>
            <loadpoint>1,420</loadpoint>
      </loadcurve>
</LoadData>
```

4.1.3.3. Donnan Equilibrium Swelling

The material type for a Donnan equilibrium swelling pressure is "*Donnan equilibrium*". The swelling pressure is described by the equations for ideal Donnan equilibrium, assuming that the material is porous, with a charged solid matrix, and the external bathing environment consists of a salt solution of monovalent counter-ions [26, 27]. Since osmotic swelling must be resisted by a solid matrix, this material is not stable on its own. It must be combined with an elastic material that resists the swelling, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

`<phiw0>`	gel porosity (fluid volume fraction) in reference (strain-free) configuration, φ_0^w	[ ]
`<cF0>`	fixed-charge density in reference (strain-free) configuration, c_0^F	[**n**/**L**^{3}]
`<bosm>`	external bath osmolarity, $\overline{c}^*$	[**n**/**L**^{3}]

The Cauchy stress for this material is the stress from the Donnan equilibrium response [8]:

$$\boldsymbol{\sigma} = -\pi \mathbf{I} ,$$

where π is the osmotic pressure, given by

$$\pi = RT\left(\sqrt{\left(c^F\right)^2 + \left(\overline{c}^*\right)^2} - \overline{c}^* \right),$$

and c^F is the fixed-charge density in the current configuration, related to the reference configuration via

$$c^F = \frac{\varphi_0^w}{J - 1 + \varphi_0^w} c_0^F ,$$

where $J = \det \mathbf{F}$ is the relative volume. The values of the universal gas constant R and absolute temperature T must be specified as global constants.

Note that c_0^F may be negative or positive; the gel porosity is unitless and must be in the range $0 < \varphi_0^w < 1$. A self-consistent set of units must be used for this model. For example:

	(m, N, s, mol, K)	(mm, N, s, nmol, K)
R	8.314 J/mol·K	8.314×10^{-6} mJ/nmol·K
T	K	K
c_0^F	Eq/m^3 = mEq/L	nEq/mm^3 = mEq/L
$\overline{c}^*$	mol/m^3 = mM	nmol/mm^3 = mM
ξ_i	Pa	MPa
π	Pa	MPa

Though this material is porous, this is not a full-fledged poroelastic material as described in Section 4.4 for example. The behavior described by this material is strictly valid only after the transient response of interstitial fluid and ion fluxes has subsided (thus Donnan *equilibrium*).

Donnan osmotic swelling reduces to zero when either $c_0^F = 0$ or $\overline{c}^* \to \infty$. Therefore, entering any other values for c_0^F and $\overline{c}^*$ at the initial time point of an analysis produces an instantaneous, non-zero swelling pressure. Depending on the magnitude of this pressure relative to the solid matrix stiffness, the nonlinear analysis may not converge due to this sudden swelling. Therefore, it is recommended to prescribe a load curve for either `<cF0>` or `<bosm>`, to ease into the initial swelling prior to the application of other loading conditions.

Example (using units of mm, N, s, nmol, K):

```
<material id="1" type="solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type="Donnan equilibrium">
            <phiw0>0.8</ phiw0>
            <cF0 lc="1">1</cF0>
            <bosm>300</bosm>
      </solid>
      <solid type="ellipsoidal fiber distribution">
            <ksi>0.01,0.01,0.01</ksi>
            <beta>3,3,3</beta>
      </solid>
</material>
<LoadData>
      <loadcurve id="1">
            <loadpoint>0,0</loadpoint>
            <loadpoint>1,150</loadpoint>
      </loadcurve>
</LoadData>
<Globals>
      <Constants>
            <R>8.314e-6</R>
            <T>310</T>
      </Constants>
</Globals>
```

4.1.3.4. Ellipsoidal Fiber Distribution

The material type for an ellipsoidal continuous fiber distribution is "*ellipsoidal fiber distribution*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable compressible material that acts as a ground matrix, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The Cauchy stress for this fibrous material is given by [7-9]:

$$\boldsymbol{\sigma} = \int_0^{2\pi} \int_0^{\pi} H\left(I_n - 1\right) \boldsymbol{\sigma}_n\left(\mathbf{n}\right) \sin\varphi d\varphi d\theta .$$

Here, $I_n = \lambda_n^2 = \mathbf{N} \cdot \mathbf{C} \cdot \mathbf{N}$ is the square of the fiber stretch λ_n, $\mathbf{N}$ is the unit vector along the fiber direction, in the reference configuration, which in spherical angles is directed along (θ, φ), $\mathbf{n} = \mathbf{F} \cdot \mathbf{N} / \lambda_n$, and $H(.)$ is the unit step function that enforces the tension-only contribution.

The fiber stress is determined from a fiber strain energy function,

$$\boldsymbol{\sigma}_n = \frac{2I_n}{J} \frac{\partial \Psi}{\partial I_n} \mathbf{n} \otimes \mathbf{n},$$

where in this material,

$$\Psi\left(\mathbf{n}, I_n\right) = \xi\left(\mathbf{n}\right)\left(I_n - 1\right)^{\beta(\mathbf{n})}.$$

The materials parameters β and ξ are assumed to vary ellipsoidally with $\mathbf{n}$, according to

$$\xi\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta \sin^2\varphi}{\xi_1^2} + \frac{\sin^2\theta \sin^2\varphi}{\xi_2^2} + \frac{\cos^2\varphi}{\xi_3^2} \right)^{-1/2}$$

$$\beta\left(\mathbf{n}\right) = \left(\frac{\cos^2\theta \sin^2\varphi}{\beta_1^2} + \frac{\sin^2\theta \sin^2\varphi}{\beta_2^2} + \frac{\cos^2\varphi}{\beta_3^2} \right)^{-1/2}.$$

The orientation of the material axis can be defined as explained in detail in section 4.1.1.

Example:

```
<material id="1" type="solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type=”neo-Hookean”>
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="ellipsoidal fiber distribution">
            <ksi>10, 12, 15</ksi>
            <beta>2.5, 3, 3</beta>
      </solid>
</material>
```

4.1.3.5. Ellipsoidal Fiber Distribution Neo-Hookean

The material type for a Neo-Hookean material with an ellipsoidal continuous fiber distribution is "*EFD neo-Hookean*". The following material parameters need to be defined:

<E>	Young's modulus	[**P**]
<v>	Poisson's ratio	[ ]
<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The Cauchy stress for this material is given by,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{NH} + \boldsymbol{\sigma}_f \,.$$

Here, $\boldsymbol{\sigma}_{NH}$ is the stress from the Neo-Hookean basis (Section 4.1.3.11.), and $\boldsymbol{\sigma}_f$ is the stress contribution from the fibers (Section 4.1.3.4.).

Example:

```
<material id="1" type="EFD neo-Hookean">
      <E>1</E>
      <v>0.3</v>
      <beta>4.5,4.5,4.5</beta>
      <ksi>1,1,1</ksi>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
```

4.1.3.6. Ellipsoidal Fiber Distribution with Donnan Equilibrium Swelling

The material type for a swelling pressure combined with an ellipsoidal continuous fiber distribution is "*EFD Donnan equilibrium*". The swelling pressure is described by the equations for ideal Donnan equilibrium, assuming that the material is porous, with a charged solid matrix, and the external bathing environment consists of a salt solution of monovalent counter-ions. The following material parameters need to be defined:

<phiw0>	gel porosity (fluid volume fraction) in reference (strain-free) configuration, φ_0^w	[ ]
<cF0>	fixed-charge density in reference (strain-free) configuration, c_0^F	[**n**/**L**^{3}]
<bosm>	external bath osmolarity, $\overline{c}^*$	[**n**/**L**^{3}]
<beta>	parameters $(\beta_1, \beta_2, \beta_3)$	[ ]
<ksi>	parameters (ξ_1, ξ_2, ξ_3)	[**P**]

The Cauchy stress for this material is given by,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{DE} + \boldsymbol{\sigma}_f \, .$$

$\boldsymbol{\sigma}_f$ is the stress contribution from the fibers, as described in Section 4.1.1. $\boldsymbol{\sigma}_{DE}$ is the stress from the Donnan equilibrium response, as described in Section 4.1.3.3.

Example (using units of mm, N, s, nmol, K):

```
<material id="1" type="EFD Donnan equilibrium">
      <phiw0>0.8</ phiw0>
      <cF0 lc="1">1</cF0>
      <bosm>300</bosm>
      <beta>3,3,3</beta>
      <ksi>0.01,0.01,0.01</ksi>
      <mat_axis type="local">0,0,0</mat_axis>
</material>
<LoadData>
      <loadcurve id="1">
            <loadpoint>0,0</loadpoint>
            <loadpoint>1,150</loadpoint>
      </loadcurve>
</LoadData>
<Globals>
      <Constants>
            <R>8.314e-6</R>
            <T>310</T>
      </Constants>
</Globals>
```

4.1.3.7. Fiber with Exponential-Power Law

The material type for a single fiber with an exponential-power law is "*fiber-exp-pow*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable compressible material that acts as a ground matrix, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

<ksi>	ξ, representing a measure of the fiber modulus	[**P**]
<alpha>	α, coefficient of exponential argument	[ ]
<beta>	β, power of exponential argument	[ ]
<theta>	θ, spherical angle for fiber orientation in local coordinate system	[**deg**]
<phi>	φ, spherical angle for fiber orientation in local coordinate system	[**deg**]

The fiber is oriented along

$$\mathbf{N} = \sin\varphi\cos\theta\,\mathbf{e}_1 + \sin\varphi\sin\theta\,\mathbf{e}_2 + \cos\varphi\,\mathbf{e}_3, \qquad 0 \le \theta < 2\pi,\ 0 \le \varphi \le \pi\ ,$$

where $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ are orthonormal vectors representing the local element coordinate system (Section 4.1.1). The Cauchy stress for this fibrous material is given by

$$\boldsymbol{\sigma} = H\left(I_n - 1\right)\frac{2I_n}{J}\frac{\partial \Psi}{\partial I_n}\mathbf{n}\otimes\mathbf{n}\ ,$$

where $I_n = \lambda_n^2 = \mathbf{N}\cdot\mathbf{C}\cdot\mathbf{N}$ is the square of the fiber stretch, $\mathbf{n} = \mathbf{F}\cdot\mathbf{N}/\lambda_n$, and $H(.)$ is the unit step function that enforces the tension-only contribution. The fiber strain energy density is given by

$$\Psi = \frac{\xi}{\alpha\beta}\left(\exp\left[\alpha\left(I_n - 1\right)^{\beta}\right] - 1\right)$$

,

where $\xi > 0$, $\alpha \ge 0$, and $\beta \ge 2$.

Note: In the limit when $\alpha \to 0$, this expressions produces a power law,

$$\lim_{\alpha\to 0}\Psi = \frac{\xi}{\beta}\left(I_n - 1\right)^{\beta}$$

Note: When $\beta > 2$, the fiber modulus is zero at the strain origin ($I_n = 1$). Therefore, use $\beta > 2$ when a smooth transition in the stress is desired from compression to tension.

Example:
Single fiber oriented along $\mathbf{e}_1$, embedded in a neo-Hookean ground matrix.

```
<material id="1" type="solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="fiber-exp-pow">
            <ksi>5</ksi>
            <alpha>20</alpha>
            <beta>3</beta>
```

```
            <theta>0</theta>
            <phi>90</phi>
      </solid>
</material>
```

Example:
Two fibers in the plane orthogonal to $\mathbf{e}_1$, oriented at ±25 degrees relative to $\mathbf{e}_3$, embedded in a neo-Hookean ground matrix.

```
<material id="1" type="solid mixture">
      <mat_axis type="local">0,0,0</mat_axis>
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="fiber-exp-pow"
            <ksi>5</ksi>
            <alpha>20</alpha>
            <beta>3</beta>
            <theta>90</theta>
            <phi>25</phi>
      </solid>
      <solid type="fiber-exp-pow">
            <ksi>5</ksi>
            <alpha>20</alpha>
            <beta>3</beta>
            <theta>-90</theta>
            <phi>25</phi>
      </solid>
</material>
```

4.1.3.8. Holmes-Mow

The material type for the Holmes-Mow material [28] is *Holmes-Mow*. This isotropic hyperelastic material has been used to represent the solid matrix of articular cartilage [28, 29] and intervertebral disc [30]. The following material parameters must be defined:

<E>	Young's modulus	[**P**]
<v>	Poisson's ratio	[ ]
<beta>	Exponential stiffening coefficient	[ ]

The coupled hyperelastic strain-energy function for this material is given by [28]:

$$W\left(I_1, I_2, J\right) = \frac{1}{2}c\left(e^Q - 1\right),$$

where I_1 and I_2 are the first and second invariants of the right Cauchy-Green tensor and J is the jacobian of the deformation gradient. Furthermore,

$$Q = \frac{\beta}{\lambda + 2\mu}\left[\left(2\mu - \lambda\right)\left(I_1 - 3\right) + \lambda\left(I_2 - 3\right) - \left(\lambda + 2\mu\right)\ln J^2\right],$$

$$c = \frac{\lambda + 2\mu}{2\beta}$$

and λ and μ are the Lamé parameters which are related to the more familiar Young's modulus and Poisson's ratio in the usual manner:

$$\lambda = \frac{E}{\left(1+\nu\right)\left(1-2\nu\right)},$$

$$\mu = \frac{E}{2\left(1+\nu\right)}.$$

Example:

```
<material id="3" type="Holmes-Mow">
      <E>1</E>
      <v>0.35</v>
      <beta>0.25</beta>
</material>
```

4.1.3.9. Isotropic Elastic

The material type for isotropic elasticity is *isotropic elastic*[*]. The following material parameters must be defined:

<E>	Young's modulus	[**P**]
<v>	Poisson's ratio	[ ]

This material is an implementation of a hyperelastic constitutive material that reduces to the classical linear elastic material for small strains, but is objective for large deformations and rotations. The hyperelastic strain-energy function is given by:

$$W = \frac{1}{2}\lambda\left(\operatorname{tr}\mathbf{E}\right)^2 + \mu\mathbf{E}:\mathbf{E}\,.$$

Here, **E** is the Euler-Lagrange strain tensor and λ and μ are the Lamé parameters, which are related to the more familiar Young's modulus E and Poisson's ratio ν as follows:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}.$$

It is often convenient to express the material properties using the bulk modulus K and shear modulus G. To convert to Young's modulus and Poisson's ratio, use the following formulas:

$$E = \frac{9KG}{3K+G}, \quad \nu = \frac{3K-2G}{6K+2G}.$$

Remark: Note that although this material is objective, it is not advised to use this model for large strains since the behavior may be unphysical. For example, it can be shown that for a uniaxial tension the stress grows unbounded and the volume tends to zero for finite strains. Also for large strains, the Young's modulus and Poisson's ratio input values have little relationship to the actual material parameters. Therefore it is advisable to use this material only for small strains and/or large rotations. To represent isotropic elastic materials under large strain and rotation, it is best to use some of the other available nonlinear material models described in this chapter, such as the Holmes-Mow material in Section 4.1.3.8. , the neo-Hookean material in Section 4.1.3.11. , or the unconstrained Ogden material in Section 4.1.3.12. .

Example:

```
<material id="1" type="isotropic elastic"
      <E>1000.0</E>
      <v>0.45</v>
</material>
```

[*] This material replaces the now-obsolete *linear elastic* and *St.Venant-Kirchhoff* materials. These materials are still available for backward compatibility although it is recommended to use the *isotropic elastic* material instead.

4.1.3.10. Orthotropic Elastic

The material type for orthotropic elasticity is "*orthotropic elastic*". The following material parameters must be defined:

<E1>	Young's modulus in the x-direction	[**P**]
<E2>	Young's modulus in the y-direction	[**P**]
<E3>	Young's modulus in the z-direction	[**P**]
<G12>	Shear modulus in the xy-plane	[**P**]
<G23>	Shear modulus in the yz-plane	[**P**]
<G31>	Shear modulus in the xz-plane	[**P**]
<v12>	Poisson's ratio between x- and y-direction	[ ]
<v23>	Poisson's ratio between y- and z-direction	[ ]
<v31>	Poisson's ratio between z- and x-direction	[ ]

The stress-strain relation for this material is given by

$$\begin{bmatrix} E_{11} \\ E_{22} \\ E_{33} \\ 2E_{23} \\ 2E_{31} \\ 2E_{12} \end{bmatrix} = \begin{bmatrix} 1/E_1 & -\nu_{21}/E_2 & -\nu_{31}/E_3 & 0 & 0 & 0 \\ -\nu_{12}/E_1 & 1/E_2 & -\nu_{32}/E_3 & 0 & 0 & 0 \\ -\nu_{13}/E_1 & -\nu_{23}/E_2 & 1/E_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/G_{23} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/G_{31} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/G_{12} \end{bmatrix} \begin{bmatrix} T_{11} \\ T_{22} \\ T_{33} \\ T_{23} \\ T_{31} \\ T_{12} \end{bmatrix}$$

Material axes may be specified as described in Section 4.1.1.2. .

Example:

```
<material id="3" type="orthotropic elastic">
      <mat_axis type="vector">
            <a>0.866,0.5,0</a>
            <d>-0.5,0.866,0</d>
      </mat_axis>
      <E1>1</E1>
      <E2>2</E2>
      <E3>3</E3>
      <v12>0</v12>
      <v23>0</v23>
      <v31>0</v31>
      <G12>1</G12>
      <G23>1</G23>
      <G31>1</G31>
</material>
```

4.1.3.11. Neo-Hookean

The material type for a Neo-Hookean material is *neo-Hookean*. The following parameters must be defined:

<E>	Young's modulus	[**P**]
<v>	Poisson's ratio	[ ]

This model describes a compressible Neo-Hookean material [23]. It has a non-linear stress-strain behavior, but reduces to the classical linear elasticity model for small strains *and* small rotations. It is derived from the following hyperelastic strain-energy function:

$$W = \frac{\mu}{2}\left(I_1 - 3\right) - \mu \ln J + \frac{\lambda}{2}\left(\ln J\right)^2 .$$

Here, I_1 and I_2 are the first and second invariants of the right Cauchy-Green deformation tensor **C** and J is the determinant of the deformation gradient tensor.

This material model uses a standard displacement-based element formulation, so care must be taken when modeling materials with nearly-incompressible material behavior to avoid element locking. For this case, use the *Mooney-Rivlin* material described in Section 4.1.2.7.

Example:

```
<material id="1" type="neo-Hookean">
      <E>1000.0</E>
      <v>0.45</v>
</material>
```

4.1.3.12. Ogden Unconstrained

This material describes a compressible (unconstrained) hyperelastic Ogden material [5]. The following material parameters must be defined:

<c[n]>	Coefficient of nth term, where n can range from 1 to 6	[**P**]
<m[n]>	Exponent of nth term, where n can range from 1 to 6	[ ]
<cp>	Bulk-like modulus	[**P**]

The hyperelastic strain energy function for this material is given in terms of the eigenvalues of the right or left stretch tensor:

$$W\left(\lambda_1,\lambda_2,\lambda_3\right)=\frac{1}{2}c_p\left(J-1\right)^2+\sum_{i=1}^{N}\frac{c_i}{m_i^2}\left(\lambda_1^{m_i}+\lambda_2^{m_i}+\lambda_3^{m_i}-3-m_i\ln J\right).$$

Here, λ_i^2 are the eigenvalues of the right or left Cauchy deformation tensor, c_p, c_i and m_i are material coefficients and N ranges from 1 to 6. Any material parameters that are not specified by the user are assumed to be zero.

Example:

```
<material id="1" type="Ogden unconstrained">
      <m1>2.4</m1>
      <c1>1</c1>
      <cp>2</cp>
</material>
```

4.1.3.13. Perfect Osmometer Equilibrium Osmotic Pressure

The material type for a perfect osmometer equilibrium swelling pressure is "*perfect osmometer*". The swelling pressure is described by the equations for a perfect osmometer, assuming that the material is porous, containing an interstitial solution whose solutes cannot be exchanged with the external bathing environment; similarly, solutes in the external bathing solution cannot be exchanged with the interstitial fluid of the porous material. Therefore, osmotic pressurization occurs when there is an imbalance between the interstitial and bathing solution osmolarities. Since osmotic swelling must be resisted by a solid matrix, this material is not stable on its own. It must be combined with an elastic material that resists the swelling, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

`<phiw0>`	gel porosity (fluid volume fraction) in reference (strain-free) configuration, φ_0^w	[ ]
`<iosm>`	interstitial fluid osmolarity in reference configuration, $\overline{c}_0$	[**n**/**L**^{3}]
`<bosm>`	external bath osmolarity, $\overline{c}^*$	[**n**/**L**^{3}]

The Cauchy stress for this material is the stress from the perfect osmometer equilibrium response:

$$\boldsymbol{\sigma} = -\pi \mathbf{I} ,$$

where π is the osmotic pressure, given by

$$\pi = RT\left(\overline{c} - \overline{c}^*\right).$$

$\overline{c}$ is the interstitial fluid in the current configuration, related to the reference configuration via

$$\overline{c} = \frac{\varphi_0^w}{J - 1 + \varphi_0^w}\overline{c}_0$$

where $J = \det \mathbf{F}$ is the relative volume. The values of the universal gas constant R and absolute temperature T must be specified as global constants.

Though this material is porous, this is not a full-fledged poroelastic material as described in Section 4.4 for example. The behavior described by this material is strictly valid only after the transient response of interstitial fluid and solute fluxes has subsided.

Example (using units of mm, N, s, nmol, K):
Hyperosmotic loading of a cell with a membrane-impermeant solute, starting from isotonic conditions.

```
<material id="1" type="solid mixture">
      <solid type="perfect osmometer">
            <phiw0>0.8</ phiw0>
            <iosm>300</cF0>
            <bosm lc="1">1</bosm>
      </solid>
      <solid type="neo-Hookean">
            <E>1.0</E>
```

```
            <v>0</v>
      </solid>
</material>
<LoadData>
      <loadcurve id="1">
            <loadpoint>0,300</loadpoint>
            <loadpoint>1,1500</loadpoint>
      </loadcurve>
</LoadData>
<Globals>
      <Constants>
            <R>8.314e-6</R>
            <T>310</T>
      </Constants>
</Globals>
```

4.1.3.14. Solid Mixture

This material describes a mixture of compressible elastic solids. It is a container for any combination of the materials described in Section 4.1.3.

`<solid>`	Container tag for compressible material

The mixture may consist of any number of solids. The stress tensor for the solid mixture is the sum of the stresses for all the solids.

Example:

```
<material id="1" type="solid mixture">
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="ellipsoidal fiber distribution">
            <ksi>10, 12, 15</ksi>
            <beta>2.5, 3, 3</beta>
      </solid>
</material>
```

4.1.3.15. Spherical Fiber Distribution

The material type for a spherical (isotropic) continuous fiber distribution is "*spherical fiber distribution*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable compressible material that acts as a ground matrix, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

<alpha>	parameter α	[ ]
<beta>	parameter β	[ ]
<ksi>	parameters ξ	[**P**]

The Cauchy stress for this fibrous material is given by [7-9]:

$$\sigma = \int_0^{2\pi}\int_0^{\pi} H\left(I_n - 1\right)\sigma_n\left(\mathbf{n}\right)\sin\varphi d\varphi d\theta \,.$$

Here, $I_n = \lambda_n^2 = \mathbf{N}\cdot\mathbf{C}\cdot\mathbf{N}$ is the square of the fiber stretch λ_n, $\mathbf{N}$ is the unit vector along the fiber direction, in the reference configuration, which in spherical angles is directed along (θ,φ), $\mathbf{n} = \mathbf{F}\cdot\mathbf{N}/\lambda_n$, and $H\left(.\right)$ is the unit step function that enforces the tension-only contribution.

The fiber stress is determined from a fiber strain energy function,

$$\sigma_n = \frac{2I_n}{J}\frac{\partial\Psi}{\partial I_n}\mathbf{n}\otimes\mathbf{n}$$

,

where in this material, the fiber strain energy density is given by

$$\Psi = \frac{\xi}{\alpha}\left(\exp\left[\alpha\left(I_n - 1\right)^{\beta}\right] - 1\right)$$

,

where $\xi > 0$, $\alpha \geq 0$, and $\beta \geq 2$.

Note: In the limit when $\alpha \to 0$, this expressions produces a power law,

$$\lim_{\alpha\to 0}\Psi = \xi\left(I_n - 1\right)^{\beta}$$

Note: When $\beta > 2$, the fiber modulus is zero at the strain origin ($I_n = 1$). Therefore, use $\beta > 2$ when a smooth transition in the stress is desired from compression to tension.

$\Psi\left(\mathbf{n}, I_n\right) = \xi\left(I_n - 1\right)^{\beta}$

Example:

```
<material id="1" type="solid mixture">
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="spherical fiber distribution">
            <ksi>10</ksi>
            <alpha>0</alpha>
            <beta>2.5</beta>
```

```
      </solid>
</material>
```

4.1.3.16. Spherical Fiber Distribution from Solid-Bound Molecule

The material type for a spherical (isotropic) continuous fiber distribution with fiber modulus dependent on solid-bound molecule content is "*spherical fiber distribution sbm*". Since fibers can only sustain tension, this material is not stable on its own. It must be combined with a stable compressible material that acts as a ground matrix, using a "*solid mixture*" container as described in Section 4.1.3.14. . The following material parameters need to be defined:

`<alpha>`	parameter α	[ ]
`<beta>`	parameter β	[ ]
`<ksi0>`	fiber modulus ξ_0	[**P**]
`<gamma>`	fiber modulus exponent γ	[ ]
`<rho0>`	fiber mass density ρ_0	[**M**/**L**^{3}]
`sbm`	index of solid-bound molecule σ	[ ]

The Cauchy stress for this fibrous material is given by [7-9]:

$$\sigma = \int_0^{2\pi}\int_0^{\pi} H\left(I_n - 1\right)\sigma_n\left(\mathbf{n}\right)\sin\varphi d\varphi d\theta \, .$$

Here, $I_n = \lambda_n^2 = \mathbf{N}\cdot\mathbf{C}\cdot\mathbf{N}$ is the square of the fiber stretch λ_n, $\mathbf{N}$ is the unit vector along the fiber direction, in the reference configuration, which in spherical angles is directed along (θ,φ), $\mathbf{n} = \mathbf{F}\cdot\mathbf{N}/\lambda_n$, and $H\left(.\right)$ is the unit step function that enforces the tension-only contribution.

The fiber stress is determined from a fiber strain energy function,

$$\sigma_n = \frac{2I_n}{J}\frac{\partial\Psi}{\partial I_n}\mathbf{n}\otimes\mathbf{n}$$

,

where in this material, the fiber strain energy density is given by

$$\Psi = \frac{\xi}{\alpha}\left(\exp\left[\alpha\left(I_n - 1\right)^{\beta}\right] - 1\right)$$

,

where $\xi > 0$, $\alpha \geq 0$, and $\beta \geq 2$. The fiber modulus is dependent on the solid-bound molecule referential density ρ_r^{σ} according to the power law relation

$$\xi = \xi_0\left(\frac{\rho_r^{\sigma}}{\rho_0}\right)^{\gamma} \, ,$$

where ρ_0 is the density at which $\xi = \xi_0$.

This type of material references a solid-bound molecule that belongs to a multiphasic mixture. Therefore this material may only be used as the solid (or a component of the solid) in a multiphasic mixture (Section 4.6). The solid-bound molecule must be defined in the `<Globals>` section (Section 3.5.3) and must be included in the multiphasic mixture using a

<solid_bound> tag. The parameter *sbm* must refer to the global index of that solid-bound molecule. The value of ρ_r^σ is specified within the <solid_bound> tag. If a chemical reaction is defined within that multiphasic mixture that alters the value of ρ_r^σ, lower and upper bounds may be specified for this referential density within the <solid_bound> tag to prevent ξ from reducing to zero or achieving excessively elevated values.

Note: In the limit when $\alpha \to 0$, the expression for Ψ produces a power law,

$$\lim_{\alpha \to 0} \Psi = \xi \left(I_n - 1 \right)^\beta$$

Note: When $\beta > 2$, the fiber modulus is zero at the strain origin ($I_n = 1$). Therefore, use $\beta > 2$ when a smooth transition in the stress is desired from compression to tension.

$\Psi \left(\mathbf{n}, I_n \right) = \xi \left(I_n - 1 \right)^\beta$

Example:

```
<solid type="solid mixture">
      <solid type="neo-Hookean">
            <E>1000.0</E>
            <v>0.45</v>
      </solid>
      <solid type="spherical fiber distribution sbm">
            <alpha>0</alpha>
            <beta>2.5</beta>
            <ksi0>10</ksi0>
            <gamma>2</gamma>
            <rho0>1</rho0>
            <sbm>1</sbm>
      </solid>
</solid>
```

4.1.3.17. Coupled Transversely Isotropic Mooney-Rivlin

This material describes a transversely isotropic Mooney-Rivlin material using a fully-coupled formulation. It is define through the *coupled trans-iso Mooney-Rivlin* material type. The following material parameters must be defined.

`c1`	Mooney-Rivlin c_1 parameter.	[**P**]
`c2`	Mooney-Rivlin c_2 parameter.	[**P**]
`c3`	exponential multiplier	[**P**]
`c4`	fiber scale factor	[ ]
`c5`	fiber modulus in linear region	[**P**]
`lam_max`	maximum fiber straightening stretch	[ ]
`k`	bulk-like modulus	[**P**]

The strain-energy function for this constitutive model is defined by

$$W = c_1\left(I_1 - 3\right) + c_2\left(I_2 - 3\right) + F\left(\lambda\right) + U\left(J\right)$$

The first two terms define the Mooney-Rivlin matrix response. The third term is the fiber response which is a function of the fiber stretch λ and is defined as in [19]. For *U*, the following form is chosen in FEBio.

$$U\left(J\right) = \frac{1}{2}k\left(\ln J\right)^2$$

where $J = \det \mathbf{F}$ is the Jacobian of the deformation.

Example:

```
<material id="1" type="coupled trans-iso Mooney-Rivlin">
      <c1>1</c1>
      <c2>0.1</c2>
      <c3>1</c3>
      <c4>1</c4>
      <c5>1.34</c5>
      <lam_max>1.3</lam_max>
      <k>100</k>
</material>
```

4.1.3.18. Coupled Transversely Isotropic Veronda-Westmann

This material describes a transversely isotropic Veronda-Westmann material using a fully-coupled formulation. It is define through the *coupled trans-iso Veronda-Westmann* material type. The following material parameters must be defined.

`c1`	Veronda-Westmann c_1 parameter.	[**P**]
`c2`	Veronda-Westmann c_2 parameter.	[ ]
`c3`	exponential multiplier	[**P**]
`c4`	fiber scale factor	[ ]
`c5`	fiber modulus in linear region	[**P**]
`lam_max`	maximum fiber straightening stretch	[ ]
`k`	bulk-like modulus	[**P**]

The strain-energy function for this constitutive model is defined by

$$W = c_1\left[\exp\left(c_2\left(I_1-3\right)\right)\right]-\frac{1}{2}c_1c_2\left(I_2-3\right)+F\left(\lambda\right)+U\left(J\right)$$

The first two terms define the Veronda-Westmann matrix response. The third term is the fiber response which is a function of the fiber stretch λ and is defined as in [19]. For *U*, the following form is chosen in FEBio.

$$U\left(J\right)=\frac{1}{2}k\left(\ln J\right)^2$$

where $J=\det\mathbf{F}$ is the Jacobian of the deformation.

Example:

```
<material id="1" type="coupled trans-iso Veronda-Westmann">
      <c1>1</c1>
      <c2>0.1</c2>
      <c3>1</c3>
      <c4>1</c4>
      <c5>1.34</c5>
      <lam_max>1.3</lam_max>
      <k>100</k>
</material>
```

4.2. Viscoelastic Solids

4.2.1. Uncoupled Viscoelastic Materials

These materials produce a viscoelastic response only for the deviatoric stress response. They must be used whenever the elastic response is uncoupled, as in the materials described in Section 4.1.2.
The material type for these materials is “uncoupled viscoelastic”. The following parameters need to be defined:

`<t1>-<t6>`	relaxation times	[**t**]
`<g1>-<g6>`	viscoelastic coefficients	[ ]
`<elastic>`	elastic component (must be an uncoupled elastic solid)	

$\mathbf{S}(t) = \int_{-\infty}^{t} G(t-s)\frac{d\mathbf{S}^e}{ds}ds$ For a uncoupled viscoelastic material, the second Piola Kirchhoff stress can be written as follows [17]:

$$\mathbf{S}(t) = pJ\,\mathbf{C}^{-1} + J^{-2/3}\int_{-\infty}^{t} G(t-s)\frac{d\left(\mathrm{Dev}\left[\tilde{\mathbf{S}}^e\right]\right)}{ds}ds\,,$$

where $\mathbf{S}^e$ $\tilde{\mathbf{S}}^e$ is the elastic stress derived from $\tilde{W}\left(\tilde{\mathbf{C}}\right)$ (see Section 4.1.2) and G is the relaxation function. It is assumed that the relaxation function is given by the following discrete relaxation spectrum:

$$G(t) = 1 + \sum_{i=1}^{N}\gamma_i \exp\left(-t/\tau_i\right),$$

Note that the user does not have to enter all the τ_i and γ_i coefficients. Instead, only the values that are used need to be entered. So, if N is 2, only τ_1, τ_2, γ_1 and γ_2 have to be entered.

The *elastic* parameter describes the elastic material as given in Section 4.1.

Example:

```
<material id="1" name="Material 1" type="uncoupled viscoelastic">
      <g1>0.95</g1>
      <t1>0.01</t1>
      <elastic type="Mooney-Rivlin">
            <c1>1</c1>
            <c2>0.0</c2>
            <k>100</k>
      </elastic>
</material>
```

4.2.2. Compressible Viscoelastic Materials

The material type for viscoelastic materials is *"viscoelastic"*. The following parameters need to be defined:

<t1>-<t6>	relaxation times	[**t**]
<g1>-<g6>	viscoelastic coefficients	[ ]
<elastic>	elastic component (must be a compressible elastic solid)	

For a viscoelastic material, the second Piola Kirchhoff stress can be written as follows [17]:

$$\mathbf{S}(t) = \int_{-\infty}^{t} G(t-s)\frac{d\mathbf{S}^e}{ds}ds,$$

where $\mathbf{S}^e$ is the elastic stress and G is the relaxation function. It is assumed that the relaxation function is given by the following discrete relaxation spectrum:

$$G(t) = 1 + \sum_{i=1}^{N} \gamma_i \exp(-t/\tau_i),$$

Note that the user does not have to enter all the τ_i and γ_i coefficients. Instead, only the values that are used need to be entered. So, if N is 2, only τ_1, τ_2, γ_1 and γ_2 have to be entered.

The *elastic* parameter describes the elastic material as given in Section 4.1.

Example:

```
<material id="1" name="Material 1" type="viscoelastic">
      <g1>0.95</g1>
      <t1>0.01</t1>
      <elastic type="neo-Hookean">
            <E>1</E>
            <v>0.0</v>
      </elastic>
</material>
```

4.3. Multigeneration Solids

This type of material [31] implements a mechanism for multigenerational interstitial growth of solids whereby each growth generation γ has a distinct reference configuration $\mathbf{X}^\gamma$ determined at the time t^γ of its deposition. Therefore, the solid matrix of a growing material consists of a multiplicity of intermingled porous bodies, each representing a generation γ, all of which are constrained to move together in the current configuration $\mathbf{x}$. The deformation gradient of each generation is $\mathbf{F}^\gamma = \partial\mathbf{x}/\partial\mathbf{X}^\gamma$. The first generation ($\gamma = 1$) is assumed to be present at time $t^1 = 0$, therefore its reference configuration is $\mathbf{X}^1 \equiv \mathbf{X}$ and its deformation gradient $\mathbf{F}^1 = \partial\mathbf{x}/\partial\mathbf{X}^1$ is equivalent to $\mathbf{F} = \partial\mathbf{x}/\partial\mathbf{X}$. Each generation's reference configuration $\mathbf{X}^\gamma$ has a one-to-one mapping $\mathbf{F}^{\gamma 1} = \partial\mathbf{X}^\gamma/\partial\mathbf{X}^1$ with the master reference configuration $\mathbf{X}^1$, which is that of the first generation. This mapping is postulated based on a constitutive assumption with regard to that generation's state of stress at the time of its deposition. In the current implementation, the newly

deposited generation is assumed to be in a stress-free state, even though the underlying material is in a loaded configuration. Therefore, the mapping between generation γ and the first generation is simply $\mathbf{F}^{\gamma 1} = \mathbf{F}^1\left(\mathbf{X}^1, t^\gamma\right) \equiv \mathbf{F}\left(\mathbf{X}, t^\gamma\right)$. In other words, when generation γ first comes into existence, its reference configuration is the current configuration at time t^γ. Note that $\mathbf{F}^{\gamma 1}$ is a time-invariant (though not necessarily homogeneous) quantity that is determined uniquely at the birth of a generation.

The state of stress in a multigeneration solid is given by

$$\boldsymbol{\sigma} = \sum_\gamma \frac{1}{J^{\gamma 1}} \boldsymbol{\sigma}^\gamma\left(\mathbf{F}^\gamma\right)$$

where $\boldsymbol{\sigma}^\gamma\left(\mathbf{F}^\gamma\right)$ is the state of stress in the generation γ, as would be evaluated from a strain energy density function whose reference configuration is $\mathbf{X}^\gamma$. In the above equation, $J^{\gamma 1} = \det \mathbf{F}^{\gamma 1}$ and the factor $1/J^{\gamma 1}$ ensures that the strain energy density of each generation is properly normalized the volume of the material in the master reference configuration $\mathbf{X}^1$, when summing up the stresses in all the generations.

Multigeneration solids typically exhibit residual stresses when $\mathbf{F}^{\gamma 1}$ is inhomogeneous.

4.3.1. General Specification of Multigeneration Solids

The material type for a multigeneration solid is *"multigeneration"*[*]. This material describes a mixture of elastic solids, each created in a specific generation. It is a container for any combination of the elastic materials described.

`<generation>`	Definition of a generation.

The `<generation>` tag defines a new generation. It takes the following child elements.

`<start_time>`	"birth"-time for this generation	[**t**]
`<solid>`	Specification of the constitutive model for this generation	

The *solid* element defins the solid matrix constitutive relation and associated material properties.

Example:

```
<material id="1" name="Growing Solid" type="multigeneration">
     <generation id="1">
          <start_time>0.0</start_time>
          <solid type="Holmes-Mow">
               <density>1</density>
               <E>1</E>
```

[*] As of FEBio version 2.0, the format for definining multi-generation materials has changed. The previous format where the generations are defined in the *Globals* section is no longer supported.

```
                <v>0</v>
                <beta>0.1</beta>
          </solid>
     </generation>
     <generation id="2">
          <start_time>1.0</start_time>
          <solid type="Holmes-Mow">
                <density>1</density>
                <E>1</E>
                <v>0</v>
                <beta>0.1</beta>
          </solid>
     </generation>
</material>
```

The corresponding value of t^{γ} for each of the generations is provided in the `<Globals>` section.

Example:

```
<Globals>
     <Generations>
          <gen id="1">0.0</gen>
          <gen id="2">1.0</gen>
     </Generations>
</Globals>
```

4.4. Biphasic Materials

Biphasic materials may be used to model a porous medium consisting of a mixture of a porous-permeable solid matrix and an interstitial fluid. Each of these mixture constituents is assumed to be intrinsically incompressible. This means that the true densities of the solid and fluid are invariant in space and time; this assumption further implies that a biphasic mixture will undergo zero volume change when subjected to a hydrostatic fluid pressure. Yet the mixture is compressible because the pores of the solid matrix may gain or lose fluid under general loading conditions. Therefore, the constitutive relation of the solid matrix should be chosen from the list of compressible materials provided in Section 4.1.3. The user is referred to the *FEBio Theory Manual* for a general description of the biphasic theory.

In addition to selecting a constitutive relation for the solid matrix, a constitutive relation must also be selected for the hydraulic permeability of the interstitial fluid flowing within the porous deformable solid matrix. The hydraulic permeability relates the volumetric flux of the fluid relative to the solid, $\mathbf{w}$, to the interstitial fluid pressure gradient, ∇p, according to

$$\mathbf{w} = -\mathbf{k} \cdot \nabla p$$

where $\mathbf{k}$ is the hydraulic permeability tensor. (Note that this expression does not account for the contribution of external body forces on the fluid.)

4.4.1. General Specification of Biphasic Materials

The material type for a biphasic material is *"biphasic"*. The following parameters must be defined:

`<solid>`	Specification of the solid matrix	
`<phi0>`	solid volume fraction φ_r^s in the reference configuration ($0 < \varphi_r^s < 1$)	[ ]
`<permeability>`	Specification of the hydraulic permeability	
`<solvent_supply>`	Specification of the fluid supply $\hat{\varphi}^w$	

The <solid> tag encloses a description of the solid matrix constitutive relation and associated material properties, as may be selected from the list provided in Section 4.1.3. The <permeability> tag encloses a description of the permeability constitutive relation and associated material properties, as may be selected from the list presented in Section 4.4.2. The parameter `<phi0>` must be greater than 0 (no solid) and less than 1 (no porosity). The volume fraction of fluid (also known as the porosity) in the reference configuration is given by $1-\varphi_r^s$.

Example:

```
<material id="1" name="Biphasic tissue" type="biphasic">
      <solid name="Elasticity" type="neo-Hookean">
            <E>1.0</E>
            <v>0.3</v>
      </solid>
      <phi0>0.2</phi0>
      <permeability name="Permeability" type="perm-const-iso">
            ... (description of permeability material)
      </permeability>
</material>
```

4.4.2. Permeability Materials

Permeability materials provide a constitutive relation for the hydraulic permeability of a biphasic material.

4.4.2.1. Constant Isotropic Permeability

The material type for constant isotropic permeability is "*const-iso-perm*". The following material parameters must be defined:

<perm>	hydraulic permeability k	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]

This isotropic material model uses the biphasic theory for describing the time-dependent material behavior of materials that consist of both a solid and fluid phase [32, 33].

When the permeability is isotropic,

$$\mathbf{k} = k\,\mathbf{I}$$

For this material model, k is constant. Generally, this assumption is only reasonable when strains are small.

Example:

```
<permeability name="Permeability" type="const-iso-perm">
      <perm>0.001</perm>
</permeability>
```

4.4.2.2. Holmes-Mow

The material type for Holmes-Mow permeability is *"perm-Holmes-Mow"*. The following material parameters need to be defined:

<perm>	isotropic hydraulic permeability k_0 in the reference state	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<M>	exponential strain-dependence coefficient M ($M \geq 0$)	[ ]
<alpha>	power-law exponent α ($\alpha \geq 0$)	[ ]

This isotropic material is similar to the constant isotropic permeability material described above, except that it uses a strain-dependent permeability tensor [28]:

$$\mathbf{k} = k(J)\mathbf{I},$$

where,

$$k(J) = k_0 \left(\frac{J-\varphi_0}{1-\varphi_0} \right)^{\alpha} e^{\frac{1}{2}M\left(J^2-1\right)},$$

and J is the Jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient.

Example:
This example defines a permeability material of the Holmes-Mow type.

```
<permeability type="perm-Holmes-Mow">
      <perm>0.001</perm>
      <M>1.5</M>
      <alpha>2</alpha>
</permeability>
```

4.4.2.3. Referentially Isotropic Permeability

The material type for a biphasic material with strain-dependent permeability which is isotropic in the reference configuration is *"perm-ref-iso"*. The following material parameters need to be defined:

`<perm0>`	hydraulic permeability k_{0r}	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<perm1>`	hydraulic permeability k_{1r}	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<perm2>`	hydraulic permeability k_{2r}	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<M>`	exponential strain-dependence coefficient M ($M \geq 0$)	[ ]
`<alpha>`	power-law exponent α ($\alpha \geq 0$)	[ ]

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\mathbf{k} = \left(k_{0r}\mathbf{I} + \frac{k_{1r}}{J^2}\mathbf{b} + \frac{k_{2r}}{J^4}\mathbf{b}^2 \right) \left(\frac{J-\varphi_r^s}{1-\varphi_r^s} \right) e^{M\left(J^2-1\right)/2},$$

where J is the Jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient, and $\mathbf{b} = \mathbf{F}\cdot\mathbf{F}^T$ is the left Cauchy-Green tensor. Note that the permeability in the reference state ($\mathbf{F} = \mathbf{I}$) is isotropic and given by $\mathbf{k} = \left(k_{0r} + k_{1r} + k_{2r}\right)\mathbf{I}$.

Example:

```
<permeability name="Permeability" type="perm-ref-iso">
      <perm0>0.001</perm0>
      <perm1>0.005</perm1>
      <perm2>0.002</perm2>
      <M>1.5</M>
      <alpha>2</alpha>
</permeability>
```

4.4.2.4. Referentially Orthotropic Permeability

The material type for a poroelastic material with strain-dependent permeability which is orthotropic in the reference configuration is *"perm-ref-ortho"*. The following material parameters need to be defined:

`<perm0>`	isotropic hydraulic permeability k_{0r}	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<perm1>`	hydraulic permeabilities k_{1r}^a along orthogonal directions ($a=1,2,3$)	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<perm2>`	hydraulic permeabilities k_{2r}^a along orthogonal directions ($a=1,2,3$)	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
`<M0>`	isotropic exponential strain-dependence coefficient M_0 ($M_0 \geq 0$)	[ ]
`<M>`	orthotropic exponential strain-dependence coefficient M_a ($a=1,2,3$, $M_a \geq 0$)	[ ]
`<alpha0>`	isotropic power-law exponent α_0 ($\alpha_0 \geq 0$)	[ ]
`<alpha>`	power-law exponent α_a ($a=1,2,3$, $\alpha_a \geq 0$)	[ ]

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\mathbf{k} = k_0\mathbf{I} + \sum_{a=1}^{3} k_1^a \mathbf{m}_a + k_2^a \left(\mathbf{m}_a \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{m}_a\right),$$

where,

$$k_0 = k_{0r}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_0} e^{M_0\left(J^2-1\right)/2}$$

$$k_1^a = \frac{k_{1r}^a}{J^2}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3 \ ,$$

$$k_2^a = \frac{k_{2r}^a}{2J^4}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3$$

J is the Jacobian of the deformation, i.e. $J=\det\mathbf{F}$ where $\mathbf{F}$ is the deformation gradient. $\mathbf{m}_a$ are second order tensors representing the spatial structural tensors describing the orthogonal planes of symmetry, given by

$$\mathbf{m}_a = \mathbf{F}\cdot\left(\mathbf{V}_a \otimes \mathbf{V}_a\right)\cdot\mathbf{F}^T, \quad a=1-3,$$

where $\mathbf{V}_a$ are orthonormal vectors normal to the planes of symmetry (defined as described in Section 4.1.1). Note that the permeability in the reference state ($\mathbf{F}=\mathbf{I}$) is given by $\mathbf{k} = k_{0r}\mathbf{I} + \sum_{a=1}^{3}\left(k_{1r}^a + k_{2r}^a\right)\mathbf{V}_a \otimes \mathbf{V}_a$.

Example:

```
<permeability name="Permeability" type="perm-ref-ortho">
      <perm0>0.001</perm0>
      <perm1>0.01, 0.02, 0.03</perm1>
```

```
      <perm2>0.001, 0.002, 0.003</perm2>
      <M0>0.5</M0>
      <M>1.5, 2.0, 2.5</M>
      <alpha0>1.5</alpha0>
      <alpha>2, 2.5, 3</alpha>
</material>
```

4.4.2.5. Referentially Transversely Isotropic Permeability

The material type for a biphasic material with strain-dependent permeability which is transversely isotropic in the reference configuration is *"perm-ref-trans-iso"*. The following material parameters need to be defined:

<perm0>	isotropic hydraulic permeability k_{0r}	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<perm1A>	axial hydraulic permeability k_{1r}^A	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<perm2A>	axial hydraulic permeability k_{2r}^A	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<perm1T>	transverse hydraulic permeability k_{1r}^T	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<perm2T>	transverse hydraulic permeability k_{2r}^T	[$\mathbf{L}^4/\mathbf{F}\cdot\mathbf{t}$]
<M0>	isotropic exponential strain-dependence coefficient M_0 ($M_0 \geq 0$)	[ ]
<MA>	axial exponential strain-dependence coefficient M_A ($M_A \geq 0$)	[ ]
<MT>	transverse exponential strain-dependence coefficient M_T ($M_T \geq 0$)	[ ]
<alpha0>	isotropic power-law exponent α_0 ($\alpha_0 \geq 0$)	[ ]
<alphaA>	axial power-law exponent α_A ($\alpha_A \geq 0$)	[ ]
<alphaT>	transverse power-law exponent α_T ($\alpha_T \geq 0$)	[ ]

This material uses a strain-dependent permeability tensor that accommodates strain-induced anisotropy:

$$\mathbf{k} = k_{0r}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_0} e^{M_0\left(J^2-1\right)/2}\mathbf{I}$$
$$+\left(\frac{k_{1r}^T}{J^2}\left(\mathbf{b}-\mathbf{m}\right)+\frac{k_{2r}^T}{2J^4}\left[2\mathbf{b}^2-\left(\mathbf{m}\cdot\mathbf{b}+\mathbf{b}\cdot\mathbf{m}\right)\right]\right)\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_T} e^{M_T\left(J^2-1\right)/2},$$
$$+\left(\frac{1}{J^2}k_{1r}^A\mathbf{m}+\frac{1}{2J^4}k_{2r}^A\left(\mathbf{m}\cdot\mathbf{b}+\mathbf{b}\cdot\mathbf{m}\right)\right)\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_A} e^{M_A\left(J^2-1\right)/2}$$

where J is the Jacobian of the deformation, i.e. $J=\det\mathbf{F}$ where $\mathbf{F}$ is the deformation gradient, and $\mathbf{b}=\mathbf{F}\cdot\mathbf{F}^T$ is the left Cauchy-Green tensor. $\mathbf{m}$ is a second order tensor representing the spatial structural tensor describing the axial direction, given by

$$\mathbf{m}=\mathbf{F}\cdot\left(\mathbf{V}\otimes\mathbf{V}\right)\cdot\mathbf{F}^T,$$

where $\mathbf{V}$ is a unit vector along the axial direction (defined as described in Section 4.1.1). Note that the permeability in the reference state ($\mathbf{F}=\mathbf{I}$) is given by $\mathbf{k}=\left(k_{0r}+k_{1r}^T+k_{2r}^T\right)\mathbf{I}+\left(k_{1r}^A-k_{1r}^T+k_{2r}^A-k_{2r}^T\right)\left(\mathbf{V}\otimes\mathbf{V}\right)$.

Example:

```
<permeability name="Permeability" type="perm-ref-trans-iso">
      <perm0>0.002</perm0>
      <perm1A>0.01</perm1A>
      <perm2A>0.01</perm2A>
```

```
      <perm1T>0.001</perm1T>
      <perm2T>0.05</perm2T>
      <M0>1.0</M0>
      <MA>0.5</MA>
      <MT>1.5</MT>
      <alpha0>1.0</alpha0>
      <alphaA>0.5</alphaA>
      <alphaT>2.0</alphaT>
</permeability>
```

4.4.3. Fluid Supply Materials

Fluid supply materials may be used to simulate an external source of fluid, such as supply from microvasculature that is not modeled explicitly. The fluid supply term, $\hat{\varphi}^w$, appears in the mass balance relation for the mixture,

$$\operatorname{div}\left(\mathbf{v}^s + \mathbf{w}\right) = \hat{\varphi}^w \ .$$

$\hat{\varphi}^w$ has units of reciprocal time [$\mathbf{t}^{-1}$]; it represents the rate at which the volume fraction of fluid changes with time.

4.4.3.1. Starling Equation

The material type for Starling's equation for fluid supply is *"Starling"*. The following material parameters need to be defined:

<kp>	hydraulic filtration coefficient, k_p	[**L**^{2}/**F**·**t**]
<pv>	fluid pressure in external source, p_v	[**P**]

The fluid supply is given by Starling's equation,

$$\hat{\varphi}^w = k_p \left(p_v - p \right) ,$$

where p is the fluid pressure in the biphasic material.

Example:
This example defines a fluid supply material of the Starling type.

```
<solvent_supply type="Starling">
      <kp>0.001</kp>
      <pv>0.1</pv>
</solvent_supply>
```

4.5. Biphasic-Solute Materials

Biphasic-solute materials may be used to model the transport of a solvent and a solute in a neutral porous solid matrix, under isothermal conditions. Each of these mixture constituents is assumed to be intrinsically incompressible. This means that their true densities are invariant in space and time; this assumption further implies that a biphasic-solute mixture will undergo zero volume change when subjected to a hydrostatic fluid pressure. Yet the mixture is compressible because the pores of the solid matrix may gain or lose fluid under general loading conditions. Therefore, the constitutive relation of the solid matrix should be chosen from the list of compressible materials provided in Section 4.1.3. The volume fraction of the solute is assumed to be negligible relative to the volume fractions of the solid or solvent. This means that the mixture will not change in volume as the solute concentration changes. As the solute transports through the mixture, it may experience frictional interactions with the solvent and the solid. This friction may act as a hindrance to the solute transport, or may help convect the solute through the mixture. The distinction between frictional exchanges with the solvent and solid is embodied in the specification of two diffusivities for the solute: The free diffusivity, which represents diffusivity in the absence of a solid matrix (frictional exchange only with the solvent) and the mixture diffusivity, which represents the combined frictional interactions with the solvent and solid matrix. The user is referred to the *FEBio Theory Manual* for a more detailed description of the biphasic-solute theory.

Due to steric volume exclusion and short-range electrostatic interactions, the solute may not have access to all of the pores of the solid matrix. In other words, only a fraction κ of the pores is able to accommodate a solute of a particular size ($0<\kappa\le 1$). Furthermore, the activity γ of the solute (the extent by which the solute concentration influences its chemical potential) may be similarly altered by the surrounding porous solid matrix. Therefore, the combined effects of volume exclusion and attenuation of activity may be represented by the effective solubility $\tilde{\kappa}=\kappa/\gamma$, such that the chemical potential μ of the solute is given by

$$\mu=\mu_0(\theta)+\frac{R\theta}{M}\ln\frac{c}{\tilde{\kappa}}.$$

In this expression, μ_0 is the solute chemical potential at some reference temperature θ; c is the solute concentration on a solution-volume basis (number of moles of solute per volume of interstitial fluid in the mixture); M is the solute molecular weight (an invariant quantity); and R is the universal gas constant. In a biphasic-solute material, a constitutive relation is needed for $\tilde{\kappa}$; in general, $\tilde{\kappa}$ may be a function of the solid matrix strain and the solute concentration. In FEBio, the dependence of the effective solubility on the solid matrix strain is currently constrained to a dependence on $J=\det\mathbf{F}$.

In a biphasic-solute material, the interstitial fluid pressure p is influenced by both mechanical and chemical environments. In other words, this pressure includes both mechanical and osmotic contributions, the latter arising from the presence of the solute. The solvent mechano-chemical potential $\tilde{\mu}^w$ is given by

$$\tilde{\mu}^w = \mu_0^w(\theta) + \frac{1}{\rho_T^w}(p - R\theta\Phi c),$$

where μ_0^w is the solvent chemical potential at some reference temperature θ; ρ_T^w is the true density of the solvent (an invariant property for an intrinsically incompressible fluid); and Φ is the osmotic coefficient which represents the extent by which the solute concentration influences the solvent chemical potential. In a biphasic-solute material, a constitutive relation is needed for Φ; in general, Φ may be a function of the solid matrix strain and the solute concentration. In FEBio, the dependence of the osmotic coefficient on the solid matrix strain is currently constrained to a dependence on $J = \det \mathbf{F}$.

The solute mechano-chemical potential is nearly equal to its chemical potential because the solute volume fraction is assumed to be negligible. In general, momentum and energy balances evaluated across a boundary surface in a biphasic-solute mixture require that the mechano-chemical potentials of solvent and solute be continuous across that surface. These continuity requirements are enforced automatically in FEBio by defining the effective fluid pressure $\tilde{p}$ and solute concentration $\tilde{c}$ as

$$\tilde{p} = p - R\theta\Phi c$$
$$\tilde{c} = \frac{c}{\tilde{\kappa}} \quad .$$

Therefore, nodal variables in FEBio consist of the solid matrix displacement $\mathbf{u}$, the effective fluid pressure $\tilde{p}$, and the effective solute concentration $\tilde{c}$. Essential boundary conditions must be imposed on these variables, and not on the actual pressure p or concentration c. (In a biphasic material however, since $c = 0$, the effective and actual fluid pressures are the same, $p = \tilde{p}$.)

The mixture stress in a biphasic-solute material is given by $\boldsymbol{\sigma} = -p\mathbf{I} + \boldsymbol{\sigma}^e$, where $\boldsymbol{\sigma}^e$ is the stress arising from the solid matrix strain. The mixture traction on a surface with unit outward normal $\mathbf{n}$ is $\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n}$. This traction is continuous across the boundary surface. Therefore, the corresponding natural boundary condition for a biphasic-solute mixture is $\mathbf{t} = \mathbf{0}$. (In other words, if no boundary condition is imposed on the solid matrix displacement or mixture traction, the natural boundary condition is in effect.)

The natural boundary conditions for the solvent and solute are similarly $\mathbf{w} \cdot \mathbf{n} = 0$ and $\mathbf{j} \cdot \mathbf{n} = 0$, where $\mathbf{w}$ is the volumetric flux of solvent relative to the solid and $\mathbf{j}$ is the molar flux of solute relative to the solid. In general, $\mathbf{w}$ and $\mathbf{j}$ are given by

$$\mathbf{w} = -\tilde{\mathbf{k}} \cdot \left(\nabla\tilde{p} + R\theta \frac{\tilde{\kappa}}{d_0} \mathbf{d} \cdot \nabla\tilde{c} \right)$$
$$\mathbf{j} = \tilde{\kappa}\mathbf{d} \cdot \left(-\varphi^w \nabla\tilde{c} + \frac{\tilde{c}}{d_0} \mathbf{w} \right) \quad ,$$

where

$$\tilde{\mathbf{k}} = \left[\mathbf{k}^{-1} + \frac{R\theta}{\varphi^w} \frac{\tilde{\kappa} c}{d_0} \left(\mathbf{I} - \frac{\mathbf{d}}{d_0} \right) \right]^{-1}$$

is the effective hydraulic permeability of the interstitial fluid solution (solvent and solute) through the porous solid matrix; $\mathbf{k}$ is the hydraulic permeability of the solvent through the porous solid matrix; $\mathbf{d}$ is the solute diffusivity through the mixture (frictional interactions with solvent and solid); and d_0 is the solute free diffusivity (frictional interactions with solvent only). $\varphi^w \approx 1 - \varphi^s$ is the solid matrix porosity in the current configuration. The above expressions for the solvent and solute flux do not account for external body forces.

4.5.1. Guidelines for Biphasic-Solute Analyses

4.5.1.1. Prescribed Boundary Conditions

In most analyses, it may be assumed that the ambient fluid pressure in the external environment is zero, thus $p_* = 0$, where the subscripted asterisk is used to denote environmental conditions. The ambient solute concentration may be represented by c_*. It follows that the effective fluid pressure in the external environment is $\tilde{p}_* = -R\theta\Phi_* c_*$ and the effective concentration is $\tilde{c}_* = c_* \big/ \tilde{\kappa}_*$. Therefore, in biphasic-solute analyses, whenever the external environment contains a solute at a concentration of c_*, the user must remember to prescribe non-zero boundary conditions for the effective solute concentration *and* the effective fluid pressure.

Letting $p_* = 0$ also implies that prescribed mixture normal tractions (Section 3.10.2.3.) represent only the traction above ambient conditions. Note that users are not obligated to assume that $p_* = 0$. However, if a non-zero value is assumed for the ambient pressure, then users must remember to incorporate this non-zero value whenever prescribing mixture normal tractions.

4.5.1.2. Prescribed Initial Conditions

When a biphasic-solute material is initially exposed to a given external environment with effective pressure $\tilde{p}_*$ and effective concentration $\tilde{c}_*$, the initial conditions inside the material should be set to $\tilde{p} = \tilde{p}_*$ and $\tilde{c} = \tilde{c}_*$ in order to produce the correct initial state. The values of $\tilde{p}_*$ and $\tilde{c}_*$ should be evaluated as described in Section 8.5.2

4.5.2. General Specification of Biphasic-Solute Materials

The material type for a biphasic-solute material is *"biphasic-solute"*. Constitutive relations must be provided for the solid matrix, the hydraulic permeability $\mathbf{k}$, the solute diffusivities $\mathbf{d}$ and d_0, the effective solubility $\tilde{\kappa}$ and the osmotic coefficient Φ. Therefore, the following parameters must be defined:

`<solid>`	specification of the solid matrix
`<phi0>`	solid volume fraction φ_r^s in the reference configuration
`<permeability>`	specification of the hydraulic permeability $\mathbf{k}$
`<osmotic_coefficient>`	specification of the osmotic coefficient Φ $\mathbf{d}$ d_0
`<solute>`	specification of the solute properties

The `<solid>` tag encloses a description of the solid matrix constitutive relation and associated material properties, as may be selected from the list provided in Section 4.1.3. The solid volume fraction in the reference configuration, `<phi0>`, must be greater than 0 (no solid) and less than 1 (only solid). The volume fraction of fluid (also known as the porosity) in the reference configuration is given by $1-\varphi_0$. The `<permeability>` tag encloses a description of the permeability constitutive relation and associated material properties, as may be selected from the list presented in Section 4.4.2.

The `<solute>` tag provides a description of the solute in the biphasic-solute mixture. This tag includes the required *sol* attribute, which should reference a solute *id* from the `<Solutes>` description in the `<Globals>` section (Section 3.5.2). The following parameters must be defined in this description:

`<diffusivity>`	specification of the solute diffusivities $\mathbf{d}$ and d_0
`<solubility>`	specification of the solute effective solubility $\tilde{\kappa}$

The `<diffusivity>` and `<solubility>` tags enclose descriptions of materials that may be selected from the lists presented in Sections 4.5.3 and 4.5.4, respectively. Each solute tag must include a "sol" attribute

Example:

```
<material id="1" name="Biological tissue" type="biphasic-solute">
      <solid name="Elasticity" type="neo-Hookean">
            <E>1.0</E>
            <v>0.3</v>
      </solid>
      <phi0>0.2</phi0>
      <permeability name="Permeability" type="perm-const-iso">
            ... (description of permeability material)
      </permeability>
      <osmotic_coefficient name="Osmotic" type="osm-coef-const">
            ... (description of osmotic coefficient material)
      </osmotic_coefficient>
```

```
      <solute sol="1">
            <diffusivity name="Diffusivity" type="diff-const-iso">
                  ... (description of diffusivity material)
            </diffusivity>
            <solubility name="Solubility" type="solub-const">
                  ... (description of solubility material)
            </solubility>
      </solute>
</material>
```

When a biphasic-solute material is employed in an analysis, it is also necessary to specify the values of the universal gas constant R [**F**·**L**/**n**·**T**] and absolute temperature θ [**T**] under `<Constants>` in the `<Globals>` section, using a self-consistent set of units. A solute must also be defined in the `<Solutes>` section, whose id should be associated with the "`sol`" attribute in the solute material description.

Example:

```
<Globals>
      <Constants>
            <R>8.314</R>
            <T>298</T>
      </Constants>
      <Solutes>
            <solute id="1" name="neutral">
                  <charge_number>0</charge_number>
            </solute>
      </Solutes>
</Globals>
```

It is also possible to create models with biphasic-solute materials that use different solutes in different regions. In that case, introduce additional solute entries in the `<Solutes>` section and refer to those solute ids in the biphasic-solute material descriptions.

4.5.3. Diffusivity Materials

Diffusivity materials provide a constitutive relation for the solute diffusivity in a biphasic-solute material. In general, the diffusivity tensor **d** may be a function of strain and solute concentration.

4.5.3.1. Constant Isotropic Diffusivity

The material type for constant isotropic diffusivity materials is *"diff-const-iso"*. The following material parameters must be defined:

`<free_diff>`	free diffusivity d_0 of solute (diffusivity in solution)	[**L**^{2}/**t**]
`<diff>`	constant isotropic diffusivity d of solute in biphasic-solute mixture	[**L**^{2}/**t**]

When the permeability is isotropic,

$$\mathbf{d} = d\,\mathbf{I}$$

For this material model, d is constant. This assumption is only true when strains are small. Note that the user must specify $d \le d_0$, since a solute cannot diffuse through the biphasic-solute mixture faster than in free solution.

Example:

```
<diffusivity name="Permeability" type="diff-const-iso">
      <free_diff>1e-9</ free_diff >
      <diff>0.5e-9</diff>
</diffusivity>
```

4.5.3.2. Constant Orthotropic Diffusivity

The material type for constant orthotropic diffusivity materials is "*diff-const-ortho*". The following material parameters must be defined:

`<free_diff>`	free diffusivity d_0 of solute (diffusivity in solution)	[$\mathbf{L}^2/\mathbf{t}$]
`<diff>`	diffusivities d^a along orthogonal directions ($a=1,2,3$)	[$\mathbf{L}^2/\mathbf{t}$]

When the permeability is orhotropic,

$$\mathbf{d}=\sum_{a=1}^{3} d^a \mathbf{V}_a \otimes \mathbf{V}_a$$

where $\mathbf{V}_a$ are orthonormal vectors normal to the planes of symmetry (defined as described in Section 4.1.1). For this material model, d^a's are constant. Therefore this model should be used only when strains are small. Note that the user must specify $d^a \le d_0$, since a solute cannot diffuse through the biphasic-solute mixture faster than in free solution.

Example:

```
<diffusivity name="Permeability" type="diff-const-ortho">
      <free_diff>0.005</ free_diff >
      <diff>0.002,0.001,0.003</diff>
</diffusivity>
```

4.5.3.3. Referentially Isotropic Diffusivity

The material type for a strain-dependent diffusivity tensor which is isotropic in the reference configuration is *"diff-ref-iso"*. The following material parameters need to be defined:

<free_diff>	free diffusivity d_0	[$\mathbf{L}^2/\mathbf{t}$]
<diff0>	diffusivity d_{0r}	[$\mathbf{L}^2/\mathbf{t}$]
<diff1>	diffusivity d_{1r}	[$\mathbf{L}^2/\mathbf{t}$]
<diff2>	diffusivity d_{2r}	[$\mathbf{L}^2/\mathbf{t}$]
<M>	exponential strain-dependence coefficient M ($M \geq 0$)	[ ]
<alpha>	power-law exponent α ($\alpha \geq 0$)	[ ]

This material uses a strain-dependent diffusivity tensor that accommodates strain-induced anisotropy:

$$\mathbf{d} = \left(d_{0r}\mathbf{I} + \frac{d_{1r}}{J^2}\mathbf{b} + \frac{d_{2r}}{J^4}\mathbf{b}^2 \right) \left(\frac{J - \varphi_r^s}{1 - \varphi_r^s} \right) e^{M\left(J^2-1\right)/2},$$

where J is the jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient, and $\mathbf{b} = \mathbf{F} \cdot \mathbf{F}^T$ is the left Cauchy-Green tensor. Note that the diffusivity in the reference state ($\mathbf{F} = \mathbf{I}$) is isotropic and given by $\mathbf{d} = \left(d_{0r} + d_{1r} + d_{2r} \right)\mathbf{I}$.

Example:

```
<diffusivity name="Diffusivity" type="diff-ref-iso">
      <phi0>0.2</phi0>
      <free_diff>0.005</free_diff>
      <diff0>0.001</diff0>
      <diff1>0.005</diff1>
      <diff2>0.002</diff2>
      <M>1.5</M>
      <alpha>2</alpha>
</diffusivity>
```

4.5.3.4. Referentially Orthotropic Diffusivity

The material type for a strain-dependent diffusivity which is orthotropic in the reference configuration is *"diff-ref-ortho"*. The following material parameters need to be defined:

`<free_diff>`	free diffusivity d_0	[$\mathbf{L}^2/\mathbf{t}$]
`<diff0>`	isotropic diffusivity d_{0r}	[$\mathbf{L}^2/\mathbf{t}$]
`<diff1>`	diffusivities d_{1r}^a along orthogonal directions ($a=1,2,3$)	[$\mathbf{L}^2/\mathbf{t}$]
`<diff2>`	diffusivities d_{2r}^a along orthogonal directions ($a=1,2,3$)	[$\mathbf{L}^2/\mathbf{t}$]
`<M0>`	isotropic exponential strain-dependence coefficient M_0 ($M_0 \geq 0$)	[ ]
`<M>`	orthotropic exponential strain-dependence coefficient M_a ($a=1,2,3$, $M_a \geq 0$)	[ ]
`<alpha0>`	isotropic power-law exponent α_0 ($\alpha_0 \geq 0$)	[ ]
`<alpha>`	power-law exponent α_a ($a=1,2,3$, $\alpha_a \geq 0$)	[ ]

This material uses a strain-dependent diffusivity tensor that accommodates strain-induced anisotropy:

$$\mathbf{d} = d_0\mathbf{I} + \sum_{a=1}^{3} d_1^a \mathbf{m}_a + d_2^a \left(\mathbf{m}_a \cdot \mathbf{b} + \mathbf{b} \cdot \mathbf{m}_a\right),$$

where,

$$d_0 = d_{0r}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_0} e^{M_0\left(J^2-1\right)/2}$$

$$d_1^a = \frac{d_{1r}^a}{J^2}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3\ ,$$

$$d_2^a = \frac{d_{2r}^a}{2J^4}\left(\frac{J-\varphi_r^s}{1-\varphi_r^s}\right)^{\alpha_a} e^{M_a\left(J^2-1\right)/2}, \quad a=1,2,3$$

J is the Jacobian of the deformation, i.e. $J=\det\mathbf{F}$ where $\mathbf{F}$ is the deformation gradient. $\mathbf{m}_a$ are second order tensors representing the spatial structural tensors describing the orthogonal planes of symmetry, given by

$$\mathbf{m}_a = \mathbf{F}\cdot\left(\mathbf{V}_a \otimes \mathbf{V}_a\right)\cdot\mathbf{F}^T, \quad a=1-3,$$

where $\mathbf{V}_a$ are orthonormal vectors normal to the planes of symmetry (defined as described in Section 4.1.1). Note that the diffusivity in the reference state ($\mathbf{F}=\mathbf{I}$) is given by $\mathbf{d} = d_{0r}\mathbf{I} + \sum_{a=1}^{3}\left(d_{1r}^a + d_{2r}^a\right)\mathbf{V}_a \otimes \mathbf{V}_a$.

Example:

```
<diffusivity name="Diffusivity" type="diff-ref-ortho">
      <phi0>0.2</phi0>
```

```
      <free_diff>0.005</free_diff>
      <diff00>0.001</diff00>
      <diff1>0.01, 0.02, 0.03</diff1>
      <diff2>0.001, 0.002, 0.003</diff2>
      <M0>0.5</M0>
      <M>1.5, 2.0, 2.5</M>
      <alpha0>1.5</alpha0>
      <alpha>2, 2.5, 3</alpha>
</diffusivity>
```

4.5.3.5. Albro Isotropic Diffusivity

The material type for a porosity and concentration-dependent diffusivity which is isotropic is *"diff-Albro-iso"*. The following material parameters need to be defined:

<free_diff>	free diffusivity d_0	[**L**^{2}/**t**]
<cdinv>	inverse of characteristic concentration for concentration-dependence c_D^{-1}	[**L**^{3}/**n**]
<alphad>	coefficient of porosity-dependence α_D	[ ]

This material uses a porosity and concentration-dependent diffusivity tensor that remains isotropic with deformation:

$$\mathbf{d} = d_0 \exp\left(-\alpha_D \frac{1-\varphi^w}{\varphi^w} - \frac{c}{c_D}\right)\mathbf{I},$$

where $c_D^{-1} = 1/c_D$ and the porosity φ^w varies with deformation according to the kinematic constraint

$$\varphi^w = 1 - \frac{\varphi_r^s}{J}.$$

J is the Jacobian of the deformation, i.e. $J = \det \mathbf{F}$ where $\mathbf{F}$ is the deformation gradient and φ_r^s is the referential solid volume fraction. Here, c represents the actual concentration of the solute whose diffusivity is given by $\mathbf{d}$. This constitutive relation is based on the experimental findings reported by Albro et al. [34].

Example:

```
<solute sol="1">
      <diffusivity type="diff-Albro-iso">
            <free_diff>123e-6</free_diff>
            <alphad>18</alphad>
            <cdinv>0.0625</cdinv>
      </diffusivity>
      <solubility type="solub-const">
            <solub>1</solub>
      </solubility>
</solute>
```

4.5.4. Solubility Materials

4.5.4.1. Constant Solubility

The material type for constant solubility materials is *"solub-const"*. The following material parameters must be defined:

<solub>	solubility $\tilde{\kappa}$	[ ]

For this material model, $\tilde{\kappa}$ is constant.

Example:

```
<solubility name="Solubility" type="solub-const">
      <solub>1</solub>
</solubility>
```

4.5.5. Osmotic Coefficient Materials

4.5.5.1. Constant Osmotic Coefficient

The material type for constant osmotic coefficient materials is "*osm-coef-const*". The following material parameters must be defined:

`<osmcoef>`	Osmotic coefficient Φ	[ ]

For this material model, Φ is constant.

Example:

```
<osmotic_coefficient name="Osmotic coefficient" type="osm-coef-const">
      <osmcoef>1</osmcoef>
</osmotic_coefficient>
```

4.6. Triphasic and Multiphasic Materials

Triphasic materials may be used to model the transport of a solvent and a pair of monovalent salt counter-ions (two solutes with charge numbers +1 and -1) in a charged porous solid matrix, under isothermal conditions. Multiphasic materials may be used to model the transport of a solvent and any number of neutral or charged solutes; a triphasic mixture is a special case of a multiphasic mixture. Each of the mixture constituents is assumed to be intrinsically incompressible. This means that their true densities are invariant in space and time; this assumption further implies that a multiphasic mixture will undergo zero volume change when subjected to a hydrostatic fluid pressure. Yet the mixture is compressible because the pores of the solid matrix may gain or lose fluid under general loading conditions. Therefore, the constitutive relation of the solid matrix should be chosen from the list of compressible materials provided in Section 4.1.3. The volume fraction of the solutes is assumed to be negligible relative to the volume fractions of the solid or solvent. This means that the mixture will not change in volume as the solute concentrations change. As the solutes transport through the mixture, they may experience frictional interactions with the solvent and the solid. This friction may act as a hindrance to the solute transport, or may help convect the solutes through the mixture. The distinction between frictional exchanges with the solvent and solid is embodied in the specification of two diffusivities for each solute: The free diffusivity, which represents diffusivity in the absence of a solid matrix (frictional exchange only with the solvent) and the mixture diffusivity, which represents the combined frictional interactions with the solvent and solid matrix. The user is referred to the *FEBio Theory Manual* for a more detailed description of triphasic and multiphasic theory.

Due to steric volume exclusion and short-range electrostatic interactions, a solute α $\alpha=+$ $\alpha=-$ may not have access to all of the pores of the solid matrix. In other words, only a fraction κ^α of the pores is able to accommodate solute α ($0<\kappa^\alpha\le 1$). Furthermore, the activity γ^α of solute α (the extent by which the solute concentration influences its chemical potential) may be similarly altered by the surrounding porous solid matrix. Therefore, the combined effects of volume exclusion and attenuation of activity may be represented by the effective solubility $\hat{\kappa}^\alpha=\kappa^\alpha/\gamma^\alpha$, such that the chemical potential μ of the solute is given by

$$\mu^\alpha=\mu_0^\alpha(\theta)+\frac{R\theta}{M^\alpha}\ln\frac{c^\alpha}{\hat{\kappa}^\alpha}.$$

In this expression, μ_0^α is the solute chemical potential at some reference temperature θ; c^α is the solute concentration on a solution-volume basis (number of moles of solute per volume of interstitial fluid in the mixture); M^α is the solute molecular weight (an invariant quantity); and R is the universal gas constant. In a triphasic material, a constitutive relation is needed for $\hat{\kappa}^\alpha$; in general, $\hat{\kappa}^\alpha$ may be a function of the solid matrix strain and the solute concentration. In FEBio, the dependence of the effective solubility on the solid matrix strain is currently constrained to a dependence on $J=\det\mathbf{F}$.

The solid matrix of a multiphasic material may be charged and its charge density is given by c^F. This charge density may be either negative or positive. The charge density varies with the deformation, increasing when the pore volume decreases. Based on the balance of mass for the solid,

$$c^F = \frac{1-\varphi_r^s}{J-\varphi_r^s} c_r^F,$$

where φ_r^s is the solid volume fraction and c_r^F is the fixed charge density in the reference configuration.

In the multiphasic theory it is assumed that electroneutrality is satisfied at all times. In other words, the net charge of the mixture is always zero (neutral). This electroneutrality condition is represented by a constraint equation on the ion concentrations,

$$c^F + \sum_\alpha z^\alpha c^\alpha = 0,$$

where z^α is the charge number of solute α $z^- = -1$. Since the concentrations of the cation and anion inside the triphasic material are not the same, an electrical potential difference is produced between the interstitial and external environments. The electric potential in the triphasic mixture is denoted by ψ and its effect combines with the chemical potential of each solute to produce the electrochemical potential $\tilde{\mu}^\alpha$, where

$$\tilde{\mu}^\alpha = \mu_0^\alpha(\theta) + \frac{R\theta}{M^\alpha}\ln\frac{c^\alpha}{\hat{\kappa}^\alpha} + \frac{z^\alpha}{M^\alpha}F_c\psi.$$

In this expression, F_c represents Faraday's constant. It is also possible to rearrange this expression as

$$\tilde{\mu}^\alpha = \mu_0^\alpha(\theta) + \frac{R\theta}{M^\alpha}\ln\left[\exp\left(z^\alpha\frac{F_c\psi}{R\theta}\right)\frac{c^\alpha}{\hat{\kappa}^\alpha}\right].$$

In a multiphasic material, the interstitial fluid pressure p is influenced by both mechanical and chemical environments. In other words, this pressure includes both mechanical and osmotic contributions, the latter arising from the presence of the solutes. The solvent mechano-chemical potential $\tilde{\mu}^w$ is given by

$$\tilde{\mu}^w = \mu_0^w(\theta) + \frac{1}{\rho_T^w}\left(p - R\theta\Phi\sum_\alpha c^\alpha\right),$$

where μ_0^w is the solvent chemical potential at some reference temperature θ; ρ_T^w is the true density of the solvent (an invariant property for an intrinsically incompressible fluid); and Φ is the osmotic coefficient which represents the extent by which the solute concentrations influence the solvent chemical potential. In a multiphasic material, a constitutive relation is needed for Φ; in general, Φ may be a function of the solid matrix strain and the solute concentrations. In FEBio, the dependence of the osmotic coefficient on the solid matrix strain is currently constrained to a dependence on $J = \det\mathbf{F}$.

The solute mechano-electrochemical potential is nearly equal to its electrochemical potential because the solute volume fraction is assumed to be negligible. The solvent mechano-electrochemical potential is the same as its mechano-chemical potential, since the solvent is neutral in a multiphasic mixture. In general, momentum and energy balances evaluated across a boundary surface in a multiphasic mixture require that the mechano-electrochemical potentials of

solvent and solutes be continuous across that surface. These continuity requirements are enforced automatically in FEBio by defining the effective fluid pressure $\tilde{p}$ and solute concentration $\tilde{c}^\alpha$ as

$$\begin{aligned} \tilde{p} &= p - R\theta\Phi\sum\nolimits_\alpha c^\alpha \\ \tilde{c}^\alpha &= c^\alpha \big/ \tilde{\kappa}^\alpha \end{aligned},$$

where

$$\tilde{\kappa}^\alpha = \hat{\kappa}^\alpha \exp\left(-z^\alpha \frac{F_c\psi}{R\theta}\right)$$

is the partition coefficient for solute α. The partition coefficient incorporates the combined effects of solubility and long-range electrostatic interactions to determine the ratio of interstitial to external concentration for that solute. Therefore, the effective concentration represents a measure of the *activity* of the solute, as understood in chemistry. The effective fluid pressure represents that part of the pressure which is over and above osmotic effects.

In FEBio, nodal variables consist of the solid matrix displacement $\mathbf{u}$, the effective fluid pressure $\tilde{p}$, and the effective solute concentrations $\tilde{c}^\alpha$. Essential boundary conditions must be imposed on these variables, and not on the actual pressure p or concentrations c^α. (In a biphasic material however, since $c^\alpha = 0$, the effective and actual fluid pressures are the same, $p = \tilde{p}$.)

The mixture stress in a triphasic material is given by $\boldsymbol{\sigma} = -p\mathbf{I} + \boldsymbol{\sigma}^e$, where $\boldsymbol{\sigma}^e$ is the stress arising from the solid matrix strain. The mixture traction on a surface with unit outward normal $\mathbf{n}$ is $\mathbf{t} = \boldsymbol{\sigma}\cdot\mathbf{n}$. This traction is continuous across the boundary surface. Therefore, the corresponding natural boundary condition for a multiphasic mixture is $\mathbf{t} = \mathbf{0}$. (In other words, if no boundary condition is imposed on the solid matrix displacement or mixture traction, the natural boundary condition is in effect.)

The natural boundary conditions for the solvent and solutes are similarly $\mathbf{w}\cdot\mathbf{n} = 0$ and $\mathbf{j}^\alpha\cdot\mathbf{n} = 0$, where $\mathbf{w}$ is the volumetric flux of solvent relative to the solid and $\mathbf{j}^\alpha$ is the molar flux of solute α relative to the solid. In general, $\mathbf{w}$ and $\mathbf{j}^\alpha$ are given by

$$\begin{aligned} \mathbf{w} &= -\tilde{\mathbf{k}}\cdot\left(\nabla\tilde{p} + R\theta\sum_\alpha \frac{\tilde{\kappa}^\alpha}{d_0^\alpha}\mathbf{d}^\alpha\cdot\nabla\tilde{c}^\alpha\right), \\ \mathbf{j}^\alpha &= \tilde{\kappa}^\alpha\,\mathbf{d}^\alpha\cdot\left(-\varphi^w\nabla\tilde{c}^\alpha + \frac{\tilde{c}^\alpha}{d_0^\alpha}\mathbf{w}\right), \end{aligned}$$

where

$$\tilde{\mathbf{k}} = \left[\mathbf{k}^{-1} + \frac{R\theta}{\varphi^w}\sum_\alpha \frac{\tilde{\kappa}^\alpha c^\alpha}{d_0^\alpha}\left(\mathbf{I} - \frac{\mathbf{d}^\alpha}{d_0^\alpha}\right)\right]^{-1}$$

is the effective hydraulic permeability of the interstitial fluid solution (solvent and solutes) through the porous solid matrix; $\mathbf{k}$ is the hydraulic permeability of the solvent through the porous solid matrix; $\mathbf{d}^\alpha$ is the diffusivity of solute α through the mixture (frictional interactions

with solvent and solid); and d_0^α is its free diffusivity (frictional interactions with solvent only). $\varphi^w \approx 1-\varphi^s$ is the solid matrix porosity in the current configuration. The above expressions for the solvent and solute flux do not account for external body forces.

Also see section 8.5 for additional guidelines for running multiphasic materials.

4.6.1. Guidelines for Multiphasic Analyses

4.6.1.1. Initial State of Swelling

Under traction-free conditions, a multiphasic material is usually in a state of swelling due to the osmotic pressure difference between the interstitial fluid and the external environment. This osmotic pressure arises from the difference in interstitial versus external concentrations of cations and anions, caused by the charged solid matrix and the requirement to satisfy electroneutrality. An osmotic pressure difference arising from such electrostatic interactions is known as the *Donnan osmotic pressure*. When the Donnan pressure is non-zero, traction-free conditions do not produce stress-free conditions for the solid matrix, since the matrix must expand until its stressed state resists the swelling pressure.

The Donnan pressure reduces to zero when the fixed charged density is zero, or when the external environment is infinitely hypertonic (having ion concentrations infinitely greater than the interstitial fixed charge density). Since these two conditions represent special cases, it is generally necessary to devise methods for achieving the desired initial state of swelling, in an analysis where loads or displacements need to be prescribed over and above this swollen state. Swelling occurs as a result of the influx of solvent into the porous solid matrix. This influx is a time-dependent process that could require extensive analysis time. Therefore, it is computationally efficacious to achieve the initial state of swelling by using a multi-step analysis (Chapter 6) where the first step is a steady-state analysis (Section 3.4.1). In this steady-state step, the fixed charge density may be ramped up from zero to the desired value using a load curve for that property or, alternatively, the external environmental conditions may be reduced from a very high hypertonic state down to the desired level. In the second step, prescribed displacement boundary conditions (when needed) may be specified to be of type *relative* (Section 3.9.1), so that they become superposed over and above the initial swelling state.

Example:

```
<Step>
      <Module type="multiphasic"/>
      <Control>
            <analysis type="steady-state"/>
            ...
      </Control>
</Step>
<Step>
      <Module type="multiphasic"/>
      <Control>
            ...
      </Control>
      <Boundary>
            <prescribe type="relative">
                  <node id="22" bc="z" lc="4">1</node>
                  ...
            </prescribe>
      </Boundary>
</Step>
```

4.6.1.2. Prescribed Boundary Conditions

In most analyses, it may be assumed that the ambient fluid pressure and electric potential in the external environment are zero, thus $p_* = 0$ and $\psi_* = 0$, where the subscripted asterisk is used to denote environmental conditions. Since the external environment does not include a solid matrix, the fixed charge density there is zero. $c_*^+ = c_*^- \equiv c_*$ It follows that the effective fluid pressure in the external environment is $\tilde{p}_* = -R\theta\Phi_*\sum_\alpha c_*^\alpha$ and the effective concentrations are $\tilde{c}_*^\alpha = c_*^\alpha \big/ \hat{\kappa}_*^\alpha$ $\tilde{c}_*^- = c_* \big/ \hat{\kappa}_*^-$. Therefore, in multiphasic analyses, whenever the external environment contains solutes c_*, the user must remember to prescribe non-zero boundary conditions for the effective solute concentrations *and* the effective fluid pressure.

Letting $p_* = 0$ also implies that prescribed mixture normal tractions (Section 3.10.2.3.) represent only the traction above ambient conditions. Note that users are not obligated to assume that $p_* = 0$. However, if a non-zero value is assumed for the ambient pressure, then users must remember to incorporate this non-zero value whenever prescribing mixture normal tractions. Similarly, users are not required to assume that $\psi_* = 0$; when a non-zero value is assumed for the electric potential of the external environment, the prescribed boundary conditions for the effective concentrations should be evaluated using the corresponding partition coefficient, $\tilde{c}_*^\alpha = c_*^\alpha \big/ \tilde{\kappa}_*^\alpha$ $\tilde{c}_*^- = c_* \big/ \tilde{\kappa}_*^-$.

4.6.1.3. Prescribed Initial Conditions

When a multiphasic material is initially exposed to a given external environment with effective pressure $\tilde{p}_*$ and effective concentrations $\tilde{c}_*^\alpha$ $\alpha = +,-$, the initial conditions inside the material should be set to $\tilde{p} = \tilde{p}_*$ and $\tilde{c}^\alpha = \tilde{c}_*^\alpha$ in order to expedite the evaluation of the initial state of swelling. The values of $\tilde{p}_*$ and $\tilde{c}_*^\alpha$ should be evaluated as described in Section 8.5.2

4.6.1.4. Prescribed Effective Solute Flux

The finite element formulation for multiphasic materials in FEBio requires that the natural boundary condition for solute α be prescribed as $\tilde{j}_n^\alpha \equiv j_n^\alpha + \sum_\beta z^\beta j_n^\beta$, where $\tilde{j}_n^\alpha$ is the effective solute flux. For a mixture containing only neutral solutes ($z^\beta = 0, \forall \beta$), it follows that $\tilde{j}_n^\alpha = j_n^\alpha$.

4.6.1.5. Prescribed Electric Current Density

The electric current density in a mixture is a linear superposition of the ion fluxes,

$$\mathbf{I}_e = F_c \sum_\alpha z^\alpha \mathbf{j}^\alpha .$$

Since only the normal component $j_n^\alpha = \mathbf{j}^\alpha \cdot \mathbf{n}$ of ion fluxes may be prescribed at a boundary, it follows that only the normal component $I_n = \mathbf{I}_e \cdot \mathbf{n}$ of the current density may be prescribed. To

prescribe I_n, it is necessary to know the nature of the ion species in the mixture, and how the current is being applied. For example, if the ions in the triphasic mixture consist of Na^+ and Cl^-, and if the current is being applied using silver/silver chloride (Ag/AgCl) electrodes, a chemical reaction occurs at the anode where Ag combines with Cl^- to produce AgCl, donating an electron to the electrode to transport the current. At the cathode, the reverse process takes place. Therefore, in this system, there is only flux of Cl^- and no flux of Na^+ ($j_n^+ = 0$) at the electrode-mixture interface, so that the prescribed boundary condition should be $j_n^- = -I_n / F_c$. Since $z^+ = +1$ and $z^- = -1$ in a triphasic mixture, the corresponding effective fluxes are given by $\tilde{j}_n^+ = 2j_n^+ - j_n^- = I_n / F_c$ and $\tilde{j}_n^- = j_n^+ = 0$.

4.6.1.6. Electrical Grounding

If a multiphasic mixture containing ions is completely surrounded by ion-impermeant boundaries it is necessary to ground the mixture to prevent unconstrained fluctuations of the electric potential. Grounding is performed by prescribing the effective concentration of one or more ions, at a location inside the mixture or on one of its boundaries corresponding to the location of the grounding electrode. The choice of ion(s) to constrain may be guided by the type of grounding electrode and the reference electrolyte bath in which it is inserted.

4.6.2. General Specification of Multiphasic Materials

The material type for a multiphasic material is "*multiphasic*". Constitutive relations must be provided for the solid matrix, the mixture fixed charge density, the hydraulic permeability $\mathbf{k}$, the osmotic coefficient Φ, and the properties of each solute: the solute diffusivity in the mixture $\mathbf{d}^\alpha$, the solute free diffusivity d_0^α, and the solute effective solubility $\hat{\kappa}^\alpha$. Therefore, the following parameters must be defined:

`<solid>`	specification of the solid matrix	
`<phi0>`	solid volume fraction φ_r^s in the reference configuration	[ ]
`<fixed_charge_density>`	fixed charge density c_r^F in the reference configuration	[**n**/**L**^{3}]
`<permeability>`	specification of the hydraulic permeability $\mathbf{k}$	
`<solvent_supply>`	specification of the solvent supply $\hat{\varphi}^w$	
`<osmotic_coefficient>`	specification of the osmotic coefficient Φ	
`<solute>`	specification of the solute properties	
`<solid_bound>`	specification of solid-bound molecule	

The `<solid>` tag encloses a description of the solid matrix constitutive relation and associated material properties, as may be selected from the list provided in Section 4.1.3. The solid volume fraction in the reference configuration, `<phi0>`, must be greater than 0 (no solid) and less than 1 (only solid). The volume fraction of fluid (also known as the porosity) in the reference configuration is given by $1-\varphi_0$. The `<fixed_charge_density>` may be negative, positive, or zero. The `<permeability>` and `<osmotic_coefficient>` tags enclose descriptions of the permeability and osmotic coefficient constitutive relations and their associated material properties, as may be selected from the list presented in Sections 4.4.2 and 4.5.5.

The optional `<solute>` tag provides a description of each solute in the multiphasic mixture. Multiple solutes may be defined. Each tag includes the required *sol* attribute, which should reference a solute *id* from the `<Solutes>` description in the `<Globals>` section (Section 3.5.2). The following parameters must be defined in this description:

`<diffusivity>`	specification of the solute diffusivities $\mathbf{d}^\alpha$ and d_0^α
`<solubility>`	specification of the solute effective solubility $\hat{\kappa}^\alpha$

The `<diffusivity>` and `<solubility>` tags enclose descriptions of materials that may be selected from the lists presented in Sections 4.5.3 and 4.5.4, respectively. Each solute tag must include a "sol" attribute

The optional `<solid_bound>` tag specifies which solid-bound molecule should be included in the multiphasic mixture. Multiple solid-bound molecules may be specified. Each tag should include the required *sbm* attribute, which references an *id* from the `<SolidBoundMolecules>`

description in the <Globals> section (Section 3.5.2). The following parameter must be defined in this description:

<rho0>	initial value of the referential apparent density of the solid-bound molecule ρ_r^σ .	[**M**/**L**^{3}]
<rhomin>	optional minimum allowable value of ρ_r^σ (zero by default)	[**M**/**L**^{3}]
<rhomax>	optional maximum allowable value of ρ_r^σ (none by default)	[**M**/**L**^{3}]

If a chemical reaction involves this solid-bound molecule its referential apparent density may evolve over time. The user may place lower and upper bounds on the allowable range of an evolving ρ_r^σ .

Example:

```
<material id="2" name="Media" type="multiphasic">
      <phi0>0.2</phi0>
      <fixed_charge_density>-40</fixed_charge_density>
      <solid name="Solid Matrix" type="Holmes-Mow">
            <density>1</density>
            <E>0.28</E>
            <v>0</v>
            <beta>0</beta>
      </solid>
      <permeability name="Permeability" type="perm-Holmes-Mow">
            <perm>1e-3</perm>
            <M>0</M>
            <alpha>0</alpha>
      </permeability>
      <osmotic_coefficient name="Ideal" type="osm-coef-const">
            <osmcoef>1.0</osmcoef>
      </osmotic_coefficient>
      <solute sol="1">
            <diffusivity name="Diffusivity" type="diff-const-iso">
                  <free_diff>1e-3</free_diff>
                  <diff>1e-3</diff>
            </diffusivity>
            <solubility name="Solubility" type="solub-const">
                  <solub>1.0</solub>
            </solubility>
      </solute>
      <solute sol="2">
            <diffusivity name="Diffusivity" type="diff-const-iso">
                  <free_diff>1e-3</free_diff>
                  <diff>1e-3</diff>
            </diffusivity>
            <solubility name="Solubility" type="solub-const">
                  <solub>1.0</solub>
            </solubility>
      </solute>
```

```
</material>
```

When a multiphasic material is employed in an analysis, it is also necessary to specify the values of the universal gas constant R [**F**·**L**/**n**·**T**], absolute temperature θ [**T**], and Faraday's constant F_c [**Q**/**n**] in the `<Globals>` section, using a self-consistent set of units.

Example:

```
<Globals>
      <Constants>
            <R>8.314e-6</R>
            <T>298</T>
            <Fc>96500e-9</Fc>
      </Constants>
      <Solutes>
            <solute id="1" name="Na">
                  <charge_number>1</charge_number>
            </solute>
            <solute id="2" name="Cl">
                  <charge_number>-1</charge_number>
            </solute>
      </Solutes>
</Globals>
```

Example:

Self-consistent units for a triphasic analysis	
Primary Units	
time	s
length	mm
force	N
mole	nmol
charge	C
temperature	K
Derived Units	
stress	N/mm^2, MPa
permeability	mm^4/N·s, mm^2/MPa ·s
diffusivity	mm^2/s
concentration	nmol/mm^3, mM
charge density	nEq/mm^3, mEq/L
voltage	mV
current density	A/mm^2
current	A

It is also possible to create models with multiphasic materials that use different solutes in different regions. In that case, introduce additional solute entries in the `<Solutes>` section and refer to those solute ids in the multiphasic material descriptions. Generally, two adjoining multiphasic regions may share the same solute (e.g., Na in both regions), in which case that

solute may transport freely across the interface separating these regions; or they may share no solute, in which case the interface is impermeable to all solutes.

4.6.3. Solvent Supply Materials

Solvent supply materials may be used to simulate an external source of solvent, such as supply from microvasculature that is not modeled explicitly. The solvent supply term, $\hat{\varphi}^w$, appears in the mass balance relation for the mixture,

$$\operatorname{div}\left(\mathbf{v}^s + \mathbf{w}\right) = \hat{\varphi}^w \ .$$

$\hat{\varphi}^w$ has units of reciprocal time [$\mathbf{t}^{-1}$]; it represents the rate at which the volume fraction of solvent changes with time.

4.6.3.1. Starling Equation

The material type for Starling's equation for fluid supply is *"Starling"*. The following material parameters need to be defined:

<kp>	hydraulic filtration coefficient, k_p	[**L**^{2}/**F**·**t**]
<pv>	effective fluid pressure in external source, $\tilde{p}_v$	[**P**]
<qc sol="n">	osmotic filtration coefficient, q_c^α	[**L**^{3}/**n**·**t**]
<cv sol="n">	effective solute concentration in external source, $\tilde{c}_v^\alpha$	[**n**/**L**^{3}]

The fluid supply is given by Starling's equation,

$$\hat{\varphi}^w = k_p\left(\tilde{p}_v - \tilde{p}\right) + \sum_\alpha q_c^\alpha\left(\tilde{c}_v^\alpha - \tilde{c}^\alpha\right),$$

where $\tilde{p}$ is the effective fluid pressure in the multiphasic material.

Example:
This example defines a solvent supply material of the Starling type for a multiphasic mixture containing one solute.

```
<solvent_supply type="Starling">
      <kp>0.001</kp>
      <pv>0.1</pv>
      <qc sol="1">1e-5</qc>
      <cv sol="1">150</cv>
</solvent_supply>
```

4.7. Chemical Reactions

4.7.1. Guidelines for Chemical Reaction Analyses

Chemical reactions may be modeled within a multiphasic mixture. The reaction may involve solutes ($\alpha = \iota$) and solid-bound molecules ($\alpha = \sigma$) that move with the solid matrix ($\mathbf{v}^\sigma = \mathbf{v}^s, \forall \sigma$). Consider a general chemical reaction,

$$\sum_\alpha \nu_R^\alpha \mathcal{E}^\alpha \rightarrow \sum_\alpha \nu_P^\alpha \mathcal{E}^\alpha, \tag{a}$$

where $\mathcal{E}^\alpha$ is the chemical species representing constituent α in the mixture; ν_R^α and ν_P^α represent stoichiometric coefficients of the reactants and products, respectively. To maintain consistency with classical chemical kinetics, the analysis of chemical reactions employs molar concentrations c^α and molar supplies $\hat{c}^\alpha$ on a solution-volume basis for all reactants and products, whether they are solutes or solid-bound molecular species.

Since the molar supply of reactants and products is constrained by stoichiometry, it follows that all molar supplies $\hat{c}^\alpha$ in a specific chemical reaction may be related to a *molar production rate* $\hat{\zeta}$ according to

$$\hat{c}^\alpha = \nu^\alpha \hat{\zeta}, \tag{b}$$

where ν^α represents the net stoichiometric coefficient for $\mathcal{E}^\alpha$,

$$\nu^\alpha = \nu_P^\alpha - \nu_R^\alpha. \tag{c}$$

Thus, formulating constitutive relations for $\hat{c}^\alpha$ is equivalent to providing a single relation for $\hat{\zeta}\left(\theta, \mathbf{F}, c^\alpha\right)$. When the chemical reaction is reversible,

$$\sum_\alpha \nu_R^\alpha \mathcal{E}^\alpha \rightleftharpoons \sum_\alpha \nu_P^\alpha \mathcal{E}^\alpha, \tag{d}$$

the relations of (b)-(c) still apply but the constitutive relation for $\hat{\zeta}$ would be different.

Example:
Consider the dissociation of $CaCl_2$ into ions Ca^{2+} and Cl^-,

$$CaCl_2 \rightleftharpoons Ca^{2+} + 2Cl^-.$$

The mixture contains three constituents. The stoichiometric coefficients of the reactants are $\nu_R^{CaCl_2} = 1$, $\nu_R^{Ca^{2+}} = 0$, $\nu_R^{Cl^-} = 0$, and those of the products are $\nu_P^{CaCl_2} = 0$, $\nu_P^{Ca^{2+}} = 1$, $\nu_P^{Cl^-} = 2$.

The reaction production rate $\hat{\zeta}$ enters into the governing equations of multiphasic mixtures via the mass balance relation for each solute,

$$\frac{1}{J}\frac{D^s\left[J\left(1-\varphi^s\right)c^\iota\right]}{Dt} + \operatorname{div}\mathbf{j}^\iota = \left(1-\varphi^s\right)\nu^\iota\hat{\zeta}, \tag{f}$$

the mass balance for the mixture,

$$\operatorname{div}\left(\mathbf{v}^s + \mathbf{w}\right) = \left(1-\varphi^s\right)\hat{\zeta}\overline{\mathcal{V}}, \tag{g}$$

where $\overline{\mathcal{V}} = \sum_{\alpha} \nu^{\alpha} \mathcal{V}^{\alpha}$ and $\mathcal{V}^{\alpha} = M^{\alpha} / \rho_T^{\alpha}$ is the molar volume of α, and the mass balance for solid-bound constituents,

$$D^s \rho_r^{\sigma} / Dt = \hat{\rho}_r^{\sigma}, \tag{h}$$

where ρ_r^{σ} is the referential apparent mass density (mass of σ per mixture volume in the reference configuration), and $\hat{\rho}_r^{\sigma}$ is the referential apparent mass supply of solid constituent σ, related to molar concentrations and supplies via

$$c^{\sigma} = \frac{\rho_r^{\sigma}}{\left(J - \varphi_r^s\right) M^{\sigma}}, \quad \hat{c}^{\sigma} = \frac{\hat{\rho}_r^{\sigma}}{\left(J - \varphi_r^s\right) M^{\sigma}}. \tag{i}$$

Internally, the content of solid-bound species is stored in ρ_r^{σ} and (i) is used to evaluate c^{σ} when needed for the calculation of $\hat{\zeta}$. If a solid-bound molecule is involved in a chemical reaction, equation (h) is integrated to produce an updated value of ρ_r^{σ}, using $\hat{\rho}_r^{\sigma} = \left(J - \varphi_r^s\right) M^{\sigma} \nu^{\sigma} \hat{\zeta}$ based on (b) and (i).

Evolving solid content due to chemical reactions implies that the referential solid volume fraction φ_r^s may not remain constant. This value is updated at every time point using

$$\varphi_r^s = \varphi_0^s + \sum_{\sigma} \frac{\rho_r^{\sigma}}{\rho_T^{\sigma}} \tag{j}$$

where φ_0^s is the solid volume fraction specified by the multiphasic material parameter *phi0* (Section 4.6.2). Thus, φ_0^s may be used to account for the solid volume fraction not contributed explicitly by solid-bound molecules. Based on kinematics, the solid volume fraction in the current configuration is given by $\varphi^s = \varphi_r^s / J$. Therefore, since $0 \le \varphi^s \le 1$ by definition, it follows that $0 \le \varphi_r^s \le J$, implying that the referential solid volume fraction may evolve to values greater than unity when growth leads to swelling of the multiphasic mixture.

Similarly, if solid-bound molecules are charged and their content evolves over time, the referential fixed charge density may also evolve with chemical reactions according to

$$c_r^F = c_0^F + \frac{1}{1 - \varphi_r^s} \sum_{\sigma} \frac{z^{\sigma} \rho_r^{\sigma}}{M^{\sigma}}, \tag{k}$$

where c_0^F is the referential fixed charge density specified by the multiphasic material parameter *fixed_charge_density* (Section 4.6.2). Thus, c_0^F may be used to account for the fixed charge density not contributed explicitly by solid-bound molecules.

A chemical reaction is properly balanced when

$$\sum_{\alpha} \nu^{\alpha} M^{\alpha} = 0, \tag{l}$$

where M^{α} is the molar mass of α. This constraint implies that the net gain in mass of products must be the same as the net loss in mass of reactants. However, this constraint is not verified in the code, allowing users to model chemical reactions with implicit constituents (constituents that

are neither explicitly modeled as solutes nor as solid-bound molecules, for which ν^α and M^α are not given). For example, a chemical reaction where cells consume glucose to form a protein from amino-acids building blocks may have the form

$$\text{cells} + \text{glucose} + \text{amino acids} \rightarrow \text{cells} + \text{protein} + \text{waste products} \ .$$

The user may opt to model only the glucose reactant and the protein product explicitly, while the presence of all other species in this reaction is implicit. In these types of analyses the user must beware of potential inconsistencies in the evolving mass of reactants and products since only some of those constituents are modeled explicitly. In particular, the evolution of φ_r^s as given in (j) can only account for the explicitly modeled solid-bound molecules. Furthermore, when some reactants and products are implicit, the value of the reaction molar volume $\overline{\mathcal{V}}$ calculated in the code becomes inaccurate and may produce unexpected results in the evaluation of the mixture mass balance relation in (g). Therefore, the user is given the option to override the value of $\overline{\mathcal{V}}$ calculated in the code. In particular, if the precise molar volumes of all the species in a reaction are not known, assuming that $\overline{\mathcal{V}} \approx 0$ is a reasonable choice equivalent to assuming that all the constituents have approximately the same density ρ_T^α, as may be deduced from (l).

Since the electroneutrality condition is enforced in multiphasic mixtures in FEBio, it follows that chemical reactions must not violate this condition. Enforcing electroneutrality in a chemical reaction is equivalent to satisfying

$$\sum_\alpha z^\alpha \nu^\alpha = 0 \ . \qquad \text{(m)}$$

This constraint is checked within the code and an error is generated when it is violated.

A constitutive relation must be provided for the molar production rate $\hat{\zeta}\left(\theta, \mathbf{F}, c^\alpha\right)$ of each chemical reaction.

4.7.2. General Specification for Chemical Reactions

The material type for a chemical reaction is "*reaction*". The `<reaction>` tag must appear inside the definition of a multiphasic mixture and the reaction is defined only for that mixture. Multiple chemical reactions may be defined in a given mixture. Each `<reaction>` tag must include a *type* attribute that identifies the constitutive relation for $\hat{\zeta}$.

The stoichiometric coefficients ν_R^α of the reactants and ν_P^α for the products must be specified in every reaction. Optionally, the net reaction molar volume $\overline{\mathcal{V}}$ may be specified explicitly to override the internal value calculated in the code. Therefore, the following parameters need to be defined in a chemical reaction:

`<vR>`	reactant stoichiometric coefficient ν_R^α	[ ]
`<vP>`	product stoichiometric coefficient ν_P^α	[ ]
`<Vbar>`	optional override value for $\overline{\mathcal{V}}$	[$\mathbf{L}^3/\mathbf{n}$]

Each `<vR>` and `<vP>` tag must include either the *sol* attribute, which should reference a solute *id* from the `<Solutes>` description in the `<Globals>` section (Section 3.5.2), or the *sbm* attribute, which should reference a solid-bound molecule *id* from the `<SolidBoundMolecules>` description in the `<Globals>` section. Only solutes and solid-bound molecules that have been included in the parent multiphasic mixture may be specified as reactants or products of a chemical reaction.

Additional parameters may be needed in the definition of a chemical reaction, depending on the specific form of the constitutive relation for the production rate.

4.7.3. Chemical Reaction Materials

4.7.3.1. Law of Mass Action for Forward Reactions

The material type for the Law of Mass Action for a forward reaction is *mass-action-forward*. The following parameters must be defined:

`<forward_rate>`	specific forward reaction rate k

For this type of reaction the constitutive relation for the molar production rate is given by

$$\hat{\zeta} = k\prod_{\alpha}\left(c^{\alpha}\right)^{\nu_R^{\alpha}} \quad .$$

The constitutive form of the specific forward reaction rate must be selected from the list of materials given in Section 4.7.4. The units of $\hat{\zeta}$ are [**n**/**L**^{3}·**t**] and those of c^{α} are [**n**/**L**^{3}].

Example:
Consider the forward reaction that produces solid copper sulfide from solid copper and solid sulfur,

$$\mathrm{Cu} + \mathrm{S} \rightarrow \mathrm{CuS}$$

All three species are modeled explicitly in the mixture as solid-bound molecules, with id's 1 (Cu), 2 (for S) and 3 (for CuS). The chemical reaction material is given by:

```
<reaction name="copper sulfide production" type="mass-action-forward">
      <vR sbm="1">1</vR>
      <vR sbm="2">1</vR>
      <vP sbm="3">1</vP>
      <forward_rate type="constant">
            <k>1e-3</k>
      </forward_rate>
</reaction>
```

4.7.3.2. Law of Mass Action for Reversible Reactions

The material type for the Law of Mass Action for a reversible reaction is *mass-action-reversible*. The following parameters must be defined:

<forward_rate>	specific forward reaction rate k_F
<reverse_rate>	specific reverse reaction rate k_R

For this type of reaction the constitutive relation for the molar production rate is given by

$$\hat{\zeta} = \hat{\zeta}_F - \hat{\zeta}_R \ ,$$

where

$$\hat{\zeta}_F = k_F \prod_\alpha \left(c^\alpha\right)^{\nu_R^\alpha}$$

$$\hat{\zeta}_R = k_R \prod_\alpha \left(c^\alpha\right)^{\nu_P^\alpha} \ .$$

The constitutive form of the specific forward and reverse reaction rates must be selected from the list of materials given in Section 4.7.4. The units of $\hat{\zeta}_F$ and $\hat{\zeta}_R$ are [**n**/**L**^{3}·**t**] and those of c^α are [**n**/**L**^{3}].

Example:
Consider the reversible dissociation of CaCl salt into Ca^{2+} and Cl^- in water,

$$CaCl_2 \rightleftharpoons Ca^{2+} + 2Cl^-$$

All three species are modeled explicitly in the mixture as solutes, with id's 1 (for $CaCl_2$), 2 (for Ca^{2+}) and 3 (for Cl^-). The chemical reaction material is given by:

```
<reaction name="CaCl2 dissociation" type="mass-action-reversible">
      <vR sol="1">1</vR>
      <vP sol="2">1</vP>
      <vP sol="3">2</vP>
      <forward_rate type="constant">
            <k>1.0</k>
      </forward_rate>
      <reverse_rate type="constant">
            <k>0.1</k>
      </reverse_rate>
</reaction>
```

4.7.3.3. Michaelis-Menten Reaction

The material type for the Michaelis-Menten reaction is *Michaelis-Menten*. The following parameters must be defined:

<forward_rate>	maximum rate at saturating substrate concentration $V_{\max}$	[**n**/**L**^{3}·**t**]
<Km>	substrate concentration when reaction rate is half of $V_{\max}$	[**n**/**L**^{3}]
<c0>	optional minimum substrate concentration to trigger reaction	[**n**/**L**^{3}]

The Michaelis-Menten reaction may be used to model enzyme kinetics where the enzyme $\mathcal{E}^e$ triggers the conversion of the substrate $\mathcal{E}^s$ into the product $\mathcal{E}^p$. The product molar supply is given by

$$\hat{c}^p = \begin{cases} \dfrac{V_{\max} c^s}{K_m + c^s} & c^s \geq c_0 \\ 0 & c^s < 0 \end{cases},$$

where c^s is the substrate concentration. The default value of c_0 is 0. This relation may be derived, with some simplifying assumptions, by applying the law of mass action to the combination of two reactions,

$$\mathcal{E}^e + \mathcal{E}^s \rightleftharpoons \mathcal{E}^{es} \rightarrow \mathcal{E}^e + \mathcal{E}^p$$

Since the enzyme is not modeled explicitly in the Michaelis-Menten approximation to these two reactions, this reaction model is effectively equivalent to

$$\mathcal{E}^s \rightarrow \mathcal{E}^p .$$

Therefore, this reaction accepts only one reactant tag <vR> and one product tag <vP>. For consistency with the formulation of this reaction, the stoichiometric coefficients should be set to $\nu_R^s = \nu_P^p = 1$, so that $\hat{c}^p = \hat{\zeta}$.

The constitutive form of the specific forward reaction rate $V_{\max}$ must be selected from the list of materials given in Section 4.7.4.

Example:

```
<reaction name="enzyme kinetics" type="Michaelis-Menten">
      <Vbar>0</Vbar>
      <vR sol="1">1</vR>
      <vP sol="2">1</vP>
      <forward_rate type="constant">
            <k>1.0</k>
      </forward_rate>
      <Km>5.0</Km>
</reaction>
```

4.7.4. Specific Reaction Rate Materials

The following sections define specific materials that can be used to define the reaction rate of a chemical reaction.

4.7.4.1. Constant Reaction Rate

The material type for a constant specific reaction rate is *constant reaction rate*. The following parameter must be defined:

<k>	constant specific reaction rate *k*	[*units vary*]

Example:

```
<reaction name="enzyme kinetics" type="Michaelis-Menten">
      <Vbar>0</Vbar>
      <vR sol="1">1</vR>
      <vP sol="2">1</vP>
      <forward_rate type="constant reaction rate">
            <k>1.0</k>
      </forward_rate>
      <Km>5.0</Km>
</reaction>
```

4.7.4.2. Huiskes Reaction Rate

The material type for the Huiskes reaction rate is *Huiskes reaction rate*. The following parameters must be defined:

<B>	reaction rate per specific strain energy B	[*units vary*]
<psi0>	specific strain energy at homeostasis ψ_0	[**F**·**L**/**M**]

The Huiskes specific reaction rate has the form

$$k = \frac{B}{J - \varphi_r^s}\left(\frac{\Psi_r}{\rho_r^s} - \psi_0\right),$$

where Ψ_r is the strain energy density of the solid (strain energy in current configuration per mixture volume in the reference configuration) and $\rho_r^s = \sum_\sigma \rho_r^\sigma$ is the referential apparent solid density (mass of solid in current configuration per mixture volume in reference configuration). The ratio Ψ_r / ρ_r^s is the specific strain energy (strain energy per mass of solid in the current configuration). The Huiskes specific reaction rate may assume positive and negative values; it reduces to zero at homeostasis, when $\Psi_r / \rho_r^s = \psi_0$.

Example:
To reproduce the interstitial solid remodeling theory proposed by Weinans et al. [35], consider the forward reaction

$$\text{cells} + \text{solutes} \rightarrow \text{cells} + \text{solid},$$

where cells convert solutes (e.g., nutrients) into synthesized solid when the specific strain energy exceeds the homeostatic value. If the cells and solutes are implicit in this reaction (e.g., if it is assumed that their concentrations change negligibly), the production rate of the solid may be given by $\hat{\zeta} = k$ using the law of mass action for a forward rection, where k has the form given above.

```
<reaction name="solid remodeling" type="mass-action-forward">
      <vP sbm="1">1</vP>
      <forward_rate type="Huiskes reaction rate">
            <B>1.0</B>
            <psi0>0.01</psi0>
      </forward_rate>
      <Km>5.0</Km>
</reaction>
```

4.8. Rigid Body

A rigid body can be defined using the rigid material model. The material type for a rigid body is "*rigid body*". The following parameters are defined:

`<density>`	Density of rigid body	[**M**/**L**^{3}]
`<center_of_mass>`	Position of the center of mass	[**L**]
`<E>`	Young's modulus	[**P**]
`<v>`	Poisson's ratio	[ ]

If the *center_of_mass* parameter is omitted, FEBio will calculate the center of mass automatically. In this case, a density *must* be specified. If the *center_of_mass* is defined the *density* parameter may be omitted. Omitting both will generate an error message.

The Young's modulus *E* and Poisson ratio *v* currently have no effect on the results. The only place where they are used is in the calculation of a material stiffness for some auto-penalty contact formulation. If you are using contact it is advised to enter sensible values. Otherwise these parameters may be omitted.

The degrees of freedom of a rigid body are initially unconstrained[*]. This implies that a rigid body is free to move in all three directions and rotate about any of the coordinate axes. To constrain a rigid body degree of freedom you need the *Constraints* section. See section 3.12.1 for more information.

Example:

```
<material id="1" type="rigid body">
      <density>1.0</density>
      <center_of_mass>0,0,0</center_of_mass>
</material>
```

[*] This is different from previous versions of FEBio where rigid bodies were initially fully constrained.

Chapter 5 Restart Input file

As of version 1.1.6, FEBio will output a binary dump restart file. The user can either run the restart on the binary dump or create a restart input file.

The restart feature in FEBio allows the user to continue a previously terminated analysis. In addition, it allows the user to modify some of the parameters during the restart. To activate the restart feature, define the *restart* element in the Control section:

```
<Control>
      <restart [file="<dump file>"]>1</restart>
      ...
</Control>
```

This describes the format of the restart input file. This file is used to redefine some parameters when restarting a previously terminated run. The structure is very similar to the FEBio input file and also uses XML formatting.

Since the file uses XML, the first line must be the XML header:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

The next line contains the root element of the restart file, and has to be:

```
<febio_restart version="1.0">
```

The restart file is composed of the following sections. These sections are sub-elements of the *febio_restart* root element.

- *Archive*: define the binary dump file used for restarting.
- *Control*: redefine some control parameters
- *LoadData*: redefine some loadcurves.

All sections are optional except for the Archive section, and need only be defined when redefining parameters. In the following paragraphs we describe the different sections in more detail.

5.1. The Archive Section

The Archive section must be the first sub-element of the *febio_restart* root element. This section defines the name of the binary dump file:

```
<Archive>archive.dmp</Archive>
```

5.2. The Control Section

The following control parameters can be redefined:

Parameter	Description
dtol	convergence tolerance for displacements
etol	convergence tolerance for energy
rtol	convergence tolerance for residual
lstol	line search tolerance
max_refs	maximum number of stiffness reformations
max_ups	maximum number of BFGS updates
restart	restart file generation flag
plot_level	defines the frequency of the plot file generation

5.3. The LoadData Section

In the LoadData section the user can redefine some or all of the load curves. The syntax is identical to the LoadData section of the FEBio input file:

```
<LoadData>
	<loadcurve id="n">
		<loadpoint>0, 0</loadpoint>
		...
		<loadpoint>1, 0.54</loadpoint>
	</loadcurve>
</LoadData>
```

In this case the loadcurve *id* is the loadcurve number of the loadcurve that the user wishes to redefine.

5.4. Example

The following example defines a restart input file. No parameters are redefined. Only the mandatory *Archive* element is defined. In this case the analysis will simply continue where it left off:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<febio_restart version="1.0">
	<Archive>out.dmp</Archive>
</febio_restart>
```

Chapter 6 Multi-step Analysis

As of version 1.1.6, multi-step problems can be solved with FEBio. A multi-step analysis is defined using multiple steps, where in each step the user can redefine control parameters and boundary conditions. This is useful, for instance, for defining time-dependant boundary conditions or for switching between different analysis types during the simulation.

6.1. The Step Section

The multi-step analysis feature introduces a new section to the input file. Each step requires its own *step* section, preferably at the bottom of the input file. In this step section, the user can redefine the control section and the boundary section. The following format is suggested when defining a multi-step analysis:

```
<febio_spec version="2.0">
	<Control>
		<!-- global control parameters -->
	</Control>
	<Material>
		<!-- materials go here -->
	</Material>
	<Geometry>
		<!-- geometry goes here -->
	</Geometry>
	<Boundary>
		<!-- global boundary conditions -->
	</Boundary>
	<LoadData>
		<!-- load curve data goes here -->
	</LoadData>
	<Step>
		<Control>
			<!-- local control settings -->
		</Control>
		<Boundary>
			<!-- local boundary conditions -->
		</Boundary>
	</Step>
</febio_spec>
```

The first part of the file looks similar to a normal input file, except that in the control section only global control parameters should be defined (e.g. the title). Also, the boundary section should only contain global boundary conditions. The differences between local and global parameters are explained below.

After the LoadData section, the user can define as many Step sections as needed. In each Step section, the user can now define the (local) control parameters and boundary conditions.

In a multi-step analysis it is important to understand the difference between local and global settings. The global settings are those settings that remain unchanged during the entire simulation.

6.1.1. Control Settings

All control settings are considered local, except the title. Therefore the title should be the only control setting that is defined in the global control section. All other control parameters, including time step parameters, linear solver parameters, convergence parameters, and so forth, should be defined in the Control section of each step. For example:

```
<febio_spec version="1.0">
	<Control>
		<title>This is the title</title>
	</Control>
	...
	<Step>
		<Control>
			<!-- place control parameters here -->
		</Control>
	</Step>
</febio_spec>
```

6.1.2. Boundary Conditions

In a multi-step analysis boundary conditions can be applied that are only active during the step. This applies to the *Boundary*, *Loads*, *Contact* and *Constraints* section of the FEBio input file. These sections can thus be placed inside the *Step* section to define a boundary condition that is only active during the step. If one of these sections is defined before the first *Step* section, the corresponding boundary conditions remain enforced during all the steps.

6.1.3. Relative Boundary Conditions

Some boundary conditions can be defined as relative boundary conditions. This means that the corresponding conditions will be applied to the final configuration of the previous step and not to the original reference configuration. For example, if a prescribed displacement is defined as relative the displacement will be taken with respect of the positions of the final configuration. This makes it possible to switch between load and displacement controlled boundary conditions in multi-step analyses.

6.2. An Example

The following example illustrates the use of the multi-step feature of FEBio. This problem defines two steps. In the first step, a single element is stretched using a prescribed boundary condition. In the second step, the boundary condition is removed and the analysis type is

switched from quasi-static to a dynamic analysis. Note the presence of the global fixed boundary constraints, which will remain enforced during both steps:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<febio_spec version="2.0">
	<Control>
		<title>Multi-step example</title>
	</Control>
	<Material>
		<material id="1" name="Material 1" type="neo-Hookean">
			<density>1.0</density>
			<E>1</E>
			<v>0.45</v>
		</material>
	</Material>
	<Geometry>
		<Nodes>
			<node id="1">-2.0,-0.5, 0.0</node>
			<node id="2">-2.0,-0.5, 1.0</node>
			<node id="3">-2.0, 0.5, 0.0</node>
			<node id="4">-2.0, 0.5, 1.0</node>
			<node id="5"> 2.0,-0.5, 0.0</node>
			<node id="6"> 2.0,-0.5, 1.0</node>
			<node id="7"> 2.0, 0.5, 0.0</node>
			<node id="8"> 2.0, 0.5, 1.0</node>
		</Nodes>
		<Elements>
			<hex8 id="1" mat="1">1,5,7,3,2,6,8,4</hex8>
		</Elements>
	</Geometry>
	<Boundary>
		<fix>
			<node id="1" bc="xyz"></node>
			<node id="2" bc="xy"></node>
			<node id="3" bc="xz"></node>
			<node id="4" bc="x"></node>
			<node id="5" bc="yz"></node>
			<node id="6" bc="y"></node>
			<node id="7" bc="z"></node>
		</fix>
	</Boundary>
	<LoadData>
		<loadcurve id="1">
			<loadpoint>0,0</loadpoint>
			<loadpoint>1,0.1</loadpoint>
		</loadcurve>
	</LoadData>
	<Step>
		<Control>
			<time_steps>10</time_steps>
			<step_size>0.1</step_size>
```

```
            </Control>
            <Boundary>
                  <prescribe>
                        <node id="5" bc="x" lc="1">1</node>
                        <node id="6" bc="x" lc="1">1</node>
                        <node id="7" bc="x" lc="1">1</node>
                        <node id="8" bc="x" lc="1">1</node>
                  </prescribe>
            </Boundary>
      </Step>
      <Step>
            <Control>
                  <time_steps>50</time_steps>
                  <step_size>0.5</step_size>
                  <analysis type="dynamic"></analysis>
            </Control>
      </Step>
</febio_spec>
```

Chapter 7 Parameter Optimization

This chapter describes FEBio's parameter optimization module. This module tries to estimate material parameters by solving an inverse finite element problem, where the "solution" of the problem is known, and the problem's material parameters are sought. In this case, the solution will often be an experimentally determined reaction force curve, and the unknown parameters are the material parameters that will recreate the reaction force curve by solving a forward FE problem. The user will need to input this reaction force curve as well as an initial guess for the material parameters. This is done using a separate input file which is described next.

7.1. Optimization Input File

Like all FEBio input files, the parameter optimization input file is an xml-formatted text file. The following sections are defined:

- *Model*: optional section defining the FE problem input file.
- *Options*: optional section defining the optimization control parameters.
- *Function*: defines the parameter that generates the function values of the solution that are to be fitted.
- *Parameters*: defines the material parameters that are to be determined.
- *Constraints*: defines linear constraints for the material parameters.
- *LoadData*: defines the experimental reaction force using load curves.

In the following paragraphs each section will be explained in detail.

7.1.1. Model Section

This section is included for backwards compatibility, but the model is now specified on the command line as described in section 7.2. The model section defines the file name for the FE input problem. This input file defines the problem that will be solved by FEBio repeatedly. This xml-tag only takes a single value, namely the name of a valid FEBio input file. For example,

```
<Model>ex1.feb</Model>
```

7.1.2. Options Section

This section defines the control parameters for the optimization. The options that can be defined will depend on the chosen optimization method, which is set with an attribute of the Options section. Currently the two implemented optimization methods are the "Levenberg-Marquardt" method (type="levmar") and the "Constrained Levenberg-Marquardt" method (type="constrained levmar"). Documentation for the Constrained Levenberg-Marquardt method can be found at http://users.ics.forth.gr/~lourakis/levmar/.

Note that this section is optional. When omitted, default values for the control parameters are chosen. The following parameters can be defined.

Parameter	Description	Default
`type attribute`	Optimization method	levmar
`obj_tol`	Convergence tolerance for objective function. (1)	0.001
`f_diff_scale`	Forward difference scale factor (2)	0.001
`log_level`	Sets the amount of output that is generated on screen and in the logfile (3)	`LOG_NEVER`
`tau`	Step size scale factor (4)	`1.0e-3`

For example,

```
<Options type="constrained levmar">
     <obj_tol>0.001</obj_tol>
     <f_diff_scale>0.001</f_diff_scale>
</Options>
```

Comments:

1. The objective function that is to be minimized is a function of the reaction force values:

$$f(\mathbf{a}) = \sum_{i=1}^{n} \left[y_i - y(x_i;\mathbf{a}) \right]^2 .$$

   Here, $y(x;\mathbf{a})$ is the function that describes the model, **a** is a vector with the (unknown) material parameters and the (x_i, y_i) are the experimentally (or otherwise) obtained data that approximates the ideal model.

2. The optimization method currently implemented requires the calculation of the gradient of the model function *y* with respect to the model parameters **a**. Since this gradient is not known, FEBio will approximate it using forward differences. For example, the *k*-th component of the gradient is approximated as follows.

$$\frac{\partial y}{\partial a_k} \approx \frac{1}{\delta a_k}\left[y(a_1,\cdots,a_k+\delta a_k,\cdots,a_m) - y(a_1,\cdots,a_k,\cdots,a_m) \right]$$

   The value for δa_k is determined from the following formula.

$$\delta a_k = \varepsilon(1+a_k)$$

   where, ε is the forward difference scale factor which can be set by the user with the *fdiff_scale* option.

3. The *log_level* allows the user to control exactly how much output from the time iterations of each optimization iteration is written to the screen and logfile. The following values are allowed:

Value	Description
`LOG_DEFAULT`	Use default settings (may be omitted)

LOG_NEVER	Don't generate any output
LOG_FILE_ONLY	Save to the logfile only
LOG_SCREEN_ONLY	Only print to the screen
LOG_FILE_AND_SCREEN	Save to the logfile and print to the screen

4. The step size scale factor *tau* can be set as an input only for the Constrained Levenberg-Marquardt method.

7.1.3. Function Section

This section defines the parameter that will generate the function values that are to be fitted to the experimental data. Currently, only one function parameter can be defined and it is defined using the *fnc* tag. For example,

```
<Function>
      <fnc lc="1">rigid.Fx</fnc>
</Function>
```

This example defines the *x*-component of a rigid body material named *rigid* as the model's function parameter. The *lc* attribute defines the loadcurve number that describes the model's experimental data.

Note that currently only rigid body materials can be used to define the function parameter, and only using the following data components.

Component	**Description**
Fx	x-component of the rigid body reaction force
Fy	y-component of the rigid body reaction force
Fz	z-component of the rigid body reaction force
Mx	x-component of the rigid body reaction moment
My	y-component of the rigid body reaction moment
Mz	z-component of the rigid body reaction moment

7.1.4. Parameters Section

This section defines the material parameters that are to be determined. Each parameter is defined using the *param* element:

```
<Parameters>
      <param name="[name]">[guess], [min], [max], [scale]</param>
</Parameters>
```

The *name* attribute gives the name of the parameter that is to be determined. The format of the name depends on the nesting of the material description in the input file. FEBio supports names in the following form:

For most materials, the following format is used to reference a parameter.

```
material_name.parameter
```

For example, assume a material, named *Material1*, of type neo-Hookean is defined. Then we can reference the Young's modulus (E) as follows.

```
name="Material1.E"
```

In FEBio many materials define nested parameters, meaning the material properties have parameters of their own. Examples are the *biphasic*, *biphasic-solute* and *multiphasic* materials. In general, nested parameters are referenced using the material name, the property and the parameter.

```
material_name.property.parameter
```

For instance, assume a biphasic material, named *Material1* is defined. To reference the *perm* parameter, use the following syntax.

```
name="Material1.permeability.perm"
```

If this material has a *neo-Hookean* solid matrix, then the Young's modulus is referenced as follows.

```
name="Material1.solid.E"
```

For multiply nested properties, the syntax extends naturally. For instance, assume a biphasic material with a visco-elastic material, having a neo-Hookean material as a solid matrix. In this case, the Young's modulus is referenced as follows.

```
name="Material1.solid.elastic.E"
```

The only exception to this is when the *solid* property references a solid mixture. In that case, the components of the mixture are referenced by name. For example, assume again a biphasic material, where the *solid* is a solid mixture of which the first component is a neo-Hookean material, named *solid1*. Now, the Young's modulus is referenced as follows.

```
name="Material1.solid.solid1.E"
```

Furthermore, if a parameter is part of an array, its name should be specified in the form `parameter[index]`, where the index is one-based.

Each parameter takes four values: `[guess]` is the initial guess for this parameter, `[min]` and `[max]` are the minimum and maximum values respectively for this parameter, and `[scale]` is a representative scale (magnitude) for this parameter. This value is used to normalize the

optimization parameter and improve convergence. If not specified by the user, it defaults to the initial guess.

For example, for a neo-Hookean solid, named *mat1*

```
<Parameters>
     <param name="mat1.E">1.0, 0.0, 5.0, 1.0</param>
     <param name="mat1.v">0.1, 0.0, 0.5, 1.0</param>
</Parameters>
```

This example defines two material parameters. The first component of the name (here *mat1*) is the name of the material as defined in the model input file (defined in the *Model* section). The second component (here *E* and *v*) are the names of the material parameters.

For a solid mixture that includes an ellipsoidal fiber distribution,

```
<Parameters>
     <param name="mat1.mat2.ksi[1]">0.2, 0, 6, 1.0</param>
</Parameters>
```

In this example, the name of the solid mixture is *mat1*; the name of the ellipsoidal fiber distribution solid is *mat2*; and *ksi[1]* is the first material parameter in an array of three values for this parameter. In order for this naming convention to work correctly, the user must ensure that a unique name is given for each solid in a solid mixture.

For a biphasic material, the parameter name for material properties of the `<solid>` and `<permeability>` materials can be referenced as in this example,

```
<Parameters>
     <param name="mat1.phi0">0.2, 0.01, 0.99, 0.5</param>
     <param name="mat1.solid.E">0.1, 0.01, 0.5, 1.0</param>
     <param name="mat1.permeability.perm">2e-3,5e-4,1e-2,1e-3</param>
</Parameters>
```

Note that *phi0* is a material parameter that belongs to the biphasic material description, whereas *E* belongs to the solid description and *perm* belongs to the permeability description.

7.1.5. Constraints Section

The Constrained Levenberg-Marquardt method allows linear constraints on the material parameters. If **a** is the material parameter vector, then the linear constraint is of the form:
$c_1a_1 + c_2a_2 + ... + c_na_n + b = 0$.
The coefficients $c_1, c_2, ..., c_n, b$ are the inputs of the constraint tag. For example, if the linear constraint is $2a_1 - a_2 + 3 = 0$, then the Constraints section would be:

```
<Constraints>
     <constraint>2, -1, 3</constraint>
```

```
</Constraints>
```

7.1.6. Load Data Section

This section serves the same purpose as in the regular FEBio input file: it defines all the load curves that are used in the parameter optimization. Currently, only one load curve needs to be defined, namely the experimental data to which the model will be fitted. The format is identical to that of the usual FEBio input file (see section 3.14).

Each load curve is defined through an array of value pairs (x_i, y_i), where the *x* component refers to the (simulation) time and the *y* component to is the function value.

7.2. Running a Parameter Optimization

As explained above, a parameter optimization problem is described using two input files. First, a standard FEBio input file that defines the geometry, materials, boundary conditions, etc. The second input file describes the parameter optimization data, such as the objective and which material parameters are to be optimized. The format of the second file is described above. A parameter optimization can only be initiated from the command line. For example:

```
>febio –i model.feb –s optim.feb
```

The output of a parameter optimization analysis is a log file that contains the screen output of the FEBio run as well as the optimized parameter values.

7.3. An Example Input File

Below follows a complete example of an optimization input file.

```
<?xml version="1.0"?>
<febio_optimize>
	<Model>ex1.feb</Model>
	<Options>
		<obj_tol>0.001</obj_tol>
		<f_diff_scale>0.001</f_diff_scale>
	</Options>
	<Function>
		<fnc lc="1">rigid.Fx</fnc>
	</Function>
	<Parameters>
		<param name="mat1.E">1, 0, 5</param>
		<param name="mat1.v">-0.5, 0, 0.5</param>
	</Parameters>
	<LoadData>
		<loadcurve id="1">
			<point>0.0, 0</point>
```

```
                <point>0.5, 1</point>
                <point>1.0, 2</point>
            </loadcurve>
        </LoadData>
</febio_optimize>
```

Comments:

1. Notice that the xml root element is *febio_optimize* for the optimization input file.
2. Here, the FEBio input file that contains the actual FE model data is *ex1.feb*. This file is a standard FEBio input file that defines all geometry, materials, boundary conditions and more.
3. The *Options* section is included here, but can be omitted. If omitted default values will be used for all control parameters.
4. The function parameter and material parameters are defined through “material name”.“parameter name” pairs. The name of the materials and their parameters are defined in the FEBio input file (in this example, *ex1.feb*).

Chapter 8 Troubleshooting

Running a nonlinear finite element analysis can be a very challenging task. The large deformations and complex constitutive models can make it very difficult to obtain a converged solution. There are many causes for nonconvergence, ranging from element inversions, material instability, failure to enforce contact constraints and many more. Sometimes it is possible that FEBio gives you a converged solution, but the solution is meaningless or at least not what was expected.

Fortunately, many of these issues can be prevented or solved with little effort. This chapter discusses some strategies to prevent common problems and to troubleshoot a problematic run. It offers some sanity checks before you run your model which can save you a lot of frustration down the road. And when things do go bad, we hope that the strategies suggested here may prove helpful.

However, keep in mind, that the finite element method cannot solve all problems. It is a very powerful numerical method, but with limitations. Understanding these limitations and how they affect your modeling work is crucial in becoming a good analysist.

8.1. Before You Run Your Model

In this section we'll discuss what a well-defined finite element model is and some things you may need to check before you run your model in FEBio.

A well-defined finite element model contains a finite element mesh, a valid material and properly defined boundary and contact conditions in order to define a unique solution to the problem under study. We will look at each of these requirements in more detail in the following sections.

8.1.1. The Finite Element Mesh

A finite element mesh is required to solve a problem with FEBio. The mesh defines a discretization of the problem domain in nodes and connected elements. FEBio only supports certain elements and thus the mesh must be composed of elements from this set. See section 3.7.2 for a discussion of the supported elements. In addition, FEBio assumes a specific ordering of the nodes of an element. FEBio cannot discover if the nodes are in the correct order, but if they are not, FEBio will most likely have trouble converging or throw negative jacobians. If FEBio discovers negative jacobians before the first time step, it is likely that the nodes of the elements are not defined in the proper order.

For shell elements, the initial thickness of an element is also important. When elements are too thick (the thickness is of the same order as the element size), FEBio may complain about negative jacobians. This may be particularly a problem in areas of high curvature. The only solution around this issue might be to remesh the problematic area.

8.1.2. Materials

A material in FEBio defines the constitutive response of the domain to which the material is assigned. See Chapter 4 for a detailed list of all the available materials in FEBio. It is important to understand that the module defines which materials you can use. For example, the biphasic material cannot be used in the *solid* module. Although some cases of invalid material use are caught, in many situations the resulting behavior is undefined. The following table shows a list of some of FEBio's special materials and the modules in which they can be used.

Material	Solid	Biphasic	Solute	Multiphasic	Heat
biphasic		**YES**	**YES**	**YES**	
biphasic-solute			**YES**	**YES**	
triphasic			**YES**	**YES**	
multiphasic				**YES**	
isotropic Fourier					**YES**

In addition to using the proper materials for a given module, it is also important to understand that most material parameters have a limited range in which they define valid constitutive behavior. For example, in a neo-Hookean material the Young's modulus must be positive and the Poisson's ratio must larger than -1 and less than 0.5. FEBio has most of these limits implemented in the code and will throw an error when a parameter value is defined that falls outside the valid range.

Since material parameters can be defined as functions of time through a loadcurve, FEBio will check all parameters at the beginning of each time step. When a parameter has become invalid the run will be aborted.

Some constitutive models are only valid for a limited amount of deformation. All materials in FEBio are designed for large deformation (in the sense that they are objective under arbitrary deformation), but that does not imply unlimited deformation. When the deformation becomes too large, the material response may become unphysical and although FEBio gives an answer, the result may be meaningless (see section 8.6 for a discussion on interpreting the result). Some materials also become unstable at extremely large deformations. A classical example is the Mooney-Rivlin material. For a certain range of values of the material parameters, and under certain loading conditions, the stress-strain response can have a zero slope at large deformations. In that case, FEBio will most likely not be able to converge since the slope of the stress-strain response is used to progress towards the solution.

8.1.3. Boundary Conditions

Boundary conditions are necessary to uniquely define the solution of the problem under study. Improperly defined boundary conditions can lead to underconstrained or overconstrained problems.

If the problem is underconstrained the solution is not uniquely defined and FEBio will not be able to find a solution. The most common example of an undercontrained problem is one where

the rigid body modes are not constrained. A rigid body mode is a mode of deformation that does alter the stress in the body. Affine translation and rotation of the entire mesh are two ways to introduce a rigid body mode. In a (quasi-)static finite element analysis all rigid body modes must be properly constrained in order to find a unique solution. This can often very easily be achieved by constraining and edge or face of the model from moving at all.

If the problem is over-constrained, FEBio will usually find an answer but it is probably not the solution that was sought. The most common cause of an over-constrained model is one that has conflicting boundary conditions. For example, a force is applied to a node that is fixed.

8.2. Debugging a Model

When a model is not running as expected, the first thing to try is rerun the problem using FEBio's debug mode. Debug mode can be activated by adding a –g on the command line. For example,

```
>febio –i file.feb -g
```

When in debug mode, FEBio will do additional checks during the solution (e.g. check for NAN's, check for zeroes on the diagonal of the stiffness matrix, etc.) and will store all non-converged states to the plot file. Inspecting these non-converged states can often be useful to identify the cause of the problem. (E.g. when the model seems to disappear at some point might indicate a rigid body mode.)

Since storing all non-converged states may make the plot-file too large for post-processing, debug mode can also be turned on during the analysis. To do this, start the problem normally (without the debug flag –g). Then, right before the problem starts to fail, interrupt the run (using ctrl+c) and wait for the FEBio prompt to appear. Once it appears, enter *debug* [ENTER], and then *cont* [ENTER] to continue the run in debug mode.

8.3. Common Issues

In this section we take a look at some of the most common problems you may run into when solving a finite element problem with FEBio. We'll discuss possible causes of and solutions to these issues.

8.3.1. Inverted elements

When an element inverts, the element becomes so distorted that in certain areas of the element the Jacobian becomes zero or negative. In order to obtain a physically realistic solution, the Jacobian of the deformation must be positive at all points of the domain. Thus, when a negative Jacobian is found in an element, FEBio cannot continue. Fortunatly, FEBio's automatic time-stepping algorithm can usually circumvent this problem by cutting the time step back and retrying the step using a smaller time step. In addition, FEBio will recalculate the global stiffness matrix. The combination of a reduced time step and a refactored stiffness matrix will often be sufficient to overcome the negative Jacobian. However, when it is not, you may have run into a

more serious problem. These are some of the common causes that may require additional user intervention.

8.3.1.1. Material instability

At large deformations, some materials can become unstable. This is usually caused by a softening of the material at large deformations. When the material softens too much the global stiffness matrix can become ill-conditioned and FEBio will not be able to find the correct solution. Unfortunately, there is no easy solution around this issue and you may need to consider using a different constitutive model.

8.3.1.2. Time step too large

Altough the auto-time stepper will automatically adjust the time step in case of trouble, it is designed to work within user defined limits and sometimes they are not set properly to solve the problem. The most common issue is that the minimum time step is set too large. Cutting back the minimum in addition to the initial time step can often help in preventing negative Jacobians.

8.3.1.3. Elements too distorted

When elemens get too distorted they might not be able to deform any further without inverting the element. In this case, it may be necessary to adjust the mesh in the area where the elements are inverting.

8.3.1.4. Shells are too thick

A shell is an element where it is assumed that one dimension is much smaller compared to the other two directions. When the mesh is really fine, this assumption may no longer be valid and it could happen that the shell becomes too thick. When a shell gets too thick it can create negative Jacobians in the out-of-plane integration points, especially in areas of high curvature. To overcome this, you may need to remesh the affected area, adjust the shell thickness or even replace the shells with solid elements.

8.3.1.5. Rigid body modes

A rigid body mode is a mode of deformation that does not alter the stress in the body. Uniform translation or rotation of the entire model are two possible rigid body modes. In the presence of a rigid body mode, the solution is not uniquely defined and FEBio will most likely throw negative Jacobians. The solution is to identify the rigid body mode and to eliminate it by applying additional constraints to the model.

8.3.2. Failure to converge

FEBio uses an iterative method to solve the nonlinear finite element equations. This means that for each time step the solution for that step is obtained by a succession of iterations, where each

iteration brings the model (hopefully) closer to the solution for that step. An iterative method requires a convergence criterium to decide when to stop iterating and FEBio uses no less than three criteria. (In biphasic and multiphasic problems additional convergence criteria are used for the pressure and concentration degrees of freedom.) The convergence norm for each of these criteria is defined by the user in the control section of the input file.

Sometimes it happens that FEBio cannot converge on a time step. We'll take a look at some of the most common causes in the next following sections. Also take a look at possible causes of element inversions above. Sometimes, the issues that cause an element inversion may also manifest themselves in convergence problems before the element actually inverts.

8.3.2.1. No loads applied

Ironically, when no loads are applied, FEBio will often struggle to find a solution. The reason (and solution!) is simple. Although no loads are applied, FEBio still performs many calculations and due to round-off errors, the result of these calculations may render the convergence norms not exactly zero. FEBio then tries to apply the convergence criteria to these really small norms and again due to round-off error will not be able to satisfy the convergence criteria.

FEBio can actually detect this situation. It looks at the residual norm and when this is smaller than a user-defined limit it assumes that no loads are present and moves on to the next time step. However, this limit is actually problem dependent and it can happen that for your particular problem the limit is set too small. In that case, the solution is to increase the limit. This can be done by setting the *min_residual* parameter in the *Control* section of the input file. For example,

```
<min_residual>1e-15</min_residual>
```

If the residual norm now drops below this value, FEBio will consider the time step converged and move on to the next time step.

8.3.2.2. Convergence Tolerance Too Tight

In some cases, the default convergence tolerances are too tight and FEBio will not be able to converge. In that case, adjusting the affected convergence norms is a possible solution. This is done by editing the corresponding norms in the Control section of the input file. However, this should only be done when no other cause can be identified for the convergence problems.

8.3.2.3. Forcing convergence

It is possible to force a time step to converge. This can be done by interrupting the run (using ctrl+c) and then enter the *conv* command at the FEBio prompt. This will force the time step to converge and FEBio will continue with the next time step. This is not a recommended practice as the solution may become unstable and in fact it is unlikely that FEBio will be able to converge any of the following time steps. However, it can be useful in some scenarios. For example, if there is no load applied in the initial time step and FEBio has trouble getting past this initial step (see section 8.3.2.1.) or when you want to store the current state to the plot file without having

to rerun the problem in debug mode (although the *plot* command or *debug* command might be better alternatives).

8.3.2.4. Problems due to Contact

When contact interfaces are defined, convergence problems can often be traced back to problems with the contact interfaces. See section 8.4 for more details on how to properly use contact interfaces in FEBio.

8.4. Guidelines for Contact Problems

FEBio offers many contact interfaces which allow the user to model complex boundary interactions. However, this capability comes at a price and the use of contact interfaces may create several problems in the solution process. It is hoped that the guidelines presented in this section may prevent many of the most common problems with contact.

Most of the contact algorithms implemented in FEBio offer two contact enforcement methods: a penalty method and an augmented Lagrangian method. Since the strategies for these two methods are slightly different, we will look at them separately. These algorithms are discussed in much more detail in the FEBio Theory Manual.

8.4.1. The penalty method

The penalty method enforces contact by "penalizing" (in an energy sense) any deviation from the contact constraint. In other words, when the two contacting surfaces penetrate, a force is generated proportional to the distance of penetration, which has the effect that the surfaces will now repell each other. The strength of this repelling force is controlled by the user through the *penalty factor*.

The obvious downside of this method is that some penetration is necessary to generate the contact force. The larger the penalty factor, the smaller this penetration needs to be and thus the better the contact constraint is enforced. However, choosing the penalty factor too large may cause the problem to become unstable since the contact force (and corresponding stiffness) will dominate the other forces in the model. Choosing a good penalty factor can thus be often difficult and may require several attempts before getting it "right". To help with choosing a penalty factor, FEBio offers the *auto_penalty* option for many of its contact interfaces which will estimate a good penalty factor based on element size and initial material stiffness. However, even then it may still take a few tries before getting the penalty factor good enough.

8.4.2. Augmented Lagrangian Method

In theory, in the presence of constraints, the corresponding Lagrange multipliers must be calculated which, in the case of contact, correspond to the contact forces that enforce the contact constraint exactly. Unfortunately, the exact evaluation of these Lagrange multiplies is numerically very challenging and therefore in FEBio it was decided to evaluate these Lagrange multipliers in an iterative method, named the *Augmented Lagrangian* method. Using this

method, FEBio will solve a time step several times (these iterations are termed *augmentations*), where each time the approximate Lagrange mutlipliers are updated (or *augmented*).

The obvious drawback of this method is that now each time step has to be solved several times. However, the advantage of avoiding the numerical problems of obtaining the exact Lagrange multipliers and the fact that in most contact problems very little extra work is needed to solve these augmentations, makes this method very attractive. In addition, it often gives better enforcement of the contact constraint compared to the penalty method.

So why not just use the augmented Lagrangian method? Well, often the penalty method will give good results and since the penalty method is much faster, it is often the preferred choice of many analysists.

In many cases, users are only interested in the final time step. For these users it may be of interest that it is possible to use the augmented Lagrange method only in the final time step. This can be done by defining a loadcurve for the *laugon* contact parameter. Then, define the corresponding loadcurve as zero everywhere except for the final time step. FEBio will now only use the augmented Lagrangian method in the final time step (and the penalty method all other steps).

8.4.3. Initial Separation

FEBio often struggles with problems that have an initial separation. This is especially so when the problem is force-driven, meaning that the deformation of the model is driven by a force (opposed to displacement-driven problems). In general, when the contact interface is necessary to define a well-constrained problem it is best to avoid initial separation if possible. Another way around this issue is to first use a displacement to bring the surfaces in contact and then replace the displacement with a force.

8.5. Guidelines for Multiphasic Analyses

8.5.1. Initial State of Swelling

Under traction-free conditions, a multiphasic material is usually in a state of swelling due to the osmotic pressure difference between the interstitial fluid and the external environment. This osmotic pressure arises from the difference in interstitial versus external concentrations of cations and anions, caused by the charged solid matrix and the requirement to satisfy electroneutrality. An osmotic pressure difference arising from such electrostatic interactions is known as the *Donnan osmotic pressure*. When the Donnan pressure is non-zero, traction-free conditions do not produce stress-free conditions for the solid matrix, since the matrix must expand until its stressed state resists the swelling pressure.

The Donnan pressure reduces to zero when the fixed charged density is zero, or when the external environment is infinitely hypertonic (having ion concentrations infinitely greater than the interstitial fixed charge density). Since these two conditions represent special cases, it is

generally necessary to devise methods for achieving the desired initial state of swelling, in an analysis where loads or displacements need to be prescribed over and above this swollen state. Swelling occurs as a result of the influx of solvent into the porous solid matrix. This influx is a time-dependent process that could require extensive analysis time. Therefore, it is computationally efficacious to achieve the initial state of swelling by using a multi-step analysis (Chapter 6) where the first step is a steady-state analysis (Section 3.4.1). In this steady-state step, the fixed charge density may be ramped up from zero to the desired value using a load curve for that property or, alternatively, the external environmental conditions may be reduced from a very high hypertonic state down to the desired level. In the second step, prescribed displacement boundary conditions (when needed) may be specified to be of type *relative* (Section 3.9.1), so that they become superposed over and above the initial swelling state.

Example:

```
<Step>
      <Module type="multiphasic"/>
      <Control>
            <analysis type="steady-state"/>
            ...
      </Control>
</Step>
<Step>
      <Module type="multiphasic"/>
      <Control>
            ...
      </Control>
      <Boundary>
            <prescribe type="relative">
                  <node id="22" bc="z" lc="4">1</node>
                  ...
            </prescribe>
      </Boundary>
</Step>
```

8.5.2. Prescribed Boundary Conditions

In most analyses, it may be assumed that the ambient fluid pressure and electric potential in the external environment are zero, thus $p_* = 0$ and $\psi_* = 0$, where the subscripted asterisk is used to denote environmental conditions. Since the external environment does not include a solid matrix, the fixed charge density there is zero. $c_*^+ = c_*^- \equiv c_*$ It follows that the effective fluid pressure in the external environment is $\tilde{p}_* = -R\theta\Phi_*\sum_\alpha c_*^\alpha$ and the effective concentrations are $\tilde{c}_*^\alpha = c_*^\alpha \big/ \hat{\kappa}_*^\alpha$ $\tilde{c}_*^- = c_* \big/ \hat{\kappa}_*^-$. Therefore, in multiphasic analyses, whenever the external environment contains solutes c_*, the user must remember to prescribe non-zero boundary conditions for the effective solute concentrations *and* the effective fluid pressure.

Letting $p_* = 0$ also implies that prescribed mixture normal tractions (Section 3.10.2.3.) represent only the traction above ambient conditions. Note that users are not obligated to assume that $p_* = 0$. However, if a non-zero value is assumed for the ambient pressure, then users must remember to incorporate this non-zero value whenever prescribing mixture normal tractions. Similarly, users are not required to assume that $\psi_* = 0$; when a non-zero value is assumed for the electric potential of the external environment, the prescribed boundary conditions for the effective concentrations should be evaluated using the corresponding partition coefficient, $\tilde{c}_*^\alpha = c_*^\alpha \big/ \tilde{\kappa}_*^\alpha$ $\tilde{c}_*^- = c_* \big/ \tilde{\kappa}_*^-$.

8.5.3. Prescribed Initial Conditions

When a multiphasic material is initially exposed to a given external environment with effective pressure $\tilde{p}_*$ and effective concentrations $\tilde{c}_*^\alpha$ $\alpha = +, -$, the initial conditions inside the material should be set to $\tilde{p} = \tilde{p}_*$ and $\tilde{c}^\alpha = \tilde{c}_*^\alpha$ in order to expedite the evaluation of the initial state of swelling. The values of $\tilde{p}_*$ and $\tilde{c}_*^\alpha$ should be evaluated as described in Section 8.5.2

8.5.4. Prescribed Effective Solute Flux

The finite element formulation for multiphasic materials in FEBio requires that the natural boundary condition for solute α be prescribed as $\tilde{j}_n^\alpha \equiv j_n^\alpha + \sum_\beta z^\beta j_n^\beta$, where $\tilde{j}_n^\alpha$ is the effective solute flux. For a mixture containing only neutral solutes ($z^\beta = 0, \forall \beta$), it follows that $\tilde{j}_n^\alpha = j_n^\alpha$.

8.5.5. Prescribed Electric Current Density

The electric current density in a mixture is a linear superposition of the ion fluxes,

$$\mathbf{I}_e = F_c \sum_\alpha z^\alpha \mathbf{j}^\alpha \ .$$

Since only the normal component $j_n^\alpha = \mathbf{j}^\alpha \cdot \mathbf{n}$ of ion fluxes may be prescribed at a boundary, it follows that only the normal component $I_n = \mathbf{I}_e \cdot \mathbf{n}$ of the current density may be prescribed. To prescribe I_n, it is necessary to know the nature of the ion species in the mixture, and how the current is being applied. For example, if the ions in the triphasic mixture consist of Na^+ and Cl^-, and if the current is being applied using silver/silver chloride (Ag/AgCl) electrodes, a chemical reaction occurs at the anode where Ag combines with Cl^- to produce AgCl, donating an electron to the electrode to transport the current. At the cathode, the reverse process takes place. Therefore, in this system, there is only flux of Cl^- and no flux of Na^+ ($j_n^+ = 0$) at the electrode-mixture interface, so that the prescribed boundary condition should be $j_n^- = -I_n \big/ F_c$. Since $z^+ = +1$ and $z^- = -1$ in a triphasic mixture, the corresponding effective fluxes are given by $\tilde{j}_n^+ = 2 j_n^+ - j_n^- = I_n / F_c$ and $\tilde{j}_n^- = j_n^+ = 0$.

8.5.6. Electrical Grounding

If a multiphasic mixture containing ions is completely surrounded by ion-impermeant boundaries it is necessary to ground the mixture to prevent unconstrained fluctuations of the electric potential. Grounding is performed by prescribing the effective concentration of one or more ions, at a location inside the mixture or on one of its boundaries corresponding to the location of the grounding electrode. The choice of ion(s) to constrain may be guided by the type of grounding electrode and the reference electrolyte bath in which it is inserted.

8.6. Understanding the Solution

Okay, FEBio found a solution. Great, but how do you know it is the right one? Well, it turns out this is in fact a harder question than it may seem. In any finite element model there are numerous assumptions and approximations and trying to discuss all of these is outside the scope of this section. Instead, we'll look at some of the most typical problems that FEBio users have run into.

8.6.1. Mesh convergence

The solution calculated by FEBio only applies to the mesh for which it was solved. However, the "exact" solution must be independent of the mesh and therefore a mesh convergence study is almost always necessary to make sure the FEBio solution is close to this exact solution. In practice this means that a model must be run several times with an increasingly finer mesh to find out at which mesh density the FEBio solution no longer changes.

8.6.2. Constraint enforcement

FEBio uses many iterative algorithms for enforcing constraints. For each of these contraints the user must verify that the solution indeed satisfies these constraints sufficiently. The two most common constraints used in FEBio are incompressibility and contact.

For uncoupled materials, incompressibility (for hexahedral elements) is handled in FEBio using a three-field formulation in addition to the enforcement of the incompressibility constraint. This constraint is usually enforced using a penalty formulation (although an augmented Lagrangian method is also available). To inspect whether the incompressibility constraint is satisfied sufficiently the user can look at the volume ratio which has to be close to one. If it deviates from one too much the user needs to increase the "bulk modulus" of the material.

For coupled materials, incompressibility is not treated explicitly and care must be taken when using coupled materials in a near-incompressible regime (e.g. setting the Poisson's ratio too close to 0.5 for a neo-Hookean material). In that case, the solution will most likely "lock" which manifests itself in displacements that are too small. Inspection of the volume ratio may not be sufficient to identify locking. However, a mesh convergence study will often help in identifying this problem.

When a contact constraint is not sufficiently enforced, the contacting surfaces will have penetrated. This problem is usually identified easily by looking at the deformed model in a post-processing software (e.g. PostView). Increasing the penalty factor and/or decreasing the augmented Lagrangian tolerance should solve this problem.

8.7. Limitations of FEBio

If you run into a problem that you can't seem to solve, maybe you ran into a limitation of FEBio. This section discusses some important limitations of the software and how they may affect the outcome of your analysis.

8.7.1. Geometrical instabilities

A geometrical instability is a point in the solution at which several paths for continuing the solution are possible. Typical examples are buckling of a column or inflation of a balloon. FEBio's Newton based solvers cannot handle buckling since usually at these points, the material's elasticity tangent is not uniquely defined. Although in some cases FEBio will be able to pass a buckling point, in most cases, FEBio will not be able to converge to a solution.

8.7.2. Material instabilities

A material instability is a point in the stress-strain response where the slope of the stress-strain curve effectively becomes zero. Since FEBio's Newton based solvers use this slope to advance the solution, a zero slope would mean an infinite step and FEBio will not be able to continue.

The slope (or more accurately the elasticity tangent tensor) does not have to be zero to cause problems. For materials that soften at large deformations, this slope may become so small that it renders the global stiffness matrix ill-conditioned. It is unlikely that FEBio will be able to continue and even if it finds a solution, it may not to useful.

8.7.3. Remeshing

FEBio is designed to solve problems that can undergo large deformations. However, when the deformations become too large it can happen that the finite element mesh cannot be distorted any further without inverting elements. In that case, FEBio will not be able to continue the solution and will most likely terminate with a negative jacobian error message.

One possible solution to this problem is to remesh the model at the point at which the deformation become too large, but currently FEBio does not have any remeshing capabilities. The only remedy at this point is to plan ahead and design your mesh in such a way that the elements will not invert in the range of deformation that you are interested in (e.g. place a butterfly mesh in sharp corners or make the mesh finer in areas of larger deformation).

8.7.4. Force-driven Problems

When the primary method of deforming the model is a force, the problem is said to be force-driven. (Opposed to a displacement-driven model where the deformation is caused by displacement boundary conditions.) Force-driven forces are inherently unstable and FEBio has limited capabilities of handling such problems[*]. The cause of this instability can easily be explained using a simple example. Imagine a box that is constrained in all directions except one. In the unconstrained direction, apply two equal but opposite forces on opposite faces. In order to prevent the box from flying away, the box must generate stresses which balance the applied forces exactly. Unfortunately, the numerical solution will only be approximate due to numerical round-off errors inherent in the calculation. Even the slightest deviation from balancing the forces exactly will create a net-force which in turn causes the model to fly off in the unconstrained direction (in which a rigid body mode now exists).

Usually, these problems are circumvented by adjusting the boundary conditions so that rigid body modes cannot exist. In the previous example this can be done by only modeling half of the box and enforcing a symmetry boundary condition. However, in some cases this is not possible. This is often the case in force-driven contact problems, where the contact interface is necessary to define a well-constrained model. When there is an initial separation (or even when the surfaces have no initial overlap) the initial contact force is zero, which is equivalent to having no contact enforcement, and thus the model is under-constrained. Therefore, for force-driven contact problems it is important to have some initial contact before starting the analysis.

8.7.5. Solutions obtained on Multi-processor Machines

Although technically not a limitation of FEBio it is important that users are aware that in some cases you may get a different convergence history or sometimes even a slightly different answer when you run the same model twice on a multi-processor machine. This is caused by the fact that on multi-processor machines the same calculations can be executed in a different order and due to the accumulation of numerical round-off errors may result in slightly different answers. The type of problems that are mostly affected by this are problems that are close too being ill-conditioned. Also, problems that have many time steps or require many iterations for each time step may be affected by this.

If you experience different answers for the same problem, try running the model on just one processor (if possible). See section 2.6 for more information on how to run FEBio on multi-processor machines.

8.8. Where to Get More Help

When you get here, you may be ready to pull all your hair out, but fret not, all is not lost. In fact, some people may have run into a similar issue and found a solution. The first place to find these people is on the FEBio user's forum (http://mrlforums.sci.utah.edu/forums/forum.php). This forum contains hundreds of posts by FEBio users and is monitored by the FEBio developers who

[*] As of FEBio 2.0, some features are available that may help with this issue, but they are still experimental.

will always be more than happy to help you with your problems. In addition, this forum can be used to report bugs or to request a new feature. It is the ideal portal to find help with your problems so don’t hesitate to use it!

Appendix A. Configuration File

As of version 1.2, FEBio requires a configuration file to run. The purpose of this file is to store platform-specific settings such as the default linear solver. See section 2.5 for more information on how to use this file. This section details the format of this file.

The configuration file uses an xml format. The root element must be *febio_config*. The required attribute *version* specifies the version number of the format. Currently this value must be set to "1.0". The following elements are defined.

Parameter	**Description**
`linear_solver`	Set the default linear solver for the platform. (1)
`import`	Load a plugin file (2)

Comments:

1. FEBio supports several linear solvers, such as Pardiso and Skyline. Not all solvers are available for all platforms. Only the Skyline solver is available for all platforms. As of version 1.2, Pardiso solver are available on most platforms. (Support for the SuperLU solver was removed as of version 2.0.)
2. As of FEBio 2.0 the user can create and use plugins designed for FEBio. These plugins extend the standard capabilities without the need to recompile the FEBio code. See Appendix B for more information on using plugins in FEBio.

Appendix B. FEBio Plugins

As of version 2.0 FEBio supports plugins. Plugins are dynamic libraries which extend the capabilities of FEBio at runtime without the need to recompile the entire source code. This offers the user a powerful mechanism for extending the default feature set of FEBio with little effort. In addition, the development of the plugin is independent of the FEBio source code, which implies that upgrading to a newer version of FEBio will not require a recompilation of the plugin[*]. The FEBio developer's documentation (available online at http://febio.org/febio/febio-documentation/) explains in detail how to create these plugins. In this section we outline how to use the plugins with FEBio. Using plugins, users can create custom materials, boundary conditions, loads, output variables, etc. Users can also define custom "tasks" with plugins. A task is simply a wrapper around FEBio. (FEBio actually defines several tasks by default. The standard task simply loads the input file, runs the model and creates the output files. Other tasks perform diagnostics). Algorithms that require repeated evaluations of FE models (e.g. parameter optimization) can easily be implemented as a plugin that defines such a custom task.

Using Plugins

A plugin is compiled into a dynamic library (dll) or shared object (so) and containes the compiled code of the library. In order to use this plugin, you must add the path to and name of the plugin file in the FEBio configuration file (see Appendix A for a discussion of the configuration file). For each plugin, an *import* item has to be added in this configuration file. For example,

```
<import>myplugin.dll</import>
```

You may need to add the full path to the plugin if FEBio has problems locating the plugin.

```
<import>C:\path\to\my\plugins\myplugin.dll</import>
```

When FEBio starts, it will first load all the plugins that are defined in the configuration file before it reads the input file. This way, sections in the input file that require the plugin's capabilities will already be available for use. If FEBio cannot find a plugin it will terminate with an error message.

[*] When upgrading to a newer version of FEBio, it may be necessary to recompile the plugin if the plugin interface was changed. As long as the plugin remains compatible with the new interface, no code changes are required.

References

[1] Maker, B. N., 1995, "NIKE3D: A nonlinear, implicit, three-dimensional finite element code for solid and structural mechanics," Lawrence Livermore Lab Tech Rept, UCRL-MA-105268.
[2] Gee, M. W., Dohrmann, C. R., Key, S. W., and Wall, W. A., 2009, "A uniform nodal strain tetrahedron with isochoric stabilization," Int. J. Numer. Meth. Engng(78), pp. 429-443.
[3] Laursen, T. A., and Maker, B. N., 1995, "Augmented Lagrangian quasi-newton solver for constrained nonlinear finite element applications," International Journal for Numerical Methods in Engineering, 38(21), pp. 3571-3590.
[4] Ateshian, G., Maas, S., and Weiss, J. A., 2009, "Finite Element Algorithm for Frictionless Contact of Porous Permeable Media under Finite Deformation and Sliding," J. Biomech. Engn.
[5] Simo, J. C., and Taylor, R. L., 1991, "Quasi-incompressible finite elasticity in principal stretches: Continuum basis and numerical algorithms," Computer Methods in Applied Mechanics and Engineering, 85, pp. 273-310.
[6] Arruda, E. M., and Boyce, M. C., 1993, "A Three-Dimensional Constitutive Model for the Large Stretch Behavior of Rubber Elastic Materials," J. Mech. Phys. Solids, 41(2), pp. 389-412.
[7] Lanir, Y., 1983, "Constitutive equations for fibrous connective tissues," J Biomech, 16(1), pp. 1-12.
[8] Ateshian, G. A., Rajan, V., Chahine, N. O., Canal, C. E., and Hung, C. T., 2009, "Modeling the matrix of articular cartilage using a continuous fiber angular distribution predicts many observed phenomena," J Biomech Eng, 131(6), p. 061003.
[9] Ateshian, G. A., 2007, "Anisotropy of fibrous tissues in relation to the distribution of tensed and buckled fibers," J Biomech Eng, 129(2), pp. 240-249.
[10] Fung, Y. C., 1993, Biomechanics : mechanical properties of living tissues, Springer-Verlag, New York.
[11] Fung, Y. C., Fronek, K., and Patitucci, P., 1979, "Pseudoelasticity of arteries and the choice of its mathematical expression," Am J Physiol, 237(5), pp. H620-631.
[12] Ateshian, G. A., and Costa, K. D., 2009, "A frame-invariant formulation of Fung elasticity," J Biomech, 42(6), pp. 781-785.
[13] Blemker, S., 2004, "3D Modeling of Complex Muscle Architecture and Geometry," Stanford University, Stanford.
[14] Criscione, J., Douglas, S., and Hunter, W., 2001, "Physically based strain invariant set for materials exhibiting transversely isotropic behavior," J. Mech. Phys. Solids, 49, pp. 871-897.
[15] Spencer, A. J. M., 1984, Continuum Theory of the Mechanics of Fibre-Reinforced Composites, Springer-Verlag, New York.
[16] Ateshian, G. A., Ellis, B. J., and Weiss, J. A., 2007, "Equivalence between short-time biphasic and incompressible elastic material response," J Biomech Eng, In press.
[17] Puso, M. A., and Weiss, J. A., 1998, "Finite element implementation of anisotropic quasi-linear viscoelasticity using a discrete spectrum approximation," J Biomech Eng, 120(1), pp. 62-70.
[18] Quapp, K. M., and Weiss, J. A., 1998, "Material characterization of human medial collateral ligament," J Biomech Eng, 120(6), pp. 757-763.

[19] Weiss, J. A., Maker, B. N., and Govindjee, S., 1996, "Finite element implementation of incompressible, transversely isotropic hyperelasticity," Computer Methods in Applications of Mechanics and Engineering, 135, pp. 107-128.
[20] Veronda, D. R., and Westmann, R. A., 1970, "Mechanical Characterization of Skin - Finite Deformations," J. Biomechanics, Vol. 3, pp. 111-124.
[21] Girard, M. J. A., Downs, J. C., and Burgoyne, C. F., 2009, "Peripapillary and posterior scleral mechanics - Part I: Development of an anisotropic hyperelastic constitutive model," J Biomech Eng, 131(5), p. 051011.
[22] Gouget, C. L. M., Girard, M. J. A., and Ethier, C. R., 2012, "A constrained von Mises distribution to describe fiber organization in thin soft tissues," Biomechanics And Modeling in Mechanobiolgy, 11(3-4), pp. 475-482.
[23] Bonet, J., and Wood, R. D., 1997, Nonlinear continuum mechanics for finite element analysis, Cambridge University Press.
[24] Carter, D. R., and Hayes, W. C., 1976, "Bone compressive strength: the influence of density and strain rate," Science, 194(4270), pp. 1174-1176.
[25] Carter, D. R., and Hayes, W. C., 1977, "The compressive behavior of bone as a two-phase porous structure," J Bone Joint Surg Am, 59(7), pp. 954-962.
[26] Overbeek, J. T., 1956, "The Donnan equilibrium," Prog Biophys Biophys Chem, 6, pp. 57-84.
[27] Lai, W. M., Hou, J. S., and Mow, V. C., 1991, "A triphasic theory for the swelling and deformation behaviors of articular cartilage," J Biomech Eng, 113(3), pp. 245-258.
[28] Holmes, M. H., and Mow, V. C., 1990, "The nonlinear characteristics of soft gels and hydrated connective tissues in ultrafiltration," J Biomech, 23(11), pp. 1145-1156.
[29] Ateshian, G. A., Warden, W. H., Kim, J. J., Grelsamer, R. P., and Mow, V. C., 1997, "Finite deformation biphasic material properties of bovine articular cartilage from confined compression experiments," J Biomech, 30(11-12), pp. 1157-1164.
[30] Iatridis, J. C., Setton, L. A., Foster, R. J., Rawlins, B. A., Weidenbaum, M., and Mow, V. C., 1998, "Degeneration affects the anisotropic and nonlinear behaviors of human anulus fibrosus in compression," J Biomech, 31(6), pp. 535-544.
[31] Ateshian, G. A., and Ricken, T., 2010, "Multigenerational interstitial growth of biological tissues," Biomech Model Mechanobiol, 9(6), pp. 689-702.
[32] Mow, V. C., Kuei, S. C., Lai, W. M., and Armstrong, C. G., 1980, "Biphasic creep and stress relaxation of articular cartilage in compression: Theory and experiments," J Biomech Eng, 102(1), pp. 73-84.
[33] Mow, V. C., Kwan, M. K., Lai, W. M., and Holmes, M. H., 1985, "A finite deformation theory for nonlinearly permeable soft hydrated biological tissues," Frontiers in Biomechanics, G. a. W. Schmid-Schonbein, SL-Y and Zweifach, BW, ed., pp. 153-179.
[34] Albro, M. B., Rajan, V., Li, R., Hung, C. T., and Ateshian, G. A., 2009, "Characterization of the Concentration-Dependence of Solute Diffusivity and Partitioning in a Model Dextran-Agarose Transport System," Cell Mol Bioeng, 2(3), pp. 295-305.
[35] Weinans, H., Huiskes, R., and Grootenboer, H. J., 1992, "The behavior of adaptive bone-remodeling simulation models," J Biomech, 25(12), pp. 1425-1441.