

COMSOL Multiphysics

Programming Reference Manual

COMSOL® Multiphysics Programming Reference Manual

© 1998–2015 COMSOL

Protected by U.S. Patents listed on www.comsol.com/patents, and U.S. Patents 7,519,518; 7,596,474; 7,623,991; 8,457,932; 8,954,302; 9,098,106; and 9,146,652. Patents pending.

This Documentation and the Programs described herein are furnished under the COMSOL Software License Agreement (www.comsol.com/comsol-license-agreement) and may be used or copied only under the terms of the license agreement.

COMSOL, COMSOL Multiphysics, Capture the Concept, COMSOL Desktop, LiveLink, and COMSOL Server are either registered trademarks or trademarks of COMSOL AB. All other trademarks are the property of their respective owners, and COMSOL AB and its subsidiaries and products are not affiliated with, endorsed by, sponsored by, or supported by those trademark owners. For a list of such trademark owners, see www.comsol.com/trademarks.

Version: COMSOL 5.2

Contact Information

Visit the Contact COMSOL page at www.comsol.com/contact to submit general inquiries, contact Technical Support, or search for an address and phone number. You can also visit the Worldwide Sales Offices page at www.comsol.com/contact/offices for address and contact information.

If you need to contact Support, an online request form is located at the COMSOL Access page at www.comsol.com/support/case. Other useful links include:

- Support Center: www.comsol.com/support
- Product Download: www.comsol.com/product-download
- Product Updates: www.comsol.com/support/updates
- COMSOL Blog: www.comsol.com/blogs
- Discussion Forum: www.comsol.com/community
- Events: www.comsol.com/events
- COMSOL Video Gallery: www.comsol.com/video
- Support Knowledge Base: www.comsol.com/support/knowledgebase

Part number: CM020007

C o n t e n t s

Chapter 1: Introduction

About the COMSOL API	14
Where Do I Find More Information?	14
Getting Started	16
The Model Object	16
Compiling a Model File for Java	16
The Model File for Java.	17
Running a Compiled Model File for Java from the Desktop	18
Running a Compiled Model File as a Batch Job from the Desktop.	18
Running a Compiled Model File with the COMSOL Batch Command	18
Getting the COMSOL Installation Path from the Windows Registry	18
Setting up Eclipse for Compiling and Running a Java File	18

Chapter 2: General Commands

About General Commands	22
Overview of General-Purpose Commands	22
getType()	23
set()	24
setEntry()	25
setIndex()	26
Methods Associated to Set, SetIndex, and the Various Get Methods	27
Selections	27
The loadFile and saveFile Methods	29
ModelUtil.	30
model	32
model.attr()	33
model.attr(<tag>)	34
model.batch()	35
model.bem()	42
model.capeopen()	44
model.coeff()	46
model.constr()	48
model.coordSystem()	51
model.cpl()	55
model.elem()	62
model.elementSet()	65
model.extraDim	67
model.field()	67
model.frame()	68
model.func()	71
model.geom()	79
model.group()	86
model.init()	86
model.intRule()	87
model.massProp()	88

model.material()	89
model.mesh()	98
model.modelNode()	102
model.multiphysics()	103
model.ode()	104
model.opt()	105
Least-Squares Objective Functions	106
model.pair()	107
model.param()	110
model.physics()	111
model.probe()	115
model.result()	118
model.savePoint()	121
model.selection()	122
model.shape()	128
model.sol()	130
model.solverEvent()	132
model.study()	134
model.unitSystem()	138
model.variable()	139
model.view()	141
model.weak()	144

Chapter 3: Geometry

About Geometry Commands	146
Features for Creating Geometric Primitives	146
Features for Geometric Operations	147
Selection Features	148
Features for Virtual Operations	149
Features for Mesh Control	149
Geometry Object Information Methods	150
Working with a Geometry Sequence	152
Adding a Model Component (Geometry)	152
Adding a Geometry Feature	152
Editing a Geometry Feature	153
Building Geometry Features	154
Feature Status	154
Accessing Geometry Object Names	155
Deleting and Disabling Geometry Features	155
Deleting Geometry Objects	156
Moving and Scaling Geometry Objects	156
Geometry Settings	157
Length Unit	157
Angular Unit	157
Scale Values When Changing Unit	157
Geometry Representation in 3D	158
Default Relative Repair Tolerance	158
Automatic Rebuild	159

Work Planes	160
Selections of Geometric Entities	161
Named Selections	161
Using Selection Features	161
Cumulative Selections	162
Virtual Operations	163
About Virtual Operations	163
Mesh Control Entities	163
Geometry Object Information	164
General Information.	164
Geometric Entity Counters	165
Adjacency	165
Evaluation on an Edge	166
Evaluation on a Face.	166
Geometry Representation Arrays	167
Measurements	169
Measuring Geometric Entities in Objects	169
Measuring Objects	169
Inserting Geometry Sequences from File	170
Example of Importing Geometry Sequences.	170
Exporting Geometry to File	171
Exporting to an ACIS File.	171
Exporting to a Parasolid File.	171
Exporting to an STL File	172
Compatibility for mphbin/mphtxt in 2D and 3D	172
Using Geometry Parts	173
Geometry Commands	175
AdjacentSelection	176
Array	177
BallSelection, BoxSelection, CylinderSelection	179
BezierPolygon	181
Block	183
Chamfer	184
Circle	186
CollapseEdges	187
CollapseFaces	187
Compose, Union, Intersection, Difference	188
CompositeDomains.	189
CompositeEdges	190
CompositeFaces	191
Cone	192
ConvertToSolid, ConvertToSurface, ConvertToCurve, ConvertToPoint	194
CrossSection	195
Cylinder	196
Delete	198
ECone	199

EditObject	201
Ellipse	203
Ellipsoid	204
ExplicitSelection	205
Extrude	207
Fillet.	208
Finalize.	209
FromMesh	210
Helix	210
Hexahedron.	212
If, Elseif, Else, EndIf	213
IgnoreEdges	214
IgnoreFaces	215
IgnoreVertices	216
Import DXF.	217
Import Geometry Sequence	218
Import Mesh Part or Meshing Sequence	220
Import mphbin/mphtxt.	221
Interpolation Curve.	222
Interval	223
MergeEdges	224
MergeVertices	225
MeshControlDomains	225
MeshControlEdges	226
MeshControlFaces	227
MeshControlVertices	227
Mirror	228
Move, Copy	229
ParameterCheck	230
ParametricCurve	231
ParametricSurface	232
PartInstance.	233
Partition	236
PartitionDomains.	237
PartitionEdges	238
Point	239
Polygon	240
Pyramid	241
Rectangle.	243
Revolve	243
Rotate	245
Scale	247
Sphere.	248
Split	249
Square	250
Sweep	252
Tangent	254
Tetrahedron.	256
Torus	257
UnionSelection, IntersectionSelection, DifferenceSelection, ComplementSelection	258
WorkPlane	260

Chapter 4: Mesh

About Mesh Commands	268
Operation Features	268
Attribute Features	269
Features for Imported Meshes	269
 Working with a Meshing Sequence	 270
Adding a Meshing Sequence	270
Adding a Mesh Feature.	270
Editing a Mesh Feature.	271
Building Mesh Features.	271
Using Mesh Parts	272
Feature Status	272
Deleting Mesh Features	272
Disabling Mesh Features	272
Clearing Meshes	273
Units	273
Selections	273
 Physics-Controlled Meshing	 275
 Information and Statistics	 276
Statistics	276
Number and Types of Elements	277
Quality of Elements	277
Volume of Elements and Mesh	278
Growth Rate in Mesh	278
Mesh Status	278
 Getting and Setting Mesh Data	 279
Accessing Mesh Data	279
Setting or Modifying Mesh Data	279
Block Versions.	282
Element Numbering Conventions	282
 Errors and Warnings	 284
Continuing Operations	284
Stopping Operations	284
The MeshError Feature	284
The MeshWarning Feature	284
Retrieving Problem Information	284
 Exporting Meshes to Files	 286
Exporting Mesh to a File	286
Exporting Mesh to a COMSOL Multiphysics File	286
Exporting Mesh to a NASTRAN [®] File.	286
 Mesh Commands	 288
Ball	288
BndLayer	289
BndLayerProp	292
Box	292

Convert	293
CopyEdge	295
CopyFace.	296
CopyDomain	298
Copy	299
CornerRefinement	301
CreateVertex	301
Cylinder	302
Delete	302
DeleteEntities	304
DetectFaces.	305
Distribution	305
Edge.	306
EdgeGroup	307
EdgeMap	308
FreeQuad	309
FreeTet	310
FreeTri.	312
Import.	313
JoinEntities	315
LogicalExpression.	316
Map	316
OnePointMap	317
Point	319
Reference	319
Refine	320
Scale	322
Size	324
Sweep	326
TwoPointMap	328

Chapter 5: Elements and Shape Functions

Shape Functions and Element Types	332
Finite Elements	332
Shape Function Types (Elements)	334

Chapter 6: Solver

About Solver Commands	354
Features Producing and Manipulating Solutions.	354
Features with Solver Settings	355
Solution Object Information Methods	355
Solution Feature Information Methods.	356
Solution Object Data	358
General Information.	358
Solution Data	359
SolutionInfo Object and Its Methods	361
Solution Creation.	364
General Matrix Information	365

Matrix Data	365
Matrix Creation	366
Adaption	367
Advanced.	370
Assemble.	372
AutoRemesh	373
AWE	374
Eigenvalue	376
FFT	377
For, EndFor	380
FullyCoupled	381
InputMatrix	383
Linear	384
Modal	392
Optimization	394
Parametric	396
Previous Solution.	397
Segregated	397
SegregatedStep	399
Sensitivity.	400
StateSpace	401
Stationary	402
StopCondition.	403
StoreSolution	404
StudyStep.	405
Time	405
TimeDiscrete	410
TimeExplicit.	411
TimeParametric	412
Variables	413
XmeshInfo	414

Chapter 7: Results

About Results Commands	420
Commands Grouped by Function	421
Use of Data Sets	425
Extracting Data	427
Retrieving Plot Data.	427
Retrieving Numerical Results	428
Solution Selection	430
About Selecting Solutions	430
Selecting Solutions by Solution Number	430
Selecting Solutions by Solution Level	430
Choosing Solution Selection Method	431
Animation	431
Annotation	434
Array2D, Array3D	436
ArrowVolume, ArrowSurface, ArrowLine	437

AvVolume, AvSurface, AvLine	440
Average, Integral, Maximum, Minimum	443
Color	445
Contour	447
Contour (data set)	451
CoordSysLine, CoordSysSurface, CoordSysVolume	451
CutLine2D, CutLine3D	454
CutPlane	455
CutPoint1D, CutPoint2D, CutPoint3D	457
Data	458
Deform	460
Edge2D, Edge3D	461
Eval	461
EvalGlobal	463
EvalGlobalMatrix	466
EvalPoint	469
EvalPointMatrix	472
FarField	475
Filter	479
Filter (Particle Tracing)	479
Filter (Point Trajectories)	480
Filter (Ray Tracing)	481
Global (numerical)	481
Global (plot)	483
Grid1D, Grid2D, Grid3D	487
Height	488
Histogram	489
HistogramHeight	493
Image1D, Image2D, Image3D	494
InterferencePattern	495
Interp	498
IntVolume, IntSurface, IntLine	500
Isosurface	504
Isosurface (data set)	507
Join	507
Line	508
LineGraph	512
MatrixHistogram	516
MaxMinVolume, MaxMinSurface, MaxMinLine	518
MaxVolume, MaxSurface, MaxLine, MinVolume, MinSurface, MinLine	520
Mesh	523
Mesh (data set)	524
Mesh (export)	525
Mirror2D, Mirror3D	525
Multislice	527
Nyquist	531
OctaveBand	533
Parametric1D, Parametric2D	536
ParCurve2D, ParCurve3D	537
ParSurface	538
Particle	539
Particle (1D Plot)	543
Particle (data set)	547
Particle (Evaluation)	548

ParticleMass	550
ParticleTrajectories	555
PhasePortrait	558
Plot	560
PlotGroup1D, PlotGroup2D, PlotGroup3D	561
PoincareMap	567
PointGraph	569
PointTrajectories	573
PolarGroup	576
PrincipalSurface, PrincipalVolume	579
Ray (1D Plot)	581
Ray (Data Set)	585
Ray (Evaluation)	585
Ray Trajectories	588
ReflectionGraph, ImpedanceGraph, AdmittanceGraph	590
Revolve1D, Revolve2D	594
ScatterVolume, ScatterSurface	595
Sector2D, Sector3D	598
Slice	599
SmithGroup	604
Solution	606
Streamline	607
Surface	612
Surface (data set)	616
SystemMatrix	616
Table	617
Table (export)	620
Table (plot)	621
TableHeight	623
TableSurface	624
TimeAverage, TimeIntegral	626
Volume	627

Chapter 8: Graphical User Interfaces

Getting Started	632
Example Graphical User Interface	633
Introduction	633
Downloading Extra Material	634
Creating the Code for the Model	634
Construction of the Initial GUI with Graphics	635
Handling of Progress Information	636
Setting Up Inputs From the GUI to the Model	638
Displaying Results in the GUI	639
Other Details	641
GUI Classes	645
ProgressContext	645
ProgressWorker	645
SWTGraphicsPanel	646
SwingGraphicsPanel	646

Chapter 9: The COMSOL File Formats

File Formats	648
Data Formats	649
Spreadsheet Data Format	649
Grid Data Format	650
Sectionwise Data Format	650
Supported Microsoft Excel File Types	651
Color Tables	652
Continuous Color Tables	652
Discrete Color Tables	652
Binary Data Files and Text Data Files	653
File Structure	653
Records	654
Terminology	654
Text File Format	654
Binary File Format	655
Serializable Types	656
Attribute	656
BezierCurve	657
BezierMfd	658
BezierSurf	658
BezierTri	659
BSplineCurve	660
BSplineMfd	661
BSplineSurf	662
Ellipse	663
Geom0	664
Geom1	664
Geom2	665
Geom3	666
Manifold	667
Mesh	667
MeshCurve	668
MeshSurf	669
Plane	670
PolChain	670
Serializable	671
Solution	671
Straight	676
Transform	677
VectorDouble	678
VectorInt	678
VectorString	679
Examples of the Serialization Format	680
A Mesh with Mixed Element Types	680
A Planar Face	680

Introduction

This *COMSOL[®] Multiphysics Programming Reference Manual* details features and techniques that help you control COMSOL Multiphysics[®] using the application programming interface (API) for use with the Java[®] programming language. The COMSOL API can be used in the Application Builder, from a standalone Java application, and from the LiveLink[™] for MATLAB[®] interface.

In this chapter:

- [About the COMSOL API](#)
- [Getting Started](#)

About the COMSOL API

You can use the COMSOL API to develop custom applications based on COMSOL. The easiest way to create such applications is by using the Application Builder available with the COMSOL software.

You can run Java class files with COMSOL API-based applications in different ways:

- In the Application Builder as part of methods that you add using the Method Editor in the Application Builder's development environment.
- From the COMSOL Desktop. A model created using a class file appears automatically in the Desktop.
- From a batch sequence in a study.
- Using the `comsol batch` command.

The LiveLink™ for MATLAB® operates using the COMSOL API and additional utility M-file functions. See the *LiveLink™ for MATLAB® User's Guide* for additional information.

Code examples for the individual API functions in this guide show what the code looks like when using Java® and when using the LiveLink™ for MATLAB®.

Where Do I Find More Information?

A number of Internet resources have more information about COMSOL, including licensing and technical information. The electronic documentation, topic-based (or context-based) help, and the application libraries are all accessed through the COMSOL Desktop.



If you are reading the documentation as a PDF file on your computer, the [blue links](#) do not work to open an application or content referenced in a different guide. However, if you are using the Help system in COMSOL Multiphysics, these links work to open other modules (as long as you have a license), application examples, and documentation sets.

CONTACTING COMSOL BY EMAIL

For general product information, contact COMSOL at info@comsol.com.

To receive technical support from COMSOL for the COMSOL products, please contact your local COMSOL representative or send your questions to support@comsol.com. An automatic notification and case number is sent to you by email.

COMSOL WEBSITES

COMSOL website	www.comsol.com
Contact COMSOL	www.comsol.com/contact
Support Center	www.comsol.com/support
Product Download	www.comsol.com/product-download
Product Updates	www.comsol.com/support/updates
Discussion Forum	www.comsol.com/community
Events	www.comsol.com/events
COMSOL Video Gallery	www.comsol.com/video
Support Knowledge Base	www.comsol.com/support/knowledgebase

Getting Started

In this section:

- [The Model Object](#)
- [Compiling a Model File for Java](#)
- [The Model File for Java](#)
- [Running a Compiled Model File for Java from the Desktop](#)
- [Running a Compiled Model File as a Batch Job from the Desktop](#)
- [Running a Compiled Model File with the COMSOL Batch Command](#)
- [Getting the COMSOL Installation Path from the Windows Registry](#)
- [Setting up Eclipse for Compiling and Running a Java File](#)

The Model Object

In the COMSOL API you access models through the *model object*, which contains all algorithms and data structures for a COMSOL model. The COMSOL Desktop also uses the model object to represent your model. This means that the model object and the COMSOL Desktop behavior are virtually identical.

You use *methods* to create, modify, and access your model. The model object provides a large number of methods, including methods for setting up and running *sequences of operations* to create geometry, meshes, and for solving your model. The methods are structured in a tree-like way, much similar to the nodes in the model tree in the *Model Builder* window on the COMSOL Desktop. The top-level methods just return references that support further methods. At a certain level the methods perform actions, such as adding data to the model object, performing computations, or returning data.

You must have a basic understanding of the Java[®] programming language in order to fully appreciate how to work with the model object. However, the Application Builder includes tools like code recording, auto-completion, and predefined code templates that make it easier to create methods.




Compiling a Model File for Java

First make sure that COMSOL Multiphysics is installed. See the *COMSOL Multiphysics Installation Guide* for more information if required.

Before getting started, also download and install a Java[®] Development Kit (JDK) from www.oracle.com/technetwork/java/index.html. You need the JDK to compile the Java file that COMSOL generates. You must compile your Java files with JDK 1.7 or lower.

To test compiling a model files for Java, load `feeder_clamp.mph` from the COMSOL Multiphysics Applications Libraries into the COMSOL Desktop.

You can learn most of the syntax for creating a model using the COMSOL API by first creating a model using the COMSOL Desktop and then saving the model as an application file for Java.

Win	<p>To open the Application Libraries window:</p> <ul style="list-style-type: none"> From the Home toolbar, Windows menu, select Application Libraries (). When a toolbar is compressed, you sometimes select it from the Windows> menu. You can also customize the Quick Access Toolbar and then click the Application Libraries () button on the toolbar. Select Application Libraries from the File menu.
<div>Mac</div> <div>Linux</div>	<p>To open the Application Libraries window:</p> <ul style="list-style-type: none"> On the Main Toolbar, click the Application Libraries () button. Select Windows>Application Libraries.

In the Application Libraries tree, expand **COMSOL Multiphysics** and then **Structural Mechanics**. Select the **feeder_clamp** model, then click the **Open** button to open it. To get a Java file to compile, choose **Save As** from the **File** menu and choose **Model file for Java (*.java)** as the file type. It is suggested that you save the file as `feeder_clamp.java` in your home directory.

To compile `feeder_clamp.java`, enter

```
<COMSOL path>\bin\win64\comsolcompile -jdkroot <JDK path> feeder_clamp.java
```

on Windows and

```
<COMSOL path>/bin/comsol compile -jdkroot <JDK path> \ feeder_clamp.java
```

on Linux and Mac, where `<COMSOL path>` is the COMSOL installation directory and `<JDK path>` is the installation directory for the JDK.

The Model File for Java

The model file for Java has the following structure:

```
import com.comsol.model.*;
import com.comsol.model.util.*;

public class feeder_clamp {

    public static void main(String[] args) {
        run();
    }

    public static Model run() {
        Model model = ModelUtil.create("Model");
        ...
        return model;
    }
}
```

Any model that you create in the COMSOL Desktop can be saved as a model file for Java.



When you compile a model file for Java into a class file and run it, COMSOL runs exactly those instructions that are included in the model file for Java. When opening an MPH-file and saving it as a Java file only those sequences that have been explicitly run are run in the Java file. But saving it as a model file for Java, the file does not contain a `runAll` command for the solver sequence. To run a solver sequence, add a line similar to `model.sol("sol1").runAll();` (where `sol1` is the tag for the solver to run) at the bottom of the Java file, above the line that contains `return model;`.

Running a Compiled Model File for Java from the Desktop

Select **Open** on the **File** menu. In the **Open** dialog box, under **File** name, select **Compiled Model File for Java (*.class)**. Click **Open**. The file is run and appears as the model in the COMSOL Desktop.

Running a Compiled Model File as a Batch Job from the Desktop

Right-click **Job Sequences** in a study and add a study. In the added study, right-click and add **External Class** under **Other**. Then right-click the batch sequence and select **Compute**.

Runs the main function of a compiled class with the system property `cs.currentmodel` set to the tag of the model calling the class. Thus you can retrieve the current model using the steps:

```
import java.io.*;

tag = System.getProperty("cs.currentmodel");
model = ModelUtil.model(tag);
```

Running a Compiled Model File with the COMSOL Batch Command

To run the file, enter

```
<COMSOL path>\bin\win64\comsolbatch -inputfile feeder_clamp.class
```

on Windows, or enter

```
<COMSOL path>/bin/comsol batch -inputfile feeder_clamp.class
```

on Linux and Mac, where `<COMSOL path>` is the COMSOL Multiphysics installation directory.

Getting the COMSOL Installation Path from the Windows Registry

If you want to have your application find your COMSOL Multiphysics installation automatically you can have your application examine the registry key

```
HKEY_LOCAL_MACHINE\SOFTWARE\COMSOL\COMSOL52\
```

The value name `COMSOLROOT` contains the installation path.

Setting up Eclipse for Compiling and Running a Java File

Instead of using the COMSOL commands for compiling and running a Java file that uses the COMSOL API one can use an Integrated Development Environment for doing these tasks. Using Eclipse makes it easier to write the Java[®] code because Eclipse has built-in support for code completion and syntax highlighting. Furthermore, the debugger that comes as a part of Eclipse can be used to run the code line by line to verify the function of the code

and check for any programming errors. Eclipse is free and can be downloaded from www.eclipse.org. In order to set up Eclipse for running an exported Java file perform the following actions in Eclipse:

- 1 Create a new Java Project and click **Next**.
- 2 Go to the Libraries tab and click **Add External JARs**. Add all the JAR files placed in the `plugins` directory under the COMSOL installation directory (typically `C:\Program Files\COMSOL\COMSOL52\Multiphysics`). This allows Eclipse to find the definitions of the classes used by the COMSOL API and to run the code in client/server mode. Click **Finish**.
- 3 Drag and drop your exported Java file the `src` folder of your Eclipse project.
- 4 Add this line to the beginning of the main method

```
ModelUtil.initStandalone(false);
```

The argument should be false for programs that do not use graphics and true for applications that does.

- 5 The Java program can now be started in either Run or Debug mode from Eclipse. The Java program is now being run as a single process where the COMSOL libraries are being loaded as requested. This is the preferred way of running normal, small model files.
- 6 For large simulation where the application itself has to hold many megabytes in memory in addition to the memory requirement of COMSOL it can be beneficial to run in client/server mode. Open the Java file in the editor and go to the main method. Now you have to remove the line added in step 4 and add two new lines that control the connection to the COMSOL server from your own program. The main method needs to look like this:

```
public static void main(String[] args) {  
    ModelUtil.connect("localhost", 2036);  
    run();  
    ModelUtil.disconnect();  
}
```

When you have edited the main method you must save the file. Eclipse automatically compiles the file.

- 7 To run the code you must first start the COMSOL server. When the server has started note the port number that is written in the console. If this number does not match the number written in the call to `ModelUtil.connect` you have to edit this call and save the file again.

The Java program can now be started in either Run or Debug mode from Eclipse. Notice that the COMSOL server window responds by writing that a connection has been set up when your application starts.

General Commands

This chapter contains reference information about general commands for creating and modifying the main parts of the model object and for creating general-purpose functionality in a model, such as functions, variables, units, coordinate systems, and component couplings. It also includes information about the available shape functions (element types).

About General Commands

Overview of General-Purpose Commands

The following table contains the available general-purpose commands and methods:

TABLE 2-1: GENERAL COMMANDS GROUPED BY FUNCTION

FUNCTION	PURPOSE
<code>getType()</code>	Access objects of the basic data types
<code>set()</code>	Assign objects of the basic data types
<code>setEntry()</code>	Set vector property value at specified entry
<code>setIndex()</code>	Assign objects at indices of the basic data types
<code>Selections</code>	Selections
<code>ModelUtil</code>	Model object utility methods
<code>model</code>	Model object
<code>model.attr()</code>	Model entity list methods
<code>model.attr(<tag>)</code>	Model entity methods
<code>model.batch()</code>	Batch jobs
<code>model.bem()</code>	Boundary elements (BEM)
<code>model.capeopen()</code>	CAPE-OPEN thermodynamics interface
<code>model.coeff()</code>	Coefficient form equations
<code>model.constr()</code>	Constraints
<code>model.coordSystem()</code>	Coordinate systems
<code>model.cpl()</code>	Component couplings
<code>model.elem()</code>	Elements
<code>model.elementSet()</code>	Mesh element sets
<code>model.extraDim</code>	Extra dimensions
<code>model.field()</code>	Fields
<code>model.frame()</code>	Frames
<code>model.func()</code>	Functions
<code>model.geom()</code>	Geometry sequences
<code>model.group()</code>	Load groups and constraint groups
<code>model.init()</code>	Initial values
<code>model.intRule()</code>	Integration orders
<code>model.massProp()</code>	Mass properties
<code>model.material()</code>	Materials
<code>model.mesh()</code>	Meshing sequences
<code>model.modelNode()</code>	Model nodes
<code>model.multiphysics()</code>	Multiphysics features container
<code>model.ode()</code>	Global equations
<code>model.opt()</code>	Optimization interface
<code>model.pair()</code>	Pairs
<code>model.param()</code>	Parameters
<code>model.physics()</code>	Physics

TABLE 2-1: GENERAL COMMANDS GROUPED BY FUNCTION

FUNCTION	PURPOSE
<code>model.probe()</code>	Probes
<code>model.result()</code>	Postprocessing interface
<code>model.savePoint()</code>	Manage selections and hide features used by result features
<code>model.selection()</code>	Named selections
<code>model.shape()</code>	Shape functions
<code>model.sol()</code>	Solver sequences
<code>model.solverEvent()</code>	Events
<code>model.study()</code>	Studies
<code>model.unitSystem()</code>	Unit systems
<code>model.variable()</code>	Variables
<code>model.view()</code>	Views
<code>model.weak()</code>	Weak form equations

ABOUT VALID TAGS

A *tag* is a string that you use to refer to a model feature. When specifying a tag, it must fulfill the following format requirements: Begin with a character a–z or A–Z followed by any number of `_` (underscores), numerals 0–9, or characters a–z or A–Z.

ABOUT ASSIGNING VALUES TO PROPERTIES

Even if a property is numeric, it is also possible to use a string or string array. The strings can contain expressions defined in terms of parameters defined in Global Definitions>Parameters. See [Table 2-2](#) under `set()` below for examples of syntaxes for assignment methods.

getType()

Use these methods to access objects of the basic data types.

SYNTAX

```
something.getStringArray(<name>);
```

Note that throughout this manual, the access methods are collectively referred to as `getType(<name>)`, where *Type* can be any of the basic data types listed above.

Use these methods to read parameter/property values.

The names of the access methods indicate the data type returned.

```
something.getString(<name>)
```

returns the value as a string.

```
something.getStringArray(<name>)
```

returns the value as a string array.

```
something.getStringMatrix(<name>)
```

returns the value as a string matrix.

```
something.getInt(<name>)
```

returns the value as an integer.

```
something.getIntArray(<name>)
```

returns the value as an integer array.

```
something.getDouble(<name>)
```

returns the value as a double.

```
something.getDoubleArray(<name>)
```

returns the value as a double array.

```
something.getDoubleMatrix(<name>)
```

returns the value as a double matrix.

```
something.getBooleanArray(<name>)
```

returns the value as a Boolean array.

```
something.getBooleanMatrix(<name>)
```

returns the value as a Boolean matrix.

```
something.selection(<name>)
```

returns the value as a selection object, which can be edited. This is not simply an access function. It used to obtain a selection object both for editing and accessing data from.

In addition,

```
something.getEntryKeyIndex(<name>, <key>)
```

returns the index of a given key in a property, and

```
something.getEntryKeys(<name>)
```

returns the possible entry keys for a given property.

NOTES

All arrays that are returned contain copies of the data; writing to the array does not change the data in the model object. This observation applies to all access methods of the model object that return arrays of basic data types.

SEE ALSO

[set\(\)](#)

set()

Use this method to assign objects of the basic data types.

SYNTAX

Use these methods to assign parameter/property values. All assignment methods return the parameter object, which means that assignment methods can be appended to each other.

The basic method for assignments is

```
something.set(name, <value>);
```


The *name* argument is a string with the name of the parameter/property. The *<value>* argument can be of different types as indicated in [Table 2-2](#), where the two different syntaxes for assignment in the COMSOL API and the LiveLink™ for MATLAB® are listed.

TABLE 2-2: SYNTAXES FOR ASSIGNMENT METHODS.

TYPE	JAVA® SYNTAX	MATLAB® SYNTAX
string	<code>set("name", "value")</code>	<code>set('name', 'value')</code>
string array	<code>set("name", new String[]{"val1", "val2"})</code>	<code>set('name', {'val1', 'val2'})</code>
double string array	<code>set("name", new String[][]{{"1", "2"}, {"3", "4"}})</code>	<code>set('name', {'1', '2'; '3', '4'})</code>
integer	<code>set("name", 17)</code>	<code>set('name', 17)</code>
integer array	<code>set("name", new int[]{1, 2})</code>	<code>set('name', [1 2])</code>
double	<code>set("name", 1.3)</code>	<code>set('name', 1.3)</code>
double array	<code>set("name", new double[]{1.3, 2.3})</code>	<code>set('name', [1.3 2.3])</code>
double matrix	<code>set("name", new double[][]{{1.3, 2.3}, {3.3, 4.3}})</code>	<code>set('name', [1.3 2.3; 3.3 4.3])</code>
boolean	<code>set("name", true)</code>	<code>set('name', true)</code>
boolean array	<code>set("name", new boolean[]{true, false})</code>	<code>set('name', [true, false])</code>
boolean matrix	<code>set("name", new boolean[][]{{true, false}, {false, false}})</code>	<code>set('name', [true, false; false, false])</code>

For matrix-type properties, `set(name, <string>)` splits the string at spaces and commas.

The following methods using `set` are deprecated in version 5.1 (use `setIndex()` instead):

```
set(name, pos, <value>)
set(name, pos1, pos2, <value>)
```

The following methods using `set` are deprecated in version 5.0:

```
com.comsol.model.ParameterEntity.set(String, int, double)
com.comsol.model.ParameterEntity.set(String, int, double[])
com.comsol.model.ParameterEntity.set(String, int, int)
com.comsol.model.ParameterEntity.set(String, int, int[])
com.comsol.model.ParameterEntity.set(String, int, int, double)
com.comsol.model.ParameterEntity.set(String, int, int, int)
com.comsol.model.ParameterEntity.set(String, int, int, String)
com.comsol.model.ParameterEntity.set(String, int, String)
com.comsol.model.ParameterEntity.set(String, int, String[])
```

SEE ALSO

[getType\(\)](#), [setIndex\(\)](#)

setEntry()

Use the `setEntry` method to set property values at specified entries.

SYNTAX

Use this method to assign property values of different types at specified entries:

```
setEntry(name, key, value);
```

where all input arguments are strings:

- *name*, representing the property name.

- *key*, representing the index key.
- *value*, representing the property value.

The *value* can also be a double, an integer, or (with limited applicability; primarily for part links) a Boolean.

For example, if *pi1* is a part instance feature with input parameters *a* and *b* in a geometry, the following lines

```
GeomFeature f = model.geom("geom1").feature("pi1");
f.setEntry("inputexpr", "a", "2");
f.setEntry("inputexpr", "b", "3");
```

set the value of the input expression for parameter *a* to the string 2 and for parameter *b* to the string 3. You can also set the values of the input expressions to numerical values (doubles):

```
f.setEntry("inputexpr", "a", 4);
f.setEntry("inputexpr", "b", 5.0);
```

setIndex()

Use this method to assign objects at indices of the basic data types. When there are no indices, you can use [set\(\)](#) instead.

SYNTAX

To use the `setIndex` method, use one of these syntaxes:

```
something.setIndex(name,<value>,<index>);
```

or

```
something.setIndex(name,<value>,<firstIndex>,<secondIndex>);
```

Use these methods to assign values to specific indices (0-based) in array or matrix properties. All assignment methods return the parameter object, which means that assignment methods can be appended to each other.

If *<index>* points beyond the current size of the array, then the array is extended as needed before element *<index>* is set. The values of any newly created intermediate elements are undefined.

The *name* argument is a string with the name of the property. *<value>* is a string representation of the value to set. A double array element, for example, can still be set from a string representation of the double, typically used when the property value depends on a model parameter. The values can also be a Boolean or a Boolean array. For example:

```
something.setIndex(name,<value>,2)
```

This sets the third index (because the indices are 0-based) of an array property *name* to be the value *value*. If the parameter later changes, this property changes accordingly. You can also use an additional input argument for a second index value, for a 2-dimensional array (matrix), for example,

```
something.setIndex(name,<value>,1,4)
```

This sets the third index for the first array dimension, and the fifth index for the second array dimension for a two-dimensional array (matrix).

For double arrays the modifying method is also of use when assigning the value in MATLAB[®], if not all arrays have the same length. When using a cell matrix, all rows must have the same length. The method

```
something.setIndex(name,<value>,<index>)
```

can be used to get around that limitation. It inserts an array in the indexed position in the double array. The MATLAB code

```
something.setIndex('name',{'1','2','3'},0)
something.setIndex('name',{'4','5'},1)
```

is equivalent to the Java[®] code

```
something.set("name",new String[][]{{"1","2","3"},{"4","5"}})
```

SEE ALSO

[set\(\)](#)

Methods Associated to Set, SetIndex, and the Various Get Methods

The following methods are available where the `set`, `setIndex`, and `get<Type>` methods are available:

```
String[] properties();
```

which returns the names of all available properties.

```
boolean hasProperty(String name);
```

which returns true if the feature has the named property.

```
String[] getAllowedPropertyValues(String name);
```

which returns the allowed values for a named properties, if it is a finite set.

Selections

This section contains information about the selection methods that are available for handling selections on the finalized geometry.

SYNTAX

This section describes the general syntax for selections on the finalized geometry. For selections in the geometry sequence, see [Geometry Object Selection Methods](#) under `model.geom()` and [Selections](#) under [Editing a Geometry Feature](#). Many objects use selections, but most of them only support a subset of the assignment methods described here. The methods supported by the named selections in `model.selection()` are listed in [model.selection\(\)](#). Other objects that use a selection support the methods relevant for the type of feature they represent. For example, a physics boundary condition feature requires a boundary selection. Therefore it does not support `selection.global()`, which makes the selection global, or `selection.allGeom()`, which makes the selection apply to the whole geometry at all levels. Using an unsupported assignment method results in an error.

There can also be a filtering of the entities assigned to a selection. Again, take a physics boundary condition as an example. Some boundary conditions only apply to the boundaries exterior to the domains where the physics interface is active, other boundary conditions only to boundaries interior to where the physics interface is active, and so on. Therefore `selection.entities(dim)` can sometimes return less entities than have been assigned using `selection.set(<entlist>)`. On the other hand `selection.inputEntities()` always returns all entities used in the assignment `selection.set(<entlist>)`.

Some selections only allow a single geometric entity, a single domain, a single boundary, edge, or point. Such selections are called *single selections*. Single selections cannot be defined by another selection and therefore do not support `selection.named(<stag>)`.

`selection.global()` sets the selection to be the global selection.

`selection.geom(<gtag>)` sets the selection to be all domains in the geometry `<gtag>`.

`selection.geom(<gtag>, dim)` specifies that subsequent calls to `all`, `set`, `add`, and `remove` refer to domains at the dimension `dim` on the geometry `<gtag>`. If there is only one possible geometry, using `selection.geom(dim)` is equivalent. Also, if there is only one allowed dimension `dim`, then `all`, `set` and `remove` can be used directly as it is then unambiguous to which geometry and dimension their arguments apply to.

`selection.geom(<gtag>, highdim, lowdim, typelist)` specifies that subsequent calls to `all`, `set`, `add`, and `remove` refer to domains of dimension `highdim` on the geometry `<gtag>`. The domains that are obtained are those that are both of dimension `lowdim` and of any of the types listed in `<typelist>`. It is required that `highdim > lowdim`. The available types are:

- **exterior**: All domains of dimension `lowdim` that lie on the exterior of the domains at dimension `highdim`.
- **interior**: All domains of dimension `lowdim` that lie in the interior of the domains at dimension `highdim`.
- **meshinterior**: All mesh boundaries of dimension `lowdim` that lie in the interior of the domains at dimension `highdim`.

`selection.allGeom()` sets the selection to a whole geometry. Can be used instead of `selection.geom(<gtag>)` when the geometry tag is unambiguous.

`selection.geom(dim)` specifies that subsequent calls to `all`, `set`, `add`, and `remove` refer to domains at the dimension `dim`. Can be used instead of `selection.geom(<gtag>, dim)` when the geometry tag is unambiguous.

`selection.all()` sets the selection to use all geometric entities in the geometry at the dimension where the selection applies.

`selection.set(<entlist>)` sets the selection to use the geometric entities in `<entlist>`. Note that the list of domain numbers is always sorted in ascending order and duplicates are removed before storing the numbers in the selection object.

`selection.add(<entlist>)` adds the geometric entities in `<entlist>` in the geometry to the set of geometric entities that the selection uses to obtain the selection.

`selection.remove(<entlist>)` removes the geometric entities in `<entlist>` in the geometry from the set of geometric entities that the selection uses.

`selection.inherit(boolean)` indicates whether the selection should include all geometric entities that are specified by any of the other methods and all geometric entities at lower dimensions that are adjacent to the ones already specified.

`selection.named(<stag>)` specifies that the selection is defined by the selection `model.selection(<stag>)`.

`selection.isGlobal()` returns true if the selection is global.

`selection.isGeom()` returns true if the selection is a whole geometry.

`selection.geom()` returns the geometry tag of the selection as a string. If the selection is global, `null` is returned.

`selection.dimension()` returns the dimensions on a geometry where the selection applies as an integer array.

`selection.entities(dim)` returns the geometric entities of the selection on the given geometry at the given dimension as an integer array.

`selection.interiorEntities(dim)` returns the interior mesh domains as an integer array.

`selection.isInheriting()` returns true if the selection is inherited to lower dimension levels.

`selection.inputDimension()` returns the dimension of the domains used as input to the selection.

`selection.inputEntities()` returns the entities used as input to the selection.

If the selection is defined by another selection, `selection.named()` returns the tag of that selection. Otherwise `selection.named()` returns an empty string.

Selections of the class `XDLocalSelection` have the following additional methods:

`selection.extraDim()` returns the tag of a feature of type `AttachDimension` in `model.extraDim()`, or an empty string if no extra dimension attachment is used.

`selection.extraDim(<attachdimtag>)` sets the extra dimension attachment feature. `<attachdimtag>` must be the tag of a feature of type `AttachDimension` in `model.extraDim()` or an empty string to indicate no extra dimension attachment.

`selection.extraDimSel(<xdgeomtag>)` returns the selection in extra dimension geometry `<xdgeomtag>`. `<xdgeomtag>` should be the tag of the geometry in one of the extra dimensions attached by the extra dimension attachment feature defined by `selection.extraDim()`.

`extraDimSel()` returns all extra dimension geometry tags that are valid arguments of `extraDimSel(<xdgeomtag>)` as a string array.

NOTES

The methods `global()`, `geom(<gtag>)`, `geom(<gtag>, dim)`, `geom(<gtag>, highdim, lowdim, typelist)`, and `geom(dim)` clear the data set by other methods.

Not all assignment methods are supported by all model entities. The list of supported methods also serves as a guide for the restriction to those named selections that can be used by that entity. All access methods are always supported.

SEE ALSO

[model.geom\(\)](#)

The loadFile and saveFile Methods

The methods `loadFile()` and `saveFile()` are now available to load and save files for the following features:

- `model.param()`
- `model.variable()`
- `model.result().table()`

You can use the following syntax for `loadFile`:

`loadFile(String path)` where the path is the path to any file type that COMSOL supports.

`loadFile(String path, char delim)` where `delim` is the delimiter used in the file.

`loadFile(String path, String sheet, String cell)`, where the string `sheet` is the name of the sheet to read from (null or empty means the first in the file), and `cell` is a cell that indicates the top-left cell to read.

For the `saveFile` method, the following syntax is available:

`saveFile(String path)`

`saveFile(String path, char delim)`

`saveFile(String path, String sheet, String cell, boolean includeHeaders, boolean overwrite)`, where `includeHeaders` determines whether to include headers or not, and `overwrite` determines if nonempty existing cells in a spreadsheet can be overwritten in the file. For example, a call like

```
model.param().saveFile(tempFile, "sheet", "C7", false, true);
```

starts saving at cell C7, does not include headers, and allows overwriting of nonempty cells.

For `model.result().table`, there are no delimiter functions.

Model object utility methods such as methods to create and remove model objects, showing progress information, and listing and saving preferences. See also [model](#).

SYNTAX

This section describes general methods that handle the environment for the model object. It also describes methods for the client/server machinery.

```
import com.comsol.model.*;
import com.comsol.model.util.*;
```

The import statements above make all model and model utility methods available.

`ModelUtil.create(<tag>)` creates a model with tag <tag>. Returns a reference to the model. If there is already a model with this tag the previous model is removed.

`ModelUtil.remove(<tag>)` removes the model tagged <tag>.

`ModelUtil.clear()` removes all models.

`ModelUtil.tags()` obtains the current list of model tags.

`ModelUtil.model(<tag>)` returns a reference to the model tagged <tag>.

`ModelUtil.closeWindow(<windowtag>)` closes the window tagged <windowtag>.

`ModelUtil.closeWindows()` closes all windows on the server.

`ModelUtil.createApplication(<tag>,<appname>)` creates a new application model with the tag <tag> and the name <appname> using the given application file reference.

`ModelUtil.createUnique(<prefix>)` creates a model with a unique tag that begins with the prefix <prefix>.

`ModelUtil.getOpenGeometryKernel()` returns the geometry kernel to use when opening models. Valid values are `model` for the geometry kernel used by the model file, and `comsol` to convert the geometry to the COMSOL kernel.

`ModelUtil.getDefaultGeometryKernel()` returns the default geometry kernel in new models. Valid values are `comsol` for the COMSOL kernel and `cadps` for the CAD kernel (Parasolid kernel). The CAD kernel requires the CAD Import Module.

`ModelUtil.setOpenGeometryKernel(<openkernel>)` specifies the geometry kernel to use when opening models. Valid values are `model` for the geometry kernel used by the model file, and `comsol` to convert the geometry to the COMSOL kernel.

`ModelUtil.setDefaultGeometryKernel(<defaultkernel>)` specifies the default geometry kernel to use in new models with the string <defaultkernel>. Valid values are `comsol` for the COMSOL kernel and `cadps` for the CAD kernel (Parasolid kernel). The CAD kernel requires the CAD Import Module.

`ModelUtil.getEntityPath(<entity>,<divider>)` creates the path from the root of the model object for the given entity <entity>. <divider> specifies a divider to use between entities in path.

`ModelUtil.load(<tag>,<filename>)` loads a model from a file <filename> in the client's file system and names it <tag>. Loading a file from a directory sets the model directory. The model directory is used for saving files if you do not provide an absolute path to the file.

`ModelUtil.loadCopy(<tag>,<filename>)` loads a copy of a model from a file <filename> in the client's file system and names it <tag>. The `loadCopy` method is the same as `load` except that the loaded model is not

associated with the file, so `model.save()` does not work. You have to specify the filename the first time you save it again.

`ModelUtil.loadProtected(<tag>,<filename>,<password>)` works like `ModelUtil.load` but with password protection.

`ModelUtil.loadRecovery(<tag>,<foldername>)` loads a model from a recovery directory or folder structure in the client's file system and names it `<tag>`.

`ModelUtil.showProgress(bool)` turns on or off showing of progress in a window or on a file when running lengthy tasks when connected to a server. The return value is a Boolean value that is true if showing progress is possible.

`ModelUtil.showProgress(<filename>)` turns on logging of progress to the file `<filename>` in the client's file system. If `<filename>` is `null` progress is logged to the standard output.

`ModelUtil.connect(<host>,<port>)` connects to a COMSOL server. The arguments `<host>` and `<port>` provide host name and port number for the COMSOL server.

`ModelUtil.initStandalone(bool)` Initializes the environment for using the COMSOL API from a standalone Java[®] application. You should *not* use this command from the LiveLink[™] for MATLAB[®]. Set the argument to true if support for plotting in a GUI using Java Swing widgets should be available.

`ModelUtil.initStandalone(bool,<guiToolkit>)` allows to specify that support for using a given Java GUI toolkit should be available. The optional `<guiToolkit>` parameter can have the values "swing" or "swt" telling that Swing widgets or widgets from the Standard Widget Toolkit (SWT) can be used.

`ModelUtil.getPreference(<prefsName>)` returns the value of a preference.

`ModelUtil.setPreference(<prefsName>,<value>)` sets the value of a preference.

`ModelUtil.listPreferences()` returns a string with a listing of the preferences names and their descriptions.

`ModelUtil.loadPreferences()` loads the preferences from file. Use this in standalone Java application which do not load the preferences at launch time.

`ModelUtil.savePreferences()` saves the preferences to file.

`ModelUtil.uniquetag()` returns a unique model tag that is not in use.

Client/Server Commands

`ModelUtil` provides functionality to control COMSOL client/server options. You can connect/disconnect and control connections from multiple clients to a server.

`ModelUtil.connect()` connects to a COMSOL server. The COMSOL command arguments `-Dcs.host=<host>` and `-Dcs.port=<port>` can provide host name and port number. In case those are not provided, and the both client and server access the same file system, the host and port can be automatically transferred.

`ModelUtil.disconnect()` disconnects from a COMSOL server.

`ModelUtil.setServerBusyHandler(<ServerBusyHandler>)` Several clients can be connected to the same server, but only one client at a time can ask the server to perform an operation. The default behavior when attempting to call the server when it is busy is to throw an exception. By registering a `ServerBusyHandler` it is possible to make the client wait until the server is free again, and to set a time-out for how long to wait.

`ServerBusyHandler` is a class which controls how long a client waits for a busy server to become free again. It also has hooks to perform any action on the client side before starting to wait and just after stopping to wait. Implement a subclass in Java to `ServerBusyHandler` to change the default behavior.

`ServerBusyHandler()` creates a server busy handler which waits for the server to be free without any time-out.

`ServerBusyHandler(<timeOut>)` creates a server busy handler which waits for the server to be free.

`postWaitForServer(<boolean>)` is a hook to perform any action right after waiting for the server to become free.

`preWaitForServer(<host>, <port>)` is a hook to perform any action before beginning to wait for the server to become free.

`ModelUtil.setModelChangedHandler(<ModelChangedHandler>)` registers a handler of changes to models made by other clients. If any other client changes any model in use by this client, the model change handler is notified.

`ModelChangedHandler` is an interface to handle updates of the client when another client has modified any models in use by this client. Implement the interface in Java to change the default behavior.

`handleModelChangeOnServer(<modelChangeInfo>)` is called when another client has changed any models in use by this client.

`ModelChangeInfo` is a class with information about changes to models by other clients.

`getModelTags()` returns the tags of the models which have changed.

model

Model object methods that set up basic parts of a model object such as the model history and saving model files.

SYNTAX

`model` is a model object that you can create, for example, using `ModelUtil.create(<tag>)`.

`model.baseSystem(<system>)` sets the unit system for the entire model to the given system. The default is the SI system, which has the tag `SI`. Other supported unit systems are `bft` (British engineering units), `cgs`, `mpa`, `emu`, `esu`, `fps`, `ips`, and `psi`.

`model.dateModified()` returns the modification date of the model.

`model.disableUpdates()` returns the current status of the disable state for the model object.

`model.disableUpdates(boolean)` Temporarily disables and re-enables the update of variables in entities that automatically generates other entities (for example, `physics` or `coordSystem`). Disable updates to speed up the evaluation of long execution sequences. Leaving this flag disabled can cause strange side effects during modeling. For example, some parameter values in a feature of a physics interface might not be valid until an update has been made. The model inputs are such parameters, which end with the suffix `_src`. Trying to set a value to any of these parameters with updates disabled might give an error message. Other effects are that the generated variables are unknown to the unit evaluator and equation view readings can be incomplete. When the disabled state goes from `true` to `false`, the program performs a full update of the variables, so the model is in a fully functional state.

`model.fontFamily(<family>)` sets the font family to be used in plots. The font `default` is always available. If using Windows, most system fonts can also be used.

`model.fontSize(<size>)` sets the font size to be used in plots.

`model.getComsolVersion()` returns the COMSOL version used to save the model or the current version if the model has never been saved.

`model.getFilePath()` returns the absolute path of the model or an empty string if the model has not been saved.

`model.getLastComputationTime(string time_format)` returns the last computation time for the model or application as a string. Use `model.getLastComputationTime("ms")` to get the time measured in ms, which you

can then use as an input to `model.setLastComputationTime`. Other supported time formats are "hr:min:sec", "h:min:s", and "detailed", which returns the time in seconds and also includes more readable units for longer times.

`model.getUsedProducts()` returns the products that this model uses.

`model.hist().complete(bool)` enables or disables history logging for methods where the arguments typically are very large objects.

`model.hist().isComplete()` returns true if history logging is enabled for methods where the arguments typically are very large objects.

`model.hist().disable()` Disables logging of top-level API calls to the history. Use this method sparingly; the normal state is that the history is logged.

`model.hist().enable()` Removes the most recent disabling of top-level API calls to the history. Calling `enable()` can be viewed as removing an entry from a stack of disable records; logging only occurs if the stack is empty.

`model.lastModifiedBy()` returns the last user to modify the model.

`model.modelPath(<path>)` sets the model path. The model path is used for reading files required by the model if no path is provided to the file. <path> is a list of directories separated by semicolon. When reading an external file, COMSOL attempts to find a file in the following locations:

- 1 The absolute path as given in the filename. If the path given in the filename is relative, it is resolved relative to the current user directory.
- 2 The model directory, if provided.
- 3 The directories defined by `modelPath` model, if any.
- 4 The directories in the `cs.path` setting (ordered and semicolon separated).
- 5 The current directory.

The model directory is used for saving and exporting files if you do not provide an absolute path to the file.

`model.modelPath()` returns the model path.

`model.resetHist()` rebuilds the model from scratch to generate a compacted model's Java or M-file history. If the model has errors, or has invalid property values, the method fails and the old history is kept.

`model.save(<filename>)` saves the model as a multiphysics model file in <filename>. The model directory is used if no path is provided.

`model.save(<filename>, <type>)` saves the multiphysics model in <filename>. If type is `java`, a model file for use with Java[®] is saved. If type is `m`, this command saves a model file as an M-file for use with the LiveLink[™] for MATLAB[®].

`model.setLastComputationTime(long time)` sets the last computation time for the model or application as the measured computation time (in ms) that you provide as the input.

SEE ALSO

[model.modelNode\(\)](#), [model.unitSystem\(\)](#)

model.attr()

Model entity list methods such as copying, duplicating, clearing, and removing model entities.

SYNTAX

`model.attr()` returns a *model entity list*. The string *attr* denotes a method name for accessing the model entity list.

`model.attr().clear()` removes all tagged model entities.

`model.attr().copy(<tag>, <copytag>)` creates a new model entity with the tag <tag>, which is a copy of the model entity with the tag <copytag>. The <copytag> should be combination of tags separated by slashes to uniquely identify the entity. For example, `pg1/surf1/htgh1` identifies

`model.result("pg1").feature("surf1").feature("htgh1")`. How to interpret the combined tag depends on the context. The difference between `duplicate` and `copy` is that `copy` can use a source anywhere in the model, whereas `duplicate` requires that the source is in the same list. Not all model entities support the `copy` operation. The difference between `copy` and `copyTo` is that `copyTo` copies the entity to a specific position in the list, whereas `copy` copies to a default position in the list. Not all model entities support the `copyTo` operation.

`model.attr().copy(<tag>, <copytag>, <modeltag>)` creates a copy and assigns it to the model <modeltag>.

`model.attr().copyTo(<tag>, <copytag>, <insertafter>)` creates a copy and inserts it in the list after the entity with tag <insertafter>. If <insertafter> is an empty string, the entity is inserted first in the list. Not all model entities support the `copyTo` operation.

`model.attr().duplicate(<tag>, <copytag>)` creates a new model entity with the tag <tag> which is a duplicate of the model entity with tag <copytag>. Not all model entities support the `duplicate` operation.

`model.attr().duplicateTo(<tag>, <copytag>, <insertafter>)` creates a new model entity and inserts it in the list after the entity with tag <insertafter>. If <insertafter> is an empty string, the entity is inserted first in the list. Not all model entities support the `duplicateTo` operation.

`model.attr().get(<tag>)` returns the entity with tag <tag> from the entity list `model.attr()`.

`model.attr().remove(<tag>)` removes the model entity with tag <tag>.

`model.attr().size()` returns the number of model entities.

`model.attr().tags()` returns a string array with the tags of all model entities.

`model.attr().uniqueTag(<tag>)` returns a unique tag in the list context.

SEE ALSO

[model](#)

model.attr(<tag>)

Model entity methods for adding and accessing name, tag, version, comments, date created, and the author information for a model entity.

SYNTAX

`model.attr(<tag>)` returns a *model entity* with tag <tag>. The string *attr* denotes a method name for accessing a model entity with tag <tag>.

`model.attr(<tag>).active(bool)` makes the entity with tag <tag> active or inactive.

`model.attr(<tag>).author()` returns the author of the entity.

`model.attr(<tag>).author(<author>)` sets the author of the entity.

`model.attr(<tag>).comments()` returns the comments of the entity.

`model.attr(<tag>).comments(<comments>)` sets the comments of the entity.

`model.attr(<tag>).dateCreated()` returns the creation date of the entity.

`model.attr(<tag>).isActive()` returns true if the entity with tag <tag> is active.

`model.attr(<tag>).label()` returns the label of the entity.

`model.attr(<tag>).label(<label>)` sets the label of the model entity. The label is an arbitrary nonempty string.

`model.attr(<tag>).resetAuthor(<author>)` sets the author of the entity and all its children. In particular, when used on the model itself, the method sets the author on all model entities of the model.

`model.attr(<tag>).tag()` returns the tag of the entity.

`model.attr(<tag>).tag(<newtag>)` assigns the new tag <newtag> to the entity <tag>.

`model.attr(<tag>).version(<version>)` sets the version of the entity. The version is a user-defined string.

`model.attr(<tag>).version()` returns the version of the entity.

`model.attr(<tag>).help()` and `model.attr(<tag>).help(string)` return a string pointer to locally installed HTML documentation. *string* is the name of a type within the model object.

SEE ALSO

[model](#)

model.batch()

Create batch jobs.

SYNTAX

Jobs

`model.batch().create(<tag>, jobtype)`; creates a batch job tagged <tag> of type *jobtype*, where *jobtype* is Parametric, Batch, or Cluster.

`model.batch().remove(<tag>)` removes a batch job.

`model.batch().size()` returns number of batch jobs.

`model.batch().tags()` returns the tags of the batch jobs.

`model.batch(<tag>).attach(<stag>)` attaches a batch job with tag <tag> to a study with tag <stag>, which includes *assigning* it to the study.

`model.batch(<tag>).create(<jtag>, <oper>)` creates a batch job sequence.

`model.batch(<tag>).detach(<stag>)` detaches a batch job from a study with tag <stag>.

`model.batch(<tag>).remove(<ttag>)` removes the task.

`model.batch(<tag>).run()` runs the batch job.

`model.batch(<tag>).set(jprop, <jvalue>)` sets the property *jprop* to the value <jvalue>.

`model.batch(<tag>).study(<stag>)` assigns a batch job to a study tag <stag>.

`model.batch(<tag>).study()` returns the study tag of batch job with tag <tag>.

`model.batch(<tag>).feature(<ttag>)`.

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

Batch Job Properties

The Parametric job type has the following properties:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
accumtable	String	new	Accumulated probe table
accumtableall	on off	off	Use all probes for the accumulated probe table
control	String	user	Controlling study
param	String array		Name of parameter and its value (output)
pdistrib	on off	off	Distributed (in parallel) the parameter values
pname	String array		Parameter name(s) to vary
plist	String array		Parameter values
plot	on off	off	Update a plot group while solving
plotgroup	String	default	Update this plot group while solving
pwork	int	1	Limit for the number of work groups
pworkactive	on off	off	Use a limit for the number of work groups
stopcond	String		A stop condition expression
err	on off	off	Stop sweep if error
error	String array		The logged error
useaccumtable	on off	off	Produce an accumulated probe table while solving

The Optimization job type sets its property through the Optimization study node, which has the following properties:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
optobj	String		Objective function to be optimized
descr	String		Description of optimization objective function
objectivetype	minimization maximization	minimization	Sets whether the objective should be minimized or maximized
objectivesolution	auto first last sum min max	auto	Determines how the objective should be evaluated for studies with more than one available PDE solution, for example time-dependent problems.
pname	String array		Names of control parameters
initval	String array		Initial values for control parameters
lbound	String array		Lower bounds on control parameters
ubound	String array		Upper bounds on control parameters
optsolver	coordsearch montecarlo neldermead bobyqa	neldermead	Optimization solver
useseed	on off	off	Use random seed for Monte Carlo solver
randseed	int	0	Random seed for Monte Carlo solver
nsolvemax	int	1000	Maximum number of objective evaluations
opttol	double	1e-2	Optimization tolerance
useobjtable	on off	off	Produce a table with all objective evaluations
objtable	String	new	Reference to table with objective evaluations
convinfo	off on detailed	on	Detail of log messages from optimization solver

The Batch job type has the following properties:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
control	String	user	Name of controlling study
np	integer	auto	Number of cores to use
graphics	on off	off	Enable graphics
maxallow	integer	1	Maximum allowed number of batch jobs to start simultaneously
maxrestart	integer	0	Maximum number of restarts before a batch job is failed
maxalive	integer	300	Maximum number of seconds before the batch job must say it is running
starttime	now 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23	now	The time, as an hour for a 24-hour clock, when the batch job should start
batchdir	String	home directory	The directory to store files used by the batch job
client	on off	off	Run the batch job as client
port	integer	2036	The host port number
host	String	localhost	Name of host
batchfile	String	batchmode1.mph	Name of batch model file
clear	on off	on	Clear the previous model file
clearmesh	on off	off	Clear meshes before saving model
clearsolution	on off	off	Clear solutions before saving model
savefile	on off	on	Save model after run
specbatchdir	on off	off	Specify different directory for batch process than used by the current process
rundir	String	home directory	The directory used by the batch job when specbatchdir is on
speccomsoldir	on off	off	Specify different directory for the COMSOL installation than used by the current process
comsoldir	String	COMSOL installation directory	The COMSOL installation directory used by the batch job when speccomsoldir is on
synchsolutions	on off	off	Synchronize solutions after batch job finishes
synchaccumprobetable	on off	off	Synchronize accumulated probe tables after batch job finishes
probesel	all none manual	all	The probes to compute
probes	String array		Probes to compute
useaccumtable	on off	off	Use the accumulated probe table
accumtable	String	new	Name of table to use
accumtableall	on off	on	Use all probes
client	on off	off	Run as client

PROPERTY	VALUE	DEFAULT	DESCRIPTION
host	String	localhost	Name of server
port	integer		Server port number

The Cluster job type has the following properties:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
batch	String		Tag of batch job to run
clustertype	general whpc2008 wccs2003 sge slurm none	general	The type of cluster job
control	String	user	Name of controlling study
corespernode	integer	0	Minimum number of cores per node on whpc2008
exclusive	on off	on	Demand exclusive right to nodes on wccs2003, whpc2008, and SLURM
filetransfercmd	none scp user	none	Command to transfer files
filetransferfromusercmd	String		Command to transfer files from remote location
filetransfertousercmd	String		Command to transfer files to remote location
hostfile	String		Path to hostfile
memorypernode	integer	0	Minimum amount of memory per node on whpc2008 and SLURM
mpd	on off	off	If an mpd is running on the computer or not
mpiargs	String		Additional MPI arguments
mpibootstrap	String		Name of bootstrap server
mpirsh	String		Path to rsh or ssh
nn	integer	1	Number of processes to start
nodegran	node socket core	node	Node granularity on whpc2008
nodegroup	String		Name of Named selection of Compute nodes in whpc2008
perhost	integer	1	Number of processes / host
postcmd	String		DOS/Linux command to execute after the batch job finished
precmd	String		DOS/Linux command to execute prior to the batch job
priority	Highest AboveNormal Normal BelowNormal Lowest	Normal	Priority of job on wccs2003 and whpc2008
remote	on off	off	Run on remote server
remotecmd	none ssh user	none	Command to use when invoking a command on a remote server
remotehosts	String		List of remote hostnames
remoteos	native windows linux	native	OS used on remote hosts
remoteusercmd	String		Command to run on remote server

PROPERTY	VALUE	DEFAULT	DESCRIPTION
reqnodes	String array		Requested nodes on wccs2003, whpc2008, and SLURM
runtime	DD:HH:MM Infinite	Infinite	Maximum time to run before stopping on wccs2003, whpc2008, and SLURM
schedargs	String		Additional scheduler arguments
scheduler	String	localhost	Name of the scheduler on wccs2003, whpc2008, and SLURM
scpargs	String		Additional SCP arguments
scpcmd	scp putty user	scp	SCP command
scpkey	String		SCP key file
scppath	String		Directory where SCP resides
scpuser	String		Username used by SCP
scpusercmd	String		Command for copying files to remote location
sgegran	host slot manual	host	Node granularity on SGE
sgenn	integer	1	Number of slots in SGE
sgepriority	integer	0	Priority of job on SGE and SLURM
sgequeue	String		Name of SGE and SLURM queue
sshargs	String		Additional SSH arguments
sshcmd	ssh putty user	ssh	SSH command
sshkey	String		SSH key file
sshpath	String		Directory where SSH resides
sshporthost	String		Port host
sshports	String		Ports that should be forwarded by SSH
sshuser	String		Username used by SSH
sshusercmd	String		User-defined SSH command
user	String		User name on wccs2003, whpc2008, and SLURM

Tasks

`model.batch(<tag>).create(<ttag>,tasktype)` ; creates a task of type *tasktype* tagged *<ttag>*. Find options for *tasktype* in [Table 2-3](#) below.

TABLE 2-3: BATCH TASK TYPE OPTIONS

TASK TYPE	DESCRIPTION
Geomseq	A geometry sequence to build
Meshseq	A mesh sequence to build
Solutionseq	A solver sequence to compute
Jobseq	A job sequence to run
Postseq	A post sequence to run
Evalnumericalseq	A numerical results seq (derived value) to run (replaced Numericalseq)
Numericalseq	A numerical results seq to run (deprecated)
Exportseq	An export sequence to run
Save	Saves the state of the model at this point in the job sequence

TABLE 2-3: BATCH TASK TYPE OPTIONS

TASK TYPE	DESCRIPTION
Class	Runs the main function of a compiled class with the system property <code>cs.currentmodel</code> set to the name of the model calling the class
Data	Created by batch jobs to store external process information

Task Type Properties

`model.batch(<tag>).feature(<ttag>).set(ttprop,<tpvalue>)` sets the task type property `ttprop` to the value `<tpvalue>`.

Task type properties can have the values listed in [Table 2-4](#).

TABLE 2-4: TASK TYPE PROPERTY VALUES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
clear	on off	on	Clear the currently stored data
filename	String		Name of file to store or open
openfile	String array	none	Name of file that was saved
param	String array		Name of parameter and its value
files	String array		Name of files for each parameter
input	String array		Input to class file
seq	String	all	Name of sequence to run
num	String array		Name of numerical result feature that generated value
paramvalue	String array		Computed numerical result
store	on off	off	Copy solution
psol	String	none	Tag of solver sequence where solutions are stored

The Data Task Type

The Data task type contains child nodes with process information of type `Process`; see [Table 2-5](#).

TABLE 2-5: DATA CHILD NODES

TASKTYPE	DESCRIPTION
Process	Contains information about running processes

`model.batch(<tag>).feature(<ttag>).feature(<ptag>).set(ptype,<pvalue>)` sets the property `ptype` to the value `<pvalue>`. `ptype` can have the values listed in [Table 2-6](#)

TABLE 2-6: PTYPE PROPERTY VALUES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
cmd	String		The command that started the external process
filename	String		Name of file where model is stored
operation	update progress cancel stop clear rerun	update	Name of operation to perform on the process
status	String		Current status of the process

EXAMPLE

Create a parametric sweep over a geometry sequence that creates a batch job that runs a parametric sweep that runs a solver. [[use intermediate variables to simplify examples below]]

Code for use with Java

```
model.batch().create("sweep1","Parametric");
model.batch("sweep1").set("pname","a");
model.batch("sweep1").set("plist",new double[] {1,2});
model.batch("sweep1").create("sol","Solutionseq");
```



```

model.batch("sweep1").feature("sol1").set("seq", "sol3");
model.batch().create("batch1", "Batch");
model.batch("batch1").create("task", "Jobseq");
model.batch("batch1").feature("task").set("seq", "sweep1");
model.batch().create("sweep2", "Parametric");
model.batch("sweep2").set("pname", "b");
model.batch("sweep2").set("plist", new double[]{1,2,3});
model.batch("sweep2").create("gtask", "Geomseq");
model.batch("sweep2").feature("gtask").set("seq", "geom1");
model.batch("sweep2").create("task", "Jobseq");
model.batch("sweep2").feature("task").set("seq", "batch1");
model.batch("sweep2").run();

```

Determine the parameter names and values from a parametric sweep that has already been run.

```
model.batch(pname).feature(fname).getString("psol")
```

where `pname` is the name of the parametric sweep feature that ran and `fname` is the name of the solution feature that stored the solutions. Use

```
model.sol(sname).feature().tags()
```

to find out the tags of the stored solutions. Use

```
model.sol(sname).feature(fname).getString("sol")
```

to find the solver sequence for a parameter. Use

```
model.sol(sname).getParamNames()
```

and

```
model.sol(sname).getParamVals()
```

Code for use with MATLAB

```

model.batch.create('sweep1', 'Parametric');
model.batch('sweep1').set('pname', 'a');
model.batch('sweep1').set('plist', [1,2]);
model.batch('sweep1').create('sol', 'Solutionseq');
model.batch('sweep1').feature('sol').set('seq', 'sol3');
model.batch.create('batch1', 'Batch');
model.batch('batch1').create('task', 'Jobseq');
model.batch('batch1').feature('task').set('seq', 'sweep1');
model.batch.create('sweep2', 'Parametric');
model.batch('sweep2').set('pname', 'b');
model.batch('sweep2').set('plist', [1,2,3]);
model.batch('sweep2').create('gtask', 'Geomseq');
model.batch('sweep2').feature('gtask').set('seq', 'geom1');
model.batch('sweep2').create('task', 'Jobseq');
model.batch('sweep2').feature('task').set('seq', 'batch1');
model.batch('sweep2').run;

```

Determine the parameter names and values from a parametric sweep that has already been run.

```
model.batch(pname).feature(fname).getString('psol')
```

where `pname` is the name of the parametric sweep feature that ran and `fname` is the name of the solution feature that stored the solutions. Use

```
model.sol(sname).feature().tags
```

to find out the tags of the stored solutions. Use

```
model.sol(sname).feature(fname).getString('sol')
```

to find the solver sequence for a parameter. Use

```
model.sol(sname).getParamNames
```

and

```
model.sol(sname).getParamVals
```

SEE ALSO

[model.sol\(\)](#), [model.study\(\)](#)

model.bem()

Create a boundary element (BEM) model.

SYNTAX

```
model.bem().create(<tag>, "CoefficientPDE");  
model.bem(<tag>).set(<prop>, <value>);  
model.bem(<tag>).selection();
```

`model.bem(<tag>).selection()`; defines the selection for single-sided BEM boundaries. In addition, the following variants are available for double-sided boundaries:

- Use `model.bem(<tag>).selection("cont")`; for a selection of double-sided boundaries where the field is continuous.
- Use `model.bem(<tag>).selection("discont")`; for a selection of double-sided boundaries where the field is allowed to be discontinuous.
- Use `model.bem(<tag>).selection("edge")`; for a selection of BEM edges in 3D.

For a complete list of methods available under `selection()`, see [Selections](#).

The following general properties are available for `model.bem`:

TABLE 2-7: GENERAL PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
background	Expression	0	Background field.
edgefluxname	String		Name of edge flux variable.
edgegradname	String[]		Names of edge gradient variables.
edgeradius	Expression		Radius of cylinders represented as edges.
fluxname	String		Name of boundary flux variable.
infval	Expression	0	Value at infinity (for Laplace equation).
normal	String[]		Boundary normals pointing out of BEM domain.
opname	String		Name of postprocessing operator.
varname	String		Name of field variable.
varnameback	String		Name of field variable on backside of double-sided boundaries.
varnamefront	String		Name of field variable on frontside of double-sided boundaries.

In addition, the following properties for the coefficient of the equation are available:

TABLE 2-8: EQUATION COEFFICIENT PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
a	String	0	Absorption coefficient.
a1	String[]	{0,0,0}	Conservative flux convection coefficient.
be	String[]	{0,0,0}	Convection coefficient.
c	String	1	Diffusion coefficient.

TABLE 2-8: EQUATION COEFFICIENT PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
cedge	String	1	Diffusion coefficient in cylinders represented as edges.
m	String	0	Condition at infinity for Helmholtz equation.

The following integration order properties are available:.

TABLE 2-9: INTEGRATION ORDER PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intorderclose	String		Integration rule for close non-adjacent pairs of mesh elements.
intorderedge	String		Integration rule for pairs of mesh elements with a common edge (3D only).
intorderfar	String		Integration rule for distant pairs of mesh elements.
intordersame	String		Integration rule for pairs of mesh elements that coincide.
intordervertex	String		Integration rule for pairs of mesh elements with a common vertex.
intorderweak	String		Integration for weak equations.

The following symmetry properties are available:.

TABLE 2-10: SYMMETRY PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sym1	off scp user	off	Use of symmetry plane orthogonal to x-axis.
sym1plane	Expression	0	Position of symmetry plane orthogonal to x-axis.
sym2	off scp user	off	Use of symmetry plane orthogonal to y-axis.
sym2plane	Expression	0	Position of symmetry plane orthogonal to y-axis.
sym3	off scp user	off	Use of symmetry plane orthogonal to z-axis.
sym3plane	Expression	0	Position of symmetry plane orthogonal to z-axis.

Finally, the following far-field approximation properties are available:.

TABLE 2-11: FAR-FIELD APPROXIMATION PROPERTIES FOR BEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
dampingparameter	Double	1	Damping parameter.
farfieldapprox	none ACA ACA+	none	Far-field approximation method.
farfieldboxsplitlimit	Integer	20	Number of mesh elements at which box splitting stops.
farfieldmindist	Double	0.5	Minimum relative distance of boxes using far-field approximation.
farfieldsvd	on off	on	Use SVD compression in far-field approximation.
farfieldtol	Double	1e-3	Tolerance used in far-field approximation.
usedamping	on off	off	Use damping parameter with iterative solver.

EXAMPLE

The following example shows how to solve for a Helmholtz equation with outgoing waves at infinity in the exterior of a geometry. The example requires that the dependent variables `u` and `bemflux` already exist on the exterior boundaries.

Code for use with Java

```
model.intRule().create("ir1", "material1");
```

```

model.intRule("ir1").create("o1").order(4);

model.bem().create("bem1", "CoefficientPDE");
model.bem("bem1").selection().geom("geom1", 2).set(<list of exterior boundary numbers>);
model.bem("bem1").set("varname", "u");
model.bem("bem1").set("fluxname", "bemflux");
model.bem("bem1").set("normal", new String[]{"-nx", "-ny", "-nz"});
model.bem("bem1").set("a", "-1");
model.bem("bem1").set("m", "-1");
model.bem("bem1").set("opname", "bemop");
model.bem("bem1").set("intorderfar", "ir1");
model.bem("bem1").set("intorderclose", "ir1");
model.bem("bem1").set("intordersame", "ir1");
model.bem("bem1").set("intorderedge", "ir1");
model.bem("bem1").set("intordervertex", "ir1");
model.bem("bem1").set("intorderweak", "ir1");

```

Code for use with MATLAB

```

model.intRule().create('ir1', 'material1');
model.intRule('ir1').create('o1').order(4);

model.bem.create('bem1', 'CoefficientPDE');
model.bem('bem1').selection.geom('geom1', 2).set(<list of exterior boundary numbers>);
model.bem('bem1').set('varname', 'u');
model.bem('bem1').set('fluxname', 'bemflux');
model.bem('bem1').set('normal', {'-nx', '-ny', '-nz'});
model.bem('bem1').set('a', '-1');
model.bem('bem1').set('m', '-1');
model.bem('bem1').set('opname', 'bemop');
model.bem('bem1').set('intorderfar', 'ir1');
model.bem('bem1').set('intorderclose', 'ir1');
model.bem('bem1').set('intordersame', 'ir1');
model.bem('bem1').set('intorderedge', 'ir1');
model.bem('bem1').set('intordervertex', 'ir1');
model.bem('bem1').set('intorderweak', 'ir1');

```

SEE ALSO

[model.coeff\(\)](#)

model.capeopen()

Create constants and functions interfacing to a CAPE-OPEN compliant thermodynamics package.

SYNTAX

Creating a CAPE-OPEN property package feature.

```
model.capeopen().create(<ptag>, "PropertyPackage");
```

Setting and getting properties in a CAPE-OPEN property package feature:

```

model.capeopen().feature(<ptag>).set(<prop>, <value>);
model.capeopen().feature(<ptag>).getString(<prop>);

```

TABLE 2-12: PROPERTIES SUPPORTED BY CAPE-OPEN PROPERTY PACKAGE

NAME	TYPE	DESCRIPTION
manager_id	String	CAPE-OPEN manager ID.
manager_version	String	CAPE-OPEN manager version.
package_id	String	CAPE-OPEN package ID.
package_desc	String	CAPE-OPEN package description. Only for display in GUI.

```
model.capeopen().feature(<ptag>).storePersistenceData();
```

Calling this method after a property package feature has been created, and `manager_id` and `package_id` have been set, stores information about how the CAPE-OPEN package was created in the COMSOL model. If the model is later opened on a computer with the CAPE-OPEN manager installed but without the property package, this information can be used to create the required property package.

```
model.capeopen().feature(<ptag>).create(<ftag>, <type>);
```

Creates a CAPE-OPEN constant, function, or flash calculation feature. Possible types are `CompoundConstant`, `TemperatureDependentProperty`, `PressureDependentProperty`, `OnePhaseProperty`, `TwoPhaseProperty`, and `FlashCalculationProperty`.

```
model.capeopen().feature(<ptag>).feature(<ftag>).set(<prop>, <value>);
model.capeopen().feature(<ptag>).feature(<ftag>).getString(<prop>);
model.capeopen().feature(<ptag>).feature(<ftag>).getStringArray(<prop>);
model.capeopen().feature(<ptag>).feature(<ftag>).getStringMatrix(<prop>);
```

Set and get properties in a CAPE-OPEN constant, function, or flash calculation feature.

TABLE 2-13: PROPERTIES SUPPORTED BY ALL CAPE-OPEN CONSTANT, FUNCTION, AND FLASH CALCULATION FEATURES

NAME	TYPE	DESCRIPTION
prop_basis	String	Basis ("mass" or "mole") for evaluated properties. Only relevant for some properties.

```
model.capeopen().feature(<ptag>).
```

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

CAPE-OPEN Constant Features

CAPE-OPEN constant features are features with the type "CompoundConstant", and are used to define constants that get their value from a thermodynamics package.

TABLE 2-14: PROPERTIES SUPPORTED BY CAPE-OPEN CONSTANT FEATURES

NAME	TYPE	DESCRIPTION
funcname	String	Name of constant; that is, a variable name that can be used in expressions.
compound	String	Identifier for compound in the CAPE-OPEN property package.
property	String	Identifier for property in the CAPE-OPEN property package.
value	String	Numerical value of constant. This value is only displayed in the GUI. When the constant is evaluated in an expression the value is obtained by calling the CAPE-OPEN property package.

CAPE-OPEN Function Features

A CAPE-OPEN function feature defines a function that can be used to evaluate properties that depend, for example, on temperature or pressure. Different types are `TemperatureDependentProperty`, `PressureDependentProperty`, `OnePhaseProperty`, and `TwoPhaseProperty`.

TABLE 2-15: PROPERTIES SUPPORTED BY CAPE-OPEN FUNCTION FEATURES

NAME	TYPE	DESCRIPTION
funcname	String	Name of function.
derivatives	String array	Names of partial derivatives of function.
compounds	String array	Identifiers for compounds in the CAPE-OPEN property package.
comp_basis	String	Basis ("mass" or "mole") for amounts of compounds. Only relevant for functions of type "OnePhaseProperty" or "TwoPhaseProperty."
property	String	Identifier for the property to evaluate in the CAPE-OPEN property package.
phase	String	Identifier for phase in the CAPE-OPEN property package. Only for features of type "OnePhaseProperty."

TABLE 2-15: PROPERTIES SUPPORTED BY CAPE-OPEN FUNCTION FEATURES

NAME	TYPE	DESCRIPTION
phase1	String	Identifier for first phase in the CAPE-OPEN property package. Only for features of type “TwoPhaseProperty.”
phase2	String	Identifier for second phase in the CAPE-OPEN property package. Only for features of type “TwoPhaseProperty.”
args	String matrix	Names, units, and descriptions for the function arguments. Only for display in GUI.
unit	String	Unit of the function. Only for display in GUI.

Functions of type `TemperatureDependentProperty` and `PressureDependentProperty` have a single argument, which is the temperature or pressure, respectively.

Functions of type `OnePhaseProperty` have temperature and pressure as their first two arguments. If there is more than one compound, there are additional arguments for the fraction of each compound.

Functions of type `TwoPhaseProperty` have temperature and pressure as their first two arguments. If there is more than one compound, there are additional arguments for the fraction of each compound in each phase.

CAPE-OPEN Flash Calculation Features

CAPE-OPEN flash calculation features are used as an interface for flash calculations, which take amounts of different compounds and two conditions (for example, temperature and pressure) as input and compute the fraction of each compound that is present in each phase.

TABLE 2-16: PROPERTIES SUPPORTED BY CAPE-OPEN FLASH CALCULATION FEATURES

NAME	TYPE	DESCRIPTION
compounds	String array	Identifiers for compounds in the CAPE-OPEN property package.
cond1	String	First flash condition.
cond2	String	Second flash condition.
temperature	String	Name of function evaluating the temperature (if temperature is not one of the flash conditions).
pressure	String	Name of function evaluating the pressure (if pressure is not one of the flash conditions).
inphase	String	Base name for functions evaluating presence of each phase. Function names for each phase are formed by appending <code>_<phase></code> to the base name.
amounts	String	Base name for functions evaluating amount of each phase. Function names for each phase are formed by appending <code>_<phase></code> to the base name.
composition	String	Base name for functions evaluating fraction of each compound in each phase. Function names are formed by appending <code>_<phase>_<compound></code> to the base name.
soltype	String	Solution type (“undefined”, “normal,” or “retrograde”).
args	String matrix	Names, units, and descriptions for the function arguments. Only for display in GUI.
phases	String array	Names of all phases. Only for display in the GUI. The phases used in the flash calculation are determined by the CAPE-OPEN property package.

Each flash calculation feature defines a number of functions. All of the functions take the values of two flash conditions as their first two arguments, followed by arguments for the total amount of each compound.

SEE ALSO

`model.func()`

`model.coeff()`

Creating equations in the coefficient form. See also `model.shape()`, `model.weak()`.

SYNTAX

```
model.coeff().create(<tag>,<fields>);
model.coeff(<tag>).field(<fields>);
model.coeff(<tag>).field(<pos>,<fields>);
model.coeff(<tag>).intRule(<irlist>);
model.coeff(<tag>).intRule(<pos>,<irule>);
model.coeff(<tag>).create(<ftag>);
model.coeff(<tag>).feature(<ftag>).getAllowedPropertyValues(property);
model.coeff(<tag>).feature(<ftag>).set(ctype,<cvalue>);

model.coeff(<tag>).field();
model.coeff(<tag>).intRule();
model.coeff(<tag>).feature(<ftag>).getType(ctype);

model.coeff(<tag>).hasProperty(String pname);
model.coeff(<tag>).properties();
model.coeff(<tag>).set(String pname, int value);
model.coeff(<tag>).set(String pname, int pos, int value);
model.coeff(<tag>).set(String pname, int pos, int[] value);
model.coeff(<tag>).set(String pname, int pos1, int pos2, int value);
model.coeff(<tag>).setIndex(String name, String value, int index);
model.coeff(<tag>).setIndex(String name, String value, int firstIndex, int secondIndex);
model.coeff(<tag>).setIndex(String name, String[] value, int index);
model.coeff(<tag>).setIndex(String name, double value, int index);
model.coeff(<tag>).setIndex(String name, double value, int firstIndex, int secondIndex);
model.coeff(<tag>).setIndex(String name, double[] value, int index);
model.coeff(<tag>).setIndex(String name, int value, int index);
model.coeff(<tag>).setIndex(String name, String value, int index);
model.coeff(<tag>).setIndex(String name, String value, int index);
model.coeff(<tag>).setIndex(String name, int value, int firstIndex, int secondIndex);
model.coeff(<tag>).setIndex(String name, int[] value, int index);
```

The `set()` methods index/position arguments are 1-based. The `setIndex()` methods index/position arguments are 0-based.

DESCRIPTION

`model.coeff(<tag>)` returns the coefficient form equations with tag `<tag>`.

`model.coeff().create(<tag>,<fields>)` creates coefficient form equations with tag `<tag>` using the fields `<fields>`. The field tags refer to the fields defined by `model.field()`. The shape functions referred to by the fields are internally also used to find the derivatives of the field variables if converting the coefficient features to weak form. By default, all coefficients are designed to be noncontributing to the equation under consideration. For example, `model.coeff().create("foo",new String[]{"u","v"})`.

`model.coeff(<tag>).field(<fields>)` sets the coefficient form field variables. `<fields>` is a string with a field tag or a vector of field tags—for example, `new String[]{"u","v"}`. Reassigning the fields has the side effect that the size of the coefficients change if the number of field variables changes.

`model.coeff(<tag>).field(<pos>,<fields>)` edits the field at position `<pos>` in the field vector `<fields>`.

`model.coeff(<tag>).intRule(<irlist>)` assigns integration rules to the coefficient form equations. The list must have the same length as the number of field variables defined by the fields or have length 1. In the latter case all equations use the same integration rule. The number of field variables is not necessarily the same as the number of strings specified in `model.coeff(<tag>).field()`.

`model.coeff(<tag>).intRule(<pos>,<irule>)` edits the integration rule at position `<pos>` in the vector `<irule>`.

`model.coeff(<tag>).feature(<ftag>)` is a coefficient form feature with tag `<ftag>` in the coefficient form equations with tag `<tag>`.

`model.coeff(<tag>).create(<ftag>)` creates a new coefficient form feature with tag `<ftag>`.

`model.coeff(<tag>).feature(<ftag>).set(ctype,<cvalue>)` sets the value of the coefficient of type `ctype` to `<value>`. All string data types that are listed in [Table 2-2](#) are supported; which argument types are applicable depends on the coefficient. `ctype` is one of `c`, `al`, `ga`, `be`, `a`, `f`, `da`, `ea`, `q`, and `g`. These coefficients are available at all dimensions. In addition at level `edim==sdim-1`, the coefficients `q` and `g` are allowed, corresponding to `a` and `f`, respectively. All coefficients have a default 0 contribution.

`model.coeff(<tag>).feature(<ftag>).selection().named(<seltag>)` assigns the coefficient form equations to the named selection `<seltag>`.

`model.coeff(<tag>).feature(<ftag>).selection().set(...)` defines a local selection that assigns the coefficient form equations to geometric entities. For a complete list of methods available under `selection()`, see [model.selection\(\)](#). Only selections at a single geometry level is allowed in the selection.

`model.coeff(<tag>).feature(<ftag>)).`

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

`model.coeff(<tag>).field()` returns the fields as a string array.

`model.coeff(<tag>).intRule()` returns the integration rule tags as a string array.

`model.coeff(<tag>).feature(<ftag>).getType(ctype)` returns the coefficient value. See the section [getType\(\)](#) for available methods.

`model.coeff(<tag>).feature(<ftag>).selection().named()` returns the named selection tag, and

`model.coeff(<tag>).feature(<ftag>).selection().getType()` returns domain information. See [model.selection\(\)](#) for available methods.

EXAMPLE

Define two uncoupled Poisson-like equations on the domain `dtag`.

Code for use with Java

```
model.coeff().create("c1",new String[]{"u","v"});
model.coeff("c1").intRule(new String[]{"gp1","gp1"});
CoeffFeature f1 = model.coeff("c1").create("f1");
f1.set("c",1,new String[]{"1","0.1","2"});
f1.set("c",2,"3");
f1.set("f",new String[]{"2","1"});
f1.selection().geom("g1",2);
f1.selection().set(1);
```

Code for use with MATLAB

```
model.coeff.create('c1',{'u','v'});
model.coeff('c1').intRule({'gp1','gp1'});
f1 = model.coeff('c1').create('f1');
f1.set('c',1',{'1','0.1','2'});
f1.set('c',2,'3');
f1.set('f',{'2','1'});
f1.selection.geom('g1',2);
f1.selection.set(1);
```

model.constr()

Creating and modifying constraints in a model.

SYNTAX

```
model.constr().create(<tag>,<shtags>);
model.constr().create(<tag>,<nglobal>);
model.constr(<tag>).shape(<shtags>);
model.constr(<tag>).shape(<pos>,<shtags>);
model.constr(<tag>).global(<nglobal>);
model.constr(<tag>).create(<ftag>);
model.constr(<tag>).feature(<ftag>).getAllowedPropertyValues(property);
model.constr(<tag>).feature(<ftag>).set(ctype,<value>);

model.constr(<tag>).shape();
model.constr(<tag>).global();
model.constr(<tag>).feature(<ftag>).getType(ctype);

model.constr(<tag>).hasProperty(String pname);
model.constr(<tag>).properties();
model.constr(<tag>).set(String pname, int value);
model.constr(<tag>).set(String pname, int pos, int value);
model.constr(<tag>).set(String pname, int pos, int[] value);
model.constr(<tag>).set(String pname, int pos1, int pos2, int value);
model.constr(<tag>).setIndex(String name, String value, int index);
model.constr(<tag>).setIndex(String name, String value, int firstIndex, int secondIndex);
model.constr(<tag>).setIndex(String name, String[] value, int index);
model.constr(<tag>).setIndex(String name, double value, int index);
model.constr(<tag>).setIndex(String name, double value, int firstIndex, int secondIndex);
model.constr(<tag>).setIndex(String name, double[] value, int index);
model.constr(<tag>).setIndex(String name, int value, int index);
model.constr(<tag>).setIndex(String name, String value, int index);
model.constr(<tag>).setIndex(String name, String value, int index);
model.constr(<tag>).setIndex(String name, int value, int firstIndex, int secondIndex);
model.constr(<tag>).setIndex(String name, int[] value, int index);
```

The `set()` methods index/position arguments are 1-based. The `setIndex()` methods index/position arguments are 0-based.

DESCRIPTION

`model.constr(<tag>)` returns the constraint with tag `<tag>`.

`model.constr().create(<tag>,<shtags>)` creates a constraint with tag `<tag>` using the shape functions `<shtags>`.

`model.constr().create(<tag>,<nglobal>)` creates a global constraint with tag `<tag>` expecting `<nglobal>` components.

`model.constr(<tag>).shape(<shtags>)` points to the shape functions associated with the constraint. Reassigning the shape functions can have the side effect of modifying the constraints since the number of constraints can change as the size of each constraint vector can change.

`model.constr(<tag>).global(<nglobal>)` specifies that the constraint is global and sets the expected number of components.

`model.constr(<tag>).feature(<ftag>)` is a feature in the constraint with tag `<tag>`.

`model.constr(<tag>).create(<ftag>)` creates a constraint feature.

`model.constr(<tag>).feature(<ftag>).set(ctype,<value>)` sets the parameter `ctype` to `<value>`, where `ctype` is either `constr` or `constrf`, and `<value>` is a single constraint expression or a list of constraint expressions. The number of elements in the constraint expression depends on the number of global constraint components or shape functions specified, and on the shape function type. A Lagrange shape function or global constraint component requires a single item, whereas a vector shape function requires one item for each space dimension. The supported `set` methods are the ones for double string arrays defined in [Table 2-2](#).

`model.constr(<tag>).feature(<ftag>).selection().named(<seltag>)` assigns the constraint to the named selection `<seltag>`.

`model.constr(<tag>).feature(<ftag>).selection().set(...)` defines a local selection that assigns the constraint to geometric entities. For a complete list of methods available under `selection()`, see [model.selection\(\)](#). Only selections at a single geometry level is allowed in the selection.

`model.constr(<tag>).shape()` returns the shape function tags as a string array.

`model.constr(<tag>).global()` returns the number of components if the constraint is global, otherwise `-1`.

`model.constr(<tag>).feature(<ftag>)`.

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

`model.constr(<tag>).feature(<ftag>).getType(ctype)` returns the constraint or constraint force value. For available methods, see [getType\(\)](#).

`model.constr(<tag>).feature(<ftag>).selection().named()` returns the named selection tag, and `model.constr(<tag>).feature(<ftag>).selection().getType()` returns domain information. For available methods, see [Selections](#).

`model.constr(<tag>).feature(<ftag>).selection(<estype>).set(...)` defines a subselection of a given lower-dimensional excluded selection type that should be excluded from the constraint selection. Excluded selection type can be `exclude0` for points, `exclude1` for edges, and `exclude2` for excluded face subselections. The constraint is not enforced on the specified excluded subselections. Excluded subselections have to have lower dimension than the constraint selection. For a complete list of methods available under `selection(<estype>)`, see [model.selection\(\)](#).

EXAMPLES

Code for use with Java

Set several constraint by using multiple constraints:

```
model.constr().create("c1",new String[]{"shu","shv"});
ConstrFeature f = model.constr("c1").create("f1");
f.set("constr",new String[]{"u-1","v"});
f.selection().geom("geom1",1);
f.selection().all();
```

Vector elements need a set of constraints:

```
model.constr().create("c2",new String[]{"shE"});
ConstrFeature f = model.constr("c2").create("f1");
f.set("constr",new String[]{"Ex-1","Ey-0","Ez-0"});
f.selection().geom("geom1",1);
f.selection().all();
```

Code for use with MATLAB

```
model.constr.create('c1',{'shu','shv'});
f = model.constr('c1').create('f1');
f.set('constr',{'u-1','v'});
f.selection.geom('geom1',1);
f.selection.all;
```

Vector elements need a set of constraints:

```
model.constr.create('c2',{'shE'});
f = model.constr('c2').create('f1');
f.set('constr',{'Ex-1','Ey-0','Ez-0'});
f.selection.geom('geom1',1);
f.selection.all;
```

SEE ALSO

[model.shape\(\)](#)

model.coordSystem()

Add coordinate systems, perfectly matched layers, and infinite elements.

SYNTAX

```
model.coordSystem().create(<tag>,<gtag>,type);
model.coordSystem(<tag>).set(property, <value>);
model.coordSystem(<tag>).setIndex(property, <value>, row);
model.coordSystem(<tag>).setIndex(property, <value>, row, col);
model.coordSystem(<tag>).selection();
```

```
model.coordSystem(<tag>).coord()
model.coordSystem(<tag>).isOrthonormal()
model.coordSystem(<tag>).isLinear()
```

DESCRIPTION

`model.coordSystem().create(<tag>,<gtag>,type)` creates a coordinate system with tag `<tag>` on geometry `<gtag>` of type `type`. There are the following types of coordinate systems: mapped system (Mapping), base-vector system (VectorBase), rotated system (Rotated), boundary system (Boundary), scaling system (Scaling), and cylindrical system (Cylindrical). The boundary system only applies to boundaries. In addition, the perfectly matched layers (PMLs) and infinite elements are also implemented as types of coordinate systems: PML and InfiniteElement, respectively.

`model.coordSystem(<tag>).selection().named(<seltag>)` assigns the coordinate system to the named selection `<seltag>`. `model.coordSystem(<tag>).selection().set(...)` defines a local selection that assigns the coordinate system to geometric entities. For a complete list of methods available under `selection()`, see [Selections](#). The selection method is only available for coordinate systems of the following types: Scaling, PML, and InfiniteElement.

`model.coordSystem(<tag>).set("orthonormal", "on")` specifies that this is a orthonormal system. This affects the internal calculation of systems, so some simplifications on expressions can be made. It is recommended to use this option when possible. Boundary systems, rotated systems, and cylindrical system are always orthonormal..

TABLE 2-17: COMMON PROPERTIES FOR COORDINATE SYSTEMS

PROPERTY	VALUE	DESCRIPTION
name	string	Coordinate system name

Mapping

`model.coordSystem().create(<tag1>,<gtag>,"Mapping")` creates a mapped system. In a mapped system you specify the coordinate mapping given in some of the available frame coordinates (usually x, y, z).

TABLE 2-18: PROPERTIES FOR MAPPING SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(x1,x2,x3)]	Coordinate names
map	string array	(x,y,z)	The map
orthonormal	string (on off)	off	If the system is orthonormal
frametype	string (mesh material spatial geometry)	spatial	The frame type

`model.coordSystem(<tag1>).setIndex("map", "x+1", 0)` sets the mapping of the first coordinate system coordinate to be a function of the first frame coordinate, x.

`model.coordSystem(<tag1>).setIndex("map", "y+1", 2)` sets the mapping of the third coordinate system coordinate to be a function of the second frame coordinate *y*.

VectorBase

`model.coordSystem().create(<tag2>, "VectorBase")` creates a base-vector system. In a base-vector system you specify the base vectors given as components of a frame system. If the components are independent of frame coordinates this is a linear system and can be applied for any frame.

TABLE 2-19: PROPERTIES FOR BASE VECTOR SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(x1,x2,x3)]	Coordinate names
base	string matrix	[(1,0,0)(0,1,0)(0,0,1)]	Base vectors
orthonormal	string (on off)	off	If the system is orthonormal or not
outofplane	string	"2" in 2D, "1,2" in 1D	Out-of-plane index

`model.coordSystem(<tag2>).setIndex("base", "1", 0, 1)` sets the first base vector's second component to one. As an alternative, it is possible to specify the full base-vector matrix using the following syntax:

`model.coordSystem(<tag2>).set("base", new String[][]{{"0", "1", "0"}, {"0", "0", "1"}, {"1", "0", "0"}})` sets the base vector matrix so the first base vector is equal to the *y*-axis of the frame system, the second is the *z*-axis, and so on. In 2D, you only use a two rows and two columns from the full base vector matrix for the in-plane base vectors. As an option, it is therefore possible to specify which of the coordinate system base vectors that corresponds to the out-of-plane axis in the frame system. Internally, this base vector always gets the components {"0", "0", "1"}. The third column is also set using these components. To make a general 3D system in 2D, you must use the mapped system.

`model.coordSystem(<tag2>).set("outofplane", "2")` sets the third base vector to represent the out-of-plane vector (*z*-axis in 2D). The value is zero based. In 1D the out-of-plane index is set using the syntax "1,2" to set second and third base vectors to represent the out-of-plane vector.

Rotated

`model.coordSystem().create(<tag3>, "Rotated")` creates a rotated system. In 3D you specify the **Z-X-Z** Euler angles, which corresponds to sequential rotation first about the *z*-axis, then the *x*-axis, and finally the *z*-axis again. In 2D you can only rotate about the out-of-plane axis.

TABLE 2-20: PROPERTIES FOR ROTATED SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(x1,x2,x3)]	Coordinate names
angle	string array	(0,0,0)	Rotation angles
outofplane	string	"2" in 2D, "1,2" in 1D	Out of plane index

`model.coordSystem(<tag3>).setIndex("angle", "12[deg]", 0)` sets the first rotation about the *z*-axis to 12 degrees. In 3D, there are three angles, so the third argument determines which angle that you specify, using 0-based indexing (that is, 0 means the first angle). The default unit for angles are radians.

Boundary

`model.coordSystem().create(<tag4>,<gtag>,"Boundary")` creates a new boundary system, which is a local base vector system on 2D boundaries (\mathbf{t} , \mathbf{n}) and on 3D boundaries (\mathbf{t}_1 , \mathbf{t}_2 , \mathbf{n}). There is always one boundary system added by default for each geometry.

TABLE 2-21: PROPERTIES FOR BOUNDARY SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(x1,x2,x3)]	Coordinate names
frametype	string (mesh material spatial geometry)	spatial	Frame type
reversenormal	string (on off)	off	Reverse normal direction
tangent	string array		Tangent direction
mastersystem	string (manual globalCartesian <tag>)	globalCartesian	Which system to create first tangential direction from
mastercoordsyscomp	string	"2" in axisymmetry, "3" otherwise	Which axis to create first tangential direction from

`model.coordSystem(<tag4>).set("reversenormal","on")` flips the normal direction for this system, so that it is opposite to the normal direction given by the geometry.

`model.coordSystem(<tag4>).set("mastersystcomp","2")` sets the first tangential direction from the second axis of the specified master system.

`model.coordSystem(<tag4>).set("mastersystem","manual")` specifies that no master system is used and that the tangential direction must be entered by the user.

`model.coordSystem(<tag4>).setIndex("tangent","1")` sets the first component of the first tangential direction.

Cylindrical

`model.coordSystem().create(<tag5>,<gtag>,"Cylindrical")` creates a cylindrical coordinate system, which you can use in 2D and 3D where rotational symmetry about the axis is required. You can specify the origin, axis direction and radial base vector.

TABLE 2-22: PROPERTIES FOR CYLINDRICAL SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(r, phi, a)]	Coordinate names
origin	string array	(0,0,0)	Origin of system
axis	string array	(0,0,1)	Axis direction
radialbasevector	string array	(1,0,0)	Radial base vector direction a j=0

`model.coordSystem(<tag5>).set("origin", new String[]{"1","0","0"})` sets the origin to (1,0,0).

Spherical

`model.coordSystem().create(<tag6>,<gtag>,"Spherical")` creates a spherical coordinate system, which you can use in 3D to define a field or property using spherical coordinates. You can specify the origin, zenith axis ($\theta = 0$), and azimuth axis ($\theta = \pi/2$, $\phi = 0$).

TABLE 2-23: PROPERTIES FOR SPHERICAL SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	string matrix	[(r, phi, a)]	Coordinate names
origin	string array	(0,0,0)	Origin of system

TABLE 2-23: PROPERTIES FOR SPHERICAL SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	string array	(0,0,1)	Zenith axis direction
radialbasevector	string array	(1,0,0)	Azimuth axis direction

`model.coordSystem(<tag6>).set("origin", new String[]{"1", "0", "0"})` sets the origin to (1,0,0).

Scaling

`model.coordSystem().create(<tag7>, <gtag>, "Scaling")` creates a scaling coordinate system that maps the geometry, as represented by the independent coordinates of an underlying frame, onto a virtual geometry represented by virtual scaling system coordinates. Physics interfaces that support infinite elements or perfectly matched layers accept the scaling system coordinates as being the physical domain, in which the underlying frame coordinates are seen as a parameterization. Therefore, using a scaling coordinate system you can arbitrarily deform the domain.

TABLE 2-24: PROPERTY FOR SCALING SYSTEM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
map	string array	(x,y,z)	Coordinate mapping

`model.coordSystem(<tag7>).setIndex("map", "y+1", 1)` sets the second coordinate mapping to y+1.

PML

`model.coordSystem().create(<tag8>, <gtag>, "PML")` creates a PML (perfectly matched layer), which acts as an artificial absorbing layer in a domain for a wave simulation. You can specify the type and scaling of the PML using the following properties:

TABLE 2-25: PROPERTIES FOR PML

PROPERTY	VALUE	DEFAULT	DESCRIPTION
PMLfactor	double	1	PML scaling factor.
PMLgamma	double	1	PML scaling curvature factor.
r0	double array	0; 0; 0	Center coordinates for cylindrical PMLs.
raxis	double array	0; 0; 0	Center axis direction for cylindrical PMLs in 3D,
ScalingType	Cartesian Cylindrical Spherical	Cartesian	The PML scaling type. Spherical is only available in 3D.
stretchingType	polynomial rational elliptic logarithmic	polynomial	The coordinate stretching type for the PML scaling.
typicalWavelength	double	1	Typical wavelength for the waves.
wavelengthSourceType	fromPhysics userDefined	fromPhysics	Take the wavelength from the physics or a user-defined wavelength.

InfiniteElement

`model.coordSystem().create(<tag9>,<gtag>,"InfiniteElement")` creates an infinite element, which acts as an unbounded domain for a simulation. You can specify the type and scaling of the infinite element using the following properties:.

TABLE 2-26: PROPERTIES FOR INFINITE ELEMENT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
r0	double array	0; 0; 0	Center coordinates for cylindrical infinite elements.
raxis	double array	0; 0; 0	Center axis direction for cylindrical infinite elements in 3D,
pole	double	dGeomChar	Pole distance
ScalingType	Cartesian Cylindrical Spherical	Cartesian	The scaling type for the infinite element. Spherical is only available in 3D.
width	double	1e3*dGeomChar	Physical width.

EXAMPLE

Create a cylindrical coordinate system with a radial base vector direction that is (0, 1, 0); that is, a coordinate points in the global *y*-direction:

Code for use with Java

```
model.coordSystem().create("sys2", "geom1", "Cylindrical");
model.coordSystem("sys2").setIndex("radialbasevector", "1", 1);
model.coordSystem("sys2").setIndex("radialbasevector", "0", 0);
```

Code for use with MATLAB

```
model.coordSystem.create('sys2', 'geom1', 'Cylindrical');
model.coordSystem('sys2').setIndex('radialbasevector', '1', 1);
model.coordSystem('sys2').setIndex('radialbasevector', '0', 0);
```

model.cpl()

Add component couplings.

SYNTAX

```
model.cpl().create(<tag>, type, <gtag>);
model.cpl(<tag>).set(property, <value>);
model.cpl(<tag>).set("opname", <opname>);
model.cpl(<tag>).selection(property).named(<seltag>);
model.cpl(<tag>).selection(property).set(...);
model.cpl(<tag>).create(<subtag>, subtype);
model.cpl(<tag>).feature(<subtag>).getAllowedPropertyValues(property);
model.cpl(<tag>).feature(<subtag>).set(property, <value>);

model.cpl(<tag>).getType(property, <value>);
model.cpl(<tag>).feature(<subtag>).properties();
model.cpl(<tag>).feature(<subtag>).getType(property, <value>);
model.cpl(<tag>).hasProperty(String pname);

model.cpl(<tag>).set(String pname, int value);
model.cpl(<tag>).set(String pname, int pos, int value);
model.cpl(<tag>).set(String pname, int pos, int[] value);
model.cpl(<tag>).set(String pname, int pos1, int pos2, int value);
model.cpl(<tag>).setIndex(String name, String value, int index);
model.cpl(<tag>).setIndex(String name, String value, int firstIndex, int secondIndex);
model.cpl(<tag>).setIndex(String name, String[] value, int index);
model.cpl(<tag>).setIndex(String name, double value, int index);
model.cpl(<tag>).setIndex(String name, double value, int firstIndex, int secondIndex);
model.cpl(<tag>).setIndex(String name, double[] value, int index);
model.cpl(<tag>).setIndex(String name, int value, int index);
model.cpl(<tag>).setIndex(String name, String value, int index);
model.cpl(<tag>).setIndex(String name, String value, int index);
model.cpl(<tag>).setIndex(String name, int value, int firstIndex, int secondIndex);
model.cpl(<tag>).setIndex(String name, int[] value, int index);
```

The `set()` methods index/position arguments are 1-based. The `setIndex()` methods index/position arguments are 0-based.

DESCRIPTION

`model.cpl().create(<tag>, type, <gtag>)` creates a component coupling of type *type* on the geometry *<gtag>*. The supported types are `GeneralExtrusion`, `LinearExtrusion`, `BoundarySimilarity`, `IdentityMapping`, `GeneralProjection`, `LinearProjection`, `Integration`, `Average`, `Maximum`, and `Minimum`. The component coupling operators provide coupling of values, typically from a source to a destination between or within model components.

`model.cpl(<tag>).selection().named(<seltag>)` assigns the component coupling's source to the named selection *<seltag>*. `model.cpl(<tag>).selection().set(...)` defines a local selection that assigns the component coupling's source to geometric entities. For a complete list of methods available under `selection()`, see [Selections](#).

`model.cpl(<tag>).set(property, <value>)` specifies properties relevant for the selected component coupling type, see below.

`model.cpl(<tag>).set("opname", <opname>)` sets the operator name of the component coupling. The default component coupling operator name is *<tag>*.

`model.cpl(<tag>).selection(property).named(<seltag>)` assigns the component coupling's selection property to the named selection *<seltag>*. `model.cpl(<tag>).selection(property).set(...)` defines a local selection that assigns the component coupling's selection property to geometric entities. For a complete list of methods available under `selection()`, see [model.selection\(\)](#).

`model.cpl(<tag>).create(<subtag>, subtype)` creates a subfeature of type *subtype*. This can only be done when the component coupling *type* is **BoundarySimilarity**. The supported values of *subtype* are **OnePointMap**, **TwoPointMap**, and **EdgeMap**.

`model.cpl(<tag>).selection().named()` returns the named source selection of the coupling.

`model.cpl(<tag>).selection().getType(...)` queries the source selection.

`model.cpl(<tag>).properties()` returns the list of assigned properties as a string array.

`model.cpl(<tag>).getType(property)` returns the value of a specified property.

`model.cpl(<tag>).selection(property).named()` returns the named selection tag of the selection property.

`model.cpl(<tag>).selection(property).getType(...)` queries a selection property.

`model.cpl(<tag>).feature(<subtag>))`.

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

Notation: *srcdim* = dimension of source selection; *srcsdm* = space dimension of source geometry.

EXTRUSION COUPLINGS

An extrusion coupling operator **oper** maps an expression **e** defined on (a part of) the source selection to an expression **oper(e)** that can be evaluated on (a part of) the destination geometries. For each point p_s in the source selection, there can be zero, one or several corresponding points p_d in the destination. The inverse mapping $p_s = m(p_d)$ is always one-to-one. The value of **oper(e)** at the point p_d is defined as the value of **e** at the point p_s .

The inverse mapping m is specified as the composition of a *destination map* m_d and the inverse of a *source map* m_s : $p_s = m(p_d) = m_s^{-1}(m_d(p_d))$. In other words, $m_s(p_s) = m_d(p_d)$ —both the destination map and the source map into the same *intermediate space*. For all operator types except **GeneralExtrusion**, the intermediate space coincides with the source geometry. The source map is always one-to-one. By default, the source map is the identity.

The operator type determines the type of destination map:

TABLE 2-27: EXTRUSION COUPLING TYPES

COUPLING TYPE	DESTINATION MAP
GeneralExtrusion	Nonlinear map described by expressions
LinearExtrusion	Linear map described by vertex mapping
BoundarySimilarity	Similarity transformation described by mapping of boundaries. Also used by copy mesh.
IdentityMapping	Identity map

For most of these coupling types, a source map described by (possibly nonlinear) expressions can be used.

TABLE 2-28: EXTRUSION COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
exttol	double	0.3	Extrapolation tolerance in mesh search
method	usetol closest	usetol	Mesh search method
usenan	on off	off	Use NaN instead of error message when source point is outside selection.

If `method=usetol`, **oper(e)** is defined when the source point p_s is within the source selection, or if it is slightly outside. The tolerance is given in the property `exttol`, which is a distance in mesh element local coordinates; that is, it is a measure relative to the mesh element size. If **oper(e)** is not defined, an error message is given (if `usenan=off`), or the value NaN is returned (if `usenan=on`).

If `method=closest`, a brute force search method is used, which makes `oper(e)` defined everywhere (the nearest point to p_s in the source selection is used).

Depending on the coupling type, additional properties are available (see below).

GeneralExtrusion

A *general extrusion* coupling operator maps an expression defined on a source to an expression that can be evaluated on any destination geometry where the destination map expressions are valid.

TABLE 2-29: GENERAL EXTRUSION MAP PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>dstmap</code>	string array	spatial coordinates	Expressions for destination map $m_d(p_d)$
<code>srcframe</code>	mesh material spatial	spatial	Frame for source mesh
<code>usesrcmap</code>	on off	off	Use source map
<code>srcmap</code>	string array	spatial coordinates	Expressions for source map $m_s(p_s)$

Trailing empty expressions in the properties `dstmap` and `srcmap` are ignored. The remaining expressions must be equal in number, and this determines the dimension `idim` of the intermediate space. Requirement: `srcdim <= idim <= srcsdm`. Changing the source selection has the side effect of changing `dstmap` and `srcmap` so that this requirement is satisfied. By default, `idim=srcsdm`.

The source mesh is viewed in the frame `srcframe`. The source mapping is taken to be linear within each source mesh element.

If `usesrcmap=off`, the `srcmap` property is not used. In this case, `dstmap` is a mapping from the destination to the source (viewed in the frame `srcframe`), and `idim=srcsdm`.

LinearExtrusion

A *linear extrusion* coupling operator linearly maps an expression defined on a source to an expression that can be evaluated in the destination.

TABLE 2-30: LINEAR EXTRUSION MAP PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>srcvertexN</code>	Selection		Source vertex number N
<code>dstgeom</code>	string	source geometry	Destination geometry
<code>dstvertexN</code>	Selection		Destination vertex number N
<code>srcframe</code>	mesh material spatial	spatial	Frame for evaluation of source vertex coordinates
<code>dstframe</code>	mesh material spatial	spatial	Frame for evaluation of destination vertex coordinates
<code>usesrcmap</code>	on off	off	Use source map
<code>srcmap</code>	string[srcsdm]	spatial coordinates	Expressions for source map $m_s(p_s)$

The number of selections `srcvertexN` and `dstvertexN` is 4. These are used only for $1 \leq N \leq \text{dim}+1$, where `dim` is a number less than or equal to $\min(\text{srcsdm}, \text{dstsdm})$. The remaining $4 - \text{dim}$ selections should be empty.

The destination map is the following linear (affine) map from the destination geometry to the source geometry:

- 1 First, if $\text{dim} < \text{dstsdm}$, an orthogonal projection onto the affine space spanned by the destination vertices. The number of destination vertices is $\text{dim}+1$. Thus, $\text{dim}=2$ gives a plane, and $\text{dim}=1$ gives a line.
- 2 Then, a linear (affine) map mapping the destination vertices onto the source vertices.

BoundarySimilarity (3D)

A *boundary similarity* coupling operator maps an expression defined on a part of a boundary to another part of a boundary with the same shape.

TABLE 2-31: BOUNDARY SIMILARITY PROPERTIES IN 3D

PROPERTY	VALUE	DEFAULT	DESCRIPTION
destination	Selection		Destination face
usesrcmap	on off	off	Use source map
srcmap	string[srcsdim]	spatial coordinates	Expressions for source map $m_s(p_s)$.

The destination transformation is a similarity transformation that maps a destination face (*destination*) onto a set of source faces (the source selection). The mesh is always viewed in the mesh frame.

By default, the algorithm automatically chooses a transformation when symmetries make several transformations possible. To control this choice, one of the following subfeatures can be added in 3D.

TABLE 2-32: SUBFEATURE TYPES

SUB FEATURE	REMARKS
EdgeMap	Specify how one source edge is mapped.
OnePointMap	Specify how one source vertex is mapped.
TwoPointMap	Specify how two source vertices are mapped.

EdgeMap

An *edge map* specifies that a certain destination edge should be mapped onto a certain source edge. Their relative direction is given by the property *direction*. The edges must be adjacent to the given faces.

TABLE 2-33: PROPERTIES FOR EDGEMAP SUBFEATURE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
srcedge	Selection		Source edge
dstedge	Selection		Destination edge
direction	auto same opposite	auto	Edge direction

OnePointMap

A *one-point map* specifies that a certain destination vertex should be mapped onto a certain source vertex.

TABLE 2-34: PROPERTIES FOR ONEPOINTMAP SUBFEATURE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
srcpoint1	Selection		Vertex on source face
dstpoint1	Selection		Vertex on destination face

TwoPointMap

A *two-point map* specifies that two destination vertices should be mapped onto two source vertices.

TABLE 2-35: PROPERTIES FOR ONEPOINTMAP SUBFEATURE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
srcpoint1	Selection		Vertex 1 on source face
srcpoint2	Selection		Vertex 2 on source face
dstpoint1	Selection		Vertex 1 on destination face
dstpoint2	Selection		Vertex 2 on destination face

BoundarySimilarity (2D)

TABLE 2-36: BOUNDARY SIMILARITY PROPERTIES IN 2D

PROPERTY	VALUE	DEFAULT	DESCRIPTION
destination	Selection		Destination edge
direction	auto same opposite	auto	Edge direction
usesrcmap	on off	off	Use source map
srcmap	string[srcsdim]	spatial coordinates	Expressions for source map $m_s(p_s)$

The destination transformation is a similarity transformation that maps a destination edge (`destination`) onto a set of source edges (the source selection). Their relative direction is given by the property `direction`. The mesh is always viewed in the mesh frame.

IdentityMapping

An *identity mapping* coupling operator maps between geometric entities that overlap, possibly when viewed in different frames. The destination transformation is an identity mapping between the given frames.

TABLE 2-37: IDENTITY MAPPING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
dstframe	mesh material spatial	spatial	Frame for evaluation of destination coordinates
srcframe	mesh material spatial	spatial	Frame for evaluation of source coordinates

PROJECTION COUPLINGS

A projection coupling operator `oper` maps an expression `e` defined on (a part of) the source selection to an expression `oper(e)` that can be evaluated on (a part of) the destination geometries. It does so by performing integration along curves in the source selection. These curves correspond to lines in an *intermediate space*, whose dimension is equal to `srcdim`. There is a *source map* m_s mapping the source selection into the intermediate space, and a *destination map* m_d mapping the destination geometries into the subspace of intermediate space where the last coordinate is zero. The source map is always one-to-one. The value of `oper(e)` at a destination point p_d is defined as follows:

- 1 In the intermediate space, consider the line that is parallel to the last coordinate axis and goes through the point $m_d(p_d)$.
- 2 Map this line to a curve in the source selection using the inverse of the source map.
- 3 Integrate the expression `e` over this curve.

This implies that the value of `oper(e)` at the destination point p_d is the integral of `e` along a curve through the source point $p_s = m_s^{-1}(m_d(p_d))$.

The coupling type determines the type of the maps:

TABLE 2-38: PROJECTION OPERATOR TYPES

COUPLING TYPE	MAP TYPES
GeneralProjection	Nonlinear map described by expressions
LinearProjection	Linear map described by vertex mapping

TABLE 2-39: PROJECTION COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intorder	int	4	Order of integration formula

Additional properties are available depending on the coupling type, see below.

GeneralProjection

Use a *general projection* component coupling to define integration along curves.

TABLE 2-40: GENERAL PROJECTION COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
dstmap	string[srcdim-1]	spatial coordinates	Expressions for destination map $m_d(p_d)$
srcframe	mesh material spatial	spatial	Frame for source mesh.
srcmap	string[srcdim]	spatial coordinates	Expressions for source map $m_s(p_s)$

LinearProjection

A *linear projection* coupling operator defines a mapping between destination and source that is given by a linear map defined by vertices. Let v be the vector from the first source vertex to the last source vertex. The value of $\text{oper}(\mathbf{e})$ at a point p_d is equal to the integral of \mathbf{e} over the line through the point $p_s = m_s^{-1}(m_d(p_d))$ with direction vector v .

TABLE 2-41: LINEAR PROJECTION COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
srcvertexN	Selection		Source vertex number N
dstgeom	string	source geometry	Destination geometry
dstvertexN	Selection		Destination vertex number N
srcframe	mesh material spatial	spatial	Frame for evaluation of source vertex coordinates
dstframe	mesh material spatial	spatial	Frame for evaluation of destination vertex coordinates

The number of selections `srcvertexN` is 4. These are used only for $1 \leq N \leq \text{srcdim}+1$. The remaining selections should be empty. The number of source vertices is `srcdim+1`. The source map is a linear (affine) map that maps the source vertices onto the points $0, e_1, e_2, \dots, e_{\text{srcdim}}$ in the intermediate space, where e_i is the i th unit vector.

The number of selections `dstvertexN` is 4. These are used only for $1 \leq N \leq \text{srcdim}$. The remaining selections should be empty. The number of destination vertices is `srcdim`. The destination map is the following linear (affine) map from the destination geometry to the intermediate space:

- 1 First, if `srcdim-1 < dstsdm`, an orthogonal projection onto the affine space spanned by the destination vertices. Thus, `srcdim=3` gives a plane, and `srcdim=2` gives a line.
- 2 Then, a linear (affine) map mapping the destination vertices onto the points $0, e_1, e_2, \dots, e_{\text{srcdim}-1}$ in the intermediate space, where e_i is the i th unit vector.

INTEGRATION COUPLINGS

Integration

By default, an *integration* coupling operator oper integrates an expression \mathbf{e} over the source selection. The resulting value $\text{oper}(\mathbf{e})$ can be used anywhere. If `method=summation`, the expression is instead summed over the nodes in the source selection.

TABLE 2-42: INTEGRATION COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intorder	string	4	Integration order
frame	mesh material spatial	spatial	Frame to integrate in (determines volume element)
method	integration summation	integration	Method of computation

Average

An *average* coupling operator `oper` integrates an expression `e` over the source selection and divides with the measure of the source selection. The resulting value `oper(e)` can be used anywhere.

TABLE 2-43: AVERAGE COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intorder	string	4	Integration order
frame	mesh material spatial	spatial	Frame to integrate in (determines volume element)

Maximum/Minimum

A *maximum* or *minimum* coupling operator `oper` finds the maximum or minimum of an expression `e` over the source selection. The resulting value `oper(e)` can be used anywhere. An optional second argument is evaluated at the point where the first argument has its maximum or minimum. Use `x`, `y`, or `z`, for example, to get the coordinate location of the maximum or minimum.

TABLE 2-44: MAXIMUM/MINIMUM COUPLING PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intorder	string	4	Integration rule
points	node integration lagrange	node	Type of point
intorder	string	4	Integration order
lagrange	string	2	Lagrange order

The maximum/minimum is approximated by evaluating the expression in the specified points.

model.elem()

Creating and modifying different types of elements.

SYNTAX

```
model.elem().create(<tag>,eltype);
model.elem(<tag>).set(<ftag>,value);
model.elem(<tag>).field().create(<ftag>,"record");
model.elem(<tag>).field(<ftag>).set(<ftag>,value);
model.elem(<tag>).field().create(<atag>,"array");
model.elem(<tag>).field(<atag>).pos().create("string",value);
model.elem(<tag>).field(<atag>).pos().create("array");
model.elem(<tag>).field(<atag>).pos(pos).create("string",value);
model.elem(<tag>).src().create(<fttag>);
model.elem(<tag>).src(<fttag>).set(<ftag>,value);
model.elem(<tag>).src(<fttag>).field().create(<ftag>,"array");
model.elem(<tag>).geomdim().create(<fttag>);
```

DESCRIPTION

`model.elem().create(<tag>,eltype)` creates a new element of type `eltype`, for example `elinterp`, `elpric`, `elvar`, `elode`, and so on.

`model.elem(<tag>).set(<ftag>,value)` sets the field tagged `<ftag>` to `value`. Examples of fields and values are, `file` and `solution_interp.txt`, `global` and `1`, and so on.

`model.elem(<tag>).field().create(<ftag>,"record")` creates a new field tagged `<ftag>` of type `record` under the element tagged `<tag>`.

`model.elem(<tag>).field(<ftag>).set(sname,value)` sets the field tagged `sname` to `value`. The field is located under the record field tagged `<ftag>`.

`model.elem(<tag>).field(<ftag>).field().create(<rtag>,"record")` creates a new record field tagged `<rtag>` that is a field under the record field `<ftag>`.

`model.elem(<tag>).field().create(<atag>,"array")` creates a new array field tagged `<atag>`.

`model.elem(<tag>).field(<atag>).pos().create("array")` adds a new element of type array to the array tagged `<atag>`.

`model...field(<atag>).pos(1).pos().create("string","1")` adds a new array element of type string with value 1 to the first array element of the array stored in the field `<atag>`.

In the text below, all occurrences of `src` can be replaced with `geomdim`.

`model.elem(<tag>).src().create(<fttag>)` creates a `src` feature tagged `<fttag>` under the element. A feature must have a domain selection.

`model.elem(<tag>).src(<fttag>).selection().dim(2).set(gname)` assigns all domains of dimension 2 from geometry `gname` to the selection of feature `<fttag>`.

`model.elem(<tag>).src(<fttag>).set(<ftag>,value)` sets the field `<ftag>` to `value` under the feature `<fttag>`.

`model.elem(<tag>).src(<fttag>).field().create(rname,"record")` adds a new record field `rname` under the feature `<fttag>`.

EXAMPLES

Specifies an interpolation element that takes its data from a file named `solution_data.txt`.

Code for use with Java

```
model.elem().create("fun1","elinterp");
model.elem("fun1").set("name",new String[]{"sol"});
model.elem("fun1").set("file","solution_data.txt");
model.elem("fun1").set("fileindex",new String[]{"1"});
model.elem("fun1").set("defvars",new String[]{"true"});
model.elem("fun1").set("method",new String[]{"linear"});
model.elem("fun1").set("extmethod",new String[]{"const"});
```

Code for use with MATLAB

```
model.elem.create('fun1','elinterp');
model.elem('fun1').set('name',{'sol'});
model.elem('fun1').set('file','solution_data.txt');
model.elem('fun1').set('fileindex',{'1'});
model.elem('fun1').set('defvars',{'true'});
model.elem('fun1').set('method',{'linear'});
model.elem('fun1').set('extmethod',{'const'});
```

The example below creates two integration couplings.

Code for use with Java

```
model.elem().create("elem1","elcplscalar");
model.elem("elem1").set("var",new String[]{"aa","bb"});
model.elem("elem1").set("global",new String[]{"1","2"});
model.elem("elem1").src().create("feat1");
model.elem("elem1").src("feat1").selection().geom("g",2).set(new int[]{1});
model.elem("elem1").src("feat1").set("expr",new String[][]{{"1"},"2"});
model.elem("elem1").src("feat1").set("ipoints",new String[][]{{"2"},"2"});
model.elem("elem1").src("feat1").set("frame",new String[][]{{"spatial"},"spatial"});
```

Code for use with MATLAB

```
model.elem.create('elem1','elcplscalar');
model.elem('elem1').set('var',{'aa','bb'});
model.elem('elem1').set('global',{'1','2'});
```

```

model.elem('elem1').src.create('feat1');
model.elem('elem1').src('feat1').selection.geom('g',2).set(1);
model.elem('elem1').src('feat1').set('expr',{ '1'},{ '2' });
model.elem('elem1').src('feat1').set('ipoints',{ '2'},{ '2' });
model.elem('elem1').src('feat1').set('frame',{ 'spatial'},{ 'spatial' });

```

This complicated example creates a constr element with two constraints (usually done with constraint features):

Code for use with Java

```

model.elem().create("elem1","elsconstr");
feat = model.elem("elem1").geomdim().create("feat1");
feat.selection().geom("g",2).set(new int[] {1});
feat.set("constr",new String[][]{{"Ex","Ey","Ez"}});
feat.set("cshape",new String[] {"1"})
feat.field().create("shelem","record");
feat.field("shelem").set("case",new String[0]);
feat.field("shelem").set("mind",new String[0]);
feat.field("shelem").field().create("default","array");
feat.field("shelem").field("default").pos().create("array");
feat.field("shelem").field("default").pos(1).pos().create("array");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("string","edg");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("string","shcurl");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("record");
feat.field("shelem").field("default").pos(1).pos(1).pos(3).set("order","2");
feat.field("shelem").field("default").pos(1).pos(1).pos(3)
.set("compnames",new String[] {"Ex","Ey","Ez"});
feat.field("shelem").field("default").pos(1).pos(1).pos(3).set("frame","ref");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("string","edg2");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("string","shcurl");
feat.field("shelem").field("default").pos(1).pos(1).pos().create("record");
feat.field("shelem").field("default").pos(1).pos(1).pos(6).set("order","2");
feat.field("shelem").field("default").pos(1).pos(1).pos(6)
.set("compnames",new String[] {"Ex","Ey","Ez"});
feat.field("shelem").field("default").pos(1).pos(1).pos(6).set("frame","ref");

```

Code for use with MATLAB

```

model.elem.create('elem1','elsconstr');
feat = model.elem('elem1').geomdim().create('feat1');
feat.selection().geom('g',2).set(1);
feat.set('constr',{ 'Ex','Ey','Ez' });
feat.set('cshape',{ '1' })
shelem = feat.field.create('shelem','record');
shelem.set('case','');
shelem.set('mind','');
shelem.field().create('default','array');
shelem.field('default').pos.create('array');
shelem.field('default').pos(1).pos.create('array');
shelem.field('default').pos(1).pos(1).pos.create('string','edg');
shelem.field('default').pos(1).pos(1).pos.create('string','shcurl');
shelem.field('default').pos(1).pos(1).pos.create('record');
shelem.field('default').pos(1).pos(1).pos(3).set('order','2');
shelem.field('default').pos(1).pos(1).pos(3).set('compnames',{ 'Ex','Ey','Ez' });
shelem.field('default').pos(1).pos(1).pos(3).set('frame','ref');
shelem.field('default').pos(1).pos(1).pos.create('string','edg2');
shelem.field('default').pos(1).pos(1).pos.create('string','shcurl');
shelem.field('default').pos(1).pos(1).pos.create('record');
shelem.field('default').pos(1).pos(1).pos(6).set('order','2');
shelem.field('default').pos(1).pos(1).pos(6).set('compnames',{ 'Ex','Ey','Ez' });
shelem.field('default').pos(1).pos(1).pos(6).set('frame','ref');

```

For all records, the statement

```
model...set("frame","ref");
```

is the equivalent to


```
model...field().create("frame","string","ref");
```

The statement

```
model...set("expr",new String[][]{{"1"},"2"}});
```

is equivalent to

```
model...field().create("expr","array");
model...field("expr").pos().create("array").
model...field("expr").pos(1).create("string","1");
model...field("expr").pos().create("array").
model...field("expr").pos(2).create("string","1");
```

so the `set` method is often a much more convenient way to create simple fields.

model.elementSet()

Mesh element sets.

SYNTAX

```
model.elementSet().create(<tag>);
model.elementSet(<tag>).set(<var>,<expr>);
model.elementSet(<tag>).remove(<var>);
model.elementSet(<tag>).model(<mtag>);
```

```
model.elementSet(<tag>).varnames();
model.elementSet(<tag>).get(<var>);
model.elementSet(<tag>).model();
model.elementSet(<tag>).scope();
```

DESCRIPTION

`model.elementSet(<tag>)` returns an element set. It contains one or several definitions of sets of mesh elements. Each element set is identified with an *element set variable* name—the variable evaluates to 1 on mesh elements that belong to the element set, and it evaluates to 0 on other mesh elements. The variable has a defining expression that is evaluated at the midpoint of each mesh element to determine whether the mesh element belongs to the element set. This evaluation is done once at the beginning of the solution process, so the expression must not depend on variables that change during the solution process. All element sets in `model.elementSet(<tag>)` are subsets of the selection `model.elementSet(<tag>).selection()`.

`model.elementSet().create(<tag>)` creates a new element set with tag `<tag>`.

`model.elementSet(<tag>).set(<var>,<expr>)` sets the defining expression for the element set variable `<var>` to `<expr>`.

`model.elementSet(<tag>).remove(<var>)` removes the element set variable `<var>`.

`model.elementSet(<tag>).model(<mtag>)` sets the model node.

`model.elementSet(<tag>).selection().selMethod` manipulates the geometric entity selection, see [Selections](#) for a description of the available methods.

`model.elementSet(<tag>).varnames()` returns all element set variables as a string array.

`model.elementSet(<tag>).get(<var>)` returns the defining expression of element set variable `<var>`.

`model.elementSet(<tag>).model()` returns the model node.

`model.elementSet(<tag>).scope()` returns the fully qualified scope name.

EXAMPLE

Let A be an element set consisting of all mesh triangles that are not adjacent to boundaries 3 or 4 in square, plus all mesh triangles that are adjacent to boundaries 1 or 2. Let the dependent variable u be defined on A . Solve Poisson's equation with Dirichlet conditions on boundaries 1 and 2:

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom().create("geom1", 2);
model.geom("geom1").create("sq1", "Square");
model.mesh().create("mesh1", "geom1");
model.mesh("mesh1").run();
model.elementSet().create("es1");
model.elementSet("es1").set("A", "!bndadj(3,4) || bndadj(1,2)");
model.elementSet("es1").selection().geom(2).all();
model.shape().create("sh1", "material1");
model.shape("sh1").create("f1", "shlag");
model.shape("sh1").feature("f1").set("basename", "u").set("order", 1);
model.shape("sh1").selection().geom(2).all();
model.shape("sh1").elementSet("A");
model.field().create("field1", "u");
model.field("field1").shape(new String[]{"sh1"});
model.intRule().create("ir1", "material1");
model.intRule("ir1").create("o2").order(2);
model.weak().create("weak1");
model.weak("weak1").weak("if(A,ux*test(ux)+uy*test(uy)-test(u),0)");
model.weak("weak1").intRule("ir1");
model.weak("weak1").selection().geom(2).all();
model.constr().create("constr1", new String[]{"sh1"});
model.constr("constr1").create("f1");
model.constr("constr1").feature("f1").set("constr",1,new String[]{"u"});
model.constr("constr1").feature("f1").set("constrf",1,new String[]{"test(u)"});
model.constr("constr1").feature("f1").selection().geom(1).set(1,2);
model.study().create("std1");
model.study("std1").create("stat", "Stationary");
model.study("std1").run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('geom1', 2);
model.geom('geom1').create('sq1', 'Square');
model.mesh.create('mesh1', 'geom1');
model.mesh('mesh1').run;
model.elementSet.create('es1');
model.elementSet('es1').set('A', '!bndadj(3,4) || bndadj(1,2)');
model.elementSet('es1').selection().geom(2).all;
model.shape.create('sh1', 'material1');
model.shape('sh1').create('f1', 'shlag');
model.shape('sh1').feature('f1').set('basename', 'u').set('order', 1);
model.shape('sh1').selection.geom(2).all;
model.shape('sh1').elementSet('A');
model.field.create('field1', 'u');
model.field('field1').shape({'sh1'});
model.intRule.create('ir1', 'material1');
model.intRule('ir1').create('o2').order(2);
model.weak.create('weak1');
model.weak('weak1').weak('if(A,ux*test(ux)+uy*test(uy)-test(u),0)');
model.weak('weak1').intRule('ir1');
model.weak('weak1').selection().geom(2).all;
model.constr.create('constr1', {'sh1'});
model.constr('constr1').create('f1');
model.constr('constr1').feature('f1').set('constr',1,{'u'});
model.constr('constr1').feature('f1').set('constrf',1,{'test(u)'});
model.constr('constr1').feature('f1').selection().geom(1).set([1,2]);
```

```

model.study.create('std1');
model.study('std1').create('stat', 'Stationary');
model.study('std1').run;

```

SEE ALSO

[model.shape\(\)](#)

model.extraDim

Create attachments of extra dimensions.

SYNTAX

```

model.extraDim().create(<tag>, type);
model.extraDim(<tag>).set(property, <value>);
model.extraDim(<tag>).model(<mtag>);
model.extraDim(<tag>).selection();
model.extraDim(<tag>).selection("point");
model.extraDim(<tag>).properties();
model.extraDim(<tag>).getType(property);
model.extraDim(<tag>).model();

```

DESCRIPTION

`model.extraDim().create(<tag>, type)` creates an extra dimension feature of the given type. The supported types are `AttachDimension`, `PointsToAttach`, and `Integration`.

`model.extraDim(<tag>).set(property, <value>)` specifies properties relevant for the selected extra dimension feature type.

- Properties for `AttachDimension`: `extradim`
- `PointsToAttach` has no properties.
- Properties for `Integration`: `opname`, `intrule`, `intorder`, `frame`, `axisym`

`model.extraDim(<tag>).model(<mtag>)` sets the model node. For features of type `AttachDimension`, this defines the base model node. For features of type `PointsToAttach` and `Integration`, this defines the extra dimension model node.

```
model.extraDim(<tag>).selection();
```

Returns the selection of the feature (for features of type `AttachDimension` and `Integration`).

`model.extraDim(<tag>).selection("point")` returns the selection of points to attach (for features of type `PointsToAttach`).

`model.extraDim(<tag>).properties()` returns the list of assigned properties as a string array.

`model.extraDim(<tag>).getType(property)` returns the value of the specified property.

`model.extraDim(<tag>).model()` returns the model node.

model.field()

Create and define properties for fields with shape functions that defined field variables.

SYNTAX

```
model.field().create(<tag>,<fname>);  
model.field(<tag>).field(<fname>);  
model.field(<tag>).shape(<shlist>);  
  
model.field(<tag>).field();  
model.field(<tag>).shape();  
model.field(<tag>).geom();
```

DESCRIPTION

`model.field().create(<tag>,<fname>)` creates a field with tag `<tag>` with the field name `<fname>`.

`model.field(<tag>).field(<fname>)` sets the field name.

`model.field(<tag>).shape(<shlist>)` sets the shape functions defining the field variables. `<shlist>` is a list of shape function tags. Each shape function defines one or more field variables. Together the shape functions specify which field variables there are in the field.

`model.field(<tag>).field()` returns the field name as a string.

`model.field(<tag>).shape()` returns the shape function tags as a string array.

`model.field(<tag>).geom()` returns the geometry associated with the field.

SEE ALSO

[model.shape\(\)](#), [model.coeff\(\)](#)

model.frame()

Create and defined properties for different types of frames: spatial frames, material frames, mesh frames, and geometry frames.

SYNTAX

```
model.frame().create(<tag>,<gtag>);
model.frame(<tag>).coord(<coordlist>);
model.frame(<tag>).coord(<pos>,<coord>);
model.frame(<tag>).meshFrame();
model.frame(<tag>).materialFrame();
model.frame(<tag>).geometryFrame();
model.frame(<tag>).spatialFrame();
model.frame(<tag>).sorder(order);
model.frame(<tag>).sshape.create(<stag>,type);
model.frame(<tag>).sshape(<stag>).type(type);
model.frame(<tag>).sshape(<stag>).sorder(order);
model.frame(<tag>).sshape(<stag>).coorddof(<dofs>);
model.frame(<tag>).sshape(<stag>).coorddof(<pos>,<dof>);
model.frame(<tag>).sshape(<stag>).refframe(<ftag>);
model.frame(<tag>).sshape(<stag>).coordexpr(<exprs>);
model.frame(<tag>).sshape(<stag>).coordexpr(<pos>,<expr>);
```

```
model.frame(<tag>).coord();
model.frame(<tag>).identifier();
model.frame(<tag>).varNameSuffix();
model.frame(<tag>).geom();
model.frame(<tag>).isMeshFrame();
model.frame(<tag>).isGeometryFrame();
model.frame(<tag>).isMaterialFrame();
model.frame(<tag>).isSpatialFrame();
model.frame(<tag>).sorder();
model.frame(<tag>).sshape(<stag>).type();
model.frame(<tag>).sshape(<stag>).sorder();
model.frame(<tag>).sshape(<stag>).coorddof();
model.frame(<tag>).sshape(<stag>).refframe();
model.frame(<tag>).sshape(<stag>).coordexpr();
```

DESCRIPTION

`model.frame().create(<tag>,<gtag>)` creates a new frame and assigns it to geometry *<gtag>*.

`model.frame(<tag>).coord(<coordlist>)` defines *<coordlist>* as a list of independent variables. (Formerly *sdim*.)

`model.frame(<tag>).coord(<pos>,<coord>)` edits the coordinate at position *<pos>* in the coordinate list.

`model.frame(<tag>).meshFrame()` sets this frame to be the mesh frame. Each geometry requires exactly one mesh frame. The first one added becomes the mesh frame. When assigning one frame to be the mesh frame, this flag is cleared in the previous frame being the mesh frame.

`model.frame(<tag>).geometryFrame()` sets this frame to be the geometry frame. Each geometry requires exactly one geometry frame. The first one added becomes the geometry frame. When assigning one frame to be the geometry frame, this flag is cleared in the previous frame being the geometry frame.

`model.frame(<tag>).materialFrame()` sets this frame to be the material frame. Each geometry requires exactly one material frame. The first one added becomes the material frame. When assigning one frame to be the material frame, this flag is cleared in the previous frame being the material frame.

`model.frame(<tag>).spatialFrame()` sets this frame to be the spatial frame. Each geometry requires exactly one spatial frame. The first one added becomes the spatial frame. When assigning one frame to be the spatial frame, this flag is cleared in the previous frame being the spatial frame.

`model.frame(<tag>).sorder(order)` sets the geometry shape order. This value is used if the frame is the mesh frame. The mesh frame ignores all *sshape* features.

`model.frame(<tag>).sshape().create(<stag>, type)` creates a frame feature of the given type. Possible types are `fixed` (default), `moving_abs`, `moving_rel`, and `moving_expr`.

`model.frame(<tag>).sshape(<stag>).type(type)` sets the type of the frame feature.

`model.frame(<tag>).sshape(<stag>).sorder(order)` sets the geometry shape order for `<stag>` to `order`.

`model.frame(<tag>).sshape(<stag>).coorddof(<dofs>)` sets the spatial coordinates for `<stag>` when the `moving_rel` type is used.

`model.frame(<tag>).sshape(<stag>).coorddof(<pos>, <dof>)` edits the coordinate name at position `<pos>` in the degree of freedom list.

`model.frame(<tag>).sshape(<stag>).refframe(<ftag>)` sets the reference frame for `<stag>` when the `moving_rel` type is used.

`model.frame(<tag>).sshape(<stag>).coordexpr(<exprs>)` sets the expressions for the mesh displacement for `<stag>`.

`model.frame(<tag>).sshape(<stag>).coordexpr(<pos>, <expr>)` edits the expression at position `<pos>` in the expression list.

`model.frame(<tag>).sshape(<stag>).selection().named(<seltag>)` assigns the frame feature to the named selection `<seltag>`.

`model.frame(<tag>).sshape(<stag>).selection().set(...)` defines a local selection that assigns the frame feature to geometric entities. For a complete list of methods available under `selection()`, see [model.selection\(\)](#). All types of selections are supported except the global one and selections containing interior mesh boundaries.

`model.frame(<tag>).coord()` returns the coordinate names as a string array.

`model.frame(<tag>).identifier()` returns the frame's identifier as a string.

`model.frame(<tag>).varNameSuffix()` returns the variable name suffix as a string.

`model.frame(<tag>).geom()` returns the geometry name as a string.

`model.frame(<tag>).isMeshFrame()` returns true if this frame is the mesh frame.

`model.frame(<tag>).isGeometryFrame()` returns true if this frame is the geometry frame.

`model.frame(<tag>).isMaterialFrame()` returns true if this frame is the material frame.

`model.frame(<tag>).isSpatialFrame()` returns true if this frame is the spatial frame.

`model.frame(<tag>).sorder()` returns the geometrical shape order.

`model.frame(<tag>).sshape(<stag>).type()` returns the type as a string.

`model.frame(<tag>).sshape(<stag>).sorder()` returns the spatial approximation order as an integer.

`model.frame(<tag>).sshape(<stag>).coorddof()` returns the spatial coordinates as a string array.

`model.frame(<tag>).sshape(<stag>).refframe()` returns the reference frame as a string.

`model.frame(<tag>).sshape(<stag>).coordexpr()` returns the spatial coordinate expressions as a string array.

`model.frame(<tag>).sshape(<stag>).selection().named()` returns the named selection tag.

`model.frame(<tag>).sshape(<stag>).selection().getType()` returns domain information. For available methods, see [model.selection\(\)](#).

SEE ALSO

[model.shape\(\)](#)

model.func()

Add different types of functions.

SYNTAX

```
model.func().create(<tag>,<type>);
model.func(<tag>).create(<tag>,<type>);
model.func(<tag>).label(<label>)
model.func(<tag>).model(<mtag>)
model.func(<tag>).set(property,<value>);
model.func(<tag>).set("funcname",<funcname>)
model.func(<tag>).discardData()
model.func(<tag>).importData()

model.func(<tag>).model()
model.func(<tag>).getType(property);
model.func(<tag>).functionNames()
model.func(<tag>).getAllowedPropertyValues(property);
```

DESCRIPTION

`model.func().create(<tag>,<type>)` creates a new function of type `<type>` with the tag `<tag>`. The types can be one of the following strings: Analytic, Elevation, External, Image, Interpolation, MATLAB (requires LiveLink™ for MATLAB®), Piecewise, GaussianPulse, Ramp, Random, Rectangle, Step, Triangle, and Wave. In addition, `model.create(<tag>,"FunctionSwitch")` creates a function switch. You can add other functions to a function switch:

```
model.func().create("sw1", "FunctionSwitch");
model.func("sw1").create("int1", "Interpolation");
model.func("sw1").create("an1", "Analytic");
model.func("sw1").create("rn1", "Random");
```

`model.func(<tag>).label(<label>)` sets a label for the function.

`model.func(<tag>).model(<mtag>)` sets the model node of the function.

`model.func(<tag>).set(property,<value>)` sets the value of a property of the function. See the available properties for each type of function below.

`model.func(<tag>).set("funcname",<funcname>)` sets the operator name of the function. The default operator name is `<tag>`.

`model.func(<tag>).model()` returns the model node.

`model.func(<tag>).getType(property)` retrieves a value of a function property.

`model.func(<tag>).importData()` imports the file that the function references into the model. This is possible for interpolation, elevation, and image functions.

`model.func(<tag>).discardData()` discards the data imported with `importData()`. This is possible for interpolation, elevation, and image functions.

`model.func(<tag>).functionNames()` returns an array containing the function names that the function feature defines. Most functions always return an array of length one, but interpolation function features, for example, can define an arbitrary number of function names.

`model.func(<tag>).getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

What properties are available depends on the type of function. The following function types are available:

Analytic

Generate an *analytic* function using a symbolic expression.

TABLE 2-45: ANALYTIC PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
argders	Nx2 String array	{}	(argument, partial derivative) pairs if dermethod is manual.
args	string array	{}	The arguments to the function.
complex	Boolean	false	True if the function can produce complex results for real inputs.
dermethod	automatic manual	Automatic	Automatic differentiation or manual control over the derivatives.
expr	string	None	The expression defining the function.
funcname	string	The tag name	The name of the function.
periodic	Boolean	false	True if the function should be extended to a periodic function.
periodiclower	string	0	The lower limit of the interval that is extended periodically.
periodicupper	string	1	The upper limit of the interval that is extended periodically.
argunit	string		A comma-separated list of required units for each argument.
fununit	string		The unit of the function's result.

Interpolation

Generate an *interpolation* function. You can use several interpolation and extrapolation methods..

TABLE 2-46: INTERPOLATION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
defineinv	on off	off	If source is table: Whether to define the inverse function.
defvars	Boolean	false	If source is file and defvars is set, the space variables are used as default arguments to the function if no arguments are supplied in a call to it.
extrap	const interior linear value	const	The extrapolation method.
extrapvalue	double	0	The extrapolation value if extrap is value.
filename	string		The file that contains the data if source is file.
funcinvname	string		If source is table and defineinv is on: The name of the inverse function.
funcname	string	The tag name	The name of the function if source is table.
funcs	string matrix		Used source is file; the first column contains function names and the second column contains the positions in the file where the corresponding function is defined
interp	neighbor linear piecewisecubic cubicspline	piecewisecubic	The interpolation method.

TABLE 2-46: INTERPOLATION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
modelres	string		If sourcetype is model, specifies the model resource that contains the interpolation data
nargs	integer (1-3)	1	The number of function arguments if struct is spreadsheet or source is resultTable.
primfunname	string		Define a primitive function with the name give as primfunname.
resultTable	string		The tag of the result table to use (tbl1, for example).
source	table file resultTable	table	If sourcetype is user, specifies whether the data is entered in a local table, read from a file, or taken from a results table.
sourcetype	model user	user	Specifies if the data for the function is stored in the model or provided by the user.
struct	grid sectionwise spreadsheet	spreadsheet	The data format if source is file.
table	Nx2 String array	Empty	Contains the point/value pairs if the source is table.
argunit	string		A comma-separated list of required units for each argument.
fununit	string		The unit of the function's result.

Piecewise

Generate a *piecewise* interpolation function, which is created by splicing together several functions, each defined on one interval.

TABLE 2-47: PIECEWISE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
arg	string	x	The argument to the function.
extrap	const interior none periodic value	const	The extrapolation method.
extrapvalue	double	0	The extrapolation value if extrap is value.
funcname	string	The tag name.	The name of the function.
pieces	Nx3 String array	Empty	(left, right, expression) for each interval.
smooth	none cont contd1 contd2	none	The type of smoothing.
smoothzone	double	0.1	The relative size of the smoothing zone if smoothing is enabled.
argunit	string		A comma-separated list of required units for each argument.
fununit	string		The unit of the function's result.

GaussianPulse

Generate a *Gaussian pulse* function. This function is the common bell-shaped curve (Gaussian function).

TABLE 2-48: GAUSSIAN PULSE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
funcname	string	The tag name	The name of the function.

TABLE 2-48: GAUSSIAN PULSE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
location	string	0	Where the pulse peaks.
sigma	string	1	The standard deviation of the underlying normal distribution.

Ramp

Generate a *ramp* function.

TABLE 2-49: RAMP PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
cutoffactive	Boolean	false	If true, then the ramp ends when it reaches the cutoff value.
cutoff	double	1	If cutoffactive is true, the level where the ramp ends.
funcname	string	The tag name	The name of the function.
location	string	0	Where the ramp starts.
slope	string	1	The slope of the ramp.
ncontder	1 or 2	2	The number of continuous derivatives if smoothing is enabled.
smoothcutoff	Boolean	false	Smooth the transition where the ramp ends at the cutoff?
smoothloc	Boolean	false	Smooth the transition where the ramp starts?
smoothzone	double	0.1	The relative size of the smoothing zone if smoothing is enabled.

Random

Generate a *random* function. The random function can have a uniform or normal distribution.

TABLE 2-50: RANDOM PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
funcname	string	The tag name	The name of the function.
mean	string	0	The average value.
nargs	integer	1	The number of arguments.
normalsigma	string	1	The standard deviation if type is Normal.
type	uniform normal	Uniform	The distribution type.
uniformrange	string	1	The range if type is Uniform.

Rectangle

Generate a *rectangle*-shaped function.

TABLE 2-51: RECTANGLE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
funcname	string	The tag name	The name of the function.
lower	string	-0.5	Where the high zone begins.
ncontder	1 or 2	2	The number of continuous derivatives if smoothing is enabled.
smooth	Boolean	true	Smooth the transitions?
smoothzone	string	0.1	The size of the smoothing zone on both sides of the transitions.
upper	string	0.5	Where the high zone ends.

Step

Generate a *step* function.

TABLE 2-52: STEP PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
from	string	0	The value to the left of the location.
funcname	string	The tag name	The name of the function.
location	string	0	Where the step is located.
ncontder	1 or 2	2	The number of continuous derivatives if smoothing is enabled.
smooth	Boolean	true	Smooth the transition?
smoothzone	string	0.1	The size of the smoothing zone on both sides of location.
to	string	1	The value to the right of the location.

Triangle

Generate a *triangle*-shaped function.

TABLE 2-53: TRIANGLE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
funcname	string	The tag name	The name of the function.
lower	string	-0.5	Where the high zone begins.
ncontder	1 or 2	2	The number of continuous derivatives if smoothing is enabled.
smooth	Boolean	true	Smooth the transitions?
smoothzone	string	0.1	Size of smoothing zone on both sides of the transitions.
upper	string	0.5	Where the high zone ends.

External

Generate an *external* function that interfaces to other external functions written in the C language.

TABLE 2-54: EXTERNAL PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
ders	Nx3 string array		(function name, argument, partial derivative) triplets.
funcs	string array		The functions defined by the library.
init	string		The string that is sent to the library when the function feature is initialized.
path	string		The path to the shared library that defines the functions.

An external function is a function defined in a shared library written by the user. The shared library must define the following three functions with C linkage:

- `int init(const char *str)` is called when the function is initialized with the string from the **Initialization data** field. It returns a nonzero value in case of success and zero in case of failure. This function might be called several times; it is always called before solving a model that uses the function.
- `int eval(const char *func, int nArgs, const double **inReal, const double **inImag, int blockSize, double *outReal, double *outImag)` is called for elementwise evaluation of the function `func` called with `nArgs` arguments of length `blockSize`. The array `inReal` contains the real parts of the arguments; it has length `nArgs`, and each element has length `blockSize`.

If the arguments are all-real, then `inImag` is null; otherwise it contains the imaginary parts of the arguments. If the function evaluation is successful, 1 is returned if it resulted in an all-real array and 2 is returned if it resulted in a complex array. The function should return 0 in case of error. In case of a real result, the function values should be written to the array `outReal`. In case of a complex result, the real parts of the function should be

written to `outReal` and the imaginary parts to `outImag`. The `outReal` and `outImag` arrays both have length `blockSize`. All matrices are allocated and deallocated by COMSOL.

- `const char *getLastError()` returns the last error that has occurred. A null or empty string is returned if no error has occurred. Calling `init()` or `eval()` must set the last error string to "" or null. All memory allocation of this string is handled by the shared library. There is no localization of the error messages.

If you are using Microsoft Visual Studio to compile your library, you can declare the functions as `__declspec(dllexport)` to export them from the DLL.

An example of a library that defines a function called `extsinc` that computes the `sinc` function ($\sin(x)/x$):

```
#include <math.h>
#include <stdlib.h>
#include <string.h>

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

static const char *error = NULL;

EXPORT int init(const char *str) {
    return 1;
}

EXPORT const char * getLastError() {
    return error;
}

EXPORT int eval(const char *func,
                int nArgs,
                const double **inReal,
                const double **inImag,
                int blockSize,
                double *outReal,
                double *outImag) {
    int i, j;

    if (strcmp("extsinc", func) == 0) {
        if (nArgs != 1) {
            error = "One argument expected";
            return 0;
        }
        for (i = 0; i < blockSize; i++) {
            double x = inReal[0][i];
            outReal[i] = (x == 0) ? 1 : sin(x) / x;
        }
        return 1;
    }
    else {
        error = "Unknown function";
        return 0;
    }
}
```

To compile this function into a library, place it in `ext.c` and proceed as follows depending on platform:



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

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

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

Elevation

Generate an *elevation* function by importing geospatial elevation data from digital elevation models (DEM files).

TABLE 2-55: ELEVATION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
extrap	const interior linear value	const	The extrapolation method.
extrapvalue	double	0	The extrapolation value if extrap is value.
filename	string		The name of the DEM file.
funcname	string	The tag name	The name of the function.
interp	neighbor linear	linear	The interpolation method.

Image

Generate an *image* function from a BMP, GIF, JPEG, or PNG file.

TABLE 2-56: IMAGE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
argunit	string		The unit of the function arguments.
clipmaxx	double	1000	If clipping is manual: The maximum pixel x-coordinate that is kept.
clipminx	double	0	If clipping is manual: The minimum pixel x-coordinate that is kept.
clipmaxy	double	1000	If clipping is manual: The maximum pixel y-coordinate that is kept.
clipminy	double	0	If clipping is manual: The minimum pixel y-coordinate that is kept.
clipping	none manual	none	The clipping method.
extrap	const interior linear value	const	The extrapolation method.
extrapvalue	double	0	The extrapolation value if extrap is value.

TABLE 2-56: IMAGE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
fununit	string		The unit of the function value.
filename	string		The name of the DEM file.
flipx	Boolean	false	If inplace is false: Whether to flip the image horizontally when mapping it to the xy-plane.
flipy	Boolean	false	If inplace is false: Whether to flip the image vertically when mapping it to the xy-plane.
funcname	string	The tag name	The name of the function.
inplace	Boolean	false	If true, the image is mapped to the xy-plane without scaling; 1 length unit corresponds to 1 pixel.
interp	neighbor linear	linear	The interpolation method.
manualexpr	string	(r+g+b)/3	If scaling is manual: The scaling function expressed in terms of the red (r), green (g), and blue (b) pixel intensities.
scaling	automatic manual	automatic	The method used for computing function values from pixel colors.
xmax	double	1	If inplace is false: The maximum x-coordinate of the region to which the image is mapped.
xmin	double	0	If inplace is false: The minimum x-coordinate of the region to which the image is mapped.
ymax	double	1	If inplace is false: The maximum y-coordinate of the region to which the image is mapped.
ymin	double	0	If inplace is false: The minimum y-coordinate of the region to which the image is mapped.

MATLAB

Declare use of function in *MATLAB*. This requires the LiveLink™ for MATLAB®.

TABLE 2-57: MATLAB PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
ders	Nx3 string array		(function name, argument, partial derivative) triplets.
funcs	string array		The functions defined by MATLAB.

Wave

Use a *wave* function to generate a wave-shaped function. The wave shape can be a sawtooth, sine wave, square wave, or triangle wave.

TABLE 2-58: WAVE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
amplitude	string	1	The amplitude.
freq	string	1	The angular frequency.
funcname	string	The tag name	The name of the function.
ncontder	1 or 2	2	The number of continuous derivatives if smoothing is enabled.
phase	string	0	The phase.
smooth	Boolean	true	Smooth the transitions? (Only used for wave forms with discontinuous function values or derivatives.)

TABLE 2-58: WAVE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
smoothzone	string	0.1	The size of smoothing zone on both sides of the transitions.
type	sawtooth sine square triangle	sine	The type of wave form.

SEE ALSO

`model.material()`

model.geom()

Creating and specifying general properties for 1D, 2D, and 3D geometries.

SYNTAX

```
model.geom().create(<tag>,<sdim>);
model.geom().create(<tag>,<meshtag>,<filename>);
model.geom(<tag>).model(<mtag>);
model.geom(<tag>).model();
model.geom(<tag>).axisymmetric(boolean);
model.geom(<tag>).isAxisymmetric();
model.geom(<tag>).lengthUnit(<unit>);
model.geom(<tag>).lengthUnit();
model.geom(<tag>).angularUnit(<unit>);
model.geom(<tag>).angularUnit();
model.geom(<tag>).scaleUnitValue(boolean);
model.geom(<tag>).scaleUnitValue();
model.geom(<tag>).repairTol(<relTol>);
model.geom(<tag>).repairTol();
model.geom(<tag>).geomRep(geomrep);
model.geom(<tag>).geomRep();
model.geom().remove(<tag>);

model.geom(<tag>).create(<ftag>,type);
model.geom(<tag>).createAfter(<ftag>,<type>,<postag>);
model.geom(<tag>).feature(<ftag>).create(<ftag2>,type);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).selection(property).selMethod;
model.geom(<tag>).feature(<ftag>).geom().geomMethod;
model.geom(<tag>).feature(<ftag>).active(boolean);
model.geom(<tag>).feature(<ftag>).isActive();
model.geom(<tag>).feature().move(<ftag>,<position>);
model.geom(<tag>).feature().remove(<ftag>);
model.geom(<tag>).feature(<ftag>).getAllowedPropertyValues(property);

model.geom(<tag>).run(<ftag>);
model.geom(<tag>).runPre(<ftag>);
model.geom(<tag>).runCurrent();

model.geom(<tag>).run();
model.geom().run();

model.geom(<tag>).current();
model.geom(<tag>).feature(<ftag>).status();
model.geom(<tag>).feature(<ftag>).message();

model.geom(<tag>).objectNames();
model.geom(<tag>).feature(<ftag>).objectNames();
model.geom(<tag>).obj(<objname>).geomInfoMethod
model.geom(<tag>).geomInfoMethod
model.geom(<tag>).selection(<seltag>).selMethod;
```


SYNTAX

```
model.geom(<tag>).measure().geomMeasurementMethod;  
model.geom(<tag>).measureFinal().geomMeasurementMethod;  
  
model.geom(<tag>).export(<filename>);  
model.geom(<tag>).exportFinal(<filename>);  
  
model.geom(<tag>).defeaturing(<tag>).defeaturingMethod;  
model.geom(<tag>).feature(<ftag>).find();  
model.geom(<tag>).feature(<ftag>).detail().selMethod;  
  
model.geom(<tag>).insertFile(<filename>, <gtag>);  
model.geom(<tag>).insertSequence(<mtag>, <gtag>);  
  
model.geom().create(<tag>, "Subsequence", sDim);  
model.geom(<tag>).inputParam().set(<name>, <expr>, <descr>);  
model.geom(<tag>).localParam().set(<name>, <expr>, <descr>);  
model.geom(<gtag>).create(<ftag>, "PartInstance");  
model.geom(<gtag>).stepInto(<ftag>);  
model.geom(<gtag>).feature(<ftag>).geom().run(<ftag2>);
```

The last group of syntaxes are only applicable for geometry subsequences (see [Using Geometry Parts](#)), but `model.geom(<gtag>).feature(<ftag>).geom()` also exists if `<ftag>` is a work plane feature.

```
model.geom().load(<tags>, <filename>, <subsequenceTagsInFile>);  
model.geom(<gtag>).loaded();  
  
model.geom(<gtag>).filename();  
model.geom(<gtag>).filename(<filename>);  
model.geom(<gtag>).tagInFile();  
model.geom(<gtag>).dateModifiedInFile();  
model.geom(<gtag>).commentsInFile();  
model.geom(<gtag>).labelInFile();  
model.geom(<gtag>).versionInFile();  
model.geom(<gtag>).reload();
```

The last group of syntaxes are only applicable for geometry subsequences (see [Using Geometry Parts](#)).

DESCRIPTION

Geometry Sequences and Geometry Objects

`model.geom(<tag>)` returns a geometry sequence consisting of geometry features. The geometry sequence also contains geometry objects resulting from building the geometry sequence.

Creating and Deleting a Geometry

`model.geom().create(<tag>, <sdim>)` creates a geometry sequence of space dimension `<sdim>` and assigns it the tag `<tag>`.

`model.geom().create(<tag>, <meshtag>, <filename>)` creates a geometry sequence tagged `<tag>` and a corresponding meshing sequence tagged `<meshtag>`. The parameter `<filename>` specifies a file that contains a geometry or a mesh, and an import feature is inserted into the geometry or mesh sequence.

`model.geom().remove(<tag>)` deletes the geometry tagged `<tag>`.

General Geometry Settings

`model.geom(<tag>).model(<mtag>)` sets the model node of the geometry `<tag>` to `<mtag>`.

`String mtag = model.geom(<tag>).model()` returns the model node tag of the geometry.

`model.geom(<tag>).axisymmetric(boolean)` indicates if the geometry is axisymmetric. This is only applicable for 1D and 2D geometries.

`model.geom(<tag>).isAxisymmetric()` returns `true` if the geometry is axisymmetric and `false` otherwise.

`model.geom(<tag>).lengthUnit(<unit>)` sets the length unit.

`String unit = model.geom(<tag>).lengthUnit()` returns the length unit.

`model.geom(<tag>).angularUnit(<unit>)` sets the angular unit.

`String unit = model.geom(<tag>).angularUnit()` returns the angular unit.

`model.geom(<tag>).scaleUnitValue(boolean)` sets the geometry to scale property values when units are changed.

`model.geom(<tag>).scaleUnitValue()` returns `true` if the geometry is set to scale property values when units are changed.

`model.geom(<tag>).repairTol(<relTol>)` sets the default relative repair tolerance to use when creating new features.

`double relTol = model.geom(<tag>).repairTol()` returns the default relative repair tolerance.

`model.geom(<tag>).geomRep(geomrep)` sets the geometry representation to use in a 3D geometry. The *geomrep* string can be `comsol`, meaning the COMSOL kernel or `cadps` (requires the CAD Import Module), meaning the CAD kernel (Parasolid).

`String geomrep = model.geom(<tag>).geomRep()` returns the geometry representation.

Creating, Editing, Disabling, and Deleting Features

`model.geom(<tag>).create(<ftag>, type)` adds a geometry feature *<ftag>* of type *type* to the geometry *<tag>*, after the current feature.

`model.geom(<tag>).feature(<ftag>).set(property, <value>)` sets a property in the geometry feature *<ftag>*. All data types listed in [Table 2-2](#) are supported; the applicable data types differ between the properties. String expressions can use parameters from `model.param()`.

`model.geom(<tag>).feature(<ftag>).getType(property)` returns the value of a property in the geometry feature *<ftag>*.

`model.geom(<tag>).feature(<ftag>).selection(property).selMethod` manages the geometry object selection property *property*. The available selection methods are described in [Geometry Object Selection Methods](#).

`model.geom(<tag>).feature(<ftag>).geom().geomMethod` manipulates the 2D geometry sequence corresponding to the work plane feature *<ftag>*. The available methods are the same as for a 2D geometry `model.geom(<gtag>)`.

`model.geom(<tag>).feature(<ftag>).active(false)` disables the feature *<ftag>*.

`model.geom(<tag>).feature(<ftag>).active(true)` enables the feature *<ftag>*.

`model.geom(<tag>).feature(<ftag>).isActive()` returns `true` if the feature *<ftag>* is enabled, and `false` otherwise.

`model.geom(<tag>).feature().remove(<ftag>)` removes the feature *<ftag>*.

`model.geom(<tag>).feature().move(<ftag>, <position>)` moves the feature *<ftag>* to the zero indexed position *<position>* in the sequence.

`model.geom(<tag>).feature(<ftag>).getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns `null`.

Building Features

After each build operation, the current feature is set as the last of the active features that were built. The current state contains all objects that are generated by these features.

`model.geom(<tag>).run(<ftag>)` builds all features up to (and including) the feature `<ftag>`.

`model.geom(<tag>).runPre(<ftag>)` builds all features preceding the feature `<ftag>`.

`model.geom(<tag>).runCurrent()` builds all features up to (and including) the current feature.

`model.geom(<tag>).run()` builds all features. The finalized geometry and all selections are also updated.

`model.geom().run()` builds the finalized geometry in all geometries.

Getting Build Status

`String fTag = model.geom(<tag>).current()` returns the tag of the current feature. If the current state is before the first feature, the empty string "" is returned.

`String status = model.geom(<tag>).feature(<ftag>).status()` returns the status of the feature `<ftag>`. The status is built, warning, needs_rebuild, edited, or error.

`String msg = model.geom(<tag>).feature(<ftag>).message()` returns the warning/error message of the feature `<ftag>`.

Getting Information About Geometry Objects

`String[] n = model.geom(<tag>).objectNames()` returns the names of all objects that exist in the current state.

`String[] n = model.geom(<tag>).feature(<ftag>).objectNames()` returns the names of the output object generated by the feature `<ftag>`.

`model.geom(<tag>).obj(<objname>).geomInfoMethod` returns information about the object `<objname>`. The available methods are described in [Geometry Object Information](#).

`model.geom(<tag>).geomInfoMethod` returns information about the finalized geometry of geometry `<tag>`.

Getting Information About Named Selections

`model.geom(<tag>).selection(<seltag>).selMethod` returns information about the named selection `<seltag>`. See [Selections of Geometric Entities](#) and [Geometry Object Selection Methods](#) for more information.

Geometric Measurements

Use `model.geom(<tag>).measure().selection().selMethod` to specify the domains, boundaries, or edges in geometry objects that you want to measure. You can also specify one vertex or two vertices to get the coordinates of the vertex or the distance between the two vertices, respectively. The available selection methods are described in [Geometry Object Selection Methods](#).

`model.geom(<tag>).measure().geomMeasurementMethod` returns the volume, area, length, vertex coordinates, or distance between two vertices according to the selection. The available measurement methods are described in [Geometry Object Information](#).

Use `model.geom(<tag>).measureFinal().selection().selMethod` to specify the domains, boundaries, or edges in the finalized geometry that you want to measure. You can also specify one vertex or two vertices to get the coordinates of the vertex or the distance between the two vertices, respectively. The available selection methods are described in [model.selection\(\)](#).

`model.geom(<tag>).measureFinal().geomMeasurementMethod` returns the volume, area, length, vertex coordinates, or distance between two vertices according to the selection. The available measurement methods are

described in [Measurements](#).

Exporting Geometry Objects

`model.geom(<tag>).export().selection().selMethod` can be used to select a number of geometry objects to export to file. The available selection methods are described in [Geometry Object Selection Methods](#).

`model.geom(<tag>).export(<filename>)` exports the selected objects to a file.

`model.geom(<tag>).exportFinal(<filename>)` exports the finalized geometry to a file.

CAD Defeaturing

If you have a license for the CAD Import Module, or a LiveLink™ product for CAD software, the following functionality is available. For details, see the *CAD Import Module User's Guide*.

`model.geom(<tag>).defeaturing(tooltag).defeaturingMethod` uses a defeaturing tool to create a feature that deletes small details. Available tools are listed in the *CAD Import Module User's Guide*.

`model.geom(<tag>).feature(<ftag>).find()` searches for small details, for a defeaturing feature `<ftag>`.

`model.geom(<tag>).feature(<ftag>).detail().selMethod` manipulates the selection of details to remove, for a defeaturing feature `<ftag>`.

Geometry Object Selection Methods

For a geometry object selection `sel`, the following methods are available:

`sel.init()` sets the selection to be a selection of whole geometry objects. Subsequent calls to `set`, `add`, and `remove` select objects.

`sel.init(dim)` sets the selection property to be a selection of geometric entities of dimension `dim`. Subsequent calls to `all`, `set`, `add`, `remove`, and `clear` select entities.

The following three methods are applicable when the selection consists of whole objects. The argument `<onames>` can be an array of strings, or several string arguments.

`sel.set(<onames>)` sets the selection to be the objects `<onames>`.

`sel.add(<onames>)` adds the objects `<onames>` to the selection.

`sel.remove(<onames>)` removes the objects `<onames>` from the selection.

The following five methods are applicable when the selection consists of geometric entities. The argument `<entities>` can be an array of integers, or several integer arguments.

`sel.all(<oname>)` sets the selection to be all the entities of object `<oname>`. The selections on other objects are not affected.

`sel.set(<oname>, <entities>)` sets the selection on object `<oname>` to be `<entities>`. The selections on other objects are not affected.

`sel.add(<oname>, <entities>)` adds the entities `<entities>` to the selection on object `<oname>`. The selections on other objects are not affected.

`sel.remove(<oname>, <entities>)` removes the entities `<entities>` from the selection on object `<oname>`. The selections on other objects are not affected.

`sel.clear(<oname>)` clears the selection on object `<oname>`. The selections on other objects are not affected.

To let the selection be defined by a named selection, use:

`sel.named(<seltag>)` where `<seltag>` is the trimmed tag of a named selection defined by a preceding feature in the geometry sequence. See [Selections of Geometric Entities](#) for more information.

To get information about the selection, use:

`String[] onames = sel.objects()` returns the names of the selected objects.

`int dim = sel.dimension()` returns the dimension for the entities in the selection if the selection consists of geometric entities.

`int[] ent = sel.entities(<oname>, dim)` returns the entities in the selection on object <oname> if the selection consists of geometric entities.

`String[] seltag = sel.named()` returns the trimmed tag of the named selection that this selection refers to, or an empty string if the selection does not refer to a named selection.

If Statements

Use `model.geom(<tag>).create(<ftag>, <type>)` to add an If, Else If, Else, or End If feature after the current feature.

Use `model.geom(<tag>).createAfter(<ftag>, <type>, <postag>)` to add an If, Else If, Else, or End If feature after the feature tagged <postag>. [[can feature(). be removed here and above?]]

Insert Sequence

`model.geom(<tag>).insertFile(<filename>, <gtag>)`; inserts a geometry sequence, with tag <gtag>, from another model file, with file name <filename>, into the geometry sequence with tag <tag>.

`model.geom(<tag>).insertSequence(<mtag>, <gtag>)`; inserts a geometry sequence, with tag <gtag>, from another model, with tag <mtag>, into the current geometry sequence with tag <tag>.

EXAMPLE

Create a 2D geometry model as the union of a circle and rectangle.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
g.create("r1", "Rectangle");
g.feature("r1").set("size", new double[] {0.5, 1});
g.feature("r1").set("pos", new double[] {-1, 0});
g.create("c1", "Circle");
g.feature("c1").set("r", 0.5);
g.feature("c1").set("pos", new double[] {0.5, 0});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
g.create('r1', 'Rectangle');
g.feature('r1').set('size', [0.5, 1]);
g.feature('r1').set('pos', [-1, 0]);
g.create('c1', 'Circle');
g.feature('c1').set('r', 0.5);
g.feature('c1').set('pos', [0.5, 0]);
g.run;
```

COMPATIBILITY

From version 5.1, the access method `argument()` is deprecated and replaced with `inputParam()`, and the access method `param()` is deprecated and replaced with `localParam()`. The methods `link(...)`, `linked()`, and `relink()` are deprecated and replaced with `load(...)`, `loaded()`, and `reload()`, respectively.

From version 4.4, the method

```
model.geom(<tag>).runAll();
```

is deprecated. Instead, use

```
model.geom(<tag>).run(<ftag>);
```

to specify to which feature node in the geometry sequence you want to run.

From version 4.3a, the methods

```
model.geom(<tag>).object(<objname>)  
String[] onames = sel.object()
```

are deprecated and replaced by the following methods:

```
model.geom(<tag>).obj(<objname>)  
String[] onames = sel.objects()
```

SEE ALSO

[model.mesh\(\)](#)

model.group()

Add load groups and constraint groups.

SYNTAX

```
model.group().create(<tag>, type);  
model.group(<tag>).identifier(<id>);
```

```
model.group(<tag>).type();  
model.group(<tag>).identifier();
```

DESCRIPTION

`model.group().create(<tag>, type)` creates a new group of the specified type, which can be either `LoadGroup` or `ConstraintGroup`.

`model.group(<tag>).identifier(<id>)` sets the group identifier, which is used for defining a corresponding parameter, group.<id>.

`model.group(<tag>).type()` returns the group type as a string.

`model.group(<tag>).identifier()` returns the group identifier.

model.init()

Creating and defining initial values.

SYNTAX

```
model.init().create(<tag>);  
model.init(<tag>).set(<fieldname>, <expr>);  
model.init(<tag>).remove(<fieldname>);
```

```
model.init(<tag>).varnames();  
model.init(<tag>).get(<fieldname>);
```

DESCRIPTION

`model.init().create(<tag>)` creates a new initial value with tag <tag>.

`model.init(<tag>).set(<fieldname>, <expr>)` defines the expression <expr> as the initial value for the dependent variable (field variable) <fieldname>.

`model.init(<tag>).remove(<fieldname>)` removes the field variable `<fieldname>` from the initial value with tag `<tag>`.

`model.init(<tag>).selection().named(<seltag>)` assigns the initial value to the named selection `<seltag>`.
`model.init(<tag>).selection().set(...)` defines a local selection that assigns the initial value to geometric entities. For a complete list of methods available under `selection()`, see [Selections](#). Only selections at a single geometry level is allowed except for ODE states which require the global selection.

`model.init(<tag>).varnames()` returns the names of the variables for the initial value with tag `<tag>` as a string array.

`model.init(<tag>).get(<fieldname>)` returns the initial value for the field variable `<fieldname>` as a string.

`model.init(<tag>).selection().named()` returns the named selection tag.

`model.init(<tag>).selection().getType()` returns domain information for the initial value with tag `<tag>`; see [Selections](#) for available methods.

model.intRule()

Integration rules.

SYNTAX

```
model.intRule().create(<tag>,<ftag>);
model.intRule(<tag>).frame(<ftag>);
model.intRule(<tag>).create(<ftag>);
model.intRule(<tag>).feature(<ftag>).order(gporder);
model.intRule(<tag>).feature(<ftag>).getAllowedPropertyValues(property);
model.intRule(<tag>).frame();
```

DESCRIPTION

`model.intRule().create(<tag>,<ftag>)` creates an integration rule for the frame `<ftag>`.

`model.intRule(<tag>).frame(<ftag>)` sets the frame for the integration rule.

`model.intRule(<tag>).create(<ftag>)` creates an integration rule feature.

`model.intRule(<tag>).feature(<ftag>).order(gporder)` specifies the integration order of the integration rule.

`model.intRule(<tag>).frame()` returns the frame as a string.

`model.intRule(<tag>).feature(<ftag>).getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

EXAMPLE

Specify two integration rules, one with the integration order 2 and one with the integration order 4.

Code for use with Java

```
model.intRule().create("ir1","f");
model.intRule("ir1").create("ir1").order(2);
model.intRule("ir1").create("ir2").order(4);
```

Code for use with MATLAB

```
model.intRule.create('ir1','f');
model.intRule('ir1').create('ir1').order(2);
model.intRule('ir1').create('ir2').order(4);
```

SEE ALSO

[model.shape\(\)](#)

model.massProp()

Compute mass properties and add mass contributions.

SYNTAX

```
model.massProp().create(<tag>,"MassProperties");
model.massProp(<tag>).selection();
model.massProp(<tag>).set(<pname>,<expr>);

model.massProp(<tag>).create(<mctag>,"MassContributions");
model.massProp(<tag>).feature(<mctag>).selection();
model.massProp(<tag>).feature(<mctag>).set(<pname>,<expr>);
```

DESCRIPTION

Creating Mass Properties and Providing Geometry Source Selections

`model.massProp().create(<tag>,"MassProperties")` creates a mass properties object that computes variables for mass properties such as total mass and the center of mass.

`model.massProp(<tag>).selection().set(...)` defines a local selection that assigns the mass properties to geometric entities that act as sources. For a complete list of methods available under `selection()`, see [model.selection\(\)](#).

Specifying Density and Mass Properties

You can specify the density to be taken from a physics. For example,

```
model.massProp(<tag>).set("densitySource", "fromSpecifiedPhysics")
model.massProp(<tag>).feature("mc1").set("physics", "solid");
```

takes the density from the Solid Mechanics interface `solid`.

You can also specify a user-defined density. For example, to specify the density to be 1107 kg/m^3 , use

```
model.massProp(<tag>).set("densitySource", "userDefined");
model.massProp(<tag>).set("expr", "1107[kg/m^3]");
```

To create variables for the center of mass and to not create variables for the moment of inertia:

```
model.massProp(<tag>).set("createCenterOfMass", "on");
model.massProp(<tag>).set("createMomentOfInertia", "off");
```

Adding a Mass Contribution

You can add mass contributions from other parts of the geometry (an adjacent boundary, for example) by creating a mass contribution:

```
model.massProp(<tag>).create("mc1","MassContributions");
model.massProp(<tag>).feature("mc1").set("expr", "3");
```

For the mass contributions, you can specify source selections and define the density in the same way as for the mass properties.

Properties and Variables for massProp

The massProp feature accepts the following properties:

TABLE 2-59: MASSPROP PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
createCenterOfMass	on off	on	Create center of mass variables.
createMass	on off	on	Create mass variable.
createMassContribution	on off	on	Add a mass contribution.
createMomentOfInertia	on off	on	Create moment of inertia variables.
createPrincipalInertia	on off	on	Create principal moment of inertia variables.
createVolume	on off	on	Create volume variable.
densitySource	fromSpecifiedPhysics userDefined	userDefined	Source for the density values.
expr	string		User-defined density expression.
includeLowerPh	on off	on	Include adjacent entities of lower dimension for density values taken from physics.
integrationFrame	frame	Material	Integration frame for user-defined density.
intorder	4	Positive integer	Integration order.
outputFrame	frame	Material	Integration frame for output variables.

The created variables for different mass properties have the following default names:

- `mass1.mass` and `mass1.volume` for the mass and volume, respectively.
- `mass1.CMX`, and so on, for the coordinates of the center of mass.
- `mass1.IXY`, and so on, for the components of the moment of inertia.
- `mass1.Ip1`, and so on, for the moment of inertia principal values.
- `mass1.Ip1X`, and so on, for the moment of inertia principal directions.

model.material()

Materials and material property groups.

SYNTAX

```
model.material().create(<tag>);
model.material().create(<tag>,<type>);
model.material().create(<tag>,<type>,<comp>);
model.material(<tag>).info(<itag>);
model.material(<tag>).info();
model.material(<tag>).label(<label>);
model.material(<tag>).materialType(<mattype>);
model.material(<tag>).propertyGroup(<mtag>);
model.material(<tag>).propertyGroup();
model.material(<tag>).prefix(<prefix>);
model.material(<tag>).prefix();
model.material(<tag>).selection();
model.material(<tag>).set(<pname>,<expr>);
model.material().move(<tag>,<position>);

MaterialModel mm = model.material(<tag>).propertyGroup().create(<mtag>,<descr>);
mm.addInput(<quantity>);
mm.func();
mm.func(<ftag>);
mm.getString(<pname>);
mm.getStringArray(<pname>);
mm.hasParam(<pname>);
mm.info(<itag>);
mm.info();
mm.input();
mm.isOutput(<pname>);
mm.param();
mm.removeInput(<quantity>);
mm.set(<pname>,<expr>);
mm.size(<pname>);
mm.suffix(<suffix>);
mm.suffix();

mm.info().create(<itag>,<descr>);
mm.info(<itag>).title(<title>);
mm.info(<itag>).title();
mm.info(<itag>).body(<body>);
mm.info(<itag>).body();
```

DESCRIPTION

A material is a collection of property groups, where each property group defines a set of material properties, material functions, and model inputs that can be used to define a temperature-dependent material property, for example. A property group usually defines properties used by a particular material model to compute a fundamental quantity. A material property can either be a visible output property or a local parameter. The output property is visible for physics interfaces; local properties are only visible inside the property group. If two property groups define the same output property, the last property group determines the value of the output property. The material function is used by the property group to calculate a property or parameter value as a function of other variables, usually model inputs. The model input is a quantity that the material model recognizes as an input variable (temperature, for example). The actual variable that represents the model input is not known until the model is solved, and it can also be different between physics interfaces.

There are two types of property groups, user-defined and specialized. When a material is created, there is always one default user-defined property group present. To this property group it is possible to add output properties from a predefined list of quantities. These quantities are recognized by all physics interfaces as material properties—for example, thermal conductivity, electric conductivity, and density. The full list is presented in the physics interface for the default property group. The specialized property groups are built in and usually define few output properties that only some physics interfaces can access. These output properties are not necessarily part of the allowed properties for the default property group. An example of such a specialized group is the refractive index material

model, which defines the real and imaginary part of the refractive index as output properties. These properties can only be accessed by the Electromagnetic Waves interface.

`model.material().create(<tag>)` creates a new material for the model component (the last component if there are more than one component in the model).

`model.material().create(<tag>,"Common")` also creates a new material for the model component (the last component if there are more than one component in the model).

`model.material().create(<tag>,"Switch")` creates a material switch for the model component (the last component if there are more than one component in the model). You can add materials to the material switch:

```
model.material().create("sw1", "Switch", "");
model.material("sw1").feature().create("mat1", "Common", "");
model.material("sw1").feature().create("mat2", "Common", "");
```

`model.material().create(<tag>,"Link")` creates a material link for the model component (the last component if there are more than one component in the model).

`model.material().create(<tag>,<type>,<comp>)` creates a material, material switch, or material link in the component `<comp>`.

`model.material().create(<tag>,<type>,"")` creates a global material, material switch, or material link.

`model.material(<tag>).info(<itag>)` returns an information item for a material.

`model.material(<tag>).input()` returns the list of model inputs.

`model.material(<tag>).materialType("solid")` or `model.material(<tag>).materialType("nonSolid")` sets the material type to a solid or a nonsolid material, respectively.

`model.material(<tag>).propertyGroup(<mtag>)` gets the property group named `<mtag>` for the material.

`model.material(<tag>).selection()` returns the selection of the material. The selection determines which geometry the material belongs to.

`model.material(<tag>).set("family",<appearance>)` sets the appearance to a family of materials (as strings; "water", for example): air, aluminum, brick, concrete, copper, gold, iron, lead, magnesium, plastic, steel, titanium, water, or custom. With custom, you can set these additional parameters for the appearance:

TABLE 2-60: PROPERTIES FOR CUSTOM APPEARANCE OF MATERIALS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
alpha	double	1	Diffuse and ambient color opacity
ambient	string[] (a color or custom)	custom	Ambient light color
customambient	double[]		RGB values for a custom ambient color
customdiffuse	double[]		RGB values for a custom diffuse color
customspecular	double[]		RGB values for a custom specular color
diffuse	string[] (a color or custom)	custom	Diffuse light color
fresnel	double	0.5	Reflectance at normal incidence for Cook-Torrance
lighting	phong cooktorrance simple	phong	Lighting model
noise	on off	off	Add noise to the appearance
noisefreq	double[]	1	Normal vector noise scale

TABLE 2-60: PROPERTIES FOR CUSTOM APPEARANCE OF MATERIALS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
noisescale	double[]	0	Normal vector noise scale
roughness	double	0.1	Surface roughness for Cook-Torrance lighting model
shininess	double	64	Specular exponent for Blinn-Phong lighting model
specular	string[] (a color or custom)	custom	Specular light color

`model.material().move(<tag>, <position>)` moves the material `<tag>` to the zero-indexed position `<position>` in the list.

For a material link, `model.material(<tag>).set("link", <linktag>);` creates a material link from a component to the global material with the tag in `<linktag>`. For example,

```
model.material().create("matlnk1", "Link", "comp1");
model.material("matlnk1").set("link", "mat1");
```

creates a material link in the component `comp1`, and then links it to the global material `mat1`. The material link can also have a selection; for example, `model.material("matlnk1").selection().all();`.

Property Groups

The following syntax examples describe the methods available for property groups added to a material (not available for material switches or material links).

`mm = model.material(<tag>).propertyGroup().create(<tag>, <descr>)` creates a new property group and stores it in the variable `mm`.

To define a method to set an output property of a material, use the following syntax (in this example you specify a density for the basic property group `def`):

```
model.material("mat1").propertyGroup("def").set("density", String[] arg);
```

where `arg` is the string value to be defined. For a list of available physical property names (such as `density`), see [Table 2-61](#).

`mm.addInput(<quantity>)` adds a new model input to the property group of the given quantity.

`mm.func(<ftag>)` returns the function object named `<ftag>`. For information about how to add and modify functions, see [model.func\(\)](#).

`mm.getString(<pname>)` returns the string value of the given parameter. If it is a vector or matrix quantity, the first value is returned.

`mm.getStringArray(<pname>)` returns the string array value of the given parameter. Matrix values are returned in a columnwise order.

`mm.hasParam(<pname>)` returns true if the parameter is defined by the property group.

`mm.info(<itag>)` returns the information object for the property group.

`mm.isOutput(<pname>)` returns true if the given parameter is an output property. For user-defined property groups the method returns true for all predefined material properties known to all physics interfaces. For the specialized property groups, it can also return true for other properties.

`mm.param()` returns a list of all parameters stored in the property group.

`mm.removeInput(<quantity>)` removes the given quantity from the list of model inputs.

`mm.set(<pname>, <expr>)` sets the expression for the given property. The expression can use local names for the properties, parameters, and model inputs. For vector and matrix properties, the expression can be string arrays of

varying size. Isotropic matrices only require one element or a string, diagonal matrices require three elements, and so forth. Vectors always require three elements.

`mm.size(<pname>)` returns the size of the stored parameter, which usually is 1-by-1, 3-by-1 or 3-by-3, but other sizes are supported.

`model.material(<tag>).propertyGroup()` returns a list of all property groups in the material.

`model.material(<tag>).propertyGroup(<tag>).info()`.

`create(<tag>,<descr>)` creates a new information object that can contain detailed information about this property group. This could, for example, be used by the Material Library to define the Phase/Condition and Orientation/Condition fields.

`mm.info(<itag>).title(<title>)` sets the title of the information object.

`mm.info(<itag>).title()` returns the title.

`mm.info(<itag>).body(<body>)` sets the body text of the information object.

`mm.info(<itag>).body()` returns the body text.



The term *material model* is sometimes used instead of *property group* in some contexts. The graphical user interface uses the term property group.

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Absorbed dose	absorbeddose
Absorption coefficient	absorption
Acceleration	acceleration
Activation energy	activationenergy
Angular acceleration	angularacceleration
Angular frequency	angularfrequency
Area	area
Bulk viscosity	bulkviscosity
Capacitance	capacitance
Catalytic activity	catalyticactivity
Characteristic acoustic impedance	acousticimpedance
Charge	charge
Coefficient of hygroscopic swelling	hygroscopicswellingcoefficient
Coefficient of thermal expansion	thermalexpansioncoefficient
Collisional power loss	inelasticpowerloss
Compliance	compliance
Compressibility of fluid	compressibility
Concentration	massconcentration
Concentration	concentration
Conductance	conductance
Corrected pressure	correctedpressure
Corrected velocity field	correctedvelocity
Current	current

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Current density	currentdensity
Current source	currentsource
Damping constant per unit area	dampingconstantperarea
Damping constant per unit length	dampingconstantperlength
Damping constant per unit volume	dampingconstantpervolume
Density	density
Diffusion coefficient	diffusion
Dimensionless	dimensionless
Dipole moment	dipolemoment
Dispersed phase volume fraction	dispersedphasevolume fraction
Displacement field	displacement
Displacement of shell normals	dimensionless_displacement
Dose equivalent	doseequivalent
Dynamic viscosity	dynamicviscosity
EEDF	eedf
Edge load	edgeloading
Effective gas density	effectivegasdensity
Effective plastic strain	effectiveplasticstrain
Elasticity	elasticity
Elastoresistive coupling	couplingelastoresistive
Electric displacement field	displacementfield
Electric field	electricfield
Electric permittivity	permittivity
Electric potential	electricpotential
Electrical conductivity	electricconductivity
Electrolyte conductivity	electrolyteconductivity
Electrolyte potential	electricpotentialionicphase
Electron density	electrondensity
Electron energy density	electronenergydensity
Electron mobility	electronmobility
Energy	energy
Energy density	energydensity
Entropy	entropydensity
External free energy	externalfreeenergy
Extinction coefficient	extinctioncoefficient
Face load	faceloading
Flow rate out from source per unit length	areaper time
Flow resistivity	pressure timeperarea
Fluid conductance	fluidconductance
Force density	forcedensity
Force load	force

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Force potential	forcepotential
Fowler-Nordheim coefficient	fowlernordheimcoefficient
Frequency	frequency
Frequency factor	frequencyfactor
Head	head
Heat capacity at constant pressure (molar)	molarheatcapacity
Heat capacity at constant pressure	heatcapacity
Heat source	powerdensity
Heat transfer coefficient	heattransfercoefficient
Henry's constant	henrysconstant
Hydraulic conductivity	hydraulicconductivity
Inductance	inductance
Initial curvature	planeangleperlength
Initial electron density	initialelectrondensity
Intensity (RMS)	intensity
Inward heat flux	heatflux
Isotropic structural loss factor	lossfactor
Kinematic viscosity	kinematicviscosity
Length	length
Level set variable	levelsetvariable
Line charge	linecharge
Line current source	linecurrentsource
Log mass fraction	logmassfraction
Log of electron density	logelectrondensity
Log of electron energy density	logelectronenergydensity
Logarithmic ratio	logarithmicratio
Logarithmic ratio per unit length	logarithmicratioperunitlength
Luminous intensity	luminousintensity
Magnetic field	magneticfield
Magnetic flux	magneticflux
Magnetic flux density	magneticfluxdensity
Magnetic permeability	permeability
Magnetic scalar potential	magneticscalarpotential
Magnetic vector potential	magneticvectorpotential
Mass	mass
Mass flow	massflow
Mass flux	massflux
Mass fraction	massfraction
Mass per unit area	massperarea
Mass per unit length	massperlength
Mass source	masssource

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Mass transfer coefficient	masstransfercoefficient
Mean electron energy	meanelectronenergy
Mean flow velocity potential	meanflowvelocitypotential
Mean molar mass	molarmass
Molar enthalpy	energyperamount
Molar flux	molarflux
Molar surface flux	molarsurfaceflux
Molar volume	molarvolume
Moment body load	torquepervol
Moment edge load	torqueperlength
Moment face load	torqueperarea
Natural logarithmic ratio	naturallogarithmicratio
Natural logarithmic ratio per unit length	naturallogarithmicratioperunitlength
Normal electron current density	normalelectroncurrentdensity
Normal ion current density	normalioncurrentdensity
Number density	numberdensity
Particle momentum	momentum
Particle position	position
Permeability	hydraulicpermeability
Phase field help variable	phasefieldhelpvariable
Phase field variable	phasefieldvariable
Piezoelectric coupling d (strain-charge)	couplingstraincharge
Piezoelectric coupling e (stress-charge)	couplingstresscharge
Piezoresistive coupling	couplingpiezoresistive
Plane angle	planeangle
Point current source	pointcurrentsource
Poiseuille coefficient	poiseuillecoefficient
Poisson's ratio	poissonsratio
Porosity	porosity
Power	power
Power flow	powerflow
Power per unit charge	powerpercharge
Power per unit length	powerpermeter
Power per unit mass	powerpermass
Pressure	pressure
Production rate	productionrate
Production/absorption coefficient	heatproduction
Radiative intensity	radiativeintensity
Radioactivity	radioactivity
Ratio of specific heats	ratioofspecificheat
Reaction rate	reactionrate

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Reciprocal area	reciprocalarea
Reciprocal initial interface distance	reciprocallength_i
Reciprocal wall distance	reciprocallength
Recombination rate (domain)	recombinationratedomain
Reduced electric field	reducedelectricfield
Reduced electron diffusivity	reduceddiffusivity
Reduced electron mobility	reducedmobility
Relative permeability	relpermeability
Relative permittivity	relpermittivity
Resistance	resistance
Resistivity	resistivity
Scattering coefficient	scattering
Secondary emission energy flux	energyflux
Secondary emission flux	particleflux
Seebeck coefficient	seebeckcoefficient
Solid angle	solidangle
Space charge density	spacechargedensity
Specific dissipation rate	specificdissipationrate
Specific energy	specificenergy
Speed of sound	soundspeed
Spring constant per unit area	springconstantperarea
Spring constant per unit length	springconstantperlength
Spring constant per unit volume	springconstantpervolume
Squared slip velocity	slipvelocity
Storage	storage
Strain energy per unit area	energydensityperarea
Strain energy per unit length	energydensityperlength
Strain reference temperature	strainreferencetemperature
Stress tensor	stress
Substance	substance
Surface capacitance	surfacecapacitance
Surface charge density	surfacechargedensity
Surface current density	surfacecurrentdensity
Surface electrical conductivity	surfaceconductivity
Surface emissivity	emissivity
Surface energy density	surfaceenergydensity
Surface magnetic current density	surfacemagneticcurrentdensity
Surface resistance	surfaceresistance
Surface site concentration	surfaceconcentration
Surface tension coefficient	surfacetensioncoefficient
Temperature	temperature

TABLE 2-61: AVAILABLE PHYSICAL QUANTITIES

PHYSICAL QUANTITY	NAME
Thermal conductivity	thermalconductivity
Time	time
Time change in pressure head	timechangeinpressurehead
Torque	torque
Total damping constant	dampingconstant
Total spring constant	springconstant
Trap density distribution (boundary)	trapdensityboundary
Trap density distribution (domain)	trapdensitydomain
Turbulent dissipation rate	turbulentdissipationrate
Turbulent kinetic energy	turbulentkineticenergy
Undamped turbulent kinematic viscosity	turbulentkinematicviscosity
Velocity field	velocity
Velocity potential	velocitypotential
Volume	volume
Volume fraction	volumefraction
Volume per time	volumevertime
Volumetric heat capacity	volumetricheatcapacity
Wave number	wavenumber
Wavelength	wavelength
Young's modulus	youngsmodulus

SEE ALSO

`model.func()`, `model.physics()`

model.mesh()

Meshing sequences.

SYNTAX

```
model.mesh().create(<tag>,<gtag>);
model.mesh().remove(<tag>);
model.mesh(<tag>).create(<ftag>,operation);
model.mesh(<tag>).feature(<ftag>).create(<ftag2>,operation);
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).feature().meshMethod
model.mesh(<tag>).current(<ftag>);
model.mesh(<tag>).feature(<ftag>).active(boolean);
model.mesh(<tag>).feature(<ftag>).isActive();
model.mesh(<tag>).feature().move(<ftag>,<position>);
model.mesh(<tag>).feature().remove(<ftag>);
model.mesh(<tag>).feature(<fgtag>).getAllowedPropertyValues(property);
model.mesh(<tag>).clear();
model.mesh(<tag>).geom();

model.mesh(<tag>).run(<ftag>);
model.mesh(<tag>).run();
model.mesh().run();

model.mesh(<tag>).current();
model.mesh(<tag>).feature(<ftag>).status();
model.mesh(<tag>).feature(<ftag>).message();

model.mesh(<tag>).meshGetMethod
model.mesh(<tag>).data().meshModificationMethod
model.mesh(<tag>).stat().meshStatisticsMethod

model.mesh(<tag>).export(<filename>);
```

DESCRIPTION

Creating and Deleting a Meshing Sequence

`model.mesh().create(<tag>,<gtag>)` creates a meshing sequence (or just mesh) for the geometry sequence `<gtag>` and assigns it the tag `<tag>`.

`model.mesh().remove(<tag>)` removes the meshing sequence `<tag>`.

Creating, Editing, and Deleting Features

`model.mesh(<tag>).create(<ftag>,operation)` adds a feature `<ftag>` of type `operation` to the meshing sequence `<tag>`, after the current feature.

`model.mesh(<tag>).feature(<ftag>).set(property,<value>)` sets the property `property` defined for the feature `<ftag>` to the value `<value>`. All data types listed in [Table 2-2](#) are supported; the applicable data types differ between the properties. String expressions can use parameters from `model.param()`.

`model.mesh(<tag>).feature(<ftag>).getType(property)` returns the value of a property in the feature `<ftag>`.

`model.mesh(<tag>).feature(<ftag>).selection().selMethod` manages the selection of the feature `<ftag>`. The available selection methods are described in [Selection Methods](#).

`model.mesh(<tag>).feature(<ftag>).selection(property).selMethod` manipulates the selection of the property `property`. The available selection methods are described in [Selection Methods](#).

`model.mesh(<tag>).feature(<ftag>).feature().meshMethod` manages the entity list for the feature `<ftag>`.

`model.mesh(<tag>).current(<ftag>)` sets the current feature to be `<ftag>`.

`model.mesh(<tag>).feature(<ftag>).active(false)` disables the feature `<ftag>`.

`model.mesh(<tag>).feature(<ftag>).active(true)` enables the feature <ftag>.

`model.mesh(<tag>).feature(<ftag>).isActive()` returns `true` if the feature <ftag> is enabled, and `false` otherwise.

`model.mesh(<tag>).feature(<ftag>)).`

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns `null`.

`model.mesh(<tag>).feature().move(<ftag>, <position>)` moves the feature <ftag> to the zero indexed position <position> in the sequence.

`model.mesh(<tag>).feature().remove(<ftag>)` removes the feature <ftag>.

`model.mesh(<tag>).clear()` removes all features from the sequence and clears the mesh.

Building Features

After each build operation, the current feature is set as the last of features that were built. The mesh is updated to be the mesh generated by these features.

`model.mesh(<tag>).run(<ftag>)` builds all features up to (and including) the feature <ftag>.

`model.mesh(<tag>).run()` builds all features.

`model.mesh().run()` builds all meshing sequences.

Getting Build Status

`String fTag = model.mesh(<tag>).current()` returns the tag of the current feature. If the current state is before the first feature, the empty string "" is returned.

`String status = model.mesh(<tag>).feature(<ftag>).status()` returns the status of the feature <ftag>. The status is `built`, `warning`, `needs_rebuild`, `edited`, or `error`.

`String msg = model.mesh(<tag>).feature(<ftag>).message()` returns the warning/error message of the feature <ftag>.

Getting and Setting Mesh Data

`model.mesh(<tag>).meshGetMethod` gets mesh data from the mesh <tag>. The available methods are described in [Accessing Mesh Data](#).

`model.mesh(<tag>).data().meshModificationMethods` are used to modify mesh data on a low level. You can access and modify individual elements. The available methods are described in [Accessing Mesh Data](#).

`model.mesh(<tag>).data().createMesh()` transfers the manipulated data into to the mesh <tag>.

Mesh Statistics

`model.mesh(<tag>).stat().selection().selMethod` can be used to select a number of geometric entities for which statistics is wanted. The available selection methods are described in [Selection Methods](#).

`model.geom(<tag>).stat().meshStatisticsMethod` returns mesh statistics about the selected geometric entities. The available methods are described in [Information and Statistics](#).

Exporting a Mesh to File

`model.mesh(<tag>).export(<filename>)` exports the mesh <tag> to an `mphbin` or `mphtxt` file.

Selection Methods

`selection.allGeom()` sets the selection to be the entire geometry (that is, all geometric entities).

`selection.remaining()` sets the selection to be the geometric entities that remains to be meshed when the feature is about to be built.

`selection.geom(<dim>).all()` sets all geometric entities in dimension `<dim>`.

`selection.geom(<dim>).set(<entities>)` sets the selection to be the geometric entities specified in the integer array `<entities>` in dimension `<dim>`.

`selection.geom(<dim>).add(<entities>)` adds the geometric entities specified in the integer array `<entities>` in dimension `<dim>` to the selection.

`selection.geom(<dim>).remove(<entities>)` removes the geometric entities specified in the integer array `<entities>` in dimension `<dim>` from the selection.

`selection.geom(<dim>).clear()` clears the selection.

To access the selections use:

`int[] dims = selection.dimension()` returns the geometric entity level in `dims[0]` for the entities in the selection. If `dims` is empty the selection defines the entire geometry.

`selection.isRemaining()` returns true if the selection specifies the remaining entities, otherwise false.

`selection.dom(<dim>)` returns the geometric entities in dimension `<dim>` for the selection.

Getting the Geometry Tag

`model.mesh(<tag>).geom()` returns the geometry tag. This can be useful when working with several geometries in the same model.

EXAMPLE

Create a 2D geometry by the union of a circle and square. Build a triangle mesh with `hmax = 0.1` in domains 1 and 3, and `hmax = 0.01` in domain 2.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.modelNode().create("mod1");
GeomSequence g = model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("c1", "Circle");
g.create("sq1", "Square");
g.create("uni1", "Union");
g.feature("uni1").selection("input").set(new String[]{"c1", "sq1"});
m.create("size1", "Size");
m.feature("size1").selection().geom(2).set(new int[]{1, 3});
m.feature("size1").set("hmax", "0.1");
m.create("size2", "Size");
m.feature("size2").selection().geom(2).set(new int[]{2});
m.feature("size2").set("hmax", "0.025");
m.create("ftri1", "FreeTri");
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.modelNode.create('mod1');
g = model.geom.create('geom1', 2);
m = model.mesh().create('mesh1', 'geom1');

g.create('c1', 'Circle');
g.create('sq1', 'Square');
g.create('uni1', 'Union');
g.feature('uni1').selection('input').set({'c1', 'sq1'});
m.create('size1', 'Size');
m.feature('size1').selection.geom(2).set([1, 3]);
m.feature('size1').set('hmax', '0.1');
m.create('size2', 'Size');
```

```

m.feature('size2').selection.geom(2).set(2);
m.feature('size2').set('hmax', '0.025');
m.create('ftri1', 'FreeTri');
m.run;

```

SEE ALSO

[model.geom\(\)](#)

model.modelNode()

Model nodes.

SYNTAX

```

model.modelNode().create(<tag>);
model.modelNode().create(<tag>, <basetag>);
model.modelNode().create(<tag>, <type>);

model.modelNode(<tag>).scope();
model.modelNode(<tag>).baseSystem();
model.modelNode(<tag>).baseSystem(<system>);
model.modelNode(<tag>).sorder();
model.modelNode(<tag>).sorder(<stype>);
model.modelNode(<tag>).defineLocalCoord();
model.modelNode(<tag>).defineLocalCoord(boolean);

```

DESCRIPTION

`model.modelNode(<tag>)` represents a model node in the model tree.

`model.modelNode().create(<tag>)` creates a model node with the given tag.

`model.modelNode().create(<tag>, <basetag>)` creates an extra dimension model node with the tag `<tag>` associated to the base model node `<basetag>`.

`model.modelNode().create(<tag>, <type>)` creates a model node with the tag `<tag>` of one of the following types, set as the string `<type>`: `Component`, for a normal geometry component; `ExtraDim`, for an extra dimension; or `MeshComponent`, for a mesh component. For example, to create a mesh component:

```

model.modelNode().create("mcomp1", "MeshComponent");
model.geom().create("mgeom1", 3);
model.mesh().create("mpart1", "mgeom1");

```

`model.modelNode(<tag>).scope()` returns the fully qualified scope name.

`model.modelNode(<tag>).baseSystem(<system>)` use the given base system as unit system for the model node. This overrides the global unit system specified for the entire model object. To use global system again, set the base system of the model node to `null`.

`model.modelNode(<tag>).sorder()` returns the geometry shape order used for the model node and its descendants.

`model.modelNode(<tag>).sorder(<stype>)` Sets the geometry shape order. Allowed values are `automatic`, `linear`, `quadratic`, `cubic`, `quartic`, and `quintic`, and the default is `automatic`. With `automatic` shape order the physics interfaces under the model node decide the most optimum shape order.

`model.modelNode(<tag>).defineLocalCoord()` returns true if local coordinate variables exist. By default, this is the case, except for model components created before version 5.0.

`model.modelNode(<tag>).defineLocalCoord(boolean)` sets a flag that determines whether local coordinate variables exist.

NOTE

You assign a model node to a physics interface, a geometry, and so forth using the methods

```
model.geom(<tag>).model(<mtag>);  
model.physics(<tag>).model(<mtag>);
```

The methods

```
model.geom(<tag>).model();  
model.physics(<tag>).model();
```

return the tag of the current model node.

The last created model node is the default model node for any geometry or physics interface that you create. If no model node exists when you create a geometry or a physics interface, a default model node with tag `mod1` is automatically created for you.

EXAMPLE

Create a model node and assign it to a geometry and an analytic function.

Code for use with Java

```
model.modelNode().create("mod1");  
  
model.geom().create("geom1", 3);  
model.geom("geom1").model("mod1");  
  
model.func().create("an1", "Analytic");  
model.func("an1").model("mod1");
```

Code for use with MATLAB

```
model.modelNode.create('mod1');  
  
model.geom.create('geom1', 3);  
model.geom('geom1').model('mod1');  
  
model.func.create('an1', 'Analytic');  
model.func('an1').model('mod1');
```

COMPATIBILITY

The `model.modelNode().createExtraDim(<tag>)` method has been deprecated in version 5.2. Use `model.modelNode().create(<tag>, "ExtraDim");` instead.

model.multiphysics()

Add multiphysics features to this feature container.

SYNTAX

```
model.multiphysics().create(<tag>,...);
```

DESCRIPTION

`model.multiphysics().create(<tag>,<coupling>,<geom>,<sdim>)`; adds a multiphysics coupling with the tag and the name to a geometry and a space dimension. Use -1 as the space dimension to indicate a coupling valid in the entire model.

EXAMPLE

In a model with a Solid Mechanics interface and a Heat Transfer in Solids interface, add a Thermal Expansion multiphysics coupling on the domain level and a Temperature Coupling on a model-wide level in a geometry `geom1`:

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom.create("geom1", 3);
model.multiphysics().create("te1", "ThermalExpansion", "geom1", 3);
model.multiphysics().create("tc1", "TemperatureCoupling", "geom1", -1);
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('geom1', 3);
model.multiphysics.create('te1', 'ThermalExpansion', 'geom1', 3);
model.multiphysics.create('tc1', 'TemperatureCoupling', 'geom1', -1);
```

model.ode()

Create global equations (ODEs and DAEs).

SYNTAX

```
model.ode().create(<tag>);
model.ode(<tag>).state(<statelist>);
model.ode(<tag>).state(<pos>, <state>);
model.ode(<tag>).ode(<state>, <equation>);
model.ode(<tag>).descr(<state>, <descr>);
model.ode(<tag>).weak(<wlist>);
model.ode(<tag>).weak(<pos>, <wexpr>);
model.ode(<tag>).discrete(<boolean>);
model.ode(<tag>).valueType(<prop>);
```

```
model.ode(<tag>).state();
model.ode(<tag>).ode(<state>);
model.ode(<tag>).descr(<state>);
model.ode(<tag>).weak();
model.ode(<tag>).discrete();
model.ode(<tag>).valueType();
```

DESCRIPTION

`model.ode(<tag>)` returns a global equation (or an ODE or a DAE) with tag `<tag>`.

`model.ode().create(<tag>)` creates an global equation with tag `<tag>`.

`model.ode(<tag>).state(<statelist>)` sets the states of the global equation tagged `<tag>` according to the list `<statelist>`.

`model.ode(<tag>).state(<pos>, <state>)` edits the state at position `<pos>` in the state vector for the global equation `<tag>`.

`model.ode(<tag>).ode(<state>, <equation>)` sets the equation for the state `<state>`. If the state variable has not previously been added using `model.ode(<tag>).state(<statelist>)` then `<state>` is appended to the list of state variables.

`model.ode(<tag>).descr(<state>, <descr>)` sets the description for the state `<state>`.

`model.ode(<tag>).weak(<wlist>)` set weak equations. `<wlist>` is a list of weak expressions.

`model.ode(<tag>).weak(<pos>, <wexpr>)` sets the weak expression at position `<pos>` in the list of weak expressions.

`model.ode(<tag>).state()` returns the state variables as a string array.

`model.ode(<tag>).ode(<state>)` returns the global equation for the state variable `<state>` as a string.

`model.ode(<tag>).descr(<state>)` returns the description of the state variable `<state>` as a string.

`model.ode(<tag>).weak()` returns the weak equations as a string vector of weak expressions.

`model.ode(<tag>).discrete(true)` specifies that the global equation contains event states.

`model.ode(<tag>).valueType(prop)` specifies value type as `real` or `complex` when splitting of complex variables in real and imaginary parts has been turned on.

`model.ode(<tag>).valueType()` returns the value type.

EXAMPLE

Define a global equations with the variables u and w , the ODEs $u_{t+1} = 0$ and $v_{t+1} = 0$, where the subscript t indicates the derivative with respect to time. Also define a weak expression $\text{test}(u) \cdot v$.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.ode().create("ode1");
model.ode("ode1").ode("u","ut+1");
model.ode("ode1").ode("v","vt-1");
model.ode("ode1").weak(new String[]{"test(u)*v"});
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.ode.create('ode1');
model.ode('ode1').ode('u','ut+1');
model.ode('ode1').ode('v','vt-1');
model.ode('ode1').weak({'test(u)*v'});
```

SEE ALSO

[model.init\(\)](#), [model.solverEvent\(\)](#)

model.opt()

Optimization interface.

SYNTAX

```
model.opt().objective().create(<tag>,type)
model.opt().objective(<tag>).set(property,<value>)
```

```
model.opt().constr().create(<tag>)
model.opt().constr(<tag>).etc
```

```
model.opt().gconstr().create(<tag>)
model.opt().gconstr(<tag>).constr(<constrExpr>)
model.opt().gconstr(<tag>).lbound(<lboundExpr>)
model.opt().gconstr(<tag>).ubound(<uboundExpr>)
```

DESCRIPTION

The purpose of `model.opt` is to manage information relating to optimization and sensitivity analysis. Most of the fields under `model.opt` are read and interpreted directly by the optimization and sensitivity solvers. They never affect the result of other solvers.

`model.opt().objective().create(<tag>,type)` adds an objective function of the specified type. The supported types are `Global` and `LeastSquares`.

`model.opt().objective(<tag>).set(property,<value>)` sets an objective function property. Objective functions of type `Global` support the single property `expr`, which takes a globally defined expression as value. Allowed properties for objectives of type `LeastSquares` are described below.

`model.opt().objective(<tag>).selection().named(<seltag>)` assigns the objective function to the named selection `<seltag>`.

`model.opt().objective(<tag>).selection().set(...)` defines a local selection that assigns the objective function to geometric entities. For a complete list of methods available under `selection()`, see `model.selection()`. Only objective functions of type `LeastSquares` require a selection. See further below.

`model.opt().constr().create(<tag>)` adds a pointwise (mesh-based) constraint on the control variables. The syntax is shared with `model.constr()` with the exception that the `ctype` parameter expects values `constr`, `lbound`, and `ubound` for constraint, lower bound, and upper bound, respectively.

`model.opt().gconstr().create(<tag>)` registers a global constraint with the optimization solvers. Such constraints consist of a globally available expression, which can depend both on optimization variables and on the forward PDE solution, together with likewise global expressions for lower and upper bound.

`model.opt().gconstr(<tag>).constr(<constrExpr>)` specifies a global constraint expression.

`model.opt().gconstr(<tag>).lbound(<lboundExpr>)` sets lower bound for the constraint.

`model.opt().gconstr(<tag>).ubound(<lboundExpr>)` sets upper bound for the constraint.

Least-Squares Objective Functions

[[not sure about this heading level. No doc marker?]]

Least-squares objective functions are specified in terms of measured values, stored on file, together with information about how corresponding expressions can be evaluated for the current control variable values. An overview of the allowed properties is given in the table below.

TABLE 2-62: PROPERTIES FOR OBJECTIVE FUNCTION TYPE LEASTSQUARES

PROPERTY	VALUE	DESCRIPTION
filename	string	Full path of the measurement data file
paramnames	string[]	Parameters used in the experiment
paramexprs	string[]	Values of the given parameters
columntypes	string[]	List of column type indicators
columnexprsweights	string[]	Column contribution weights
columnexprs	string[]	Measurement expressions

In principle, you must specify the following for each measured value:

- To which experiment the value belongs and parameters for that experiment
- Which expression to evaluate
- Where the expression must be evaluated
- For which time or parameter value the evaluation must be performed

Each *experiment* corresponds to a solution of the forward problem with a given set of parameter values. In practice, measurements for each experiment must be stored in a separate file, and specified as a separate `LeastSquares` objective feature where you give the full path of the measurement data file in the `filename` property. Parameters specified in the `paramnames` property are given the values specified using `paramexprs` property during the forward solution. One forward solution is performed for each unique set of parameter names and values.

The required measurement data file format is row- and column-oriented. Entries on each row must be separated by commas or semicolons, while rows are separated by line feeds. Use the `columntypes` property to specify the content of each column, in the order that they appear in the data file, according to the following table:

TABLE 2-63: ALLOWED COLUMN TYPES

TYPE	COLUMN CONTENTS
time	Actual measurement times
param	Actual parameter values
coord	Actual measurement coordinates
value	Measured values
none	Ignored column

Columns of type `time` are only allowed for transient problems. The measurements on the same row are assumed to be made at the specified time. Forward model values are interpolated to the given times. There must only be one column of type `time`, and it requires no further parameters.

Columns of type `param` contain parameter values for which the measurements on the same row have been made, and for which the forward problem must be solved. A data file can contain multiple parameter columns. Corresponding parameter names must be given in the `columnnames` property.

Columns of type `coord` contain global coordinates where the measurements on the same row have been made. The coordinate columns must be coupled to a coordinate variable by specifying the coordinate variable name in the `columnnames` property for the given column and the frame tag `spatial`, `material`, `mesh` or `geometry` in the `columnexprs` property. For example, in a 3D model, you need three columns of type `coord` with `columnnames` entries `x`, `y`, and `z`, respectively.

A `value` column contains measured data. For each `value` column, a corresponding expression to be evaluated must be specified in the `columnexprs` property. Entries in `value` columns are interpreted as real numbers when possible. Anything else, including for example hash marks (`#`) and the literal strings `nan`, `Nan`, `NaN` and `NAN` is interpreted as an illegal value which is excluded from the least squares objective function evaluation. A weight for the objective contribution from a column, multiplying the squared difference between the measured value and the expression, can be specified as a positive globally expression that can be evaluated using the `columnexprsweights` property. To exclude a measurement from a comma-separated file, you can also simply leave a value column empty.

Columns of type `none` can be used to exclude columns from the data file.

Coordinates are interpreted as global in the context of the objective feature's selection. This means that the value column expressions are evaluated at the points within the selection that best match the given coordinates. If the interpolation fails for some point because its coordinates lie too far outside the selection, the corresponding value is ignored.

model.pair()

Create and define identity pairs and contact pairs.

SYNTAX

```
model.pair().create(<tag>,type,<gtag>);
model.pair(<tag>).type(type);
model.pair(<tag>).type();
model.pair(<tag>).pairName(<pname>);
model.pair(<tag>).pairName();
model.pair(<tag>).source().selMethod;
model.pair(<tag>).source().named(<seltag>);
model.pair(<tag>).source().named();
model.pair(<tag>).destination().selMethod;
model.pair(<tag>).destination().named(<seltag>);
model.pair(<tag>).destination().named();
model.pair(<tag>).swap();
model.pair(<tag>).srcFrame(<frame>);
model.pair(<tag>).srcFrame();
model.pair(<tag>).dstFrame(<frame>);
model.pair(<tag>).dstFrame();
model.pair(<tag>).hasAutoSelection();
model.pair(<tag>).manualSelection(manual);
model.pair(<tag>).manualSelection();
model.pair(<tag>).searchMethod(method);
model.pair(<tag>).searchMethod();
model.pair(<tag>).manualDist(manual);
model.pair(<tag>).manualDist();
model.pair(<tag>).searchDist(<dist>);
model.pair(<tag>).searchDist();
model.pair(<tag>).opName(src2dst);
model.pair(<tag>).mphOpName(src2dst);
model.pair(<tag>).gapName(src2dst);
model.pair(<tag>).active(boolean);
model.pair(<tag>).isActive();
model.pair().remove(<tag>);
```

DESCRIPTION

`model.pair().create(<tag>,type,<gtag>)` creates a pair with tag `<tag>` in the geometry with tag `<gtag>`. The type `type` is either `Contact` or `Identity`. `model.pair(<tag>).type()` returns the pair type as a string.

`model.pair(<tag>).type(type)` changes the pair type.

`model.pair(<tag>).pairName(<pname>)` sets the pair name, which is used as a suffix in operator names and variable names. By default, the pair name is the same as the tag. `model.pair(<tag>).pairName()` returns the pair name.

`model.pair(<tag>).source().named(<seltag>)` assigns the source boundaries to the named selection `<seltag>`.

`model.pair(<tag>).source().set(...)` defines a local selection that assigns the source boundaries to geometric entities. For a complete list of methods available under `model.pair(<tag>).source()`, see [model.selection\(\)](#).

`model.pair(<tag>).destination().named(<seltag>)` assigns the destination boundaries to the named selection `<seltag>`.

`model.pair(<tag>).destination().set(...)` defines a local selection that assigns the destination boundaries to geometric entities. For a complete list of methods available under `model.pair(<tag>).destination()`, see [model.selection\(\)](#).

`model.pair(<tag>).swap()` swaps the source and destination selections.

`model.pair(<tag>).srcFrame(<frame>)` and `model.pair(<tag>).dstFrame(<frame>)` sets the source and destination frames for the identity mapping. The argument `<frame>` can have the values `spatial`, `material`, or `mesh`. The default is `spatial`. These frames are only used for identity pairs. `model.pair(<tag>).srcFrame()` and `model.pair(<tag>).dstFrame()` returns the frame tags.

`model.pair(<tag>).hasAutoSelection()` returns true if the contact pair was created automatically, using the create pairs check box in the finalize geometry node.

`model.pair(<tag>).manualSelection(manual)` enables or disables manual control of the selections for a pair that was created automatically. `model.pair(<tag>).manualSelection()` returns true if manual control of selections is enabled, and false otherwise.

`model.pair(<tag>).searchMethod(method)` sets the search method for a contact pair. The argument *method* can be `fast` or `direct`. The default is `fast`. `model.pair(<tag>).searchMethod()` returns the search method

`model.pair(<tag>).manualDist(manual)` enables or disables manual control of the search distance for a contact pair. The argument *manual* is Boolean. The default value `false` means that the search distance is automatically computed based on the size of the geometry. `model.pair(<tag>).manualDist()` returns true if manual control of search distance is enabled, and false otherwise.

`model.pair(<tag>).searchDist(<dist>)` sets the search distance for a contact pair, when manual control of the search distance is enabled. The argument *<dist>* is a string whose default unit is the geometry's length unit. The default is `1e-2`. `model.pair(<tag>).searchDist()` returns the search distance as a string.

`model.pair(<tag>).opName(src2dst)` returns the name of the operator transferring an expression from source to destination (if *src2dst*=true) or from destination to source (if *src2dst*=false).

`model.pair(<tag>).mphOpName(src2dst)` returns the name of the multiphysics operator transferring an expression from source to destination (if *src2dst*=true) or from destination to source (if *src2dst*=false). When the test operator is applied on this operator, it does not give any contribution (reaction force) for the structural mechanics interfaces' degrees of freedom due to the variable point mapping. These operators are available only for contact pairs.

`model.pair(<tag>).gapName(src2dst)` returns the name of the geometric gap variable seen from the destination (if *src2dst*=true) or seen from the source (if *src2dst*=false). These variables are available only for contact pairs.

`model.pair(<tag>).active(boolean)` enables or disables the pair.

`boolean enabled = model.pair(<tag>).isActive()` returns true if the pair is enabled, and false otherwise.

`model.pair().remove(<tag>)` deletes the pair.

EXAMPLE

Create a contact pair in the geometry `geom1` with source boundaries 4 and 6 and destination boundaries 10 and 12.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("blk1", "Block");
g.create("blk2", "Block");
g.feature("blk2").set("pos", new String[]{"0.5", "0.5", "1"});
g.feature("fin").name("Form Assembly");
g.feature("fin").set("action", "assembly");
g.feature("fin").set("imprint", true);
g.feature("fin").set("createpairs", false);
g.run();

model.pair().create("p1", "Contact", "geom1");
model.pair("p1").source().set(new int[]{4, 6});
model.pair("p1").destination().set(new int[]{10, 12});
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom().create('geom1', 3);
```

```

g.create('blk1', 'Block');
g.create('blk2', 'Block');
g.feature('blk2').set('pos', {'0.5', '0.5', '1'});
g.feature('fin').name('Form Assembly');
g.feature('fin').set('action', 'assembly');
g.feature('fin').set('imprint', true);
g.feature('fin').set('createpairs', false);
g.run;

model.pair.create('p1', 'Contact', 'geom1');
model.pair('p1').source().set([4, 6]);
model.pair('p1').destination.set([10, 12]);

```

model.param()

Add, define, and remove parameters.

SYNTAX

```

model.param().set(<param>, <expr>);
model.param().set(<param>, <expr>, <descr>);
model.param().descr(<param>, <descr>);
model.param().remove(<param>);

```

```

model.param().varnames();
model.param().get(<param>);
model.param().descr(<param>);
model.param().evaluate(<param>);
model.param().evaluateComplex(<param>);
model.param().evaluateUnit(<param>);
model.param().loadFile(tempFile, ...);
model.param().saveFile(tempFile, ...);

```

DESCRIPTION

`model.param()` is a collection of model parameters.

`model.param().set(<param>, <expr>)` defines the parameter `<param>` as `<expr>`.

`model.param().set(<param>, <expr>, <descr>)` defines the parameter `<param>` as `<expr>` and assigns it the description `<descr>`.

`model.param().descr(<param>, <descr>)` sets the description for the parameter `<param>`.

`model.param().remove(<param>)` removes the parameter `<param>`.

`model.param().varnames()` returns the names of all parameters as a string array.

`model.param().get(<param>)` returns the parameter value as a string.

`model.param().descr(<param>)` returns the parameter description as a string.

`model.param().evaluate(<param>)` evaluates the value of the parameter `<param>` as a double real-valued floating-point value. For complex-valued parameters, use the `evaluateComplex` method instead.

`model.param().evaluateComplex(<param>)` evaluates the value of the parameter `<param>` as a double floating-point array with the real and imaginary part of a complex-valued parameter.

`model.param().evaluateUnit(<param>)` returns the unit of the parameter `<param>` if defined. It returns `null` if the parameter has no unit defined, or if the model does not use any unit system.

For `model.param().loadFile()` and `model.param().saveFile()`, see [The loadFile and saveFile Methods](#).

EXAMPLE

Define the parameter `c` in terms of another parameter `a` and then remove `c`.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.param().set("c", "1+a");
model.param().remove("c");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.param.set('c','1+a');
model.param.remove('c');
```

SEE ALSO

[model.variable\(\)](#)

model.physics()

Create and define properties for a physics interface.

SYNTAX

```
model.physics().create(<tag>,physint);
model.physics().create(<tag>,physint,<geomtag>);
model.physics().create(<tag>,physint,<geomtag>,<varnames>);
model.physics(<tag>).model(<mtag>);
model.physics(<tag>).field(fieldname).fieldname(<namelist>);
model.physics(<tag>).field(fieldname).fieldname(<pos>,<name>);
model.physics(<tag>).prop(propname).set(property,<value>);
model.physics(<tag>).create(<ftag>,feature);
model.physics(<tag>).create(<ftag>,feature,<dim>);
model.physics(<tag>).feature(<ftag>).create(<ftag2>,feature);
model.physics(<tag>).feature(<ftag>).create(<ftag2>,feature,<dim>);
model.physics(<tag>).feature(<ftag>).set(property,<value>);
model.physics(<tag>).feature().move(<ftag>,<position>);

model.physics(<tag>).feature(<ftag>).feature();
model.physics(<tag>).feature(<ftag>).feature(<ftag2>);
model.physics(<tag>).feature(<ftag>).featureInfo();
model.physics(<tag>).feature(<ftag>).featureInfo("info");
feature = model.physics(<tag>).feature(<ftag>);
feature.featureInfo("info").set(variable,<value>);
feature.featureInfo("info").getInfoTable(id);

model.physics(<tag>).model();
model.physics(<tag>).field(fieldname).fieldname();
model.physics(<tag>).scope();
model.physics(<tag>).geom();
model.physics(<tag>).prop(propname).getType(<pname>);
model.physics(<tag>).prop(propname).param();
model.physics(<tag>).prop(propname).getAllowedPropertyValues(property);
model.physics(<tag>).feature(<ftag>).getAllowedPropertyValues(property);
model.physics(<tag>).feature(<ftag>).getType(<pname>);
model.physics(<tag>).feature(<ftag>).param();
model.physics(<tag>).diagram(<dtag>).getAllowedPropertyValues(property);

model.physics(<tag>).feature(<tag>).set(String pname, int value);
model.physics(<tag>).feature(<tag>).set(String pname, int pos, int value);
model.physics(<tag>).feature(<tag>).set(String pname, int pos, int[] value);
model.physics(<tag>).feature(<tag>).set(String pname, int pos1, int pos2, int value);
model.physics(<tag>).feature(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).feature(<tag>).
    setIndex(String name, String value, int firstIndex, int secondIndex);
model.physics(<tag>).feature(<tag>).setIndex(String name, String[] value, int index);
model.physics(<tag>).feature(<tag>).setIndex(String name, double value, int index);
model.physics(<tag>).feature(<tag>).
    setIndex(String name, double value, int firstIndex, int secondIndex);
model.physics(<tag>).feature(<tag>).setIndex(String name, double[] value, int index);
model.physics(<tag>).feature(<tag>).setIndex(String name, int value, int index);
model.physics(<tag>).feature(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).feature(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).feature(<tag>).
    setIndex(String name, int value, int firstIndex, int secondIndex);
model.physics(<tag>).feature(<tag>).setIndex(String name, int[] value, int index);

model.physics(<tag>).prop(<tag>).set(String pname, int value);
model.physics(<tag>).prop(<tag>).set(String pname, int pos, int value);
model.physics(<tag>).prop(<tag>).set(String pname, int pos, int[] value);
model.physics(<tag>).prop(<tag>).set(String pname, int pos1, int pos2, int value);
model.physics(<tag>).prop(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).prop(<tag>).
    setIndex(String name, String value, int firstIndex, int secondIndex);
model.physics(<tag>).prop(<tag>).setIndex(String name, String[] value, int index);
model.physics(<tag>).prop(<tag>).setIndex(String name, double value, int index);
model.physics(<tag>).prop(<tag>).
```



```

        setIndex(String name, double value, int firstIndex, int secondIndex);
model.physics(<tag>).prop(<tag>).setIndex(String name, double[] value, int index);
model.physics(<tag>).prop(<tag>).setIndex(String name, int value, int index);
model.physics(<tag>).prop(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).prop(<tag>).setIndex(String name, String value, int index);
model.physics(<tag>).prop(<tag>).
        setIndex(String name, int value, int firstIndex, int secondIndex);
model.physics(<tag>).prop(<tag>).setIndex(String name, int[] value, int index);

```

The `set()` methods index/position arguments are 1-based. The `setIndex()` methods index/position arguments are 0-based.

DESCRIPTION

`model.physics().create(<tag>,physint)` returns a physics interface (or just physics).

`model.physics().create(<tag>,physint)` or `model.physics().create(<tag>,physint,<geomtag>)` adds physics to the model and initializes it with defaults. The *physint* argument specifies which physics to create. There can be several different values of *physint* which create the same internal physics interface class, but which set different defaults. The constructor without the *<geomtag>* argument can only be used (and should be used) by 0D (space-independent) interfaces.

`model.physics().create(<tag>,physint,<geomtag>,<varnames>)` adds an interface with the field variable names *<varnames>*. Only interfaces supporting a varying number of field variables considers this argument. Providing the variable names in the `create` method rather than changes them afterward using `model.physics(<tag>).field(fieldname).fieldname(<namelist>)` ensures that the default features are correct.

`model.physics(<tag>).model(<mtag>)` sets the model node of a 0D interface. For interfaces with a geometry, the model is deduced from the geometry.

`model.physics(<tag>).field(fieldname).fieldname(<namelist>)` sets a name of a dependent variable. The entity *fieldname* (which could be, for example, temperature, x-velocity, electric field) specifies which dependent variable to set the name for. The available fields are provided by the physics interface. The argument *<namelist>* can be a list of names for physics interfaces supporting an arbitrary number of dependent variables. The physics interfaces provide default names for the dependent variables.

`model.physics(<tag>).field(fieldname).fieldname(<pos>,<name>)` changes the name at position *<pos>* in the list of field names.

`model.physics(<tag>).selection().named(<seltag>)` specifies that the physics interface is active on the named selection *<seltag>*.

`model.physics(<tag>).selection().set(...)` defines a local selection that makes the physics active on the selection's geometric entities. For a complete list of methods available under `selection()`, see [Selections](#). The selection must apply to the physics interface's maximum geometry level. The `create()` method makes the physics interface active in all domains. 0D interfaces are always active globally and do not support these methods.

`model.physics(<tag>).prop(propname).set(pname,<value>)` sets the value of some property parameter. All string types listed in [Table 2-2](#) are supported.

`model.physics(<tag>).create(<ftag>,feature)` adds a new feature instance to the physics interface and initializes the feature with defaults. The available features are given by the physics interface.

`model.physics(<tag>).create(<ftag>,feature,<dim>)` adds a new feature instance to the physics interface and initializes the feature with defaults. The feature is assigned to the domain level *<dim>*. Use this constructor for features which can be applied to more than one domain level. The constructor without the *<dim>* argument assigns the feature to the highest domain level, which the feature supports.

`model.physics(<tag>).feature(<ftag>).set(pname,<value>)` sets a parameter value. All string types listed in the section [Table 2-2](#) are supported.

`model.physics(<tag>).feature(<ftag>).selection().named(<seltag>)` assigns the physics feature to the named selection `<seltag>`. `model.physics(<tag>).selection().set(...)` defines a local selection that assigns the physics feature to geometric entities. For a complete list of methods available under `selection()`, see [Selections](#). 0D features need no domain selection.

`model.physics(<tag>).feature().move(<ftag>,<position>)` moves the feature `<ftag>` to the zero indexed position `<position>` in the list. A feature cannot be moved before a default feature and the default features cannot be moved.

`model.physics(<tag>).create(<itag>,"init")` creates an initial value feature, using the reserved feature ID `init`.

`model.physics(<tag>).feature(<itag>).set(varname,<value>)` specifies an initial value. The variable names are the field variables. For wave problems, the time derivatives of the field variables are also included in the list of variables.

`model.physics(<tag>).model()` returns the model node of the interface.

`model.physics(<tag>).field(fieldname).fieldname()` returns the field names as a string array.

`model.physics(<tag>).scope()` returns the fully qualified scope name.

`model.physics(<tag>).geom()` returns the geometry tag as a string.

`model.physics(<tag>).selection().named()` returns the selection tag as a string.

`model.physics(<tag>).selection().getType()` returns domain information. See [Selections](#) for available methods.

`model.physics(<tag>).prop(propname).getType(pname)` returns the parameter value. See [getType\(\)](#) for available methods.

`model.physics(<tag>).prop(propname).param()` returns the parameter names as a string array.

`model.physics(<tag>).feature(<ftag>).getType(<pname>)` returns the parameter value. See [getType\(\)](#) for available methods.

`model.physics(<tag>).feature(<ftag>).param()` returns the parameter names as a string array.

`model.physics(<tag>).feature(<ftag>).selection().named()` returns the selection tag as a string array.

`model.physics(<tag>).feature(<ftag>).feature()` returns the list of feature attributes. This list supports the same methods as `model.physics(<tag>).feature()`.

`model.physics(<tag>).feature(<ftag>).feature(<ftag2>)` returns the feature attribute `<ftag2>`. The feature attributes support the same methods as `model.physics(<tag>).feature(<ftag>)`.

`model.physics(<tag>).feature(<ftag>).featureInfo()` returns a list of info objects.

`model.physics(<tag>).feature(<ftag>).featureInfo("info")` returns the info object that contains information about the variables, weak expressions, and constraints that a feature generates. The `model.physics(<tag>)` and `model.coordSystem(<tag>)` objects also have this list that you access with `model.physics(<tag>).featureInfo("info")`. These objects do not support the `set` method, which only works for the object `model.physics(<tag>).feature(<ftag>)`.

`model.physics(<tag>).feature(<ftag>).getAllowedPropertyValues(property)`, `model.physics(<tag>).prop(propname).getAllowedPropertyValues(property)`, and

`model.physics(<tag>).diagram(<dtag>).getAllowedPropertyValues(property)` return the set of allowed values for a property if the set is a finite set of strings; otherwise, they return null.

`feature.featureInfo("info").set(variable,<value>)` locks the named variable to the given expression. The expression must be given as a string array.

`feature.featureInfo("info").getInfoTable(id)` returns a table that lists all information about a certain table id. The supported IDs are Expression, Shape, Weak, and Constraint.

EXAMPLE

This example creates an Electrostatics interface. It sets boundaries 3 and 8 to the ground potential and assigns the electric potential of 1 V at boundary 4.

When the physics interface is created a couple of default features are automatically added. One of them is the Charge Conservation feature, which has the tag `ccn1`. The relative permittivity of this feature is set to 1.

Code for use with Java

```
model.physics().create("es","Electrostatics","geom1");
model.physics("es").create("gnd1", "Ground", 2);
model.physics("es").feature("gnd1").selection().set(new int[]{3, 8});
model.physics("es").create("pot1", "ElectricPotential", 2);
model.physics("es").feature("pot1").selection().set(new int[]{4});
model.physics("es").feature("pot1").set("V0", "1");
model.physics("es").feature("ccn1").set("epsilon_r_mat", "userdef");
model.physics("es").feature("ccn1").set("epsilon_r", "1");
```

Code for use with MATLAB

```
model.physics.create('es','Electrostatics','geom1');
model.physics('es').create('gnd1','Ground',2);
model.physics('es').feature('gnd1').selection().set([3,8]);
model.physics('es').create('pot1','ElectricPotential',2);
model.physics('es').feature('pot1').selection.set(4);
model.physics('es').feature('pot1').set('V0','1');
model.physics('es').feature('ccn1').set('epsilon_r_mat','userdef');
model.physics('es').feature('ccn1').set('epsilon_r','1');
```

COMPATIBILITY

From version 4.3 the methods

```
model.physics(<tag>).feature(<ftag>).params();
model.physics(<tag>).prop(propname).params();
```

are deprecated and replaced by the methods

```
model.physics(<tag>).feature(<ftag>).param();
model.physics(<tag>).prop(propname).param();
```

SEE ALSO

[model.material\(\)](#), [model.study\(\)](#)

model.probe()

Create and defined properties for probes, which you can add to a model to monitor some quantity (real or complex-valued number) during a time-dependent, frequency-domain, or parametric simulation.

SYNTAX

```
model.probe().create(<tag>, type);
model.probe(<tag>).model(<mtag>);
model.probe(<tag>).set(property,<value>);
model.probe(<tag>).create(<etag>, etype);
model.probe(<tag>).feature(<etag>).set(eproperty,<evalue>);
model.probe(<tag>).feature(<etag>).getAllowedPropertyValues(property);
model.probe(<tag>).genResult(String sol)
```

DESCRIPTION

`model.probe().create(<tag>, type)` creates a probe of type `type` with tag `<tag>`.

`model.probe(<tag>).model(<mtag>)` sets the model node to `<mtag>`.

`model.probe(<tag>).set(property,<value>)` set `property` to `<value>`.

`model.probe(<tag>).selection(...)` sets the selection for the probe. This is possible for the probes of the types Domain, Boundary, and Edge.

`model.probe(<tag>).create(<etag>, etype)` creates a point probe expression of type `etype` and tag `<tag>`.

`model.probe(<tag>).feature(<etag>).set(eproperty,<evalue>)` sets the property `eproperty` on the point probe expression `<etag>`.

`model.probe(<tag>).feature(<etag>).getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

`model.probe(<tag>).genResult(String sol)` where `sol` is a solver sequence tag. This function prepares for using a probe while solving or during postprocessing. The command is invoked automatically when a solver or study is run from the COMSOL Desktop for all active probes but needs to be invoked explicitly when run through the API. The function `genResult(String sol)` sets up result features for evaluating the corresponding probe using the solver sequence `sol`. If null is used, the default solver sequence for a solution data set is used. When `sol` is none then the corresponding probe solution data set does not refer to any solver sequence. This means, for example, that the current model is used for selections used for this probe. When a solver sequence is run, then solution to use for the probes is always reset to use the current solver.

Boundary Probes, Domain Probes, Edge Probes, Global Variable Probes, and Probe Point Expressions

Probes can be of the following types:

TABLE 2-64: PROBE TYPES

TYPE	DESCRIPTION
Boundary	Probe that defines a value as an integral, maximum, minimum, or average over boundaries.
Domain	Probe that defines a value as an integral, maximum, minimum, or average over domains.
Edge	Probe that defines a value as an integral, maximum, minimum, or average over edges (in 3D).
GlobalVariable	Probe that defines a value using a global variable.
PointExpr	Probe that defines a value by interpolation of an expression in a probe point. The probe point is defined by the parent, a <code>DomainPoint</code> or a <code>BoundaryPoint</code> .

Boundary Point Probes and Domain Point Probes

Probe points can be of the following types:

TABLE 2-65: PROBE POINT TYPES

TYPE	DESCRIPTION
BoundaryPoint	Defines a probe coordinate on a boundary in 3D.
DomainPoint	Defines a probe coordinate in a domain.

Probes take the following properties:

TABLE 2-66: PROBE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
descr	string		Description of the probe. Used for model.result().
descriptive	on off	off	Manual control of description.
expr	string		The expression defining the probe.
frame	string	spatial frame	Frame used for defining the probe.
intorder	Integer	4	Integration order (DomainProbe and BoundaryProbe).
intsurface	Boolean	false	Compute surface integral for 1D axisymmetric average and integral probe types (DomainProbe and BoundaryProbe).
intvolume	Boolean	false	Compute volume integral for 2D axisymmetric average and integral probe types (DomainProbe and BoundaryProbe).
method	integration summation	integration	Method used (DomainProbe and BoundaryProbe).
probename	string	probe tag	Probe variable name.
table	string	default	Table to use for probe evaluation.
type	average maximum minimum integral	average	Type of probe (DomainProbe and BoundaryProbe).
unit	string	unit of expr	Unit for the probe. Used for model.result().
window	string	default	The plot window to use for the probe.

A probe point of the type DomainPoint takes the following properties:

TABLE 2-67: PROBE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
bndsnap1	on off	off	Snap to nearest point (1D).
bndsnap2	on off	off	Snap to nearest boundary point (2D).
bndsnap3	on off	off	Snap to nearest boundary point (3D).
coords	Matrix of doubles		Probe coordinates.
depthpointnormal	double	0	Depth along line defined by the pointnormal method.
depthpointdirection	double	0	Depth along line defined by the pointdirection method.
depthtwopoints	double	0	Depth along line defined by the twopoints method.
dimension	1 2 3	3	The spatial dimension in which the point resides.
first	Double array		The coordinates of the first point on the probe line.
method	pointnormal pointdirection twopoints none	pointnormal	Line entry method.
second	Double array		The coordinates of the second point (for method=twopoints)
twopointscurrent	first second	first	Point selector (for method=twopoints)

A probe point of BoundaryPoint types take the following properties:

TABLE 2-68: PROBE PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
coords	Double array	0,0,0	Probe coordinates.
rawcoords	Double array	0,0,0	Full precision probe coordinates.
snapcoords	Double array	0,0,0	The boundary-snapped coordinates.

model.result()

Postprocessing and results interface.

SYNTAX

```

model.result().create(<pgtag>,dim);
model.result().create(<pgtag>,ftype);
model.result().autoplot();
model.result().autoplot(<value>);
model.result(<pgtag>).set(property,<value>);
model.result(<pgtag>).run();

model.result(<pgtag>).create(<ftag>,ftype);
model.result(<pgtag>).feature(<ftag>).getPlotGroup();
model.result(<pgtag>).feature(<ftag>).getType();
model.result(<pgtag>).feature(<ftag>).getSDim();
model.result(<pgtag>).feature(<ftag>).isPlotGroup();
model.result(<pgtag>).feature(<ftag>).selection(...);
model.result(<pgtag>).feature(<ftag>).set(property,<value>);
model.result(<pgtag>).feature(<ftag>).run();
model.result(<tag>).feature(<ftag>).getAllowedPropertyValues(property);

model.result(<pgtag>).feature(<ftag>).create(<attrtag>, attrtype);
model.result(<pgtag>).feature(<ftag>).feature(<attrtag>).set(property,<value>);

model.result().dataset().create(<dtag>,dtype);
model.result().dataset(<dtag>).create(<dtag2>,dtype);
model.result().dataset(<dtag>).selection(...);
model.result().dataset(<dtag>).set(property,<value>);
model.result().dataset(<dtag>).getAllowedPropertyValues(property);

model.result().export().create(<etag>,<pgtag>,ctype);
model.result().export(<etag>).create(<e2tag>,ctype);
model.result().export(<etag>).set(property,<value>);
model.result().export(<etag>).run();
model.result().export(<etag>).getAllowedPropertyValues(property);

model.result().numerical().create(<ntag>,ntype);
model.result().numerical(<ntag>).selection(...);
model.result().numerical(<ntag>).set(property,<value>);
model.result().numerical(<ntag>).run();

model.result().table().create(<ftag>,ntype);
model.result().table(<ftag>).setColumnHeaders(<headers>);
model.result().table(<ftag>).setTableData(<realData>,<imagData>);
model.result().table(<ftag>).getColumnHeaders();
model.result().table(<ftag>).getReal();
model.result().table(<ftag>).getImag();
model.result().table(<ftag>).isComplex();
model.result().table(<ftag>).clearTableData();
model.result().table(<ftag>).save(<filename>);

```

```

model.result().report().create(<rtag>);
model.result().report(<rtag>).getAllowedPropertyValues(property);
model.result().report(<rtag>).set(rprop,<value>);
model.result().report(<rtag>).create(<r2tag>,frtype);
model.result().report(<rtag>).feature(<r2tag>).set(rprop,<value>);
model.result().report(<rtag>).feature(<r2tag>).create(<r3tag>,frtype);
model.result().report(<rtag>).feature(<r2tag>).feature(<r3tag>).set(rprop,<value>);
model.result().report(<rtag>).run();

```

DESCRIPTION

`model.result(<pgtag>)` returns a plot group with tag `<pgtag>`.

`model.result().create(<pgtag>,dim)` creates a plot group with the tag `<pgtag>`, of dimension `dim`, where `dim` can be 1, 2, or 3.

`model.result().create(<pgtag>,frtype)` creates a plot group of type `frtype`.

`model.result().autoplot()` returns true if plot features are plotted automatically when selected.

`model.result().autoplot(true)` and `model.result().autoplot(false)` enable and disable automatic plotting, respectively.

`model.result(<pgtag>).create(<ftag>,frtype)` creates a plot feature of type `frtype` tagged `<ftag>` belonging to the plot group `<pgtag>`.

`model.result(<pgtag>).feature(<ftag>).getType()` returns the type of the feature `<ftag>`. This is the same string `frtype` that was used to create the feature.

`model.result(<pgtag>).feature(<ftag>).getSDim()` returns the spatial dimension of the plot group.

`model.result(<pgtag>).feature(<ftag>).isPlotGroup()` return true if the feature is a plot group. This method is also available on the child features.

`model.result(<pgtag>).feature(<ftag>).getPlotGroup()` returns the plot group the feature belongs to. This method is also available on the child features.

`model.result(<pgtag>).feature(<ftag>).create(<attrtag>, attrtype)` creates an attribute feature with the tag `<attrtag>` of type `attrtype`, belonging to the feature `<ftag>`.

`model.result(<pgtag>).run()` plots the plot group.

`model.result().dataset().create(<dtag>,dtype)` creates a data set feature with the tag `<dtag>` and the type `dtype`.

`model.result().export().create(<etag>,<pgtag>,etype)` creates an export feature with the tag `<etag>`, belonging to plot group `<pgtag>` and of export type `etype`.

`model.result().numerical().create(<ntag>, ntype)` creates a numerical results feature with the tag `<ntag>` of the numerical feature type `ntype`.

`model.result().numerical(<ntag>).run()` evaluates the numerical results feature.

`model.result(<tag>).feature(<ftag>).getAllowedPropertyValues(property)`,

`model.result().dataset(<dtag>).getAllowedPropertyValues(property)`,

`model.result().export(<etag>).getAllowedPropertyValues(property)`, and

`model.result().report(<rtag>).getAllowedPropertyValues(property)` return the set of allowed values for a property if the set is a finite set of strings; otherwise, they return null.

`model.result().table().create(<ftag>,ntype)` creates a table feature with the tag `<ftag>`. The set and get methods used to manipulate tables are described in [Table](#).

The data extraction methods used to retrieve plot or numerical data are described in [Results](#).

`model.result().report().create(<rtag>, "Report")` creates a report with tag <rtag>.

`report.create(<tptag>, "TitlePage")` adds a title page to the report `report`. Only one title-page feature can be added.

`report.feature("<tptag>").set(prop, value)` sets the title-page property `prop` to the value `value`.

`report().create(<toctag>, "TableOfContents")` adds a table of contents to the report `report`. Only one table-of-contents feature can be added.

`report.create(<stag>, "Section")` adds an additional section level to a report.

`report.feature(<stag>).set(prop, value)`

`report.feature(<stag>).create(<ftag>, feature)`

`report.feature(<stag>).feature(<ftag>).set(prop, value)`

To add a report contents feature—that is, a feature corresponding to content in the report—to a report section feature `section`, type, `section.create(<frtag>, frtype, ...)`. Depending on the report feature type `frtype`, the create operation includes zero, one or two tags that refer to the model feature to report about. The tags must refer to an existing feature of the correct type. The following report feature types are available:

REPORT FEATURES	DESCRIPTION
Model	Prints information about the model root, such as model file.
ModelNode	Prints information about a model, its identifier.
Parameter	Reports on the global model parameters.
Variables	Reports on a variables feature.
Functions	Reports on a function feature.
Probe	Reports on a probe feature.
ModelCoupling	Reports on a model-coupling feature.
Selection	Reports on a selection feature.
CoordinateSystem	Reports on a coordinate system feature.
PML	Reports on a PML feature.
InfiniteElements	Reports on an infinite elements feature.
Pair	Reports on a pair feature.
Geometry	Reports on a geometry.
Material	Reports on a material feature.
Physics	Reports on a physics interface and its features.
Mesh	Reports on a mesh.
Study	Reports on a study.
Solver	Reports on a solver.
DataSet	Reports on a data-set feature.
DerivedValues	Reports on a derived-values feature.
Table	Includes a results table in the report.
PlotGroup	Includes a plot group in the report.
Export	Includes an export feature in the report.

To point a report feature `rFeature` to another feature with tag `<ftag>` in the tree, use the method `rFeature.set("noderef", <ftag>)` method. A report contents feature must point to a feature of the type it is designed to report on; see the table above. Instead of a feature tag, set "noderef" to "none" to clear a reference.

```
model.result().report(<rtag>).feature(<stag>).feature(<frtag>).set(frprop, <value>)
```

to set a property in a report feature.

EXAMPLES

Create a data set and set it to point to the tagged solution `sol1` from a solver sequence:

Code for use with Java

```
model.result().dataset().create("dset", "Solution");
model.result().dataset("dset").set("solution", "Sol1");
```

Code for use with MATLAB

```
model.result.dataset.create('dset', 'Solution');
model.result.dataset('dset').set('solution', 'Sol1');
```

Create a 3D plot group containing a streamline plot and a plane with a contour plot on:

Code for use with Java

```
result().create("pg1", 3);
result("pg1").set("data", "dset");
result("pg1").create("stream", "Streamline");
model.result("pg1").feature("stream").set("expr", new String[]{"2-x", "0", "z"});
model.result("pg1").feature("stream").selection().set(new int[]{2});
result().dataset().create("cutp1", "CutPlane");

result("pg1").create("cont1", "Contour");
result("pg1").feature("cont1").set("data", "cutp1");
result("pg1").run();
```

Code for use with MATLAB

```
result.create('pg1', 3);
result('pg1').set('data', 'dset');
result('pg1').create('stream', 'Streamline');
model.result('pg1').feature('stream').set('expr', {'2-x', '0', 'z'});
model.result('pg1').feature('stream').selection.set(2);
result.dataset.create('cutp1', 'CutPlane');

result('pg1').create('cont1', 'Contour');
result('pg1').feature('cont1').set('data', 'cutp1');
result('pg1').run;
```

model.savePoint()

Manage selections and hide features used by result features.

SYNTAX

```
model.savePoint(<tag>).geom(<gtag>)
model.savePoint(<tag>).geom(<gtag>).selection(<stag>)
model.savePoint(<tag>).geom(<gtag>).view(<vtag>)
```

DESCRIPTION

`model.savePoint(<tag>)` is a container of selections and hide features used by result features. When solving, a copy of the model is made—a save point model—which is used in results and analysis. The selections and hide features contained in `model.savePoint(<tag>)` refer to the geometry in this copy.

Editing the data in `model.savePoint(<tag>)` can only be done in the following circumstances.

- The geometry on which the analysis is done has been modified after solving. In this case the selections and hide feature can be edited but not created or removed.
- The geometry on which the analysis is done has been removed. In this case the selections and hide features can be both edited, created, and deleted.

In all other circumstances, edit the selections in `model.selection()`, and the hide features in `model.view()`. Changes there are synchronized with the data in `model.savePoint()`.

`model.savePoint(<tag>).geom(<gtag>)` returns a container with selections and views with hide features for a geometry in the save point model.

`model.savePoint(<tag>).geom(<gtag>).selection(<stag>)` returns a selection.

`model.savePoint(<tag>).geom(<gtag>).view(<vtag>)` returns a view. Contrary to the views in `model.view()`, only the hide features in `view.hideEntities()` can be edited.

SEE ALSO

[model.selection\(\)](#), [model.weak\(\)](#)

model.selection()

Named selections.

SYNTAX

```
model.selection().create(<tag>);
model.selection().create(<tag>,<type>);
model.selection(<tag>).model(<mtag>);
model.selection(<tag>).set(property,<value>);
model.selection(<tag>).geom(<gtag>,dim);
model.selection(<tag>).geom(<gtag>,highdim,lowdim,typelist);
model.selection(<tag>).geom(dim);
model.selection(<tag>).all();
model.selection(<tag>).set(<entlist>);
model.selection(<tag>).add(<entlist>);
model.selection(<tag>).remove(<entlist>);
model.selection(<tag>).inherit(bool);

model.selection(<tag>).model();
model.selection(<tag>).isGeom();
model.selection(<tag>).geom();
model.selection(<tag>).dimension();
model.selection(<tag>).entities(dim);
model.selection(<tag>).interiorEntities(dim);
model.selection(<tag>).isInheriting();
model.selection(<tag>).inputDimension();
model.selection(<tag>).inputEntities();
```

DESCRIPTION

`model.selection(<tag>)` returns a named selection. Anywhere where you define a selection, you can point to a named selection by using its tag, for example, `selection.named(<tag>)`.

`model.selection().create(<tag>)` creates a named selection of type Explicit.

`model.selection().create(<tag>,<type>)` creates a named selection of type `<type>`.

`model.selection(<tag>).model(<mtag>)` sets the model node of the selection. Only geometries in the model can be used in selections.

`model.selection(<tag>).model()` returns the model node of the selection.

`model.selection(<tag>).set(property,<value>)` sets a property value for the selection. Which properties are available for the different selection types are listed on the following pages. All other assignment methods are only supported by the Explicit selection type.

All other methods are explained in the section [Selections](#).

Other entities can use any of the selections in `model.selection()` when defining its selection. For example, create a selection `sel1`:

```
model.selection().create("sel1");
model.selection("sel1").model("mod1");
```

Then, for example, a variable entities can use this selection:

```
model.variable().create("var1");
model.variable("var1").model("mod1");
model.variable("var1").selection().named("sel1");
```

What properties are available depends on the type of selection. The following selection types are available:

Explicit

Selection defined by an explicit set of geometric entities such as domains or boundaries.

TABLE 2-69: EXPLICIT SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
angletol	double	5	Angle tolerance for continuity evaluation
groupcontang	on off	off	Continuous tangent mode

When `groupcontang` is set to on, the `set`, `add`, and `remove` methods operate on groups of adjacent entities that have continuous tangents at their junctions.

The `angletol` property defines the tolerance for the continuity evaluation.

Ball

Selection of entities that are inside or intersect a ball.

TABLE 2-70: BALL SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
angletol	double	5	Angle tolerance for continuity evaluation
groupcontang	on off	off	Continuous tangent mode
inputent	all selections	all	Use all entities or entities defined by input property
input	string[]	{}	Input selections
condition	intersects inside somevertex allvertices	intersects	Condition for inclusion of an entity
posx	double	0	Center of ball, first coordinate
posy	double	0	Center of ball, second coordinate
posz	double	0	Center of ball, third coordinate
r	double	0	Radius

The `posx`, `posy`, and `posz` properties define the center of the ball, and `r` defines the radius. These properties take their units from the corresponding geometry sequence.

When condition is `intersects`, all entities that intersect the ball are included in the selection. The rendering mesh is used for the calculation. You can set the accuracy of the rendering mesh using

```
ModelUtil.setPreference("graphics.rendering.detail",<detail>);
```

where `<detail>` is `coarse`, `normal`, `fine`, or `wireframe`.

When condition is `inside`, all entities that are completely inside the ball are included in the selection. The rendering mesh is used for the calculation.

When condition is `somevertex`, all entities that have at least one adjacent vertex inside the ball are included in the selection.

When condition is `allvertices`, all entities that have all adjacent vertices inside the ball are included in the selection.

When inputent is `selections`, the selection is restricted to the entities in the selections defined by the input property. When inputent is `all`, all entities in the geometry are considered.

When groupcontang is set to `on`, the selection operates on groups of entities that have continuous tangents at their junctions.

The `angletol` property defines the tolerance for the continuity evaluation.

Box

Selection of entities that are inside or intersect a box.

TABLE 2-71: BOX SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
angletol	double	5	Angle tolerance for continuity evaluation
groupcontang	on off	off	Continuous tangent mode
inputent	all selections	all	Use all entities or entities defined by input property
input	string[]	{}	Input selections
condition	intersects inside somevertex allvertices	intersects	Condition for inclusion of an entity
xmax	double	inf	Maximum x-coordinate of box
xmin	double	-inf	Minimum x-coordinate of box
ymax	double	inf	Maximum y-coordinate of box
ymin	double	-inf	Minimum y-coordinate of box
zmax	double	inf	Maximum z-coordinate of box
zmin	double	-inf	Minimum z-coordinate of box

The `xmax`, `xmin`, `ymax`, `ymin`, `zmax`, and `zmin` properties define the box. These properties take their units from the corresponding geometry sequence.

When condition is `intersects`, all entities that intersect the box are included in the selection. The rendering mesh is used for the calculation. You can set the accuracy of the rendering mesh using

```
ModelUtil.setPreference("graphics.rendering.detail",<detail>);
```

where `<detail>` is `coarse`, `normal`, `fine`, or `wireframe`.

When condition is `inside`, all entities that are completely inside the box are included in the selection. The rendering mesh is used for the calculation.

When condition is `somevertex`, all entities that have at least one adjacent vertex inside the box are included in the selection.

When condition is `allvertices`, all entities that have all adjacent vertices inside the box are included in the selection.

When inputent is `selections`, the selection is restricted to the entities in the selections defined by the input property. When inputent is `all`, all entities in the geometry are considered.

When groupcontang is set to `on`, the selection operates on groups of entities that have continuous tangents at their junctions.

The `angletol` property defines the tolerance for the continuity evaluation.

Cylinder

Selection of entities that are inside or intersect a cylinder in 3D.

TABLE 2-72: CYLINDER SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	3	Dimension of entities to select
angletol	double	5	Angle tolerance for continuity evaluation
groupcontang	on off	off	Continuous tangent mode
inputent	all selections	all	Use all entities or entities defined by input property
input	string[]	{}	Input selections
condition	intersects inside somevertex allvertices	intersects	Condition for inclusion of an entity
pos	double[]	{0,0,0}	Cylinder base point
axis	double[]	{0,0,1}	Direction of the cylinder axis. Vector has length 3 if axistype is cartesian and length 2 if axistype is spherical. Not used if axistype is x, y, or z.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
top	double	inf	Coordinate of upper face in local coordinate system
bottom	double	-inf	Coordinate of lower face in local coordinate system
r	double	0	Radius

The `pos` property defines the center of the cylinder and the `axis` property defines the cylinder axis. The `top`, `bottom`, and `r` properties define the size of the cylinder. These properties take their units from the corresponding geometry sequence.

When condition is `intersects`, all entities that intersect the cylinder are included in the selection. The rendering mesh is used for the calculation. You can set the accuracy of the rendering mesh using

```
ModelUtil.setPreference("graphics.rendering.detail",<detail>);
```

where `<detail>` is `coarse`, `normal`, `fine`, or `wireframe`.

When condition is `inside`, all entities that are completely inside the cylinder are included in the selection. The rendering mesh is used for the calculation.

When condition is `somevertex`, all entities that have at least one adjacent vertex inside the cylinder are included in the selection.

When condition is `allvertices`, all entities that have all adjacent vertices inside the cylinder are included in the selection.

When `inputent` is `selections`, the selection is restricted to the entities in the selections defined by the input property. When `inputent` is `all`, all entities in the geometry are considered.

When `groupcontang` is set to `on`, the selection operates on groups of entities that have continuous tangents at their junctions.

The `angleto1` property defines the tolerance for the continuity evaluation.

Union

Selection defined by the union of a set of selections.

TABLE 2-73: UNION SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
input	string[]	{}	Selections to add

Intersection

Selection defined by the intersection of a set of selections.

TABLE 2-74: INTERSECTION SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
input	string[]	{}	Selections to intersect

Difference

Selection defined by the difference between two sets of selections.

TABLE 2-75: DIFFERENCE SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
add	string[]	{}	Selections to add
subtract	string[]	{}	Selections to subtract

Complement

Selection defined by the complement of a set of selections.

TABLE 2-76: COMPLEMENT SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
input	string[]	{}	Selections to invert

Adjacent

Selection of entities that are adjacent to entities in another selection.

TABLE 2-77: ADJACENT SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
entitydim	0 1 2 3	sdim	Dimension of entities to select
input	String[]	{}	Input selections
outputdim	0 1 2 3	sdim-1	Dimension of output entities

TABLE 2-77: ADJACENT SELECTION PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
exterior	on off	on	Include exterior boundaries/edges
interior	on off	off	Include interior boundaries/edges.

EXAMPLES

Define the selection `equ1` as the domain of a rectangle and the selection `bnd1` as the boundary of the rectangle.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom().create("geom1",2);
model.geom("geom1").create("f1", "Rectangle");
model.geom("geom1").run("f1");
model.selection().create("equ1").geom(2);
model.selection("equ1").all();
model.selection().create("bnd1").geom(1);
model.selection("bnd1").all();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('geom1',2);
model.geom('geom1').create('f1', 'Rectangle');
model.geom('geom1').run('f1');
model.selection.create('equ1').geom(2);
model.selection('equ1').all;
model.selection.create('bnd1').geom(1);
model.selection('bnd1').all;
```

The (outer) boundaries for the model can be set with the following selection:

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom().create("geom1",2);
model.geom("geom1").create("r1", "Rectangle");
model.selection().create("outer").geom("geom1",2,1,new String[]{"exterior"});
model.selection("outer").all();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('geom1',2);
model.geom('geom1').create('r1', 'Rectangle');
model.selection.create('outer').geom('geom1',2,1,{ 'exterior' });
model.selection('outer').all;
```

Create a selection for all boundaries of a block intersecting a ball with radius 0.5 and center (1,1,1):

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom().create("g1",3).create("blk1", "Block");
model.geom("g1").run();
model.selection().create("ball1", "Ball");
model.selection("ball1").set("entitydim", "2");
model.selection("ball1").set("posx", "1");
model.selection("ball1").set("posy", "1");
model.selection("ball1").set("posz", "1");
model.selection("ball1").set("r", "0.5");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('g1',3).create('blk1', 'Block');
model.geom('g1').run;
model.selection.create('ball1', 'Ball');
model.selection('ball1').set('entitydim', '2');
```

```

model.selection('ball1').set('posx', '1');
model.selection('ball1').set('posy', '1');
model.selection('ball1').set('posz', '1');
model.selection('ball1').set('r', '0.5');

```

Create a selection of all edges adjacent to the boundaries in the ball selection:

Code for use with Java

```

model.selection().create("adj1", "Adjacent");
model.selection("adj1").set("entitydim", "2");
model.selection("adj1").set("outputdim", "1");
model.selection("adj1").set("input", new String[]{"ball1"});

```

Code for use with MATLAB

```

model.selection.create('adj1', 'Adjacent');
model.selection('adj1').set('entitydim', '2');
model.selection('adj1').set('outputdim', '1');
model.selection('adj1').set('input', {'ball1'});

```

SEE ALSO

[Selections](#)

model.shape()

Create and define shape functions for the field variables' elements.

SYNTAX

```

model.shape().create(<tag>, <frame>);
model.shape(<tag>).elementSet(<condition>);
model.shape(<tag>).frame(<ftag>);
model.shape(<tag>).create(<ftag>, <func>);
model.shape(<tag>).feature(<ftag>).set(property, <value>);
model.shape(<tag>).slits().named(<seltag>);
model.shape(<tag>).slits().set(...);
model.shape(<tag>).upFlux(<varName>);
model.shape(<tag>).downFlux(<varName>);
model.shape(<tag>).domainFlux(<expressions>, <frame>);

model.shape(<tag>).elementSet();
model.shape(<tag>).frame();
model.shape(<tag>).feature(<ftag>).shape();
model.shape(<tag>).feature(<ftag>).getType(<property>);
model.shape(<tag>).feature(<ftag>).properties;
model.shape(<tag>).fieldVariable();
model.shape(<tag>).slits().named();
model.shape(<tag>).slits().getType();
model.shape(<tag>).upFlux();
model.shape(<tag>).downFlux();

```

DESCRIPTION

`model.shape(<tag>)` returns a shape function.

`model.shape().create(<tag>, <frame>)` creates a shape function with tag `<tag>` and assigns the frame `<frame>` to it.

`model.shape(<tag>).frame(<ftag>)` assigns frame `<ftag>` to the shape function. See [model.frame\(\)](#) for a discussion on the default frame.

`model.shape(<tag>).create(<ftag>, <func>)` creates a shape feature with the shape function expression `func`. `func` can be a shape function name (`shlag`, for example) or a shape function with arguments (`shlag(2,u)`, for example). The latter is interpreted as an assignment of some property values.

`model.shape(<tag>).feature(<ftag>).set(property,<value>)` sets a property for the shape function. Of the data types listed in [Table 2-2](#), the ones supported are those for integers, strings, and string arrays. Which ones are applicable differs for each property.

`model.shape(<tag>).selection().named(<seltag>)` assigns the shape function to the named selection `<seltag>`.

`model.shape(<tag>).selection().set(...)` defines a local selection that assigns the shape function to geometric entities. For a complete list of methods available under `selection()`, see [Selections](#).

`model.shape(<tag>).upFlux(<varName>)` and `model.shape(<tag>).downFlux(<varName>)` set the names of the up and down boundary flux variables. `model.shape(<tag>).domainFlux(<expressions>,<frame>)` sets expressions for the domain flux in a given frame. This is required to make the boundary flux variables produce accurate results. Only Lagrange shape functions support boundary flux variables.

`model.shape(<tag>).frame()` returns the frame tag as a string.

`model.shape(<tag>).feature(<ftag>).shape()` returns the shape function expression as a string.

`model.shape(<tag>).feature(<ftag>).getType(property)` returns a property value. For available data types, see [getType\(\)](#).

`model.shape(<tag>).feature(<ftag>).properties()` returns the names of the properties as a string array.

`model.shape(<tag>).fieldVariable()` returns the field variables which the shape functions define.

`model.shape(<tag>).selection().named()` returns the selection tag as a string.

`model.shape(<tag>).elementSet(<condition>)` sets the element set condition to the given string. The condition should be an expression containing *element set variables* (defined in `model.elementSet()`) and the logical operators `&&`, `||`, and `!`. The shape function is defined only on the mesh elements for which the condition is true (nonzero).

`model.shape(<tag>).elementSet()` returns the element set condition. An empty string means no condition.

`model.shape(<tag>).selection().getType()` returns domain information. For available methods, see [model.selection\(\)](#).

`model.shape(<tag>).slits()` returns a selection used to generate a slit on the shape. Works exactly as selections.

`model.shape(<tag>).upFlux()` and `model.shape(<tag>).downFlux()` return the names of the up and down flux variables (an empty string if the variable names have not been set.)

EXAMPLE

Define the shape function `shlag(2,"u")`:

Code for use with Java

```
model.shape().create("shu","f");
model.shape("shu").create("f1","shlag");
model.shape("shu").feature("f1").set("order",2);
model.shape("shu").feature("f1").set("basename","u");
model.shape("shu").selection().named("equ1");
```

Code for use with MATLAB

```
model.shape.create('shu','f');
model.shape('shu').create('f1','shlag');
model.shape('shu').feature('f1').set('order',2);
model.shape('shu').feature('f1').set('basename','u');
model.shape('shu').selection.named('equ1');
```

See also [Shape Functions and Element Types](#) for information and syntax examples for all shape functions (element types).

SEE ALSO

[model.coeff\(\)](#), [model.intRule\(\)](#), [model.weak\(\)](#) and the [Elements and Shape Functions](#) chapter.

model.sol()

Solver sequences.

SYNTAX

```
model.sol().create(<tag>)
model.sol().create(<tag>, <studytag>)
model.sol().create(<tag>, <studytag>, <varstag>)
model.sol().remove(<tag>)

model.sol(<tag>).create(<ftag>, <oper>)
model.sol(<tag>).feature().remove(<ftag>)
model.sol(<tag>).feature(<ftag>).create(<f2tag>, <oper>)
model.sol(<tag>).feature(<ftag>).set(property, <value>)
model.sol(<tag>).feature(<ftag>).getAllowedPropertyValues(property);
model.sol(<tag>).clearSolution()
model.sol(<tag>).copySolution(<ctag>)
model.sol(<tag>).createAutoSequence(<stag>)
model.sol(<tag>).createSolution()
model.sol(<tag>).updateSolution()
model.sol(<tag>).run(<ftag>)
model.sol(<tag>).runFrom(<ftag>)
model.sol(<tag>).runFromTo(<ftagstart>, <ftagstop>)
model.sol(<tag>).runAll()
model.sol(<tag>).run()
model.sol(<tag>).continueCun()
```

DESCRIPTION

`model.sol().create(<tag>)` adds a solver sequence to the model.

`model.sol().create(<tag>, <studytag>)` adds a solver sequence to the model. The constructor adds one feature of the type `StudyStep` to the solver sequence with the tag `<studytag>`. This `StudyStep` feature is connected to a study step (see [model.study\(\)](#)).

`model.sol().create(<tag>, <studytag>, <varstag>)` adds a solver sequence to the model. The constructor adds one feature of the type `StudyStep` with the tag `<studytag>` and one feature of the type `Variables` with the tag `<varstag>` to the solver sequence.

`model.sol().remove(<tag>)` removes a solver sequence from the model.

`model.sol(<tag>).create(<ftag>, <oper>)` creates a solver feature. Each solver feature is a solver operation.

`model.sol(<tag>).feature().remove(<ftag>)` removes the solver feature `<ftag>`.

`model.sol(<tag>).feature(<ftag>).set(property, <value>)` sets the property `property` for the feature `<ftag>`.

`model.sol(<tag>).feature(<ftag>).getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

`model.sol(<tag>).clearSolution()` clears the solution data associated with the solution `<tag>`. The features are not changed.

`model.sol(<tag>).copySolution(<ctag>)` copies the solution data associated with the solver sequence `<tag>` to a new solver sequence `<ctag>`. The features are not copied.

`model.sol(<tag>).createAutoSequence(<stag>)` creates a solver sequence of features automatically from the study `<stag>`. The sequence of study steps are used as input to the sequence generation algorithm but also the physics used in the study steps are used to automatically adopt the solver settings.

`model.sol(<tag>).createSolution()` creates a solution object from one or more set operations (`setU(..)`, ...), see [Solution Creation](#) for details.

`model.sol(<tag>).updateSolution()` updates a solution data associated with the solver sequence to make it consistent with the current model.

`model.sol(<tag>).run(<ftag>)` runs the features for a solver sequence up to and including the feature `<ftag>`.

`model.sol(<tag>).runFrom(<ftag>)` runs the features for a solver sequence from and including the feature `<ftag>`.

`model.sol(<tag>).runFromTo(<ftagstart>,<ftagstop>)` runs the features for a solver sequence from and including the feature `<ftagstart>` to and including the feature `<ftagstop>`.

`model.sol(<tag>).runAll()` and `model.sol(<tag>).run()` run all the features for a solver sequence.

`model.sol(<tag>).continueRun()` continues to run a solver sequence.

EXAMPLES

Assume that a study `st1` represents one stationary study step with the tag `stat1` for some equations.

Code for use with Java

```
model.sol().create("s","step1","vars1");
model.sol("s").feature("step1").set("study","st1");
model.sol("s").feature("step1").set("studystep","stat1");
model.sol("s").create("solver1","Stationary");
```

Code for use with MATLAB

```
model.sol.create('s','step1','vars1');
model.sol('s').feature('step1').set('study','st1');
model.sol('s').feature('step1').set('studystep','stat1');
model.sol('s').create('solver1','Stationary');
```

Assume that a second study step with frequency response is added to the study with tag `freq1` and that you want to make a frequency sweep from 10 to 1000 using the parametric solver and the solution above as the linearization point (bias solution).

Code for use with Java

```
model.sol("s").create("step2","StudyStep");
model.sol("s").feature("step2").set("study","st1");
model.sol("s").feature("step2").set("studystep","freq1");
model.sol("s").create("vars2","Variables");
SolverFeature s2 = (SolverFeature) model.sol("s").create("solver2","Stationary");
s2.set("nonlin","linper"); // (*)
s2.set("linpmethod","sol");
s2.set("linpsol","s");
s2.set("storelinpoint","on");
s2.create("par","Parametric");
s2.feature("par").set("pname","freq");
s2.feature("par").set("plist",new double[]{10,1000});
s2.runAll();
```

Code for use with MATLAB

```
model.sol('s').create('step2','StudyStep');
model.sol('s').feature('step2').set('study','st1');
```

```

model.sol('s').feature('step2').set('studystep','freq1');
model.sol('s').create('vars2','Variables');
s2 = model.sol('s').create('solver2','Stationary');
s2.set('nonlin','linper'); // (*)
s2.set('linpmethod','sol');
s2.set('linpsol','s');
s2.set('storelinpoint','on');
s2.create('par','Parametric');
s2.feature('par').set('pname','freq');
s2.feature('par').set('plist',[10,1000]);
s2.runAll;

```

At this point the solution `s` is associated to the study step `freq1` (but it depends indirectly on the bias study step `stat1` as well).

(*) Uses the small-signal study functionality, which makes it possible to access also the linearization point for postprocessing together with the small-signal solution. Here it is assumed that the bias problem and the small-signal problem can be set up independently for the two study steps.

SEE ALSO

[model.study\(\)](#)

model.solverEvent()

Create and define events for the solver.

SYNTAX

```

model.solverEvent().create(<tag>,evtype);
model.solverEvent(<tag>).start(expr);
model.solverEvent(<tag>).start();
model.solverEvent(<tag>).period(expr);
model.solverEvent(<tag>).period();
model.solverEvent(<tag>).condition(expr);
model.solverEvent(<tag>).condition();
model.solverEvent(<tag>).reinit();
model.solverEvent(<tag>).reinit().create(<tag>);
model.solverEvent(<tag>).reinit(<tag>).set(<var>,expr);

```

DESCRIPTION

Create events and control event settings. There are two types of events; Explicit and Implicit.

`model.solverEvent().create(<tag>,evtype)` creates a new event of type `evtype`, either Explicit and Implicit.

Explicit Events

Explicit events triggers on a predefined timing.

`model.solverEvent(<tag>).start(expr)` sets the start time for an explicit event.

`model.solverEvent(<tag>).period(expr)` sets the period for an explicit event. After the start time, the event then triggers after each period.

Implicit Events

Implicit events trigger when a condition goes from false to true.

`model.solverEvent(<tag>).condition(expr)` sets the condition for an implicit event.

Re-Initialization

When an event is triggered, any degree of freedom can be re-initialized. This typically means that they get a new value. You specify these values with a re-initialization method, `reinit(...)`, which has the same syntax as `model.init(...)`.

`model.solverEvent(<tag>).reinit().create(<tag>)` adds a new reinit feature to the event. In most cases you only need one, but you need more when you have reinitialization conditions on several geometric entity levels, for example on a global selection and a domain selection.

Event State Variables

An event needs state variables in most cases. There are discrete states and indicator states. Discrete states are just ODE states that only change during re-initialization, and can only have a zero-valued equation (or no equation). The indicator states are needed for implicit events and are ODE states with nonzero equations.

`model.ode().create(<tag>).type(<ode type>)` creates a new global equation that contains event state variables if you set the ode type to `discrete` for discrete states and `quadrature` for indicator states.

`model.ode(<tag>).state(<states>)` adds a new discrete states to the global equation.

`model.ode(<tag>).ode(<state>, "sin(2*pi*t)")` adds a new indicator state and its right-hand side to the global equation. The left-hand side of the equation is the state variable, so the full equation for the indicator state becomes `nojac(sin(2*pi*t)) - <state>`.

EXAMPLE

Example of an idealized bouncing ball using implicit events.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.study().create("std1");
model.study("std1").create("time1", "Transient");
model.study("std1").feature("time1").set("tlist", "0 10");
model.study("std1").feature("time1").set("rtol", 1e-6); // Nondiscrete states
model.ode().create("ode1");
model.ode("ode1").ode("y", "-2*y-ytt");
model.init().create("ode1");
model.init("ode1").selection().global();
model.init("ode1").set("y", "1");
// Discrete states
model.ode().create("ode2").type("quadrature");
model.ode("ode2").ode("z1", "y");
// Implicit event
model.solverEvent().create("impl1", "Implicit");
model.solverEvent("impl1").condition("! (z1>=0)");
model.solverEvent("impl1").reinit().create("reinit");
model.solverEvent("impl1").reinit("reinit").selection().global();
model.solverEvent("impl1").reinit("reinit").set("y", "y");
// Bounce reverts velocity
model.solverEvent("impl1").reinit("reinit").set("yt", "-yt");
model.sol().create("sol1");
model.sol("sol1").createAutoSequence("std1");
// Special solver settings for events
model.sol("sol1").feature("t1").set("tout", "tsteps");
model.sol("sol1").feature("t1").set("atolglobal", "1e-6");
model.sol("sol1").feature("t1").set("initialstepbdfactive", "on");
model.sol("sol1").feature("t1").set("initialstepbdf", "1e-6");
model.sol("sol1").feature("t1").set("eventtol", "2e-6");
model.sol("sol1").feature("t1").set("ewtrescale", "off");
model.sol("sol1").runAll();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.study.create('std1');
model.study('std1').create('time1', 'Transient');
model.study('std1').feature('time1').set('tlist', '0 10');
model.study('std1').feature('time1').set('rtol', 1e-6);
% Nondiscrete states
model.ode.create('ode1');
model.ode('ode1').ode('y', '-2*y-ytt');
model.init.create('ode1');
model.init('ode1').selection().global();
model.init('ode1').set('y', '1');
% Discrete states
model.ode.create('ode2').type('quadrature');
model.ode('ode2').ode('z1', 'y');
% Implicit event
model.solverEvent.create('impl1', 'Implicit');
model.solverEvent('impl1').condition('!(z1>=0)');
model.solverEvent('impl1').reinit().create('reinit');
model.solverEvent('impl1').reinit('reinit').selection().global();
model.solverEvent('impl1').reinit('reinit').set('y', 'y');
% Bounce reverts velocity
model.solverEvent('impl1').reinit('reinit').set('yt', '-yt');
model.sol().create('sol1');
model.sol('sol1').createAutoSequence('std1');
% Special solver settings for events
model.sol('sol1').feature('t1').set('tout', 'tsteps');
model.sol('sol1').feature('t1').set('atolglobal', '1e-6');
model.sol('sol1').feature('t1').set('initialstepbdfactive', 'on');
model.sol('sol1').feature('t1').set('initialstepbdf', '1e-6');
model.sol('sol1').feature('t1').set('eventtol', '2e-6');
model.sol('sol1').feature('t1').set('ewtrescale', 'off');
model.sol('sol1').runAll;
```

SEE ALSO

[model.ode\(\)](#), [model.init\(\)](#)

model.study()

Create and define studies.

SYNTAX

```
model.study().create(<tag>);
model.study(<tag>).create(<ftag>,type);
model.study(<tag>).feature().move(<ftag>,position);

model.study(<tag>).run()
model.study(<tag>).runNoGen()
model.study(<tag>).createAutoSequences(type)
model.study(<tag>).showAutoSequences(type)
model.study(<tag>).getSolverSequences(type)
model.study(<tag>).setStoreSolution(boolean)
model.study(<tag>).isStoreSolution()
model.study(<tag>).feature(<ftag>).getAllowedPropertyValues(property);

step = model.study(<tag>).feature(<ftag>);

step.mesh(<geom>,<mesh>);
step.activate(<physpath>,<bool>);
step.discretization(<physpath>,<discr>);
step.mglevel.create(<mglevel>);
step.mglevel(<mglevel>).mesh(<geom>,<mesh>);
step.mglevel(<mglevel>).discretization(<physpath>,<discr>);

step.type();
step.activate(<physpath>);
step.discretization(<physpath>);
step.mesh(<geom>);
step.mglevel(<mglevel>).mesh(<geom>);
step.mglevel(<mglevel>).discretization(<physpath>);
```

DESCRIPTION

`model.study` stores a list of studies, each of which consists of a number of study steps. Each study step, in turn, defines a solver-ready problem. This means that a study step can be turned into an extended mesh, and a basic solver (Stationary, Time, Eigenvalue, Modal, AWE, Optimization) can be applied, resulting in a solution object.

The central property of a study step is its *study type*, which on one hand controls the equations generated by physics interfaces, and on the other hand triggers automatic selection of a suitable solver. Another important property of a study step is which mesh to use (for each geometry in the model). Other fundamental simulation parameters can also be found among the study step settings, like the time span for a Time Dependent study type and frequency range for a Frequency Domain study type.

Under a study step, you can add *multigrid levels*. The parent node still defines the problem to be solved (for example, the study type and the mesh). Therefore, the added multigrid levels must necessarily be coarser than the parent study step.

Most physics features and also some other parts of the model object (for example expression features) must support a `step` member, which (in analogy to the spatial `selection`) controls for which study steps the feature is active. In many ways, the study selection can be seen as a fourth, discrete, dimension.

`model.study().create(<tag>)` creates a new study sequence.

`model.study(<tag>).run()` computes the study.

`model.study(<tag>).runNoGen()` runs the attached solver sequence without regenerating it.

`model.study(<tag>).createAutoSequences(type)` creates an attached solver sequence or job using default solver settings. The argument `type` is one of `all`, `jobs`, or `sol`, corresponding to creating both jobs and solver sequences or one of them.

`model.study(<tag>).showAutoSequences(type)` generates a new attached solver sequence or job using default solver settings. The previous item describes the *type* argument.

`model.study(<tag>).getSolverSequences(type)` returns a list of tags for solver sequences (see `model.sol()`) connected to this study. The *type* argument is one of `SolverSequence`, `CopySolution`, `ParametricStore`, `Stored`, `Parametric`, `None`, or `All`.

`model.study(<tag>).setStoreSolution(boolean)` inserts a Solution Store node between each study step in a multistep study if set to `true`. If set to `false`, Solution Store nodes are only inserted in certain cases. Use `model.study(<tag>).isStoreSolution()` to check if a Solution Store node is inserted between each study step (it then returns `true`).

`model.study(<tag>).create(<ftag>, type)` creates a new study step of the given type within the specified sequence. The set of allowed values should be limited to study types supported by at least one physics interface present in the model (`Stationary`, `Time`, `Frequency`, and `Eigenvalue` should always be allowed).

`model.study(<tag>).feature().move(<ftag>, position)` moves the feature *<ftag>* to the zero indexed position *<position>* in the list.

`step = model.study(<tag>).feature(<ftag>)` obtains a reference to a specified study step.

`model.study(<tag>).feature(<ftag>)`.

`getAllowedPropertyValues(property)` returns the set of allowed values for a property if the set is a finite set of strings; otherwise, it returns null.

`step.mesh(<geom>, <mesh>)` specifies which mesh to use for geometry *<geom>* in the model.

`step.activate(<physpath>, <bool>)` activates or deactivates a physics interface or a physics feature. The string *<physpath>* is a path to a node in a physics interface. Currently only the tag of the physics interface tag itself is supported.

`step.discretization(<physpath>, <discr>)` assigns discretization for a physics interface. The string *<physpath>* is the path of a physics interface. The string *<discr>* is a tag of a discretization feature under a physics mode. The default *<discr>* the physics interface tag. It can be changed to the tag of a discretization node under a physics interface.

`step.mglevel.create(<mglevel>)` adds a (coarser) multigrid level to a study.

`step.mglevel(<mglevel>).mesh(<geom>, <mesh>)` specifies a mesh for the multigrid level. The set of allowed values must, in addition to the actual meshes, include an option “from parent”. This should be the default choice and indicates that the multigrid level uses the same mesh as the parent study.

`step.mglevel(<mglevel>).discretization(<physpath>, <discr>)` assigns discretization for a multigrid level. The string *<physpath>* is the path of a physics interface. The string *<discr>* is a tag of a discretization feature under a physics mode. The default *<discr>* the physics interface tag. It can be changed to the tag of a discretization node under a physics interface.

`step.type()` returns the study type.

`step.mesh(<geom>)` returns the mesh selected for the given geometry.

`step.activate(<physpath>)` returns activation status of a physics interface or a physics interface feature. Currently only the tag of the physics interface tag itself is supported.

`step.mglevel(<mglevel>).mesh(<geom>)` returns the mesh for the selected multigrid level and geometry.

`step.mglevel(<mglevel>).discretization(<discpath>)` returns activation status of a discretization feature.

EXAMPLE

The following code sets up a study sequence to analyze the influence of structural deformation on a waveguide with a numerical port boundary condition. It consists of three steps: stationary structural mechanics followed by an eigenvalue study for the port and finally a wave propagation problem solved with manual multigrid levels (to get nested meshes).

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.geom().create("geom1", 3);
model.geom("geom1").create("blk1", "Block");
model.geom().run();
model.mesh().create("mesh1", "geom1");
model.mesh().create("mesh2", "geom1");
model.mesh().create("mesh3", "geom1");

model.physics().create("rfw1", "ElectromagneticWaves", "geom1");
model.study().create("seq1");
Study s1 = model.study("seq1");
s1.create("struct", "Stationary");
s1.feature("struct").mesh("geom1", "mesh1");
s1.create("port", "BoundaryModeAnalysis");
s1.feature("port").set("PortName", "port1");
s1.feature("port").mesh("geom1", "mesh2");
s1.create("wave", "Frequency");
s1.feature("wave").mesh("geom1", "mesh2");
s1.feature("wave").mglevel().create("mgl1");
s1.feature("wave").mglevel().create("mgl2");
s1.feature("wave").mglevel("mgl2").mesh("geom1", "mesh3");
model.physics("rfw1").create("mgl1", "Discretization");
model.physics("rfw1").feature("mgl1").set("order", "1");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.geom.create('geom1', 3);
model.geom('geom1').create('blk1', 'Block');
model.geom.run;
model.mesh.create('mesh1', 'geom1');
model.mesh.create('mesh2', 'geom1');
model.mesh.create('mesh3', 'geom1');

model.physics.create('rfw1', 'ElectromagneticWaves', 'geom1');
model.study.create('seq1');
s1 = model.study('seq1');
s1.create('struct', 'Stationary');
s1.feature('struct').mesh('geom1', 'mesh1');
s1.create('port', 'BoundaryModeAnalysis');
s1.feature('port').set('PortName', 'port1');
s1.feature('port').mesh('geom1', 'mesh2');
s1.create('wave', 'Frequency');
s1.feature('wave').mesh('geom1', 'mesh2');
s1.feature('wave').mglevel.create('mgl1');
s1.feature('wave').mglevel.create('mgl2');
s1.feature('wave').mglevel('mgl2').mesh('geom1', 'mesh3');
model.physics('rfw1').create('mgl1', 'Discretization');
model.physics('rfw1').feature('mgl1').set('order', '1');
```

In this case, the only settings that must be applied in the study members of other features are the ones relating to multigrid levels. The physics interfaces' equation form is by default set to `automatic`, which means that they respond suitably to the study type each time an extended mesh (xmesh) is created.

SEE ALSO

[model.batch\(\)](#), [model.physics\(\)](#), [model.sol\(\)](#)

Unit systems.

SYNTAX

```
UnitSystem us = model.unitSystem().create(<tag>);
us.baseUnit().create(<tag>,<symbol>,<quantity>)
us.derivedUnit().create(<tag>,<units>,<powers>);
us.additionalUnit().create(<tag>,<dim>);
model.unitSystem().builtInTags();

us.baseUnit(<tag>);
us.derivedUnit(<tag>);
us.additionalUnit(<tag>)
us.derivedUnit(<tag>).aliases();
us.baseUnit(<tag>).dimension();
us.derivedUnit(<tag>).quantity();
us.derivedUnit(<tag>).offset();
us.derivedUnit(<tag>).scale();
us.derivedUnit(<tag>).symbol();
us.derivedUnit(<tag>).definition(<units>,<powers>);
us.additionalUnit(<tag>).aliases(<aliases>);
us.additionalUnit(<tag>).quantity(<quantity>);
us.additionalUnit(<tag>).offset(<offset>);
us.additionalUnit(<tag>).scale(<scale>);
us.additionalUnit(<tag>).offset(<offset>);
us.additionalUnit(<tag>).symbol(<symbol>);
```

DESCRIPTION

`model.unitSystem().create(<uname>)` creates a unit system *<uname>*.

`us.baseUnit().create(<tag>,<symbol>,<quantity>)` creates a base unit for the quantity *<quantity>*, tagged *<tag>* with the symbol *<symbol>*. The quantity is any of the seven base dimensions (length, mass, time, current, temperature, substance, and intensity).

`us.derivedUnit().create(<tag>,<units>,<powers>)` creates a new derived unit tagged *<tag>* and derived from the units in *<units>* each to the power of the powers in *<powers>*.

`us.derivedUnit(<tag>).definition(<units>,<powers>)` sets the definition of a derived unit in powers of other units. The resulting dimension must agree with any previously specified dimension for this unit. Use the `create` method to define a dimension from the derived units.

`us.additionalUnit().create(<tag>,<dim>)` creates a new additional unit.

All methods below are valid for all units, no matter what unit list they belong to. Furthermore, only the set methods are described here, but there is also a corresponding get method.

`model.unitSystem().builtInTags()` returns the tags of the built-in unit systems. The method `model.unitSystem().tags()` returns the tags of the user-defined unit systems. Both sets of tags can be used to retrieve the unit system using `model.unitSystem(<tag>)`.

`us.additionalUnit(<tag>).aliases(<aliases>)` sets alternate names for the unit that can be used in unit expressions.

`us.additionalUnit(<tag>).quantity(<quantity>)` assigns a physical quantity to the given unit.

`us.additionalUnit(<tag>).scale(<scale>)` sets the scale of the additional unit.

`us.derivedUnit(<tag>).symbol(<symbol>)` sets the symbol of the derived unit.

`us.derivedUnit(<tag>).offset(<offset>)` sets the offset of the derived unit.

NOTES

You can set the base unit system for the entire model using `model.baseSystem(<utag>)` or separately for each model node using `model.modelNode(<tag>).baseSystem(<utag>)`.

The SI system is read only and always created by default.

EXAMPLE

Create a cgs2 unit system with the base unit for length set to centimeter (cm). Also add meter/second (m/s) as a derived unit for speed and degrees Celsius as an additional unit for temperature:

Code for use with Java

```
Model model = ModelUtil.create("Model");
UnitSystem us = model.unitSystem().create("cgs2");
model.baseSystem("cgs2");
us.baseUnit().create("centimeter", "cm", "length");
us.derivedUnit().create("meter_per_second", new int[]{1,0,-1,0,0,0,0,0});
Unit du = us.derivedUnit("meter_per_second");
du.definition(new String[]{"meter", "second"}, new int[]{1,-1,0,0,0,0,0,0});
Unit au = us.additionalUnit().create("celsius", new int[]{0,0,0,0,1,0,0,0});
au.offset(273.15);
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
us = model.unitSystem.create('cgs2');
model.baseSystem('cgs2');
us.baseUnit.create('centimeter', 'cm', 'length');
us.derivedUnit.create('meter_per_second', [1,0,-1,0,0,0,0,0]);
du = us.derivedUnit('meter_per_second');
du.definition({'meter', 'second'}, [1,-1,0,0,0,0,0,0]);
au = us.additionalUnit.create('celsius', [0,0,0,0,1,0,0,0]);
au.offset(273.15);
```

SEE ALSO

[model.physics\(\)](#)

model.variable()

Create, define, and remove variables.

SYNTAX

```
model.variable().create(<tag>);
model.variable(<tag>).set(<var>, <expr>);
model.variable(<tag>).set(<var>, <expr>, <descr>);
model.variable(<tag>).descr(<var>, <descr>);
model.variable(<tag>).remove(<var>);
model.variable(<tag>).model(<mtag>);
```

```
model.variable(<tag>).varnames();
model.variable(<tag>).get(<var>);
model.variable(<tag>).descr(<var>);
model.variable(<tag>).model();
model.variable(<tag>).scope();
model.variable(<tag>).loadFile(tempFile, ...);
model.variable(<tag>).saveFile(tempFile, ...);
```

DESCRIPTION

`model.variable(<tag>)` returns a variable collection. Each variable collection can contain several variables, but only one selection.

`model.variable().create(<tag>)` creates a variables node with tag `<tag>`.

`model.variable(<tag>).set(<var>,<expr>)` defines the variable `<var>` by the expression `<expr>`.

`model.variable(<tag>).set(<var>,<expr>,<descr>)` defines a variable and gives it a description.

`model.variable(<tag>).descr(<var>,<descr>)` defines a description for the variable `<var>`.

`model.variable(<tag>).model(<mtag>)` sets the model node.

`model.variable(<tag>).selection().named(<seltag>)` assigns the variable node to the named selection `<seltag>`.

`model.shape(<tag>).selection().set(...)` defines a local selection that assigns the variable collection to geometric entities. Before assigning a selection, the variable's model must be set using `model.variable(<tag>).model(<mtag>)`. Only the global selection and selections on a geometry in the model can be used. For a complete list of methods available under `selection()`, see [Selections](#).

`model.variable(<tag>).remove(<var>)` removes a variable from the variable collection.

`model.variable(<tag>).varnames()` returns the names of all expressions as a string array.

`model.variable(<tag>).get(<var>)` returns the variable value as a string.

`model.variable(<tag>).descr(<var>)` returns the variable description as a string.

`model.variable(<tag>).model()` returns the model node.

`model.variable(<tag>).scope()` returns the fully qualified scope name.

`model.variable(<tag>).selection().named()` returns the selection tag as a string.

`model.variable(<tag>).selection().getType()` returns domain information. For available methods, see [model.selection\(\)](#).

For `model.param().loadFile()` and `model.param().saveFile()`, see [The loadFile and saveFile Methods](#).

EXAMPLES

Define the expression `e` as `x+1` in Domains 1 and 2 and as `x-1` in Domain 3.

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.modelNode().create("mod1");
model.geom().create("geom1",3);
model.geom("geom1").create("blk1", "Block");
model.geom("geom1").run();
model.variable().create("e1").set("e","x+1");
model.variable("e1").model("mod1");
model.variable("e1").selection().geom("geom1",2);
model.variable("e1").selection().set(new int[]{1,2});
model.variable().create("e2").set("e","x-1");
model.variable("e2").model("mod1");
model.variable("e2").selection().geom("geom1",2);
model.variable("e2").selection().set(3);
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.modelNode.create('mod1');
model.geom.create('geom1',3);
model.geom('geom1').create('blk1', 'Block');
model.geom('geom1').run;
model.variable.create('e1').set('e','x+1');
model.variable('e1').model('mod1');
model.variable('e1').selection.geom('geom1',2);
model.variable('e1').selection.set([1,2]);
model.variable.create('e2').set('e','x-1');
```

```

model.variable('e2').model('mod1');
model.variable('e2').selection().geom('geom1',2);
model.variable('e2').selection().set(3);

```

SEE ALSO

[model.selection\(\)](#)

model.view()

Create and manage views.

SYNTAX

```

model.view().create(<vtag>,<gtag>)
model.view().create(<vtag>,<viewdim>)
model.view().create(<vtag>,<gtag>,<workplane>)
model.view(<vtag>).set(<pname>,<pvalue>)
model.view(<vtag>).getType(<pname>)

model.view(<vtag>).axis().set(<pname>,<pvalue>)
model.view(<vtag>).axis().getType(<pname>)
model.view(<vtag>).camera().set(<pname>,<pvalue>)
model.view(<vtag>).camera().getType(<pname>)
model.view(<vtag>).light().create(<ltag>,<ltype>)
model.view(<vtag>).light(<ltag>).set(<pname>,<pvalue>)
model.view(<vtag>).light(<ltag>).getType(<pname>)
model.view(<vtag>).hideObjects().create(<htag>)
model.view(<vtag>).hideObjects(<htag>).set(<pname>,<pvalue>)
model.view(<vtag>).hideObjects(<htag>).getType(<pname>)
model.view(<vtag>).hideEntities().create(<htag>)
model.view(<vtag>).hideEntities(<htag>).set(<pname>,<pvalue>)
model.view(<vtag>).hideEntities(<htag>).getType(<pname>)

```

DESCRIPTION

View Settings

`model.view()` returns a list of view that can be used when viewing geometry/mesh and plot groups. Each view has an axis and some properties on the top level. In 3D a view also has a camera and a list of lights. All views also have a list of hide features.

`model.view().create(<vtag>,<gtag>)` creates a view tied to the geometry with the given tag. The dimension of the view is the same as the dimension for the geometry.

`model.view().create(<vtag>,<viewdim>)` creates a view with the given tag for the given dimension (1, 2, or 3). These views are not tied to any geometry and show up under the Views node under Results in the COMSOL Desktop.

`model.view().create(<vtag>,<gtag>,<workplane>)` creates a view tied to the work plane with the given tag in the geometry sequence with the given tag. The dimension of the view is 2.

`model.view(<vtag>).set(<pname>,<pvalue>)` sets the given property to the given value.

`model.view(<vtag>).getType(<pname>)` returns the property with the given name of type *Type*.

TABLE 2-78: VIEW PROPERTIES

NAME	VALUE	DIMENSIONS	DESCRIPTION
default	on off	1, 2, 3	If this is the default view to use when viewing the geometry and mesh.
headlight	on off	3	If the light in the camera should be turned on.

TABLE 2-78: VIEW PROPERTIES

NAME	VALUE	DIMENSIONS	DESCRIPTION
locked	on off	1, 2, 3	If the settings should be updated from interactive changes or not.
scenelight	on off	3	If the background lights as specified by the added lights should be turned on.
viewscaletype	none automatic manual	2, 3	Control the view scale to achieve a suitable axis scaling (default: None).
wireframe	on off	3	If wireframe rendering should be used.
xscale	double	2, 3	Scaling factor for x-axis when using a manual view scale.
yscale	double	2, 3	Scaling factor for y-axis when using a manual view scale.
zscale	double	3	Scaling factor for z-axis when using a manual view scale.

Axis Settings

The axis settings apply to 2D views with an *x*- and a *y*-axis.

`model.view(<vtag>).axis().set(<pname>, <pvalue>)` sets the given axis property to the given value. Which axis properties that are available in the different dimensions are given in the table below.

`model.view(<vtag>).axis().getType(<pname>)` returns the axis property with the given name.

TABLE 2-79: AXIS PROPERTIES

NAME	VALUE	DIMENSIONS	DESCRIPTION
auto	on off	1, 2, 3	Set if axis settings should be automatically stored and updated from interactive changes using mouse and toolbar buttons.
equal	on off	2, 3	Should the same scaling be used for all directions.
logx	on off	1	Should log scale be used for the x-axis.
logy	on off	1	Should log scale be used for the y-axis.
manualspacing	on off	2	Should manual spacing be used for x and y grid lines.
manuallimits	on off	1, 2, 3	Should manual axis limits be used. If not a zoom extents is performed each time something is plotted into the axis.
xextra	double array	2	An array with extra x grid lines.
xmax	double	1, 2, 3	The maximum <i>x</i> -coordinate.
xspacing	double	2	Manual spacing for x grid lines.
yextra	double array	2	An array with extra y grid lines.
ymin	double	2, 3	The minimum <i>y</i> -coordinate.
ymax	double	2, 3	The maximum <i>y</i> -coordinate.
yspacing	double	2	Manual spacing for y grid lines.
zmin	double	3	The minimum <i>z</i> -coordinate.
zmax	double	3	The maximum <i>z</i> -coordinate.

Camera Settings

These settings apply to the camera for 3D views.

`model.view(<vtag>).camera().set(<pname>, <pvalue>)` sets the given camera property to the given value.

`model.view(<vtag>).camera().getType(<pname>)` returns the camera property with the given name.

TABLE 2-80: CAMERA PROPERTIES

NAME	VALUE	DESCRIPTION
manualspacing	on off	Should manual spacing be used for grid lines.
position	double array	The position of the camera.
target	double array	The point the camera looks at.
up	double array	The up direction.
xextra	double array	An array with extra x grid lines.
xspacing	double	Manual spacing for x grid lines.
yextra	double array	An array with extra y grid lines.
yspacing	double	Manual spacing for y grid lines.
zextra	double array	An array with extra z grid lines.
zspacing	double	Manual spacing for z grid lines.
zoomanglefull	double	The full field of view angle in degrees.

Light Settings

These settings control the different types of lighting—direction light, spotlight, head light, and point light—that you can add to a 3D view.

`model.view(<vtag>).light().create(<ltag>, <ltype>)` creates a light with the given tag and type. `<ltype>` can be any of 'DirectionalLight', 'PointLight', 'SpotLight', and 'HeadLight'.

`model.view(<vtag>).light(<ltag>).set(<pname>, <pvalue>)` sets the given light property to the given value. Different properties are available for the different types of lights according to the table below.

`model.view(<vtag>).light(<ltag>).getType(<pname>)` returns the light property with the given name.

TABLE 2-81: LIGHT PROPERTIES

NAME	VALUE	LIGHT TYPES	DESCRIPTION
color	string or RGB triplet	all	The color of the light.
cameracoord	on off	all	If the light should be defined in the camera coordinate system or the world coordinate system.
direction	double array	DirectionalLight, SpotLight	The direction from which the light shines or is directed at.
position	double array	PointLight, SpotLight	The position from which the light shines.
spreadangle	double	SpotLight	The spread angle for the light.

Hiding Objects and Entities

`model.view(<vtag>).hideObjects().create(<htag>)` creates a hide feature of geometric objects. The API for controlling it is similar to the API for selection in the geometry sequence; see [Geometry Object Selection Methods](#) under `model.geom()`.

`model.view(<vtag>).hideEntities().create(<htag>)` creates a hide feature of geometric entities. The API for controlling it is similar to the API for selections on the finalized geometry; see `model.selection()`.

SEE ALSO

`model.result()`

Weak form equations.

SYNTAX

```
model.weak().create(<tag>);
model.weak(<tag>).weak(<wlist>);
model.weak(<tag>).weak(<pos>,<wexpr>);
model.weak(<tag>).intRule(<irlist>);
model.weak(<tag>).intRule(<pos>,<irule>);
model.weak(<tag>).condition(<condition>);
```

```
model.weak(<tag>).weak();
model.weak(<tag>).intRule();
model.weak(<tag>).condition();
```

DESCRIPTION

`model.weak(<tag>)` returns the weak form equations with tag `<tag>`.

`model.weak().create(<tag>)` creates weak form equations with tag `<tag>`.

`model.weak(<tag>).weak(<wlist>)` sets the equations. You can supply a single weak expression or a list of weak expressions. `<wlist>` is a string or a string array.

`model.weak(<tag>).weak(<pos>,<wexpr>)` sets the equations at position `<pos>` in the list.

`model.weak(<tag>).intRule(<irlist>)` assigns the integration rules to the weak form equations. The list of integration rules must have the same length as the list of equations, or be of length 1. In the latter case all weak expressions use the same integration rule.

`model.weak(<tag>).intRule(<pos>,<irule>)` sets the integration rule at position `<pos>` in the integration rule list.

`model.weak(<tag>).condition(<condition>)` introduces conditional assembly. The feature is assembled if `<condition>` is true.

`model.weak(<tag>).selection().named(<seltag>)` assigns the weak equations to the named selection `<seltag>`.

`model.weak(<tag>).selection().named(<seltag>)` defines a local selection that assigns the weak equations to geometric entities. Before assigning a selection, the variable's model must be set using `model.variable(<tag>).model(<mtag>)`. Only the global selection and selections on a geometry in the model can be used. For a complete list of methods available under `selection()`, see [Selections](#).

EXAMPLE

Define the weak expressions `u*test(u)` and `v*test(v)` on the selection `dom1`, using the integration rule `gp1` and the frame `ref`.

Code for use with Java

```
model.weak().create("w1").selection().named("dom1");
model.weak("w1").intRule("gp1");
model.weak("w1").weak(new String[]{"u*test(u)", "v*test(v)"});
```

Code for use with MATLAB

```
model.weak.create('w1').selection.named('dom1');
model.weak('w1').intRule('gp1');
model.weak('w1').weak({'u*test(u)', 'v*test(v)'});
```

SEE ALSO

[model.coeff\(\)](#), [model.shape\(\)](#)

Geometry

This chapter includes reference information about the geometry commands and how to work with a geometry sequence and the geometry objects to create the model geometry.

About Geometry Commands



- [Features for Creating Geometric Primitives](#)
- [Features for Geometric Operations](#)
- [Features for Virtual Operations](#)
- [Features for Mesh Control](#)
- [Geometry Object Information Methods](#)



For documentation of additional geometry commands available in the Design Module, CAD Import Module, ECAD Import Module, and LiveLink™ for CAD products, see the documentation for each of those products.

Features for Creating Geometric Primitives

Table 3-1 is an overview of the features for creating 3D geometric primitives:

TABLE 3-1: 3D GEOMETRIC PRIMITIVES

NAME	DESCRIPTION
BezierPolygon	Chain of connected line segments, quadratic or cubic curves
Block	Right-angled parallelepiped (box)
Cone	Right circular cone or cone frustum
Cylinder	Right circular cylinder
ECone	Oblique cone or cone frustum with elliptic base
Ellipsoid	Ellipsoid
Helix	Helix solid, surface, or curve
Hexahedron	Hexahedron bounded by bilinear faces
ParametricCurve	Curve defined by coordinate expressions
ParametricSurface	Surface defined by coordinate expressions
Point	One or several points
Polygon	Chain of connected line segments
Pyramid	Rectangular pyramid
Sphere	Sphere or ball
Tetrahedron	Tetrahedron
Torus	Torus

Table 3-2 is an overview of the features for creating 2D geometric primitives:

TABLE 3-2: 2D GEOMETRIC PRIMITIVES

NAME	DESCRIPTION
BezierPolygon	Chain of connected line segments, quadratic or cubic curves
Circle	Circle or disc
Ellipse	Ellipse
ParametricCurve	Curve defined by coordinate expressions
Point	One or several points
Polygon	Chain of connected line segments

TABLE 3-2: 2D GEOMETRIC PRIMITIVES

NAME	DESCRIPTION
Rectangle	Rectangle
Square	Square

[Table 3-3](#) is an overview of the features for creating 1D geometric primitives:

TABLE 3-3: 1D GEOMETRIC PRIMITIVES

NAME	DESCRIPTION
Interval	One interval, or a chain of connected intervals
Point	One or several points

Features for Geometric Operations

The [Import](#) feature imports geometry objects from a file or from another geometry. The [FromMesh](#) feature constructs a geometry object from a (deformed) mesh.

[Table 3-4](#) through [Table 3-8](#) list the features that create new geometric objects from existing ones, [Table 3-9](#) lists programming features for construction of geometry parts with conditionally active geometry features, for example.

TABLE 3-4: WORK-PLANE RELATED FEATURES (ONLY 3D, EXCEPT FOR CROSSSECTION)

NAME	DESCRIPTION
WorkPlane	Create a work plane for drawing 2D objects that are embedded into 3D
Extrude	Extrude planar faces in 3D
Revolve	Revolve planar faces in 3D
Sweep	Sweep one or several faces along a spine curve to create a solid in 3D
CrossSection	Create 2D geometry from intersection of 3D geometry with work plane

TABLE 3-5: BOOLEAN AND PARTITIONING OPERATIONS

NAME	DESCRIPTION
Compose	Compose geometry objects using a set formula
Difference	Subtract geometry objects from geometry objects
Intersection	Intersect geometry objects
Union	Unite geometry objects
Partition	Partition a 3D geometry using tool objects or a work plane
PartitionDomains	Partition domains in 2D or 3D geometries along some partitioning lines, edges, or faces
PartitionEdges	Partition edges in 2D or 3D geometries at some partitioning vertices along the edges

See [Compose](#), [Union](#), [Intersection](#), [Difference](#) for information about those Boolean operations.

TABLE 3-6: LINEAR TRANSFORMATIONS

NAME	DESCRIPTION
Array	Rectangular or linear array of geometry objects
Mirror	Reflect objects in a plane (3D), a line (2D), or a point (1D)
Rotate	Rotate geometry objects about a center point
Scale	Scale geometric objects about a center point
Move	Translate geometry objects
Copy	Make a displaced copy of geometry objects

See [Move](#), [Copy](#) for details about those linear transformations.

TABLE 3-7: OBJECT TYPE CONVERSIONS

NAME	DESCRIPTION
ConvertToSolid	Unite and convert objects to a single solid object
ConvertToSurface	Unite and convert 3D objects to a single surface object
ConvertToCurve	Unite and convert 2D or 3D objects to a single curve object
ConvertToPoint	Unite and convert objects to a single point object

See [ConvertToSolid](#), [ConvertToSurface](#), [ConvertToCurve](#), [ConvertToPoint](#) for information about those conversion operations.

TABLE 3-8: OTHER OPERATIONS

NAME	DESCRIPTION
Chamfer	Chamfer corners in 2D geometry objects
Fillet	Fillet corners in 2D geometry objects
Tangent	Line segment tangent to an edge in 2D
Delete	Delete entities (domains, boundaries, edges, or points) from objects, or delete entire geometry objects
Split	Split geometry objects into their constituent entities
Finalize	Form union or assembly by combining all geometry objects

TABLE 3-9: PROGRAMMING AND PARTS FEATURES

NAME	DESCRIPTION
If , ElseIf , Else , EndIf	Construct an If statement, enabling or disabling features depending on conditions in terms of parameters
ParameterCheck	Check the value of parameters.
PartInstance	Create an instance of a geometry part.

Selection Features

[Table 3-10](#) lists the features that correspond to selections:

TABLE 3-10: SELECTIONS

NAME	DESCRIPTION
AdjacentSelection	Selection of entities or objects that are adjacent to given selections
ExplicitSelection	Explicit selection of entities or objects
BallSelection	Selection of entities or objects that (partly) lie inside a ball
BoxSelection	Selection of entities or objects that (partly) lie inside a box
CylinderSelection	Selection of entities or objects that (partly) lie inside a cylinder
ComplementSelection	Selection of entities or objects that is the complement of the input selections
DifferenceSelection	Selection of entities or objects that is the difference of the input selections
IntersectionSelection	Selection of entities or objects that is the intersection of the input selections
UnionSelection	Selection of entities or objects that is the union of the input selections

See [BallSelection](#), [BoxSelection](#), [CylinderSelection](#) and [UnionSelection](#), [IntersectionSelection](#), [DifferenceSelection](#), [ComplementSelection](#) for information about those selections.

Features for Virtual Operations

Table 3-11 lists the features that correspond to virtual operations:

TABLE 3-11: VIRTUAL GEOMETRY RELATED FEATURES (ONLY 2D AND 3D)

NAME	DESCRIPTION
IgnoreVertices	Virtually remove isolated vertices or vertices adjacent to two edges only
IgnoreEdges	Virtually remove isolated edges or edges adjacent to precisely two faces or between two domains
IgnoreFaces	Virtually remove isolated faces or faces between two domains
CompositeEdges	Form virtual composite edges from sets of connected edges by ignoring the vertices between the edges in each set
CompositeFaces	Form virtual composite faces from sets of connected faces by ignoring the edges between the faces in each set
CompositeDomains	Form virtual composite domains from sets of connected domains by ignoring the boundaries between the domains in each set
CollapseEdges	Virtually collapse each edge into a vertex by merging its adjacent vertices
CollapseFaces	Virtually collapse faces
MergeEdges	Virtually merge edges adjacent to face
MergeVertices	Virtually merge one adjacent vertex of an edge with the other adjacent vertex

Features for Mesh Control

Table 3-12 lists the features that correspond to mesh control operations:

TABLE 3-12: MESH CONTROL RELATED FEATURES (ONLY 2D AND 3D)

NAME	DESCRIPTION
MeshControlVertices	Use vertices for mesh control only
MeshControlEdges	Use edges for mesh control only
MeshControlFaces	Use faces for mesh control only
MeshControlDomains	Use domains for mesh control only

GENERAL INFORMATION

TABLE 3-13: GENERAL GEOMETRY INFORMATION METHODS

METHOD	DESCRIPTION
check	Check object for errors
getBoundingBox	Get bounding box around object
getSDim	Get space dimension
getType	Get object type (solid, surface, curve, point, mixed, empty)
hasCadRep	Is represented using CAD kernel

GEOMETRIC ENTITY COUNTERS

TABLE 3-14: GEOMETRIC ENTITY COUNTERS

METHOD	DESCRIPTION
getNEntities	Get number of entities of different dimensions
getNVertices	Get number of vertices
getNEdges	Get number of edges
getNFaces	Get number of faces
getNBoundaries	Get number of boundaries
getNDomains	Get number of domains
getNEntitiesMesh	Get number of entities of different dimensions in the geometry used for meshing

ADJACENCY

TABLE 3-15: ADJACENCY BETWEEN GEOMETRIC ENTITIES

METHOD	DESCRIPTION
getStartEnd	Get start and end vertices of edges
getUpDown	Get up and down domain indices
getUpDownExt	Get up and down extended domain indices
getVertexDomain	Get domain index for isolated vertices
getSD	Get domain index for isolated vertices
getAdj	Get adjacency matrices
getAdjOrient	Get adjacency orientation
getAdjSparse	Get adjacency matrix in sparse format

EDGE EVALUATION

TABLE 3-16: EDGE EVALUATION METHODS

METHOD	DESCRIPTION
edgeParamRange	Get parameter range of edge
edgeX	Evaluate coordinates
edgeDX	Evaluate first derivative
edgeDDX	Evaluate second derivative
edgeNormal	Evaluate normal vector in 2D
edgeCurvature	Evaluate curvature
edgeTorsion	Evaluate torsion in 3D

FACE EVALUATION

TABLE 3-17: FACE EVALUATION METHODS

METHOD	DESCRIPTION
faceParamRange	Get parameter ranges of face
faceX	Evaluate coordinates
faceDX	Evaluate first derivatives
faceDDX	Evaluate second derivatives
faceNormal	Evaluate normal vector
faceFF1	Evaluate first fundamental form
faceFF2	Evaluate second fundamental form
faceGaussCurvature	Evaluate Gauss curvature
faceMeanCurvature	Evaluate mean curvature

GEOMETRY REPRESENTATION ARRAYS



TABLE 3-18: GET ARRAYS IN GEOMETRY REPRESENTATION

METHOD	DESCRIPTION
getVertex	Get vertex matrix
getEdges	Get edge matrix
getFaces	Get face matrix
getPVertex	Get parameter vertices (embeddings of vertices in faces)
getPEdge	Get parameter edges (embeddings of edges in faces)
getVertexCoord	Get vertex coordinates

Working with a Geometry Sequence

This section describes how to construct geometries using Java[®] methods. A *geometry* is defined by a *geometry sequence* consisting of *geometry features*. Each feature generates a set of *output geometry objects* when you *build* the feature. An *operation feature* takes previously generated geometry objects as input and usually deletes them. You can create named selections by adding *selection features*. Each geometry sequence in 1D ends with a **Finalize** feature that forms a single output object by uniting all existing geometry objects. A geometry sequence in 2D or 3D also contains a **Finalize** feature, but in 2D and 3D it is possible to add features corresponding to *virtual operations* after the **Finalize** feature (see [Virtual Operations](#) for more information). The output object of the last feature of a sequence is referred to as the *finalized geometry*. The finalized geometry is used for meshing and physics modeling.

In 3D, you can also add work planes (see [Work Planes](#) for more information), where you can add 2D geometry sequences that build 2D geometries that you can use to embed, extrude, and revolve in the 3D geometry.

	Methods on <code>model.geom(<tag>)</code> can also be used in work planes on <code>model.geom(<tag>).feature(<wptag>).geom()</code> .
	Geometry Modeling and CAD Tools in the <i>COMSOL Multiphysics Reference Manual</i>

Adding a Model Component (Geometry)

To add a new geometry to the model object `model`, enter

```
model.geom().create(<tag>,sDim);
```

where `<tag>` is the geometry's tag (an identifier of your choice), and `sDim` is its space dimension (1, 2, or 3).

The geometry is added to the last created model component (`modelNode`). If no model component exists in the model, a model component node tagged `mod1` is automatically created for you. A physics interface using the geometry must belong to the same model component as the geometry.

You can change the model component of a geometry by entering

```
model.geom(<tag>).model(<mtag>);
```

where `<mtag>` is the tag of a model node.

	<code>model.modelNode()</code>
-------------------------------------------------------------------------------------	--------------------------------

Adding a Geometry Feature

To add a feature to a geometry tagged `<tag>`, enter

```
model.geom(<tag>).create(<ftag>,ftype);
```

where `<ftag>` is the feature's tag (an identifier of your choice), and `ftype` is the feature's type. Feature types are capitalized and case-sensitive, for example `Rectangle`.

When you add a feature, it is inserted after the *current feature*. You can get the tag of the current feature type by entering

```
String ftag = model.geom(<tag>).current();
```

If `ftag` is the empty string, the current feature is the beginning of the geometry sequence, that is, the empty state before all features. When the feature has been added, it automatically becomes current, but it is not built automatically.

All properties in a new feature get a default value.

Editing a Geometry Feature

To change a property value in a feature, enter

```
model.geom(<tag>).feature(<ftag>).set(property,<value>);
```

where *property* is a property name and *<value>* is a property value.

All numeric properties can be given either as a numeric value or as a string expression that can contain parameters defined in `model.param()`. When building the feature, the string expressions are evaluated using the current values of the parameters.



Behavior has changed since version 3.5a. String expression arguments are retained and not converted to numeric values.

To get the value of a property, enter one of the following, depending on the type of the property:

```
double d = model.geom(<tag>).feature(<ftag>).getDouble(property);
String s = model.geom(<tag>).feature(<ftag>).getString(property);
double[] da = model.geom(<tag>).feature(<ftag>).getDoubleArray(property);
String[] sa = model.geom(<tag>).feature(<ftag>).getStringArray(property);
double[][] dm = model.geom(<tag>).feature(<ftag>).getDoubleMatrix(property);
String[][] sm = model.geom(<tag>).feature(<ftag>).getStringMatrix(property);
```

If you request a numerical value for a string property, it is evaluated using the current values of the parameters in `model.param()`.



- `getType()`
- `set()`

SELECTIONS

There are primitive features and operation features. Operations features take existing geometry objects as input and create new geometry objects from them. The input objects are usually specified in the `input` selection:

```
model.geom(<tag>).feature(<ftag>).selection("input").set(inputObjects);
```

where `inputObjects` is a string array with object or feature names. If `inputObjects` contains a feature name, it refers to all objects generated by this feature. If you have defined a named selection of objects, you can refer to it using the named method (see [Selection Features](#) below).



Geometry Object Selection Methods

Usually, the input objects of an operation feature is removed when building the feature. To change this behavior, a property `keep` is available for many operations features. If `keep` is set to `on`, the input objects are kept when building the feature.

Building Geometry Features

To generate the output objects of a feature, you must *build* the feature. Enter

```
model.geom(<tag>).run(<ftag>);
```

to build the feature `<ftag>` and all its preceding features (the features are built in the order from the first to the last). When the build has completed, the feature `<ftag>` becomes current.

To build all preceding features of the feature `<ftag>`, enter

```
model.geom(<tag>).runPre(<ftag>);
```

To build all features, including the Finalize feature and the virtual operations, enter

```
model.geom(<tag>).run();
```

ERRORS

If an error occurs when building a feature, the build stops, and the feature before the failing feature becomes current. The failing feature gets an *error feature* appended, which contains the error message. To access the error message, enter

```
String msg = model.geom(<tag>).feature(<ftag>).message();
```

To access the error feature, its message and detailed message, enter

```
String msg = model.geom(<tag>).feature(<ftag>).problem("error").getString("message");  
String det = model.geom(<tag>).feature(<ftag>).problem("error").getString("details");
```

WARNINGS

After a successful build, a feature can get warning features appended, which contain warning messages. To access the first warning message, enter

```
String msg = model.geom(<tag>).feature(<ftag>).message();
```

To access the warning features, their messages and detailed messages, enter

```
String msg = model.geom(<tag>).feature(<ftag>).problem(<wtag>).getString("message");  
String det = model.geom(<tag>).feature(<ftag>).problem(<wtag>).getString("details");
```

where `<wtag>` is `warning1`, `warning2`, and so on.

Feature Status

The *status* of a feature can be one of the following:

- *Built* or *warning*. This means that the none of the feature's properties have changed since the feature was last built, and the features of the input objects are all built. If the status is *warning*, the feature contains warning messages.
- *Edited*. This means that some of the feature's properties have changed since the feature was last built.
- *Needs rebuild*. This means that the feature generating some input object is not built.
- *Error*. This means that the feature contains an error message.

You can examine the status of a feature by entering

```
String status = model.geom(<tag>).feature(<ftag>).status();
```

Accessing Geometry Object Names

Each feature produces one or several output geometry objects. To get the names of these objects, enter

```
String[] oNames = model.geom(<tag>).feature(<ftag>).objectNames();
```

To get the names of all currently existing geometry objects (the geometry objects that were generated by the last build), enter

```
String[] oNames = model.geom(<tag>).objectNames();
```

To access one of these objects, you can enter

```
GeomObject go = model.geom(<tag>).obj(<objname>);
```

where the string *<objname>* is an object name. If *<objname>* does not exist in the current state, you get an error message. You can get information about the geometry object *go* by using the *geometry information methods*, for example,

```
int numberOfFaces = go.getNFaces();
```



Geometry Object Information

To access the finalized geometry (the output of the last feature), use

```
model.geom(<tag>)
```

NAMING OF GEOMETRY OBJECTS

The names of the output objects of a feature are formed by appending characters after the feature's tag, in one of the following ways:

- *ftag(index)*. For example, *split1(1)*, *split1(2)*, *split1(3)* if the feature tagged *split1* has three output objects. This method is used for most features.
- *ftag(i1,i2,...)*. For example, *arr1(1,1)*, *arr1(1,2)*, *arr(2,1)*, *arr1(2,2)* for a 2-by-2 array feature tagged *arr1*. This method is only used for the Array feature.
- *ftag.objectNameIn2D*. For example, *wp1.r1*, *wp1.pt1(1)*, *wp1.pt1(2)* if the work plane feature *wp1* contains the 2D objects *r1*, *pt1(1)*, and *pt1(2)*. This method is only used for the WorkPlane feature.
- *ftag.objectNameIn3D*. For example, *cro1.blk1* and *cro1.cyl1*. This method is used for the CrossSection feature.
- *ftag.objectName*. This method can be used for the Import feature, and then *objectName* is taken from the CAD file.

Deleting and Disabling Geometry Features

To delete a feature, enter

```
model.geom(<tag>).remove(<ftag>);
```

To disable a feature, enter

```
model.geom(<tag>).feature(<ftag>).active(false);
```

The disabled feature does not affect the finalized geometry—its output is empty. To enable a disabled feature, enter

```
model.geom(<tag>).feature(<ftag>).active(true);
```

You can get the enabled/disabled status of a feature by entering

```
boolean isEnabled = model.geom(<tag>).feature(<ftag>).active();
```

Deleting Geometry Objects

You can use the following operation to delete objects from the geometry sequence.

```
model.geom(<tag>).delete(String[]);
```

In the input string array you specify the names of the objects to delete. The operation deletes objects that correspond to primitive geometry features by removing these features from the sequence. The operation then deletes the remaining objects by adding and building a `Delete` feature with the objects in its selection.



When using the delete operation the status of the current feature and all its preceding features must be built.

Moving and Scaling Geometry Objects

You can use the following operations to move or scale objects from the geometry sequence.

```
model.geom(<tag>).move(String[] obj, double[] dist);  
model.geom(<tag>).scale(String[] obj, double[] factor, double[] center);  
model.geom(<tag>).scale(String[] obj, double factor, double[] center);
```

The input array `obj` specifies the objects to move or scale. The `dist` array specifies the move distance in each axis direction. The `factor` array specifies an anisotropic scaling and the `factor` scalar specifies an isotropic scaling. The `center` array specifies the scaling center point. When possible, the move and scale operations modify the corresponding geometry features in the sequence. Not all features can be moved or scaled by modifying their properties, in which case move or scale features are added to the geometry sequence instead.



When using the move or scale operations the status of the current feature and all its preceding features must be built.

Geometry Settings

You can control the following general settings for a geometry:

- [Length Unit](#)
- [Angular Unit](#)
- [Scale Values When Changing Unit](#)
- [Geometry Representation in 3D](#)
- [Default Relative Repair Tolerance](#)
- [Automatic Rebuild](#)

Length Unit

The default length unit is meter. To change the length unit, enter

```
model.geom(<tag>).lengthUnit(newLengthUnit);
```

where *newLengthUnit* is a string like "mm", "in", or "ft".

To get the current length unit, enter

```
String currentUnit = model.geom(<tag>).lengthUnit();
```

The length unit is used in fields for lengths, and for visualization of the geometry. In fields you can override the unit, for example by entering 13[mm]. When solving the model, all lengths are converted to meters.

Angular Unit

The default angular unit is degrees. To change the angular unit, enter

```
model.geom(<tag>).angularUnit(newAngularUnit);
```

where *newAngularUnit* is deg or rad.

To get the current angular unit, enter

```
String currentUnit = model.geom(<tag>).angularUnit();
```

The angular is used in fields for angles. You can override the unit, for example by entering 0.3[rad]. Numeric inputs and outputs of trigonometric functions are always assumed to be in radians, though.

Scale Values When Changing Unit

When you change the length unit or angular unit there are two possibilities to interpret pure numeric values in the geometry and meshing sequences. The first possibility is to reinterpret the numeric value in the new unit; a circle of radius 1.0 (meter) becomes a circle of radius 1.0 (millimeter), assuming the length unit changes from meter to millimeter.

The other possibility is to scale the numbers; a circle of radius 1.0 (meter) becomes a circle of radius 1000.0 (millimeter). To control the behavior, use

```
model.geom(<tag>).scaleUnitValue(newScaleValue);
```

where *newScaleValue* is **true** if you want values to be scaled, and **false** otherwise. The default value is **false**.

To get the currently used method, enter

```
boolean currentScaleValue = model.geom(<tag>).scaleUnitValue();
```

Geometry Representation in 3D

This settings is only relevant if you have a license for the CAD Import Module. The geometry representation determines which kernel (geometric modeler) that COMSOL uses to represent and operate on the geometry objects: the CAD Import Module's kernel (Parasolid) or COMSOL's own kernel. To change the geometry representation, enter

```
model.geom(<tag>).geomrep(newGeomRep);
```

where *newGeomRep* is `comsol` or `cadps`.

- If you choose `cadps`, all objects and operations that support the CAD kernel (Parasolid kernel) use it. For example, the Work Plane, Extrude, and Revolve features currently do not support this kernel.
- If you choose `comsol`, all objects are represented using COMSOL's kernel.

When you change the geometry representation, all nodes that support the CAD kernel get an *edited* status. To rebuild the geometry using the new kernel, use the `run` method.

When you create a new model (application), its default geometry representation is controlled by the preference setting **Geometry>Geometry representation>In new applications** (`geometry.geomrep.default`). To change or read this preference setting, enter

```
ModelUtil.setDefaultGeometryKernel(defaultGeomRep);  
ModelUtil.getDefaultGeometryKernel();
```

where *defaultGeomRep* is `cadps` or `comsol`.

When you open an existing model, you normally use the geometry representation used in the model. To always convert the geometry to the COMSOL kernel, change the preference setting **Geometry>Geometry representation>When opening an existing model** to **Convert to COMSOL kernel** (`geometry.geomrep.open`). To change or read this preference setting, enter

```
ModelUtil.setOpenGeometryKernel(openGeomRep);  
ModelUtil.getOpenGeometryKernel();
```

where *openGeomRep* is `model` or `comsol`.

Default Relative Repair Tolerance

The *default relative repair tolerance* is $1e-6$ by default. You can change it by entering `model.geom(<tag>).repairTol(<newDefaultRepairtol>);`

To get the current default relative repair tolerance, enter

```
double reptol = model.geom(<tag>).repairTol();
```

The default relative repair tolerance is the default value that is used when you add a new feature that has a `repairTol` property—for example, Boolean operations and conversions. Changing the default relative repair tolerance does not affect the tolerances in existing features. Adjust the relative repair tolerance if you experience problems with a Boolean operation. The absolute repair tolerance is the relative repair tolerance times the maximum coordinate of the input objects. The geometry kernel merges entities that have a distance less than the absolute repair tolerance.

Automatic Rebuild

This setting controls if the geometry sequence is automatically rebuilt when clicking a node in the model tree outside the geometry sequence. You can change it by entering:

```
model.geom(<tag>).autoRebuild(<newAutoRebuild>);
```

where *<newAutoRebuild>* is on or off.

The default geometry representation is controlled by the preference setting **Geometry>Automatic rebuild>Default in new geometries**.

Work Planes

In 3D, you can create 3D objects by defining 2D objects in *work planes* and then *extruding* and *revolving* these into 3D objects. You can also get a *cross section* 2D object by intersecting a 3D object with the work plane.

To add a WorkPlane feature, use

```
model.geom(<tag>).create(<ftag>,"WorkPlane");
```

You access a work planes 2D geometry sequence by the `geom()` method. To add a 2D feature to a work plane, use

```
model.geom(<tag>).feature(<ftag>).geom().create(<ftag1>,ftype);
```

where `<ftag>` refers to a WorkPlane feature and `ftype` refers to a 2D feature type.



A work plane's geometry sequence does not contain a `Finalize` feature.

A work plane's geometry sequence inherits its settings from its 3D sequence.



For more information on the settings for a geometry sequence see [Geometry Settings](#).

When you build a work plane feature its corresponding 2D sequence builds automatically and the geometry objects defined by the 2D sequence *embed* into 3D geometry objects in the 3D sequence. You can then extrude or revolve these embedded point, curve, or surface objects into curve, surface, or solid objects, respectively, by `Extrude` or `Revolve` features, respectively.



Methods on `model.geom(<tag>)` can also be used on `model.geom(<tag>).feature(<wptag>).geom()`.

Selections of Geometric Entities

Named Selections

You can create named selections of entities in geometry objects or of whole geometry objects in two ways:

- By setting the `selresult` property to `on` in an arbitrary geometry feature. This creates a selection containing the feature's output objects, and derived selections of entities for each entity type (domain, boundary, edge, and point). For the Import feature, there is an additional `selindividual` property, which creates selections corresponding to individual geometry objects.
- By adding a selection feature, see [Using Selection Features](#). If this defines a selection of whole objects, derived selections of entities are also created for each entity type (domain, boundary, edge, and point).

You get the tags of the created named selections by

```
String[] selTags = model.geom(<tag>).feature(<ftag>).outputSelection();
```

You can access a named selection by `model.geom(<gtag>).selection(<seltag>)`, where `<seltag>` is the selection's tag. Usually, `<seltag>` is the same as the tag of the feature that created the selection, but derived selections of entities have a suffix `.dom`, `.bnd`, `.edg`, or `.pnt`.

You can use a named selection as an input selection in a geometry feature that comes after the feature that created the selection in the geometry sequence, as follows:

```
model.geom(<gtag>).selection(<propname>).named(<trimmedseltag>);
```

Here, `<trimmedseltag>` is the selection's tag without the suffix.

Each named selection of entities in the geometry sequence can also be used as a named selection on the finalized geometry. You can access this named selection by `model.selection(<gtag>_<seltag>)`. However, if the selection was derived from a selection of whole objects, you access the corresponding selection on the finalized geometry by `model.selection(<gtag>_<trimmedseltag>_<lvl>)`, where `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt`. Named selections on the finalized geometry are described in [model.selection\(\)](#).

Using Selection Features

A selection features in the geometry sequence creates a selection of geometric entities or objects that is a subset of all entities or objects that were generated by the sequence of features preceding the selection feature.

- Use the `ExplicitSelection` feature to create a selection of entities or objects that you specify explicitly.
- Use the `BallSelection`, `BoxSelection`, or `CylinderSelection` feature to create a selection of entities or objects that partly or completely lie inside a given ball, box, or cylinder. The input entities or objects to select among can be all entities or objects generated by the preceding features, or a subset of these entities or objects defined by a set of input selections.
- Use the `UnionSelection`, `IntersectionSelection`, `DifferenceSelection`, and `ComplementSelection` features to combine a set of input selections using a Boolean operation.
- Use `AdjacentSelection` to create a selection of entities of a given dimension that are adjacent to entities in a set of input selections.

When using input selections, the input selections must be defined by features that come before the selection feature in the geometry sequence. You set the input selections in the `input` property (`add` and `subtract` for `DifferenceSelection`), where you use trimmed tags (that is, tags without the suffix) to refer to the input selections. Usually, the trimmed tag is the same as the tag of the feature that defined the input selection. It can happen that

an input selection selects entities or objects that no longer exist after building the feature preceding the selection feature. In this case, the input selection is interpreted by using an associative mapping to existing entities or objects.

Cumulative Selections

To create a cumulative selection tagged `<seltag>`, use

```
model.geom(<tag>).selection().create(<seltag>, "CumulativeSelection");
```

To make a geometry feature contribute its selection to a cumulative selection, use

```
model.geom(<tag>).feature(<ftag>).set("contributeto", <seltag>);
```

To remove the contribution of a geometry feature to a cumulative selection, use

```
model.geom(<tag>).feature(<ftag>).set("contributeto", "none");
```

To control whether the cumulative selection is available outside the geometry sequence, use

```
model.geom(<tag>).selection(<seltag>).show(<boolean>);
```

To query whether the cumulative selection is available outside the geometry sequence, use

```
boolean show = model.geom(<tag>).selection(<seltag>).show();
```

Virtual Operations

About Virtual Operations

In 2D and 3D it is possible to reduce the number of vertices, edges, faces, and domains of the geometry by using *virtual operation features*. To add the first virtual operation feature to a sequence you need to build the finalize feature by entering

```
model.geom(<tag>).run("fin");
```

You can then add the virtual operation feature by entering

```
model.geom(<tag>).create(<ftag>, ftype);
```

where *<ftag>* is the feature's tag (an identifier of your choice), and *ftype* is the feature's type. To build the feature, enter

```
model.geom(<tag>).run(<ftag>);
```

To build all features, including the finalize feature and all virtual operation features, and to create the finalized geometry, enter

```
model.geom(<tag>).run();
```

The finalized geometry of a sequence that contains virtual operation features is referred to as a *virtual geometry*. If you form a composite edge, face, or domain by using a [CompositeEdges](#), [CompositeFaces](#), or [CompositeDomains](#) feature, respectively (or the analogues [IgnoreVertices](#), [IgnoreEdges](#), or [IgnoreFaces](#) features) the resulting edge, face, or domain is referred to as a *virtual composite edge*, *virtual composite face*, or *virtual composite domain*, respectively, or more generally, a *virtual composite entity*.

Mesh Control Entities

Sometimes it is desirable to use certain geometric entities only when constructing the mesh. For example, you can add a curve inside a domain to control mesh element size there. If you mark this curve as a *mesh control entity*, it is not included in the geometry used when defining the physics. An advantage is that the final mesh need not respect this curve exactly; it is used only to control element size. You can use the `keepformesh` property of the Composite and Ignore features described above to define mesh control entities. Alternatively, you can use the [MeshControlDomains](#), [MeshControlVertices](#), [MeshControlEdges](#), or [MeshControlFaces](#) features.

Geometry Object Information

You can get the geometry object named *<objname>* via

```
model.geom(<tag>).obj(<objname>)
```



Accessing Geometry Object Names

The geometry itself,

```
model.geom(<tag>)
```

works as an object, namely the final geometry resulting from the sequence. To get information about these objects, you can apply the methods described in this section.

The corresponding COMSOL Multiphysics 3.5a functionality is also indicated in the tables below. This functionality existed in three forms: as options to the function `geominfo`, as `get` methods on geometry objects, and as utility functions starting with `flgeom`.

General Information

TABLE 3-19: GENERAL GEOMETRY INFORMATION METHODS

METHOD	OUTPUT TYPE
<code>check()</code>	Boolean
<code>exists()</code>	Boolean
<code>getBoundingBox()</code>	double[sdim*2]
<code>getSDim()</code>	int
<code>getType()</code>	String
<code>hasCadRep()</code>	Boolean

- `check()` returns true if the object is valid, and false if it contains errors.
- `exists()` returns true if an object exists.
- `getBoundingBox()` returns a bounding box for the object in the order `xmin`, `xmax`, `ymin`, `ymax`, `zmin`, and `zmax`.
- `getSDim()` returns the space dimension of the geometry.
- `getType()` returns the object type: `solid`, `surface`, `curve`, `point`, `mixed`, or `empty`.
- `hasCadRep()` returns true if the object is represented using the CAD kernel (Parasolid).

Geometric Entity Counters

The following geometric entity counter methods are available:

TABLE 3-20: GEOMETRIC ENTITY COUNTER METHODS

METHOD	1D	2D	3D	OUTPUT TYPE	3.5A NAME
getNEntities()	√	√	√	int[]	
getNVertices()	√	√	√	int	nv, flgeomnv
getNEdges()	√	√	√	int	ne, flgeomnes
getNFaces()			√	int	nf
getNBoundaries()	√	√	√	int	nbs, flgeomnbs
getNDomains()	√	√	√	int	ns, flgeomnmr
getNEntitiesMesh()	√	√	√	int[]	

- `getNEntities` returns a vector of length 2 in 1D, length 3 in 2D, and length 4 in 3D. The vectors contain the number of geometric entities for each entity dimension. The methods `getNVertices`, `getNEdges`, `getNFaces`, `getNBoundaries`, and `getNDomains` return the number of entities of the specified type.
- `getNEntitiesMesh` returns a vector of length 2 in 1D, length 3 in 2D, and length 4 in 3D. The vectors contain the number of geometric entities for each entity dimension in the geometry used for meshing. If there are no mesh control entities in the geometry, the output is identical to that of `getNEntities`.

Adjacency

The following geometry adjacency information methods are available:

TABLE 3-21: GEOMETRY ADJACENCY INFORMATION METHODS

METHOD	1D	2D	3D	OUTPUT TYPE	3.5A NAME
getStartEnd()	√	√	√	int[2][]	se, flgeomse
getUpDown()	√	√	√	int[2][]	ud, flgeomud
getUpDownExt()	√	√	√	int[2][]	
getVertexDomain()		√	√	int[]	sd, flgeomsd
getSD()		√	√	double[]	sd, flgeomsd
getAdj(int, int)	√	√	√	int[][]	adj, flgeomadj
getAdj(int, int, int)	√	√	√	int[]	adj, flgeomadj
getAdjOrient(int, int)	√	√	√	int[][]	adj, flgeomadj
getAdjOrient(int, int, int)	√	√	√	int[][]	adj, flgeomadj
getAdjSparse(int, int)	√	√	√	int[3][]	adj, flgeomadj

- `getStartEnd` returns the start and end vertices of all edges in the first and second row of the returned matrix.
- `getUpDown` returns the up and down domain number for all boundaries in the first and second row of the returned matrix. All void regions have the domain number 0.
- `getUpDownExt` returns the up and down domain number for all boundaries in the first and second row of the returned matrix, using an extended domain numbering where void regions have different domain numbers. The infinite void region has domain number 0. The finite void regions have negative domain numbers. However, if `voidAreLabeled()` returns `false`, the extended domain numbering is not available, and then all void regions have domain number 0.
- `getVertexDomain` returns the domain index for each vertex. For non-isolated vertices, the domain index is -1.
- `getSD` returns the domain index for each vertex. For non-isolated vertices, the domain index is NaN.

- `a = getAdj(fromDim, toDim)` returns a matrix where `a[fromIdx]=getAdj(fromDim,toDim,fromIdx)` contains the entities in dimension `toDim` that are adjacent to entity `fromIdx` in dimension `fromDim`.
- `ao = getAdjOrient(fromDim, toDim)` returns a matrix where `ao[fromIdx]=getAdjOrient(fromDim,toDim,fromIdx)` contains the orientation flag for the entities in `getAdj(fromDim,toDim,fromIdx)`. The orientation flag is 1 if the adjacent entities have the same orientation, and -1 if they have the opposite orientation, and 2 if the relative orientation cannot be determined (for instance, for an edge interior to a face).
- `as = getAdjSparse(fromDim, toDim)` returns the adjacency matrix from entities in dimension `fromDim` to entities in dimension `toDim` on a sparse format, that is, `as[0]` are the entity numbers in dimension `fromDim`, `as[1]` are the entity numbers in dimension `toDim`, and `as[2]` are the corresponding orientation flags.

Evaluation on an Edge

The following edge evaluation methods are available in 2D and 3D:

TABLE 3-22: EDGE EVALUATION METHODS IN 2D AND 3D

METHOD	2D	3D	OUTPUT TYPE	3.5A NAME
<code>edgeParamRange(int)</code>	√	√	<code>double[2]</code>	<code>rng</code>
<code>edgeX(int, double[])</code>	√	√	<code>double[][D]</code>	<code>xx</code>
<code>edgeDX(int, double[])</code>	√	√	<code>double[][D]</code>	<code>dxx</code>
<code>edgeDDX(int, double[])</code>	√	√	<code>double[][D]</code>	<code>ddx</code>
<code>edgeNormal(int, double[])</code>	√		<code>double[][D]</code>	<code>nor</code>
<code>edgeCurvature(int, double[])</code>	√	√	<code>double[]</code>	<code>crv</code>
<code>edgeTorsion(int, double[])</code>		√	<code>double[]</code>	<code>crv</code>

The first input argument of all methods is the edge number. The second input argument, when it exists, is an array of parameter values for which to perform evaluation on the edge. For all but the first method, the first index in the output corresponds to the different parameter values, and the second index corresponds to the spatial coordinates.

- `edgeParamRange` returns the parameter range for evaluation on the edge.
- `edgeX` evaluates the parameters to coordinate values.
- `edgeDX` evaluates the parameters to first order derivative values.
- `edgeDDX` evaluates the parameters to second order derivative values.
- `edgeNormal` evaluates the parameters to normal vector values.
- `edgeCurvature` evaluates the parameters to curvature values.
- `edgeTorsion` evaluates the parameters to torsion values.

Evaluation on a Face

Use the following method for face evaluation in 3D. They do not work on virtual geometry objects.

TABLE 3-23: FACE EVALUATION METHODS IN 3D

METHOD	OUTPUT TYPE	3.5A NAME
<code>faceParamRange(int)</code>	<code>double[4]</code>	<code>rng, flgeomfs</code>
<code>faceX(int, double[][2])</code>	<code>double[][3]</code>	<code>xx</code>
<code>faceDX(int, double[][2])</code>	<code>double[][3][2]</code>	<code>dx</code>
<code>faceDDX(int, double[][2])</code>	<code>double[][3][2][2]</code>	<code>ddx</code>
<code>faceNormal(int, double[][2])</code>	<code>double[][3]</code>	<code>nor</code>

TABLE 3-23: FACE EVALUATION METHODS IN 3D

METHOD	OUTPUT TYPE	3.5A NAME
faceFF1(int, double[][2])	double[][2]	ff1
faceFF2(int, double[][2])	double[][2]	ff2
faceGaussCurvature(int, double[][2])	double[]	crv
faceMeanCurvature(int, double[][2])	double[]	crv

The first input argument of all methods is the face number. The second input argument, when it exists, is a matrix of parameter points, for which to perform evaluation. For all but the first method, the first index in the output corresponds to the different parameter points.

- `faceParamRange` returns two parameter ranges for evaluation on the face.
- `faceX` evaluates the parameters to coordinate values.
- `faceDX` evaluates the parameters to first order derivative values.
- `faceDDX` evaluates the parameters to second order derivative values.
- `faceNormal` evaluates the parameters to normal vector values.
- `faceFF1` evaluates the parameters to the first fundamental form values.
- `faceFF2` evaluates the parameters to the second fundamental form values.
- `faceGaussCurvature` evaluates the parameters to Gauss curvature values.
- `faceMeanCurvature` evaluates the parameters to mean curvature values.

Geometry Representation Arrays

Use the following methods to access the arrays in the internal representation of COMSOL Multiphysics geometry objects. They do not work on objects represented using the CAD kernel, assembly geometries, or virtual geometries.

TABLE 3-24: GET ARRAYS IN GEOMETRY REPRESENTATION

METHOD	1D	2D	3D	OUTPUT TYPE	3.5A NAME
getVertex()	√	√	√	double[][]	vertex
getEdge()		√	√	double[][]	edge
getFace()			√	double[][]	face
getPVertex()			√	double[][]	pvertex
getPEdge()			√	double[][]	pedge
getVertexCoord()	√	√	√	double[][]	mp, flgeomvtx
voidsAreLabeled()	√	√	√	Boolean	

- In 2D and 3D, `getVertex` returns (sdim+2)-by-nv matrix representing the vertices of the object. The first sdim rows are the coordinates of the vertices. Row sdim+1 contains the domain number if the vertex is isolated and is unspecified otherwise. The last row contains a relative local tolerance for the vertex. For nontolerant vertices the tolerance is NaN. This method does not work on virtual geometry objects.
- In 1D, `getVertex` returns a 3-by-nvtx matrix representing the vertices of the 1D object. Row 1 provides the coordinates of the vertices. Rows 2 and 3 provide the up and down domain numbers, respectively.
- `getPVertex` returns a 6-by-npv matrix containing embeddings of vertices in faces. Row 1 contains the vertex index (that is, column from `getVertex`), rows 2 and 3 contain (s, t) coordinates of the vertex on the face, row 4 contains a face index, and row 5 contains the manifold index into the manifolds. Row 6 contains a relative local tolerance for the vertex. This method does not work on virtual geometry objects.

- In 3D, `getEdge` returns a 7-by-`ne` matrix representing the edges of the 3D object. Rows 1 and 2 contain the start and end vertex indices of the edge (0 if they do not exist), respectively. Rows 3 and 4 give the parameter values of these vertices. Row 5 gives the index of a domain if the edge is not adjacent to a face, and is unspecified otherwise. Row 6 gives a sign and an index to the underlying manifold. The sign indicates the direction of the edge relative the curve. Finally, row 7 contains a relative local tolerance for the edge. This method does not work on virtual geometry objects.
- In 2D, `getEdge` returns a 8-by-`ne` matrix representing the edges of the 2D object. Rows 1 and 2 contain the start and end vertex indices of the edge, respectively (0 if they do not exist). Rows 3 and 4 give the parameter values of these vertices. Rows 5 and 6 contain the left and right domain number of the edge, respectively. Row 7 gives a sign and an index to the array of underlying curves. The sign indicates the direction of the edge relative the curve. Row 8 contains a relative local tolerance for the edge.
- `getPEdge` returns a 10-by-`npe` matrix representing the embeddings of the edges in faces. The first row gives the index of the edge in `getEdge`. Rows 2 and 3 contain the start and end vertex indices from `getPVertex`, respectively. Rows 4 and 5 give the parameter values of these vertices. Row 6 and 7 give the indices of the faces to the left and right of the edge, respectively. Row 8 gives a sign and index to the parameter curve (if any), and row 9 gives the index to the surface. Row 10 contains a relative local tolerance for the edge. This method does not work on virtual geometry objects.
- `getFace` returns a 4-by-`nf` matrix representing the faces of the 3D geometry. Rows 1 and 2 contain the up and down domain index of the face, respectively, and row 3 contains the manifold index of the face. Row 4 contains a relative local tolerance for the face. This method does not work on virtual geometry objects.
- `getVertexCoord` returns a matrix with the vertex coordinates. Its dimension is the space dimension times the number of vertices.

Measurements

Geometric measurements is a tool to measure geometric entities and objects. You access it by entering

```
model.geom(<tag>).measure()
```

Measuring Geometric Entities in Objects

To select entities you want to measure, enter

```
model.geom(<tag>).measure().selection().init(entDim);  
model.geom(<tag>).measure().selection().set(<objname>,entities);
```

where *entDim* is the dimension of the entities, *<objname>* is the object name, and *entities* is an integer array containing the entity numbers.

To get the volume/area/length of the selected entities, enter

```
double vol = model.geom(<tag>).measure().getVolume();
```

To get the area/length of the boundary of the selected entities, enter

```
double bndVol = model.geom(<tag>).measure().getBoundaryVolume();
```

If you have selected two vertices, you can get their distance by entering

```
double[] d = model.geom(<tag>).measure().getVtxDistance();
```

d[0] is the distance, and *d[i]* is the distance in the *i*th coordinate (*i* = 1, 2, 3).

If you have selected one vertex, you can get its coordinates by entering

```
double[] coord = model.geom(<tag>).measure().getVtxCoord();
```

Measuring Objects

To select objects you want to measure, enter

```
model.geom(<tag>).measure().selection().init();  
model.geom(<tag>).measure().selection().set(<objnames>);
```

where *<objnames>* is a string array containing the object names.

To get the total number of entities in the selected objects, enter

```
int[] entitiesPerDimension = model.geom(<tag>).measure().getNEntities();
```

Inserting Geometry Sequences from File

To insert a geometry sequence from an MPH-file, enter

```
model.geom(<tag>).insertFile(<filename>, <sequencename>);
```

where <filename> and <sequencename> are strings.

To insert a geometry sequence from a different model, enter

```
model.geom(<tag>).insertSequence(<modeltag>, <sequencename>);
```

where <modeltag> and <sequencename> are strings.

Example of Importing Geometry Sequences

The following sequence imports three different geometry sequences from two different files:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("g", 2);
g.insertFile("filename", "geom1");
ModelUtil.load("Model2", "filename2");
g.insertSequence("Model2", "geom1/wp1");
g.insertSequence("Model2", "geom1/wp2");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('g', 2);
g.insertFile('filename', 'geom1');
ModelUtil.load('Model2', 'filename2');
g.insertSequence('Model2', 'geom1/wp1');
g.insertSequence('Model2', 'geom1/wp2');
```

Exporting Geometry to File

To export the result of the Finalize feature to a file, enter

```
model.geom(<tag>).exportFinal(<filename>);
```

where *<filename>* is a string.

To export selected geometry objects to a file, first select the objects to export using

```
model.geom(<tag>).export().selection().set(<objnames>);
```

where *<objnames>* is a string array of object names. Then export them by entering

```
model.geom(<tag>).export(<filename>);
```

The file can be of any of the following formats:

TABLE 3-25: VALID FILE FORMATS

FILE FORMAT	NOTE	FILE EXTENSIONS
COMSOL Multiphysics Binary		.mphbin
COMSOL Multiphysics Text		.mphtxt
Parasolid Binary (3D)	1	.x_b
Parasolid Text (3D)	1	.x_t
ACIS Binary (3D)	1, 3	.sab
ACIS Text (3D)	1, 3	.sat
STL Binary (3D)	2	.stl
STL Text (3D)	2	.stl
DXF (2D)		.dxf

¹ This format requires a license for the COMSOL CAD Import Module.

² Use `model.geom(<tag>).export().setSTLFormat(<format>)` to specify the STL file format ("binary" or "text").

³ Use `model.geom(<tag>).export().setAcisVersion(<version>)` to specify the ACIS file format version ("major.minor"; default "23.0").

Exporting to an ACIS File

When exporting to an ACIS file you can set the ACIS file format version using

```
model.geom(<tag>).export().setAcisVersion(<version>);
```

where *<version>* is a string 1.0, 2.0, ..., or 23.0. Default is 23.0.

Exporting to a Parasolid File

When exporting to Parasolid text or binary format, a unit conversion can optionally be performed during export. Use the following method to select the export length unit:

```
model.geom(<tag>).export().setLengthUnit(<unit>);
```

where *<unit>* is either `fromgeom` to disable unit conversion or a COMSOL Multiphysics length unit, such as `m` for meters or `in` for inches.

Exporting to an STL File

To export to an STL file, start with specifying a file format using

```
model.geom(<tag>).export().setSTLFormat(<format>);
```

where *<format>* is string with only two allowed values: **binary** and **text**

Use the following methods to select domains or boundaries to export:

```
model.geom(<tag>).export().selection().init(<edim>);  
model.geom(<tag>).export().selection().set(<objnames>, <entlst>);
```

Use the following methods to select objects to export:

```
model.geom(<tag>).export().selection().init();  
model.geom(<tag>).export().selection().set(<objnames>);
```

Finish the export by using the following line

```
model.geom(<tag>).export(<filename>);
```

Compatibility for mphbin/mph.txt in 2D and 3D

If you want to open a COMSOL Multiphysics geometry file in an earlier versions of COMSOL, you might need to set the COMSOL version using

```
model.geom(<tag>).export().setCompat(<ver>);
```

where *<ver>* is a string **3.5a**, **4.0** (only 3D), **4.0a** (only 3D), **4.2a**, **4.3b**, or **5.1**.

Using Geometry Parts

For a description of geometry parts, see [Using Geometry Parts](#) in the *COMSOL Multiphysics Reference Guide*.

To create a geometry part, enter

```
model.geom().create(<tag>, "Part", sDim);
```

where *<tag>* is the parts tag, and *sDim* is its space dimension (1, 2, or 3).

To add an input parameter to the part, enter

```
model.geom(<tag>).inputParam().set(<name>, <expr>, <descr>);
```

where the description *<descr>* can be omitted. Similarly, add a local parameter by

```
model.geom(<tag>).localParam().set(<name>, <expr>, <descr>);
```

The containers `model.geom(<tag>).inputParam()` and `model.geom(<tag>).localParam()` also support the other methods listed in [model.param\(\)](#).

To load one or more geometry parts, enter

```
model.geom().load(<tags>, <filename>, <partTagsInFile>);
```

where *<tags>* is a list of part tags, *<filename>* is the filename of the MPH-file where the parts are defined, and *<partTagsInFile>* is a list of the parts' tags in that file.

To get the filename of a linked part, enter

```
model.geom(<gtag>).filename();
```

To change the filename of a linked part, enter

```
model.geom(<gtag>).filename(<filename>);
```

where *<filename>* is the new filename.

For a loaded part, to return the comments from the MPH-file, enter

```
model.geom(<gtag>).commentsInFile();
```

For a loaded part, to return the last modification date from the MPH-file, enter

```
model.geom(<gtag>).dateModifiedInFile();
```

To get the tag that a linked part has in the MPH-file, enter

```
model.geom(<gtag>).tagInFile();
```

To get the label that a linked part has in the MPH-file, enter

```
model.geom(<gtag>).labelInFile();
```

For a loaded part, to return the version from the MPH-file, enter

```
model.geom(<gtag>).versionInFile();
```

To reload (update) a loaded part after its definition has been changed, enter

```
model.geom(<gtag>).reload();
```

To call a geometry part, add a `PartInstance` feature:

```
model.geom(<gtag>).create(<ftag>, "PartInstance");
```

See [PartInstance](#) for details.

To debug a call to `part`, you can step into it using

```
model.geom(<gtag>).stepInto(<ftag>);
```

You can then apply the usual geometry sequence methods on the local part instance, for example,

```
model.geom(<gtag>).feature(<ftag>).geom().run(<ftag2>);
```

to build the feature `<ftag2>` in the local part instance.

Geometry Commands

Geometry Commands (A to L)

- [AdjacentSelection](#)
- [Array](#)
- [BallSelection](#), [BoxSelection](#), [CylinderSelection](#)
- [BezierPolygon](#)
- [Block](#)
- [Chamfer](#)
- [Circle](#)
- [CollapseEdges](#)
- [CollapseFaces](#)
- [Compose](#), [Union](#), [Intersection](#), [Difference](#)
- [CompositeDomains](#)
- [CompositeEdges](#)
- [CompositeFaces](#)
- [Cone](#)
- [ConvertToSolid](#), [ConvertToSurface](#), [ConvertToCurve](#), [ConvertToPoint](#)
- [CrossSection](#)
- [Cylinder](#)
- [Delete](#)
- [ECone](#)
- [EditObject](#)
- [Ellipse](#)
- [Ellipsoid](#)
- [ExplicitSelection](#)
- [Extrude](#)
- [Fillet](#)
- [Finalize](#)
- [FromMesh](#)
- [Helix](#)
- [Hexahedron](#)
- [If](#), [ElseIf](#), [Else](#), [EndIf](#)
- [IgnoreEdges](#)
- [IgnoreFaces](#)
- [IgnoreVertices](#)
- [Import DXF](#)
- [Import Geometry Sequence](#)
- [Import Mesh Part or Meshing Sequence](#)
- [Import mphbin/mphtxt](#)
- [Interpolation Curve](#)
- [Interpolation Curve](#)
- [Interval](#)

Geometry Commands (M to Z)

- [MergeEdges](#)
- [MergeVertices](#)
- [MeshControlDomains](#)
- [MeshControlEdges](#)
- [MeshControlFaces](#)
- [MeshControlVertices](#)
- [Mirror](#)
- [Move, Copy](#)
- [ParametricCurve](#)
- [ParametricSurface](#)
- [PartInstance](#)
- [Partition](#)
- [Point](#)
- [Polygon](#)
- [Pyramid](#)
- [Rectangle](#)
- [Revolve](#)
- [Rotate](#)
- [Scale](#)
- [Sphere](#)
- [Split](#)
- [Square](#)
- [Sweep](#)
- [Tangent](#)
- [Tetrahedron](#)
- [Torus](#)
- [UnionSelection, IntersectionSelection, DifferenceSelection, ComplementSelection](#)
- [WorkPlane](#)

AdjacentSelection

Create a selection of entities or objects that are adjacent to given selections.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "AdjacentSelection");  
model.geom(<tag>).feature(<ftag>).set(property, <value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

`AdjacentSelection` creates a selection of all entities of dimension `outputdim` that are adjacent to at least one entity in the input selections. If the output selection has lower dimension than the input selections, you can use the `exterior` and `interior` properties to exclude or include output entities that are exterior/interior to the union of the input selections.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>entitydim</code>	0 1 2 3	space dimension	Dimension of input entities
<code>exterior</code>	on off	on	Include output entities that are exterior to the union of the input selections
<code>input</code>	String[]	{}	Tags of input selections
<code>interior</code>	on off	off	Include output entities that are interior to the union of the input selections
<code>outputdim</code>	0 1 2 3	space dimension - 1	Dimension of entities to select
<code>selkeep</code>	on off	on	Keep the selection within the geometry sequence.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selshow	on off	on	Show selection in physics, materials, and so on, or in part instances. This property is not available in a work plane's Plane Geometry.
contributeto	String	none	Tag of cumulative selection to contribute to.

See [Selections of Geometric Entities](#) for general information about selections.

EXAMPLE

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("sph1", "Sphere");
g.run("sph1");
g.create("sel1", "ExplicitSelection");
g.feature("sel1").selection("selection").init(0);
g.feature("sel1").selection("selection").set("sph1", new int[]{4});
g.create("adjsel1", "AdjacentSelection");
g.feature("adjsel1").set("entitydim", 0);
g.feature("adjsel1").set("input", new String[]{"sel1"});
g.run("adjsel1");
g.create("del1", "Delete");
g.feature("del1").selection("input").named("adjsel1");
g.run("del1");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('sph1', 'Sphere');
g.run('sph1');
g.create('sel1', 'ExplicitSelection');
g.feature('sel1').selection('selection').init(0);
g.feature('sel1').selection('selection').set('sph1', 4);
g.create('adjsel1', 'AdjacentSelection');
g.feature('adjsel1').set('entitydim', 0);
g.feature('adjsel1').set('input', 'sel1');
g.run('adjsel1');
g.create('del1', 'Delete');
g.feature('del1').selection('input').named('adjsel1');
g.run('del1');
```

SEE ALSO

[BallSelection](#), [BoxSelection](#), [CylinderSelection](#), [ExplicitSelection](#), [UnionSelection](#), [IntersectionSelection](#), [DifferenceSelection](#), [ComplementSelection](#)

Array

Create a block-shaped (3D), rectangular (2D, 3D), or linear array of geometry objects.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "Array");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Array")` to create an array of geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to array. The default selection is empty.

The following properties are available:

TABLE 3-26: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Objects to array
displ	double[sdim]	1	Displacements in axis directions
size	int int[sdim]	1	Array size
selresult	on off	off	Create selections of all resulting objects of this feature.
selresultshow	all obj dom bnd edg pnt off	The highest available entity level except obj; usually dom.	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.

If `size` is a scalar, a linear (oblique) array with `size` copies of the input objects is constructed. The displacement between two consecutive copies is given by the vector `displ`. The names of the output objects are `ftag(i)`, where `ftag` is the tag of the feature, and `i` is a 1-based index. If there are more than one input object, the output objects are named `ftag(i,in)`, where `in` is a 1-based index corresponding to the input objects.

2D: If `size` is an array of length 2, a rectangular array with `size[0]`-by-`size[1]` copies of the input object is constructed. The *x*- and *y*-displacements are `displ[0]` and `displ[1]`, respectively. The names of the output objects are `ftag(i1,i2)`, where `ftag` is the name of the feature, and `i1` and `i2` are 1-based indices. If there are more than one input object, the output objects are named `ftag(i1,i2,in)`, where `in` is a 1-based index corresponding to the input objects.

3D: If `size` is an array of length 3, a three-dimensional (block shaped) array with `size[0]`-by-`size[1]`-by-`size[2]` copies of the input object is constructed. The *x*-, *y*-, and *z*-displacements are `displ[0]`, `displ[1]`, and `displ[2]`, respectively. The names of the output objects are `ftag(i1,i2,i3)`, where `ftag` is the name of the feature, and `i1`, `i2`, and `i3` are 1-based indices. If there are more than one input object, the output objects are named `ftag(i1,i2,i3,in)`, where `in` is a 1-based index corresponding to the input objects.

The input object is deleted and an identical object is constructed as a part of the array.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

In COMSOL Multiphysics 5.2, the `selresult` property replaces the `createselection` property. `createselection` is still supported for backward compatibility.

In COMSOL Multiphysics version 3.5a it was possible to use `geomarrayr` to copy an object to an arbitrary set of positions. Now, use the `Copy` feature to do this.

`model.geom(<tag>).create(<ftag>,"arrayr")` constructs an Array feature

EXAMPLE

The following sequence creates a block with four equally-sized holes:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1", 3);
g.create("cyl1", "Cylinder");
g.create("arr1", "Array");
g.feature("arr1").selection("input").set("cyl1");
```

```

g.feature("arr1").set("displ", "4 4 0");
g.feature("arr1").set("size", "2 2 1");
g.create("blk1", "Block");
g.feature("blk1").set("size", "10 14 5");
g.feature("blk1").set("pos", "-3 -5 -4");
g.create("dif1", "Difference");
g.feature("dif1").selection("input").set("blk1");
g.feature("dif1").selection("input2").set("arr1");
g.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('cyl1', 'Cylinder');
g.create('arr1', 'Array');
g.feature('arr1').selection('input').set('cyl1');
g.feature('arr1').set('displ', '4 4 0');
g.feature('arr1').set('size', '2 2 1');
g.create('blk1', 'Block');
g.feature('blk1').set('size', '10 14 5');
g.feature('blk1').set('pos', '-3 -5 -4');
g.create('dif1', 'Difference');
g.feature('dif1').selection('input').set('blk1');
g.feature('dif1').selection('input2').set('arr1');
g.run;

```

SEE ALSO

[Move](#), [Copy](#)

BallSelection, BoxSelection, CylinderSelection

Create selections of geometric entities or objects that (partly) lie inside a ball/box/cylinder.

SYNTAX

```

model.geom(<tag>).create(<ftag>, "BallSelection");
model.geom(<tag>).create(<ftag>, "BoxSelection");
model.geom(<tag>).create(<ftag>, "CylinderSelection");
model.geom(<tag>).set(property, <value>);
model.geom(<tag>).getType(property);

```

DESCRIPTION

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
angletol	double	5	Angle tolerance for continuity evaluation
condition	intersects inside somevertex allvertices	intersects	Condition for inclusion of an entity
entitydim	-1 0 1 2 3	space dimension	Dimension of entities to select. -1 means Object.
groupcontang	on off	off	Continuous tangent mode
input	String[]	{}	Tags of input selections, only used when inputent is selections
inputent	all selections	all	Select among all entities or entities defined by input property
selkeep	on off	on	Keep the selection within the geometry sequence.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selshow	on off	on	Show selection in physics, materials, and so on, or in part instances. This property is not available in a work plane's Plane Geometry.
contributeto	String	none	Tag of cumulative selection to contribute to.

For BallSelection, you define the ball using the following properties (using the geometry sequence's length unit):

PROPERTY	VALUE	DEFAULT	DESCRIPTION
posx	double	0	Center of ball, first coordinate
posy	double	0	Center of ball, second coordinate
posz	double	0	Center of ball, third coordinate
r	double	0	Radius of ball

For BoxSelection, you define the box using the following properties (using the geometry sequence's length unit):

PROPERTY	VALUE	DEFAULT	DESCRIPTION
xmax	double	inf	Maximum x-coordinate of box
xmin	double	-inf	Minimum x-coordinate of box
ymax	double	inf	Maximum y-coordinate of box
ymin	double	-inf	Minimum y-coordinate of box
zmax	double	inf	Maximum z-coordinate of box
zmin	double	-inf	Minimum z-coordinate of box

For CylinderSelection, you define the cylinder using the following properties (using the geometry sequence's length unit):

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,0}	Direction of cylinder axis. Vector has length 3 if axistype is cartesian and length 2 if axistype is x, y, or z.
axistype	x y z cartesian spherical	z	Type of axis or coordinate system used for axis. The value is synchronized with axis.
bottom	double	-inf	Coordinate of bottom face in local coordinate system
pos	double[]	{0,0,0}	Base point
r	double	0	Radius
top	double	inf	Coordinate of top face in local coordinate system

You select the input entities/objects to select among using the properties entitydim, inputent, and input. For a boundary or edge selection in 2D or 3D, you can force the selection to select whole groups of entities by setting the property groupcontang to on. Each group consists of adjacent entities that meet at an angle less than angletol.

The output entities/objects are determined by the property condition:

- **intersects:** All entities/objects that intersect the ball/box/cylinder are included.
- **inside:** All entities/objects that are completely inside the ball/box/cylinder are included.
- **somevertex:** All entities/objects that have at least one adjacent vertex inside the ball/box/cylinder are included.
- **allvertices:** All entities/objects that have all adjacent vertices inside the ball/box/cylinder are included.

For `intersects` and `inside`, the rendering mesh is used for the calculation. You can set the resolution of the rendering mesh using

```
ModelUtil.setPreference("graphics.rendering.detail",<detail>);
```

where `<detail>` is `coarse`, `normal`, `fine`, or `wireframe`.

See [Selections of Geometric Entities](#) for general information about selections.

EXAMPLE

In a 10-by-10 array of squares, delete the squares that lie in the box $x > 9.5, y > 9.5$.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
g.create("sq1", "Square");
g.create("arr1", "Array");
g.feature("arr1").selection("input").set("sq1");
g.feature("arr1").set("fullsize", new int[]{10,10});
g.feature("arr1").set("displ", new double[]{2,2});
g.run("arr1");
g.create("boxsel1", "BoxSelection");
g.feature("boxsel1").set("entitydim", -1);
g.feature("boxsel1").set("xmin", 9.5);
g.feature("boxsel1").set("ymin", 9.5);
g.create("del1", "Delete");
g.feature("del1").selection("input").init();
g.feature("del1").selection("input").named("boxsel1");
g.run("del1");
// g.objectNames().length = 75
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
g.create('sq1', 'Square');
g.create('arr1', 'Array');
g.feature('arr1').selection('input').set('sq1');
g.feature('arr1').set('fullsize', [10,10]);
g.feature('arr1').set('displ', [2,2]);
g.run('arr1');
g.create('boxsel1', 'BoxSelection');
g.feature('boxsel1').set('entitydim', -1);
g.feature('boxsel1').set('xmin', 9.5);
g.feature('boxsel1').set('ymin', 9.5);
g.create('del1', 'Delete');
g.feature('del1').selection('input').init;
g.feature('del1').selection('input').named('boxsel1');
g.run('del1');
% length(g.objectNames)= 75
```

SEE ALSO

[AdjacentSelection](#), [ExplicitSelection](#), [UnionSelection](#), [IntersectionSelection](#),
[DifferenceSelection](#), [ComplementSelection](#)

BezierPolygon

Create a curve or solid polygon consisting of Bézier segments in 2D or 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"BezierPolygon");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"BezierPolygon")` to create a Bézier polygon or a line segment. The following properties are available

TABLE 3-27: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
degree	int[] int	1	Degree of Bézier segments.
p	double[][]		Control points.
type	solid open closed	solid (2D) open (3D)	Object type. solid is not available in 3D.
w	double[]		Weights.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom in 2D; edg in 3D.	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `type` is `open` or `closed`, a curve consisting of line, quadratic, or cubic rational Bézier segments is constructed. If `type` is `solid`, the solid enclosed by such a closed polygon is constructed. If `type` is `closed` or `solid`, but the first and last control points are different, an extra linear segment is added to close the curve.

The degree of the n th segment is `degree[n]`, and it must be 1 (linear), 2 (quadratic), or 3 (cubic). The n th segment has `degree[n]+1` control points and weights. The weights are stored consecutively in the array `w`, which has length `degree[0]+...+degree[N-1]+N`, where N is the number of segments. The i th coordinates of the control points are stored consecutively in the array `p[i]`. Adjacent segments share the common control point, which means that `p[i]` has length `degree[0]+...+degree[N-1]+1`.

For a linear or cubic segment, the default weights are 1. For a quadratic segment, the default weights are $1, 1/\sqrt{2}, 1$.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

Construct a solid triangle `b1` and an elliptic arc `b2`:

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",2);  
g.create("b1","BezierPolygon");  
g.feature("b1").set("p", new double[][]{{0, 0, 2}, {1, 0, 0}});  
g.create("b2","BezierPolygon");  
g.feature("b2").set("type","open");  
g.feature("b2").set("degree",2);  
g.feature("b2").set("p", new double[][]{{0, 1, 0}, {1, 2, 0}});  
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');  
g = model.geom.create('geom1',2);  
g.create('b1','BezierPolygon');  
g.feature('b1').set('p', [[0, 0, 2]; [1, 0, 0]]);  
g.create('b2','BezierPolygon');
```

```

g.feature('b2').set('type','open');
g.feature('b2').set('degree',2);
g.feature('b2').set('p', [[0, 1, 0]; [1, 2,0]]);
g.run;

```

SEE ALSO

[Polygon](#)

Block

Create a right-angled solid or surface block in 3D.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"Block");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Block")` to create a block. The following properties are available:

TABLE 3-28: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the edge on the local z-axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
base	corner center	corner	Positions the object either centered about pos or with one corner in pos.
layer	double[]		Thicknesses of layers.
layertop	on off	off	Apply layers on top.
layerbottom	on off	on	Apply layers on bottom.
layerleft	on off	off	Apply layers to the left.
layerright	on off	off	Apply layers to the right.
layerfront	on off	off	Apply layers on front.
layerback	on off	off	Apply layers on back.
size	double[]	{1,1,1}	Edge lengths.
pos	double[]	{0,0,0}	Position of the object.
rot	double	0	Rotational angle about axis.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"block2")` constructs a solid block.

`model.geom(<tag>).create(<ftag>,"block3")` constructs a surface block.

The following properties are also available:

TABLE 3-29: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical.
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian.
lx, ly, lz	double	1	Alias for size.
x, y, z	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLE

The following commands create a solid and surface block, where the position is defined in the two alternative ways.

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("b1","Block");
g.feature("b1").set("size","1 2.1 0.5");
g.feature("b1").set("base","center");
g.feature("b1").set("pos","1 0 1");
g.feature("b1").set("axis","1 0 0");
g.feature("b1").set("rot",30);
double[] a = g.feature("b1").getDoubleArray("pos");
g.create("b2","Block");
g.feature("b2").set("type","surface");
g.feature("b2").set("size","1 2.1 0.5");
g.feature("b2").set("pos",a);
String b = g.feature("b2").getString("pos");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('b1','Block');
g.feature('b1').set('size','1 2.1 0.5');
g.feature('b1').set('base','center');
g.feature('b1').set('pos','1 0 1');
g.feature('b1').set('axis','1 0 0');
g.feature('b1').set('rot',30);
a = g.feature('b1').getDoubleArray('pos');
g.create('b2','Block');
g.feature('b2').set('type','surface');
g.feature('b2').set('size','1 2.1 0.5');
g.feature('b2').set('pos',a);
b = g.feature('b2').getString('pos');
```

SEE ALSO

[Hexahedron](#)

Chamfer

Create flattened corners in 2D objects. The Design Module also supports 3D chamfers.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Chamfer");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Chamfer")` to chamfer corners in 2D.

Use `model.geom(<tag>).feature(<ftag>).selection("point")` to select the corners to chamfer. The default selection is empty.

TABLE 3-30: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
dist	double	0	Distance from vertex to chamfer
point	Selection		Vertices to chamfer
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

In version 3.5a, the default was to chamfer all vertices in the input objects. The 3.5a properties `angles`, `dist1`, `dist2`, `edges`, `length`, and `out` are no longer available.

EXAMPLE

Chamfer a rectangle.

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",2);  
g.create("r1","Rectangle");  
g.create("cha1","Chamfer");  
g.feature("cha1").selection("point").set("r1(1)",new int[]{1,2,3,4});  
g.feature("cha1").set("dist",0.1);  
g.run();
```

Code for use MATLAB

```
Model model = ModelUtil.create('Model1');  
g = model.geom.create('geom1',2);  
g.create('r1','Rectangle');  
g.create('cha1','Chamfer');  
g.feature('cha1').selection('point').set('r1(1)',{1,2,3,4});  
g.feature('cha1').set('dist',0.1);  
g.run;
```

DIAGNOSTICS

If a chamfer cannot be created according to the specified properties this vertex is ignored. When the chamfers generate intersections with other edges in the geometry, an error message is given.

SEE ALSO

[Fillet](#)

Circle

Create a circle or disk in 2D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Circle");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Circle")` to create a disk in 2D. The following properties are available:

TABLE 3-31: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
angle	double	360	Circle sector angle.
base	corner center	center	Positions the object either centered about pos or with the lower-left corner of a surrounding box in pos
layer	double[]		Thicknesses of layers
pos	double[]	{0,0}	Position of the object
r	double	1	Radius
rot	double	0	Rotational angle about pos
type	solid curve	solid	Object type
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"circ2")` creates a solid disk.

`model.geom(<tag>).create(<ftag>,"circ1")` creates a circle curve.

The following properties are also available:

TABLE 3-32: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
x, y	double	0	Alias for pos

The property `const` is no longer available.

EXAMPLE

The sequence below creates a unit disk (solid circle object).

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("c1","Circle");
g.feature("c1").set("pos",new double[]{2,3});
String base = g.feature("c1").getString("base");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('c1','Circle');
g.feature('c1').set('pos',[2,3]);
base = g.feature('c1').getString('base');
g.run;
```

SEE ALSO

[Ellipse](#)

CollapseEdges

Collapse edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"CollapseEdges");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"CollapseEdges")` to collapse edges.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the edges to collapse. The default selection is empty.

The feature collapses an edge by removing it, merging its adjacent vertices to the vertex with lowest index, and reconnecting the adjacent edges to the merged vertex.

The output object is a virtual geometry.

The following properties are available:

TABLE 3-33: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Edges to collapse
ignoremerged	on off	on	Specifies if the operation tries to ignore the resulting merged vertices

SEE ALSO

[MergeVertices](#)

CollapseFaces

Collapse faces.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"CollapseFaces");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"CollapseFaces")` to collapse faces.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the faces to collapse. The default selection is empty.

The feature collapses a face by removing it, merging its adjacent opposite edges into one or more edges or collapsing all adjacent edges into one vertex, and reconnecting the adjacent faces to the merged edges or vertex.

The output object is a virtual geometry.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Faces to collapse
ignoremerged	on off	off	Specifies if the operation tries to ignore the resulting merged entities

SEE ALSO

[MergeEdges](#)

Compose, Union, Intersection, Difference

Compose objects using a Boolean set formula.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Compose");
model.geom(<tag>).create(<ftag>,"Union");
model.geom(<tag>).create(<ftag>,"Intersection");
model.geom(<tag>).create(<ftag>,"Difference");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,operationName)` to combine geometric objects in different ways.

Use `model.geom(<tag>).feature(<ftag>).selection(property)` to select the objects to combine. The default selection is empty.

The following properties are available:

TABLE 3-34: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
formula	String		Set formula (only for Compose feature).
input	Selection		Objects to compose.
input2	Selection		Objects to subtract (only for Difference feature).
intbnd	on off	on	Keep interior boundaries.
keep	on off	off	Keep input objects.
repairtol	double	1e-6	Repair tolerance, relative to size of union of inputs.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

The following Boolean operation is performed:

- For **Compose**, the input objects are combined using the set formula in the property formula. The operators +, *, and - correspond to the set operations union, intersection, and difference, respectively. The precedence of the operators + and - are the same. * has higher precedence.
- For **Union**, the objects in **input** are united.
- For **Intersection**, the objects in **input** are intersected.
- For **Difference**, the objects in **input2** are subtracted from the union of the objects in **input** to form a set difference.

For information about the **selresult** and **contributeto** properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The following properties are also supported, see the [Delete](#) feature:

TABLE 3-35: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
edge	all none	none	Delete isolated edges on a face (3D) Delete interior edges and edges not adjacent to a domain (2D, alias for indbnd).
face	all none	none	Delete interior faces (3D, alias for intbnd).
point	all none	none	Delete isolated vertices on a face (3D). Delete isolated vertices in a domain (2D).

The property **out** is no longer available.

SEE ALSO

[ConvertToSolid](#), [ConvertToSurface](#), [ConvertToCurve](#), [ConvertToPoint](#), [Finalize](#), [Partition](#)

CompositeDomains

Form composite domains.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"CompositeDomains");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"CompositeDomains")` to form composite domains.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the domains to composite. The default selection is empty.

The feature forms a composite domain for each connected domain component of the selected domains by ignoring the boundaries between the domains. The output object is a virtual geometry.

The following properties are available:

TABLE 3-36: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
ignoreadj	on off	on	Ignore edges (3D only) and vertices on boundary.
input	Selection		Edges to composite.
keepformesh	on off	off	Keep input domains for mesh control.

Use `ignoreadj` to specify if the feature also removes the ignorable edges (3D only) and vertices on the boundary of each resulting composite domain.

Use `keepformesh` to keep the input domains while meshing, to help you in constructing the mesh.

EXAMPLE

Create a composite domain of domain 2 and 3.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom", 2);
g.create("r1", "Rectangle");
g.create("c1", "Circle");
g.run("fin");
g.create("cmd1", "CompositeDomains");
g.feature("cmd1").selection("input").set("fin", 2, 3);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom', 2);
g.create('r1', 'Rectangle');
g.create('c1', 'Circle');
g.run('fin');
g.create('cmd1', 'CompositeDomains');
g.feature('cmd1').selection('input').set('fin', 2, 3);
g.run;
```

SEE ALSO

[CompositeEdges](#), [CompositeFaces](#), [IgnoreEdges](#), [IgnoreFaces](#)

CompositeEdges

Form composite edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "CompositeEdges");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "CompositeEdges")` to form composite edges.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the edges to concatenate. The default selection is empty.

The feature forms a composite edge for each connected edge component (of manifold type) of the selected edges by ignoring the vertices between the edges. The output object is a virtual geometry.

The following properties are available:

TABLE 3-37: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Edges to composite.
keepformesh	on off	off	Keep input edges for mesh control.

Use `keepformesh` to keep the input edges while meshing, to help you in constructing the mesh.

Note that the operation never forms composite edges that are closed loops or periodic, that is, every resulting edge has distinct start and end vertices.

EXAMPLE

Compose edges 2 and 4 of a circle into one edge.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
g.create("c1", "Circle");
g.run("fin");
g.create("cme1", "CompositeEdges");
g.feature("cme1").selection("input").set("fin", 2, 4);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
g.create('c1', 'Circle');
g.run('fin');
g.create('cme1', 'CompositeEdges');
g.feature('cme1').selection('input').set('fin', 2, 4);
g.run;
```

SEE ALSO

[CompositeDomains](#), [CompositeFaces](#), [IgnoreVertices](#)

CompositeFaces

Form composite faces.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"CompositeFaces");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"CompositeFaces")` to form composite faces.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the faces to concatenate. The default selection is empty.

The feature forms a composite face for each connected face component (of manifold type) of the selected faces by ignoring the edges between the faces. The output object is a virtual geometry.

The following properties are available:

TABLE 3-38: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Faces to composite.
ignorevtx	on off	on	Ignore vertices on boundary.
keepformesh	on off	off	Keep input faces for mesh control.

Use `ignorevtx` to specify if the feature also removes the ignorable vertices on the boundary of each resulting composite face.

Use `keepformesh` to keep the input faces while meshing, to help you in constructing the mesh.

EXAMPLE

A COMSOL Multiphysics standard cone has six faces. Using the following composite face operation, the result is a cone with three faces: top, bottom, and side.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("cone1", "Cone");
g.run("fin");
g.create("cmf1", "CompositeFaces");
g.feature("cmf1").selection("input").set("fin", 1, 2, 5, 6);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('cone1', 'Cone');
g.run('fin');
g.create('cmf1', 'CompositeFaces');
g.feature('cmf1').selection('input').set('fin', [1, 2, 5, 6]);
g.run;
```

SEE ALSO

[CompositeDomains](#), [CompositeEdges](#), [IgnoreEdges](#)

Cone

Create a right circular cone or cone frustum (conical frustum, truncated cone) in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "Cone");
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Cone")` to create a cone. The following properties are available:

TABLE 3-39: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ang	double	$\arctan(1/2)$	Semi-angle, that is, the angle between the axis and a generator of the conical surface.
axis	double[]	{0,0,1}	Direction of the axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
h	double	1	Height.
layer	double[]		Thicknesses of layers.
layertop	on off	off	Apply layers on top.
layerbottom	on off	off	Apply layers on bottom.
layerside	on off	on	Apply layers on side.
pos	double[]	{0,0,0}	Center of the bottom circle.
r	double	1	Radius of bottom circle.
rot	double	0	Rotational angle about axis.

TABLE 3-39: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
rtop	double	0.5	Radius of top circle.
specifytop	angle radius	angle	If axistype is angle, the radius of the top circle is given by the ang property. If axistype is radius, the radius of the top circle is given by the rtop property.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the selresult and contributeto properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"cone3")` creates a solid cone.

`model.geom(<tag>).create(<ftag>,"cone2")` creates a surface cone.

The following properties are also available:

TABLE 3-40: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian
x, y, z	double	0	Alias for pos

The property const is no longer available.

EXAMPLE

Create a cone with an apex:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
double h = 3;
double r = 2;
GeomSequence g = model.geom().create("geom1",3);
g.angularUnit("rad");
g.create("c1","Cone");
g.feature("c1").set("r",r);
g.feature("c1").set("h",h);
g.feature("c1").set("ang", Math.atan(r/h));
double ang = g.feature("c1").getDouble("ang");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
h = 3;
r = 2;
g = model.geom.create('geom1',3);
g.angularUnit('rad');
g.create('c1','Cone');
g.feature('c1').set('r',r);
g.feature('c1').set('h',h);
g.feature('c1').set('ang', atan2(r,h));
ang = g.feature('c1').getDouble('ang');
```

Create a truncated and rotated cone:

Code for use with Java

```
g.create("c2","Cone");
g.feature("c2").set("pos", "1 -2 4");
g.feature("c2").set("axis", "1 -1 0.3");
g.feature("c2").set("rot",Math.PI/3);
g.run();
```

Code for use with MATLAB

```
g.create('c2','Cone');
g.feature('c2').set('pos', '1 -2 4');
g.feature('c2').set('axis', '1 -1 0.3');
g.feature('c2').set('rot', pi/3);
g.run;
```

SEE ALSO

[Cylinder](#), [ECone](#)

ConvertToSolid, ConvertToSurface, ConvertToCurve, ConvertToPoint

Unite and convert objects to a solid, surface, curve, or point object.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ConvertToSolid");
model.geom(<tag>).create(<ftag>,"ConvertToSurface");
model.geom(<tag>).create(<ftag>,"ConvertToCurve");
model.geom(<tag>).create(<ftag>,"ConvertToPoint");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,convertOperation)` to reduce or extend the topological dimension of objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to convert. The default selection is empty.

The following properties are available:

TABLE 3-41: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Objects to convert.
keep	on off	off	Keep input objects.
repairtol	double	1e-6	Repair tolerance, relative to size of union of inputs.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom for ConvertToSolid; bnd for ConvertToSurface; bnd (2D) or edg (3D) for ConvertToCurve; and pnt for ConvertToPoint.	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

The input objects are united, and the resulting object is then converted to the requested type.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The object type `Surface` was called `Face` in version 3.5a.

SEE ALSO

[Compose](#), [Union](#), [Intersection](#), [Difference](#)

CrossSection

Create a 2D geometry from a cross section of a 3D geometry.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"CrossSection");
model.geom(<tag>).feature(<ftag>).selection("input");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

In a 2D geometry, use `model.geom(<tag>).create(<ftag>,"CrossSection")` to create 2D geometry objects by intersecting 3D geometry objects with a work plane. Select the work plane using the `workplane` property, whose value is the 3D sequence's tag followed by a slash and the work plane feature's tag, for example `geom1/wp1`. By default, you get the last work plane in the last 3D geometry.

In a 2D sequence of a work plane feature, use

`model.geom(<tag>).feature(<wptag>).geom().create(<ftag>,"CrossSection")` to create 2D geometry objects by intersecting 3D geometry objects with the work plane.

By default, you get the intersection for all 3D objects that were generated by the features preceding the work plane feature. To select a subset of these objects, set the `intersect` property to `selected`, and use the property `input` to select the 3D objects to intersect.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>selresult</code>	<code>on</code> <code>off</code>	<code>off</code>	Create selections of all resulting objects.
<code>selresultshow</code>	<code>all</code> <code>obj</code> <code>dom</code> <code>bnd</code> <code>pnt</code> <code>off</code>	<code>dom</code>	Show selections of resulting objects in physics, materials, and so on. <code>obj</code> is not available in a component's geometry. <code>dom</code> , <code>bnd</code> , and <code>edg</code> are not available in all features.
<code>contributeto</code>	String	<code>none</code>	Tag of cumulative selection to contribute to.
<code>input</code>	Selection	<code>empty</code>	Selection of objects to intersect.
<code>intersect</code>	<code>all</code> <code>selected</code>	<code>all</code>	Intersect all objects or selected objects.
<code>repairtol</code>	double	<code>1e-6</code>	Repair tolerance, relative to size of each input object.
<code>workplane</code>	String		Work plane to intersect with.

EXAMPLE

Create a work plane though the axis of a torus. In a 2D axisymmetric geometry, create the cross section of the torus using the work plane. Note that the last `run()` command removes the part of the cross section that falls in the region $r < 0$.

Code for use with Java

```
Model model = ModelUtil.create("Model");
```

```

model.modelNode().create("mod1");
GeomSequence g1 = model.geom().create("geom1", 3);
g1.create("tor1", "Torus");
g1.run("tor1");
g1.create("wp1", "WorkPlane");
g1.feature("wp1").set("planetype", "circularedge");
g1.feature("wp1").selection("circedge").set("tor1", 15);

model.modelNode().create("mod2");
GeomSequence g2 =model.geom().create("geom2", 2);
g2.axisymmetric(true);
g2.create("cro1", "CrossSection");
g2.run("cro1"); // Two circles
g2.run(); // One circle

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
model.modelNode.create('mod1');
g1 = model.geom.create('geom1', 3);
g1.create('tor1', 'Torus');
g1.run('tor1');
g1.create('wp1', 'WorkPlane');
g1.feature('wp1').set('planetype', 'circularedge');
g1.feature('wp1').selection('circedge').set('tor1', 15);

model.modelNode.create('mod2');
g2 = model.geom.create('geom2', 2);
g2.axisymmetric(true);
g2.create('cro1', 'CrossSection');
g2.run('cro1'); % Two circles
g2.run(); % One circle

```

SEE ALSO

[WorkPlane](#)

Cylinder

Create a solid or hollow (surface) cylinder in 3D. The cylinder is a right circular cylinder—that is, a cylinder that has circles as bases aligned one directly above the other.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"Cylinder");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Cylinder")` to create a cylinder. The following properties are available:

TABLE 3-42: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
layer	double[]		Thicknesses of layers.
layertop	on off	off	Apply layers on top.

TABLE 3-42: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
layerbottom	on off	off	Apply layers on bottom.
layerside	on off	on	Apply layers on side.
h	double	1	Height.
pos	double[]	{0,0,0}	Center of the bottom circle.
r	double	1	Radius of bottom circle.
rot	double	0	Rotational angle about axis.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the selresult and contributeto properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>, "cylinder3")` creates a solid cylinder.

`model.geom(<tag>).create(<ftag>, "cylinder2")` creates a surface cylinder.

The following properties are also available:

TABLE 3-43: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian
x, y, z	double	0	Alias for pos

The property const is no longer available.

EXAMPLE

The following commands generate a surface cylinder and a solid cylinder:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.angularUnit("rad");
g.create("c2", "Cylinder");
g.feature("c2").set("type", "surface");
g.feature("c2").set("r", 0.5);
g.feature("c2").set("h", 4);
g.feature("c2").set("pos", "1 1 0");
g.feature("c2").set("axis", "pi/2 0");
g.create("c3", "Cylinder");
g.feature("c3").set("r", 20);
g.feature("c3").set("h", 40);
g.feature("c3").set("pos", "0 0 -100");
g.feature("c3").set("axis", "1 1 1");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
```

```

g.angularUnit('rad');
g.create('c2','Cylinder');
g.feature('c2').set('type','surface');
g.feature('c2').set('r',0.5);
g.feature('c2').set('h',4);
g.feature('c2').set('pos','1 1 0');
g.feature('c2').set('axis','pi/2 0');
g.create('c3','Cylinder');
g.feature('c3').set('r',20);
g.feature('c3').set('h',40);
g.feature('c3').set('pos','0 0 -100');
g.feature('c3').set('axis','1 1 1');
g.run;

```

SEE ALSO

[Cone](#), [ECone](#)

Delete

Delete vertices, edges, faces, domains, or geometric objects.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"Delete");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Delete")` to delete geometric entities.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the entities to delete. The default selection is empty.

TABLE 3-44: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Vertices, edges, faces, domains, or objects to delete.
compat	4.2a 4.3	4.3	Algorithm version.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

Deleting a domain, face, or edge automatically deletes all lower-dimensional adjacent entities, except those needed to bound surviving entities.

In 2D and 3D, vertices that are adjacent to an edge cannot be deleted.

In 3D, an edge can be deleted if it has no adjacent faces, or if it is interior to a face.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

In version 4.3, the algorithm was changed slightly. The main difference is that the old algorithm preserved the object type for solid, surface, and curve objects. To get the old behavior, set `compat` to 4.2a.

`model.geom(<tag>).create(<ftag>,"del")` creates a `Delete` feature.

In 3D, the default value for `edge` and `point` was `all` in version 3.5a. Now the default is empty (corresponds to `none` in 3.5a). The value `all` for `subdomain`, `face`, `edge`, and `point` is no longer available. The property `out` is no longer available.

EXAMPLE

Delete face 5 from a surface block:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("blk1","Block");
g.feature("blk1").set("type", "surface");
g.run("blk1");
g.create("del1","Delete");
g.feature("del1").selection("input").set("blk1",5);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('blk1','Block');
g.feature('blk1').set('type', 'surface');
g.run('blk1');
g.create('del1','Delete');
g.feature('del1').selection('input').set('blk1',5);
g.run;
```

SEE ALSO

[Compose](#), [Union](#), [Intersection](#), [Difference](#)

ECone

Create a solid or surface eccentric oblique cone or frustum in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ECone");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"ECone")` to create an eccentric oblique cone. The following properties are available:

TABLE 3-45: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
angle	double	360	Ellipse sector angle.
axis	double[]	{0,0,1}	Direction of the normal to the bottom ellipse. Vector has length 3 if <code>axistype</code> is cartesian, and length 2 if <code>axistype</code> is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with <code>axis</code> .
displ	double[2]	{0,0}	Displacement of top ellipse relative to bottom ellipse in the local coordinate system.
h	double	1	Height.
pos	double[3]	{0,0,0}	Center of the bottom ellipse.
r	double	1	Radius of bottom ellipse.

TABLE 3-45: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
rat	double	0.5	Ratio between perimeter for top ellipse and bottom ellipse.
rot	double	0	Rotational angle about axis.
semiaxes	double[2]	{1,1}	Semi-axes of bottom ellipse.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"econe3")` creates a solid eccentric cone.

`model.geom(<tag>).create(<ftag>,"econe2")` creates a surface eccentric cone.

The following properties are also available:

TABLE 3-46: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
a, b	double	1	Alias for semiaxes.
ax2	double[]	{0,0}	Alias for axis when axistype is spherical.
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian.
x, y, z	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLES

Create a truncated eccentric cone with the base face in the *xy*-plane:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("e1","ECone");
g.feature("e1").set("semiaxes","10 40");
g.feature("e1").set("h",20);
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('e1','ECone');
g.feature('e1').set('semiaxes','10 40');
g.feature('e1').set('h',20);
```

Create an eccentric cone with an apex, that is, a singular patch, on top:

Code for use with Java

```
g.create("e2","ECone");
g.feature("e2").set("semiaxes","1 2");
g.feature("e2").set("h",4);
g.feature("e2").set("rat",0);
g.feature("e2").set("displ","1 1");
```



```
g.feature("e2").set("pos","100 100 100");
g.feature("e2").set("axis","0 1 4");
g.feature("e2").set("rot",45);
g.run();
```

Code for use with MATLAB

```
g.create('e2','ECone');
g.feature('e2').set('semiaxes','1 2');
g.feature('e2').set('h',4);
g.feature('e2').set('rat',0);
g.feature('e2').set('displ','1 1');
g.feature('e2').set('pos','100 100 100');
g.feature('e2').set('axis','0 1 4');
g.feature('e2').set('rot',45);
g.run;
```

SEE ALSO

[Cone](#), [Cylinder](#)

EditObject

Create an edit object feature in 2D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"EditObject");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).vertexNew();
model.geom(<tag>).feature(<ftag>).vertexDelete(<vertex>);
model.geom(<tag>).feature(<ftag>).vertexSnap(<vertex>);
model.geom(<tag>).feature(<ftag>).startVertexDisconnect(<edge>);
model.geom(<tag>).feature(<ftag>).endVertexDisconnect(<edge>);
model.geom(<tag>).feature(<ftag>).edgeNew();
model.geom(<tag>).feature(<ftag>).edgeDelete(<edge>);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"EditObject")` to create an edit object feature.

Use `model.geom(<tag>).feature(<ftag>).vertexNew()` to add a new vertex to the object.

Use `model.geom(<tag>).feature(<ftag>).vertexDelete(<vertex>)` to delete `<vertex>` from the object.

Use `model.geom(<tag>).feature(<ftag>).vertexSnap(<vertex>)` to delete `<vertex>` from the object, and move any adjacent edges to the closest remaining vertex.

Use `model.geom(<tag>).feature(<ftag>).startVertexDisconnect(<edge>)` to create a new vertex and use this vertex as the start vertex for `<edge>`.

Use `model.geom(<tag>).feature(<ftag>).endVertexDisconnect(<edge>)` to create a new vertex and use this vertex as the end vertex for `<edge>`.

Use `model.geom(<tag>).feature(<ftag>).edgeNew()` to add a new edge to the object.

Use `model.geom(<tag>).feature(<ftag>).edgeDelete(<edge>)` to delete `<edge>` from the object.

The following properties are available:

TABLE 3-47: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Geometry object to edit.
vertex	integer ""		Vertex to edit.
xvertex	double	0	x-coordinate of vertex being edited.
yvertex	double	0	y-coordinate of vertex being edited.
edge	integer ""		Edge to edit.
x	double[]		x-coordinates of control points of edge being edited.
y	double[]		y-coordinates of control points of edge being edited.
weights	double[]		Weights of control points of edge being edited.
knots	double[]		Knots of NURBS curve for edge being edited.
degree	1 2 3		Degree of edge being edited.
start	integer ""		Start vertex of edge being edited.
end	integer ""		End vertex of edge being edited.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the selresult and contributeto properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following sequence edits a circle, setting the degree of one edge to one to create a straight edge:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
g.create("c1", "Circle");
g.run("c1");
g.create("edo1", "EditObject");
g.feature("edo1").selection("input").set(new String[]{"c1"});
g.feature("edo1").set("edge", "1");
g.feature("edo1").set("degree", "1");
g.run("edo1");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
g.create('c1', 'Circle');
g.run('c1');
g.create('edo1', 'EditObject');
g.feature('edo1').selection('input').set({'c1'});
g.feature('edo1').set('edge', '1');
g.feature('edo1').set('degree', '1');
g.run('edo1');
```

SEE ALSO

[BezierPolygon](#)

Ellipse

Create a solid or curved ellipse in 2D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Ellipse");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Ellipse")` to create an ellipse. The following properties are available:

TABLE 3-48: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
base	corner center	center	Positions the object either centered about pos or with the lower left corner of surrounding box in pos.
layer	double[]		Thicknesses of layers.
pos	double[]	{0,0}	Position of the object.
rot	double	0	Rotational angle about pos.
semiaxes	double[]	{1,1}	Semi-axes.
type	solid curve	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"ellip2")` is a solid ellipse.

`model.geom(<tag>).create(<ftag>,"ellip1")` is an ellipse curve.

The following properties are also available:

TABLE 3-49: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
a, b	double	1	Alias for semiaxes.
x, y	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLE

The following sequence creates a solid ellipse:

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",2);  
g.create("e1","Ellipse");  
g.feature("e1").set("semiaxes","1 0.3");  
g.feature("e1").set("rot",45);  
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('e1','Ellipse');
g.feature('e1').set('semiaxes','1 0.3');
g.feature('e1').set('rot',45);
g.run;
```

SEE ALSO

[Circle](#)

Ellipsoid

Create a solid or surface ellipsoid in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Ellipsoid");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Ellipsoid")` to create an ellipsoid. The following properties are available:

TABLE 3-50: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the local z-axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
layer	double[]		Thicknesses of layers.
pos	double[]	{0,0,0}	Center.
rot	double	0	Rotational angle about axis.
semiaxes	double[3]	{1,1,1}	Semi-axes.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"ellipsoid3")` creates a solid ellipsoid.

`model.geom(<tag>).create(<ftag>,"ellipsoid2")` creates a surface ellipsoid.

The following properties are also available:

TABLE 3-51: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
a, b, c	double	1	Alias for semiaxes.
ax2	double[]	{0,0}	Alias for axis when axistype is spherical.
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian.
x, y, z	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLE

The following commands create a surface and solid ellipsoid, where the position and semi-axes are defined in two alternative ways:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("e2","Ellipsoid");
g.feature("e2").set("type","surface");
g.feature("e2").set("pos","0 1 0");

g.create("e3","Ellipsoid");
g.feature("e3").set("semiaxes","12 10 8");

g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('e2','Ellipsoid');
g.feature('e2').set('type','surface');
g.feature('e2').set('pos','0 1 0');

g.create('e3','Ellipsoid');
g.feature('e3').set('semiaxes','12 10 8');
g.run;
```

SEE ALSO

[Sphere](#)

ExplicitSelection

Create explicit selections of geometric entities or objects.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ExplicitSelection");
model.geom(<tag>).feature().selection("selection");
model.geom(<tag>).feature().set(property,<value>);
model.geom(<tag>).feature().getType(property);
```

DESCRIPTION

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
angletol	double	5	Angle tolerance for continuity evaluation.
groupcontang	on off	off	Continuous tangent mode.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selection	Selection	empty	Selection of entities or objects.
selkeep	on off	on	Keep the selection within the geometry sequence.
selshow	on off	on	Show selection in physics, materials, and so on, or in part instances. This property is not available in a work plane's Plane Geometry.
contributeto	String	none	Tag of cumulative selection to contribute to.

Use the selection methods described in the section “Geometry Object Selection Methods” under `model.geom()` to specify the selection `model.geom(<tag>).feature().selection("selection")`.

For a boundary or edge selection in 2D or 3D, you can force the selection to select whole groups of entities by setting the property `groupcontang` to `on`. Each group consists of adjacent entities that meet at an angle less than `angletol`.

See [Selections of Geometric Entities](#) for general information about selections.

COMPATIBILITY

The following alias can also be used:

```
model.geom(<tag>).create(<ftag>,"Selection");
```

EXAMPLE

The sequence below creates a block and a cylinder and creates a selection of face 4 of the block. This corresponds to faces 9 and 13 in the finalized geometry.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("blk1", "Block");
g.run("blk1");
g.create("sel1", "ExplicitSelection");
g.feature("sel1").selection("selection").init(2);
g.feature("sel1").selection("selection").set("blk1", new int[]{4});
g.create("cyl1", "Cylinder");
g.run();
int[] faces = model.selection("geom1_sel1").entities(2);
// faces = 9, 13
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('blk1', 'Block');
g.run('blk1');
g.create('sel1', 'ExplicitSelection');
g.feature('sel1').selection('selection').init(2);
g.feature('sel1').selection('selection').set('blk1', 4);
g.create('cyl1', 'Cylinder');
g.run;
faces = model.selection('geom1_sel1').entities(2);
% faces = 9, 13
```

SEE ALSO

[AdjacentSelection](#), [BallSelection](#), [BoxSelection](#), [CylinderSelection](#), [UnionSelection](#), [IntersectionSelection](#), [DifferenceSelection](#), [ComplementSelection](#)

Extrude

Extrude planar faces into 3D objects.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Extrude");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Extrude")` to extrude objects from a work plane or planar faces in the 3D geometry.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the work plane objects to extrude. The default selection is all available objects from the last preceding work plane.

Use `model.geom(<tag>).feature(<ftag>).selection("inputface")` to select the faces to extrude. Faces are extruded when the `workplane` property is none; otherwise work plane objects are extruded.

The following properties are available:

TABLE 3-52: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
crossfaces	on off	on	Keep cross-sectional faces.
displ	double[n_d][2]	{{0,0}}	Displacement (parallel to work plane) of extrusion top (of each layer) in local coordinate system.
distance	double double[n_d]	1	Extrusion distance(s), that is, local z-coordinate for the top (of each layer).
extrudefrom	workplane faces		Extrude work plane objects or planar faces in 3D.
input	Selection	all objects	Objects to extrude.
inputface	Selection		Faces to extrude.
polres	double	50	Polygon resolution of edges.
reverse	on off	off	Reverse the extrude direction.
scale	double[n_d][2]	{{1,1}}	Scale of extrusion top (of each layer).
twist	double double[n_d]	0	Twist angle (of each layer).
unite	on off	on	Unite extruded objects with input objects.
workplane	String		Work plane to extrude.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. <code>obj</code> is not available in a component's geometry. <code>dom</code> , <code>bnd</code> , and <code>edg</code> are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

Each planar input is extruded in n_d layers defined by a local coordinate system. By default, $n_d=1$. The property `distance` is the extrusion distance (of each layer) in the z -axis direction of the local system. The properties `displ`, `scale`, and `twist` define the translation displacements, scale factors and rotation of the top (of each layer) with respect to the bottom of the extruded object. The last array dimension in the properties `displ`, `scale`, and `twist` can be omitted if the same value is desired for all layers.

When extruding work plane objects, the local system is defined as the local system of the work plane. When extruding faces, the local system is defined by the face with the smallest face number in the object that comes first

in the geometry sequence. The local z -axis is parallel to the face normal and located at the center of the face. The local x -axis is defined by the tangent direction corresponding to the first parameter in the surface representation for the face.

When `unite` is set to `on`, the input objects are united with the corresponding extruded objects, after which the input objects are removed. When extruding from work plane object, this has the same effect as deleting the input objects. When `unite` is set to `off`, the extruded objects remain separate from the input objects and the input objects are kept.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The cubic interpolated extrusion is no longer supported.

The following property is also supported:

TABLE 3-53: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
face	all none	all	Cross-sectional faces to delete, alias for <code>crossfaces</code>
keep	on off	off	Alias for <code>unite</code> property with opposite value

EXAMPLE

Creation of a cylinder of height 1.3:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("wp1", "WorkPlane");
g.feature("wp1").geom().create("c1", "Circle");
g.run("wp1");
g.create("e1", "Extrude");
g.feature("e1").set("distance",1.3);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('wp1', 'WorkPlane');
g.feature('wp1').geom.create('c1', 'Circle');
g.run('wp1');
g.create('e1', 'Extrude');
g.feature('e1').set('distance',1.3);
g.run;
```

SEE ALSO

[Revolve](#), [WorkPlane](#)

Fillet

Create circular rounded corners (fillets) in 2D geometry objects. The Design Module supports 3D fillets.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Fillet");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```


DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Fillet")` to round corners in 2D.

Use `model.geom(<tag>).feature(<ftag>).selection("point")` to select which corners to round. The default selection is empty.

TABLE 3-54: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
point	Selection		Vertices to fillet.
radius	double	0	Radius of fillet.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

In version 3.5a, the default was to fillet all vertices in the input objects.

The properties `edges`, `out`, and `radii` are no longer available.

EXAMPLE

Fillet a rectangle object:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("r1","Rectangle");
g.create("fil1","Fillet");
g.feature("fil1").selection("point").set("r1(1)",new int[]{1,2,3,4});
g.feature("fil1").set("radius",0.1);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('r1','Rectangle');
g.create('fil1','Fillet');
g.feature('fil1').selection('point').set('r1(1)',1:4);
g.feature('fil1').set('radius',0.1);
g.run;
```

DIAGNOSTICS

If `Fillet` does not succeed in creating a rounded corner according to the specified radius, the vertex is skipped.

When a fillet intersects another edge, the function generates an error message.

SEE ALSO

[Chamfer](#)

Finalize

Form a union or assembly by combining all geometry objects.

SYNTAX

```
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

The Finalize feature (Form Union/Assembly) combines all available geometry objects in the sequence to form a single geometry object. In 2D and 3D you can modify this object by using virtual operations. The output of the last geometry feature is the *finalized geometry* used when meshing and when setting up physics. If the property action is set to union, and multiple geometry objects are present in the geometry sequence, the objects are combined into a single object with multiple domains corresponding to the input objects and overlaps between these. In a 1D and 2D axisymmetric geometry, the union action also removes the part of the geometry that falls in the region $r < 0$.

Set the property action to `assembly` to keep multiple objects in the finalized geometry. Use this option when modeling physics that needs separate geometry objects, for example, when modeling mechanical contact.

TABLE 3-55: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
action	union assembly	union	Handling of multiple objects,
createpairs	on off	on	Create pairs (used if action=assembly),
imprint	on off	off	Create imprints (used if action=assembly),
repairtol	double	1e-6	Repair tolerance, relative to size of union of inputs,
pairtype	identity contact	identity	Type of pairs to create,
splitpairs	on off	off	Create one pair for each connected set of touching boundaries,

SEE ALSO

[Compose](#), [Union](#), [Intersection](#), [Difference](#)

FromMesh

Create geometry (deformed configuration) from a (deformed) mesh.

SYNTAX

```
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);  
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

To create a geometry sequence from a deformed mesh, use the `createDeformedConfig` method on a solution dataset, see [Solution](#). Such a geometry sequence contains a `FromMesh` feature. This feature has the following properties

TABLE 3-56: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
solnum	non-negative integer	Last solution	The index of the solution to use.

`model.geom(<tag>).feature(<ftag>).importData()` updates the geometry based on the current value of the solution in the feature's corresponding solver sequence.

Helix

Create a solid, surface, or curve helix (coil) with a circular cross section in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Helix");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Helix")` to create a helix. The following properties are available:

TABLE 3-57: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axialpitch	double	0.3	Axial pitch.
axis	double[]	{0,0,1}	Direction of the helix axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
chirality	right left	right	Chirality.
endcaps	paraaxis perpaxis perpspine	paraaxis	Direction of end caps.
grep	bezier spline	spline	Geometry representation.
pos	double[]	{0,0,0}	Position of the object.
radialpitch	double	0	Radial pitch.
rmaj	double	1	Major radius.
rmin	double	0.1	Minor radius.
rot	double	0	Rotational angle about axis.
rtol	double	1e-4	Relative tolerance.
turns	double	3	Number of turns.
type	solid surface	solid	Object type.
twistcomp	on off	on	Twist compensation.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following sequence generates a surface helix and a solid helix:

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",3);  
GeomFeature h = g.create("h1","Helix");  
h.set("type","surface");  
h.set("rmaj",2);  
h.set("rmin",0.3);  
h.set("axialpitch",1);  
  
GeomFeature h2 = g.create("h2","Helix");  
h2.set("rmaj",10);  
h2.set("rmin",2);
```

```

h2.set("axialpitch",1);
h2.set("pos","0,0,-100");
h2.set("axis","1,1,1");
h2.set("rot",60);
g.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
h = g.create('h1','Helix');
h.set('type','surface');
h.set('rmaj',2);
h.set('rmin',0.3);
h.set('axialpitch',1);

h2 = g.create('h2','Helix');
h2.set('rmaj',10);
h2.set('rmin',2);
h2.set('axialpitch',1);
h2.set('pos','0,0,-100');
h2.set('axis','1,1,1');
h2.set('rot',60);
g.run;

```

SEE ALSO

[Torus](#), [Sweep](#)

Hexahedron

Create a solid or surface hexahedron bounded by bilinear faces.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"Hexahedron");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Hexahedron")` to create a general hexahedron. The following properties are available:

TABLE 3-58: VALID PROPERTY/VALUE PAIR

PROPERTY NAME	PROPERTY VALUE	DEFAULT	DESCRIPTION
p	double[3][8]	{0,0,1,1,0,0,1,1}, {0,1,1,0,0,1,1,0}, {0,0,0,0,1,1,1,1}}	Corner coordinates.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For a hexahedron approximately aligned to the coordinate planes, the points in `p` are ordered as follows:

- The first four points and the last four points projected down to the (x, y) -plane defines two negatively oriented quadrangles (quadrilaterals).

- The corresponding plane for the second quadrangle must lie above the plane of the first quadrant in the *z* direction.
- Generally oriented hexahedra have the points of *p* ordered in a similar way, except for a rigid transformation of the defining point set.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following command generates a solid hexahedron object:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("h1", "Hexahedron");
g.feature("h1").set("p", new double[][]
    {{0,0.0,1,1.0,0,0,1.0,1},
     {0,0.8,1,0.0,0,1,1.2,0},
     {0,0.1,0,0.2,1,1,2.0,1}});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('h1', 'Hexahedron');
g.feature('h1').set('p',...
    [[0,0.0,1,1.0,0,0,1.0,1];...
     [0,0.8,1,0.0,0,1,1.2,0];...
     [0,0.1,0,0.2,1,1,2.0,1]]);
g.run;
```

SEE ALSO

[Block](#), [Pyramid](#), [Tetrahedron](#)

If, ElseIf, Else, EndIf

Construct an If statement, enabling or disabling features depending on conditions in terms of parameters.

SYNTAX

```
model.geom(<tag>).create(<ftag>,<type>);
model.geom(<tag>).createAfter(<ftag>,<type>,<postag>);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,<type>)` to add an If, ElseIf, Else, or EndIf feature after the current feature.

Use `model.geom(<tag>).feature().createAfter(<ftag>,<type>,<postag>)` to add an If, ElseIf, Else, or EndIf feature after the feature tagged `<postag>`.

The following property is available for If and ElseIf only:

TABLE 3-59: VALID PROPERTY

NAME	VALUE	DEFAULT	DESCRIPTION
condition	double	1	Logical condition in terms of parameters

EXAMPLE

Build a block if `variant = 1`, else build a cone:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
model.param().set("variant", "1");
GeomSequence g = model.geom().create("geom1", 3);
g.create("if1", "If");
g.feature("if1").set("condition", "variant==1");
g.create("blk1", "Block");
g.create("else1", "Else");
g.create("cone1", "Cone");
g.create("endif1", "EndIf");
g.run();
model.param().set("variant", "2");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
model.param.set('variant', '1');
g = model.geom.create('geom1', 3);
g.create('if1', 'If');
g.feature('if1').set('condition', 'variant==1');
g.create('blk1', 'Block');
g.create('else1', 'Else');
g.create('cone1', 'Cone');
g.create('endif1', 'EndIf');
g.run;
model.param.set('variant', '2');
g.run;
```

IgnoreEdges

Ignore edges by removing selected edges that are isolated, adjacent to precisely two faces, or between two domains.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "IgnoreEdges");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "IgnoreEdges")` to ignore edges.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the edges to ignore. The default selection is empty.

The feature removes the selected edges that are isolated, that are adjacent to precisely two faces, or that are between two domains. If an edge is adjacent to two faces in 3D, the operation forms a composite face, if an edge is between two domains in 2D, the operation forms composite domain. The output object is a virtual geometry.

The following properties are available:

TABLE 3-60: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Edges to ignore
ignorevtx	on off	on	Ignore vertices on boundary
keepformesh	on off	off	Keep edges for mesh control

Use `ignorevtx` to specify if the feature also removes the ignorable vertices on the boundary of each resulting composite face.

Use `keepformesh` to keep the ignored edges while meshing, to help you in constructing the mesh.

EXAMPLE

Create a sphere and ignore all edges and, implicitly, all vertices.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("sph1", "Sphere");
g.create("ige1", "IgnoreEdges");
g.feature("ige1").selection("input").set("fin(1)", 1,2,3,4,5,6,7,8,9,10,11,12);
g.run("ige1");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('sph1', 'Sphere');
g.create('ige1', 'IgnoreEdges');
g.feature('ige1').selection('input').set('fin(1)', 1:12);
g.run('ige1');
```

SEE ALSO

[CompositeFaces](#), [IgnoreFaces](#), [IgnoreVertices](#), [MeshControlEdges](#)

IgnoreFaces

Ignore faces by removing the selected faces that are isolated or that are between two domains.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "IgnoreFaces");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "IgnoreFaces")` to ignore faces in 3D.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the faces to ignore. The default selection is empty.

The feature removes the selected faces that are isolated or that are between two domains. In the latter case, the operation forms a composite domain. The output object is a virtual geometry.

The following properties are available:

TABLE 3-61: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Faces to ignore
ignoreadj	on off	on	Ignore edges and vertices on boundary
keepformesh	on off	off	Keep faces for mesh control

Use `ignoreadj` to specify if the feature also removes the ignorable edges and vertices on the boundary of each resulting composite domain.

Use `keepformesh` to keep the ignored faces while meshing, to help you in constructing the mesh.

EXAMPLE

Ignore faces to form one composite domain. The operation also creates composite faces and composite edges.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("blk1", "Block");
g.create("cyl1", "Cylinder");
g.run("fin");
g.create("igf1", "IgnoreFaces");
g.feature("igf1").selection("input").set("fin", 6, 7, 10);
g.run("igf1");
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('blk1', 'Block');
g.create('cyl1', 'Cylinder');
g.run('fin');
g.create('igf1', 'IgnoreFaces');
g.feature('igf1').selection('input').set('fin', [6, 7, 10]);
g.run('igf1');
```

SEE ALSO

[CompositeDomains](#), [IgnoreEdges](#), [IgnoreVertices](#), [MeshControlFaces](#)

IgnoreVertices

Ignore vertices by removing the selected vertices that are isolated or that are adjacent to precisely two edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "IgnoreVertices");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "IgnoreVertices")` to ignore vertices.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the vertices to ignore. The default selection is empty.

The feature removes the selected vertices that are isolated or that are adjacent to precisely two edges. If a vertex is adjacent to two edges, the operation forms a composite edge. The output object is a virtual geometry.

The following properties are available:

TABLE 3-62: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Vertices to ignore
keepformesh	on off	off	Keep vertices for mesh control

Use `keepformesh` to keep the ignored vertices while meshing, to help you in constructing the mesh.

EXAMPLE

Create an ellipse and ignore vertices 1 and 3, which gives you two remaining edges in the final geometry.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
model.mesh().create("mesh1", "geom1");
g.create("e1", "Ellipse");
```



```

g.run("fin");
g.create("igv1", "IgnoreVertices");
g.feature("igv1").selection("input").set("fin", 1, 3);
g.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
model.mesh.create('mesh1', 'geom1');
g.create('e1', 'Ellipse');
g.run('fin');
g.create('igv1', 'IgnoreVertices');
g.feature('igv1').selection('input').set('fin', 1, 3);
g.run();

```

SEE ALSO

[CompositeEdges](#), [IgnoreEdges](#), [IgnoreFaces](#), [MeshControlVertices](#)

Import DXF

Import geometry objects from a DXF file to a 2D geometry.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"Import");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).getLayers();
model.geom(<tag>).feature(<ftag>).importData();

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Import")` to create a geometry import feature. When the property filename is set to a file recognized as a DXF CAD drawing, the property type is set to `dxf` and the following properties are available:

TABLE 3-63: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
convert	solid curve off	solid	Unite all objects in each layer, and convert it to a solid or curve object.
filename	String		File name.
layers	String[]	all layers	Layers to import.
repairgeom	on off	on	Repair geometry.
repairtol	double	1e-5	Repair tolerance, relative to size of union of imported objects.
type	dxf		Type of import.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
selindividual	on off	off	Create selections of individual objects.
selindividualshow	all dom bnd pnt off	dom	Show selections of individual objects in physics, materials, and so on, or in part instances, when selindividual is on.
contributeto	String	none	Tag of cumulative selection to contribute to.

The file specified by `filename` can be of any of the following formats:

TABLE 3-64: SUPPORTED FILE FORMATS

FILE FORMAT	FILE EXTENSIONS
DXF	.dxf

The imported objects are represented using the COMSOL geometry modeler.

The property `layers` controls which DXF layers are imported. The value is an array of strings containing the names of the layers that are to be imported. The layers found in the file defined by the property `filename` can be retrieved by the command

```
model.geom(<tag>).feature(<ftag>).getLayers()
```

The method

```
model.geom(<tag>).feature(<ftag>).importData()
```

imports the file again.

If `selresult` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point), for use in the geometry sequence. To access the object selection, use `model.geom(<tag>).selection(<ftag>)`, where `<tag>` is the geometry tag and `<ftag>` is the feature tag. To access the other selections, use `model.geom(<tag>).selection(<ftag>.<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt` (`edg` is not available for DXF import in 2D). If, in addition, `selresultshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<ftag>_<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt` (`edg` is not available for DXF import in 2D).

If `selindividual` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point) of each individual object, for use in the geometry sequence. To access the object selections, use `model.geom(<tag>).selection(<otag>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`. To access the other selections, use `model.geom(<tag>).selection(<otag>.<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. If, in addition, `selindividualshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`.

COMPATIBILITY

The following property is also supported:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coercion	<code>solid</code> <code>face</code> <code>curve</code> <code>off</code>	<code>solid</code>	Alias for <code>convert</code> . The value <code>face</code> is equivalent to <code>solid</code> .

Import Geometry Sequence

Import geometry objects from another geometry sequence.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Import");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);  
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Import")` to create a geometry import feature. Set the property mesh to the tag of a meshing sequence of another model component in the model.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sequence	String		Tag of other geometry sequence.
type	sequence		Type of import.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
selindividual	on off	off	Create selections of individual objects.
selindividualshow	all dom bnd edg pnt off	dom	Show selections of individual objects in physics, materials, and so on, or in part instances, when selindividual is on.
contributeto	String	none	Tag of cumulative selection to contribute to.

When building, the import feature takes all the existing objects in the specified sequence and imports them into the feature's sequence.

The method

```
model.geom(<tag>).feature(<ftag>).importData()
```

imports the sequence again. The imported objects are represented using the COMSOL Multiphysics geometry modeler or the CAD Import Module's geometry modeler (Parasolid).

If `selresult` is set to on, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point), for use in the geometry sequence. To access the object selection, use `model.geom(<tag>).selection(<ftag>)`, where `<tag>` is the geometry tag and `<ftag>` is the feature tag. To access the other selections, use `model.geom(<tag>).selection(<ftag>.<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of dom, bnd, edg, or pnt (edg is not available for DXF import in 2D). If, in addition, `selresultshow` is set to a value other than off, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<ftag>_<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of dom, bnd, edg, or pnt (edg is not available for DXF import in 2D).

If `selindividual` is set to on, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point) of each individual object, for use in the geometry sequence. To access the object selections, use `model.geom(<tag>).selection(<otag>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`. To access the other selections, use `model.geom(<tag>).selection(<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. If, in addition, `selindividualshow` is set to a value other than off, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`.

Import Mesh Part or Meshing Sequence

Create a geometry object from an imported mesh.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Import");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);  
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Import")` to create a geometry import feature. Set the property sequence to the tag of another geometry sequence in the model.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
mesh	String	none	Tag of meshing sequence or mesh part to import, or none to create a new mesh part from a mesh file.
meshfilename	String		Path to mesh file to import when mesh is set to none.
type	Mesh		Type of import.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
selindividual	on off	off	Create selections of individual objects.
selindividualshow	all dom bnd edg pnt off	dom	Show selections of individual objects in physics, materials, and so on, or in part instances, when selindividual is on.
contributeto	String	none	Tag of cumulative selection to contribute to.
defectremoval	double	1.0	Relative size factor for identification local defects (3D only).
simplifymesh	on off	on	Boolean specifying if the original mesh should be simplified (3D only).
simplifytol	double	0.01	Relative simplification tolerance (3D only).

When building, the import feature takes the finalized mesh (if the sequence imports a mesh) or the current mesh (if it is not an imported mesh) of the specified sequence and constructs a corresponding geometry object.

The method

```
model.geom(<tag>).feature(<ftag>).importData()
```

imports the sequence again. The imported objects are represented using the COMSOL Multiphysics geometry modeler. The CAD Import Module's geometry modeler (Parasolid) does not support these types of geometries.

If `selresult` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point), for use in the geometry sequence. To access the object selection, use `model.geom(<tag>).selection(<ftag>)`, where `<tag>` is the geometry tag and `<ftag>` is the feature tag. To access the other selections, use `model.geom(<tag>).selection(<ftag>.<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt`. If, in addition, `selresultshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<ftag>_<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt`.

If `selindividual` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point) of each individual object, for use in the geometry sequence. To access the object selections, use `model.geom(<tag>).selection(<otag>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`. To access the other selections, use `model.geom(<tag>).selection(<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. If, in addition, `selindividualshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`.

Import mphbin/mphtxt

Import geometry objects from a file using COMSOL Multiphysics geometry formats: binary or text.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Import");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Import")` to create a geometry import feature. When the property `filename` is set to a file recognized as an `mphbin` or `mphtxt` file, the property `type` is set to `native` and the following properties are available:

TABLE 3-65: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>filename</code>	String		File name.
<code>type</code>	<code>native</code>		Type of import.
<code>selresult</code>	<code>on</code> <code>off</code>	<code>off</code>	Create selections of all resulting objects.
<code>selresultshow</code>	<code>all</code> <code>obj</code> <code>dom</code> <code>bnd</code> <code>edg</code> <code>pnt</code> <code>off</code>	<code>dom</code>	Show selections of resulting objects in physics, materials, and so on, or in part instances. <code>obj</code> is not available in a component's geometry. <code>dom</code> , <code>bnd</code> , and <code>edg</code> are not available in all features.
<code>selindividual</code>	<code>on</code> <code>off</code>	<code>off</code>	Create selections of individual objects.
<code>selindividualshow</code>	<code>all</code> <code>dom</code> <code>bnd</code> <code>edg</code> <code>pnt</code> <code>off</code>	<code>dom</code>	Show selections of individual objects in physics, materials, and so on, or in part instances, when <code>selindividual</code> is <code>on</code> .
<code>contributeto</code>	String	<code>none</code>	Tag of cumulative selection to contribute to.

The file specified by `filename` can be of any of the following formats:

TABLE 3-66: SUPPORTED FILE FORMATS

FILE FORMAT	FILE EXTENSIONS
COMSOL Multiphysics Binary	<code>.mphbin</code>
COMSOL Multiphysics Text	<code>.mphtxt</code>

The imported objects are represented using COMSOL's geometry kernel or the CAD Import Module's geometry kernel (Parasolid).

The method

```
model.geom(<tag>).feature(<ftag>).importData()
```

imports the file again.

If `selresult` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point), for use in the geometry sequence. To access the object selection, use `model.geom(<tag>).selection(<ftag>)`, where `<tag>` is the geometry tag and `<ftag>` is the feature tag. To access the other selections, use `model.geom(<tag>).selection(<ftag>.<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt` (`edg` is not available for DXF import in 2D). If, in addition, `selresultshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<ftag>_<lvl>)`, where `<tag>` is the geometry tag, `<ftag>` is the feature tag, and `<lvl>` is one of `dom`, `bnd`, `edg`, or `pnt` (`edg` is not available for DXF import in 2D).

If `selindividual` is set to `on`, a selection is created for all resulting entities of each type (object, domain, boundary, edge, and point) of each individual object, for use in the geometry sequence. To access the object selections, use `model.geom(<tag>).selection(<otag>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`. To access the other selections, use `model.geom(<tag>).selection(<otag>.<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. If, in addition, `selindividualshow` is set to a value other than `off`, all or some of these selections appear for use outside the geometry sequence. To access these selections, use `model.selection(<tag>_<otag>_<lvl>)`, where `<otag>` is a tag derived from the name of the imported object. For standard object names of the form `<ftag>(<n>)`, where `<n>` is an object number, the corresponding `<otag>` is `<ftag>_<n>`.

Interpolation Curve

Create a curve interpolating or approximating a sequence of points in 2D or 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"InterpolationCurve");
model.geom(<tag>).feature(<ftag>).importToTable();
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

To create an interpolation curve use `model.geom(<tag>).create(<ftag>,"InterpolationCurve")`. The following properties are available:

TABLE 3-67: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
filename	String		If source is file, the file that contains the data.
rtol	double	0	Maximum relative error. 0 implies interpolation.
source	table file vectors	table	Whether data is specified as vectors, a table, or read from a file.
struct	sectionwise spreadsheet	spreadsheet	The data format if source is file.
table	double[][]		Data points, size N*sdim.
type	open closed solid	open	Type of curve.
x	double[]	{}	x-coordinates for data points.
y	double[]	{}	y-coordinates for data points.
z	double[]	{}	z-coordinates for data points.
selresult	on off	off	Create selections of all resulting objects.

TABLE 3-67: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresultshow	all obj dom bnd edg pnt off	dom in 2D; edg in 3D.	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

Use `model.geom(<tag>).feature(<ftag>).importToTable()` to read data from the file defined by the `filename` property and store the data in the `table` property. The source property is also changed to `table`.

If source is file, the interpolation curve is not automatically rebuilt when the data in the file changes. Use `model.geom(<tag>).feature(<ftag>).importData()` to rebuild the interpolation curve after such a change.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following commands create a curve interpolating four points in 2D:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
g.create("ic1", "InterpolationCurve");
g.feature("ic1").set("table", new double[][]{{0,0}, {1,0}, {1,1}, {0,1}});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
g.create('ic1', 'InterpolationCurve');
g.feature('ic1').set('table', [[0,0]; [1,0]; [1,1]; [0,1]]);
g.run;
```

SEE ALSO

[BezierPolygon](#)

Interval

Create one or several connected intervals in 1D.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "Interval");
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Interval")` to create one or more intervals. The following properties are available:

TABLE 3-68: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intervals	one many	one	Use properties p1 and p2, or p.
p	double[]	{0, 1}	Vertex coordinates.
p1	double	0	Left endpoint.
p2	double	1	Right endpoint.

TABLE 3-68: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeTo	String	none	Tag of cumulative selection to contribute to.

To specify one interval, set the properties p1 and p2. Then, intervals is automatically set to one.

To specify a sequence of connected intervals, set the property p. Then, intervals is automatically set to many.

For information about the selresult and contributeTo properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

model.geom(<tag>).create(<ftag>, "solid1") creates an interval.

EXAMPLE

The following commands create a solid consisting of two intervals:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",1);
g.create("i1","Interval");
g.feature("i1").set("p","0 1 3");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',1);
g.create('i1','Interval');
g.feature('i1').set('p','0 1 3');
g.run;
```

SEE ALSO

[BezierPolygon](#)

MergeEdges

Merge edges adjacent to faces.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "MergeEdges");
model.geom(<tag>).feature().selection(property);
model.geom(<tag>).feature().set(property, <value>);
model.geom(<tag>).feature().getType(property);
```

DESCRIPTION

Use model.geom(<tag>).create(<ftag>, "MergeEdges") to merge edges adjacent to face.

Use model.geom(<tag>).feature(<ftag>).selection("keepedge") to select the edges to keep. The default selection is empty.

Use model.geom(<tag>).feature(<ftag>).selection("removeedge") to select the edges to remove. The default selection is empty.

The feature merges the edges by collapsing the face between the edges and reconnecting the faces adjacent to the removed edges to the resulting merged edges.

The output object is a virtual geometry.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
keepedge	Selection		Edges to keep
removeedge	Selection		Edges to remove

SEE ALSO

[CollapseFaces](#)

MergeVertices

Merge two vertices.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"MergeVertices");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"MergeVertices")` to merge two vertices.

Use `model.geom(<tag>).feature(<ftag>).selection("keepvtx")` to select the vertex to keep. The default selection is empty.

Use `model.geom(<tag>).feature(<ftag>).selection("removevtx")` to select the vertex to remove. The default selection is empty.

The feature merges the two vertices by collapsing the edge between the vertices and reconnecting the edges adjacent to the removed vertex to the resulting merged vertex.

The output object is a virtual geometry.

The following properties are available:

TABLE 3-69: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
keepvtx	Selection		Vertex to keep
removevtx	Selection		Vertex to remove

SEE ALSO

[CollapseEdges](#)

MeshControlDomains

Define mesh control domains.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"MeshControlDomains");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"MeshControlDomains")` to define mesh control domains.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the domains to include. The default selection is empty.

The feature creates a composite domain by removing all faces (in 3D) or edges (in 2D) between the selected domains and adjacent domains. The removed entities are kept for mesh control.

The following property is available:

TABLE 3-70: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Edges to ignore

SEE ALSO

[MeshControlFaces](#), [MeshControlEdges](#)

MeshControlEdges

Define mesh control edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"MeshControlEdges");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"MeshControlEdges")` to define mesh control edges.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the edges to include. The default selection is empty.

The feature removes the selected edges that are isolated, that are adjacent to precisely two faces (in 3D), or that are between two domains (in 2D). The edges are kept for mesh control.

The following properties are available:

TABLE 3-71: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Edges to ignore
includevtx	on off	on	Include start and end vertices

Use `includevtx` to specify if the feature also removes the ignorable start and end vertices of the edge.

SEE ALSO

[IgnoreEdges](#), [MeshControlDomains](#), [MeshControlFaces](#), [MeshControlVertices](#)

MeshControlFaces

Define mesh control faces.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"MeshControlFaces");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"MeshControlFaces")` to define mesh control faces in 3D.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the faces to include. The default selection is empty.

The feature removes the selected faces that are isolated or that are between two domains. The faces are kept for mesh control.

The following properties are available:

TABLE 3-72: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Faces to include
includeadj	on off	on	Include edges and vertices on boundary

Use `includeadj` to specify if the feature also includes the ignorable edges and vertices on the boundary of each resulting composite domain.

SEE ALSO

[IgnoreFaces](#), [MeshControlDomains](#), [MeshControlEdges](#), [MeshControlVertices](#)

MeshControlVertices

Define mesh control vertices.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"MeshControlVertices");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"MeshControlVertices")` to define mesh control vertices.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the vertices to include. The default selection is empty.

The feature removes the selected vertices that are isolated or that are adjacent to precisely two edges. The vertices are kept for mesh control.

The following properties are available:

TABLE 3-73: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Vertices to include

SEE ALSO

[IgnoreVertices](#), [MeshControlFaces](#), [MeshControlEdges](#)

Mirror

Reflect (mirror) objects in a plane (3D), a line (2D), or a point (1D).

SYNTAX

```
model.geom(<tag>).create(<ftag>, "Mirror");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Mirror")` to mirror geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to mirror. The default selection is empty.

The following properties are available:

TABLE 3-74: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Objects to reflect
keep	on off	off	Keep input objects
pos	double[]	0	A point to be fixed during reflection.
axis	double[]	{0 0 1} (3D) {1 0} (2D) {1} (1D)	Vector in the direction to reflect.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

In 3D, the input objects are reflected in the plane through `pos` with normal vector `axis`. In 2D, the input objects are reflected in the line through `pos` with normal vector `axis`. In 1D, the input objects are reflected in the point `pos`.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The property `out` is no longer available.

EXAMPLE

A 2D example, mirroring a rectangle:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1", 2);
g.create("r1", "Rectangle");
g.create("m1", "Mirror");
g.feature("m1").selection("input").set("r1");
g.feature("m1").set("pos", "2 2");
```

```
g.feature("m1").set("axis", "1 1");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('r1','Rectangle');
g.create('m1','Mirror');
g.feature('m1').selection('input').set('r1');
g.feature('m1').set('pos','2 2');
g.feature('m1').set('axis','1 1');
g.run;
```

A 3D example, mirroring a block:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("blk1","Block");
g.create("m1","Mirror");
g.feature("m1").selection("input").set("blk1");
g.feature("m1").set("pos","2 2 2");
g.feature("m1").set("axis","1 1 1");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('blk1','Block');
g.create('m1','Mirror');
g.feature('m1').selection('input').set('blk1');
g.feature('m1').set('pos','2 2 2');
g.feature('m1').set('axis','1 1 1');
g.run;
```

SEE ALSO

[Move](#), [Copy](#), [Rotate](#), [Scale](#)

Move, Copy

Move or copy geometry objects by translation.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Move");
model.geom(<tag>).create(<ftag>,"Copy");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Move")` to move geometry objects.

Use `model.geom(<tag>).create(<ftag>,"Copy")` to move a copy of geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to move or copy. The default selection is empty.

The following properties are available:

TABLE 3-75: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
input	Selection		Objects to move or copy.
keep	on off	off (Move) on (Copy)	Keep input objects.
displ	double[] double[][]	0	Displacement vector(s).
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `displ` is a one-dimensional array, a single copy of each input object is created using the translation vector `displ`. If `displ` is a two-dimensional array, several copies can be created, where the *n*th copy has translation `displ[i][n]` in the *i*th coordinate.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The sequence below moves a circle from the origin to (2, 3):

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("c1","Circle");
g.create("m1","Move");
g.feature("m1").selection("input").set("c1");
g.feature("m1").set("displ", new double[][]{{2},{3}});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('c1','Circle');
g.create('m1','Move');
g.feature('m1').selection('input').set('c1');
g.feature('m1').set('displ', [2,3]);
g.run;
```

SEE ALSO

[Array](#), [Mirror](#), [Rotate](#), [Scale](#)

ParameterCheck

Check the values of parameters.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ParameterCheck");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
```

DESCRIPTION

User `model.geom(<tag>).create(<ftag>,"ParameterCheck")` to check parameter values and issue an error if the check condition is fulfilled (being nonzero); for example, the condition `r>30[mm]` results in an error if the value of parameter `r` is larger than 30 mm. The following properties are available:

TABLE 3-76: VALID PROPERTY/VALUE PAIRS FOR PARAMETERCHECK

PROPERTY	VALUE	DEFAULT	DESCRIPTION
condition	double	1	The condition that checks some value of the parameters. The error appears if the value of the condition is nonzero.
message	string	empty string	The error message that is displayed if condition is fulfilled.

ParametricCurve

Create a parametric curve defined by coordinate expressions in 2D or 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ParametricCurve");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"ParametricCurve")` to create a parametric curve. Self-intersecting curves are not supported, except the case of a closed curve (that is, when the start and end points coincide). The following properties are available:

TABLE 3-77: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the z-axis of the local coordinate system. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
coord	String[2] String[3]	empty	Coordinates of parametric curve as function of parameter.
maxknots	int	1000	Maximum number of knots.
parname	String	s	Parameter name.
parmax	double	1	Maximum parameter value.
parmin	double	0	Minimum parameter value.
pos	double[]	{0,0,0}	Position of the object.
rot	double	0	Rotational angle about axis.
rtol	double	1e-6	Relative tolerance.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	bnd in 2D; edg in 3D	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

The expressions in `coord` can contain functions defined in the model. If the definition of such a function is changed, the parametric curve is not automatically rebuilt. Use `model.geom(<tag>).feature(<ftag>).importData()` to rebuild the parametric curve after such a change.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following commands create a parametric curve in 3D with the shape of a helix:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("pc1","ParametricCurve");
g.feature("pc1").set("parmax","2*pi");
g.feature("pc1").set("coord",new String[]{"cos(s)","sin(s)","s*0.2"});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('pc1','ParametricCurve');
g.feature('pc1').set('parmax','2*pi');
g.feature('pc1').set('coord',{'cos(s)','sin(s)','s*0.2'});
g.run;
```

SEE ALSO

[BezierPolygon](#), [ParametricSurface](#)

ParametricSurface

Create a parametric surface defined by coordinate expressions in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"ParametricSurface");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"ParametricSurface")` to create a parametric surface.

Self-intersecting surfaces are not supported. The following properties are available:

TABLE 3-78: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the z-axis of the local coordinate system. Vector has length 3 if <code>axistype</code> is cartesian, and length 2 if <code>axistype</code> is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with <code>axis</code> .
coord	String[3]	empty	Coordinates of parametric surface as function of parameters.
maxknots	int	10	Maximum number of knots in each parameter coordinate.
parname1	String	s1	First parameter name.
parname2	String	s2	Second parameter name.

TABLE 3-78: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
parmax1	double	1	Maximum value of first parameter.
parmax2	double	1	Maximum value of second parameter.
parmin1	double	0	Minimum value of first parameter.
parmin2	double	0	Minimum value of second parameter.
pos	double[]	{0,0,0}	Position of the object.
rot	double	0	Rotational angle about axis.
rto1	double	1e-6	Relative tolerance.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	bnd	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

The expressions in `coord` can contain functions defined in the model. If the definition of such a function is changed, the parametric surface is not automatically rebuilt. Use `model.geom(<tag>).feature(<ftag>).importData()` to rebuild the parametric surface after such a change.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The following commands create a parametric surface in 3D with the shape of a twisted rectangle:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("ps1", "ParametricSurface");
g.feature("ps1").set("parmin1", "-1");
g.feature("ps1").set("parmax2", "pi");
g.feature("ps1").set("coord", new String[] {"s1*cos(s2)", "s1*sin(s2)", "s2"});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('ps1', 'ParametricSurface');
g.feature('ps1').set('parmin1', '-1');
g.feature('ps1').set('parmax2', 'pi');
g.feature('ps1').set('coord', {'s1*cos(s2)', 's1*sin(s2)', 's2'});
g.run;
```

SEE ALSO

[ParametricCurve](#)

PartInstance

Create an instance of a geometry part.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "PartInstance");
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).geom().geomSequenceMethod;
```

DESCRIPTION

This feature creates an instance of a geometry part with new values of its input parameters. Use `model.geom(<tag>).feature(<ftag>).geom()` to access its local copy of the part.

The following properties are available:

TABLE 3-79: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
part	String		Tag of part to call, or local to use a local part.
inputname	String[]		Names of the input parameters (read-only).
inputexpr	String[]		Expressions for the input parameters.

The default for part is the first part in `model.geom()`, or local if there is none.

In 3D, the following additional properties are available to control the positioning of the output objects:

TABLE 3-80: VALID PROPERTY/VALUE PAIRS IN 3D

PROPERTY	VALUES	DEFAULT	DESCRIPTION
workplanepart	String	xyplane	Tag of work plane in the geometry part, or xyplane.
workplanesrc	String	this	Tag of PartInstance feature to take work plane from, or this to take it from this sequence.
workplane	String	xyplane	Tag of work plane to match, or xyplane.
displ	double[3]	{0,0,0}	Displacement vector.
axistype	x y z Cartesian spherical	z	Type of rotation axis specification.
axis	double[]	{0,0,1}	Direction of rotation axis. Vector has length 3 if axistype is Cartesian and length 2 if axistype is spherical.
rot	double	0	Rotation angle.

In 2D, the following additional properties are available to control the positioning of the output objects:

TABLE 3-81: VALID PROPERTY/VALUE PAIRS IN 2D

PROPERTY	VALUES	DEFAULT	DESCRIPTION
displ	double[2]	{0,0}	Displacement vector
rot	double	0	Rotation angle

For each geometric entity level (object, domain, boundary, edge, and point) that exists in the geometry, there is in addition the following properties related to the output selections for that level:

TABLE 3-82: VALID PROPERTY/VALUE PAIRS FOR OUTPUT SELECTIONS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
selkeepnoncontr	on off	on	Keep all noncontributing selections.
seltaglevel	String[]		Tags of selections (read-only)
selnamelevel	String[]		Names of selections (read-only)
selcontributelevel	String[]	all none	Tags of cumulative selections to contribute to, or none to not contribute.
selkeeplevel	String[] with on/off values	all off	Keep selection from part (only used when selkeepnoncontr is off).
selshowlevel	String[] with on/off values	all on	Show selection in physics, materials, and so on, and in part instances. This property is not available in a work plane's Plane Geometry.

where *level* is obj, dom, bnd, edg, or pnt for geometry objects, domains, boundaries, edges, and points, respectively.

COMPATIBILITY

In version 5.1, the following properties are deprecated and replaced:

TABLE 3-83: DEPRECATED PROPERTIES

PREVIOUS NAME	NEW NAME IN 5.1
arg	inputname
argexpr	inputexpr
argvalue	inputvalue
argdescr	inputdescr
subsequence	part
workplanesub	workplanepart

EXAMPLE

Create a geometry part that makes a torus of revolution angle α , where α is an argument (default value: 90 degrees). Then add work planes for the two planar face to make it easy to position the result:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence p = model.geom().create("part1", "Part", 3);
p.inputParam().set("a", 90);
p.create("tor1", "Torus");
p.feature("tor1").set("angle", "a");
p.run("tor1");
p.create("wp1", "WorkPlane");
p.feature("wp1").set("planetype", "faceparallel");
p.feature("wp1").selection("face").set("tor1", new int[]{1});
p.feature("wp1").set("reverse", "on");
p.create("wp2", "WorkPlane");
p.feature("wp2").set("planetype", "faceparallel");
p.feature("wp2").selection("face").set("tor1", new int[]{6});
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
p = model.geom.create('part1', 'Part', 3);
p.inputParam.set('a', 90);
p.create('tor1', 'Torus');
p.feature('tor1').set('angle', 'a');
p.run('tor1');
p.create('wp1', 'WorkPlane');
p.feature('wp1').set('planetype', 'faceparallel');
p.feature('wp1').selection('face').set('tor1', 1);
p.feature('wp1').set('reverse', 'on');
p.create('wp2', 'WorkPlane');
p.feature('wp2').set('planetype', 'faceparallel');
p.feature('wp2').selection('face').set('tor1', 6);
```

Create two part instances of this geometry part. The first has $\alpha = 90$ (the default value), and the second has $\alpha = 120$. The objects are positioned so that the two circular faces match, with a rotation angle of 50 degrees.

Code for use with Java

```
GeomSequence g = model.geom().create("geom1", 3);
g.create("pi1", "PartInstance");
g.create("pi2", "PartInstance");
g.feature("pi2").setIndex("inputexpr", 120, 0);
g.feature("pi2").set("workplanepart", "wp1");
g.feature("pi2").set("workplanesrc", "pi1");
```

```
g.feature("pi2").set("workplane", "wp2");
g.feature("pi2").set("rot", 50);
g.run("pi2");
```

Code for use with MATLAB

```
g = model.geom.create('geom1', 3);
g.create('pi1', 'PartInstance');
g.create('pi2', 'PartInstance');
g.feature('pi2').setIndex('inputexpr', 120, 0);
g.feature('pi2').set('workplanepart', 'wp1');
g.feature('pi2').set('workplanesrc', 'pi1');
g.feature('pi2').set('workplane', 'wp2');
g.feature('pi2').set('rot', 50);
g.run('pi2');
```

SEE ALSO

[If](#), [ElseIf](#), [Else](#), [EndIf](#)

Partition

Partition geometry objects using tool objects or a work plane.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "Partition");
model.geom(<tag>).feature().selection(property);
model.geom(<tag>).feature().set(property, <value>);
model.geom(<tag>).feature().getType(property);
```

DESCRIPTION

The Partition Boolean operation partitions each input object using a set of tool objects or a work plane. That is, within the input object new boundaries, edges, and vertices are created that come from the tools or the work plane.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.
input	Selection	empty	Objects to partition.
keep	on off	off	Keep input objects.
partitionwith	objects workplane	objects	Partition with tool objects or a work plane.
repairtol	double	1e-6	Repair tolerance, relative to size of input objects.
tool	Selection	empty	Tool objects.
workplane	String		Work plane to partition with.

For information about the selresult and contributeto properties, see [Selections of Geometric Entities](#).

EXAMPLE

Create an interior boundary in a cylinder by partitioning it with an oblique work plane:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
```

```

g.create("cyl1", "Cylinder");
g.feature("cyl1").set("h", 10);
g.create("wp1", "WorkPlane");
g.feature("wp1").set("planetype", "general");
g.feature("wp1").setIndex("genpoints", 4, 0, 2);
g.feature("wp1").setIndex("genpoints", 5, 1, 2);
g.feature("wp1").setIndex("genpoints", 5, 2, 2);
g.create("par1", "Partition");
g.feature("par1").selection("input").set("cyl1");
g.feature("par1").set("partitionwith", "workplane");
g.run("par1");

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom', 3);
g.create('cyl1', 'Cylinder');
g.feature('cyl1').set('h', 10);
g.create('wp1', 'WorkPlane');
g.feature('wp1').set('planetype', 'general');
g.feature('wp1').setIndex('genpoints', 4, 0, 2);
g.feature('wp1').setIndex('genpoints', 5, 1, 2);
g.feature('wp1').setIndex('genpoints', 5, 2, 2);
g.create('par1', 'Partition');
g.feature('par1').selection('input').set('cyl1');
g.feature('par1').set('partitionwith', 'workplane');
g.run('par1');

```

SEE ALSO

[Compose](#), [Union](#), [Intersection](#), [Difference](#), [WorkPlane](#), [PartitionDomains](#), [PartitionEdges](#)

PartitionDomains

Partition domains in 2D or 3D geometries with curves and surfaces defined in various ways.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"PartitionDomains");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature().set(property,<value>);
model.geom(<tag>).feature().getType(property);

```

DESCRIPTION

The Partition Domains operation partitions selected domains using curves or surfaces defined by vertices, edges, faces, work planes, or objects.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.
domain	Selection		Selection of domains to partition.
edge	Selection		Edges that define the partitioning curves (2D).
extendededge	Selection		Edges whose underlying curves define the partitioning curves (2D).

PROPERTY	VALUE	DEFAULT	DESCRIPTION
extendedface	Selection		Faces whose underlying surfaces define the partitioning surfaces (3D).
face	Selection		Faces that define the partitioning surfaces (3D).
object	Selection		Geometry objects used to partition domains.
partitionwith	linesegments lines edges extendededges objects in 2D workplane faces extendedfaces objects in 3D	linesegments in 2D, workplane in 3D	Method for partitioning the domains.
repairtol	double	1e-6	Repair tolerance, relative to size of input domains.
vertexsegment	Selection		Vertices that define the line segments (2D).
vertexline	Selection		Vertices that define the lines (2D).
workplane	String		Work plane to partition with (3D).

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

SEE ALSO

[Partition](#), [PartitionEdges](#)

PartitionEdges

Partition edges in 2D or 3D geometries at some positions along the edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>, "PartitionEdges");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature().set(property, <value>);
model.geom(<tag>).feature().getType(property);
```

DESCRIPTION

The Partition Edges operation partitions selected edges at specified locations. You can specify the positions using parameters based on the arc length or existing vertices whose orthogonal projections on the edges specify the positions.

The following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. <code>obj</code> is not available in a component's geometry. <code>dom</code> , <code>bnd</code> , and <code>edg</code> are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.
edge	Selection		Selection of edges to partition.
position	arclength projection	arclength	Specifies the position along the selected edges.
param	double[]		Relative arc length parameters.
vertexproj	Selection		Vertices to project on the selected edges.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

SEE ALSO

[Partition](#), [PartitionDomains](#)

Point

Create a point object in 1D, 2D, or 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Point");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Point")` to create one or more points. The following property is available:

TABLE 3-84: VALID PROPERTY/VALUE PAIR

PROPERTY NAME	PROPERTY VALUE	DEFAULT	DESCRIPTION
p	double[] double[][]	0	Coordinates.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj bnd pnt off	pnt in 2D and 3D; bnd in 1D	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `p` is a one-dimensional array, a single point with these coordinates is constructed. If `p` is a two-dimensional array, a point object containing several points is constructed, where the *n*th point has *i*th coordinate `p[i][n]`.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The following aliases work in 1D, 2D, and 3D, respectively:

```
model.geom(<tag>).create(<ftag>,"point1");
model.geom(<tag>).create(<ftag>,"point2");
model.geom(<tag>).create(<ftag>,"point3");
```

EXAMPLE

The following commands generate a point at (1, 2) in a 2D geometry:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("p1","Point");
g.feature("p1").set("p",new double[][]{{1},{2}});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('p1','Point');
g.feature('p1').set('p',[1,2]);
g.run;
```

Polygon

Create curve or solid polygon consisting of line segments in 2D or 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Polygon");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Polygon")` to create a polygon or a line segment. The following properties are available

TABLE 3-85: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
filename	String		If source is file, the file that contains the vertex coordinates.
source	table file vectors	vectors	Whether vertex coordinates are specified as vectors, a table, or read from a file.
table	double[][]		The vertex coordinates when source is table, size N*sdim.
type	solid open closed	solid (2D) open (3D)	Object type. solid is not available in 3D.
x	double[]	{}	x-coordinates for vertices.
y	double[]	{}	y-coordinates for vertices.
z	double[]	{}	z-coordinates for vertices.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom in 2D; edg in 3D.	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `type` is `open` or `closed`, a curve consisting of line segments is constructed. If `type` is `solid`, the solid enclosed by such a closed polygon is constructed. If `type` is `closed` or `solid`, but the first and last control points are different, an extra segment is added to close the curve.

Use `model.geom(<tag>).feature(<ftag>).importToTable()` to read data from the file defined by the `filename` property and store the data in the `table` property. The `source` property is also changed to `table`.

If `source` is `file`, the polygon is not automatically rebuilt when the data in the file changes. Use `model.geom(<tag>).feature(<ftag>).importData()` to rebuild the polygon after such a change.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"line1")` constructs an open polygon.

`model.geom(<tag>).create(<ftag>,"poly1")` constructs a closed polygon.

`model.geom(<tag>).create(<ftag>,"line2")` or `model.geom(<tag>).create(<ftag>,"poly2")` constructs a solid polygon.

EXAMPLE

Construct a solid triangle pol1:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("pol1","Polygon");
g.feature("pol1").set("x","0,0,2").set("y","1,0,0");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('pol1','Polygon');
g.feature('pol1').set('x','0,0,2').set('y','1,0,0');
g.run;
```

SEE ALSO

[BezierPolygon](#)

Pyramid

Create solid or surface rectangular pyramid or frustum in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Pyramid");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Pyramid")` to create a pyramid. The following properties are available:

TABLE 3-86: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
a, b	double	1	Side lengths for bottom rectangle.
axis	double[]	{0,0,1}	Direction of the axis orthogonal to the bottom rectangle. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
h	double	1	Height.
pos	double[]	{0,0,0}	Center of the bottom rectangle.
rat	double	0.5	Ratio of perimeter of top rectangle and bottom rectangle.
rot	double	0	Rotational angle about axis.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>, "pyramid3")` creates a solid pyramid.

`model.geom(<tag>).create(<ftag>, "pyramid2")` creates a surface pyramid.

The following properties are also available:

TABLE 3-87: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian
x, y, z	double	0	Alias for pos

The property `const` is no longer available.

EXAMPLE

Create a pyramid frustum with the base face in the *xy*-plane:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("e1", "Pyramid");
g.feature("e1").set("a",10).set("b",40);
g.feature("e1").set("h",20);
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('e1', 'Pyramid');
g.feature('e1').set('a',10).set('b',40);
g.feature('e1').set('h',20);
```

Create a pyramid with an apex:

Code for use with Java

```
g.create("e2", "Pyramid");
g.feature("e2").set("a",1).set("b",2);
g.feature("e2").set("h",4);
g.feature("e2").set("rat",0);
g.feature("e2").set("pos", "100 100 100");
g.feature("e2").set("axis", "0 1 4");
g.feature("e2").set("rot",45);
g.run();
```

Code for use with MATLAB

```
g.create('e2', 'Pyramid');
g.feature('e2').set('a',1).set('b',2);
g.feature('e2').set('h',4);
g.feature('e2').set('rat',0);
g.feature('e2').set('pos', '100 100 100');
g.feature('e2').set('axis', '0 1 4');
g.feature('e2').set('rot',45);
g.run;
```

SEE ALSO

[Cone](#), [ECone](#)

Rectangle

Create a solid or curve rectangle in 2D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Rectangle");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Rectangle")` to create a rectangle. The following properties are available:

TABLE 3-88: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
base	corner center	corner	Positions the object either centered about pos or with the lower-left corner in pos.
layer	double[]		Thicknesses of layers.
layerleft	on off	off	Apply layers to the left.
layerright	on off	off	Apply layers to the right.
layertop	on off	off	Apply layers on top.
layerbottom	on off	on	Apply layers on bottom.
pos	double[]	{0,0}	Position of the object.
rot	double	0	Rotational angle about pos.
size	double[]	{1,1}	Side lengths.
type	solid curve	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"rect2")` creates a solid rectangle.

`model.geom(<tag>).create(<ftag>,"rect1")` creates a curve rectangle.

The following properties are also available:

TABLE 3-89: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
lx, ly	double	1	Alias for size
x, y	double	0	Alias for pos

The property `const` is no longer available.

Revolve

Revolve planar faces in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Revolve");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Revolve")` to revolve objects from a work plane.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the work plane objects to revolve. The default selection is all available objects from the last preceding work plane.

Use `model.geom(<tag>).feature(<ftag>).selection("inputface")` to select the faces to revolve. Faces are revolved when the `workplane` property is none, otherwise work plane objects are revolved.

The following properties are available:

TABLE 3-90: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
angle1	double	0	Start revolution angle.
angle2	double	0	End revolution angle.
angtype	specang full	specang	Type of specification.
axis	double[2]	{0,1}	Direction of axis of revolution (in local coordinate system).
axis3	double[3]	{0,1,0}	Direction of axis of revolution (in 3D coordinate system).
axistype	2d 3d	2d	Type of revolution axis.
input	Selection	all objects	Objects to revolve.
inputface	Selection		Faces to revolve.
origfaces	on off	on	Keep original faces.
polres	double	50	Polygon resolution of edges.
pos	double[2]	{0,0}	A point on the axis of revolution (in work plane's coordinate system).
pos3	double[3]	{0,0,0}	A point on the axis of revolution (in 3D coordinate system).
revolvefrom	workplane faces		Revolve work plane objects or faces from 3D objects.
unite	on off	on	Unite revolved objects with input objects.
workplane	String		Work plane to revolve or none to revolve faces.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

Each 2D object in `input` or planar face in `inputface` is revolved about the revolution axis. The range of angles is given by the properties `angle1` and `angle2`. If `axistype` is 2d, the revolution axis is defined in a local coordinate system. The revolution axis goes through `pos` with direction `axis`. If `axistype` is 3d, the revolution axis is defined in the 3D coordinate system. The revolution axis goes through `pos3` with direction `axis3`.

When revolving work plane objects, the local system is defined as the local system of the work plane. When revolving faces, the local system is defined by the face with the smallest face number in the object that comes first

in the geometry sequence. The local z -axis is parallel to the face normal and located at the center of the face. The local x -axis is defined by the tangent direction corresponding to the first parameter in the surface representation for the face.

When `unite` is on, the input objects are united with the corresponding revolved objects, after which the input objects are removed. When revolving from work plane objects, this has the same effect as deleting the input objects. When `unite` is off, the revolved objects remain separate from the input objects and the input objects are kept.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

Additional properties:

TABLE 3-91: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
angles	double double[2]	2*pi	Alias for angle1 and angle2
keep	on off	off	Alias for unite property with opposite value
revaxis	double[2][2]	{0,0}, {0,1}}	Alias for pos (first column) and axis (second column)

EXAMPLE

Create torus about the y -axis:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("wp1","WorkPlane");
g.feature("wp1").geom().create("c1", "Circle");
g.feature("wp1").geom().feature("c1").set("pos", "2 0");
g.run("wp1");
g.create("r1","Revolve");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('wp1','WorkPlane');
g.feature('wp1').geom.create('c1', 'Circle');
g.feature('wp1').geom.feature('c1').set('pos', '2 0');
g.run('wp1');
g.create('r1','Revolve');
g.run;
```

SEE ALSO

[Extrude](#), [WorkPlane](#)

Rotate

Rotate objects about a point in 2D or an axis in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Rotate");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Rotate")` to rotate geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to rotate. The default selection is empty.

The following properties are available:

TABLE 3-92: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Rotation axis in 3D. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
input	Selection		Objects to rotate.
keep	on off	off	Keep input objects.
pos	double[]		Center of rotation.
rot	double[]	0	Rotation angles for one or more rotations of the input objects.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

The possibility to set and get a rotation matrix has been removed.

The property `out` is no longer available.

EXAMPLE

The commands below create and then rotate an ellipse by 10 degrees about (2, 3):

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("e1", "Ellipse");
g.feature("e1").set("semiaxes", "1 3");
g.create("r1", "Rotate");
g.feature("r1").selection("input").set("e1");
g.feature("r1").set("rot", 10);
g.feature("r1").set("pos", "2 3");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('e1', 'Ellipse');
g.feature('e1').set('semiaxes', '1 3');
g.create('r1', 'Rotate');
g.feature('r1').selection('input').set('e1');
g.feature('r1').set('rot', 10);
g.feature('r1').set('pos', '2 3');
```

```
g.run;
```

SEE ALSO

[Mirror](#), [Move](#), [Copy](#), [Scale](#)

Scale

Scale objects around a point.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Scale");  
model.geom(<tag>).feature(<ftag>).selection(property);  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Scale")` to scale geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects scale. The default selection is empty.

The following properties are available:

TABLE 3-93: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
factor	double double[]	1	Scale factor(s).
input	Selection		Objects to scale.
keep	on off	off	Keep input objects.
pos	double[]	0	Center of scaling.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `factor` is an array, the inputs are scaled by the `factor[i]` in the `i`th coordinate.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

The sequence below scales the unit circle by (1, 2) about (2, 3):

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",2);  
g.create("c1","Circle");  
g.create("s1","Scale");  
g.feature("s1").selection("input").set("c1");  
g.feature("s1").set("factor", "1,2");  
g.feature("s1").set("pos",new double[]{2,3});  
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');  
g = model.geom.create('geom1',2);  
g.create('c1','Circle');
```

```

g.create('s1', 'Scale');
g.feature('s1').selection('input').set('c1');
g.feature('s1').set('factor', '1,2');
g.feature('s1').set('pos', [2,3]);
g.run;

```

COMPATIBILITY

The property out is no longer available.

SEE ALSO

[Mirror](#), [Move](#), [Copy](#), [Rotate](#)

Sphere

Create a solid ball or surface sphere in 3D.

SYNTAX

```

model.geom(<tag>).create(<ftag>, "Sphere");
model.geom(<tag>).feature(<ftag>).set(property, <value>);
model.geom(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>, "Sphere")` to create a sphere. The following properties are available:

TABLE 3-94: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
axis	double[]	{0,0,1}	Direction of the local z-axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
layer	double[]		Thicknesses of layers.
pos	double[]	{0,0,0}	Center.
r	double	1	Radius.
rot	double	0	Rotational angle about axis.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>, "sphere3")` creates a solid sphere.

`model.geom(<tag>).create(<ftag>, "sphere2")` creates a surface sphere.

The following properties are also available:

TABLE 3-95: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical.
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian.
x, y, z	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLE

The following commands create a surface and solid sphere, where the position and radius are defined differently:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("s2","Sphere");
g.feature("s2").set("type","surface");
g.feature("s2").set("pos","0 1 0");
g.create("s3","Sphere");
g.feature("s3").set("r",4);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('s2','Sphere');
g.feature('s2').set('type','surface');
g.feature('s2').set('pos','0 1 0');
g.create('s3','Sphere');
g.feature('s3').set('r',4);
g.run;
```

SEE ALSO

[Ellipsoid](#)

Split

Split (explode) objects into domains, faces, edges, or vertices.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Split");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Split")` to split geometry objects.

Use `model.geom(<tag>).feature(<ftag>).selection("input")` to select the objects to split. The default selection is empty.

TABLE 3-96: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
input	Selection		Objects to split.
keep	on off	off	Keep input objects.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

- A solid object is split into solids corresponding to its domains.
- A surface object is split into surface objects corresponding to its faces.
- A curve object is split into curve objects corresponding to its edges.
- A point object is split into point objects corresponding to its vertices.
- A general (mixed) object is split into solids (corresponding to the domains), surface objects (corresponding to faces not adjacent to a domain), curve objects (corresponding to edges not adjacent to a face or domain), and point objects (corresponding to vertices not adjacent to an edge, face, or domain).

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

EXAMPLE

Split union of a solid circle and a solid rectangle.

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("r1","Rectangle");
g.create("c1","Circle");
g.create("u1","Union");
g.feature("u1").selection("input").set(new String[]{"r1","c1"});
g.create("spl1","Split");
g.feature("spl1").selection("input").set("u1");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('r1','Rectangle');
g.create('c1','Circle');
g.create('u1','Union');
g.feature('u1').selection('input').set({'r1','c1'});
g.create('spl1','Split');
g.feature('spl1').selection('input').set('u1');
g.run;
```

SEE ALSO

[Compose](#), [Union](#), [Intersection](#), [Difference](#), [Delete](#)

Square

Create a solid or curve square in 2D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Square");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Square")` to create a square. The following properties are available:

TABLE 3-97: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
base	corner center	corner	Positions the object either centered about pos or with the lower left corner in pos.
layer	double[]		Thicknesses of layers.
layerleft	on off	off	Apply layers to the left.
layerright	on off	off	Apply layers to the right.
layertop	on off	off	Apply layers on top.
layerbottom	on off	on	Apply layers on bottom.
pos	double[]	{0,0}	Position of the object.
rot	double	0	Rotational angle about pos.
size	double	1	Side length.
type	solid curve	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"square2")` creates a solid square.

`model.geom(<tag>).create(<ftag>,"square1")` creates a curve square.

The following properties are also available:

TABLE 3-98: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
l	double	1	Alias for size
x, y	double	0	Alias for pos

The property `const` is no longer available.

EXAMPLE

The sequence below creates a unit solid square:

Code for use with Java

```
Model model = ModelUtil.create("Model1");  
GeomSequence g = model.geom().create("geom1",2);  
g.create("sq1","Square");  
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom',2);
g.create('sq1','Square');
g.run;
```

SEE ALSO

[Rectangle](#)

Sweep

Sweep one or several faces along a spine curve into a solid in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Sweep");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Sweep")` to sweep faces along a spine curve.

Use `model.geom(<tag>).feature(<ftag>).selection("face")` to select the faces to sweep.

Use `model.geom(<tag>).feature(<ftag>).selection("edge")` to select the edges to sweep along.

Use `model.geom(<tag>).feature(<ftag>).selection("diredge")` to select the edge whose direction defines the positive sweep direction. If this selection is empty, it is automatically set when the `edge` selection is set. The `diredge` selection can be empty if the `edge` selection contains a single edge.

The following properties are available:

TABLE 3-99: VALID PROPERTY/VALUE PAIRS FOR SWEEP

PROPERTY	VALUES	DEFAULT	DESCRIPTION
adjustlen	double	0	Spine adjustment parameter length. Used when <code>align</code> is set to <code>adjustspine</code> .
align	noadjust adjustspine moveface	noadjust	Type of alignment between face and spine curve.
diredge	Selection		Direction-defining edge.
edge	Selection		Edges that form spine curve.
face	Selection		Faces to sweep.
grep	bezier spline	spline	Geometry representation.
includefinal	on off	off	Include all used input objects in Finalize operation.
keep	on off	on	Keep input objects.
maxknots	int	1000	Maximum number of knots.
parname	String	s	Parameter name.
reversedir	on off	off	Reverse sweep direction.
rtol	double	1e-4	Relative tolerance.
scale	String	1	Scale factor for cross section.
twist	String	0	Twist angle for cross section.
twistcomp	on off	on	Twist compensation.
selresult	on off	off	Create selections of all resulting objects.

TABLE 3-99: VALID PROPERTY/VALUE PAIRS FOR SWEEP

PROPERTY	VALUES	DEFAULT	DESCRIPTION
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

The expressions in `scale` and `twist` can contain functions defined in the model. If the definition of such a function is changed, the swept object is not automatically rebuilt. Use `model.geom(<tag>).feature(<ftag>).importData()` to rebuild the swept object after such a change.

If `includefinal` is `off`, input objects are automatically removed in the Finalize (Form Union/Assembly) operation if they are completely used by this feature. Objects used in `face` are considered completely used if they contain a single face. Objects used in `edge` are considered completely used if they contain no faces and all their edges are included in `edge`. If an object is considered completely used by one property but not completely used by another property, the object is not removed in the Finalize operation. If `includefinal` is `on`, input objects are not removed in the Finalize operation.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

In COMSOL Multiphysics 4.2a and earlier versions, the positive sweep direction was defined as the curve direction instead of the edge direction.

EXAMPLE

Create a half torus about the *y*-axis using a sweep operation:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("wp1","WorkPlane");
g.feature("wp1").geom().create("c1","Circle");
g.create("pc1","ParametricCurve");
g.create("swe1","Sweep");
g.feature("pc1").set("parmax","pi");
g.feature("pc1").set("coord",new String[]{"(cos(s)-1)*3","0","sin(s)*3"});
g.feature("swe1").selection("face").set("wp1.c1", new int[]{1});
g.feature("swe1").selection("edge").set("pc1(1)",new int[]{1});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('wp1','WorkPlane');
g.feature('wp1').geom.create('c1','Circle');
g.create('pc1','ParametricCurve');
g.create('swe1','Sweep');
g.feature('pc1').set('parmax','pi');
g.feature('pc1').set('coord',{'(cos(s)-1)*3','0','sin(s)*3'});
g.feature('swe1').selection('face').set('wp1.c1', 1);
g.feature('swe1').selection('edge').set('pc1(1)', 1);
g.run;
```

SEE ALSO

[Extrude](#), [Helix](#), [Revolve](#), [WorkPlane](#)

Tangent

Create a tangent line segment to one or two 2D edges.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Tangent");
model.geom(<tag>).feature(<ftag>).selection(property);
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Tangent")` to create a line segment tangent to two edges or tangent to one edge with a fixed end point. The following properties are available:

TABLE 3-100: VALID PROPERTY/VALUE PAIRS FOR TANGENT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
edge	Selection		Edge in an geometry object to find tangent to.
start	double	0.5	Start guess for parameter value of point of tangency.
type	edge point coord	edge	Type of tangent.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj bnd edg pnt off	bnd	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

If `type` is `edge` a common tangent line to two edges are constructed. Then, the following additional properties are available:

TABLE 3-101: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
edge2	Selection		Second edge in some geometry object to find tangent to.
start2	double	0.5	Start guess for parameter value of point of tangency.

If `type` is `point` a tangent line through a given point is constructed. Then, the following additional property is available:

TABLE 3-102: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
point	Selection		Point in some geometry object.

If `type` is `coord` a tangent line through a point with given coordinates are constructed. Then, the following additional property is available:

TABLE 3-103: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
coord	double[]	{0,0}	Coordinates.

If a tangent cannot be found, a tangent to some adjacent edge is constructed, if possible.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(gname).create(fname,"tangent")` creates a Tangent feature.

The following properties are no longer supported:

TABLE 3-104: OBSOLETE PROPERTY/VALUE PAIRS

PROPERTY NAME	PROPERTY VALUE	DEFAULT	DESCRIPTION
edim1	0 1	geometry dependent	Starting point element dimension: 0 for vertex, 1 for edge.
edim2	0 1	geometry dependent	Ending point element dimension: 0 for vertex, 1 for edge.
dom1	integer	1	Starting point entity number.
dom2	integer	1	Ending point entity number.
out	cell array of Strings	{ }	Additional output data.
start1	double	0.5	Starting point parameter value on specified edge.

EXAMPLE

The following sequence generates a tangent from the unit circle to the point (2, 0):

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("c1","Circle");
g.run("c1");
g.create("tan1","Tangent");
g.feature("tan1").set("type","coord");
g.feature("tan1").selection("edge").set("c1",3);
g.feature("tan1").set("coord","2 0");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('c1','Circle');
g.run('c1');
g.create('tan1','Tangent');
g.feature('tan1').set('type','coord');
g.feature('tan1').selection('edge').set('c1',3);
g.feature('tan1').set('coord','2 0');
g.run;
```

The following sequence generates a common tangent between two circles:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",2);
g.create("c1","Circle");
g.create("c2","Circle");
g.feature("c2").set("pos", "2 2");
g.run("c2");
g.create("tan1","Tangent");
g.feature("tan1").selection("edge").set("c1",4);
g.feature("tan1").selection("edge2").set("c2",4);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',2);
g.create('c1','Circle');
g.create('c2','Circle');
g.feature('c2').set('pos', '2 2');
g.run('c2');
g.create('tan1','Tangent');
```

```
g.feature('tan1').selection('edge').set('c1',4);
g.feature('tan1').selection('edge2').set('c2',4);
g.run;
```

SEE ALSO

[BezierPolygon](#)

Tetrahedron

Create a solid or surface tetrahedron in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Tetrahedron");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Tetrahedron")` to create a tetrahedron. The following properties are available:

TABLE 3-105: VALID PROPERTY/VALUE PAIR FOR TETRAHEDRON

PROPERTY NAME	PROPERTY VALUE	DEFAULT	DESCRIPTION
p	double[3][4]	{{0,0,1,0}, {0,1,0,0}, {0,0,0,1}}	Corner coordinates.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"tetrahedron3")` creates a solid tetrahedron.

`model.geom(<tag>).create(<ftag>,"tetrahedron2")` creates a surface tetrahedron.

EXAMPLE

The following commands generate a solid tetrahedron object:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("t1","Tetrahedron");
g.feature("t1").set("p", new double[][]{{0,0,1,0},{0,0.8,1,0},{0,0.1,0,0.2}});
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('t1','Tetrahedron');
g.feature('t1').set('p', [[0,0,1,0];[0,0.8,1,0];[0,0.1,0,0.2]]);
g.run;
```


SEE ALSO

[Hexahedron](#), [Pyramid](#)

Torus

Create a solid or surface torus in 3D.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"Torus");  
model.geom(<tag>).feature(<ftag>).set(property,<value>);  
model.geom(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.geom(<tag>).create(<ftag>,"Torus")` to create a torus. The following properties are available:

TABLE 3-106: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
angle	double	360	Revolution angle.
axis	double[]	{0,0,1}	Direction of the revolution axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
axistype	x y z cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
intfaces	on off	off	Create cross section faces inside the torus.
pos	double[]	{0,0,0}	Center coordinates.
rmaj	double	1	Directrix radius.
rmin	double	0.5	Generatrix radius.
rot	double	0	Rotational angle about axis.
type	solid surface	solid	Object type.
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj dom bnd edg pnt off	dom	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#).

COMPATIBILITY

`model.geom(<tag>).create(<ftag>,"torus3")` creates a solid torus.

`model.geom(<tag>).create(<ftag>,"torus2")` creates a surface torus.

The following properties are also available:

TABLE 3-107: VALID PROPERTY/VALUE PAIRS FOR TORUS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ax2	double[]	{0,0}	Alias for axis when axistype is spherical.
ax3	double[]	{0,0,1}	Alias for axis when axistype is cartesian.
x, y, z	double	0	Alias for pos.

The property `const` is no longer available.

EXAMPLE

The following sequence generates a surface torus and a solid torus:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("t2","Torus");
g.feature("t2").set("type","surface");
g.feature("t2").set("rmaj",2);
g.feature("t2").set("rmin",1);

g.create("t3","Torus");
g.feature("t3").set("rmaj",10);
g.feature("t3").set("rmin",2);
g.feature("t3").set("pos","0,0,-100");
g.feature("t3").set("axis","1,1,1");
g.feature("t3").set("rot",60);
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('t2','Torus');
g.feature('t2').set('type','surface');
g.feature('t2').set('rmaj',2);
g.feature('t2').set('rmin',1);

g.create('t3','Torus');
g.feature('t3').set('rmaj',10);
g.feature('t3').set('rmin',2);
g.feature('t3').set('pos','0,0,-100');
g.feature('t3').set('axis','1,1,1');
g.feature('t3').set('rot',60);
g.run;
```

SEE ALSO

[Cylinder](#)

UnionSelection, IntersectionSelection, DifferenceSelection, ComplementSelection

Combine selections of entities or objects using a Boolean operation.

SYNTAX

```
model.geom(<tag>).create(<ftag>,"UnionSelection");
model.geom(<tag>).create(<ftag>,"IntersectionSelection");
model.geom(<tag>).create(<ftag>,"DifferenceSelection");
model.geom(<tag>).create(<ftag>,"ComplementSelection");
model.geom(<tag>).feature().set(property,<value>);
model.geom(<tag>).feature().getType(property);
```

DESCRIPTION

Use UnionSelection to get all entities/objects that belong to at least one of the input selections. Use IntersectionSelection to get all entities/objects that belong to all input selections. Use DifferenceSelection to get all entities/objects that belong some of the add selections, but do not belong to any of the subtract selections. Use ComplementSelection to get all entities/objects of the given dimension that do not belong to any input selection.

For DifferenceSelection, the following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
add	String[]	{}	Tags of selections to add.
entitydim	-1 0 1 2 3	space dimension	Dimension of entities to select. -1 means Object.
subtract	String[]	{}	Tags of selections to subtract.
selkeep	on off	on	Keep the selection within the geometry sequence.
selshow	on off	on	Show selection in physics, materials, and so on, or in part instances. This property is not available in a work plane's Plane Geometry.
contributeto	String	none	Tag of cumulative selection to contribute to.

For the other selections, the following properties are available:

PROPERTY	VALUE	DEFAULT	DESCRIPTION
entitydim	-1 0 1 2 3	space dimension	Dimension of entities to select. -1 means Object.
input	String[]	{}	Tags of input selections.
selkeep	on off	on	Keep the selection within the geometry sequence.
selshow	If the level is not Object, the allowed values are on off. If the level is Object, the allowed values are all obj dom bnd edg pnt off.	If the level is not Object, the default value is on. If the level is Object, the default value is all in a component's geometry, obj in a part, and dom in a work plane's Plane Geometry.	Show selection in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. This property is not available in a work plane's Plane Geometry.
contributeto	String	none	Tag of cumulative selection to contribute to.

See [Selections of Geometric Entities](#) for general information about selections.

EXAMPLE

In an array of blocks, select all vertices whose distance to the origin is between 2.5 and 3.5. This results in 22 vertices in 7 different objects.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
g.create("blk1", "Block");
g.create("arr1", "Array");
g.feature("arr1").selection("input").set("blk1");
g.feature("arr1").set("size", new int[]{3,3,1});
g.feature("arr1").set("displ", new double[]{1.5,1.5,0});
g.run("arr1");
g.create("ballsel1", "BallSelection");
g.feature("ballsel1").set("entitydim", 0);
g.feature("ballsel1").set("r", 3.5);
g.feature().duplicate("ballsel2", "ballsel1");
g.feature("ballsel2").set("r", 2.5);
g.create("difs1", "DifferenceSelection");
g.feature("difs1").set("entitydim", 0);
g.feature("difs1").set("add", new String[]{"ballsel1"});
g.feature("difs1").set("subtract", new String[]{"ballsel2"});
g.run("difs1");
String[] obj = g.selection("difs1").objects();
```

```

int nVtx = 0;
for (int i=0; i<obj.length; ++i)
    nVtx += g.selection("difsel1").entities(obj[i],0).length;
// obj.length = 7, nVtx = 22

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
g.create('blk1', 'Block');
g.create('arr1', 'Array');
g.feature('arr1').selection('input').set('blk1');
g.feature('arr1').set('size', [3,3,1]);
g.feature('arr1').set('displ', [1.5,1.5,0]);
g.run('arr1');
g.create('ballsel1', 'BallSelection');
g.feature('ballsel1').set('entitydim', 0);
g.feature('ballsel1').set('r', 3.5);
g.feature.duplicate('ballsel2','ballsel1');
g.feature('ballsel2').set('r', 2.5);
g.create('difsel1', 'DifferenceSelection');
g.feature('difsel1').set('entitydim', 0);
g.feature('difsel1').set('add', {'ballsel1'});
g.feature('difsel1').set('subtract', {'ballsel2'});
g.run('difsel1');
obj = g.selection('difsel1').objects;
nVtx = 0;
for i=1:length(obj)
    nVtx = nVtx + length(g.selection('difsel1').entities(obj(i),0));
end
% length(obj) = 7, nVtx = 22

```

SEE ALSO

[AdjacentSelection](#), [BallSelection](#), [BoxSelection](#), [CylinderSelection](#), [ExplicitSelection](#)

WorkPlane

Create a work plane in 3D for drawing 2D objects that can be extruded or revolved.

SYNTAX

```

model.geom(<tag>).create(<ftag>,"WorkPlane");
model.geom(<tag>).feature(<ftag>).set(property,<value>);
model.geom(<tag>).feature(<ftag>).getType(property);
model.geom(<tag>).feature(<ftag>).geom().geomSequenceMethod
model.geom(<tag>).feature(<ftag>).geom().feature();

```

DESCRIPTION

A work plane embeds 2D objects in 3D. The sections below describe how to define the location of the work plane and how to create 2D objects in it. You can also use a work plane in the [CrossSection](#) and [Partition](#) features. In that case, you do not need to draw anything in the work plane.

Unite Objects

There is an option to unite all objects in the work plane before using the 2D geometry in 3D. Uniting all objects can improve the handling of the 2D geometry when extruding it, for example. You can control the union of 2D objects using the following properties:

TABLE 3-108: VALID PROPERTY/VALUE PAIRS, UNITE OBJECTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
repairtol	double	See below	Repair tolerance, relative to size of union of objects.
unite	on off	off	Unite objects.

The default values for the repair tolerance is taken from the geometry sequence's default repair tolerance, given by the `repairTol()` method.

Visualization

To specify the in-plane visualization of the 3D geometry and activate the ability to draw directly on the work plane in 3D, use the following properties:

TABLE 3-109: VALID PROPERTY/VALUE PAIRS, VISUALIZATION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
showcoincident	on off	on	Show coincident 3D geometry.
showintersection	on off	on	Show intersection of 3D geometry.
showprojection	on off	on	Show projection of 3D geometry.
workplane3d	on off	of	Draw on work plane in 3D.

Defining the Location of the Work Plane

A work plane has a local coordinate system that is orthonormal and positively oriented (right-handed). The work plane coincides with the *xy*-plane in the local coordinate system. The following properties control how the work plane is defined.

TABLE 3-110: VALID PROPERTY/VALUE PAIRS, LOCATION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
planetype	quick faceparallel edgeparallel edgeangle circleperpendicular vertices coordinates transformed	quick	Type of data defining the work plane.
contributeto	String	none	Tag of cumulative selection to contribute to.

For information about the `selresult` and `contributeto` properties, see [Selections of Geometric Entities](#). Note that in a work plane's Plane Geometry, the `selresultshow` and `selindividualshow` properties are not available.

Depending on `planetype`, additional properties are available.

Quick

This creates a work plane parallel to one of the global coordinate planes.

TABLE 3-111: VALID PROPERTY/VALUE PAIRS, QUICK

PROPERTY	VALUE	DEFAULT	DESCRIPTION
quickplane	xy yz zx yx zy xz	xy	Coordinate plane.
quickx	double	0	x-coordinate for work plane (used when plane is yz or zy).
quicky	double	0	y-coordinate for work plane (used when plane is xz or zx).
quickz	double	0	z-coordinate for work plane (used when plane is xy or yx).
quickoffsettype	distance vertex	distance	Type of offset specification.
offsetvertex	Selection		Vertex for offset.
quickorigin	global vertexproj	global	Origin of local coordinate system.
originvertex	Selection		Vertex for origin.
quickaxis	natural vertexproj	natural	Local x-axis.
axisvertex	Selection		Vertex for axis.

TABLE 3-111: VALID PROPERTY/VALUE PAIRS, QUICK

PROPERTY	VALUE	DEFAULT	DESCRIPTION
displ	double[2]	{0,0}	Displacement of local coordinate system.
rot	double	0	Rotation angle of local coordinate system.

Face Parallel

This creates a work plane that is parallel to a planar face in a geometry object

TABLE 3-112: VALID PROPERTY/VALUE PAIRS, FACE PARALLEL

PROPERTY	VALUE	DEFAULT	DESCRIPTION
face	Selection		Planar face.
offset	double	0	Signed offset in the direction of the local z-axis.
reverse	on off	off	Reverse direction of local z-axis.
offsettype	distance vertex	distance	Type of offset specification.
offsetvertex	Selection		Vertex for offset.
origin	facecenter boxcorner vertexproj	facecenter	Origin of local coordinate system.
originvertex	Selection		Vertex for origin.
faceparallelaxis	s1 s2 vertexproj	s1	Local x-axis.
axisvertex	Selection		Vertex for axis.
displ	double[2]	{0,0}	Displacement of local coordinate system.
rot	double	0	Rotation angle of local coordinate system.

Edge Parallel

This creates a work plane that is parallel to a planar edge in a geometry object.

TABLE 3-113: VALID PROPERTY/VALUE PAIRS, FACE PARALLEL

PROPERTY	VALUE	DEFAULT	DESCRIPTION
edge	Selection		Planar edge.
offset	double	0	Signed offset in the direction of the local z-axis.
origin	edgecenter boxcorner	edgecenter	Origin of local coordinate system.
reverse	on off	off	Reverse direction of local z-axis.
offsettype	distance vertex	distance	Type of offset specification.
offsetvertex	Selection		Vertex for offset.
edgeparallelorigin	startvertex endvertex vertexproj	startvertex	Origin of local coordinate system.
originvertex	Selection		Vertex for origin.
edgeparallelaxis	tangent vertexproj	tangent	Local x-axis.
axisvertex	Selection		Vertex for axis.
displ	double[2]	{0,0}	Displacement of local coordinate system.
rot	double	0	Rotation angle of local coordinate system.

Edge Angle

This creates a work plane through a straight edge of a geometry object. The work plane makes a given angle with the tangent plane of a face in the same geometry object. The face must be adjacent to the edge, and its tangent plane must be the same at all points on the edge. The origin of the local coordinate system coincides with the start vertex (if `reverse` is `off`) or end vertex (if `reverse` is `on`) of the edge. The direction of the local *x*-axis coincides

with the direction of the edge (if `reverse` is `off`) or its opposite (if `reverse` is `on`). If the property `angle` is zero, the direction of the local *y*-axis points into the face. In general, the local coordinate system is rotated by `angle` about the local *x*-axis.

TABLE 3-114: VALID PROPERTY/VALUE PAIRS, EDGE ANGLE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>angle</code>	<code>double</code>	<code>0</code>	Angle between face and work plane.
<code>edge</code>	<code>Selection</code>		Straight edge.
<code>adjface</code>	<code>Selection</code>		Face adjacent to edge in the same object.
<code>reverse</code>	<code>on off</code>	<code>off</code>	Reverse direction of local <i>x</i> -axis.
<code>displ</code>	<code>double[2]</code>	<code>{0,0}</code>	Displacement of local coordinate system.
<code>rot</code>	<code>double</code>	<code>0</code>	Rotation angle of local coordinate system.

Circle Perpendicular

This creates a work plane that is perpendicular to a given circular edge. The origin of the local coordinate system is at the circle's center. By default, the local *x*-axis goes through the edge's start vertex. Thus, if the geometry is rotationally symmetric, the symmetry axis coincides with the local *y*-axis.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>circedge</code>	<code>Selection</code>	<code>empty</code>	Circular edge.
<code>circpoint</code>	<code>startvertex endvertex othervertex</code>	<code>startvertex</code>	Point on plane.
<code>circvertex</code>	<code>Selection</code>		Vertex on plane.
<code>circoffset</code>	<code>double</code>	<code>0</code>	Offset angle.
<code>reverse</code>	<code>on off</code>	<code>off</code>	Reverse direction of local <i>x</i> -axis.
<code>displ</code>	<code>double[2]</code>	<code>{0,0}</code>	Displacement of local coordinate system.
<code>rot</code>	<code>double</code>	<code>0</code>	Rotation angle of local coordinate system.

Vertices

This creates a work plane parallel to a plane through three vertices `v1`, `v2`, and `v3`. When `offset=0`, the origin of the local coordinate system coincides with the first vertex `v1`. The *x*-axis of the local coordinate system is in the direction `v2-v1`. The direction of the local *z*-axis is given by the cross product $(v2-v1) \times (v3-v1)$ or its opposite (if `reverse` is `on`).

TABLE 3-115: VALID PROPERTY/VALUE PAIRS, VERTICES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>offset</code>	<code>double</code>	<code>0</code>	Signed offset in the direction of the local <i>z</i> -axis.
<code>reverse</code>	<code>on off</code>	<code>off</code>	Reverse direction of local <i>z</i> -axis.
<code>vertex1</code>	<code>Selection</code>		First vertex.
<code>vertex2</code>	<code>Selection</code>		Second vertex.
<code>vertex3</code>	<code>Selection</code>		Third vertex.
<code>displ</code>	<code>double[2]</code>	<code>{0,0}</code>	Displacement of local coordinate system.
<code>rot</code>	<code>double</code>	<code>0</code>	Rotation angle of local coordinate system.

Coordinates

This creates a work plane through three points p_1 , p_2 , and p_3 . The origin of the local coordinate system coincides with the first point p_1 . The x -axis of the local coordinate system is in the direction $p_2 - p_1$. The direction of the local z -axis is given by the cross product $(p_2 - p_1) \times (p_3 - p_1)$.

TABLE 3-116: VALID PROPERTY/VALUE PAIRS, COORDINATES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
genpoints	double[3][3]	{{0,0,0},{1,0,0},{0,1,0}}	Points.

`genpoints[n][i]` is the i th coordinate of the n th point.

Transformed

This creates a work plane as a transformation of another work plane, using a displacement and a rotation.

TABLE 3-117: VALID PROPERTY/VALUE PAIRS, TRANSFORMED

PROPERTY	VALUE	DEFAULT	DESCRIPTION
workplanesrc	String	this	Tag of PartInstance feature to take work plane from, or this to take work plane from this sequence.
workplane	String	xyplane	Tag of input work plane, or xyplane.
transdispl	double[3]	{0,0,0}	Displacement in local coordinate system.
transaxistype	x y z cartesian spherical	z	Type of rotation axis.
transaxis	double[]	{0,0,0}	Rotation axis in local coordinate system. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
transrot	double	0	Rotation angle.

Part Instances

In a part instance, the following property is available:

TABLE 3-118: VALID PROPERTY IN PART INSTANCES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
showworkplane	on off	on	Show work plane in part instances. This property is only available if the work plane is in a geometry part.

Selections of Resulting Entities

For selections of resulting entities, the following properties are available:

TABLE 3-119: VALID PROPERTY/VALUE PAIRS FOR SELECTIONS OF RESULTING ENTITIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
selresult	on off	off	Create selections of all resulting objects.
selresultshow	all obj bnd edg pnt off	bnd	Show selections of resulting objects in physics, materials, and so on, or in part instances. obj is not available in a component's geometry. dom, bnd, and edg are not available in all features.

Creating 2D Objects in the Work Plane

The work plane owns a geometry sequence that contains the features that define the 2D objects you draw in the work plane. You access this geometry sequence by

```
model.geom(<tag>).feature(<ftag>).geom()
```

where `<ftag>` is the name of the work plane feature. You can add geometry features in this 2D sequence as usual.

COMPATIBILITY

The plane type `circularedge` from earlier versions is still valid as an alternative to its replacement `circleperpendicular`, and the plane type `general` from earlier versions is still valid as an alternative to its replacement `coordinates`.

EXAMPLE

Create a work plane with a rectangle. When the work plane is built, the rectangle is embedded in the space of the 3D sequence:

Code for use with Java

```
Model model = ModelUtil.create("Model1");
GeomSequence g = model.geom().create("geom1",3);
g.create("wp1", "WorkPlane");
g.feature("wp1").set("quickplane", "yz");
g.feature("wp1").geom().create("r1", "Rectangle");
g.feature("wp1").geom().feature("r1").set("pos", "1 1");
g.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model1');
g = model.geom.create('geom1',3);
g.create('wp1', 'WorkPlane');
g.feature('wp1').set('quickplane', 'yz');
g.feature('wp1').geom.create('r1', 'Rectangle');
g.feature('wp1').geom.feature('r1').set('pos', '1 1');
g.run;
```

SEE ALSO

[CrossSection](#), [Extrude](#), [Partition](#), [Revolve](#), [Sweep](#)

4

Mesh

Details include reference information about the mesh commands and utility methods.

In this chapter:

- [About Mesh Commands](#)
- [Working with a Meshing Sequence](#)
- [Physics-Controlled Meshing](#)
- [Information and Statistics](#)
- [Getting and Setting Mesh Data](#)
- [Errors and Warnings](#)
- [Exporting Meshes to Files](#)
- [Mesh Commands](#)

About Mesh Commands

The following list includes the mesh commands that are documented in this chapter (listed in alphabetical order):

- [Ball](#)
- [BndLayer](#)
- [BndLayerProp](#)
- [Box](#)
- [Convert](#)
- [CopyEdge](#)
- [CopyFace](#)
- [CopyDomain](#)
- [Copy](#)
- [CornerRefinement](#)
- [CreateVertex](#)
- [Cylinder](#)
- [Delete](#)
- [DeleteEntities](#)
- [DetectFaces](#)
- [Distribution](#)
- [Edge](#)
- [EdgeGroup](#)
- [EdgeMap](#)
- [FreeQuad](#)
- [FreeTet](#)
- [FreeTri](#)
- [Import](#)
- [JoinEntities](#)
- [LogicalExpression](#)
- [Map](#)
- [OnePointMap](#)
- [Point](#)
- [Reference](#)
- [Refine](#)
- [Scale](#)
- [Size](#)
- [Sweep](#)
- [TwoPointMap](#)



- [Operation Features](#)
- [Attribute Features](#)
- [Features for Imported Meshes](#)

Operation Features

Table 4-1 is an overview of the features that create or modify the mesh corresponding to a geometry.

TABLE 4-1: MESH OPERATION FEATURES

FEATURE TYPE	LOCALIZED TYPE DESCRIPTION IN GUI	DEFAULT NAME IN GUI
BndLayer	Boundary Layers	bl
Convert	Convert	conv
Copy	Copy	copy
CopyEdge	Copy Edge	cpe
CopyFace	Copy Face	cpf
CopyDomain	Copy Domain	cpd
Delete	Delete	del
Edge	Edge	edg
FreeQuad	Free Quad	fq

TABLE 4-1: MESH OPERATION FEATURES

FEATURE TYPE	LOCALIZED TYPE DESCRIPTION IN GUI	DEFAULT NAME IN GUI
FreeTet	Free Tetrahedral	ftet
FreeTri	Free Triangle	ftri
Map	Mapped	map
Reference	Reference	rf
Refine	Refine	ref
Sweep	Sweep	swe

Attribute Features

Table 4-2 is an overview of the features that contain properties used by operation features to build the mesh.

TABLE 4-2: MESH ATTRIBUTE FEATURES

FEATURE TYPE	LOCALIZED TYPE DESCRIPTION IN GUI	DEFAULT NAME IN GUI
BndLayerProp	Boundary Layer Properties	blp
CornerRefinement	Corner Refinement	cr
Distribution	Distribution	dis
EdgeGroup	Edge Groups	eg
EdgeMap	Edge Map	em
OnePointMap	One-Point Map	pm
Scale	Scale	sca
Size	Size	size
TwoPointMap	Two-Point Map	ppm

Features for Imported Meshes

Table 4-3 is an overview of the features that operate on imported meshes.

TABLE 4-3: FEATURES FOR IMPORTED MESHES

FEATURE TYPE	LOCALIZED TYPE DESCRIPTION IN GUI	DEFAULT NAME IN GUI
Ball	Ball	ball
Box	Box	box
CreateVertex	Create Vertex	vtx
Cylinder	Cylinder	cyl
DeleteEntities	Delete Entities	dele
DetectFaces	Detect Faces	detf
Import	Import	imp
JoinEntities	Join Entities	join
LogicalExpression	Logical Expression	le

Working with a Meshing Sequence

This section describes how to build meshes using Java[®] methods. A *mesh* is defined by a *meshing sequence* consisting of *mesh features*. A meshing feature is either an *attribute feature* or an *operation feature*. Each operation feature modifies the mesh when you *build* the feature using properties defined by attribute features.

An attribute feature is defined on a geometric entity selection and has a set of properties. Running an attribute feature does not change the mesh, but affects the subsequent operation features in the sequence. For example, the **FreeTet** operation feature, that creates a tetrahedral mesh, uses properties from the **Distribution**, **Scale**, and **Size** attribute features, that define the size and distribution of the mesh elements. You can add an attribute feature directly to the meshing sequence, this is referred to as a *global attribute feature*, or add it to an operation feature, this is referred to as a *local attribute feature*. Properties defined in local attribute features of an operation feature overrides corresponding properties defined in preceding global feature properties (on the same selection).

An operation features makes operations on the mesh as defined by the meshing sequence. Some operation features, like **FreeTet** and **Sweep** generate new mesh. Other operation features, like **Refine** and **Convert** modify existing mesh.

In this section:

- [Adding a Meshing Sequence](#)
- [Adding a Mesh Feature](#)
- [Editing a Mesh Feature](#)
- [Building Mesh Features](#)
- [Using Mesh Parts](#)
- [Feature Status](#)
- [Deleting Mesh Features](#)
- [Disabling Mesh Features](#)
- [Clearing Meshes](#)
- [Units](#)
- [Selections](#)



Meshing in the *COMSOL Multiphysics Reference Manual*

Adding a Meshing Sequence

To add a new meshing sequence to a model object `model`, enter

```
model.mesh().create(<mTag>,<gTag>);
```

where `mTag` is the mesh's tag (an identifier of your choice) and `gTag` is the tag of the associated geometry. If you want to import a mesh, you must specify an empty geometry sequence; the geometry is then defined by the mesh.

Adding a Mesh Feature

To add a feature to a mesh with tag `<tag>`, enter

```
model.mesh(<tag>).create(<fTag>,ftype);
```

where *<ftag>* is the feature's tag (an identifier of your choice), and *ftype* is the feature's type. Feature types are capitalized and case-sensitive (for example, `FreeTet`).

When you add a feature, it is inserted after the *current feature*. You can get the tag of the current feature type by entering

```
String ftag = model.mesh(<tag>).current();
```

If *ftag* is the empty string, the current feature is the beginning of the meshing sequence, that is, the empty state before all features. Adding a meshing feature, it automatically becomes current, but it is not built automatically.

For some operation features it is possible to add attribute features. To add an attribute feature to an operation feature, enter

```
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype);
```

where *<ftag1>* is the attribute feature's tag (an identifier of your choice), and *ftype* is the attribute feature's type.

All properties in a new feature get a default value.

Editing a Mesh Feature

To change a property value in a feature, enter

```
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
```

where *property* is a property name and *<value>* is a property value.

All numeric properties can be given either as a numeric value or as a string expression that can contain parameters defined in `model.param()`. When building the feature, the string expressions are evaluated using the current values of the parameters.



Behavior has changed since COMSOL Multiphysics version 3.5a. String expression arguments are retained and not converted to numeric values.

To get the value of a property, enter one of the following, depending on the property type:

```
double d = model.mesh(<tag>).feature(<ftag>).getDouble(property);
String s = model.mesh(<tag>).feature(<ftag>).getString(property);
double[] da = model.mesh(<tag>).feature(<ftag>).getDoubleArray(property);
String[] sa = model.mesh(<tag>).feature(<ftag>).getStringArray(property);
```

If you request a numerical value for a string property, it is evaluated using the current values of the parameters in `model.param()`.

Building Mesh Features

To modify the mesh, you must *build* an operation feature. Enter

```
model.mesh(<tag>).run(<ftag>);
```

to build the feature *<ftag>* and all its preceding features (the features are built in the order from the first to the last). When the build has completed, the feature *<ftag>* becomes the current feature.

To build all features, enter

```
model.mesh(<tag>).run();
```

Using Mesh Parts

For a description of mesh parts, see [Using Mesh Parts](#) in the *COMSOL Multiphysics Reference Guide*.

A mesh part is defined by a model component of `MeshComponent` type (a *mesh component*). A mesh component contains a single meshing sequence and a geometry that defines the dimension, selections, and geometrical properties such as the length and angular unit. When you work with a Mesh Part node in the user interface, you can modify the meshing sequence of the mesh component. When working with the API it is important to be aware of the corresponding mesh component and its geometry.

To create a mesh part, enter

```
model.modelNode().create(<mcomptag>, "MeshComponent");
model.geom().create(<mgeomtag>, sDim);
model.mesh().create(<tag>, <mgeomtag>);
```

where `<mcomptag>` is the tag of the mesh component, `<mgeomtag>` is the tag of the geometry, `sDim` is its space dimension (1, 2, or 3), and `<tag>` is the mesh part's tag.

Use `model.geom(<mgeomtag>)` to access the geometry properties, such as length units (see `model.geom(<tag>)`).

To work with the mesh part, use `model.mesh(<tag>)`, see the sections above in [Working with a Meshing Sequence](#) and [Table 4-3](#), Features for Imported Meshes, for details.

For using a mesh part in a model component, see [Import Mesh Part or Meshing Sequence](#) under Geometry Commands to create a geometry from the part, or [Import](#) under Mesh Commands to use the resulting mesh as an imported mesh.

To remove a mesh part, use `model.modelNode().remove(<mcomptag>);`

Feature Status

The *status* of a feature can be one of the following:

- *Built*. This means that the none of the feature's properties have changed since the feature was last built, and the features of the input objects are all built. The feature can contain warning messages.
- *Edited*. This means that some of the feature's properties have changed since the feature was last built.
- *Needs rebuild*. This means that any of the preceding features is edited.
- *Error*. This means that the feature contains an error message.
- *Warning*. This means that the feature contains a warning message.

You can examine the status of a feature by entering

```
boolean built = model.mesh(<tag>).feature(<ftag>).isBuilt();
boolean edited = model.mesh(<tag>).feature(<ftag>).isEdited();
boolean hasError = model.mesh(<tag>).feature(<ftag>).hasError();
boolean hasWarning = model.mesh(<tag>).feature(<ftag>).hasWarning();
boolean needsRebuild = !(built || edited || hasError || hasWarning);
```

Deleting Mesh Features

To delete a feature, enter

```
model.mesh(<tag>).feature().remove(<ftag>);
```

Disabling Mesh Features

To disable a feature, enter


```
model.mesh(<tag>).feature(<ftag>).active(false);
```

To enable a disabled feature, enter

```
model.mesh(<tag>).feature(<ftag>).active(true);
```

You can get the enabled/disabled status of a feature by entering

```
boolean isEnabled = model.mesh(<tag>).feature(<ftag>).active();
```

Clearing Meshes

To clear the mesh of a sequence, enter

```
model.mesh(<tag>).clearMesh();
```

To clear the mesh and remove all features in a sequence, enter

```
model.mesh(<tag>).feature().clear();
```

To clear all meshes for all geometries in a model, enter

```
model.mesh().clearMeshes();
```

Units

The meshing sequence uses the same base unit system as the geometry sequence. The string versions of setters and getters support units and unit conversion using the standard machinery.

Selections

Most mesh features have selections, to specify where they operate. To access a feature's selection, use the syntax

```
model.mesh(<tag>).feature(<ftag>).selection();
```

To specify the entire geometry, write

```
model.mesh(<tag>).feature(<ftag>).selection().allgeom();
```

To specify all geometric entities in dimension *<dim>*, write

```
model.mesh(<tag>).feature(<ftag>).selection().geom(<dim>).all();
```

To specify the geometric entities that remains to be meshed when the feature is about to be built, use

```
model.mesh(<tag>).feature(<ftag>).selection().remaining();
```

It is not possible to retrieve the geometric entities of this selection, unless the feature is built.

If *entities* is an integer array of geometric entities in dimension *dim*, use the following syntax to select these entities

```
model.mesh(<tag>).feature(<ftag>).selection().geom(dim).set(entities);
```

For example, to selection domain 1 and 2 in a 3D geometry, write

```
model.mesh(<tag>).feature(<ftag>).selection().geom(3).set(new int[]{1,2});
```

To add the geometric entities specified in the integer array *<entities>* in dimension *<dim>* to the selection, write

```
model.mesh(<tag>).feature(<ftag>).selection().geom(<dim>).add(<entities>);
```

To remove the geometric entities specified in the integer array *<entities>* in dimension *<dim>* from the selection, write

```
model.mesh(<tag>).feature(<ftag>).selection().geom(<dim>).remove(<entities>);
```

To clear the selection in dimension *<dim>*, write

```
model.mesh(<tag>).feature(<ftag>).selection().geom(<dim>).clear();
```

Some features have more than one selection, for example **sweep**, where it is possible to specify **source** and **destination** faces. Use the following syntax to access these selections.

```
model.mesh(<tag>).feature(<ftag>).selection(<property>);
```

Thus, to specify boundary 5 as source face on the sweep feature **swe1**, write

```
model.mesh(<tag>).feature(<ftag>).selection("sourceface").geom(2).set(5);
```

Physics-Controlled Meshing



When a physics-controlled sequence is built, a sequence of ordinary meshing features is created. This sequence can be customized by editing these features. However, do not assume the existence of a certain feature in a Java program designed to run with future versions of COMSOL Multiphysics. The actual contents of the sequence might change.

A physics-controlled meshing sequence examines the physics to automatically determine size attributes and sequence operations needed to create a mesh adapted to the problem. The physics-controlled sequence is based on heuristics and knowledge built-in by application experts. It is not adapted by numerical error estimates—that type of adaptation is provided by mesh adaptation in the solver sequence.

When a mesh is built or a problem solved, the physics-controlled sequence is updated to match the currently active physics. If the sequence is in any other state than physics-controlled, it is not updated or modified before it is built.

By default, a meshing sequence is in the physics-controlled state. If you manually add a feature to the sequence or edit a feature, the sequence automatically switches to the user-controlled state. It is also possible to explicitly switch to user-controlled state by entering

```
model.mesh(<tag>).automatic(false);
```

To switch back to physics-controlled mesh, enter

```
model.mesh(<tag>).automatic(true);
```

The current sequence is then modified or overwritten next time the sequence is built or the problem is solved.

Use `model.mesh(<tag>).isAutomatic()` to determine in which state the sequence is.

You can adjust the overall size of a physics-induced mesh by using the method

```
model.mesh(<tag>).autoMeshSize(<size>);
```

The value 5 of `<size>` corresponds to the default size, the values 4, 3, 2, and 1 give you an increasingly finer mesh, whereas the values 6, 7, 8, and 9 give you a coarser mesh. The method `model.mesh(<tag>).autoMeshSize()` returns the current size adjustment.

Information and Statistics

In this section:

- [Statistics](#)
- [Number and Types of Elements](#)
- [Quality of Elements](#)
- [Volume of Elements and Mesh](#)
- [Mesh Status](#)

Statistics

Use the `stat()` method on the meshing sequence to determine the number of elements of different types and the quality of elements. Use the `info()` method to obtain topological information, the number of geometric entities, and so forth. When importing mesh, use `infoCurrent()` to obtain topological information about the current (last built) mesh feature, and `info()` to obtain this information about the finalized mesh. See [Geometry Object Information Methods](#) for a list of available methods.

The `stat()` method returns an object with a collection of methods that can be queried for statistical information about the current mesh. There is also a selection,

```
model.mesh(<tag>).stat().selection()
```

which is used to select geometric entities for which the statistics is calculated. The default selection is the entire geometry. The methods described below also exist directly on the meshing sequence. These methods always return statistics for the entire geometry.

Statistics can be requested per element type. The type is given as a string, denoted *type*, and the possible types are listed in the following table.

TABLE 4-4:

STRING	ELEMENT	ELEMENT DIMENSION
vtx	Vertex element	0
edg	Edge element	1
tri	Triangular element	2
quad	Quadrilateral element	2
tet	Tetrahedral element	3
pyr	Pyramid element	3
prism	Prism element	3
hex	Hexahedral element	3
all	All elements of maximal dimension in the selection	

The parameter string `all` gives statistics for all elements with the same dimension as the maximal dimension of the current selection. For example, if the entire geometry is selected in 2D, the parameter `all` provides combined statistics for triangular and quadrilateral elements.



- [Geometry Object Information](#)
- [Selections](#)

Number and Types of Elements

To determine the number of elements of a certain type, use

```
int model.mesh(<tag>).stat().getNumElem(type);
```

To get the number of elements of all types, use

```
int model.mesh(<tag>).stat().getNumElem();
```

To determine the element types, use

```
String[] model.mesh(<tag>).stat().getTypes();
```

If the current selection is not the entire geometry, only elements and types in the current selection is returned. You can also use the methods

```
int model.mesh(<tag>).getNumElem(type);
int model.mesh(<tag>).getNumElem();
String[] model.mesh(<tag>).getTypes();
```

to obtain information about the entire geometry.

Quality of Elements

The absolute value of the mesh element quality is based on the ratios of the inscribed and circumscribed circles' or spheres' radii for the simplex corresponding to each corner of the element. If the simplex cannot be clearly determined (an apex of the pyramid, for example), the corresponding corner is excluded from the consideration. The absolute value is always between 0 and 1, where 0.0 represents a degenerated element and 1.0 represents the best possible element. A negative value means a contradiction to the COMSOL Multiphysics numbering convention for mesh element vertices (see [Element Numbering Conventions](#)), and the element is then referred to as an *inverted element*.

To retrieve the minimal quality, use

```
double model.mesh(<tag>).stat().getMinQuality(type);
double model.mesh(<tag>).stat().getMinQuality();
```

To retrieve the mean quality, use

```
double model.mesh(<tag>).stat().getMeanQuality(type);
double model.mesh(<tag>).stat().getMeanQuality();
```

To calculate a distribution of qualities, use the `getQualityDistr` method.

```
int[] model.mesh(<tag>).stat().getQualityDistr(type, <size>);
int[] model.mesh(<tag>).stat().getQualityDistr(<size>);
```

The size parameter is a positive integer determining how detailed the distribution is and equals the size of the output array. The distribution can be used to plot a histogram of the element quality. For example, if size equals 10, the first entry in the returned array is the number of elements with quality less than 0.1, and the last entry is the number of elements with quality better than 0.9.

The following methods are available directly on the sequence and provide statistics for the entire geometry:

```
double model.mesh(<tag>).getMinQuality(type);
double model.mesh(<tag>).getMinQuality();
double model.mesh(<tag>).getMeanQuality(type);
double model.mesh(<tag>).getMeanQuality();
int[] model.mesh(<tag>).getQualityDistr(type, <size>);
int[] model.mesh(<tag>).getQualityDistr(<size>);
```

Volume of Elements and Mesh

To determine minimum element volume, area, or length of a certain type, use the method `getMinVolume`:

```
double model.mesh(<tag>).stat().getMinVolume(type);
double model.mesh(<tag>).stat().getMinVolume();
```

To determine maximum element volume, area, or length of a certain type, use the method `getMaxVolume`:

```
double model.mesh(<tag>).stat().getMaxVolume(type);
double model.mesh(<tag>).stat().getMaxVolume();
```

To determine the volume, area, or length of the mesh, use the method `getVolume`:

```
double model.mesh(<tag>).stat().getVolume(type);
double model.mesh(<tag>).stat().getVolume();
```

The following methods are available directly on the sequence, and provide volume information about the entire geometry:

```
double model.mesh(<tag>).getMinVolume(type);
double model.mesh(<tag>).getMinVolume();
double model.mesh(<tag>).getMaxVolume(type);
double model.mesh(<tag>).getMaxVolume();
double model.mesh(<tag>).getVolume(type);
double model.mesh(<tag>).getVolume();
```

Growth Rate in Mesh

The growth rate value is a local measure greater than or equal to 1 indicating the maximum element size growth rate between two neighboring elements.

To retrieve the maximal growth rate value for a selection, use

```
double model.mesh(<tag>).stat().getMaxGrowthRate();
```

To retrieve the average growth rate for a selection, use

```
double model.mesh(<tag>).stat().getMeanGrowthRate();
```

The following methods are available directly on the sequence, and provide statistics for the entire geometry:

```
double model.mesh(<tag>).getMaxGrowthRate();
double model.mesh(<tag>).getMeanGrowthRate();
```

Mesh Status

You can check if the entire selected geometry has a mesh by calling the `isComplete` method.

```
boolean model.mesh(<tag>).stat().isComplete();
```

To check if the entire geometry is meshed, use

```
boolean model.mesh(<tag>).isComplete();
```

You can also check if the selected geometry has an empty mesh by calling the `isEmpty` method.

```
boolean model.mesh(<tag>).stat().isEmpty();
```

To check if the entire geometry has an empty mesh, use

```
boolean model.mesh(<tag>).isEmpty();
```

Getting and Setting Mesh Data

The data in the mesh object can be accessed and manipulated via *getters and setters*. You can get vertex coordinates, elements, and for each element the number of its geometric entity. The element matrix consists of indexes into the vertex list. The entity list contains the entity number of each element. There is one element matrix and one entity number list for each type.

Accessing Mesh Data

To get the number of mesh vertices, use

```
int model.mesh(<tag>).getNumVertex();
```

To get the coordinates of the mesh vertices, use

```
double[][] model.mesh(<tag>).getVertex();
```

which gives you a matrix where each column corresponds to a mesh vertex.

To get the element types in the mesh, use

```
String[] model.mesh(<tag>).getTypes();
```

The following table lists the possible types. See [Element Numbering Conventions](#) for an explanation of each type.

STRING	ELEMENT	DIMENSION	NUMBER OF NODES
vtx	Vertex element	0	1
edg	Edge element	1	2
tri	Triangular element	2	3
quad	Quadrilateral element	2	4
tet	Tetrahedral element	3	4
pyr	Pyramid element	3	5
prism	Prism element	3	6
hex	Hexahedral element	3	8

To get the number of elements of a specific type, use

```
int model.mesh(<tag>).getNumElem(type);
```

To get the elements for a specific type, use

```
int[][] model.mesh(<tag>).getElem(type);
```

which gives you a matrix where each column contains the mesh vertex indices of an element's corners.

To get the geometric entity number for the elements of a specific type, use

```
int[] model.mesh(<tag>).getElemEntity(type);
```



[Information and Statistics](#)

Setting or Modifying Mesh Data

You can modify the mesh object of a meshing sequence via the `data()` method. Using this method you access a temporary object (`MeshData`) storing mesh data. When you use the `data()` method the first time the `MeshData`

object is empty. You can fill it with mesh data by using various set methods or by transferring mesh data from the mesh of the meshing sequence. Call the method `data().createMesh` to construct a complete mesh from the `MeshData` object and store it in the meshing sequence. If the geometry is not empty, the new mesh is checked to ensure that it matches the geometry. Thus, to create an arbitrary mesh, you need to create an empty geometry sequence and a corresponding empty meshing sequence and construct the mesh on the empty meshing sequence.

To set the mesh vertices, use

```
model.mesh(<tag>).data().setVertex(double[][]);
```

where each column of the input matrix contains the coordinates of a mesh vertex.

To set the elements of a specific type, use

```
model.mesh(<tag>).data().setElem(type, int[][]);
```

where each column of input element matrix contains the mesh vertex indices of an element's corners.

If you want to specify the geometric entity number for the elements of a specific type, use

```
model.mesh(<tag>).data().setElemEntity(type, int[]);
```

The `MeshData` object has the same access methods as the meshing sequence.

```
int model.mesh(<tag>).data().getNumVertex();
double[][] model.mesh(<tag>).data().getVertex();
String[] model.mesh(<tag>).data().getTypes();
int model.mesh(<tag>).data().getNumElem(type);
int[][] model.mesh(<tag>).data().getElem(type);
int[] model.mesh(<tag>).data().getElemEntity(type);
```



Accessing Mesh Data

It is also possible to fill the `MeshData` object with mesh data from the mesh of a meshing sequence. To transfer the mesh from the current meshing sequence into the `MeshData` object, use

```
model.mesh(<tag>).data().transferMesh();
```

To transfer the mesh from another meshing sequence, specified by *mtag*, into the `MeshData` object, use

```
model.mesh(<tag>).data().transferMesh(mtag);
```

To clear the `MeshData` object, use

```
model.mesh(<tag>).data().clearData();
```

To create a complete mesh from the `MeshData` object and store it in the sequence, use

```
model.mesh(<tag>).data().createMesh();
```

This method uses several properties when creating a complete mesh from the specified mesh data. To set a property, use

```
model.mesh(<tag>).data().set(property, <value>);
```

To get a property, use

```
model.mesh(<tag>).data().getType(property);
```


The following properties are available.

TABLE 4-5: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	2D	3D	DESCRIPTION
extrangle	double	0.6 degrees		√	Maximum angle between boundary element normal and extrusion plane that causes the element to be a part the extruded face if possible
faceangle	double	360 degrees		√	Maximum angle between any two boundary elements in the same face
facecleanup	double	0.01		√	Avoid creating small faces. Faces with an area less than Facecleanup * the mean face area, are merged with adjacent faces
facecurv	double	10 degrees		√	Maximum relative angle deviation between any two boundary elements in the same face
minareaecurv	double	1		√	Minimum relative area of face to be considered as a face with constant curvature
minareaextr	double	0.05		√	Minimum relative area of face to be considered extruded
minareaeplane	double	0.005	√	√	Minimum relative area of face to be considered planar
neighangle	double	20 degrees	√	√	Maximum angle between a boundary element and a neighbor that causes the elements to be part of the same boundary domain if possible
planarangle	double	0.6 degrees	√	√	Maximum angle between boundary element normal and a neighbor that causes the element to be a part the planar face if possible

EXAMPLES OF SETTING OR MODIFYING MESH DATA

The following examples create a triangular mesh on a square, extracts the vertices and the triangles. Then the vertices are transformed and inserted into a new meshing sequence.

Code for use with Java

```

Model model = ModelUtil.create("Model");

model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

// Create a rectangle and a mesh
model.geom("geom1").create("r1", "Rectangle");
m.create("ftri1", "FreeTri");
m.run();

double[][] vtx = m.getVertex();
int[][] tri = m.getElem("tri");

// Transform x coordinates
for (int k=0; k<vtx[0].length; k++)
    vtx[0][k] *= 0.5;

// Create a new geometry and mesh
model.geom().create("geom2", 2);
MeshSequence m2 = model.mesh().create("mesh2", "geom2");

// Insert vertices and triangles and create mesh
m2.data().setElem("tri", tri);
m2.data().setVertex(vtx);
m2.data().createMesh();

```

Code for use with MATLAB

```
model = ModelUtil.create('Model');

model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

% Create a rectangle and a mesh
model.geom('geom1').create('r1', 'Rectangle');
m.create('ftri1', 'FreeTri');
m.run;

vtx = m.getVertex;
tri = m.getElem('tri');

% Transform x coordinates
vtx(1,:) = vtx(1, :)*0.5;

% Create a new geometry and mesh
model.geom.create('geom2', 2);
m2 = model.mesh.create('mesh2', 'geom2');

% Insert vertices and triangles and create mesh
m2.data.setElem('tri', tri);
m2.data.setVertex(vtx);
m2.data.createMesh;
```

Block Versions

Since the amount of available Java memory might be limited, there are block versions of the mesh setters and getters, which sets or gets a subset of the data. The getters take a *position* argument, which specifies the first item to get, and a *number* argument, which specifies the number of items to get. The setters takes only the position argument; the number of items is determined by the size of the provided data. When working with the setters, remember that it is more efficient to set the data at the last position first, since sufficient space is then allocated directly and no copying and reallocation is needed.

```
double[][] model.mesh(<tag>).getVertex(int position, int number);
int[][] model.mesh(<tag>).getElem(type, int position, int number);
int[] model.mesh(<tag>).getGeomEntity(type, int position, int number);

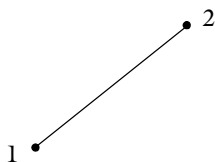
model.mesh(<tag>).data().setVertex(double[][], int position);
model.mesh(<tag>).data().setElem(type, int[][], int position);
model.mesh(<tag>).data().setGeomEntity(type, int[], int position);

double[][] model.mesh(<tag>).data().getVertex(int position, int number);
int[][] model.mesh(<tag>).data().getElem(type, int position, int number);
int[] model.mesh(<tag>).data().getGeomEntity(type, int position, int number);
```

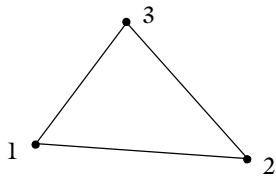
Element Numbering Conventions

The (local) numbering of the corners of an element is defined according to the following.

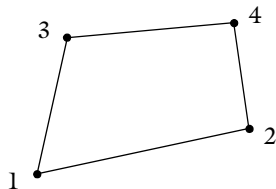
Edge element (edg):



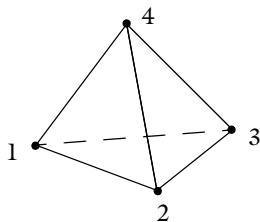
Triangular element (tri):



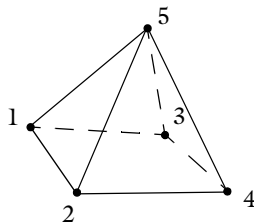
Quadrilateral element (quad):



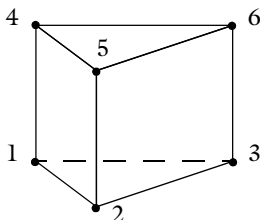
Tetrahedral element (tet):



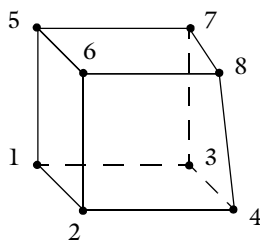
Pyramid element (pyr):



Prism element (prism):



Hexahedral element (hex):



Errors and Warnings

COMSOL treats problems encountered when building a meshing feature in two different ways depending on if it is possible to avoid the problem and continue the operation or if the operation must be stopped.

Continuing Operations

When you build `BndLayer`, `FreeTri`, `FreeQuad`, `FreeTet`, `Map`, or `Sweep` features all problems related to meshing of faces and domains are avoided by leaving the corresponding faces and domains unmeshed. The operation continues meshing the remaining entities and stores information about the encountered problems in the feature. A feature that encountered this type of problems during the build gets a warning status. However, if all the selected entities of a feature fail to mesh the feature gets an error status. If you build several features in a sequence the build is not stopped by a feature that gets a warning status.

The information on the encountered problems for a feature with warning status is contained in `MeshError` and `MeshWarning` features stored in the feature. There is one `MeshError` feature for each geometric entity that failed to be meshed and one `MeshWarning` feature for each dimension where there exists an entity that could not be meshed due to failures meshing any of its adjacent entities.

Use the `continue` property to control whether any of the previously listed meshing features should avoid encountered problems and continue to mesh or if it should stop at the first encountered problem.

Stopping Operations

When you build a feature other than any of the `BndLayer`, `FreeTri`, `FreeQuad`, `FreeTet`, `Map`, or `Sweep` features, the operation always stops if a problem is encountered. This means that no changes are made to the mesh. The feature gets an error status and if it is part of a sequence build the build stops and the preceding feature becomes the current feature. Information on the error is contained in a `MeshError` feature stored in the feature.

The MeshError Feature

A `MeshError` feature contains an error message and can be equipped with a selection defining the failed entity, or a coordinate value specifying a position related to the error. It can also be equipped with details about the error. A `MeshError` feature can have a children feature of `MeshError` type that contains low-level error information. This means that an error can be represented by a stack of `MeshError` features that reflects the stack trace of the error.

The MeshWarning Feature

One `MeshWarning` feature is generated for each dimension where there are entities that cannot be meshed due to previous errors. For example, if there are edges that fail to be meshed for a `FreeTet` feature operating on the entire geometry there is one `MeshWarning` feature for the adjacent faces and one `MeshWarning` feature for the adjacent domains.

The `MeshWarning` feature contains a selection defining the entities that could not be meshed and a message describing the cause for this.

Retrieving Problem Information

Build a geometry:

```
Model mdl = com.comsol.model.util.ModelUtil.create("Model1");
```

```
mdl.geom().create("g",3);
mdl.geom("g").create("cyl1","Cylinder").set("h",3.0);
mdl.geom("g").create("cyl2","Cylinder").set("h",3.0).set("r",0.95);
mdl.geom("g").create("co1","Difference");
mdl.geom("g").feature("co1").selection("input").set("cyl1");
mdl.geom("g").feature("co1").selection("input2").set("cyl2");
```

Build a mesh:

```
MeshSequence ms = mdl.mesh().create("m", "g");
ms.feature("size").set("hauto", 9);
ms.create("ftri1","FreeTri");
ms.feature("ftri1").selection().geom(2).set(1, 2, 7, 10);
ms.create("ftet1","FreeTet");
ms.feature("ftet1").create("ms1","Size");
ms.run();
```

Check if the mesh was built with problems.

```
boolean problem = ms.hasProblems();
```

Get the names of the features with problems. In this case the feature ftet1.

```
String[] problemNames = ms.feature().problemNames();
```

Get error information.

```
String[] errorNames = ms.feature(problemNames[0]).problem().
errorNames();
for (String errorName : errorNames) {
    int errorDepth = ms.feature(problemNames[0]).
    problem(errorName).depth();
    for (int i=0; i<errorDepth; ++i) {
        MeshFeature errorFeature = ms.feature(problemNames[0]).problem(errorName).get(i);
        String errorMessage = errorFeature.getString("message");
        MeshSelection sel = null;
        if (errorFeature.hasSelection())
            sel = errorFeature.selection();
    }
}
```

Get warning information.

```
String[] warningNames = ms.feature(problemNames[0]).problem().warningNames();
MeshFeature warningFeature = ms.feature(problemNames[0]).problem(warningNames[0]);
String warningMessage = warningFeature.getString("message");
MeshSelection sel = warningFeature.selection();
```

Exporting Meshes to Files

Exporting Mesh to a File

To export a mesh to a file, enter

```
model.mesh(<tag>).export(<filename>);
```

where *<filename>* is a string. The file can be any of the following formats.

TABLE 4-6: VALID FILE FORMATS

FILE FORMAT	NOTE	FILE EXTENSIONS
COMSOL Multiphysics Binary		.mphbin
COMSOL Multiphysics Text		.mphtxt
NASTRAN file		.nas, .bdf, .nastran, .dat
STL Binary (3D)	1	.stl
STL Text (3D)	1	.stl

¹ Use `model.mesh(<tag>).export().set("stlformat",<format>)` to specify the STL file format ("binary" or "text")

Exporting Mesh to a COMSOL Multiphysics File

To specify the dimensions of the elements to export or to choose to include or exclude the geometric entity information, enter

```
model.mesh(<tag>).export().set(<property>,<value>);
```

The following table lists the available properties:

TABLE 4-7: AVAILABLE COMSOL MESH EXPORT PROPERTIES AND THEIR VALID VALUES.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
domelem	on off	on	Specify if domain elements are exported.
bndelem	on off	on	Specify if boundary elements are exported.
edgelem	on off	on	Specify if edge elements are exported (3D only).
vtxelem	on off	on	Specify if vertex elements are exported (2D and 3D only).
geominfo	on off	on	Specify if geometric entity information for each element is exported.

Exporting Mesh to a NASTRAN® File

To specify the dimensions of the elements to export, to choose to include or exclude the geometric entity information, or to set the file field format or element order, enter

```
model.mesh(<tag>).export().set(<property>,<value>);
```

The following table lists the available properties:

TABLE 4-8: AVAILABLE NASTRAN MESH EXPORT PROPERTIES AND THEIR VALID VALUES.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solidelem	on off	on	Specify if domain elements are exported.
shellelem	on off	off	Specify if boundary elements are exported (3D only).
geominfo	on off	on	Specify if geometric entity information for each element is exported.

TABLE 4-8: AVAILABLE NASTRAN MESH EXPORT PROPERTIES AND THEIR VALID VALUES.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
fieldformat	small large free	small	Specify file field format.
linear	on off	off	Specify if elements are exported as linear or second-order (quadratic) elements.

Mesh Commands

The following list includes the available commands for creating and modifying meshes (listed in alphabetical order):

- [Ball](#)
- [BndLayer](#)
- [BndLayerProp](#)
- [Box](#)
- [Convert](#)
- [CopyEdge](#)
- [CopyFace](#)
- [CopyDomain](#)
- [Copy](#)
- [CornerRefinement](#)
- [CreateVertex](#)
- [Cylinder](#)
- [Delete](#)
- [DeleteEntities](#)
- [DetectFaces](#)
- [Distribution](#)
- [Edge](#)
- [EdgeGroup](#)
- [EdgeMap](#)
- [FreeQuad](#)
- [FreeTet](#)
- [FreeTri](#)
- [Import](#)
- [JoinEntities](#)
- [LogicalExpression](#)
- [Map](#)
- [OnePointMap](#)
- [Point](#)
- [Reference](#)
- [Refine](#)
- [Scale](#)
- [Size](#)
- [Sweep](#)
- [TwoPointMap](#)

Ball

Split geometric entities of an imported mesh by a ball.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Ball");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Ball")` to split geometric entities of an imported 2D or 3D mesh by an element set defined by a ball.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify geometric entities to split. If you do not specify the selection, the feature operates on the entire geometry.

The following properties are available:

TABLE 4-9: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
posx	double	0	Center, first coordinate
posy	double	0	Center, second coordinate
posz	double	0	Center, third coordinate

TABLE 4-9: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
r	double	1	Radius
condition	allvertices somevertex	allvertices	Condition for inclusion of an element

SEE ALSO

[Import](#), [Box](#), [Cylinder](#), [DetectFaces](#), [LogicalExpression](#)

BndLayer

Create a boundary layer mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"BndLayer");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"BndLayerProp");
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"BndLayer")` to create a boundary layer mesh in 2D or 3D.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain selection. If you do not specify the selection, the feature operates on all domains.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"BndLayerProp")` to add a `BndLayerProp` attribute feature defining the locations and properties of the boundary layers.

The feature reads properties from the `BndLayerProp` attribute feature.

The following properties are available:

TABLE 4-10: AVAILABLE PROPERTIES FOR BNDLAYER

PROPERTY	VALUE	DEFAULT	DESCRIPTION
continue	on off	on	When enabled, the mesher does not stop if there is an error
sharpcorners	none split trim	split	Specifies the handling of sharp corners in 2D and sharp edges in 3D.
splitangle	double	240[deg]	Minimum angle between boundary layer boundaries to introduce a boundary layer split.
splitdivangle	double	100[deg]	Maximum angle per split
trimmaxangle	double	50[deg]	Maximum angle between boundary layer boundaries for boundary layer trimming.
trimminangle	double	240[deg]	Minimum angle between boundary layer boundaries for boundary layer trimming.
layerdec	integer	2	Maximum difference in number of boundary layers between neighboring points on boundary layer boundaries.
smoothtransition	on off	on	Specifies if the operation smooths the transition to interior mesh.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	16 in 2D, 6 in 3D	Specifies the maximum element smoothing depth.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshbndlayer`, have been removed:

TABLE 4-11: REMOVED PROPERTIES

PROPERTY	REASON
<code>blbnd</code>	Defined by the selection of <code>BndLayerProp</code>
<code>hauto</code>	Input mesh provided by meshing sequence
<code>mcase</code>	Defined by meshing sequence
<code>meshstart</code>	Input mesh provided by meshing sequence
<code>out</code>	Output mesh provided by meshing sequence
<code>subdomain</code>	Defined by the selection

EXAMPLES

Insert boundary layers to an existing mesh containing both quadrilateral elements and triangular elements.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("sq1", "Square");
g.create("sq2", "Square");
g.feature("sq2").setIndex("pos", "1", 0);
g.create("c1", "Circle");
g.feature("c1").set("r", "0.2");
g.feature("c1").setIndex("pos", "1.5", 0);
g.feature("c1").setIndex("pos", "0.5", 1);
g.create("co1", "Compose");
g.feature("co1").selection("input").init().set(new String[]{"c1", "sq1", "sq2"});
g.feature("co1").set("formula", "sq1+sq2-c1");
g.run();

m.create("map1", "Map");
m.feature("map1").selection().geom("geom1", 2);
m.feature("map1").selection().set(new int[]{1});
m.create("ftri1", "FreeTri");
m.create("bl1", "BndLayer");
m.feature("bl1").create("blp", "BndLayerProp");
m.feature("bl1").feature("blp").selection().set(new int[]{2, 3, 5, 6, 8, 9, 10, 11});
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

g.create('sq1', 'Square');
g.create('sq2', 'Square');
g.feature('sq2').setIndex('pos', '1', 0);
g.create('c1', 'Circle');
g.feature('c1').set('r', '0.2');
g.feature('c1').setIndex('pos', '1.5', 0);
g.feature('c1').setIndex('pos', '0.5', 1);
g.create('co1', 'Compose');
g.feature('co1').selection('input').init.set({'c1', 'sq1', 'sq2'});
g.feature('co1').set('formula', 'sq1+sq2-c1');
g.run;

m.create('map1', 'Map');
```

```

m.feature('map1').selection.geom('geom1', 2);
m.feature('map1').selection.set(1);
m.create('ftri1', 'FreeTri');
m.create('bl1', 'BndLayer');
m.feature('bl1').create('blp', 'BndLayerProp');
m.feature('bl1').feature('blp').selection.set([2, 3, 5, 6, 8, 9, 10, 11]);
m.run;

```

Create a boundary layer mesh consisting of prism elements along the boundary layer boundaries and tetrahedral elements in the interior:

Code for use with Java

```

Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.feature("blk1").setIndex("size", "10", 0);
g.feature("blk1").setIndex("size", "5", 1);
g.feature("blk1").setIndex("size", "5", 2);
g.create("sph1", "Sphere");
g.feature("sph1").setIndex("pos", "3", 0);
g.feature("sph1").setIndex("pos", "2.5", 1);
g.feature("sph1").setIndex("pos", "2.5", 2);
g.create("dif1", "Difference");
g.feature("dif1").selection("input").init().set(new String[]{"blk1"});
g.feature("dif1").selection("input2").init().set(new String[]{"sph1"});
g.run();

m.create("ftet1", "FreeTet");
m.create("bl1", "BndLayer");
m.feature("bl1").create("blp", "BndLayerProp");
m.feature("bl1").feature("blp").selection().
    set(new int[]{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13});
m.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.feature('blk1').setIndex('size', '10', 0);
g.feature('blk1').setIndex('size', '5', 1);
g.feature('blk1').setIndex('size', '5', 2);
g.create('sph1', 'Sphere');
g.feature('sph1').setIndex('pos', '3', 0);
g.feature('sph1').setIndex('pos', '2.5', 1);
g.feature('sph1').setIndex('pos', '2.5', 2);
g.create('dif1', 'Difference');
g.feature('dif1').selection('input').init.set({'blk1'});
g.feature('dif1').selection('input2').init.set({'sph1'});
g.run;

m.create('ftet1', 'FreeTet');
m.create('bl1', 'BndLayer');
m.feature('bl1').create('blp', 'BndLayerProp');
m.feature('bl1').feature('blp').selection.set(2:13);
m.run;

```

SEE ALSO

[BndLayerProp](#), [Map](#), [Sweep](#)

BndLayerProp

Set the boundary layer meshing properties.

SYNTAX

```
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"BndLayerProp");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"BndLayerProp")` to define boundary layer properties for the `BndLayer` feature `<ftag>`.

Use `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the boundary selection. If you do not specify the selection, it is empty.

The following properties are available:

TABLE 4-12: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
blhminfact	string double	1	Factor used to adjust the default thickness
blhmin	string double		Manual initial boundary layer thickness
blnlayers	string integer	8	Number of boundary layers
blstretch	string double	1.2	Boundary layer stretching factor
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
inittype	blhminfact blhmin	blhminfact	Selects whether blhmin or blhminfact is used.

Use the properties `blhmin`, `blstretch`, and `blnlayers` to specify the distribution of the boundary layers. `blhmin` specifies the thickness of the initial boundary layer, `blstretch` a stretching factor, and `blnlayers` the number of boundary layers. This means that the thickness of the m th boundary layer ($m=1$ to `blnlayers`) is `blstretch(m-1)blhmin`. The number of boundary layers and the thickness of the boundary layers might be automatically reduced in thin regions.

It is also possible to specify the thickness of the initial layer by using the `blhminfact` property. Then, the thickness of the first layer is $1/20$ of the local domain element height. Use the `blhminfact` property to specify a scaling factor that multiplies this default size.

The property `inittype` determines which of `blhminfact` or `blhmin` that is used. You do not need to set it explicitly because the feature automatically changes it when you set one of `blhmin` or `blhminfact`.

The values of `blhmin`, `blhminfact`, and `blstretch` are positive real scalars, or strings that evaluate to positive real scalars, given the evaluation context provided by the property `const`.

The value of `blnlayers` is a positive integer scalar, or a string that evaluates to a positive integer, given the evaluation context provided by the property `const`.

SEE ALSO

[BndLayer](#), [Scale](#), [Size](#)

Box

Split geometric entities of an imported mesh by a box.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Box");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Box")` to split geometric entities of an imported 2D or 3D mesh by an element set defined by a box.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify geometric entities to split. If you do not specify the selection, the feature operates on the entire geometry.

The following properties are available:

TABLE 4-13: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
xmin	double	- inf	Minimum x-coordinate of box
xmax	double	inf	Maximum x-coordinate of box
ymin	double	- inf	Minimum y-coordinate of box
ymax	double	inf	Maximum y-coordinate of box
zmin	double	- inf	Minimum z-coordinate of box
zmax	double	inf	Maximum z-coordinate of box
condition	allvertices somevertex	allvertices	Condition for inclusion of an element

SEE ALSO

[Import](#), [Ball](#), [Cylinder](#), [DetectFaces](#), [LogicalExpression](#)

Convert

Convert a mesh to a simplex mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Convert");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Convert")` to convert non-simplex elements in a mesh to simplex elements, that is, triangles and tetrahedra.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain or face selection. If you do not specify the selection the feature converts all quadrilateral, pyramidal, prismatic, and hexahedral elements in the mesh.

The following properties are available:

TABLE 4-14: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
splitmethod	diagonal center	diagonal	Split method for quadrilateral and hexahedral elements

Use the property `splitmethod` to specify how to split quadrilateral and hexahedral elements into triangular and tetrahedral elements, respectively. Use the `diagonal` option to split each quadrilateral element into two triangular elements and each hexahedral element into five tetrahedral element. Use the `center` option to split each

quadrilateral element into four triangular elements and each hexahedral element into 28 tetrahedral elements. The conversion also affects quadrilateral elements on the boundaries of the specified domains in 3D, which are converted into two triangular elements (when the option `diagonal` is used) or four triangular elements (when the option `center` is used).

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshconvert`, have been removed:

TABLE 4-15: REMOVED PROPERTIES

PROPERTY	REASON
face	Defined by the selection
out	Output mesh provided by meshing sequence
domain	Defined by the selection

EXAMPLES

Create a mapped quad mesh on a unit rectangle and convert each quadrilateral element into four triangular elements:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("r1", "Rectangle");
g.run();

m.create("map1", "Map");
m.create("conv1", "Convert");
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

g.create('r1', 'Rectangle');
g.run;

m.create('map1', 'Map');
m.create('conv1', 'Convert');
m.run;
```

Create a prism mesh and convert each prism into three tetrahedral elements.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("ftri1", "FreeTri");
m.feature("ftri1").selection().set(new int[]{1});
m.create("swe1", "Sweep");
m.create("conv1", "Convert");
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run;

m.create('ftri1', 'FreeTri');
m.feature('ftri1').selection().set(1);
m.create('swe1', 'Sweep');
m.create('conv1', 'Convert');
m.run;
```

SEE ALSO

[BndLayer](#), [Map](#), [Refine](#), [Sweep](#)

CopyEdge

Copy an edge mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>, "CopyEdge");
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "CopyEdge")` to copy mesh between edges in a 2D or 3D geometry.

The following properties are available:

TABLE 4-16: AVAILABLE PROPERTIES FOR COPYEDGE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
direction	auto same opposite	auto	Direction of the copied mesh
copymethod	auto singlecopy arraycopy	auto	Type of copy operation
source	Selection	Empty	Source edges
destination	Selection	Empty	Destination edges
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

Use the properties `source` and `destination` to specify the source and destination edges. The `copymethod` property determines if many-to-one or many-to-many copying is used. The `direction` property controls the orientation of the copied mesh, and is relative the direction of the source edge with smallest number and the direction of the destination edge.

Copying a mesh is only possible if the destination edge is not adjacent to a meshed domain. The copy feature overwrites any existing mesh on the destination edge.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshcopy`, have been removed:

TABLE 4-17: REMOVED PROPERTIES

PROPERTY	REASON
<code>mcase</code>	Defined by the meshing sequence

EXAMPLE

Mesh Edge 1 and copy the mesh to Edges 2, 3, and 4.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("sq1", "Square");
g.run();

m.create("edg1", "Edge");
m.feature("edg1").selection().set(new int[]{1});
m.create("cpe1", "CopyEdge");
m.feature("cpe1").selection("source").set(new int[]{1});
m.feature("cpe1").selection("destination").set(new int[]{2, 3, 4});
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

g.create('sq1', 'Square');
g.run;

m.create('edg1', 'Edge');
m.feature('edg1').selection().set(1);
m.create('cpe1', 'CopyEdge');
m.feature('cpe1').selection('source').set(1);
m.feature('cpe1').selection('destination').set(2:4);
m.run;
```

SEE ALSO

[CopyFace](#), [CopyDomain](#), [Copy](#)

CopyFace

Copy a face mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>, "CopyFace");
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, maptype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "CopyFace")` to copy mesh between faces in a 3D geometry.

If you want to specify the orientation of the source mesh on the destination, use

`model.mesh(<tag>).feature(<ftag>).create(<ftag1>, maptype)` to add an `EdgeMap`, a `OnePointMap`, or a `TwoPointMap` attribute feature.

The following properties are available:

TABLE 4-18: AVAILABLE PROPERTIES FOR COPYFACE

PROPERTY	VALUE	DEFAULT	DESCRIPTION
copymethod	auto singlecopy arraycopy	auto	Type of copy operation
source	Selection	Empty	Source boundaries
destination	Selection	Empty	Destination boundary
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

Use the properties `source` and `destination` to specify the source and destination boundaries. The `copymethod` property determines if many-to-one or many-to-many copying is used.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshcopy`, have been removed:

TABLE 4-19: REMOVED PROPERTIES

PROPERTY	REASON
direction	Defined by the attribute feature <code>EdgeMap</code>
mcase	Defined by the meshing sequence
sourceedg	Defined by the attribute feature <code>EdgeMap</code>
targetedg	Defined by the attribute feature <code>EdgeMap</code>

EXAMPLE

Mesh Face 1 of a block and copy the mesh to the opposite Face 6.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m= model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("ftri1", "FreeTri");
m.feature("ftri1").selection().set(new int[]{1});
m.create("cpf1", "CopyFace");
m.feature("cpf1").selection("source").set(new int[]{1});
m.feature("cpf1").selection("destination").set(new int[]{6});
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run;

m.create('ftri1', 'FreeTri');
m.feature('ftri1').selection().set(1);
m.create('cpf1', 'CopyFace');
```

```

m.feature('cpf1').selection('source').set(1);
m.feature('cpf1').selection('destination').set(6);
m.run;

```

SEE ALSO

[CopyEdge](#), [CopyDomain](#), [Copy](#), [EdgeMap](#), [OnePointMap](#), [TwoPointMap](#)

CopyDomain

Copy a domain mesh.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "CopyDomain");
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, maptype);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "CopyDomain")` to copy mesh between domains in a 2D or 3D geometry.

If you want to specify the orientation of the source mesh on the destination, use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, maptype)` to add an `EdgeMap`, a `OnePointMap`, or a `TwoPointMap` attribute feature.

The following properties are available:

TABLE 4-20: AVAILABLE PROPERTIES FOR COPYDOMAIN

PROPERTY	VALUE	DEFAULT	DESCRIPTION
copymethod	auto singlecopy arraycopy	auto	Type of copy operation
source	Selection	Empty	Source domains
destination	Selection	Empty	Destination domains
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

Use the properties `source` and `destination` to specify the source and destination boundaries. The `copymethod` property determines if many-to-one or many-to-many copying is used.

EXAMPLE

Mesh block 1 and copy the mesh to the block 2.

Code for use with Java

```

Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.create("blk2", "Block");
g.feature("blk2").setIndex("pos", "2", 0);
g.run();

m.create("ftet1", "FreeTet");
m.feature("ftet1").selection().set(new int[]{1});

```

```

m.create("cpd1", "CopyDomain");
m.feature("cpd1").selection("source").set(new int[]{1});
m.feature("cpd1").selection("destination").set(new int[]{2});
m.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.create('blk2', 'Block');
g.feature('blk2').setIndex('pos', '2', 0);
g.run;

m.create('ftet1', 'FreeTet');
m.feature('ftet1').selection().set(1);
m.create('cpd1', 'CopyDomain');
m.feature('cpd1').selection('source').set(1);
m.feature('cpd1').selection('destination').set(2);
m.run;

```

SEE ALSO

[CopyEdge](#), [CopyFace](#), [Copy](#), [EdgeMap](#), [OnePointMap](#), [TwoPointMap](#)

Copy

Copy a mesh.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "Copy");
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, maptype);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "Copy")` to copy a mesh between mesh sequences. Any mesh sequence can be used as the source of the operation, whereas the destination sequence cannot contain an imported mesh. The dimension of the source sequence must be less than or equal to the dimension of the destination sequence.

The following properties are available:

TABLE 4-21: AVAILABLE PROPERTIES FOR COPY

PROPERTY	VALUE	DEFAULT	DESCRIPTION
mesh	String	Native sequence	Specifies the source mesh.
copymethod	auto singlecopy arraycopy	auto	Type of copy operation.
dimension	1, 2, or 3 (in 3D)	2 in 2D, 3 in 3D	Specifies the dimension for the operation.
source	Selection	Empty	Specifies the selection of source entities.
destination	Selection	Empty	Specifies the selection of destination entities.
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.

TABLE 4-21: AVAILABLE PROPERTIES FOR COPY

PROPERTY	VALUE	DEFAULT	DESCRIPTION
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

Use the properties `source` and `destination` to specify the geometric entities of the source and destination. The `copymethod` property determines if many-to-one or many-to-many copying is used.

EXAMPLE

The following example shows how to use the Copy feature with a modified geometry from an imported mesh,

Code for use with Java

```
Model model = ModelUtil.create("Model");
model.modelNode().create("comp1");
GeomSequence geom1 = model.geom().create("geom1", 2);
geom1.create("c1", "Circle");
MeshSequence mesh1 = model.mesh().create("mesh1", "geom1");
mesh1.run();

model.modelNode().create("comp2");
GeomSequence geom2 = model.geom().create("geom2", 2);
GeomFeature imp1 = geom2.create("imp1", "Import");
imp1.set("type", "mesh");
imp1.set("mesh", "mesh1");
GeomFeature r1 = geom2.create("r1", "Rectangle");
r1.set("size", new String[]{"3", "3"});
r1.set("base", "center");

MeshSequence mesh2 = model.mesh().create("mesh2", "geom2");
MeshFeature copy1 = mesh2.create("copy1", "Copy");
copy1.set("mesh", "mesh1");
copy1.set("dimension", 2);
copy1.selection("source").set(1);
copy1.selection("destination").set(2);
mesh2.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
model.modelNode.create('comp1');
geom1 = model.geom.create('geom1', 2);
geom1.create('c1', 'Circle');
mesh1 = model.mesh.create('mesh1', 'geom1');
mesh1.run;

model.modelNode.create('comp2');
geom2 = model.geom.create('geom2', 2);
imp1 = geom2.create('imp1', 'Import');
imp1.set('type', 'mesh');
imp1.set('mesh', 'mesh1');
r1 = geom2.create('r1', 'Rectangle');
r1.set('size', {'3', '3'});
r1.set('base', 'center');

mesh2 = model.mesh.create('mesh2', 'geom2');
copy1 = mesh2.create('copy1', 'Copy');
copy1.set('mesh', 'mesh1');
copy1.set('dimension', 2);
copy1.selection('source').set(1);
copy1.selection('destination').set(2);
mesh2.run;
```

SEE ALSO

[CopyEdge](#), [CopyFace](#), [CopyDomain](#)

CornerRefinement

Decrease element size at sharp corners.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"CornerRefinement");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"CornerRefinement");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection(property);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"CornerRefinement")` to decrease the element size defined in the sequence at vertices in 2D and edges in 3D that define a sharp corner. Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"CornerRefinement")` to decrease the element size for the feature `<ftag>` that can be any of the types `Edge`, `FreeQuad`, `FreeTri`, or `FreeTet`.

Use `model.mesh(<tag>).feature(<ftag>).selection()` or `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the domain selection. If you do not specify any selection the feature is defined on the entire geometry.

The following properties are available:

TABLE 4-22: VALID PROPERTIES

NAME	VALUE	DEFAULT	DESCRIPTION
boundary	Selection		Boundary selection
minangle	double	240 [deg]	Minimum angle between two boundaries (sharp corner) for decreasing the element size.
refinement	double	0.25 in 2D, 0.35 in 3D	Factor multiplying the element size at sharp corners. The valid range is [0 1].

CreateVertex

Create an additional vertex in an imported mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"CreateVertex");
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"CreateVertex")` to create an additional vertex in the mesh point closest to a specified position.

The following properties are available:

TABLE 4-23: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
x	double	0	x-coordinate
y	double	0	y-coordinate
z	double	0	z-coordinate

SEE ALSO

[Import](#)

Cylinder

Split geometric entities of an imported mesh by a cylinder.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Cylinder");  
model.mesh(<tag>).feature(<ftag>).selection();  
model.mesh(<tag>).feature(<ftag>).set(property,<value>);  
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Cylinder")` to split geometric entities of an imported 3D mesh by an element set defined by a cylinder.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify geometric entities to split. If you do not specify the selection, the feature operates on the entire geometry.

The following properties are available:

TABLE 4-24: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
r	double	1	Radius of the cylinder
top	double	inf	Coordinate of upper boundary circle in local coordinate system
bottom	double	-inf	Coordinate of lower boundary circle in local coordinate system
pos	double[]	{0,0,0}	Position of the cylinder
axistype	x y z Cartesian spherical	z	Coordinate system used for axis. The value is synchronized with axis.
axis	double[]	{0,0,1}	Direction of the axis. Vector has length 3 if axistype is cartesian, and length 2 if axistype is spherical.
condition	allvertices somevertex	allvertices	Condition for inclusion of an element

SEE ALSO

[Import](#), [Ball](#), [Box](#), [DetectFaces](#), [LogicalExpression](#)

Delete

Delete elements from mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Delete");  
model.mesh(<tag>).feature(<ftag>).selection();  
model.mesh(<tag>).feature(<ftag>).set(property,<value>);  
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Delete")` to delete elements from the mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the geometric entity selection. Allowed selections are meshed geometric entities of any dimension and the entire geometry, which can be partially meshed.

The following properties are available:

TABLE 4-25: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
deladj	on off	on	Specifies if elements belonging to adjacent domains of lower dimensions are deleted as well

Deleting elements corresponding to a specific domain, all elements on adjacent domains of higher dimension are deleted as well.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshdel`, have been removed:

TABLE 4-26: REMOVED PROPERTIES

PROPERTY	REASON
out	Output mesh provided by meshing sequence
edge	Defined by the selection
face	Defined by the selection
point	Defined by the selection
domain	Defined by the selection

EXAMPLES

Create a mesh of a 2D geometry with 3 domains. First, delete the elements belonging to Domain 3 only. Then delete the elements belonging to Domain 1 and all adjacent domains of lower dimensions that can be deleted. At last, delete the edge elements belonging to Edge 1. The elements belonging to the adjacent domain (Domain 2) are deleted as well.

Code for use with Java

```
Model model = ModelUtil.create("Model");  
model.modelNode().create("mod1");  
GeomSequence g = model.geom().create("geom1", 2);  
MeshSequence m = model.mesh().create("mesh1", "geom1");  
  
g.create("R1", "Rectangle");  
g.create("C1", "Circle");  
g.run();  
  
m.create("ftri1", "FreeTri");  
  
m.create("del1", "Delete");  
m.feature("del1").selection().geom(2).set(3);  
m.feature("del1").set("deladj", "off");  
  
m.create("del2", "Delete");
```

```

m.feature("del2").selection().geom(2).set(1);
m.feature("del2").set("deladj", "on");

m.create("del3", "Delete");
m.feature("del3").selection().geom(1).set(1);

m.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
model.modelNode.create('mod1');
g = model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

g.create('R1', 'Rectangle');
g.create('C1', 'Circle');
g.run;

m.create('ftri1', 'FreeTri');

m.create('del1', 'Delete');
m.feature('del1').selection().geom(2).set(3);
m.feature('del1').set('deladj', 'off');

m.create('del2', 'Delete');
m.feature('del2').selection().geom(2).set(1);
m.feature('del2').set('deladj', 'on');

m.create('del3', 'Delete');
m.feature('del3').selection().geom(1).set(1);

m.run;

```

SEE ALSO

[FreeTet](#), [Map](#), [Sweep](#)

DeleteEntities

Delete geometric entities from an imported mesh.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "DeleteEntities");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "DeleteEntities")` to delete geometric entities from an imported 2D or 3D mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify geometric entities to delete.

The following properties are available:

TABLE 4-27: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
deleteadj	Boolean	true	Specifies if the operation removes lower dimensional adjacent entities

SEE ALSO

[Import](#), [JoinEntities](#)

DetectFaces

Split geometric boundary entities by detecting faces in the mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"DetectFaces");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"DetectFaces")` to split geometric boundary entities of an imported 3D mesh by detecting shapes in the mesh that are likely to constitute faces.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the boundary entities to split. If you do not specify the selection, it is left empty.

The following properties are available:

TABLE 4-28: VALID PROPERTY/VALUE PAIRS FOR DETECTFACES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
detectadjfillets	on off	on	Whether to detect cylindrical faces adjacent to the detected planar faces.
detectfacesplanar	on off	on	Whether to detect planar faces.
facemaxangle	double	40 degrees	Maximum tolerated angle between neighboring boundary elements in the same face.
planarfacemaxangle	double	0.6 degrees	Maximum tolerated angle between neighboring boundary elements in the same planar face.
planarfaceminareafraction	double	0.005	Minimum relative area for a planar face to be created.

SEE ALSO

[Import](#), [Ball](#), [Box](#), [Cylinder](#), [LogicalExpression](#)

Distribution

Mesh element distribution properties.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Distribution");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"Distribution");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Distribution")` to specify element distribution properties in the sequence. Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"Distribution")` to specify element distribution properties for the feature `<ftag>` that can be any of the types `Edge`, `FreeQuad`, `FreeTri`, `FreeTet`, `Map`, or `Sweep`.

Use `model.mesh(<tag>).feature(<ftag>).selection()` or `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the edge or domain selection.

You can specify a mesh element distribution in three different ways; by specifying the number of elements only, by specifying the number of elements together with properties determining the distribution of the elements, or by specifying the element distribution explicitly. The property `type` determines which of the three alternatives you want to use. However, you need not set `type` manually since it is automatically updated when you set a property from one of the three groups below.

The following group of properties are available:

TABLE 4-29: AVAILABLE PROPERTIES WHEN TYPE IS NUMBER

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>numelem</code>	<code>integer</code>		Number of elements

Use the property `numelem` to specify the number of elements, but let the algorithm determine a suitable distribution, taking geometry and surrounding mesh into account.

TABLE 4-30: AVAILABLE PROPERTIES WHEN TYPE IS EXPLICIT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>explicit</code>	<code>double[]</code>		Explicitly defined element distribution

Use the `explicit` property to specify an explicit element distribution. The value of this property is an array with increasing values starting at 0.

TABLE 4-31: AVAILABLE PROPERTIES WHEN TYPE IS PREDEFINED

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>elemcount</code>	<code>integer</code>		Number of elements
<code>elemratio</code>	<code>double</code>	1	Specifies the ratio in size between the last element and first element along the edge
<code>method</code>	<code>arithmetic geometric</code>	<code>arithmetic</code>	Specifies if the element distribution is an arithmetic or a geometric sequence
<code>reverse</code>	<code>on off</code>	<code>off</code>	Specifies if the distribution is defined in the opposite edge direction for the edge in the selection with lowest index
<code>symmetric</code>	<code>on off</code>	<code>off</code>	Specifies if the distribution is made symmetric

When the type is predefined, the distribution is calculated from the parameters given above.

This Distribution feature can be assigned to edges in 2D and 3D, domains in 3D or the entire geometry. The `FreeTet`, `FreeTri`, `FreeQuad`, `Edge`, and `Map` features use this property when defined on edges, the `Sweep` feature uses this property defined on domains.

SEE ALSO

[Scale](#), [Size](#)

Edge

Create an edge mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Edge");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Edge")` to create an edge mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the edge selection. If you do not specify a selection, the feature creates a mesh on the remaining entities in 1D. In 3D and 2D, the default selection is empty.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype)` to add Size or Distribution attribute features.

The following properties are available:

TABLE 4-32: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

SEE ALSO

[Distribution](#), [Point](#), [Size](#)

EdgeGroup

Define edge groups for mapped meshes.

SYNTAX

```
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"EdgeGroup");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection(property);
```

DESCRIPTION

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"EdgeGroup")` to define edge groups for the Map feature `<ftag>`.

Use `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the domain.

The following properties are available:

TABLE 4-33: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
edge1	Selection		First group of edges
edge2	Selection		Second group of edge
edge3	Selection		Third group of edge
edge4	Selection		Fourth group of edge

The value of each property is an edge selection that combines edges to defines a logical side of the corresponding domain (in 2D) or boundary (in 3D). No specific ordering of the edges is required.

SEE ALSO

[Distribution](#), [Map](#), [Size](#)

EdgeMap

Specify an edge map for a face copy or a domain copy operation.

SYNTAX

```
model.mesh(<tag>).create(<ftag1>, "EdgeMap");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection(property);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, "EdgeMap")` to define an edge mapping for `CopyFace` or `CopyDomain` feature `<ftag>`.

The following properties are available:

TABLE 4-34: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
direction	auto same opposite	auto	The direction of dstedge relative srcedge.
dstedge	Selection		Edge on destination face/domain
srcedge	Selection		Edge on source face/domain

Use the `EdgeMap` feature if you need to control how the source and destination faces/domains are matched in a copy face or a copy domain mesh operation. When this feature is present, the source mesh is transformed so that `srcedge` is mapped onto `dstedge`. The relative orientation of the edges is specified by the `direction` property.

EXAMPLE

Create a block and mesh Face 1 with a fine mesh on Edge 1. Copy this mesh to face 6 and ensure that the fine mesh of Edge 1 ends up on Edge 12.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("ftri1", "FreeTri");
m.feature("ftri1").selection().set(new int[]{1});
m.feature("ftri1").create("size1", "Size");
m.feature("ftri1").feature("size1").selection().geom("geom1", 1).set(new int[]{1});
m.feature("ftri1").feature("size1").set("hmax", "0.01");

m.create("cpf1", "CopyFace");
m.feature("cpf1").selection("source").set(new int[]{1});
m.feature("cpf1").selection("destination").set(new int[]{6});
m.feature("cpf1").create("em1", "EdgeMap");
m.feature("cpf1").feature("em1").selection("dstedge").set(new int[]{1});
m.feature("cpf1").feature("em1").selection("dstedge").set(new int[]{12});

m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
```

```

g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run;

m.create('ftri1', 'FreeTri');
m.feature('ftri1').selection().set(1);
m.feature('ftri1').create('size1', 'Size');
m.feature('ftri1').feature('size1').selection().geom('geom1', 1).set(1);
m.feature('ftri1').feature('size1').set('hmax', '0.01');

m.create('cpf1', 'CopyFace');
m.feature('cpf1').selection('source').set(1);
m.feature('cpf1').selection('destination').set(6);
m.feature('cpf1').create('em1', 'EdgeMap');
m.feature('cpf1').feature('em1').selection('dstedge').set(1);
m.feature('cpf1').feature('em1').selection('dstedge').set(12);

m.run;

```

SEE ALSO

[CopyFace](#), [CopyDomain](#), [OnePointMap](#), [TwoPointMap](#)

FreeQuad

Create an unstructured quadrilateral mesh.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "FreeQuad");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "FreeQuad")` to create an unstructured quadrilateral mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain (boundary in 3D) selection. If you do not specify any selection the feature creates a mesh on the remaining geometric entities in 2D. In 3D, the default selection is empty.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype)` to add Size or Distribution attribute features.

The following properties are available:

TABLE 4-35: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
continue	on off	on	When enabled, the mesher does not stop if there is an error
xscale	double	1	Scale geometry in x direction before meshing
yscale	double	1	Scale geometry in y direction before meshing
zscale	double	1	Scale geometry in z direction before meshing
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth

See the [FreeTet](#) feature for more information on the properties.

This feature uses the same attribute feature as the [FreeTet](#) feature.

COMPATIBILITY

See [FreeTet](#).

SEE ALSO

[Distribution](#), [FreeTri](#), [Size](#)

FreeTet

Create a free tetrahedral mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"FreeTet");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"FreeTet")` to create an unstructured tetrahedral mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain selection. If you do not specify any selection the feature creates a mesh on the remaining geometric entities.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype)` to add [Size](#) or [Distribution](#) attribute features.

The following properties are available:

TABLE 4-36: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
continue	on off	on	When enabled, the mesher does not stop if there is an error.
method	auto del1 del2	auto	Delaunay algorithm to use.
xscale	double	1	Scale geometry in x direction before meshing.
yscale	double	1	Scale geometry in y direction before meshing.
zscale	double	1	Scale geometry in z direction before meshing.
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations.
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth.

If `continue` is on, the mesher does not stop if it encounters an error. Instead, it continues to mesh remaining entities. Before finishing, all errors are collected and reported as feature problems. You can use the output to visually examine the partial mesh; this can help you understand what the problems are and how they can be fixed.

The `method` property determines the Delaunay tessellation algorithm to use. The default is `auto`, which makes the [FreeTet](#) mesh generator determine the best algorithm for each domain. The `del1` Delaunay algorithm is the Delaunay algorithm used in earlier COMSOL versions. The `del2` Delaunay algorithm is an alternative version of the algorithm, which under some conditions can modify the boundary mesh to simplify the meshing.

The properties `xscale`, `yscale`, and `zscale` specify scalar factors in each axis direction that the geometry is scaled by before meshing. The resulting mesh is then scaled back to fit the original geometry. The values of other properties correspond to the scaled geometry. By default, no scaling is done.

The following attribute features are used:

TABLE 4-37: ATTRIBUTE FEATURES USED

FEATURE	REMARKS
Distribution	Used when defined on edges.
Scale	Scales Size and Distribution.
Size	All properties are used.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshinit`, have been removed:

TABLE 4-38: REMOVED PROPERTIES

PROPERTY	REASON
edge	Defined by the selection.
edgelem	Defined by the attribute feature Distribution.
face	Defined by the selection.
hauto	Defined by attribute feature Size.
hcurveedg	Defined by the selection for hcurve in Size.
hcurvefac	Defined by the selection for hcurve in Size.
hcutoffedg	Defined by the selection for hmin in Size.
hcutofffac	Defined by the selection for hmin in Size.
hgradvtx	Defined by the selection for hgrad in Size.
hgradedg	Defined by the selection for hgrad in Size.
hgradfac	Defined by the selection for hgrad in Size.
hgradsub	Defined by the selection for hgrad in Size.
hmaxvtx	Defined by the selection for hmax in Size.
hmaxedg	Defined by the selection for hmax in Size.
hmaxfac	Defined by the selection for hmax in Size.
hmaxfact	Defined by the selection for hmax in Size.
hmaxsub	Defined by the selection for hmax in Size.
hnumedg	Defined by the selection for hnum in Distribution.
hnumsub	Defined by the selection for hnum in Distribution.
hpntedg	No longer supported.
hpnt	No longer supported.
hpntfac	No longer supported.
jiggle	No longer supported. Mesh quality improvements is always done.
mcase	Defined by meshing sequence.
meshstart	Input mesh provided by meshing sequence.
minit	No longer supported.
mlevel	Defined by the selection.
point	Defined by the selection.

TABLE 4-38: REMOVED PROPERTIES

PROPERTY	REASON
out	Output mesh provided by meshing sequence.
subdomain	Defined by the selection.

SEE ALSO

[Distribution](#), [FreeTri](#), [Size](#)

FreeTri

Create an unstructured triangular mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"FreeTri");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"FreeTri")` to create an unstructured triangular mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain (boundary in 3D) selection. If you do not specify any selection the feature creates a mesh on the remaining geometric entities in 2D. In 3D, the default selection is empty.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype)` to add `Size` or `Distribution` attribute features.

The following properties are available:

TABLE 4-39: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
continue	on off	on	When enabled, the mesher does not stop if there is an error
method	auto af del	auto	Triangulation method
xscale	double	1	Scale geometry in x direction before meshing
yscale	double	1	Scale geometry in y direction before meshing
zscale	double	1	Scale geometry in z direction before meshing
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth

Use the property `method` to specify the method used to triangulate domains in 2D and faces in 3D. A Delaunay based method is used if the property is set to `del` and an advancing front method is used if the property is set to `af`. If `method` is set to `auto`, the program tries to choose the best method for each geometric entity.

COMPATIBILITY

See [FreeTet](#).

SEE ALSO

[Distribution](#), [FreeTet](#), [FreeQuad](#), [Size](#)

Import

Import mesh from a file or from another meshing sequence. You can import a mesh from a COMSOL Multiphysics native file. In 3D you can also import meshes from NASTRAN, STL, or VRML files. In 2D you can also import 2D meshes from NASTRAN (the third coordinate must then be the same for all mesh points).

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Import");
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).importData();
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Import")` to import a mesh into a sequence without a corresponding geometry. It is only possible to use this feature when the geometry sequence is empty. If the sequence already contains a mesh, the imported mesh is added to the existing mesh.

Use `model.mesh(<tag>).feature(<ftag>).importData()` to import the file again.

The following mesh formats are supported:

FORMAT	FILE EXTENSION
COMSOL Multiphysics text file	.mphtxt
COMSOL Multiphysics binary file	.mphbin
NASTRAN file	.nas .bdf .dat

The following properties are available:

TABLE 4-40: AVAILABLE PROPERTIES FOR THE MESH IMPORT

PROPERTY	VALUES	DEFAULT	DESCRIPTION
allowshellpartition	on off	on	Specifies whether the boundary entities that are defined based on the data in the file can be split into smaller parts.
data	all mesh	all	Specifies the data to import from the NASTRAN file.
elemsplit	on off	off	Specifies if mesh elements of different element forms get different domain labels when importing a NASTRAN file.
facepartition	auto minimal detectfaces manual	auto	Boundary partitioning algorithm. detectfaces is only available in 3D, and manual is only available in 2D.
filename	String		The filename.
linearelem	on off	off	Specifies if the elements in the NASTRAN file are imported as linear elements.
materialsplitted	on off	on	Specifies if material data in the NASTRAN file is used to determine the domain partitioning of the domain elements.
selcreation	on off	on	Specifies whether selections corresponding to the groups of domain and boundary elements in the NASTRAN file should be created.
sequence	String		Meshing sequence name.
source	file sequence native nastran stlvrml	file	Source for the import.

The properties `elemsplit`, `linearelem`, `materialsplitted`, and `data` are only used for import of NASTRAN files. The properties `selcreation` and `allowshellpartition` are only used when `materialsplitted` is on.

elemsplit specifies if mesh elements of different element forms—that is, tetrahedral, pyramid, prism, or hexahedral—get different domain labels. The default value is **off**.

linearelem specifies if the elements in the NASTRAN file are imported as linear elements. If the value is **on** all imported elements are linear. Otherwise, the order of the imported elements is determined from the order of the elements in the file. The default value is **off**.

materialsplrit determines if material data in the file is used (if available) to determine the domain partitioning of the domain elements. If the value is **off** all domain elements in the imported mesh belongs to the same domain if possible. The default value is **off**.

If you set **facepartition** to **minimal**, the operation keeps the original partition from the file (if any), adding minimal partitioning in order to satisfy topological requirements.

If you set **facepartition** to **manual**, you can use the following properties. If you set any of these properties without setting **facepartition** to **manual**, the operation automatically switches **facepartition** to **manual**.

TABLE 4-41: VALID PROPERTY/VALUE PAIRS FOR FACEPARTITION = MANUAL

PROPERTY	VALUE	DEFAULT	DESCRIPTION
curvedface	auto manual	auto	Control parameters for partitioning of curved faces. Available only in 3D.
faceangle	double	360 degrees	Maximum angle between any two boundary elements in the same face.
minareaplane	double	0.005	Minimum relative area of face to be considered planar.
neighangle	double	40 degrees	Maximum angle between a boundary element and a neighbor that causes the elements to be part of the same boundary domain if possible.
planar	on off	on	Detect planar faces.
planarangle	double	0.6 degrees	Maximum angle between boundary element normal and a neighbor that causes the element to be a part the planar face if possible.

The following properties are available in 3D when **facepartition** is set to **detectfaces**:

TABLE 4-42: VALID PROPERTY/VALUE PAIRS FOR FACEPARTITION = DETECTFACES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
detectadjfillets	on off	on	Whether to detect cylindrical faces adjacent to the detected planar faces.
detectfacesplanar	on off	on	Whether to detect planar faces.
facemaxangle	double	40 degrees	Maximum tolerated angle between neighboring boundary elements in the same face.
planarfacemaxangle	double	0.6 degrees	Maximum tolerated angle between neighboring boundary elements in the same planar face.
planarfaceminareafraction	double	0.005	Minimum relative area for a planar face to be created.

The table below specifies the supported NASTRAN bulk data entries.

BULK DATA ENTRY			
CBAR	CORD2C	CQUAD4	GRID
CHEXA	CORD2R	CQUAD8	MAT I
CORDIC	CORD2S	CTETRA	MAT I0

BULK DATA ENTRY			
CORDIR	CPENTA	CTRIA3	PSHELL
CORDIS	CPYRAM	CTRIA6	PSOLID

The NASTRAN bulk data format uses reduced second-order elements; that is, the center node on quadrilateral mesh faces (**quadNode**) and the center node of hexahedral elements (**hexNode**) are missing. Importing a NASTRAN mesh with second-order elements, COMSOL Multiphysics interpolates the coordinates of these missing node points from the surrounding node points using the following formulas: $\text{quadNode} = 0.5 \cdot \text{quadEdgeNodes} - 0.25 \cdot \text{quadCornerNodes}$, where **quadEdgeNodes** is the sum of the coordinates of the surrounding 4 edge nodes and **quadCornerNodes** is the sum of the coordinates of the surrounding 4 corner nodes, and $\text{hexNode} = 0.25 \cdot \text{hexEdgeNodes} - 0.25 \cdot \text{hexCornerNodes}$, where **hexEdgeNodes** is the sum of the coordinates of the surrounding 12 edge nodes and **hexCornerNodes** is the sum of the coordinates of the surrounding 8 corner nodes.



The Import feature does not handle NASTRAN files in free field format where the data fields are separated by blanks.

COMPATIBILITY

For 3D mesh sequences, the setting **manual** of the property **facepartition** in the mesh Import feature, as well as all the properties associated with this setting, are deprecated as of COMSOL 5.1 and may be removed in a future version. In COMSOL 5.1, these properties are still available with unchanged behavior for backward compatibility.

The following properties from the corresponding COMSOL 3.5a command, **meshimport**, have been removed:

TABLE 4-43: REMOVED PROPERTIES

PROPERTY	REASON
mcase	Defined by meshing sequence
out	Output mesh provided by meshing sequence

SEE ALSO

[Ball](#), [Box](#), [CreateVertex](#), [DeleteEntities](#), [DetectFaces](#), [JoinEntities](#), [LogicalExpression](#)

JoinEntities

Join geometric entities of an imported mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"JoinEntities");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"JoinEntities")` to join adjacent geometric entities of an imported 2D or 3D mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify geometric entities to join.

The following properties are available:

TABLE 4-44: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
joinadj	Boolean	true	Specifies if the operation joins lower dimensional adjacent entities

SEE ALSO

[Import](#), [DeleteEntities](#)

LogicalExpression

Split geometric entities of an imported mesh by specifying a logical expression.

SYNTAX

```
model.mesh(<tag>).create(<ftag>, "LogicalExpression");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "LogicalExpression")` to split entities of an imported mesh by specifying an element set based on a logical expression.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the geometric entities for which you want to define an element selection. If you do not specify the selection, the feature operates on the entire geometry.

The following properties are available:

TABLE 4-45: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
expression	String	1	Logical expression
condition	allvertices somevertex	allvertices	Condition for inclusion of an element

SEE ALSO

[Import](#), [Ball](#), [Box](#), [Cylinder](#), [DetectFaces](#)

Map

Create a structured (mapped) quadrilateral mesh. The mapped mesher maps a regular grid defined on a logical unit square onto each domain. The mapping method is based on transfinite interpolation.

SYNTAX

```
model.mesh(<tag>).create(<ftag>, "Map");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "Map")` to create a structured quadrilateral mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain (boundary in 3D) selection. If you do not specify any selection the feature creates a mesh on the remaining domains in 2D. In 3D, the default selection is empty.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype)` to add a `Size`, `Distribution`, or `EdgeGroup` attribute feature.

The following properties are available:

TABLE 4-46: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
adjustedgdistr	on off	off	When enabled, the mapped mesher adjusts evenly distributed edge mesh
continue	on off	on	When enabled, the mesher does not stop if there is an error
interpmethod	auto transfinite2D transfinite3D	auto	Interpolation method (3D meshes only)
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth

The following attribute features are used:

TABLE 4-47: ATTRIBUTE FEATURES USED

FEATURE	REMARKS
EdgeGroup	Defined on the domain/face to be meshed
Distribution	Used when defined on edges
Scale	Scales Size and Distribution
Size	Defined on domain/face. Uses only hauto and hmax.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, meshmap, have been removed:

TABLE 4-48: REMOVED PROPERTIES

PROPERTY	REASON
edgelem	Defined by the Distribution
face	Defined by the selection
mcase	Defined by the meshing sequence
meshstart	Input mesh provided by meshing sequence
out	Output mesh provided by meshing sequence
domain	Defined by the selection

SEE ALSO

[Distribution](#), [FreeTri](#)

OnePointMap

Specify a one-point map for a face copy or a domain copy operation.

SYNTAX

```
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"OnePointMap");  
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection(property);
```

DESCRIPTION

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, "OnePointMap")` to define a one-point map for `CopyFace` or `CopyDomain` feature `<ftag>`.

The following properties are available:

TABLE 4-49: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
srcpoint1	Selection		Point on source face/domain
dstpoint1	Selection		Point on destination face/domain

Use the `OnePointMap` feature if you need to control how the source and destination faces/domains are matched in a copy face or a copy domain mesh operation. When this feature is present, the source mesh is transformed so that `srcpoint1` is mapped to `dstpoint1`.

EXAMPLE

Create a block and mesh face 4 with a fine mesh near point 8. Copy this mesh onto face 3 and ensure that the fine mesh near point 8 ends up near point 3:

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("ftri1", "FreeTri");
m.feature("ftri1").selection().set(new int[]{4});
m.feature("ftri1").create("size1", "Size");
m.feature("ftri1").feature("size1").selection().geom("geom1", 0).set(new int[]{8});
m.feature("ftri1").feature("size1").set("hmax", "0.01");
m.create("cpf1", "CopyFace");
m.feature("cpf1").selection("source").geom("geom1", 2).set(new int[]{4});
m.feature("cpf1").selection("destination").geom("geom1", 2).set(new int[]{3});
m.feature("cpf1").create("pm1", "OnePointMap");
m.feature("cpf1").feature("pm1").selection("srcpoint1").set(new int[]{8});
m.feature("cpf1").feature("pm1").selection("dstpoint1").set(new int[]{3});
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom().create('geom1', 3);
m = model.mesh().create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run();

m.create('ftri1', 'FreeTri');
m.feature('ftri1').selection().set(4);
m.feature('ftri1').create('size1', 'Size');
m.feature('ftri1').feature('size1').selection().geom('geom1', 0).set(8);
m.feature('ftri1').feature('size1').set('hmax', '0.01');
m.create('cpf1', 'CopyFace');
m.feature('cpf1').selection('source').geom('geom1', 2).set(4);
m.feature('cpf1').selection('destination').geom('geom1', 2).set(3);
m.feature('cpf1').create('pm1', 'OnePointMap');
m.feature('cpf1').feature('pm1').selection('srcpoint1').set(8);
m.feature('cpf1').feature('pm1').selection('dstpoint1').set(3);
m.run();
```

SEE ALSO

[CopyFace](#), [CopyDomain](#), [EdgeMap](#), [TwoPointMap](#)

Point

Create a point mesh.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Point");
model.mesh(<tag>).feature(<ftag>).selection();
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Point")` to mesh geometry vertices.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the point selection. If you do not specify any selection the feature creates a mesh on the remaining points.

SEE ALSO

[Edge](#)

Reference

Refer to another meshing sequence.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Reference");
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).expand();
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Reference")` to refer to another meshing sequence. Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype)` to add **Scale** attribute features.

Use `model.mesh(<tag>).feature(<ftag>).expand()` to replace the reference with a copy of the referred sequence, where the attributes have been scaled with the scale attribute features of the reference.

The following properties are available:

TABLE 4-50: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sequence	String		Tag of referred sequence.

Use the sequence property to specify another meshing sequence on the same geometry. When running the feature, all features of the specified sequence are run in the current context.

It is not allowed to introduce circular references.

EXAMPLES

Create a mixed mesh with quads and triangles on a geometry. Create a second meshing sequence with a **scale** feature and a reference to the first meshing sequence. The result is a coarser version of the first mesh.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
```

```

MeshSequence m1 = model.mesh().create("mesh1", "geom1");

g.create("sq1", "Square");
g.create("sq2", "Square");
g.feature("sq2").set("size", "0.5");
g.run();

m1.create("map1", "Map");
m1.feature("map1").selection().geom("geom1", 2).set(new int[]{1});
m1.create("ftri1", "FreeTri");
m1.feature("ftri1").selection().geom("geom1", 2).set(new int[]{2});
m1.run();

MeshSequence m2 = model.mesh().create("mesh2", "geom1");
m2.create("sca1", "Scale");
m2.feature("sca1").set("scale", "2");
m2.create("rf1", "Reference");
m2.feature("rf1").set("sequence", "mesh1");
m2.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
m1 = model.mesh.create('mesh1', 'geom1');

g.create('sq1', 'Square');
g.create('sq2', 'Square');
g.feature('sq2').set('size', '0.5');
g.run();

m1.create('map1', 'Map');
m1.feature('map1').selection().geom('geom1', 2).set(1);
m1.create('ftri1', 'FreeTri');
m1.feature('ftri1').selection().geom('geom1', 2).set(2);
m1.run();

m2 = model.mesh().create('mesh2', 'geom1');
m2.create('sca1', 'Scale');
m2.feature('sca1').set('scale', '2');
m2.create('rf1', 'Reference');
m2.feature('rf1').set('sequence', 'mesh1');
m2.run();

```

SEE ALSO

[Scale](#)

Refine

Refine a mesh.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "Refine");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "Refine")` to refine the mesh.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain selection or the entire geometry. If you do not specify any selection the feature refines all triangular and tetrahedral elements in the mesh.

The following properties are available:

TABLE 4-51: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
boxcoord	on off	off	Use coordinates of a bounding box to determine the elements to refine
rmethod	longest regular	see below	Refinement method
element	int[]	all	List of elements to refine
numrefine	int or int[]	1	Number of refinements
xmax, xmin, ymax, ymin, zmax, zmin	double		Coordinates of bounding box

Use the `boxcoord` property to refine elements inside a bounding box. To define the bounding box, set the properties `xmin`, `xmax`, `ymin`, `ymax`, `zmax`, and `zmin` on the feature, where `(xmin,ymin,zmin)` defines the lower-left corner, and `(xmax,ymax,zmax)` defines the upper-right corner of the bounding box. The elements that have all its corner points in the bounding box are refined once. `boxcoord` is automatically set to `on` if one of the coordinates are set.

The default refinement method in 2D is regular refinement, where all of the specified triangles are divided into four triangles of the same shape. Longest edge refinement, where the longest edge of each specified triangle is bisected, can be selected by giving `longest` as `rmethod`. Using `regular` as `rmethod` results in regular refinement. Some triangles outside of the specified set can also be refined, in order to preserve the triangulation and its quality.

In 3D, the default refinement method is `longest`. The regular refinement method is only implemented for uniform refinements (that is, when all elements are refined).

In 1D, regular refinement, where each element is divided into two elements of the same shape, is always used.

By default, all elements are refined once. By using the `element` property, you can specify which elements are refined and the `numrefine` property specifies how many times the elements is refined. If `numrefine` is an integer, all refined elements are refined `numrefine` times. If `numrefine` is a vector, it must have the same length as the `element` vector, and gives the number of refinements for each element.

COMPATIBILITY

The following properties from the corresponding COMSOL Multiphysics 3.5a command, `meshrefine`, have been removed:

TABLE 4-52: REMOVED PROPERTIES

PROPERTY	REASON
mcase	Defined by meshing sequence
out	Output mesh provided by meshing sequence
domain	Defined by the selection
tri, tet	Replaced by <code>element</code> and <code>numrefine</code>

EXAMPLES

Mesh two squares with free mesh. Refine the mesh on `sq2` once and refine the elements inside a box in `sq1` twice.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 2);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("sq1", "Square");
g.create("sq2", "Square");
```

```

g.feature("sq2").setIndex("pos", "1", 0);
g.run();

m.create("ftri1", "FreeTri");
m.create("ref1", "Refine");
m.feature("ref1").selection().geom("geom1", 2).set(new int[]{2});
m.create("ref2", "Refine");
m.feature("ref2").set("xmin", "0.2");
m.feature("ref2").set("xmax", "0.8");
m.feature("ref2").set("ymin", "0.2");
m.feature("ref2").set("ymax", "0.6");
m.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 2);
m = model.mesh.create('mesh1', 'geom1');

g.create('sq1', 'Square');
g.create('sq2', 'Square');
g.feature('sq2').setIndex('pos', '1', 0);
g.run();

m.create('ftri1', 'FreeTri');
m.create('ref1', 'Refine');
m.feature('ref1').selection().geom('geom1', 2).set(2);
m.create('ref2', 'Refine');
m.feature('ref2').set('xmin', '0.2');
m.feature('ref2').set('xmax', '0.8');
m.feature('ref2').set('ymin', '0.2');
m.feature('ref2').set('ymax', '0.6');
m.run();

```

SEE ALSO

[Convert](#)

Scale

Scale mesh size properties.

SYNTAX

```

model.mesh(<tag>).create(<ftag>, "Scale");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>, "Scale");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property, <value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);

```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>, "Scale")` to scale size properties defined in the sequence and use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, "Scale")` to scale size properties defined in the sequence referred to by the Reference feature `<ftag>`.

Use `model.mesh(<tag>).feature(<ftag>).selection()` or `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the geometric entity selection or the entire geometry (which is default).

The following properties are available:

TABLE 4-53: FEATURE PROPERTIES DEFINED

PROPERTY	VALUE	DEFAULT	DESCRIPTION
scale	double	1	Scale factor

Scale is a positive number. The feature scales mesh size properties, distribution properties, and boundary layer properties affecting mesh elements generated by features following the scale feature. The scale feature also affects size properties defined by Size, Distribution, and BndLayerProp features occurring later in the sequence.

A scale less than 1 gives smaller (more) elements; a scale greater than 1 gives larger (fewer) elements. The scale feature has no effect on any mesh generated earlier in the sequence.

If two or more scale features exist on the same selection, the resulting scale on that selection is the product of the given scales.

EXAMPLE

Create a block and mesh it with 10-by-10-by-10 hexahedra. Setting scale to 2 gives you a block with 5-by-5-by-5 hexahedra and setting the scale to 0.5 gives you a block with 20-by-20-by-20 hexahedra.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g = model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("sca1", "Scale");
m.create("map1", "Map");
m.feature("map1").selection().set(new int[]{1});
m.create("swe1", "Sweep");
m.run();

m.feature("sca1").set("scale", "2");
m.run();

m.feature("sca1").set("scale", "0.5");
m.run();
```

Code for use with MATLAB

```
model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run;

m.create('sca1', 'Scale');
m.create('map1', 'Map');
m.feature('map1').selection().set(1);
m.create('swe1', 'Sweep');
m.run;

m.feature('sca1').set('scale', '2');
m.run;

m.feature('sca1').set('scale', '0.5');
m.run;
```

SEE ALSO

[BndLayerProp](#), [Distribution](#), [Size](#)

Size

Specify mesh size properties.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Size");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"Size");
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection();
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).getType(property);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Size")` to specify element size properties in the sequence. Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"Size")` to specify element size properties for the feature `<ftag>` that can be any of the types `Edge`, `FreeQuad`, `FreeTri`, `FreeTet`, `Map`, or `Sweep`.

Use `model.mesh(<tag>).feature(<ftag>).selection()` or `model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection()` to specify the geometric entity selection. If you do not specify any selection the size feature is defined on all geometric entities. The selection is not available for the *default size feature*, tagged size.

The following properties are available:

TABLE 4-54: FEATURE PROPERTIES DEFINED

PROPERTY	VALUE	DEFAULT	DESCRIPTION
custom	on off	off	Setting custom to on deactivates all mesh parameters.
hauto	double	5	Automatic settings for all mesh parameters.
hcurve	double	0.3 0.6	Curvature mesh size
hcurveactive	on off	on	Specifies if hcurve is used
hgrad	double	1.3 1.5	Element growth rate
hgradactive	on off	on	Specifies if hgrad is used
hmax	double	geometry dependent	Maximum element size
hmaxactive	on off	on	Specifies if hmax is used
hmin	double	geometry dependent	Minimum element size
hminactive	on off	on	Specifies if hmin is used
hnnarrow	double	0.5	Resolution of narrow regions
hnnarrowactive	on off	on	Specifies if hnnarrow is used
table	cfid default plasma semi	default	Specifies for which physics the element size is calibrated



The properties with names ending in active are not available for the default size feature.

The property `table` specifies the physics for which the element size is calibrated.

`Hauto` is a positive scalar. This value is used to set several mesh parameters in order to get a mesh of desired size. Smaller values of `hauto` generate finer meshes with more elements. The integers between 1 and 9 has a special interpretation; they correspond to the mesh settings **Normal**, **Fine**, **Coarse**, and so forth in COMSOL Multiphysics. The value 5 correspond to **Normal**. When you set the property `hauto`, all other properties are set to their default value, according to the following tables (for `table` set to `default`).

TABLE 4-55: MESH PARAMETERS SET BY THE PROPERTY `HAUTO` IN 2D (FOR DEFAULT TABLE)

<code>HAUTO</code>	<code>HMAXFACT</code>	<code>HCURVE</code>	<code>HGRAD</code>	<code>HMINFACT</code>	<code>HNARROW</code>
1	0.01	0.2	1.1	2e-5	1
2	0.02	0.25	1.2	7.5e-5	1
3	0.037	0.25	1.25	1.25e-4	1
4	0.053	0.3	1.3	3e-4	1
5	0.067	0.3	1.3	3e-4	1
6	0.1	0.4	1.4	0.002	1
7	0.13	0.6	1.5	0.006	1
8	0.2	0.8	1.8	0.016	1
9	0.33	1	2	0.05	0.9

mesh parameters set by the property `hauto` in 3d (for default table)

<code>HAUTO</code>	<code>HMAXFACT</code>	<code>HCURVE</code>	<code>HGRAD</code>	<code>HMINFACT</code>	<code>HNARROW</code>
1	0.02	0.2	1.3	2e-4	1
2	0.035	0.3	1.35	0.0015	0.85
3	0.055	0.4	1.4	0.004	0.7
4	0.08	0.5	1.45	0.01	0.6
5	0.1	0.6	1.5	0.018	0.5
6	0.15	0.7	1.6	0.028	0.4
7	0.19	0.8	1.7	0.04	0.3
8	0.3	0.9	1.85	0.054	0.2
9	0.5	1	2	0.07	0.1

The property `hcurve` is a real value that relates the mesh size to the curvature of the geometry boundaries. The Gaussian radius of curvature is multiplied by the `hcurve` factor to obtain the mesh size along the boundary. The specified `hcurve` is only used if `hcurveactive` is on, otherwise `hcurve` is taken from a preceding size feature in the sequence. In the default size feature, tagged `size`, `hcurve` is always active and there is no `hcurveactive` property.

The property `hgrad` tells how fast the element size—measured as the length of the longest edge of the element—can grow from a region with small elements to a region with larger elements. If two elements lie one unit length apart, the difference in element size can be at most `hgrad`. The specified `hgrad` is only used if `hgradactive` is on, otherwise `hgrad` is taken from a preceding size feature in the sequence. In the default size feature, `hgrad` is always active and there is no `hcurvegrad` property.

The `hmax` parameter controls the size of the elements in the mesh. The algorithm aims at creating a mesh where no element size exceeds `hmax`. The default `hmax` value is `hmaxfact * maxdist`, where `maxdist` is the longest axis parallel distance in the geometry. The specified `hmax` is only used if `hmaxactive` is on, otherwise `hmax` is taken from a preceding size feature in the sequence. In the default size feature, `hmax` is always active and there is no `hmaxactive` property.

You can use `hmin` to control the minimum size of the elements. The main purpose of this parameter is to prevent the generation of many small elements near small curved parts of the geometry. The default `hmin` value is `hminfact * maxdist`, where `maxdist` is the longest axis parallel distance in the geometry. The specified `hmin` is only used if `hminactive` is on, otherwise `hmin` is taken from a preceding size feature in the sequence. In the default size feature, `hmin` is always active and there is no `hminactive` property.

The `hnnarrow` parameter controls the size of the elements in narrow regions. Increasing values of this property decrease the size of the elements in narrow regions. If the value of `hnnarrow` is less than one, elements that are anisotropic in size might be generated in narrow regions. The specified `hnnarrow` is only used if `hnnarrowactive` is on, otherwise `hnnarrow` is taken from a preceding size feature in the sequence. In the default size feature, `hnnarrow` is always active and there is no `hnnarrowactive` property.

The values of `hauto`, `hcurve`, `hgrad`, `hmax`, `hmin`, and `hnnarrow` are positive real scalars, or strings that evaluate to positive real scalars, given the evaluation context provided by `model.param()`.

It is not possible to specify coarser size settings on the boundary of a domain than on the domain. The finer settings on the domain is inherited by its boundaries and, in 3D, edges. A warning is issued when settings are overwritten by inheritance. If you need to create coarser mesh on a boundary, you should first mesh the boundary then add the finer size settings on the domain and a corresponding free mesh operation.

COMPATIBILITY

The following size control properties from the COMSOL Multiphysics 3.5a command `meshinit` have been removed:

TABLE 4-56: REMOVED PROPERTIES

PROPERTY	REASON
<code>hcutoff</code>	Replaced by <code>hmin</code> . The relation is given by <code>hmin = hcutoff * hcurve * maxdist</code> , where <code>maxdist</code> is the maximal axis parallel distance in the geometry.
<code>hmaxfact</code>	Replaced by <code>hmax</code> . The relation is given by <code>hmax = hmaxfact * maxdist/10</code> in 3D and <code>hmax = hmaxfact * maxdist/15</code> in 2D and 1D, where <code>maxdist</code> is defined as above.
<code>hmesh</code>	No longer supported
<code>mesh</code>	No longer supported

SEE ALSO

[Distribution](#), [Scale](#)

Sweep

Create a swept mesh in 3D by sweeping the mesh from the source face along the domain to an opposite destination face.

SYNTAX

```
model.mesh(<tag>).create(<ftag>,"Sweep");
model.mesh(<tag>).feature(<ftag>).selection();
model.mesh(<tag>).feature(<ftag>).selection(property);
model.mesh(<tag>).feature(<ftag>).set(property,<value>);
model.mesh(<tag>).feature(<ftag>).getType(property);
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,ftype);
```

DESCRIPTION

Use `model.mesh(<tag>).create(<ftag>,"Sweep")` to create a swept mesh in 3D.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the domain selection. If you do not specify any selection the feature creates a mesh on the remaining domains.

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>, ftype)` to add a Size or Distribution attribute feature.

Use `model.mesh(<tag>).feature(<ftag>).selection()` to specify the 3D domain selection. If you do not specify any selection the feature creates a mesh on the remaining domains.

The following properties are available:

TABLE 4-57: AVAILABLE PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
continue	on off	on	When enabled, the mesher does not stop if there is an error
facemethod	tri quad	quad	Face meshing method
sourceface	Selection		Source faces selection
sweepath	String	auto	Sweep path
targetface	Selection		Destination face selection
targetmesh	String	auto	Destination mesh method
smoothcontrol	on off	on	Specifies if the operation smooths the mesh across removed control entities.
smoothmaxiter	integer	8 in 2D, 4 in 3D	Specifies the number of smoothing iterations
smoothmaxdepth	integer	8 in 2D, 4 in 3D	Specifies the maximum element smoothing depth

Use the property `sourceface` and `targetface` to specify the source faces and the destination faces of the sweep, respectively. For domains in the feature selection where none of the surrounding faces are specified as either a source or a destination face, the software automatically tries to determine these faces.

Use the property `sweepath` if you want to specify the shape of the sweep path. The string is either `auto`, `straight`, `circular`, or `general`. `straight` means that all interior mesh points are located on straight lines between the corresponding source and destination points. `circular` means that all interior mesh points are located on circular arcs between the corresponding source and destination points. `general` means that the positions of the interior mesh points are determined by a general interpolation procedure. `auto`, which is default, means that the sweeping algorithm automatically tries to determine if the sweep path is straight or circular. If this is the case `sweepath` is set to `straight` or `circular`, respectively. Otherwise, `sweepath` is set to `general`.

Any source face that is not meshed, is meshed automatically. The property `facemethod` controls which face meshing method is used. If `facemethod` is `quad`, you get quadrilateral face mesh and therefore hexahedral domain mesh. If `facemethod` is `tri`, you get triangular face mesh and prism elements in the domain.

Use the property `targetmesh` if you want to specify the method to be used for transferring the source mesh to the destination. The string is either `auto`, `rigid`, `morph`, or `project`. `rigid` means that the destination mesh is created by a rigid transformation of the source mesh, `morph` means that the destination mesh is created from the source mesh by a morphing technique, and `project` means that the destination mesh is created from the source mesh by a projection technique. The value `auto`, which is the default, means that the sweeping algorithm automatically tries to determine a suitable method for creating the destination mesh.

The following attribute features are used:

TABLE 4-58: ATTRIBUTE FEATURES USED

FEATURE	REMARKS
Distribution	Used when defined on domains
Scale	Scales Size and Distribution
Size	Defined on domain.

If a Distribution feature is defined on a domain, it is used to determine the distribution of element layer in the sweep direction. Otherwise, if `haut0` or `hmax` is specified, equidistant element layers are generated.

COMPATIBILITY

The following properties from the corresponding COMSOL 3.5a command, `meshsweep`, have been removed:

TABLE 4-59: REMOVED PROPERTIES

PROPERTY	REASON
<code>elsweeplayers</code>	Defined by the property <code>elemdistr</code> in <code>Distribution</code>
<code>meshstart</code>	Input mesh provided by meshing sequence
<code>out</code>	Output mesh provided by meshing sequence
<code>domain</code>	Defined by the selection

SEE ALSO

[Distribution](#), [FreeQuad](#), [FreeTri](#), [Map](#)

TwoPointMap

Specify a two-point map for a face copy or a domain copy operation.

SYNTAX

```
model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"TwoPointMap")
model.mesh(<tag>).feature(<ftag>).feature(<ftag1>).selection(property)
```

DESCRIPTION

Use `model.mesh(<tag>).feature(<ftag>).create(<ftag1>,"TwoPointMap")` to define a two-point map for the `CopyFace` or `CopyDomain` feature `<ftag>`.

The following properties are available:

TABLE 4-60: AVAILABLE PROPERTIES

PROPERTY	VALUE	DESCRIPTION
<code>srcpoint1</code>	Selection	First point on source face/domain
<code>srcpoint2</code>	Selection	Second point on source face/domain
<code>dstpoint1</code>	Selection	First point on destination face/domain
<code>dstpoint2</code>	Selection	Second point on destination face/domain

Use the `TwoPointMap` feature if you need to control how the source and destination faces/domains are matched in a copy face or a copy domain mesh operation. When this feature is present, the source mesh is transformed so that `srcpoint1` is mapped to `dstpoint1` and `srcpoint2` is mapped to `dstpoint2`.

EXAMPLE

Create a block and mesh face 2. Copy this mesh onto the opposite face 5 and ensure that point 6 is mapped to point 4 and point 5 is mapped to Point 8.

Code for use with Java

```
Model model = ModelUtil.create("Model");
GeomSequence g= model.geom().create("geom1", 3);
MeshSequence m = model.mesh().create("mesh1", "geom1");

g.create("blk1", "Block");
g.run();

m.create("ftri1", "FreeTri");
```



```

m.feature("ftri1").selection().set(new int[]{2});
m.feature("ftri1").create("size1", "Size");
m.feature("ftri1").feature("size1").selection().geom("geom1", 1).set(new int[]{9});
m.feature("ftri1").feature("size1").set("hmax", "0.01");
m.create("cpf1", "CopyFace");
m.feature("cpf1").selection("source").geom("geom1", 2).set(new int[]{2});
m.feature("cpf1").selection("destination").geom("geom1", 2).set(new int[]{5});
m.feature("cpf1").create("ppm1", "TwoPointMap");
m.feature("cpf1").feature("ppm1").selection("srcpoint1").set(new int[]{6});
m.feature("cpf1").feature("ppm1").selection("dstpoint1").set(new int[]{4});
m.feature("cpf1").feature("ppm1").selection("srcpoint2").set(new int[]{5});
m.feature("cpf1").feature("ppm1").selection("dstpoint2").set(new int[]{8});
m.run();

```

Code for use with MATLAB

```

model = ModelUtil.create('Model');
g = model.geom.create('geom1', 3);
m = model.mesh.create('mesh1', 'geom1');

g.create('blk1', 'Block');
g.run;

m.create('ftri1', 'FreeTri');
m.feature('ftri1').selection().set(2);
m.feature('ftri1').create('size1', 'Size');
m.feature('ftri1').feature('size1').selection().geom('geom1', 1).set(9);
m.feature('ftri1').feature('size1').set('hmax', '0.01');
m.create('cpf1', 'CopyFace');
m.feature('cpf1').selection('source').geom('geom1', 2).set(2);
m.feature('cpf1').selection('destination').geom('geom1', 2).set(5);
m.feature('cpf1').create('ppm1', 'TwoPointMap');
m.feature('cpf1').feature('ppm1').selection('srcpoint1').set(6);
m.feature('cpf1').feature('ppm1').selection('dstpoint1').set(4);
m.feature('cpf1').feature('ppm1').selection('srcpoint2').set(5);
m.feature('cpf1').feature('ppm1').selection('dstpoint2').set(8);
m.run;

```

SEE ALSO

[CopyFace](#), [CopyDomain](#), [EdgeMap](#), [OnePointMap](#)

Elements and Shape Functions

This chapter contains reference information about the shape functions (elements) available in COMSOL and some background to the computational methods associated with those elements. See also [model.shape\(\)](#) in the *General Commands* chapter for details about the syntax for specifying shape functions.

Shape Functions and Element Types

This section provides an overview of the finite elements and shape functions that COMSOL uses. It also describes the available shape functions (element types) with their properties and syntax examples.

Finite Elements

Once you have a mesh, you can introduce approximations to the dependent variables. For this discussion, concentrate on the case of a single dependent variable, u . The idea is to approximate u with a function that you can describe with a finite number of parameters, the so-called *degrees of freedom* (DOF). Inserting this approximation into the weak form of the equation generates a system of equations for the degrees of freedom.

Start with a simple example: linear elements in 1D. Assume that a mesh consists of just two mesh intervals: $0 < x < 1$ and $1 < x < 2$. Linear elements means that on each mesh interval the continuous function u is linear (affine). Thus, the only thing you need to know in order to characterize u uniquely is its values at the *node points* $x_1 = 0$, $x_2 = 1$, and $x_3 = 2$. Denote these as $U_1 = u(0)$, $U_2 = u(1)$, $U_3 = u(2)$. These are the *degrees of freedom*.

Now you can write

$$u(x) = U_1\phi_1(x) + U_2\phi_2(x) + U_3\phi_3(x)$$

where $\phi_i(x)$ are certain piecewise linear functions. Namely, $\phi_i(x)$ is the function that is linear on each mesh interval, equals 1 at the i^{th} node point, and equals 0 at the other node points. For example,

$$\phi_1(x) = \begin{cases} 1-x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } 1 \leq x \leq 2 \end{cases}$$

The $\phi_i(x)$ are called the *basis functions*. The set of functions $u(x)$ is a linear function space called the *finite element space*.

For better accuracy, consider another finite element space corresponding to quadratic elements. Functions u in this space are second-order polynomials on each mesh interval. To characterize such a function, introduce new node points at the midpoint of each mesh interval: $x_4 = 0.5$ and $x_5 = 1.5$. You must also introduce the corresponding degrees of freedom $U_i = u(x_i)$. Then, on each mesh interval, the second-degree polynomial $u(x)$ is determined by the degrees of freedom at the endpoints and the midpoint. In fact, you get

$$u(x) = U_1\phi_1(x) + U_2\phi_2(x) + U_3\phi_3(x) + U_4\phi_4(x) + U_5\phi_5(x)$$

where the basis functions $\phi_i(x)$ now have a different meaning. Specifically, $\phi_i(x)$ is the function that is quadratic on each mesh interval, equals 1 at the i^{th} node point, and equals 0 at the other node points. For example,

$$\phi_1(x) = \begin{cases} (1-x)(1-2x) & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } 1 \leq x \leq 2 \end{cases}$$

In general, you specify a finite element space by giving a set of basis functions. The description of the basis functions is simplified by the introduction of *local (barycentric) coordinates* (or *element coordinates*) ξ_i . Consider a mesh element of dimension d in an n -dimensional geometry (whose spatial coordinates are denoted x_1, \dots, x_n). Consider also the *standard d -dimensional simplex*

$$\xi_1 \geq 0, \xi_2 \geq 0, \dots, \xi_d \geq 0, \xi_1 + \dots + \xi_d \leq 1$$

which resides in the local coordinate space parameterized by the local coordinates ξ_1, \dots, ξ_d . If $d = 1$, then this simplex is the unit interval. If $d = 2$, it is a triangle with two 45 degree angles, and if $d = 3$ it is a tetrahedron. Now you can consider the mesh element as a linear transformation of the standard simplex. Namely, by letting the global spatial coordinates x_i be suitable linear (affine) functions of the local coordinates, you get the mesh element as the image of the standard simplex.

When described in terms of local coordinates, the basis functions assume one of a few basic shapes. These are the *shape functions*. In the example with linear elements in 1D, any basis function on any mesh element is one of the following:

$$\phi = \xi_1, \quad \phi = 1 - \xi_1, \quad \phi = 0$$

Thus the first two are the shape functions in this example (0 is not counted as a shape function). In the example with quadratic elements in 1D, the shape functions are

$$\phi = (1 - \xi_1)(1 - 2\xi_1), \quad \phi = 4\xi_1(1 - \xi_1), \quad \phi = \xi_1(2\xi_1 - 1)$$

CURVED MESH ELEMENTS

When using higher-order elements (that is, elements of an order > 1), the solution has a smaller error. The error also depends on how well the mesh approximates the true boundary. To keep errors in the finite element approximation and the boundary approximation at the same level, it is wise to use *curved mesh elements*. They are distorted mesh elements that can approximate a boundary better than ordinary straight elements (if the model's boundary is curved). You can get curved mesh elements by writing the global coordinates x_i as polynomials of order k (the *geometry shape order*) in the local coordinates ξ_j (the earlier example took $k = 1$). Then the mesh element is the image of the standard simplex. For mesh elements that do not touch the boundary, there is no reason to make them curved, so they are straight. It is customary to use the same order k here as for the order of the (Lagrange) element. This is referred to as using *isoparametric elements*.

The order k is determined by the geometry shape order for the frame (coordinate system) associated with the finite element. The frame is determined by the property `frame` to the finite element (the default is the reference frame). For certain finite elements, the geometry shape order given by the frame can be overridden by the property `sorder`. In the COMSOL Desktop, the default setting is to use an automatic geometry shape order, which means that the geometry shape order is equal to the highest order of any shape function used in the model.

If a curved mesh element becomes too distorted, it can become inverted and cause problems in the solution. The software can then reduce the geometry shape order automatically to avoid inverted elements.



In the *COMSOL Multiphysics Reference Manual*:

- [Shape Function Variables](#)
- [Avoiding Inverted Mesh Elements](#)

LOCAL COORDINATES FOR DIFFERENT ELEMENT TYPES

The mesh element types available in COMSOL have the following representations in local coordinates ξ_1, \dots, ξ_d :

MESH ELEMENT TYPE	LOCAL COORDINATES
Edge	$0 \leq \xi_1 \leq 1$
Triangle	$\xi_1 \geq 0, \xi_2 \geq 0, \xi_1 + \xi_2 \leq 1$
Quadrilateral	$0 \leq \xi_1 \leq 1, 0 \leq \xi_2 \leq 1$
Tetrahedron	$\xi_1 \geq 0, \xi_2 \geq 0, \xi_3 \geq 0, \xi_1 + \xi_2 + \xi_3 \leq 1$

MESH ELEMENT TYPE	LOCAL COORDINATES
Hexahedron	$0 \leq \xi_1 \leq 1, 0 \leq \xi_2 \leq 1, 0 \leq \xi_3 \leq 1$
Prism	$\xi_1 \geq 0, \xi_2 \geq 0, \xi_1 + \xi_2 \leq 1, 0 \leq \xi_3 \leq 1$
Pyramid	$0 \leq \xi_1 \leq 1 - \xi_3, 0 \leq \xi_2 \leq 1 - \xi_3, 0 \leq \xi_3 \leq 1$

Shape Function Types (Elements)

THE LAGRANGE ELEMENT

The preceding examples are special cases of the *Lagrange element*. Consider a positive integer k , the *order* of the Lagrange element. For simplex mesh elements, the functions u in this finite element space are piecewise polynomials of degree k ; that is, on each mesh element u is a polynomial of degree k . In general, for a mesh element of type T , u belongs to the Lagrange shape function space $\text{Lag}_k(T)$, which is spanned by the following monomials (or rational functions, in the case of pyramid elements):

MESH ELEMENT TYPE	BASIS FOR LAGRANGE SHAPE FUNCTION SPACE
Edge	$\xi_1^l, 0 \leq l \leq k$
Triangle	$\xi_1^l \xi_2^m, 0 \leq l, 0 \leq m, l + m \leq k$
Quadrilateral	$\xi_1^l \xi_2^m, 0 \leq l \leq k, 0 \leq m \leq k$
Tetrahedron	$\xi_1^l \xi_2^m \xi_3^n, 0 \leq l, 0 \leq m, 0 \leq n, l + m + n \leq k$
Hexahedron	$\xi_1^l \xi_2^m \xi_3^n, 0 \leq l \leq k, 0 \leq m \leq k, 0 \leq n \leq k$
Prism	$\xi_1^l \xi_2^m \xi_3^n, 0 \leq l, 0 \leq m, l + m \leq k, 0 \leq n \leq k$
Pyramid	$\left(\frac{\xi_1}{1 - \xi_3}\right)^l \left(\frac{\xi_2}{1 - \xi_3}\right)^m (1 - \xi_3)^n, 0 \leq l \leq k, 0 \leq m \leq k, 0 \leq n \leq k$

To describe such a function it suffices to give its values in the *Lagrange points* of order k . These are the points whose local (element) coordinates are integer multiples of $1/k$. For example, for a triangular mesh in 2D with $k = 2$, this means that you have node points at the corners and side midpoints of all mesh triangles. For each of these node points p_i , there exists a degree of freedom $U_i = u(p_i)$ and a basis function ϕ_i . The restriction of the basis function ϕ_i to a mesh element is a function in $\text{Lag}_k(T)$ such that $\phi_i = 1$ at node i , and $\phi_i = 0$ at all other nodes. The basis functions are continuous and you have

$$u = \sum_i U_i \phi_i$$

The Lagrange element of order 1 is called the linear element. The Lagrange element of order 2 is called the quadratic element.

The Lagrange elements are available with all types of mesh elements, and it is the most common element type in the COMSOL physics interfaces. The order k can be arbitrary, but the available numerical integration formulas usually limits its usefulness to $k \leq 5$ ($k \leq 4$ for tetrahedral meshes).

Examples

For convenience we provide explicit formulas for Lagrange shape functions for the most common element types and lowest orders.

Edge mesh elements For order $k = 1$, the Lagrange shape space has dimension 2 and has the following basis:

NODE POINT P	SHAPE FUNCTION
0	$1 - \xi_1$
1	ξ_1

For order $k = 2$ the Lagrange shape space has dimension 3 and has the following basis:

NODE POINT P	SHAPE FUNCTION
0	$(1 - \xi_1)(1 - 2\xi_1)$
1/2	$4\xi_1(1 - \xi_1)$
1	$\xi_1(2\xi_1 - 1)$

Triangular mesh elements For order $k = 1$, the Lagrange shape space has dimension 3 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0)	$1 - \xi_1 - \xi_2$
(1,0)	ξ_1
(0,1)	ξ_2

For order $k = 2$, the Lagrange shape space has dimension 6 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0)	$(1 - \xi_1 - \xi_2)(1 - 2\xi_1 - 2\xi_2)$
(1/2,0)	$4\xi_1(1 - \xi_1 - \xi_2)$
(1,0)	$\xi_1(2\xi_1 - 1)$
(0,1/2)	$4\xi_2(1 - \xi_1 - \xi_2)$
(1/2,1/2)	$4\xi_1\xi_2$
(0,1)	$\xi_2(2\xi_2 - 1)$

Quadrilateral mesh elements For order $k = 1$, the Lagrange shape space has dimension 4 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0)	$(1 - \xi_1)(1 - \xi_2)$
(1,0)	$\xi_1(1 - \xi_2)$
(0,1)	$(1 - \xi_1)\xi_2$
(1,1)	$\xi_1\xi_2$

For order $k = 2$, the Lagrange shape space has dimension 9 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0)	$(1 - \xi_1)(1 - 2\xi_1)(1 - \xi_2)(1 - 2\xi_2)$
(1/2,0)	$4\xi_1(1 - \xi_1)(1 - \xi_2)(1 - 2\xi_2)$
(1,0)	$\xi_1(2\xi_1 - 1)(1 - \xi_2)(1 - 2\xi_2)$
(0,1/2)	$4(1 - \xi_1)(1 - 2\xi_1)\xi_2(1 - \xi_2)$
(1/2,1/2)	$16\xi_1(1 - \xi_1)\xi_2(1 - \xi_2)$
(1,1/2)	$4\xi_1(2\xi_1 - 1)\xi_2(1 - \xi_2)$
(0,1)	$(1 - \xi_1)(1 - 2\xi_1)\xi_2(2\xi_2 - 1)$
(1/2,1)	$4\xi_1(1 - \xi_1)\xi_2(2\xi_2 - 1)$
(1,1)	$\xi_1(2\xi_1 - 1)\xi_2(2\xi_2 - 1)$

Tetrahedral mesh elements For order $k = 1$, the Lagrange shape space has dimension 4 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0,0)	$1 - \xi_1 - \xi_2 - \xi_3$
(1,0,0)	ξ_1
(0,1,0)	ξ_2
(0,0,1)	ξ_3

For order $k = 2$, the Lagrange shape space has dimension 10 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0,0)	$(1 - \xi_1 - \xi_2 - \xi_3)(1 - 2\xi_1 - 2\xi_2 - 2\xi_3)$
(1/2,0,0)	$4\xi_1(1 - \xi_1 - \xi_2 - \xi_3)$
(1,0,0)	$\xi_1(2\xi_1 - 1)$
(0,1/2,0)	$4\xi_2(1 - \xi_1 - \xi_2 - \xi_3)$
(1/2,1/2,0)	$4\xi_1\xi_2$
(0,1,0)	$\xi_2(2\xi_2 - 1)$
(0,0,1/2)	$4\xi_3(1 - \xi_1 - \xi_2 - \xi_3)$
(1/2,0,1/2)	$4\xi_1\xi_3$
(0,1/2,1/2)	$4\xi_2\xi_3$
(0,0,1)	$\xi_3(2\xi_3 - 1)$

Hexahedral mesh elements For order $k = 1$, the Lagrange shape space has dimension 8 and has the following basis:

NODE POINT P	SHAPE FUNCTION
(0,0,0)	$(1 - \xi_1)(1 - \xi_2)(1 - \xi_3)$
(1,0,0)	$\xi_1(1 - \xi_2)(1 - \xi_3)$
(0,1,0)	$(1 - \xi_1)\xi_2(1 - \xi_3)$
(1,1,0)	$\xi_1\xi_2(1 - \xi_3)$
(0,0,1)	$(1 - \xi_1)(1 - \xi_2)\xi_3$
(1,0,1)	$\xi_1(1 - \xi_2)\xi_3$
(0,1,1)	$(1 - \xi_1)\xi_2\xi_3$
(1,1,1)	$\xi_1\xi_2\xi_3$

Syntax for the Lagrange Element (shlag)

Specify Lagrange elements in the `model.shape` field of the model object. The constructor of the Lagrange shape function is `shlag`. The following properties are allowed:

TABLE 5-1: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHLAG SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
basename	Variable name		Base variable name
order	Positive integer		Basis function order
sorder	Positive integer	Determined by frame	Geometry shape order

It is not possible to abbreviate the property names, and you must write them in lowercase letters enclosed in quotation marks. For example:

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "shlag");
model.shape("shu").feature("f1").set("order", 2);
model.shape("shu").feature("f1").set("basename", "u");
```

The Lagrange element defines the following variables. Denote `basename` with u , and let x and y denote (not necessarily distinct) spatial coordinates. The variables are (`sdim` = space dimension and `edim` = mesh element dimension):

- u
- ux , meaning the derivative of u with respect to x , defined on `edim` = `sdim`
- uxy , meaning a second derivative, defined on `edim` = `sdim`
- uTx , the tangential derivative variable, meaning the x -component of the tangential projection of the gradient, defined on `edim` < `sdim`
- $uTxy$, meaning xy -component of the tangential projection of the second derivative, defined when `edim` < `sdim`

When calculating the derivatives, the global spatial coordinates are expressed as polynomials of degree (at most) `sorder` in the local coordinates.



The use of isoparametric elements means that u is not a polynomial in the global coordinates (if $k > 1$), only in the local coordinates.

THE ARGYRIS ELEMENT

For a function represented with Lagrange elements, the first derivatives between mesh elements can be discontinuous. In certain equations (for example, the biharmonic equation) this can be a problem. The *Argyris element* has basis functions with continuous derivatives between mesh triangles (it is defined in 2D only). The second order derivative is continuous in the triangle corners. On each triangle, a function u in the Argyris finite element space is a polynomial of degree 5 in the local coordinates.

The Argyris element is available with triangular meshes only.

When setting Dirichlet boundary conditions on a variable that has Argyris shape functions, a locking effect can occur if the boundary is curved and constraint order (`cporder`) 5 is used. Use `cporder=4` if the boundary is curved and `cporder=5` for straight boundaries.

Syntax for the Argyris Element (`sharg_2_5`)

Specify Argyris shape functions in the `model.shape` field of the model object. The constructor of the Argyris shape function is `sharg_2_5`. The following properties are allowed:

TABLE 5-2: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHARG SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>basename</code>	Variable name		Base variable name

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "sharg_2_5");
model.shape("shu").feature("f1").set("basename", "u");
```

The Argyris element defines the following degrees of freedom (where u is the base name and x and y are the spatial coordinate names):

- u at corners
- ux and uy at corners, meaning derivatives of u
- uxx , uxy , and uyy at corners, meaning second derivatives
- un at side midpoints, meaning a normal derivative. The direction of the normal is to the right if moving along an edge from a corner with lower mesh vertex number to a corner with higher number

The Argyris element defines the following field variables (where `sdim` = space dimension = 2 and `edim` = mesh element dimension):

- u
- ux , meaning the derivative of u with respect to x
- uxy , meaning a second derivative, defined for `edim` = `sdim` and `edim` = 0
- $uxTy$, the tangential derivative variable, meaning the y -component of the tangential projection of the gradient of ux , defined for $0 < \text{edim} < \text{sdim}$

When calculating the derivatives, the global spatial coordinates are always expressed with shape order 1 in the Argyris element.

THE HERMITE ELEMENT

On each mesh element, the functions in the Hermite finite element space are the same as for the Lagrange element, namely, all polynomials of degree (at most) k in the local coordinates. The difference lies in which DOFs are used. For the Hermite element, a DOF u exists at each Lagrange point of order k , except at those points adjacent to a corner of the mesh element. These DOFs are the values of the function. In addition, other DOFs exist for the first derivatives of the function (with respect to the global coordinates) at the corners (ux and uy in 2D). Together, these DOFs determine the polynomials completely. The functions in the Hermite finite element space have continuous

derivatives between mesh elements at the mesh vertices. However, at other common points for two mesh elements, these derivatives are not continuous. You can think of the Hermite element as lying between the Lagrange and Argyris elements.

The Hermite element is available with all types of mesh elements. The order $k \geq 3$ can be arbitrary, but the available numerical integration formulas usually limits its usefulness to $k \leq 5$ ($k \leq 4$ for tetrahedral meshes).

When setting Dirichlet boundary conditions on a variable that has Hermite shape functions, a locking effect can occur if the boundary is curved and the constraint order `cporder` is the same as the order of the Hermite element. This means that the derivative becomes over constrained at mesh vertices at the boundary, due to the implementation of the boundary conditions. To prevent this locking, you can specify `cporder` to be the element order minus 1.

Syntax for the Hermite Element (*shherm*)

Specify Hermite shape functions in the `model.shape` field of the model object. The constructor of the Hermite shape function is `shherm`. The following properties are allowed:

TABLE 5-3: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHHERM SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>basename</code>	Variable name		Base variable name
<code>order</code>	Integer ≥ 3		Basis function order
<code>sorder</code>	Positive integer	Determined by frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "shherm");
model.shape("shu").feature("f1").set("order", 3);
model.shape("shu").feature("f1").set("basename", "u");
```

The Hermite element defines the following degrees of freedom:

- The value of the variable `basename` at each Lagrange node point that is not adjacent to a corner of the mesh element.
- The values of the first derivatives of `basename` with respect to the global spatial coordinates at each corner of the mesh element. The names of these derivatives are formed by appending the spatial coordinate names to `basename`.

The Hermite element defines the following field variables. Denote `basename` with u , and let x and y denote (not necessarily distinct) spatial coordinates. The variables are (`sdim` = space dimension and `edim` = mesh element dimension):

- u
- ux , meaning the derivative of u with respect to x , defined when `edim` = `sdim` or `edim`=0
- uxy , meaning a second derivative, defined when `edim` = `sdim`
- uTx , the tangential derivative variable, meaning the x -component of the tangential projection of the gradient, defined when $0 < \text{edim} < \text{sdim}$
- $uTxy$, meaning xy -component of the tangential projection of the second derivative, defined when `edim` < `sdim`

When calculating the derivatives, the global spatial coordinates are expressed as polynomials of degree (at most) `sorder` in the local coordinates.

BUBBLE ELEMENTS

Bubble elements have shape functions that are zero on the boundaries of the mesh element and have a maximum in the middle of the mesh element. The shape function (there is only one for each mesh element) is defined by a lowest-order polynomial that is zero on the boundary of the element.

The bubble element are available with all types of mesh elements.

Syntax for Bubble Elements (shbub)

Specify bubble shape functions in the `model.shape` field of the model object. The constructor of a bubble shape function is `shbub`. The following properties are allowed:

TABLE 5-4: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHBUB SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
basename	variable name		Base variable name
mdim	nonnegative integer	sdim	Dimension of the mesh elements on which the bubble exist
sorder	positive integer	determined by frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "shbub");
model.shape("shu").feature("f1").set("mdim", 2);
model.shape("shu").feature("f1").set("basename", "u");
```

The bubble element has a single degree of freedom, `basename`, at the midpoint of the mesh element.

The bubble element defines the following field variables. Denote `basename` with u , and let x and y denote (not necessarily distinct) spatial coordinates. The variables are (`sdim` = space dimension and `edim` = mesh element dimension):

- u , defined when $\text{edim} \leq \text{mdim}$, $u = 0$ if $\text{edim} < \text{mdim}$.
- ux , meaning the derivative of u with respect to x , defined when $\text{edim} = \text{mdim} = \text{sdim}$.
- uTx , the tangential derivative variable, meaning the x -component of the tangential projection of the gradient, defined when $\text{mdim} < \text{sdim}$ and $\text{edim} \leq \text{mdim}$. $uTx = 0$ if $\text{edim} < \text{mdim}$.
- $uTxy$, meaning the xy -component of the tangential projection of the second derivative, defined when $\text{mdim} < \text{sdim}$ and $\text{edim} \leq \text{mdim}$. $uTxy = 0$ if $\text{edim} < \text{mdim}$.

THE CURL ELEMENT

In electromagnetics, *curl elements* (also called *vector elements* or *Nédélec's edge elements*) are widely used. Each mesh element has DOFs corresponding only to tangential components of the field. For example, in a tetrahedral mesh in 3D each of the three edges in a triangle face element has degrees of freedom that are tangential components of the vector field in the direction of the corresponding edges, and in the interior of the triangle there are degrees of freedom that correspond to vectors tangential to the triangle itself (if the element order is high enough). Finally, in the interior of the mesh tetrahedron there are degrees of freedom in all coordinate directions (if the element order is high enough). This implies that tangential components of the vector field are continuous across element boundaries, but the normal component is not necessarily continuous. This also implies that the curl of the vector field is an integrable function, so these elements are suitable for equations using the curl of the vector field.

The curl elements are available for all types of mesh elements. The polynomial order of the curl element can be at most three in 3D and at most four in 2D and 1D.

Detailed description of the Curl Element

The precise relation between the vector field (actually a one-form) and the DOFs of the curl element is rather complicated. We start by introducing for every mesh element T of dimension d the local vector shape function space $\text{Curl}_k(T)$ consisting of vector fields $\Phi = (\Phi_1, \dots, \Phi_d)$ satisfying the following condition:

$$\text{Curl}_k(T) = \{\Phi; \Phi_i \in \text{Lag}_k(T), \xi_1 \Phi_1 + \dots + \xi_d \Phi_d \in \text{Lag}_k(T)\}$$

where $\text{Lag}_k(T)$ is the Lagrange shape function space defined in the section on Lagrange shape functions. There is one exception; for pyramid mesh elements the definition needs to be modified as follows:

$$\text{Curl}_k(T) = \{\Phi = \Psi + \nabla \psi; \Psi_i \in \text{Lag}_k(T), \psi \in \text{Lag}_k(T), \xi_1 \Phi_1 + \dots + \xi_d \Phi_d \in \text{Lag}_k(T)\}$$

Next we introduce a transformation between local and global coordinates for a vector field in a mesh element T of dimension d embedded in space of dimension D . For a vector field in global coordinates $\varphi = (\varphi_1, \dots, \varphi_D)$, the corresponding vector field in local coordinates is $\Phi = (\Phi_1, \dots, \Phi_d)$ with components defined by

$$\Phi_i = \sum_{j=1}^D \frac{\partial x_j}{\partial \xi_i} \varphi_j$$

For mesh elements of full dimension $d = D$, this relation can be inverted to give

$$\varphi_j = \sum_{i=1}^d \frac{\partial \xi_i}{\partial x_j} \Phi_i$$

In particular, if T has full dimension, $\tilde{T} \subset T$ is a face or edge of T , and Φ is a vector field on T in local coordinates, then a vector field φ can be computed, and from its restriction to \tilde{T} a vector field $\tilde{\Phi}$ in local coordinates on \tilde{T} .

Only the components of the global coordinate vector field φ can be directly evaluated as variables.

The relation between DOFs and vector fields is defined by specifying in each mesh element a number of pairs (p, v) , where p is a point in the mesh element (given in local coordinates) and v is a vector. The DOFs are then defined as

$$\text{DOF}_{(p,v)} = v \cdot \Phi(p)$$

If the number of pairs in each mesh element of type T is chosen to be equal to the dimension of $\text{Curl}_k(T)$, the values of these DOFs uniquely define the vector field Φ . Typically, some of the pairs (p, v) have p in the interior of T , while others have p on some face or edge $\tilde{T} \subset T$. It is convenient to think of the DOFs corresponding to the latter points as inherited from \tilde{T} .

Since there might be more than one DOF in the same point p , DOF names have a suffix consisting of two integers, to distinguish different DOFs at the same point.

The choice of pairs (p, v) must take into account the orientation of each mesh element. Failing to do this would result in a vector field whose tangential component changes direction at the boundary between mesh elements. This is done by means of a global numbering of all mesh vertices. These indices can be accessed by the API method `model.mesh(<tag>).getElem(<type>)`.

Edge mesh elements Each edge mesh element has k DOFs. Let P_0, P_1 be the local coordinates of the endpoints of the edge, written in order of increasing mesh vertex indices (so P_0, P_1 is some permutation of the points 0 and

1 on the real line). Let v_1 be the 1-dimensional vector $\overrightarrow{P_0 P_1}$. In local coordinates, the DOFs on the edge element are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$\frac{l}{k+1}, 1 \leq l \leq k$	v_1	10

Triangular mesh elements Each triangular mesh element has k DOFs on each edge (inherited from the corresponding edge elements), and $k \geq 2k(k-1)$ DOFs in the interior. Let P_0, P_1, P_2 be the local coordinates of the triangle vertices, in order of increasing mesh vertex indices (so P_0, P_1, P_2 is some permutation of the points $(0, 0), (1, 0), (0, 1)$). Let $v_1 = \overrightarrow{P_0 P_1}$ and $v_2 = \overrightarrow{P_0 P_2}$. Then the interior DOFs in this triangle are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$(\frac{l}{k+1}, \frac{m}{k+1}), 1 \leq l, 1 \leq m, l+m \leq k$	v_1	20
$(\frac{l}{k+1}, \frac{m}{k+1}), 1 \leq l, 1 \leq m, l+m \leq k$	v_2	21

Quadrilateral mesh elements Each quadrilateral mesh element has k DOFs on each edge (inherited from the corresponding edge elements), and $2k(k-1)$ DOFs in the interior. Let P_0 be the local coordinate of the vertex with lowest global index, P_1 the other vertex on the same horizontal edge as P_0 , and P_2 the other vertex on the same vertical edge as P_0 (so P_0, P_1, P_2 is some subset of the points $(0, 0), (1, 0), (0, 1), (1, 1)$). Let $v_1 = \overrightarrow{P_0 P_1}$ and $v_2 = \overrightarrow{P_0 P_2}$. Then the interior DOFs in this quadrilateral are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$(\frac{l}{k+1}, \frac{m}{k}), 1 \leq l \leq k, 1 \leq m \leq k-1$	v_1	10
$(\frac{l}{k}, \frac{m}{k+1}), 1 \leq l \leq k-1, 1 \leq m \leq k$	v_2	10

Tetrahedral mesh elements Each tetrahedral mesh element has k DOFs on each edge, $k(k-1)$ DOFs on each face, and $k(k-1)(k-2)/2$ DOFs in the interior. Let P_0, P_1, P_2, P_3 be the local coordinates of the tetrahedron vertices, in order of increasing mesh vertex indices (so P_0, P_1, P_2, P_3 is some permutation of the points $(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)$). Let

$$v_1 = \overrightarrow{P_0 P_1}, v_2 = \overrightarrow{P_0 P_2}, \text{ and } v_3 = \overrightarrow{P_0 P_3}$$

Then the interior DOFs in this tetrahedron are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}), 1 \leq l, 1 \leq m, 1 \leq n, l+m+n \leq k$	v_1	30
$(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}), 1 \leq l, 1 \leq m, 1 \leq n, l+m+n \leq k$	v_2	31
$(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}), 1 \leq l, 1 \leq m, 1 \leq n, l+m+n \leq k$	v_3	32

Hexahedral mesh elements Each hexahedral mesh element has k DOFs on each edge, $2k(k-1)$ DOFs on each face, and $3k(k-1)^2$ DOFs in the interior. Let P_0 be the local coordinate of the vertex with lowest global index, and let P_1, P_2, P_3 be the other vertices that differ from P_0 only in the ξ_1, ξ_2, ξ_3 coordinates respectively. Let

$$v_1 = \overrightarrow{P_0 P_1}, v_2 = \overrightarrow{P_0 P_2}, v_3 = \overrightarrow{P_1 P_3}$$

Then the interior DOFs in this hexahedron are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$\left(\frac{l}{k+1}, \frac{m}{k}, \frac{n}{k}\right), 1 \leq l \leq k, 1 \leq m \leq k-1, 1 \leq n \leq k-1$	v_1	10
$\left(\frac{l}{k}, \frac{m}{k+1}, \frac{n}{k}\right), 1 \leq l \leq k-1, 1 \leq m \leq k, 1 \leq n \leq k-1$	v_2	10
$\left(\frac{l}{k}, \frac{m}{k}, \frac{n}{k+1}\right), 1 \leq l \leq k-1, 1 \leq m \leq k-1, 1 \leq n \leq k$	v_3	10

Prism mesh elements Each prism mesh element has k DOFs on each edge, $k(k-1)$ DOFs on each triangular face, $2k(k-1)$ DOFs on each quadrilateral face, and $k(k-1)(3k-4)/2$ DOFs in the interior. Let P_0 be the local coordinate of the vertex with lowest global index, let P_1, P_2 be the other two vertices on the same triangular face as P_0 , numbered so that P_1 has lower index than P_2 , and let P_3 be the third point connected to P_0 by an edge. Let

$$v_1 = \overrightarrow{P_0 P_1}, v_2 = \overrightarrow{P_0 P_2}, v_3 = \overrightarrow{P_1 P_3}$$

Then the interior DOFs in this prism are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$\left(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k}\right), 1 \leq l, 1 \leq m, l+m \leq k, 1 \leq n \leq k-1$	v_1	20
$\left(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k}\right), 1 \leq l, 1 \leq m, l+m \leq k, 1 \leq n \leq k-1$	v_2	21
$\left(\frac{l}{k}, \frac{m}{k}, \frac{n}{k+1}\right), 1 \leq l, 1 \leq m, l+m \leq k-1, 1 \leq n \leq k$	v_3	10

Pyramid mesh elements Each pyramid mesh element has k DOFs on each edge, $k(k-1)$ DOFs on each triangular face, $2k(k-1)$ DOFs on the quadrilateral face, and $k(k-1)(2k-1)/2$ DOFs in the interior. Let $v_1 = (1, 0, 0)$, $v_2 = (0, 1, 0)$, and $v_3 = (0, 0, 1)$. Then the interior DOFs in this pyramid are defined by the following pairs (p, v) :

P	V	DOF NAME SUFFIX
$\left(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}\right), 1 \leq l \leq k-n, 1 \leq m \leq k-n, 1 \leq n \leq k$	v_1	30
$\left(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}\right), 1 \leq l \leq k-n, 1 \leq m \leq k-n, 1 \leq n \leq k$	v_2	31
$\left(\frac{l}{k+1}, \frac{m}{k+1}, \frac{n}{k+1}\right), 1 \leq l \leq k-n, 1 \leq m \leq k-n, 1 \leq n \leq k$	v_3	32

Examples

For convenience we provide here explicit formulas for the Curl shape functions for the lowest orders and most common mesh elements. For each element type, the shape function corresponding to each DOF is written out in

local coordinates, for a particular ordering of the mesh element vertices. For other numbers, the difference is that the vector v changes direction for some of the DOFs. For DOFs with suffix 10 this simply means that the corresponding shape functions change sign. Other shape functions might be replaced by a linear combination of the shape functions corresponding to DOFs at the same node point.

Triangular mesh elements The following formulas assume that the vertices of the triangle, in order of increasing mesh vertex index, have local coordinates $(0, 0)$, $(1, 0)$, $(0, 1)$.

For order $k = 1$, the Curl shape space has dimension 3, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/2,0)	(1,0)	10	$(1 - \xi_2, \xi_1)$
(0,1/2)	(0,1)	10	$(\xi_2, 1 - \xi_1)$
(1/2,1/2)	(-1,1)	10	$(-\xi_2, \xi_1)$

For order $k = 2$, the Curl shape space has dimension 8, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/3,0)	(1,0)	10	$(2 - 3\xi_1 - 5\xi_2 + 3\xi_1\xi_2 + 3\xi_2^2, 2\xi_1 - 3\xi_1^2 - 3\xi_1\xi_2)$
(2/3,0)	(1,0)	10	$(-1 + 3\xi_1 + \xi_2 - 3\xi_1\xi_2, -\xi_1 + 3\xi_1^2)$
(0,1/3)	(0,1)	10	$(2\xi_2 - 3\xi_1\xi_2 - 3\xi_2^2, 2 - 5\xi_1 - 3\xi_2 + 3\xi_1^2 + 3\xi_1\xi_2)$
(0,2/3)	(0,1)	10	$(-\xi_2 + 3\xi_2^2, -1 + \xi_1 + 3\xi_2 - 3\xi_1\xi_2)$
(2/3,1/3)	(-1,1)	10	$(\xi_2 - 3\xi_1\xi_2, -\xi_1 + 3\xi_1^2)$
(1/3,2/3)	(-1,1)	10	$(\xi_2 - 3\xi_2^2, -\xi_1 + 3\xi_1\xi_2)$
(1/3,1/3)	(1,0)	20	$(6\xi_2 - 3\xi_1\xi_2 - 6\xi_2^2, -3\xi_1 + 3\xi_1^2 + 6\xi_1\xi_2)$
(1/3,1/3)	(0,1)	21	$(-3\xi_2 + 6\xi_1\xi_2 + 3\xi_2^2, 6\xi_1 - 6\xi_1^2 - 3\xi_1\xi_2)$

Quadrilateral mesh elements The following formulas assume that the vertices of the quadrilateral, in order of increasing mesh vertex index, have local coordinates $(0, 0)$, $(1, 0)$, $(0, 1)$, $(1, 1)$.

For order $k = 1$ the Curl shape space has dimension 4, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/2,0)	(1,0)	10	$(1 - \xi_2, 0)$
(1/2,1)	(1,0)	10	$(\xi_2, 0)$
(0,1/2)	(0,1)	10	$(0, 1 - \xi_1)$
(1,1/2)	(0,1)	10	$(0, \xi_1)$

For order $k = 2$ the Curl shape space has dimension 12, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/3,0)	(1,0)	10	$((2 - 3\xi_1)(1 - \xi_2)(1 - 2\xi_2), 0)$
(2/3,0)	(1,0)	10	$((3\xi_1 - 1)(1 - \xi_2)(1 - 2\xi_2), 0)$
(1/3,1)	(1,0)	10	$((2 - 3\xi_1)\xi_2(2\xi_2 - 1), 0)$
(2/3,1)	(1,0)	10	$((3\xi_1 - 1)\xi_2(2\xi_2 - 1), 0)$
(0,1/3)	(0,1)	10	$(0, (2 - 3\xi_2)(1 - \xi_1)(1 - 2\xi_1))$
(0,2/3)	(0,1)	10	$(0, (3\xi_2 - 1)(1 - \xi_1)(1 - 2\xi_1))$
(1,1/3)	(0,1)	10	$(0, (2 - 3\xi_2)\xi_1(2\xi_1 - 1))$
(1,2/3)	(0,1)	10	$(0, (3\xi_2 - 1)\xi_1(2\xi_1 - 1))$
(1/3,1/2)	(1,0)	10	$(4(2 - 3\xi_1)\xi_2(1 - \xi_2), 0)$
(2/3,1/2)	(1,0)	10	$(4(3\xi_1 - 1)\xi_2(1 - \xi_2), 0)$
(1/2,1/3)	(0,1)	10	$(0, 4(2 - 3\xi_2)\xi_1(1 - \xi_1))$
(1/2,2/3)	(0,1)	10	$(0, 4(3\xi_2 - 1)\xi_1(1 - \xi_1))$

Tetrahedral mesh elements The following formulas assume that the vertices of the tetrahedron, in order of increasing mesh vertex index, have local coordinates $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$.

For order $k = 1$ the Curl shape space has dimension 6, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/2,0,0)	(1,0,0)	10	$(1 - \xi_2 - \xi_3, \xi_1, \xi_1)$
(0,1/2,0)	(0,1,0)	10	$(\xi_2, 1 - \xi_1 - \xi_3, \xi_2)$
(0,0,1/2)	(0,0,1)	10	$(\xi_3, \xi_3, 1 - \xi_1 - \xi_2)$
(1/2,1/2,0)	(-1,1,0)	10	$(-\xi_2, \xi_1, 0)$
(1/2,0,1/2)	(-1,0,1)	10	$(-\xi_3, 0, \xi_1)$
(0,1/2,1/2)	(0,-1,1)	10	$(0, -\xi_3, \xi_2)$

Hexahedral mesh elements The following formulas assume that the vertices of the hexahedron, in order of increasing mesh vertex index, have local coordinates $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(1, 1, 0)$, $(0, 0, 1)$, $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 1)$.

For order $k = 1$ the Curl shape space has dimension 12, and has the following basis:

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/2,0,0)	(1,0,0)	10	$((1 - \xi_2)(1 - \xi_3), 0, 0)$
(1/2,1,0)	(1,0,0)	10	$(\xi_2(1 - \xi_3), 0, 0)$
(1/2,0,1)	(1,0,0)	10	$((1 - \xi_2)\xi_3, 0, 0)$

P	V	DOF NAME SUFFIX	LOCAL VECTOR FIELD
(1/2,1,1)	(1,0,0)	10	$(\xi_2 \xi_3, 0, 0)$
(0,1/2,0)	(0,1,0)	10	$(0, (1 - \xi_1)(1 - \xi_3), 0)$
(1,1/2,0)	(0,1,0)	10	$(0, \xi_1(1 - \xi_3), 0)$
(0,1/2,1)	(0,1,0)	10	$(0, (1 - \xi_1)\xi_3, 0)$
(1,1/2,1)	(0,1,0)	10	$(0, \xi_1 \xi_3, 0)$
(0,0,1/2)	(0,0,1)	10	$(0, 0, (1 - \xi_1)(1 - \xi_2))$
(1,0,1/2)	(0,0,1)	10	$(0, 0, \xi_1(1 - \xi_2))$
(0,1,1/2)	(0,0,1)	10	$(0, 0, (1 - \xi_1)\xi_2)$
(1,1,1/2)	(0,0,1)	10	$(0, 0, \xi_1 \xi_2)$

Syntax for the Curl Element (*shcurl*)

Specify curl shape functions in the `model.shape` field of the model object. The constructor of the curl shape function is `shcurl`. The following properties are allowed:

TABLE 5-5: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHCURL SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
fieldname	string		Field name
compnames	string array	derived from fieldname	Names of vector field components
dofbasename	string	See below	Base name of degrees of freedom
dcompnames	string array	See below	Names of the anti-symmetrized components of the gradient of the vector field
order	integer		Basis function order
sorder	positive integer	Given by frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("shu","f");
model.shape("shu").create("f1","shcurl");
model.shape("shu").feature("f1").set("fieldname","E");
model.shape("shu").feature("f1").set("order",2);
model.shape("shu").feature("f1").set("compnames",new String[]{"Ex","Ey"});
model.shape("shu").feature("f1").set("dofbasename","tE");
```

The default for `compnames` is `fieldname` concatenated with the spatial coordinate names. The default for `dofbasename` is `tallcomponents`, where `allcomponents` is the concatenation of the names in `compnames`.

The property `dcompnames` lists the names of the component of the antisymmetric matrix

$$dA_{ij} = \frac{\partial A_j}{\partial x_i} - \frac{\partial A_i}{\partial x_j},$$

where A_i are the vector field components and x_i are the spatial coordinates. The components are listed in row order. If a name is the empty string, the field variable corresponding to that component is not defined. If you have provided `compnames`, the default for the entries in `dcompnames` is `compnames(j) sdimnames(i) compnames(i) sdimnames(j)` for off-diagonal elements. If only `fieldname` has been given, the default for the entries are `dfieldname sdimnames(i) sdimnames(j)`. Diagonal elements are not defined per defaults. For example,

```
shcurl('order',3,'fieldname','A','dcompnames',
{'',' ','curlAy','curlAz','',' ','curlAx',''}).
```

The curl element defines the following degrees of freedom: $\text{dofbasename } d c$, where $d = 1$ for DOFs in the interior of an edge, $d = 2$ for DOFs in the interior of a surface, and so forth, and c is a number between 0 and $d - 1$.

The curl element defines the following field variables (where `comp` is a component name from `compnames`, and `dcomp` is a component from `dcompnames`, `sdim` = space dimension and `edim` = mesh element dimension):

- `comp`, meaning a component of the vector, defined when `edim = sdim`.
- `tcomp`, meaning one component of the tangential projection of the vector onto the mesh element, defined when `edim < sdim`.
- `comp x` , meaning the derivative of a component of the vector with respect to global spatial coordinate x , defined when `edim = sdim`.
- `tcomp Tx` , the tangential derivative variable, meaning the x component of the projection of the gradient of `tcomp` onto the mesh element, defined when `edim < sdim`. Here, x is the name of a spatial coordinate.
- `dcomp`, meaning a component of the anti-symmetrized gradient, defined when `edim = sdim`.
- `tdcomp`, meaning one component of the tangential projection of the anti-symmetrized gradient onto the mesh element, defined when `edim < sdim`.

For performance reasons, use `dcomp` in expressions involving the curl rather than writing it as the difference of two gradient components.

For the computation of components, the global spatial coordinates are expressed as polynomials of degree (at most) `sorder` in the local coordinates.

DISCONTINUOUS ELEMENTS

The functions in the discontinuous elements space are the same as for the Lagrange element, with the difference that the basis functions are discontinuous between the mesh elements.

Syntax for the Discontinuous Elements (shdisc and shhwdisc)

Specify discontinuous shape functions in the `model.shape` field of the model object. The constructor of the discontinuous shape functions is either `shdisc` or `shhwdisc`. The difference between these two is that the latter has optimal placement of degrees of freedom on triangle and tetrahedral meshes with respect to certain interpolation error estimates, whereas the former are available on all types of mesh elements with arbitrary polynomial order k . However, the available numerical integration formulas usually limits the usefulness to $k \leq 5$ ($k \leq 4$ for tetrahedral meshes). The following properties are allowed:

TABLE 5-6: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHDISC SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>basename</code>	variable name		Base variable name
<code>order</code>	integer		Basis function order
<code>mdim</code>	nonnegative integer	<code>sdim</code>	Dimension of the mesh elements where the discontinuous element exists
<code>sorder</code>	positive integer	given by frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("sh1","frame1");
model.shape("sh1").create("f1","shdisc");
model.shape("sh1").feature("f1").set("order",2);
model.shape("sh1").feature("f1").set("basename","u");
```

The discontinuous element defines the following field variables. Denote `basename` with u , and let x denote the spatial coordinates. The variables are (`edim` is the mesh element dimension):

- u , defined when `edim = mdim`.
- ux , meaning the derivative of u with respect to x , defined when `edim = mdim = sdim`.
- uTx , the tangential derivative variable, meaning the derivative of u with respect to x , defined when `edim = mdim < sdim`.

DENSITY ELEMENTS

The functions in the density elements space are the same as for the discontinuous element if the mesh element is not curved. If the element is curved, the functions define a density of the given order in *local* coordinates, and the value in global coordinates depends on the transformation between local and global coordinates. The DOF values are proportional to the volume of the mesh elements.

The discontinuous elements are available with all types of mesh elements. The order k can be arbitrary, but the available numerical integration formulas usually limits its usefulness to $k \leq 5$ ($k \leq 4$ for tetrahedral meshes).

Syntax for the Density Element (`shdens`)

Specify density shape functions in the `model.shape` field of the model object. The constructor of the density shape function is `shdens`. The following properties are allowed:

TABLE 5-7: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHDENS SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>basename</code>	variable name		Base variable name
<code>order</code>	integer		Basis function order
<code>sorder</code>	positive integer	given by frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "shdens");
model.shape("shu").feature("f1").set("order", 2);
model.shape("shu").feature("f1").set("basename", "u");
```

The density element defines the following field variables. Denote `basename` with u , and let x denote the spatial coordinates. The variables are (`edim` is the mesh element dimension):

- u , defined when `edim = sdim`.
- ux , meaning the derivative of u with respect to x , defined when `edim = sdim`.

GAUSS POINT DATA ELEMENTS

The Gauss point data elements are not shape functions in the usual sense. Instead, they define discrete degrees of freedom associated with the Gauss points used in an integration rule of a given order. To make these degrees of freedom available for evaluation and use in equations, a field variable with the same name as the degree of freedom is also defined. This field can be evaluated at any point inside the element but simply evaluates to the degree of freedom value at the closest Gauss point.

Use the Gauss point data elements to, for example, store historical values at the integration points in models with hysteresis, or to solve an implicit equation for a material property at each point. Make sure to use elements of the same order as the integration rule used for assembly of the main PDE, typically twice the discretization order of the main dependent variables. This ensures that the Gauss point data really behaves as a truly local quantity without any spatial correlation.

Syntax for the Gauss Point Data Elements (shgp)

Specify Gauss point data shape functions in the `model.shape` field of the model object. The constructor of the density shape function is `shgp`. The following properties are allowed:

TABLE 5-8: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHGP SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
basename	variable name		Base variable name
order	integer		Basis function order
mdim	integer		Element dimension
valuetype	real complex	complex	Value type in case of using split representation of complex variables ^a

a. The value type is ignored when split representation of complex variables is not used.

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks. The following code creates a Gauss point data shape function declaring the degree of freedom u at integration points of order 4 in three-dimensional mesh elements.

```
model.shape().create("shu", "f");
model.shape("shu").create("f1", "shgp");
model.shape("shu").feature("f1").set("order", 4);
model.shape("shu").feature("f1").set("basename", "u");
model.shape("shu").feature("f1").set("mdim", "3");
```

The Gauss point data element defines the following field variables. Denote `basename` with u and let `edim` be the evaluation dimension:

- u , defined when `edim` \leq `mdim`.

DIVERGENCE ELEMENTS

For modeling the **B** (magnetic flux density) and **D** (electric displacement) fields in electromagnetics, the divergence elements are useful. The DOFs on the boundary of a mesh element correspond to normal components of the field. In addition, there are DOFs corresponding to all vector field components in the interior of the mesh element of dimension `sdim` (if the order is high enough). This implies that the normal component of the vector field is continuous across element boundaries, but the tangential components are not necessarily continuous. This also implies that the divergence of the vector field is an integrable function, so these elements are suitable for equations using the divergence of the vector field.

The divergence element are available with all types of mesh elements. The polynomial order of the divergence element can be at most three in 3D, and at most four in 2D and 1D.

Syntax for Divergence Elements (shdiv)

Specify divergence shape functions in the `model.shape` field of the model object. The constructor of the divergence shape function is `shdiv`. The following properties are allowed:

TABLE 5-9: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHDIV SHAPE FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
fieldname	variable name		Name of vector field
compnames	string array	Derived from fieldname	Names of vector field components
dofbasename	string	see below	Base name of degrees of freedom
divname	string	see below	Name of divergence field
order	integer	1	Basis function order
sorder	positive integer	Given by f frame	Geometry shape order

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```

model.shape().create("shu","f");
model.shape("shu").create("f1","shdiv");
model.shape("shu").feature("f1").set("order",2);
model.shape("shu").feature("f1").set("compnames",new String[]{"Bx","By"});
model.shape("shu").feature("f1").set("dofbasename","nB");

```

The default for `compnames` is `fieldname` concatenated with the spatial coordinate names. The default for `dofbasename` is `nallcomponents`, where `allcomponents` is the concatenation of the names in `compnames`.

The vector element defines the following degrees of freedom: `dofbasename` on element boundaries, and `dofbasename sdim c, c = 0, ..., sdim - 1` for DOFs in the interior.

The divergence element defines the following field variables (where `comp` is a component name from `compnames`, `divname` is the `divname`, `sdim` = space dimension and `edim` = mesh element dimension):

- `comp`, meaning a component of the vector, defined when `edim = sdim`.
- `ncomp`, meaning one component of the projection of the vector onto the normal of mesh element, defined when `edim = sdim - 1`.
- `compx`, meaning the derivative of a component of the vector with respect to global spatial coordinate x , defined when `edim = sdim`.
- `ncompTx`, the tangential derivative variable, meaning the x component of the projection of the gradient of `ncomp` onto the mesh element, defined when `edim < sdim`. Here, x is the name of a spatial coordinate. `ncompTx` = 0.
- `divname`, means the divergence of the vector field.

For performance reasons, prefer using `divname` in expressions involving the divergence rather than writing it as the sum of `sdim` gradient components.

For the computation of components, the global spatial coordinates are expressed as polynomials of degree (at most) `sorder` in the local coordinates.

SCALAR PLANE WAVE BASIS FUNCTION

The scalar plane wave basis function, `shuwhelm`, is used to implement scalar plane wave basis functions for solving scalar wave equations of Helmholtz type using an *ultraweak variational formulation* (UWVF). These basis functions are discontinuous in between mesh elements.

Syntax for the Scalar Plane Wave Basis Function (`shuwhelm`)

Specify scalar plane wave basis functions in the `model.shape` field of the model object. The constructor of the density shape function is `shuwhelm`. The following properties are allowed:

TABLE 5-10: VALID PROPERTY NAME/VALUE PAIRS FOR THE SHUWHELM WAVE BASIS FUNCTION

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>basename</code>	variable name		Base variable name
<code>ndir</code>	integer		Number of wave directions
<code>kexpr</code>	string		Variable for the wave number
<code>xexpr</code>	string array	{ 'x', 'y', 'z' }	Expressions for the x, y, and z coordinate transformations

The property names cannot be abbreviated and must be written in lowercase letters enclosed in quotation marks.

```

model.shape().create("shu","f");
model.shape("shu").create("f1","shuwhelm");
model.shape("shu").feature("f1").set("ndir",2);
model.shape("shu").feature("f1").set("basename","u");

```

The scalar plane wave basis function defines the following field variables. Denote `basename` with u , and let x denote the spatial coordinates. The variables are (`sdim` = space dimension):

- u
- ux , meaning the derivative of u with respect to x , defined on `edim` = `sdim`

6

Solver

This chapter contains reference information about the solver command and utility commands for producing and handling solutions.


In this chapter:

- [About Solver Commands](#)
- [Solution Object Data](#)

About Solver Commands

The following list includes the solver commands that are documented in this chapter (listed in alphabetical order):

- [Adaption](#)
- [Advanced](#)
- [Assemble](#)
- [AutoRemesh](#)
- [AWE](#)
- [Eigenvalue](#)
- [FFT](#)
- [For, EndFor](#)
- [FullyCoupled](#)
- [InputMatrix](#)
- [Linear](#)
- [Modal](#)
- [Optimization](#)
- [Parametric](#)
- [Previous Solution](#)
- [Segregated](#)
- [SegregatedStep](#)
- [Sensitivity](#)
- [StateSpace](#)
- [Stationary](#)
- [StopCondition](#)
- [StoreSolution](#)
- [StudyStep](#)
- [Time](#)
- [TimeDiscrete](#)
- [TimeExplicit](#)
- [Variables](#)
- [XmeshInfo](#)

	<ul style="list-style-type: none">• Features Producing and Manipulating Solutions• Features with Solver Settings• Solution Object Information Methods• Solution Feature Information Methods• Studies and Solvers in the <i>COMSOL Multiphysics Reference Manual</i>
-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Features Producing and Manipulating Solutions

[Table 6-1](#) is an overview of the available features for producing and handle solution objects.

TABLE 6-1: SOLUTION OBJECT FEATURES

FEATURE	PURPOSE
Assemble	Assembles and stores the matrices generated during assembly
AWE	Solve parametric problem with asymptotic waveform evaluation
Eigenvalue	Solve eigenvalue problem
FFT	Perform a forward or inverse fast Fourier transform (FFT or IFFT)
InputMatrix	Input matrices or vectors, for example load vectors or stiffness matrices, to a solver.
Modal	Solve time-dependent or parametric problem with modal analysis
Optimization	Solve optimization problem
StateSpace	Assembles and stores matrices that describe a model as a dynamic system
Stationary	Solve stationary problem
StudyStep	Specifies which problem to compile
Time	Solve time-dependent problem with implicit time stepping

TABLE 6-1: SOLUTION OBJECT FEATURES

FEATURE	PURPOSE
TimeDiscrete	Solve time-dependent problem with user's own time stepping
TimeExplicit	Solve time-dependent problem with explicit time stepping
Variables	Handle variables solved for (initial values, scaling) and not solved for (prescribed values)

Features with Solver Settings

[Table 6-2](#) is an overview of the available features for solver settings.

TABLE 6-2: SOLVER SETTING FEATURES

FEATURE	SETTINGS HANDLED
Adaption	Solve with adaptive mesh refinement
Advanced	Advanced general settings
AutoRemesh	Automatically remesh deformed geometries
ControlField	Control fields (a set of control variables)
ControlState	Set of global control variables
Field	Fields (a set of dependent variables)
FullyCoupled	Fully coupled nonlinear solution approach
Parametric	Parameter stepping
Previous Solution	Previous solution solvers
Segregated	Segregated nonlinear solution approach
SegregatedStep	Segregated steps
Sensitivity	Sensitivity analysis
StopCondition	Stop conditions
See Linear for the following:	
Direct	Direct linear system solvers
DirectPreconditioner	Direct linear system solvers as preconditioners
DomainDecomposition	Domain decomposition solver
IncompleteLU	Incomplete factorization preconditioners
Iterative	Iterative linear system solvers
Jacobi	Jacobi linear system preconditioners
KrylovPreconditioner	Krylov linear system preconditioners
Multigrid	Multigrid linear system preconditioners
SOR	SOR linear system preconditioners
SORGauge	SOR Gauge linear system preconditioners
SORLine	SOR Line linear system preconditioners
SORVector	SOR Vector linear system preconditioners
State	Sets of global dependent variables
Vanka	Vanka linear system preconditioners

Solution Object Information Methods

The following tables are an overview of the solution object information methods.

GENERAL INFORMATION

TABLE 6-3: GENERAL SOLUTION INFORMATION METHODS

METHOD	DESCRIPTION
getType	Get solution type
getSize	Get number of dynamic solutions and length of solution vector
getSizeMulti	Get number of local solution objects and total number of solutions
getMesh	Get mesh name associated with solution and geometry
getNU	Get number of solutions of a certain solution data type
getPNames	Get parameter names
getParamName	Get parameter names for parametric sweep
getParamVals	Get parameter values for parametric sweep

SOLUTION DATA

TABLE 6-4: SOLUTION DATA ACCESS METHODS

METHOD	DESCRIPTION
getU	Get real part of solution vector
getUDot	Get real part of the first time-derivative solution vector
getUImag	Get imaginary part of solution vector
getUDotImag	Get imaginary part of first time-derivative solution vector
getPVals	Get the real part of the parameter values
getPValsImag	Get the imaginary part of the parameter values
getUBlock	A blocked version of the getU method
getUDotBlock	A blocked version of the getUDot method
getUImagBlock	A blocked version of the getUImag method
getUDotImagBlock	A blocked version of the getUDotImag method

SOLUTION CREATION

TABLE 6-5: SOLUTION CREATION METHODS

METHOD	DESCRIPTION
setU	Set real part of solution vector
setUDot	Set real part of the first time-derivative solution vector
setUImag	Set imaginary part of solution vector
setUDotImag	Set imaginary part of first time-derivative solution vector
setPVals	Set the real part of the parameter values
setPValsImag	Set the imaginary part of the parameter values
setUBlock	A blocked version of the setU method
setUDotBlock	A blocked version of the setUDot method
setUImagBlock	A blocked version of the setUImag method
setUDotImagBlock	A blocked version of the setUDotImag method

Solution Feature Information Methods

GENERAL INFORMATION

TABLE 6-6: GENERAL MATRIX INFORMATION METHODS

METHOD	DESCRIPTION
isReal	Check if matrix is real
getM	Get number of rows
getN	Get number of columns
getNnz	Get number of nonzeros in sparse matrix

MATRIX DATA

TABLE 6-7: MATRIX ACCESS METHODS

METHOD	DESCRIPTION
getSparseMatrixVal	Get matrix values
getSparseMatrixValImag	Get the imaginary matrix values
getSparseMatrixCol	Get column indices of matrix values
getSparseMatrixRow	Get row indices of matrix values
getVector	Get the vector associated with the matrix type
getVectorImag	Get the imaginary part of the vector associated with the matrix type
getSparseMatrixValBlock	A blocked version of getSparseMatrixVal
getSparseMatrixValImagBlock	A blocked version of getSparseMatrixValImag
getSparseMatrixColBlock	A blocked version of getSparseMatrixCol
getSparseMatrixRowBlock	A blocked version of getSparseMatrixRow
getVectorBlock	A blocked version of getVector
getVectorImagBlock	A blocked version of getVectorImag

MATRIX CREATION

TABLE 6-8: MATRIX CREATION METHODS

METHOD	DESCRIPTION
createSparseMatrix	Create sparse matrix
addSparseMatrixVal	Add matrix values to the created matrix
addSparseMatrixValImag	Add imaginary matrix values to the created matrix
createVector	Create vector
setVector	Set the vector associated with the matrix type
setVectorImag	Set the imaginary part of the vector associated with the matrix type
setVectorBlock	A blocked version of setVector
setVectorImagBlock	A blocked version of setVectorImag

Solution Object Data

The solver sequence works as a solution object itself. The solution object data produced by running the sequence (partially or in whole) can be obtained by a number of access methods on the sequence. See [Table 6-3](#) and [Table 6-5](#) for an overview.

In this section:

- [General Information](#)
- [Solution Data](#)
- [SolutionInfo Object and Its Methods](#)
- [Solution Creation](#)
- [General Matrix Information](#)
- [Matrix Data](#)
- [Matrix Creation](#)

General Information

TABLE 6-9: GENERAL SOLUTION OBJECT INFORMATION METHODS

METHOD	OUTPUT TYPE
<code>isRealU()</code>	Boolean
<code>isRealU(int)</code>	Boolean
<code>isRealU(int,string)</code>	Boolean
<code>isRealU(int,string,int)</code>	Boolean
<code>isRealUDot()</code>	Boolean
<code>isRealUDot(int)</code>	Boolean
<code>isRealPVals()</code>	Boolean
<code>getType()</code>	String
<code>getSize()</code>	int[2]
<code>getSize(int)</code>	int[2]
<code>getSizeMulti()</code>	int[2]
<code>getMesh(string)</code>	String
<code>getMesh(string,int)</code>	String
<code>getNU(String)</code>	int
<code>getPNames()</code>	String[]
<code>getPVals()</code>	double[]
<code>getParamNames()</code>	String[]
<code>getParamVals()</code>	double[]
<code>getSolutioninfo()</code>	SolutionInfo

- `model.sol(<tag>).isRealU()` returns true if the solution is real.
- `model.sol(<tag>).isRealU(<solnum>)` returns true if solution <solnum> is real.
- `model.sol(<tag>).isRealU(<solnum>,<uType>)` returns true if the solution <solnum> of type <uType> is real.
- `model.sol(<tag>).isRealU(<solnum>,<uType>,<uNum>)` returns true if the solution <solnum> of type <uType> and solution index <uNum> is real.

- `model.sol(<tag>).isRealUDot()` returns true if the first time derivative is real.
- `model.sol(<tag>).isRealUDot(<solnum>)` returns true if first time derivative <solnum> is real.
- `model.sol(<tag>).isRealPVals()` returns true if the parameter values are real.
- `model.sol(<tag>).isRealU()` returns true if the solution vector is real.
- `model.sol(<tag>).getType()` returns a string for the solution type which can be any of the strings; Stationary, Parametric, Time, Eigenvalue, and None.
- `model.sol(<tag>).getSequenceType()` returns a string for the solver sequence type, which can be any of the strings SolverSequence, CopySolution, ParametricStore, Stored, Parametric, and None.
- `model.sol(<tag>).getSize()` returns an array of sizes for the solution data. The number of degrees of freedoms is stored in the first position and the number of solutions (solnums) in the second.
- `model.sol(<tag>).getSize(<iMulti>)` returns an array of sizes for the solution number <iMulti> of the multi-solution. The number of degrees of freedoms is stored in the first position and the number of solutions (solnums) in the second.
- `model.sol(<tag>).getSizeMulti()` returns an array of sizes for the multi-solution. The number of local solution objects is stored in the first position and the total number of solutions (solnums) in the second.
- `model.sol(<tag>).getMesh(<geom>)` returns the mesh name associated with the solution and the geometry <geom>.
- `model.sol(<tag>).getMesh(<geom>,<iMulti>)` returns the mesh name associated with the solution number <iMulti> of the multi solution and the geometry <geom>.
- `model.sol(<tag>).getNU(<uType>)` returns the number of solutions stored of the type <uType>. Here <uType> is the solution type as a string: Sol (main solution), Reacf (reaction force), Adj (adjoint solution), Fsens (functional sensitivity), and Sens (forward sensitivity).
- `model.sol(<tag>).getPNames()` returns the parameter names from continuation solver as an array of strings.
- `model.sol(<tag>).getParamNames()` returns the parameter names from a parametric sweep as an array of strings.
- `model.sol(<tag>).getParamVals()` returns the parameter values from a parametric sweep as an array of double.

Solution Data

TABLE 6-10: SOLUTION DATA ACCESS METHODS, REAL PART

METHOD	OUTPUT TYPE
<code>getU(int,string,int)</code>	<code>double[]</code>
<code>getU(int,string)</code>	<code>double[]</code>
<code>getU(int)</code>	<code>double[]</code>
<code>getU()</code>	<code>double[]</code>
<code>getUDot()</code>	<code>double[]</code>
<code>getUDot(int)</code>	<code>double[]</code>
<code>getPVals()</code>	<code>double[]</code>
<code>getPVals(int)</code>	<code>double[]</code>

- `model.sol(<tag>).getU(<solnum>,<uType>,<uNum>)` returns the real part of the solution vector for solution number <solnum>, the solution type <uType>, and the 1-based solution index <uNum>. Here, $1 \leq \text{<uNum>} \leq N$, where $N = \text{model.sol}(\text{<tag>}).\text{getNU}(\text{<uType>})$.

- `model.sol(<tag>).getU(<solnum>,<uType>)` returns the real part of the solution vector for the solution number `<solnum>` and the solution data type `<uType>`. The solution index `<uNum>=1`.
- `model.sol(<tag>).getU(<solnum>)` returns the real part of the solution vector for solution number `<solnum>`. The solution data type `<uType>=Sol` and the solution index `<uNum>=1`.
- `model.sol(<tag>).getU()` returns the real part of the solution vector. For a Time-dependent and Parametric type, the last solution number is used, and for a Eigenvalue type the first solution number. The solution data type `<uType>=Sol` and the solution index `<uNum>=1`.
- `model.sol(<tag>).getUDot()` returns the real part of the first time-derivative solution vector for a Time-dependent type and if the time-derivatives have been stored. The last solution number is used. For other types and if the time derivatives have not been stored, an error message is given.
- `model.sol(<tag>).getUDot(<solnum>)` returns the real part of the first time-derivative solution vector for the solution number `<solnum>`.
- `model.sol(<tag>).getPVals()` returns for a solution of a Parametric type the real part of all the parameter values stored. For multiple parameters all the parameter tuples are concatenated. For a solution of a Time-dependent type, this is the times for which solution data is stored. For a solution of an Eigenvalue type, this is the real part of the eigenvalues stored. For a Time-dependent and Parametric type, the last solution number is used, and for a Eigenvalue type the first solution number.
- `model.sol(<tag>).getPVals(<solnum>)` returns for a solution of a Parametric type the real part of the parameter tuples stored for solution number `<solnum>`. For a solution of a Time-dependent type, this is the time for solution number `<solnum>`. For a solution of an Eigenvalue type, this is the real part of the eigenvalue stored at solution number `<solnum>`.

TABLE 6-11: SOLUTION DATA ACCESS METHODS, IMAGINARY PART

METHOD	OUTPUT TYPE
<code>getUImag()</code>	<code>double[]</code>
<code>getUImag(int)</code>	<code>double[]</code>
<code>getUImag(int,string)</code>	<code>double[]</code>
<code>getUImag(int,string,int)</code>	<code>double[]</code>
<code>getUDotImag()</code>	<code>double[]</code>
<code>getUDotImag(int)</code>	<code>double[]</code>
<code>getPValsImag()</code>	<code>double[]</code>
<code>getPValsImag(int)</code>	<code>double[]</code>

- `model.sol(<tag>).getUImag()` returns the imaginary part of the solution vector. The same `<solnum>`, `<uType>`, and `<uNum>` is used as for the method `getU()`. And similarly for the other `Imag` methods.

TABLE 6-12: SOLUTION DATA ACCESS METHODS, BLOCKED VERSIONS

METHOD	OUTPUT TYPE
<code>getUBlock(int,int)</code>	<code>double[]</code>
<code>getUBlock(int,int,int)</code>	<code>double[]</code>
<code>getUBlock(int,string,int,int)</code>	<code>double[]</code>
<code>getUBlock(int,string,int,int,int)</code>	<code>double[]</code>
<code>getUDotBlock(int,int)</code>	<code>double[]</code>
<code>getUDotBlock(int,int,int)</code>	<code>double[]</code>
<code>getUImagBlock(int,int)</code>	<code>double[]</code>
<code>getUImagBlock(int,int,int)</code>	<code>double[]</code>
<code>getUImagBlock(int,string,int,int)</code>	<code>double[]</code>
<code>getUImagBlock(int,string,int,int,int)</code>	<code>double[]</code>
<code>getUDotImagBlock(int,int)</code>	<code>double[]</code>
<code>getUDotImagBlock(int,int,int)</code>	<code>double[]</code>

- `model.sol(<tag>).getUBlock(<startpos>,<endpos>)` returns a subset of the vector returned by `getU()`, the subarray from position `<startpos>` to the position `<endpos>`. And similarly for the other `Block` methods.

SolutionInfo Object and Its Methods

For each solver sequence, there is an associated `SolutionInfo` object, which can be accessed by the function `getSolutioninfo()`. This object has several methods to access the parameter sweep generated solution data. Parametric sweep generated solution data is normally stored in solver sequences of type `Parametric` or of type `SolverSequence`, depending on whether so-called outer parametric sweeps (see `model.jobs()`) have been used or not. It can also be used to convert between so-called loop-level settings and solution numbers. The following methods in [Table 6-13](#) are supported.

TABLE 6-13: SOLUTIONINFO METHODS

METHOD	OUTPUT TYPE
<code>getIndicies(int, int[])</code>	<code>int[]</code>
<code>getISol(int, double)</code>	<code>int</code>
<code>getISol(int, int)</code>	<code>int[]</code>
<code>getLevelDescription(int)</code>	<code>String</code>
<code>getLevelNames()</code>	<code>String[]</code>
<code>getLevels()</code>	<code>int</code>
<code>getMaxInner(int[])</code>	<code>int</code>
<code>getMaxLevels()</code>	<code>int</code>
<code>getName(int)</code>	<code>String</code>
<code>getOuterSolnum()</code>	<code>int[]</code>
<code>getPNamesOuter()</code>	<code>String[]</code>
<code>getPUnitsOuter()</code>	<code>String[]</code>
<code>getSol(int)</code>	<code>String</code>
<code>getSolDescriptions(int, int[], boolean, boolean)</code>	<code>String[]</code>
<code>getSolnum(int, boolean)</code>	<code>int[]</code>

TABLE 6-13: SOLUTIONINFO METHODS

METHOD	OUTPUT TYPE
<code>getSplitLevelDescriptions()</code>	<code>String[]</code>
<code>getSplitLevelNames()</code>	<code>String[]</code>
<code>getSplitLevelUnits()</code>	<code>String[]</code>
<code>getSplitName(int)</code>	<code>String[]</code>
<code>getSplitUnit(int)</code>	<code>String[]</code>
<code>getUnit(String)</code>	<code>String</code>
<code>getVals(int, int[])</code>	<code>double[][]</code>
<code>isStructured()</code>	<code>Boolean</code>
<code>isValid()</code>	<code>Boolean</code>
<code>mapToLevel(int[], int[], boolean)</code>	<code>int[][]</code>
<code>mapToSolnum(int[][], boolean)</code>	<code>int[][]</code>

```
info = model.sol(<tag>).getSolutioninfo()
```

- `getIndicies(int level, int[] levels)` Returns the one-based indices available for the loop level, `level` (index zero based). The current level setting can be given in `levels` (index one based). The returned values are $1, \dots, N$ where N is the number of values or tuples for the given level. When `level=getMaxLevel()-1` the indices can be the result of an outer product between levels. When the format is unstructured, and when `levels` is set, then the unstructured list of indices is returned. When the format is unstructured, and when `levels=null` or `levels.length=0`, an error is given if `level` is such that there are no structured data to return (currently `level=0` and multiple inner parameter names).
- `getISol(int outersolnum, int innersolnum)` returns the index zero based multi-solution object number and the index zero based solution number within it, for the index one based outer and inner solution numbers. The solution object number is returned in the first position and the corresponding solution number in the second. The solution object number is normally the same (0) for all `innersolnum`, but can vary for time-dependent adaptation or for automatic remeshing.
- `getISol(int outersolnum, double t)` returns the index-zero based multi solution object number for the one-based outer solution number `outersolnum` and time value `t`. The returned solution number is normally the same (0) for all `t`, but can vary for time-dependent adaptation or for automatic remeshing.
- `getLevelDescription(int level)` returns a description of the index zero based loop level, `level`.
- `getLevelNames()` returns the names of the different loop levels. Some of these can be a concatenated string such as "p1,p2".
- `getLevels()` returns the number of loop levels, `getLevels()<=getMaxLevels()`.
- `getMaxInner(int[] outersolnum)` returns the maximum number of inner solutions for the given index one based outer solution numbers. If `outersolnum` is null, the maximum is taken over all outer solutions.
- `getMaxLevels()` returns the maximum number of used loop levels, `getLevels()<=getMaxLevels()`.
- `getName(int level)` returns the parameter name for the index zero based loop level `level`. This name can be a concatenated string such as "p1, p2".
- `getOuterSolnum()` returns the one based indices for the outer solutions. If there are no outer parameters or added corresponding parameter values the array is empty.
- `getPNamesOuter()` returns the subset of parameter names that are looped by a job sequence parametric sweep.
- `getSol(int outersolnum)` returns the solver sequence tag for the index one based outer solution number `outersolnum`. If the solution number is invalid, null is returned.
- `getSolDescriptions(int level, int[] levels, boolean paramInclusion, boolean indexInclusion)` returns the descriptions for the solutions for the index zero based loop level, `level`. The current level setting

can be given in `levels` (index one based). One string for each solution is returned. When `paramInclusion` is true, the description always includes the parameter name, even if this level contains only one. When `paramInclusion` is false, the parameter name is only included when there is more than one parameter name on this level. When the format is unstructured, and when `levels` is set, then the unstructured list of descriptions is returned. When the format is unstructured, and when `levels==null` or `levels.length==0`, an error is given if `level` is such that there are no structured data to return.

- `getSolnum(int outersolnum, boolean strict)` returns the one based inner solution numbers for the index one based outer solution number `outersolnum`. If `strict` is true the inner solution numbers is returned if `outersolnum` is a valid outer solution number and else a zero array is returned. If `strict` is false and if the `outersolnum` does not match, then the solution numbers for the containing solution object is returned.
- `getSplitLevelDescriptions()` returns the description of the different parameters, split into an array for the case when there is more than one parameter for a loop level.
- `getSplitLevelNames()` returns the names of the different parameters, split into an array for the case when there's more than one parameter for a looplevel.
- `getSplitNames(int level)` returns the parameter names for the index zero based looplevel `level`.
- `getUnit(String name)` returns the unit of the sweep parameter name.
- `getVals(int level, int[] levels)` returns the parameter values for the index zero based loop level, `level`. The current level setting can be given in `levels` (index one based). For `level < getMaxLevels() - 1` this is just the values of the parameters for this level. The number of rows is the same as the number of parameters for this level. The columns are the values. For `level = getMaxLevels() - 1` the values are expanded into tuples for the case that levels have been merged. When the format is unstructured, and when `levels` is set, then the unstructured lists of values are returned. When the format is unstructured, and when `levels==null` or `levels.length=0`, an error is given if `level` is such that there are no structured data to return.
- `isStructured()` returns true unless the underlying solution object/objects has a parameter variation that depends on the solution process itself. Examples are time-dependent simulations where the output is determined by the steps taken by the solver or eigenvalue simulations.
- `isValid()` returns true if the underlying solution data is consistent with this info object.
- `mapToLevel(int[] outersolnum, int[] innersolnum, boolean compressedOutput)` returns the index one based level representation of the index-one based outer and inner solution numbers, `outersolnum` and `innersolnum` respectively. The number of rows of the returned data is equal to the number of levels. When `compressedOutput` is false, the columns represent the tuples, which is the most general format. When `compressedOutput` is true, the level settings are made unique on each level.



When `compressedOutput` is true and if the compressed representation does not match the input, an array with the right number of rows, but each with zero length, are returned.

- `mapToSolnum(int[][] levelSetting, boolean expandInput)` returns the one based solution number representation of a loop level setting `levelSetting`. The first row in the output is the inner and the second the outer solution numbers. The `levelSettings` must have the same number of rows as there are levels. On each row, index one based settings for each level should be given. If `expandInput` is false the number of columns must be the same and the columns are treated as level-tuples. If `expandInput` is true, the number of columns can be different and the output is expanded to the outer product of each levels setting.

TABLE 6-14: SOLUTION CREATION METHODS, REAL PART

METHOD	OUTPUT TYPE
setU(double[])	
setU(int,double[])	
setPNames(String[])	
setPVals(double[])	
setPVals(int,double[])	
createSolution()	

- `model.sol(<tag>).setU(<vals>)` sets the real part of the solution vector to `<vals>`
- `model.sol(<tag>).setU(<solnum>,<vals>)` sets the real part of solution vector `<solnum>`, to `<vals>`.
- `model.sol(<tag>).setPNames(<pnames>)` sets parameter names of the solution vectors to `<pnames>`.
- `model.sol(<tag>).setPVals(<vals>)` sets the parameter values to `<vals>`.
- `model.sol(<tag>).setPVals(<solnum>,<vals>)` sets the parameter values `<solnum>` to `<vals>`.
- `model.sol(<tag>).createSolution()` creates solutions based on the input from the vectors previously set. The solution is created at this stage. Afterward the user input is cleared. If a created solution is used before this function is run the result is unpredictable.

TABLE 6-15: SOLUTION CREATION METHODS, IMAGINARY PART

METHOD	OUTPUT TYPE
setUImag(double[])	
setUImag(int,double[])	
setPValsImag(double[])	
setPValsImag(int,double[])	

- `model.sol(<tag>).setUImag(<solnum>,<vals>)` sets the imaginary part of solution vector `<solnum>` to `<vals>` (and similarly for the other Imag methods).

TABLE 6-16: SOLUTION CREATION METHODS, REAL PART

METHOD	OUTPUT TYPE
setUBlock(double[],int)	
setUBlock(int,double[],int)	
setUImagBlock(double[],int)	
setUImagBlock(int,double[],int)	
setUDotBlock(double[],int)	
setUDotBlock(int,double[],int)	
setUImagBlock(double[],int)	
setUImagBlock(int,double[],int)	

- `model.sol(<tag>).setUBlock(<solnum>,<vals>,<start>)` sets the real part of solution vector `<solnum>`, the subarray from position `<start>` to position `<start>+<vals>.length-1` to `<vals>` (and similarly for the other Imag methods).

General Matrix Information

TABLE 6-17: GENERAL MATRIX OBJECT INFORMATION METHODS

METHOD	OUTPUT TYPE
<code>isReal(String)</code>	Boolean
<code>getM(String)</code>	int
<code>getN(String)</code>	int
<code>getNnz(String)</code>	int

- `model.sol(<tag>).feature(<ftag>).isReal(<mname>)` returns true if the matrix <mname> is real.
- `model.sol(<tag>).feature(<ftag>).getM(<mname>)` returns number of rows in the matrix <mname>.
- `model.sol(<tag>).feature(<ftag>).getN(<mname>)` returns number of columns in the matrix <mname>.
- `model.sol(<tag>).feature(<ftag>).getNnz(<mname>)` returns number of nonzero entries in the matrix <mname>.

Matrix Data

TABLE 6-18: MATRIX DATA ACCESS METHODS, REAL PART

METHOD	OUTPUT TYPE
<code>getSparseMatrixVal(String)</code>	double[]
<code>getSparseMatrixCol(String)</code>	int[]
<code>getSparseMatrixRow(String)</code>	int[]
<code>getVector(String)</code>	double[]

- `model.sol(<tag>).feature(<ftag>).getSparseMatrixVal(<mname>)` returns the real part of the sparse matrix values of matrix <mname> stored in feature <ftag>. Here, <mname>, is one of “K”, “D”, “E”, “N”, “NF”, “NP”, “Kc”, “Dc”, “Ec”, “Null”, “Nullf”, “Mc”, “MA”, “MB”, “C”.
- `model.sol(<tag>).feature(<ftag>).getSparseMatrixCol(<mname>)` returns the column numbers of the sparse matrix values of matrix <mname> stored in feature <ftag>. Here, <mname>, is one of “K”, “D”, “E”, “N”, “NF”, “NP”, “Kc”, “Dc”, “Ec”, “Null”, “Nullf”, “Mc”, “MA”, “MB”, “C”.
- `model.sol(<tag>).feature(<ftag>).getSparseMatrixRow(<mname>)` returns the row numbers of the sparse matrix values of matrix <mname> stored in feature <ftag>. Here, <mname>, is one of “K”, “D”, “E”, “N”, “NF”, “NP”, “Kc”, “Dc”, “Ec”, “Null”, “Nullf”, “Mc”, “MA”, “MB”, “C”.
- `model.sol(<tag>).feature(<ftag>).getVector(<mname>)` returns the real part of the vector <mname> stored in feature <ftag>. Here, <mname>, is one of “L”, “M”, “MP”, “MLB”, “MUB”, “ud”, “uscale”, “x0”.

TABLE 6-19: MATRIX DATA ACCESS METHODS, IMAGINARY PART

METHOD	OUTPUT TYPE
<code>getSparseMatrixValImag(String)</code>	double[]
<code>getVectorImag(String)</code>	double[]

- `model.sol(<tag>).feature(<ftag>).getSparseMatrixValImag(<mname>)` returns the imaginary part of the sparse matrix values of matrix <mname> stored in feature <ftag>. Here, <mname>, is one of “K”, “D”, “E”, “N”, “NF”, “NP”, “Kc”, “Dc”, “Ec”, “Null”, “Nullf”, “Mc”, “MA”, “MB”, “C”.

- `model.sol(<tag>).feature(<ftag>).getVectorImag(<mname>)` returns the imaginary part of the vector `<mname>` stored in feature `<ftag>`. Here, `<mname>`, is one of “L”, “M”, “MP”, “MLB”, “MUB”, “ud”, “uscale”, “x0”.

TABLE 6-20: MATRIX DATA ACCESS METHODS, BLOCKED VERSIONS

METHOD	OUTPUT TYPE
<code>getSparseMatrixValBlock(String,int,int)</code>	<code>double[]</code>
<code>getSparseMatrixColBlock(String,int,int)</code>	<code>int[]</code>
<code>getSparseMatrixRowBlock(String,int,int)</code>	<code>int[]</code>
<code>getVectorBlock(String,int,int)</code>	<code>double[]</code>
<code>getSparseMatrixValImagBlock(String,int,int)</code>	<code>double[]</code>
<code>getVectorImagBlock(String,int,int)</code>	<code>double[]</code>

- `model.sol(<tag>).feature(<ftag>).getSparseMatrixValBlock(<mname>, <startpos>,<endpos>)` returns a subset of the real part of the sparse matrix values returned by `getSparseMatrixVal(<mname>)`, the subarray from the position `<startpos>` to the position `<endpos>`. And similarly for the other **Block** methods.
- `model.sol(<tag>).feature(<ftag>).getVectorBlock(<mname>,<vals>)` returns a subset of the real part of the vector values returned by `getVector(<mname>)`, the subvector from the position `<startpos>` to the position `<endpos>`. Here, `<mname>`, is one of “L”, “M”.

Matrix Creation

TABLE 6-21: MATRIX DATA CREATION METHODS, REAL PART

METHOD	OUTPUT TYPE
<code>createSparseMatrix(String,int,int,int,boolean)</code>	<code>void</code>
<code>addSparseMatrixVal(String,int[],int[],double[])</code>	<code>void</code>
<code>createVector(String,int,boolean)</code>	<code>void</code>
<code>setVector(String,double[])</code>	<code>void</code>

- `model.sol(<tag>).feature(<ftag>).createSparseMatrixVal(<mname>,<M>,<N>,<Nnz>,<isReal>)` creates a sparse matrix `<mname>` stored in feature `<ftag>`. Here, `<mname>`, is one of “K”, “D”, “E”, “N”, “NF” and `<M>` is the number of rows, `<N>` is the number of columns, `<Nnz>` is the number of nonzeros, and `<isReal>` is true if the matrix is zero.
- `model.sol(<tag>).feature(<ftag>).addSparseMatrixVal(<mname>,<row>,<col>,<val>)` adds the values stored in `<val>` to the sparse matrix `<mname>` stored in feature `<ftag>`. Here, `<mname>`, is one of “K”, “D”, “E”, “N”, “NF”, and `<row>` is the rows, `<col>` is the columns, and `<val>` is the values of the entries.
- `model.sol(<tag>).feature(<ftag>).createVector(<mname>,<M>,<isReal>)` creates a vector `<mname>` stored in feature `<ftag>`. Here, `<mname>`, is one of “L”, “M”, and `<M>` is the size of the vector and `<isReal>` is true if the vector is real.
- `model.sol(<tag>).feature(<ftag>).setVector(<mname>,<val>)` sets the real part of the vector `<mname>` stored in feature `<ftag>`. Here, `<mname>`, is one of “L”, “M”, and `<val>` is the values to store in the vector.

TABLE 6-22: MATRIX DATA CREATION METHODS, IMAGINARY PART

METHOD	OUTPUT TYPE
<code>addSparseMatrixValImag(int[],int[],double[])</code>	<code>void</code>
<code>setVectorImag(String,double[])</code>	<code>void</code>

- `model.sol(<tag>).feature(<ftag>).addSparseMatrixValImag(<mname>,<M>,<N>,<Nnz>,<isReal>)` creates the imaginary part of the sparse matrix values of matrix `<mname>` stored in feature `<ftag>`. Here,

<mname>, is one of “K”, “D”, “E”, “N”, “NF”, and <row> is the rows, <col> is the columns, and <val> is the imaginary values of the entries.

- `model.sol(<tag>).feature(<ftag>).setVectorImag(<mname>,<val>)` sets the imaginary part of the vector <mname> stored in feature <ftag>. Here, <mname>, is one of “L”, “M”, and <val> is the values to store in the vector.

TABLE 6-23: MATRIX DATA CREATION METHODS, BLOCKED VERSIONS

METHOD	OUTPUT TYPE
<code>setVectorBlock(String,double[],int)</code>	void
<code>setVectorImagBlock(String,double[],int)</code>	void

- `model.sol(<tag>).feature(<ftag>).setVectorBlock(<mname>,<vals>,<startpos>)` sets a subset of the real part of the vector values set by `setVector(<mname>,<vals>)`, the subvector from the position <startpos>. Here, <mname>, is one of “L”, “M”.

Adaption

Handle adaptive mesh refinement parameters.

SYNTAX

```
model.sol(sname).feature(solv).create(fname,"Adaption")
model.sol(sname).feature(solv).create(fname,"TimeAdaption")
model.sol(sname).feature(solv).feature(fname).set(pname,value)
```

DESCRIPTION

Handles settings for adaptive mesh refinement. There are two sorts of features for adaptive mesh refinement: Adaption, which can be added to solvers of Eigenvalue or Stationary types, and TimeAdaption, which can be added to a solver of the Time Dependent type.

The Adaption feature runs an overall adaptation iteration a number of steps, and in each iteration a new mesh is generated and solved for. The last mesh generated and the solution on this mesh is added to the model.

The TimeAdaption feature splits the overall time range into subintervals and in each interval an adapted mesh is generated and used. The meshes for these intervals as well as the solutions are added to the model. The solutions are stored in one container node (`model.sol()`) to facilitate the result processing.

The Adaption feature accepts the following property/value pairs

TABLE 6-24: VALID ADAPTION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
adjppr	auto on off	auto	Use recovery for adjoint solution error estimate
eefun	12 func	12	Error estimation function
eefunc	String		Error estimate functional name (eefun=func)
eigselect	vector of positive scalars	1	Weights for eigenmodes
elementspar	positive scalar		Controls refinement if elselect=elements
elselect	globalmin worst elements		Method for selecting elements to refine.
geomnum	String		Name of geometry sequence
globalminpar	positive scalar		Controls refinement if elselect=globalmin
l2scale	vector of positive scalars	1	Scale factors for the L2 error norm

TABLE 6-24: VALID ADAPTION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
l2staborder	vector of positive integers	2	Orders in the stability estimate for the L2 error estimate
maxt	positive scalar	Inf	Maximum number of mesh elements
ngen	scalar integer	5 (1D) 2 (2D) 1 (3D)	Maximum number of refinements
plot	on off	off	Plot while solving
plotgroup	String	default	Plot group to use for plot while solving
resorder	auto scalar vector	auto	Order of decrease of equation residuals
resorderactive			
rmethod	regular longest meshinit	longest	Refinement method
worstpar	positive scalar		Controls refinement if elselect=worst

The feature TimeAdaption accepts the following property/value pairs

TABLE 6-25: VALID TIMEADAPTION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
consistentrestart	on off	off	Consistent initialization after restart
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");)
convertmesh	on off	on	Convert to simplex mesh
eefuntime	user	user	Error indicator function
eefunctime	String		Error indicator name (eefuntime=user)
elfrac	positive scalar	0.2	Fraction of maximum refinement if tauto=automatic
elselect	globalmin worst elements		Method for selecting elements to refine.
elselectauto	globalmin		Method for selecting elements to refine if tauto=automatic.
gf	positive scalar	2	Interval growth factor
globalminpar	positive scalar		Controls refinement if elselect=globalmin
globalminparauto	positive scalar		Controls refinement if elselectauto=globalmin
initialsteprestart	positive scalar	0.001	Initial time step size after restart
initialsteprestart active			
message	String		The log message from the last solution process
minti	positive scalar	0.01	Minimal length of adaptation time intervals
ngenlocal	scalar integer	2	Maximum number of element refinements
rf	positive scalar	0.5	Interval reduction factor
rmethod	regular longest	longest	Refinement method
samplepts	scalar numeric vector	range(0,0.1,1)	Where to check the error in next subinterval
tauto	manual automatic	manual	Time interval control
tfrac	positive scalar	0.1	Length of adaptation time intervals

TABLE 6-25: VALID TIMEADAPTION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
tfracauto	positive scalar	0.1	Length of initial adaptation time interval if tauto=automatic
timeadapgeom	String		Name of geometry sequence
worstpar	positive scalar		Controls refinement if elselect=worst

The **Adaption** algorithm solves a sequence of problems using a sequence of refined meshes. The first mesh is obtained from the mesh sequence. The following generations of meshes are obtained by solving the problem, computing a mesh element error indicator based on the error estimate function, selecting a set of elements based on the element pick function, and then finally refining these elements. The solution to the problem is then recomputed. The loop continues until the maximum number of element generations has been reached or until the maximum number of elements is obtained.

The adaptive solver works in one geometry at a time. You specify the name of the geometry sequence in the property **geomnum**. The adaptive solver only supports simplex meshes.

Error Estimation

Then, the residuals in the equations are computed for all mesh elements. The *error estimation function* is given by the property **eefunc**. For solver type **stationary** there are two functions available: **l2** and **func**. The function **l2** computes the error indicator using the L^2 norm, and the function **func** computes the error indicator using a functional. For the solver type **eigenvalue** only the **l2** function is available.

If the error estimate function **functional** is used, you must provide a functional variable name using the property **eefunc**. This variable must have global scope. The algorithm computes the adjoint solution related to the functional and estimates the error in this solution. This error estimate is used for the selection of elements to refine. Two methods are supported to estimate the error in the adjoint solution: a recovery technique and a gradient-based method. By selecting **adjppr=on** you can enforce using the recovery technique, and with **adjppr=off** the adaptive solver uses the gradient method. When using the **auto** option, the algorithm automatically detects if the model uses only Lagrange basis function. If so, the recovery technique is used. Otherwise, the algorithm chooses the gradient-based method. In the Solver Log you can inspect which method that is selected. See [Adaptive Mesh Refinement \(Utility Node\)](#) in the *COMSOL Multiphysics Reference Manual* for more information.

The error estimator gives local error indicators $(f(i,j)h(j)^{\beta(i)})^{\alpha}\text{Vol}(j)$, where i is the equation number, j is the mesh element number, h is the mesh element size, and **Vol** is the mesh element volume. $f(i,j)$ is the scaled absolute value of the i th equation residual on the j th mesh element. The mesh element error indicator is the sum of these local error indicators over the equation index i . The global error indicator is the α root of the sum of the mesh element error indicators over the mesh element index j .

If the eigenvalue solver is used, you can specify the weighting of the error indicators for the different eigenfunctions using the property **Eigselect**. The n th component of the **Eigselect** vector is the weight for the error indicator of the n th eigenfunction.

Mesh Refinement

Then, a refinement of the mesh is generated based on the local error indicators. The aim is to refine the mesh most where the errors are largest. The mesh refinements ratios are determined by the property **elselect**, and depending on the value of **elselect** on one of the parameters **globalminpar**, **worstpar**, and **elementspar**. If **elselect=globalmin**, the solver tries to minimize the total error for a prescribed mesh size, namely **globalminpar** times the current number of mesh elements. Each element can be refined several times. Otherwise, each element is refined at most once. If **elselect=worst**, the solver refines the elements with an error greater than a fraction **worstpar** of the worst error, whereas if **elselect=elements** the solver refines a given fraction of the elements, given in the property **elementspar**.

The property `resorder` is a scalar or vector that gives the order of decrease of the equation residuals as the mesh size h tends to 0. If it is a vector, the residual of the n th equation is $O(h^{\text{Resorder}(n)})$. If `resorder` equals `auto`, the software determines the order of decrease in equation residuals on the basis of the shape function orders in the model. Roughly speaking, the order is one less than the shape function order for the corresponding equation.

Once the mesh refinement ratios have been determined, the mesh is refined using the method given in the property `rmethod`, and the algorithm starts a new iteration. The refinement method is either `longest`, `regular`, or `meshinit`. Details on the first two refinement methods can be found in the entry on `meshrefine`. `Meshinit` means that a new mesh is generated through a call to `meshinit` using the `Hmesh` property to control the element sizes.

Convergence Control

No more than `ngen` successive refinements are attempted. Refinement is also stopped when the number of elements in the mesh exceeds `maxt`.

TIME ADAPTATION

The `TimeAdaption` algorithm solves a sequence of problems on a sequence of adapted meshes. The first mesh, the base mesh, is obtained from the mesh sequence. The new adapted mesh is obtained by evaluating the mesh element error indicator, selecting a set of elements based on the element pick function, and then finally refining these elements. The solution to the problem on the previous mesh is then mapped to the new mesh and time integration continues until the next mesh adaptation takes place. The time of mesh adaptation can be determined manually or automatically.

The time adaptive solver works in one geometry at a time. You specify the name of the geometry sequence in the property `timeadapgeom`. The solver only supports simplex meshes, and if the base mesh is not simplex it can be converted by using the property `convertmesh`.

The length of the time interval using a fixed adapted spatial mesh can be controlled manually or automatically by the property `tauto`. If the time integrator runs into problems the computation is restarted at the beginning of the previous time interval. The length of the new interval is reduced to a fraction of the current interval length. This fraction is specified by the property `rf`. In the `tauto=manual` case the time interval length is given by property `tfrac`; if `tauto=automatic` the property `tfracauto` controls the initial interval length. For both cases the shortest possible interval length is given by the property `minti`.

If the property `tauto` is set to `automatic` the `TimeAdaption` algorithm tries to determine the length of the time interval according to the requested fraction of maximum refinement. The fraction is given by the value of the property `elfrac`. A value of zero means no refinement of the base mesh and a value of one means refinement everywhere with maximum element refinements (set through property `ngenlocal`). The algorithm strives to assume the given value of `elfrac` by controlling the size of the time interval. The shortening and lengthening of the interval is determined by the interval reduction and growth factors. These are the properties `rf` and `gf` respectively.

The error indicator is specified using the property `eefuncime`. A solution on the coarse base mesh is computed in the next time interval and the error indicator is evaluated at the points given by property `samplepts`. In this way a new adapted mesh appropriate for the next time interval can be generated and the computation on this new mesh is then started. The sample points must be specified as a number between 0 and 1 because they are interpreted as being relative to the time interval under consideration. Entering a scalar value of 0.5 means that the error indicator is evaluated at the midpoint of the interval.

After each mesh adaptation the time integration is restarted and you can control the time stepping by the `Time` type analogous properties `consistentrestart` and `initialsteprestart`.

Advanced

Handle advanced general solver properties.

SYNTAX

```
model.sol(sname).feature(solv).create(fname, "Advanced")  
model.sol(sname).feature(solv).feature(fname).set(pname, value)
```

DESCRIPTION

Advanced feature.

TABLE 6-26: VALID GENERAL PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
autorescale	on off	on	Automatic rescaling of linear equations (for the Stationary solver).
blocksize	positive integer auto	auto	Assembly block size.
checkmatherr	on off	off	Check for undefined numerical values after each operation.
complexfun	on off	off	Use complex-valued functions with real input.
convinfo	on detailed off	on	Print info to log.
D, E, K, L, M, N	on off		Manual control of reassembly.
keep	on off		Manual control of reassembly.
logsampling	double, zero or positive	0.005	Minimum time (in seconds) for log sampling of time-dependent solvers.
matherr	on off	on	Error for undefined operations.
matrixformat	auto sparse filled matrixfree	auto	Matrix format.
orthonormallimit	double	1e7	Limit, as an estimate of the complexity (number of operations), for using the orthonormal method in the automatic null-space function.
nullfun	fnullorth flspnull auto explicitorth explicitsp	auto	Null-space function.
rowscale	on off	on	Equilibrate rows.
storeresidual	off solving solvingandoutput	off	Store the latest residual while solving or while solving and in the output.
symmetric	auto on off hermitian	auto	Symmetric matrices.

The [Advanced](#) section in the *COMSOL Multiphysics Reference Manual*, describes the functionality corresponding to the properties `blocksize`, `complexfun`, `nullfun`, and `rowscale`.

You can use the property `symmetric` to tell the solver that the model is symmetric/hermitian or you can use the automatic feature to find out (see [Advanced](#) in the *COMSOL Multiphysics Reference Manual*).

You can set `convinfo=detailed` to print more detailed information about the solver process in the log window. For example information about individual linear iterations or the scales per field computed by the automatic scaling algorithm. When `convinfo=off` only minimal information about the solution process is printed.

By default, COMSOL Multiphysics gives an error message if the solver encounters an undefined mathematical operation when solving the model, for instance $0/0$ or $\log(0)$. If you instead want the solver to proceed, put the property `matherr=off`. Then $0/0=\text{NaN}$ (not a number) and $\log(0)=-\text{Inf}$.

The properties `keep` and `D`, `E`, `K`, `L`, `M`, and `N` allow manual control of reassembly. If `keep=on`, each of the other properties controls reassembly of a specific matrix or vector. Setting the property value to `on`, means that the quantity is constant, and therefore can be assembled once and then kept. The letters have the following meaning:

E=constant mass, D=constant damping, K=constant Jacobian, L=constant load, M=constant constraint, N=constant constraint Jacobian.

The `autorescale` property control if the automatically computed scales should be recomputed. This property only affects stationary nonlinear problems and fields that are using the automatic scaling method and for the constant damping technique. The initially computed scales are based on the initial assembled matrix. When `autorescale=on` the scales are recomputed in each nonlinear iteration based on the current solution.

You can use the property `matrixformat` to tell the solver which matrix format to store the system matrices in:

- `sparse` stores the matrix in a sparsely populated matrix format.
- `filled` stores the matrix in a densely populated format.
- `matrixfree` does not store matrices. The effects of matrix operations are assembled when needed.

When you specify `auto` (the default), the format is automatically determined based on the solver used.

By default, COMSOL does not check for undefined numerical values (for example, from numerical overflow) after each numerical operation. Set the property `checkmatherr` to `on` to make COMSOL check for such undefined numerical values, which will give more accurate error messages if such undefined numerical values occur.

Assemble

Assemble and store the matrices generated during assembly.

SYNTAX

```
model.sol(sname).create(fname, "Assemble")
model.sol(sname).feature(fname).set(pname, value)
model.sol(sname).feature(fname).getSparseMatrixVal(mname)
model.sol(sname).feature(fname).getSparseMatrixValImag(mname)
model.sol(sname).feature(fname).getSparseMatrixRow(mname)
model.sol(sname).feature(fname).getSparseMatrixCol(mname)
model.sol(sname).feature(fname).getVector(vname)
model.sol(sname).feature(fname).getVectorImag(vname)
model.sol(sname).feature(fname).getSparseMatrixValBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixValImagBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixRowBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixColBlock(mname, start, stop)
model.sol(sname).feature(fname).getVectorBlock(vname, start, stop)
model.sol(sname).feature(fname).getVectorImagBlock(vname, start, stop)
model.sol(sname).feature(fname).isReal(mname)
model.sol(sname).feature(fname).getM(mname)
model.sol(sname).feature(fname).getN(mname)
```

DESCRIPTION

Assemble feature.

TABLE 6-27: VALID GENERAL PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
K	on off	off	Assemble the stiffness matrix
L	on off	off	Assemble the load vector
M	on off	off	Assemble the constraint vector
N	on off	off	Assemble the constraint Jacobian
D	on off	off	Assemble the damping matrix
E	on off	off	Assemble the mass matrix
NF	on off	off	Assemble the constraint force Jacobian

TABLE 6-27: VALID GENERAL PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
NP	on off	off	Assemble the optimization constraint Jacobian
MP	on off	off	Assemble the optimization constraint vector
MLB	on off	off	Assemble the lower bound constraint vector
MUB	on off	off	Assemble the upper bound constraint vector
Kc	on off	off	Assemble the eliminated stiffness matrix
Lc	on off	off	Assemble the eliminated load vector
Dc	on off	off	Assemble the eliminated damping matrix
Ec	on off	off	Assemble the eliminated mass matrix
Null	on off	off	Assemble the constraint null-space basis
Nullf	on off	off	Assemble the constraint force null-space basis
ud	on off	off	Assemble the particular solution ud
uscale	on off	off	Assemble the scale vector
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");)
eiguse	on off	off	Assemble an eigenvalue problem using the eigname as the eigenvalue
eigname	String	lambda	The name of the eigenvalue
eigref	String	0	Value of eigenvalue linearization point
message	String		The log message from the last assembly process

The assemble feature assembles the matrices specified as output matrices and stores them in the feature. The output is stored in the feature. You can access the result using the matrix and vector access methods. The linearization point is the solution stored in the sequence when the assemble feature is called. The linearization point is stored in the sequence after the run. For information about the eliminated system, see [Advanced](#) in the *COMSOL Multiphysics Reference Manual*.

AutoRemesh

Iteratively and automatically create deformed geometries and remesh these geometries. In each step, map the solution and restart the simulation.

SYNTAX

```
model.sol(sname).feature(tname).create(fname, "AutoRemesh")
model.sol(sname).feature(tname).feature(fname).set(pname, pvalue)
```

DESCRIPTION

Operation feature. The following property/values are accepted:

TABLE 6-28: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
autoremeshgeom	String		Name of geometry sequence
consistentremesh	on off	off	Consistent initialization after remesh
initialstepremesh	positive scalar	0.001	Initial time step size after remesh
initialstepremesh active	on off	off	Use initialstepremesh
solutionremesh	tout tstep	tstep	Solution to use for remeshing

TABLE 6-28: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
stopcondtype	quality distortion general	quality	Type of condition for remeshing
stopdistexpr	String		Distortion expression
stopdistval	String	2	Maximum distortion allowed
stopexpr	String		Mesh quality expression
stopgenexpr	String		Logical condition for remeshing
stopval	String	0.2	Minimal mesh quality

The automatic remeshing solver works in one geometry at a time. You specify the name of the geometry sequence in the property `autoremeshgeom`. Automatic remeshing is available for Time-Dependent studies and is intended for use with the Moving Mesh and Deformed Geometry interfaces.

Use the `stopcondtype` property to select the type of condition for when remeshing should occur:

- **quality.** This means that the solver remeshes when the value of `stopexpr` becomes smaller than the value of `stopval`.
- **distortion.** This means that the solver remeshes when the value of `stopdistexpr` becomes larger than the value of `stopdistval`.
- **general.** This means that the solver remeshes when the `stopgenexpr` property becomes true (nonzero).

The `solutionremesh` property determines which previous solution is used for the remeshing:

- **tout** means that remeshing is done on the last solution that would have been stored by the solver if remeshing would not have occurred. This setting discards any solver progress done since the last output.
- **tstep** means that the remeshing is done using the solution from the last solver step before the condition for remeshing became fulfilled. Only the very last solver step, at which the condition was triggered, is discarded. Typically this setting is preferred because then the progress of the automatic remeshing does not depend on the solver's list of output times.

After each remeshing the time integration is restarted and you can control the time stepping by the `Time` type analogous properties `consistentremesh` and `initialstepremesh`.

If the time integrator runs into problems the computation is restarted at the beginning of the previous time interval using stricter time stepping controls.

AWE

Solve a parametric problem using asymptotic waveform evaluation (AWE).

SYNTAX

```
model.sol(sname).create(fname, "AWE")
model.sol(sname).feature(fname).set(pname, pvalue)
```

DESCRIPTION

Operation feature. The following property/values are accepted:

TABLE 6-29: VALID AWE PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
acceptshort	on off	on	If on, the solver accepts short intervals unconditionally
atol	scalar	0.001	Absolute tolerance for parameter sweep

TABLE 6-29: VALID AWE PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
aweassemble	all one	all	Either assemble all the needed matrices at once, or one at a time
awefunc	string vector of strings		Expression(s) used in the search algorithm
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");)
expeval	scalar numeric vector	range(0.1, 0.1, 0.9)	Where to check the error in each subinterval.
expsize	scalar	3	Number of terms in expansion.
exptype	pade taylor	pade	Use Padé or Taylor expansions to approximate the unknown.
minint	scalar	0	The shortest allowed subinterval length
minintactive	on off	off	If off, rtol times the parameter span is. used. If on, minint is used.
message	String		The log message from the last solution process.
outsollinearized	du u	du	Store the total solution (u) or deviation and linerization point (du), when storelinpoint=off.
plist	scalar numeric vector		Parameter list.
plot	on off	off	Plot while solving.
plotgroup	String	default	Plot group to use for plot while solving.
pname	vector of strings		Parameter names.
pout	plist psteps	plist	Output either the parameters in plist or the solution at the expansion points.
probesel	all none manual	all	The probes to compute.
probes	vector of strings		Probes to use when probesel>manual.
rtol	scalar	0.01	Relative tolerance for parameter sweep.
storelinpoint	on off	off	Whether to store the linearization point.

The AWE solver computes expansions of an underlying problem around certain parameter values. In the first step the largest and smallest values of `plist` are used as expansion points. Using these expansions, the values of one or more functionals at intermediate parameter values are computed. If the two expansions give similar enough functional values at these internal points, the interval is accepted and no subdivisions of that particular interval are deemed necessary.

The property `awefunc` is used to specify the functionals of interest and the property `expeval` determines at which internal points these functionals are to be evaluated. The values for `expeval` are given relative the interval. That is, a value of 0.5 means that the functionals are evaluated at the midpoint of each interval. When the functional-values from the two expansions are compared, a check is performed to see if they fulfill the specified tolerances `atol` and `rtol`. If neither of the tolerances are fulfilled, the interval is bisected and the process is repeated for each subinterval. Before a bisection is performed a check is made to make sure that the new intervals are not shorter than the shortest allowed. By default the shortest allowed interval is given by the relative tolerance times the length of the interval defined by `plist` (when `minintactive` is set to off). If `minintactive` has been set to on the value of `minint` is the shortest interval allowed. If `minint` has been specified, the value of `minintactive` is on by default. The property `acceptshort` determines how to handle too short intervals. If `acceptshort` is set to off and a short

interval is detected, the solver is interrupted with an error/warning. If `acceptshort` is set to on and a short interval is detected, the solver accepts the interval even if the tolerances have not been fulfilled.

In AWE several matrices are needed to compute each expansion. There are two options when it comes to assembling these matrices: With `aweassemble` set to `all` everything is assembled in a single call to the `Xmesh`. With `aweassemble` set to `one`, the matrices are assembled one at a time. The first option is faster but requires more memory.

Eigenvalue

Solve a PDE eigenvalue problem.

SYNTAX

```
model.sol(sname).create(fname,"Eigenvalue")
model.sol(sname).feature(fname).set(pname,value)

model.sol(sname).feature(fname).create(fname2,LinearType)
model.sol(sname).feature(fname).create(fname2,"Advanced")
```

Here `LinearType` is any of the allowed linear solver feature types.

DESCRIPTION

Operation feature.

For both linear and nonlinear problems, the eigenvalue problem is that of the linearization about a solution U_0 . If the eigenvalue appears nonlinearly, COMSOL Multiphysics reduces the problem to a quadratic approximation around a value λ_0 specified by the property `eigref`. The discretized form of the problem reads

$$KU - (\lambda - \lambda_0)DU + (\lambda - \lambda_0)^2 EU = -N_F \Lambda$$

$$NU = M$$

where K , D , E , N , and N_F are evaluated for $U = U_0$ and $\lambda = \lambda_0$. Λ is the Lagrange multiplier vector, and λ is the eigenvalue. The eigenvalue name can be given by the property `eigname`. The linearization point U_0 can be given with the property `U`. The shift, described below, is compensated according to the linearization point for the eigenvalue. Therefore, changing the linearization point has no effect at all for linear or quadratic eigenvalue problems. The eigenvalue search method can be manual or a region in the complex plane (controlled by the property `eigmethod`).

The feature `eigenvalue` accepts the following property/value pairs:

TABLE 6-30: VALID EIGENVALUE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
<code>appnreigs</code>	positive integer		Approximate number of eigenvalues (for <code>eigmethod</code> = <code>region</code>)
<code>const</code>	String array		Provide constants as input parameters using an N-by-2 string array (for example, <code>.set("const", "a 1.0 b 1.0");</code>)
<code>control</code>	String	<code>user</code>	Name of controlling study
<code>eigfunscale</code>			
<code>eigli</code>	real scalar	0	Largest imaginary value of search region
<code>eiglrr</code>	real scalar	0	Largest real value of search region
<code>eigmethod</code>	manual region	manual	Eigenvalue search method

TABLE 6-30: VALID EIGENVALUE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
eigname	String	lambda	Name of eigenvalue variable
eigref	String	0	Linearization point for the eigenvalue
eigsi	real scalar	0	Smallest imaginary value of search region
eigsr	real scalar	0	Smallest real value of search region
eigwhich	lm lr sr li si	lm	Eigenvalue search direction (for eigmethod>manual)
krylovdim	positive integer		Dimension of Krylov space
linpmethod	init solution	init	Method used for linearization point
linpsol	zero solution object	zero	Linearization point solution
linpsoluse	current solution store	current	Linearization point solution to use
maxeigit	positive integer	300	Maximum number of eigenvalue iterations
maxnreigs	positive integer	200	Maximum number of eigenvalue
message	String		The log message from the last solution process
neigs	positive integer	6	Number of eigenvalues sought
neigsactive	on off	off	Specify the number of eigenvalues to search for
rtol	scalar	1e-4	Relative tolerance
shift	scalar	0	Eigenvalue search location
shiftactive	on off	off	Specify eigenvalue search location
shiftregman	scalar	0	Eigenvalue shift (when shiftregselect = manual)
shiftregselect	auto manual	auto	Search for eigenvalues around (auto = center of search region)
solfile	on off	off	Store solution on file
solfileblock	positive scalar	16	Max size of solution block (MB)
solnum	auto all positive integer	auto	
storelinpoint	on off	off	Whether to store the linearization point
transform	String	none	Eigenvalue transformation

Specify where to look for the desired eigenvalues with the property `shift`. Enter a real or complex scalar; the default value is 0, meaning that the solver tries to find eigenvalues close to 0.

For more information about the eigenvalue solver, see [Eigenvalue Solver](#) in the *COMSOL Multiphysics Reference Manual*.

If `Solfile` is on, the solution is stored on a temporary file. The temporary file is stored in the temporary directory created at startup. You can decide the temporary directory with the `-tmpdir` switch. A part of the solution is stored in memory in a few *blocks* (usually 1–5 blocks reside in memory). The maximum block size (in megabytes) can be controlled with the property `Solfileblock`.

FFT

Compute a fast Fourier transform (FFT) or inverse fast/nonuniform Fourier transform (IFFT/INFT) using an FFT study step and solver,

SYNTAX

```
model.sol(sname).create(fname, "FFT")
model.sol(sname).feature(fname).set(pname, pvalue)
```

DESCRIPTION

Operation feature. The following property/values are accepted:

TABLE 6-31: VALID FFT SOLVER PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
addstatso1	on off	off	Add stationary solution.
control	String	user	Set control for FFT solver by study.
fftendf	double	1.0	End point of frequency interval, only applicable when <code>fftsteptypef = interval</code> .
fftendtime	double	1.0	End time for the time interval in a forward FFT.
fftextend	on off	on	For an inverse NFT/FFT, extend the input data samples by adding complex conjugate pairs.
fftmaxfreq	double	10	Maximum output frequency in forward FFT.
fftoutorder	nat sym	nat	Output order (natural or symmetric) of frequencies for forward FFT.
fftouttrange	double array		Output times for inverse NFT/FFT.
fftperiodic	on off	off	Periodicity of input data; only available if <code>fftextend</code> is set to <code>off</code> for inverse NFT/FFT (always available for forward FFT).
fftphaseinexpr	String		Expression for input phase function, can be expressed in terms of <code>t</code> , <code>freq</code> , and <code>niterFFTin</code> (if applicable).
fftphaseoutexpr	String		Expression for output phase function, can be expressed in terms of <code>t</code> , <code>freq</code> , and <code>niterFFTout</code> (if applicable).
fftphasetypein	none fromexpr	none	Method for input phase function.
fftphasetypeout	none fromexpr	none	Method for output phase function.
fftrealstore	on off	on	Do not store negative frequencies for real input.
fftscaling	cont discrete	discrete	Use a discrete or continuous scaling for the Fourier transform.
fftstartf	double	0.0	Start point of frequency interval, only applicable when <code>fftsteptypef = interval</code> .
fftstarttime	double	0.0	Start time for the time interval in a forward FFT.
fftsteptypef	allfreqs interval	allfreqs	Method for selecting frequencies from range.
ffttranstype	transfft transiFFT	transfft (depends on type of FFT study step)	Type of FFT transformation: forward or inverse.
fftwinalpha	double	0.5	Window parameter for a Tukey window.
fftwincenterfw	double	0.5	Window center for a Gaussian window function in a forward FFT.
fftwincenterinv	double	50	Window center for a Gaussian window function in an inverse NFT/FFT.
fftwindeV	double	1	Standard deviation for a Gaussian window function.
fftwincutoff	double	1.0	Cut-off fraction for window function in [0, 1].
fftwindowfw	on off	off	Use window function in forward FFT.
fftwindowinv	on off	off	Use window function in inverse NFT/FFT.

TABLE 6-31: VALID FFT SOLVER PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
fftwinexpr	String		Expression for window function, can be expressed in terms of t, freq, niterFFT _{in} , and niterFFT _{out} (if applicable).
fftwinmaxfw	double	1	Maximum (end) value for window in forward FFT.
fftwinmaxinv	double	100	Maximum (end) value for window in inverse NFT/FFT.
fftwinminfw	double	0	Minimum (start) value for window in forward FFT.
fftwinmininv	double	0	Minimum (start) value for window in inverse NFT/FFT.
fftwintypefw	fromexpr cutoff rectangle gauss hamming hanning blackman tukey	fromexpr	Method for window function in a forward FFT.
fftwintypeinv	fromexpr cutoff rectangle gauss hamming hanning blackman tukey	fromexpr	Method for window function in an inverse NFT/FFT.
linpsol	String	current	Solution that defines the input values for the FFT study step/FFT solver.
linpsoluse	String	current	Subsolution that defines the input value for the FFT study step/FFT solver.
linpstudy	String	current	Study that defines input values for the FFT study step.
punit	String	Hz	Frequency unit.
statmethod	init sol	sol	For an added stationary solution, use a solution or initial expression.
statsol	String		Solution to use as an added stationary solution.
statsolnum	String		The solution number to use for the solution used as an added stationary solution.
statsoluse	A stored solution or current.	current	Use the current solution or a stored solution as the added stationary solution.
statstudy	String		Study from which the added stationary solution is chosen.
statt	double		Time value at which the added stationary solution is chosen.
statmanualsolnum	integer		Solution number of the added stationary solution.
tunit	String	s	Time unit.
winpunit	unit	Hz	Frequency unit for window in an inverse NFT/FFT.

EXAMPLE*Code for use with Java*

```
model.study("std2").create("tdfft", "TimeToFreqFFT");
```

```

model.study("std2").feature("tdfft").set("linpstudy", "std1");
model.study("std2").feature("tdfft").set("fftstarttime", "0.5");
model.study("std2").feature("tdfft").set("fftendtime", "2.0");
model.study().create("std3");
model.study("std3").create("fdfft2", "FreqToTimeFFT");
model.study("std3").feature("fdfft2").set("linpstudy", "std2");

...

model.sol("sol7").create("fft1", "FFT");
model.sol("sol7").feature("fft1").set("ffttranstype", "transfft");
model.sol("sol7").feature("fft1").set("fftstarttime", "0.5");
model.sol("sol7").feature("fft1").set("fftendtime", "2.0");
model.sol("sol7").feature("fft1").set("control", "tdfft");

...

model.sol("sol9").feature("fft1").set("ffttranstype", "transfft");
model.sol("sol9").feature("fft1").set("control", "fdfft2");

model.study("std3").feature("fdfft2").set("linpmethod", "sol1");
model.study("std3").feature("fdfft2").set("linpstudy", "std2");
model.study("std3").feature("fdfft2").set("linpsol", "current");
model.study("std3").feature("fdfft2").set("linpsoluse", "current")

...

model.sol("sol2").feature("ft1").set("fftstarttime", "0.5");
model.sol("sol2").feature("ft1").set("fftendtime", "2.0");
model.sol("sol2").feature("ft1").set("fftperiodic", "off");
model.sol("sol2").feature("ft1").set("fftoutorder", "nat");

...

model.sol("sol2").feature("ft1").set("fftphase", "on");
model.sol("sol2").feature("ft1").set("fftwintypefw", "fromexpr");
model.sol("sol2").feature("ft1").set("fftwinexpr", "(niterFFTin<5)*niterFFTin");
model.sol("sol2").feature("ft1").set("fftphasetypein", "fromexpr");
model.sol("sol2").feature("ft1").set("fftphaseinexpr", "5+i*2*niterFFTin");
model.sol("sol2").feature("ft1").set("fftphasetypeout", "fromexpr");
model.sol("sol2").feature("ft1").set("fftphaseoutexpr", "2+i*4*niterFFTout");

```

COMPATIBILITY

The properties `fftbwalgtype` and `fftmeasure` from earlier versions are no longer available in version 5.2.

In version 5.2, the property `fftwintype` for the window type in earlier versions has been replaced by `fftwintypefw` and `fftwintypeinv` for the window type for a forward FFT and for an inverse NFT/FFT, respectively.

In version 5.2, the property `fftwindow` for switching a window on and off in earlier versions has been replaced by `fftwindowfw` and `fftwindowinv` for the window type for a forward FFT and for an inverse NFT/FFT, respectively.

In version 5.2, the property `tlist` for the input time range to a forward FFT in earlier versions has been replaced by the properties `fftstarttime` and `fftendtime` for the start time and end time, respectively.

For, EndFor

Add for loops to a solver sequence.

SYNTAX

```
model.sol("sol1").create("for1", "For");  
model.sol("sol1").create("endfor1", "EndFor");
```

DESCRIPTION

`model.sol("sol1").create("for1", "For")` adds the start of a for loop.

`model.sol("sol1").create("endfor1", "EndFor")` adds the end of a for loop.

Use `For` and `EndFor` to enclose a sequence of solver commands that you want to iterate in a for loop. You can add more than one for loop, but they must be balanced so that each `For` ends with a corresponding `EndFor`.

You control the number of iterations in the loop using the following property:

TABLE 6-32: PROPERTY/VALUE PAIR FOR THE FOR LOOP

PROPERTY	VALUE	DEFAULT	DESCRIPTION
iter	Integer	5	Number of iterations (0 or greater)

FullyCoupled

Handle the fully coupled nonlinear solution approach.

SYNTAX

```
model.sol(sname).feature(solv).create(fname, 'FullyCoupled')  
model.sol(sname).feature(solv).feature(fname).set(pname, value)  
model.sol(sname).feature(fname).feature(sname).set(pname, value)
```

DESCRIPTION

This feature can be used as an attribute to the `Time` and `Stationary` features. The nonlinear solver is an affine invariant form of the damped Newton method.

TABLE 6-33: VALID FULLY COUPLED PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
cfltol	positive scalar	0.1	Target error estimate for pseudo time stepping.
damp	positive real	1	Damping factor for the damped Newton method.
ddoginitdamp	nonnegative scalar	1	Initial damping factor for dtech set to ddog.
dtech	const auto hnlin ddog	auto const (Time)	Damping technique.
initcfl	positive scalar	5.0	Initial CFL number for pseudo time stepping.
initstep	nonnegative scalar	1	Initial damping factor for dtech set to auto.
initsteph	nonnegative scalar	1e-4	Initial damping factor for dtech set of hnlin.
jtech	minimal once onevery	onevery minimal (Time)	Jacobian update technique for dtech set to const.
kdpid	positive scalar	0.05	PID regulator-Derivative for pseudo time stepping.
kipid	positive scalar	0.05	PID regulator-Integrative for pseudo time stepping.
kppid	positive scalar	0.65	PID regulator-Proportional for pseudo time stepping.
maxiter	positive integer	25 4 (Time)	Maximum number of Newton iterations.

TABLE 6-33: VALID FULLY COUPLED PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
minstep	positive scalar	1	Minimum damping factor for dtech set to auto.
minsteph	positive scalar	1e-4	Minimum damping factor for dtech set to hnlin.
niter	positive integer	1	Fixed number of iterations.
ntermauto	tol itertol	tol	Termination techniques for dtech set to auto/hnlin.
ntermconst	iter tol itertol	tol	Termination techniques for dtech set to const.
ntolfact	positive scalar	1	Tolerance factor.
ratelimit	positive scalar	0.9 (dtech set to const)	Limit on nonlinear convergence rate.
aaccdim	positive integer	10	Dimension of Anderson iteration space when segstabacc = segaacc.
plot	on off	off	Plot while solving.
plotgroup	String	default	Plot group to use for plot while solving.
probesel	all none manual	all	The probes to compute.
probes	vector of strings		Probes to use when probesel=manual.
reserrfact	positive scalar	100	Residual factor for termonres=auto.
resscale	scalefieldwise scaleuniform	scalefieldwise	Residual scaling technique for dtech set to ddog.
rstep	positive scalar	10	Restrictions for step-size update.
stabacc	none clfcmp aacc	none	Stabilization and acceleration: None, pseudo time stepping (for stationary solvers), or Anderson acceleration.
termonres	off on auto	auto	Termination criterion: Solution, residual, solution or residual.
useratelimit	on off	off on (Time)	Use limit on nonlinear convergence rate.

The property `dtech` controls which damping factor to use in the damped Newton iterations.

For `dtech` set to `auto`, the solver determines an appropriate damping factor automatically. For this method the initial and minimally allowed damping factors are controlled by the properties `initstep` and `minstep` respectively. The termination technique is controlled by the property `ntermauto`.

For `dtech` set to `hnlin`, the solver determines an appropriate damping factor automatically but treat the problem as being highly nonlinear. This option can be tried if there is no convergence with `dtech` set to `auto`. For this method the initial and minimally allowed damping factors are controlled by the properties `initsteph` and `minsteph` respectively. Moreover, certain internal control structures are adapted. Especially, the error control is biased from a more absolute norm towards a relative norm. So this parameter is also useful if a solution with components of highly varying orders of magnitudes are present. In the context of parameter stepping, you can also try this option if the step sizes in the parameter seem to be too small.

When `dtech=const`, the constant damping factor specified in the property `damp` is used. For this method the termination technique is controlled by the property `ntermconst`. Furthermore, the property `jtech` can be used to control how often the Jacobian is updated. With `jtech=minimal`, the Jacobian is updated as seldom as possible (only once for a stationary problem and at most once per time step for a time-dependent problem). For time-dependent problems, the choice `jtech=once` makes the solver update the Jacobian once per time step. With

`jtech=onevery`, the Jacobian is updated on every Newton iteration. The default is `onevery` for stationary problems and `minimal` for time-dependent problems.

When `dtech` is set to `ddog` (stationary problems), the double dogleg solver is used. The initial damping factor is controlled by the property `ddoginitdamp` and the property `resscale` controls the residual scaling. The option `resscale=scalefieldwise` scales the equations based on the field-wise sizes of the initial residual. When the option `resscale=scaleuniform` is selected the algorithm terminates on the relative residual based on the initial residual.

The tolerance `ntol` gives the criterion for convergence for a stationary problem; see [Stationary Solver](#) in the *COMSOL Multiphysics Reference Manual*.

The property `ntolfact` controls how accurately the nonlinear system of equations is solved. The value given in `ntolfact` is multiplied with the main solver tolerance and used in the convergence criteria. Also, the solution process is interrupted (and the Jacobian updated or the time step reduced) if the convergence is too slow. This can be disabled by setting `useratelimt=off`. When `useratelimt=on`, what is to be considered as too slow convergence can be controlled through the property `ratelimit`. The solution process is interrupted if the estimated linear convergence rate (of all steps, when the segregated solver is used) becomes larger than the value given in `ratelimit`.

The property `stabacc` enables or disables pseudo time stepping (for stationary problems) or Anderson acceleration. When enabled the pseudo time stepping is controlled by the scalar-valued regulator parameters `cfltol`, `initcfl`, `kdpid`, `kipid`, and `kppid`. For the Anderson acceleration, the parameter `aacddim` specifies the dimension of the Anderson iteration space.

The property `termmonres` controls the termination criterion for stationary problems when `dtech=const` (and `ntermconst` is not `iter`), `auto`, or `hnlm`. When `termmonres=off` the estimated error is solution-based, with `termmonres=on` it is based on a relative residual and for `termmonres=auto` the estimated error is the minimum of the solution and residual based errors. For `termmonres=auto` the property `reserrfact` is a scalar factor multiplying the relative residual error.

COMPATIBILITY

The property `usecflcmp` from earlier versions of COMSOL Multiphysics is not used in version 5.0. Use the property `stabacc` instead.

InputMatrix

Input matrices and vectors to the linear solvers.

SYNTAX

```
solver=model.sol(sname).feature(solver)
solver.create(fname, 'InputMatrix')
solver.feature(fname).set(pname,value)
solver.feature(fname).addSparseMatrixVal(mname,row,col,val)
solver.feature(fname).addSparseMatrixValImag(mname,row,col,val)
solver.feature(fname).createSparseMatrix(mname,M,N,Nnz,isReal)
solver.feature(fname).createVector(mname,M,isReal)
solver.feature(fname).getSparseMatrixVal(mname)
solver.feature(fname).getSparseMatrixValImag(mname)
solver.feature(fname).getSparseMatrixRow(mname)
solver.feature(fname).getSparseMatrixCol(mname)
solver.feature(fname).getVector(vname)
solver.feature(fname).getVectorImag(vname)
solver.feature(fname).getSparseMatrixValBlock(mname,start,stop)
solver.feature(fname).getSparseMatrixValImagBlock(mname,start,stop)
solver.feature(fname).getSparseMatrixRowBlock(mname,start,stop)
solver.feature(fname).getSparseMatrixColBlock(mname,start,stop)
solver.feature(fname).getVectorBlock(vname,start,stop)
solver.feature(fname).getVectorImagBlock(vname,start,stop)
solver.feature(fname).setVector(vname,val)
solver.feature(fname).setVectorImag(vname,val)
solver.feature(fname).setVectorBlock(vname,val,start)
solver.feature(fname).setVectorImagBlock(vname,val,start)
solver.feature(fname).isReal(mname)
solver.feature(fname).getM(mname)
solver.feature(fname).getN(mname)
```

DESCRIPTION

The `InputMatrix` feature can be used to create the raw data of an assembled matrix or vector from Java[®]. The `InputMatrix` feature can exist as a subfeature of the `Eigenvalue`, `Stationary`, and `Time` solver features. These solver feature automatically pick up matrices from the `InputMatrix` subfeature instead of automatically assembling the matrices. The matrices are not stored in the model when the model is saved. They must be created before computing the solver features.

TABLE 6-34: VALID GENERAL PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
K	on off	off	Input the stiffness matrix
L	on off	off	Input the load vector
M	on off	off	Input the constraint vector
N	on off	off	Input the constraint Jacobian
D	on off	off	Input the damping matrix
E	on off	off	Input the mass matrix
NF	on off	off	Input the constraint force Jacobian

Linear

Handle linear system solvers with three different attribute features: `Direct`, `Iterative`, and `Multigrid`.

SYNTAX

```
sol.feature(fname).create(lname,"Direct")
sol.feature(fname).feature(lname).set(pname,value)

sol.feature(fname).create(lname,"Iterative")
sol.feature(fname).feature(lname).set(pname,value)
sol.feature(fname).feature(lname).create(pcname,PType);
sol.feature(fname).feature(lname).feature(pcname).set(pname,value)

sol.feature(fname).feature(lname).create(pname,'Multigrid')
sol.feature(fname).feature(lname).feature(pname).feature('presmooth').
    create(prename,SType)
sol.feature(fname).feature(lname).feature(pname).feature('postsmooth').
    create(postname,SType)
sol.feature(fname).feature(lname).feature(pname).feature('csolver').create(cname,CType)
sol.feature(fname).feature(lname).feature(pname).feature('presmooth').
    feature(prename).set(pname,value)
sol.feature(fname).feature(lname).feature(pname).feature('postsmooth').feature(postname).
    set(pname,value)
sol.feature(fname).feature(lname).feature(pname).feature('csolver').feature(cname).
    set(pname,value)
```

PType is any of the allowed preconditioner feature types. These types are Direct preconditioner, Multigrid, Domain decomposition, SOR, SOR Gauge, SOR Vector, Vanka, SOR Line, SCGS, AMS, Incomplete LU, and Krylov preconditioners. *SType* is any of the allowed smoother types. These are the same as the *PType* except Domain decomposition, Multigrid, and AMS. *CType* is any of the allowed coarse grid solver types: Direct and all types listed for *PType*.

DESCRIPTION

Three attribute features for linear system solvers.

EXAMPLES:

GMRES with ILU as preconditioner:

Code for use with Java

```
SolverFeature solver = sol.feature(fname).create("iter1","Iterative");
solver.set("solver","gmres");
solver.create("ilu","IncompleteLU");
```

Code for use with MATLAB

```
solver = sol.feature(fname).create('iter1','Iterative');
solver.set('solver','gmres');
solver.create('ilu','IncompleteLU');
```

Change the preconditioner to GMG/SORVector/SPOOLES:

Code for use with Java

```
solver.create("gmg","Multigrid");
solver("gmg").set("solver","gmg");
solver("gmg").feature("presmooth").create("p1","SORVector");
solver("gmg").feature("postsmooth").create("p1","SORVector");
SolverFeature csolver = solver("gmg").feature("csolver").create("c1","Direct");
csolver.set("solver","spooles");
csolver.set("errorchkd","on");
```

Code for use with MATLAB

```
solver.create('gmg','Multigrid');
solver('gmg').set('solver','gmg');
solver('gmg').feature('presmooth').create('p1','SORVector');
solver('gmg').feature('postsmooth').create('p1','SORVector');
csolver = solver('gmg').feature('csolver').create('c1','Direct');
```

```
csolver.set('solver','spooles');
csolver.set('errorchkd','on');
```

Use Conjugate Gradients instead of GMRES:

Code for use with Java

```
solver.set("solver","cg");
```

Code for use with MATLAB

```
solver.set('solver','cg');
```

Use the sparse approximate inverse (SAI) preconditioner using a sparsity pattern of SAI defined by a power of 3 of the system matrix:

Code for use with Java

```
model.sol("sol1").feature("t1").create("i1", "Iterative");
model.sol("sol1").feature("t1").feature("i1").set("linsolver", "cg");
model.sol("sol1").feature("t1").feature("i1").create("sai1", "SAI");
model.sol("sol1").feature("t1").feature("i1").feature("sai1").set("saisymm", "on");
model.sol("sol1").feature("t1").feature("i1").feature("sai1").
    set("saipattern","saipowa");
model.sol("sol1").feature("t1").feature("i1").feature("sai1").set("saipowera", "3");
```

Code for use with MATLAB

```
iter = model.sol('sol1').feature('t1').create('i1', 'Iterative');
iter.set('linsolver', 'cg');
iter.create('sai1', 'SAI');
iter.feature('sai1').set('saisymm', 'on');
iter.feature('sai1').set('saipattern','saipowa');
iter.feature('sai1').set('saipowera', '3');
```

THE PREFUN PROPERTY

The prefun property for the solver (preconditioner) accepts a different set of values depending on the context. In the section below, its possible values and default value is listed for each preconditioner that supports it.

DIRECT PROPERTIES

TABLE 6-35: VALID DIRECT PROPERTY/VALUE PAIRS (FOR ALL SOLVER)

PROPERTY	VALUES	DEFAULT	DESCRIPTION
errorchk	off on auto	auto	Check error estimate.
linsolver	mumps pardiso spooles dense	mumps	Method to use.
prefun	mumps pardiso spooles dense	mumps	Solver for preconditioner (MUMPS, PARDISO, SPOOLES, or Dense Matrix).
rhob	scalar > 1	400 1 (coarse solver)	Factor in linear error estimate.

TABLE 6-36: OPTIONAL DIRECT PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
incore	auto manual	auto	In-core memory method (PARDISO, MUMPS).
internalmemusage	positive real	3	Internal memory usage factor (PARDISO, MUMPS).
minicmemory	positive real	512.0	Minimum in-core memory in MB (PARDISO, MUMPS).
mumpsalloc	scalar at least 1	1.2	MUMPS memory allocation factor (MUMPS).

TABLE 6-36: OPTIONAL DIRECT PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
mumpsreorder	auto amd amf qamd nd	auto	Preordering algorithm (MUMPS).
mumpsrreorder	on off	on	Row preordering (MUMPS).
ooc	on off	off	Use out-of-core (PARDISO, MUMPS).
oocmemory	positive real	512.0	In-core memory in MB (PARDISO, MUMPS).
pivotenable	on off	on	Use pivoting (MUMPS).
pivotperturb	scalar between 0 and 1	1e-8	Pivot perturbation threshold (PARDISO, MUMPS).
pivotrefines	nonnegative integer	0	Number of forced iterative refinements (PARDISO, MUMPS).
pivotstrategy	on off	off	Use 2-by-2 Bunch-Kaufman pivoting (on) or 1-by-1 diagonal pivoting (off) (PARDISO).
preorder	mmd nd ms bestof	nd	Preordering algorithm (SPOOLES).
pardmtsolve	on off	off	Multithreaded forward and backward solve (PARDISO).
pardreorder	mmd nd ndmt	nd	Preordering algorithm (PARDSIO).
pardrreorder	on off	on	Row preordering algorithm (PARDISO).
pardschedule	auto one two	auto	Scheduling method (PARDISO).
thresh	scalar between 0 and 1	0.1	Pivot threshold (MUMPS, SPOOLES).
usetotmemory	scalar between 0 and 1	0.8	Used fraction of total memory (PARDISO, MUMPS).

ITERATIVE PROPERTIES

TABLE 6-37: VALID ITERATIVE PROPERTY/VALUE PAIRS (FOR ALL SOLVERS)

PROPERTY	VALUES	DEFAULT	DESCRIPTION
maxlinit	positive integer	10000 500 (coarse solver)	Maximum number of linear iterations (when used with a tolerance).
rhob	scalar > 1	400 1 (coarse solver)	Factor in linear error estimate.
linsolver	gmres fgmres bicgstab cg precondition	gmres, ilu (precond)	Method to use.
errorchk	off on auto	auto	Validate error estimate.

TABLE 6-38: OPTIONAL ITERATIVE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
droptol	scalar between 0 and 1	0.01 when used as preconditioner, 1 when used as presmoothen, postsmoothen, or coarse solver	Drop tolerance (SPOOLES, Incomplete LU with tolerance element dropping strategy).
droptype	tol fill	tol	Element dropping strategy for Incomplete LU (Tolerance, Fill ratio).

TABLE 6-38: OPTIONAL ITERATIVE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
fillratio	nonnegative integer	2	Fill ratio (Incomplete LU with Fill ratio element dropping strategy).
ilutdroptol	scalar between 0 and 1	0.01	Drop tolerance (ILUT).
ilutfillratio	nonnegative integer	2	Fill ratio (ILUT).
iter	nonnegative integer	2 (1 for ILU and ILUT)	Fixed number of iterations (when used as preconditioner, smoother, or coarse solver), for all iterative solvers except direct preconditioner.
iterm	tol iter	iter	Termination technique (when Incomplete LU, ILUT, or ILU0 are used as coarse solver).
itrestart	positive integer	50	Number of iterations before restart (gmres, fgmres).
prefun	ilu ilut ilu0 spooles	ilu	Solver (preconditioner) for Incomplete LU, ILUT (Intel MKL), ILU0 (Intel MKL), or SPOOLES).
prefuntype	left right	left	Left or right preconditioning (gmres, cg, bicgstab).
relax	scalar between 0 and 2	1	Relaxation factor (Jacobi, SOR-based algorithms, ILU, ILUT, SAI, and Vanka).
respectpattern	on off	on	Respect the matrix pattern (ILU).
saisymm	auto off on hermitian	auto	Symmetry characteristics of SAI method corresponding to Automatic, Nonsymmetric, Symmetric, and Hermitian, respectively.
saipattern	sysmat saipowa	sysmat	Sparsity pattern of SAI defined by system matrix or power of system matrix.
saipowera	positive integer	2	Power of system matrix for SAI sparsity pattern.
seconditer	nonnegative integer	1	Number of secondary iterations (SOR vector and SOR gauge algorithms), number of SSOR updates (vanka).
sorblocked	on off	on	Blocked SOR method.
sorvecdof	vector of strings		Vector element variables (SOR vector and SOR gauge algorithms).
symmetric	on off	off	Use symmetric form; ssor instead of sor, and so forth. (SOR, SORVector, SORGauge, SORLine).
thresh	scalar between 0 and 1	1	Pivot threshold (ILU).
transpose	on off	off	Use transposed form; soru instead of sor and so forth. (SOR, SORVector, SORGauge, SORLine).
hybridization	single multi	single	Use a single preconditioner or multiple preconditioner as a hybrid preconditioner.
hybridvar	vector of strings		Fields/States in step.
hybridvarspec	all manual	all	Include all components or specify which manually.
hybridcomp	vector of strings		Field/State components in step if hybridvarspec=manual.
vankablocked	on off	on	Blocked Vanka method.
vankarelay	scalar between 0 and 2	0.8	Relaxation factor for Vanka update.
vankarestart	positive integer	100	GMRES restart value (vanka).

TABLE 6-38: OPTIONAL ITERATIVE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
vankasolv	gmres direct	gmres	Local block solver (vanka).
vankatol	positive scalar	0.02	GMRES tolerance (vanka).
vankavars	vector of strings	{}	Lagrange multiplier variables (vanka).

The property `divcleantol` is used in the inequality $|T^T b| < \text{divcleantol} \cdot |b|$ to ensure that the numerical divergence after divergence cleaning is small enough; see [SOR Gauge](#) in the *COMSOL Multiphysics Reference Manual*.

MULTIGRID PROPERTIES

TABLE 6-39: VALID MULTIGRID PROPERTIES

PROPERTY	VALUES	DEFAULT	DESCRIPTION
gmglevels	positive integer	1	Maximum number of geometric multigrid levels.
iter	integer	2	Fixed number of iterations (when used as preconditioner, smoother, or coarse solver).
linsolver	gmg amg	gmg	Method to use.
maxlinit	positive integer	500	Maximum number of linear iterations (when used with a tolerance).
mgcycle	v w f	v	Cycle type.
mglevels	positive integer	5	Maximum number of algebraic multigrid levels.
prefun	gmg amg	amg if used as Krylov preconditioner. gmg, otherwise.	Solver for preconditioner (Geometric multigrid, Algebraic multigrid).
rhob	scalar > 1	400 1 (coarse solver)	Factor in linear error estimate.

TABLE 6-40: OPTIONAL MULTIGRID PROPERTIES

PROPERTY	VALUES	DEFAULT	DESCRIPTION
amgauto	integer from 1 to 10	3	Quality of multigrid hierarchy (amg)
geomuse	vector of strings		Geometries for geometric multigrid hierarchy
hybridization	single multi	single	Use a single preconditioner or multiple preconditioner as a hybrid preconditioner.
hybridvar	vector of strings		Fields/States in step
hybridvarspec	all manual	all	Include all components or specify which manually
hybridcomp	vector of strings		Field/State components in step if hybridvarspec=manual
massem	on off	on	Assemble on multigrid levels (gmg)
maxcoarsedof	positive integer	5000	Maximum number of DOFs at coarsest level (amg)
mcaseassem	vector of strings		Multigrid levels where assemble should be performed (gmg, mcasegen=manual)
mcasegen	manual all any coarse coarseorder refine refineany refineall	any	Hierarchy generation method (gmg)
mcaseuse	vector of strings		Multigrid levels which should be used (gmg, mcasegen=manual)
mkeep	on off	off	Keep generated mesh cases (gmg)

TABLE 6-40: OPTIONAL MULTIGRID PROPERTIES

PROPERTY	VALUES	DEFAULT	DESCRIPTION
<code>rmethod</code>	<code>regular</code> <code>longest</code>	<code>regular</code>	Mesh refinement method (gmg)
<code>scale</code>	vector of positive numbers	2	Mesh scale factor (gmg)

For the geometric multigrid solver/preconditioners, the multigrid hierarchy is controlled in the following way (see also [Multigrid](#) in the *COMSOL Multiphysics Reference Manual*):

- If `mcasegen=all`, `any`, or `coarse`, `coarseorder`, then the multigrid hierarchy is automatically constructed starting from the mesh and discretization set by the study. The number of multigrid levels generated is given in the property `gmglevels`. The method `all` and `any` first tries to lower the discretization order for the shape functions used, and secondly coarsens the mesh. The method `all` lowers the order (by one) if all used shape functions can be lowered. The method `any` lowers the order (by one) if at least one shape function can be lowered. The method `coarse` does not lower the order, it only coarsens the mesh. The method `coarseorder` both lowers the order (for any shape functions that can be lowered by one) and coarsen the mesh, at the same time.
- If `mcasegen=refine`, `refineany`, or `refineall` then the multigrid hierarchy is automatically constructed by a combination of refining the mesh given by the study and changing the discretization. The number of multigrid levels generated is given in the property `gmglevels`. The refinement method can be specified using the property `rmethod`. The originally selected mesh for the study is used, in the case of refining the mesh, in a multigridlevel and the finest multigrid level generated is used for the study (solved for). The generated multigrid levels are kept in the model and the `mcasegen` property is changed into `manual`. The method `refine` only refines the mesh and does not change the shape function order. The method `refineany` and `refineall` first tries to lower the order, and secondly refines the mesh. The method `refineany` constructs a multigrid level by lowering the order (by one) if at least on shape function can be lowered. The method `refineall` generates multigrid levels by lowering the order (by one) if all used shape function can be lowered.
- If `mcasegen>manual`, then the existing multigrid levels (children to the current study) can be used. The subset to use is selected by giving their tags to the `mcaseuse` property.

The construction of coarse level matrices is controlled by the property `massem` and `mcaseassem`. The first property controls if the matrices should be assembled for the automatically generated levels. If set to `off`, prolongation and restriction matrices are used to project the matrices from the top level in the hierarchy. The second property controls which multigrid levels that should use the assemble technique in the `mcasegen>manual` case.

When an iterative solver is used as preconditioner, smoother, or coarse solver you can choose whether to solve using a tolerance or to perform a fixed number of iterations. When used as a coarse solver the default is to solve using a tolerance. When used as a preconditioner or smoother the default is to perform a fixed number of iterations. If both properties `itol` and `iter` are given, the program solves using a tolerance.

DOMAIN DECOMPOSITION PROPERTIES

TABLE 6-41: VALID DOMAIN DECOMPOSITION PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
prefun	ddadd ddmul ddhyb ddsym	ddmul	Domain decomposition solver (Additive Schwarz/Multiplicative Schwarz/Hybrid Schwarz/ Symmetric Schwarz).
iter	integer	1	Number of iterations.
ndom	integer	2	Number of subdomains.
ddreordermethod	none space	space	Element reordering method: none or using a space-filling curve.
domdofmax	integer	1	Maximum number of nodes per subdomain.
domnodesmax	integer	100000	Maximum number of DOFs per subdomain.
overlap	integer	1	Additional overlap.
overlapmethod	auto matrix mesh	auto	Overlap method.
prefermatfree	on off	off	Automatically choose matrix free format.
usecoloring	on off	on	Use coloring (for prefun = ddmul ddsym).
userac	on off	off	Recompute and clear subdomain data.
domgeom	vector of strings		Partition geometries.
usecoarse	on off	on	Use coarse level.
mcasegen	coarseorder all any coarse refineall refineany refine manual	any	Coarse level generation method.
scale	scalar > 0	2.0	Mesh coarsening factor (for mcasegen = coarseorder any all coarse).
rmethod	longest regular	longest	Mesh refinement method (for mcasegen = refineall refineany refine).
geomuse	vector of strings		Use coarse level in geometries (for mcasegen = coarseorder all any coarse refineall refineany refine).
massem	on off	on	Assemble on coarse level (for mcasegen = coarseorder all any coarse refineall refineany refine manual).
mkeep	on off	on	Keep generated coarse level (for mcasegen = all any coarse).
mcaseuse	vector of strings		Coarse level that should be used (for mcasegen = manual).
hybridization	single multi	single	Use a single preconditioner or multiple preconditioner as a hybrid preconditioner.
hybridvar	vector of strings		Fields/states in step / Preconditioner variables (for hybridization = multi).
hybridvarspec	all manual	all	Include all components or specify which manually (for hybridization = multi).
hybridcomp	vector of strings		Field/state components in step / Preconditioner selection (for hybridvarspec= manual).

Specify the Coarse Solver/Domain Solver via `feature("cs")` and `feature("ds")`, respectively.

Example Code for Use With Java

```
model.sol("sol1").feature("s1").create("i1", "Iterative");
model.sol("sol1").feature("s1").feature("i1").create("dd1", "DomainDecomposition");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").feature("ds").
    create("mg1", "Multigrid");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").set("prefun", "ddadd");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").set("mcasegen", "refine");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").set("domnodesmax", "2");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").
    set("overlapmethod", "mesh");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").set("ndom", "16");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").
    set("domgeom", new String[]{"geom1"});
model.sol("sol1").feature("s1").feature("i1").feature("dd1").feature("cs").
    feature("dDef").set("linsolver", "pardiso");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").feature("ds").
    feature("mg1").set("gmglevels", "5");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").
    set("hybridization", "multi");
model.sol("sol1").feature("s1").feature("i1").feature("dd1").
    set("hybridcomp", new String[]{"comp1.u"});
```

SOR PREFUN PROPERTY

TABLE 6-42: SOR PREFUN PROPERTY

PROPERTY	VALUE	DEFAULT	DESCRIPTION
prefun	ssor sor soru	ssor	Solver (SSOR, SOR, or SORU).

SSOR GAUGE PREFUN PROPERTY

TABLE 6-43: SSOR GAUGE PREFUN PROPERTY

PROPERTY	VALUE	DEFAULT	DESCRIPTION
prefun	ssorgauge sorgauge sorugauge	sorgauge when used as presmoothing; sorugauge when used as postsmoothing; ssorgauge otherwise.	Solver (SSOR gauge, SOR gauge, or SORU gauge).

SOR VECTOR PREFUN PROPERTY

TABLE 6-44: SOR VECTOR PREFUN PROPERTY

PROPERTY	VALUE	DEFAULT	DESCRIPTION
prefun	ssorvec sorvec soruvec	sorvec when used as presmoothing; soruvec when used as postsmoothing; ssorvec otherwise.	Solver (SSOR vector, SOR vector, or SORU vector).

Modal

Solve parametric or time-dependent problem using the eigenmodal method.

SYNTAX

```
model.sol(sname).create(fname, "Modal")
model.sol(sname).feature(fname).set(pname, pvalue)
```


DESCRIPTION

Operation feature. The following property/values are accepted:

TABLE 6-45: VALID MODAL PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
analysistype	frequency transient	frequency	Solve for frequency response or transient response
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");)
control	String	user	Name of controlling study
dampratio	scalar numeric vector	0	Damping ratios for participating modes
eigsol	solution object		Pre-computed eigenpairs (or other vectors) to be used in the modal analysis
loadfact	string vector of strings	empty	Scalar function(s) for time and/or parameter dependent boundary conditions
message	String		The log message from the last solution process
modes	integer vector	all	Participating modes
outsollinearized	du u	du	Store the total solution (u) or deviation and linearization point (du), when analysistype=frequency and storelinpoint=off
plist	scalar numeric vector		Frequency list. Only applicable when analysistype has been set to frequency
pname	vector of strings		Parameter names
pout	plist fraction spread	plist	Use plist as it stands or modify in relation to the participating modes
rtol	scalar	0.01	Relative tolerance. Only applicable when analysistype has been set to transient
storelinpoint	on off	off	Whether to store the linearization point
tlist	scalar numeric vector		Time list. Only applicable when analysistype has been set to transient

For frequency response analysis nonconstant Neumann boundary conditions and constant Dirichlet boundary conditions are supported. The only allowed type of parameter dependent Dirichlet boundary condition are those that can be written as a constant vector times a scalar function. The scalar function is specified via the property `loadfact`. For transient response analysis only constant Dirichlet boundary conditions are supported. Neumann conditions that can be written as a constant vector times a scalar function (which is specified via the property `loadfact`) are supported for transient response.

The property `modes` is index 0 based.

If `pout` is set to `fraction` the output frequencies are the ones in `plist` multiplied by the absolute value of the largest eigenvalue in `eigsol` (or some other fraction of the largest participating eigenvalue of `eigsol`). The purpose of this property is to be able to automatically compute the frequency response for reasonable frequencies. If `pout` is set to `spread` then `plist` is interpreted as an interval around each participating eigenvalue. For example, if `plist` is set to `range(0.9,0.04,1.1)` then each participating eigenvalue is multiplied by this list and the resulting lists are concatenated into the `plist` that is used.

REMOVED PROPERTIES

TABLE 6-46: REMOVED PROPERTIES FOR THE MODAL SOLVER SEQUENCE FEATURE

PROPERTY	REASON
Callblevel	Given by the solver sequence attribute feature.

Optimization

Handle optimization solver properties.

SYNTAX

```
model.sol(sname).create(fname,"Optimization")
model.sol(sname).feature(fname).set(pname,value)
model.sol(sname).feature(fname).create(aname,SolverAttribute)
```

DESCRIPTION

Operation feature. Use this feature to solve PDE-constrained optimization problems. The computed solution object contains the PDE solution evaluated for the optimal set of design variables. When the gradient-evaluation method is analytic, it also returns the adjoint solution.

The Optimization Module includes SNOPT and several other solvers. See the *Optimization Module Manual* for details.

Choosing solver is done with the following property

TABLE 6-47: PROPERTY/VALUE PAIR TO SELECT OPTIMIZATION SOLVER

PROPERTY	VALUE	DEFAULT	DESCRIPTION
optsolver	bobyqa cobyla coordsearch lm mma montecarlo neldermead snopt	snopt	Optimization solver

When the optimization solver is set to SNOPT (snopt), the following property/value pairs are accepted:

TABLE 6-48: VALID PROPERTY/VALUE PAIRS FOR OPTSOLVER SNOPT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");).
feastol	real scalar	1e-6	Linear constraint tolerance.
funcprec	real scalar	3.8e-11	Function precision.
gradientmma	analytic forward adjoint forward_numeric	analytic	Gradient/Jacobian evaluation method for MMA.
gradientsnopt	analytic numeric	analytic	Gradient/Jacobian evaluation method for SNOPT.
hessupd	integer	10	Hessian updates.
majfeastol	real scalar	1e-6	Nonlinear constraint tolerance.
manualhessupd	on off	off	Whether to use property hessupd.
manualstepcond	on off	off	Whether to use manual step condition.
message	String		The log message from the last solution process.
nsolveimax	positive integer	10000	Maximum number of objective evaluations.
objcontrib	all manual	all	Whether to use all objective contributions present or specify manually.

TABLE 6-48: VALID PROPERTY/VALUE PAIRS FOR OPTSOLVER SNOPT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
optobj	String		Objective function that is minimized when objcontrib>manual.
opttol	real scalar	1e-6	Tolerance.
pdistrib	on off	off	Distribute parametric sweep.
plot	on off	off	Whether to plot while solving.
plotgroup	String	default	Name of plot group for plot while solving.
probes	vector of strings		Probes to use if probesel>manual.
probesel	all none manual	all	The probes to compute.
qpsolver	cholesky cg qn	cholesky	QP subproblem algorithm.
stepcond	String	empty	Manual step condition.

The property `gradientsnopt` is used to control if the gradient should be computed analytically (by solving the adjoint problem) or numerically. If the number of design variables is large, numerical computation of the gradient can be very time consuming. Analytic gradient is only supported when the underlying PDE-problem is stationary.

If `manualstepcond` is set to on, the expression in the property `stepcond` is evaluated when new values for the design variables have been computed. If the expression becomes negative, the new values are discarded and the optimization solver reduces the step length in the current line search.

When the optimization solver is set to Levenberg-Marquardt (1m), the following property/value pairs are accepted:

TABLE 6-49: VALID PROPERTY/VALUE PAIRS FOR OPTSOLVER LM

PROPERTY	VALUE	DEFAULT	DESCRIPTION
gradientlm	numeric	numeric	Gradient/Jacobian evaluation method.
gradorder	first second	first	Approximation order of the gradient.
lmfact	real scalar	1e-3	Initial Levenberg-Marquardt factor.
nsolveimax	positive integer	10000	Maximum number of objective evaluations.
opttol	real scalar	1e-6	Tolerance.
plot	on off	off	Whether to plot while solving.
plotgroup	String	default	Name of plot group for plot while solving.
probes	vector of strings		Probes to use if probesel>manual.
probesel	all none manual	all	The probes to compute.

When the optimization solver is set to MMA, the following property/value pairs are accepted:

TABLE 6-50: VALID PROPERTY/VALUE PAIRS FOR OPTSOLVER MMA

PROPERTY	VALUE	DEFAULT	DESCRIPTION
mmamaxiter	positive integer	1	Maximum outer iterations.
mmamaxiteractive	on off	off	Enable maximum outer iterations.
mmainmax	positive integer	10	Maximum inner iterations per outer iteration.
mmagepsfactor	real scalar	0.1	Internal tolerance factor.
mmacfactor	real scalar	1000	Constraint penalty factor.
mmaghinit	real scalar	0.5	Initial asymptote factor (ghinit).
mmaghdecr	real scalar	0.7	Decreasing asymptote factor (ghdecr).
mmaghincr	real scalar	1.2	Increasing asymptote factor (ghincr).
mmaasymin	real scalar	0.01	Minimum asymptote level (asymin).
mmaasymax	real scalar	10	Maximum asymptote level (asymax).

TABLE 6-50: VALID PROPERTY/VALUE PAIRS FOR OPTSOLVER MMA

PROPERTY	VALUE	DEFAULT	DESCRIPTION
mmaalbeta	real scalar	0.1	Bounds asymptote factor (albefa).
mmaxxmove	real scalar	0.5	Bounds control factor (xxmove).
mmaraai	real scalar	0.00001	Approximation increment (raai).
mmaraamin	real scalar	0.000001	Lower approximation bound (raamin).
mmalsq	on off	on	Automatically transform least-squares objectives to constraints for efficiency.
mmaminmax	on off	on	Automatically transform minimax and maximin problems to constraints for efficiency.
gmma	on off	on	Use the globally convergent MMA algorithm.

For a description of the optimization properties, see [Advanced Solver Properties](#) in the *Optimization Module User's Guide*.

TABLE 6-51: REMOVED FEMOPTIM PROPERTIES

PROPERTY	REASON
Callblevel	Handled by attribute features
Solprop	Handled by stationary or time
Solcomp	Handled by variables
Report	Handled by variables
Out	Solution should be exported

Parametric

Handle parameters for parameter stepping for stationary problems.

SYNTAX

```
model.sol(sname).create(fname, "Stationary")
model.sol(sname).feature(fname).create(parname, "Parametric")
model.sol(sname).feature(fname).feature(parname).set(pname, pvalue)
```

DESCRIPTION

Attribute feature.

TABLE 6-52: PARAMETRIC PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
control	String	user	Name of controlling study.
initdampall	off on auto	auto	Setting this property to on the initial damping factor is used for all parameter steps.
paramtuning	on off	off	Setting this property to on enables the use of the properties <code>pinitstep</code> , <code>pmaxstep</code> , and <code>pminstep</code> .
pdistrib	on off	off	If the solver should distribute the parameter sweep.
pinitstep	positive real		Initial step size for parameter. See <code>paramtuning</code> .
plist	real array		List of parameter values. Obsolete, use <code>plistarr</code> instead.
plistarr	real matrix		Lists of parameter values. One row of values for each parameter name.
pcontinuationmode	no last manual	last	Determines if a continuation sweep should be performed for one of the parameters in <code>pname</code> .

TABLE 6-52: PARAMETRIC PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
pcontinuation	String		For pcontinuationmode=manual this is one of the parameter names from pname.
pdistrib	on off	off	Distribute parametric sweep.
preusesol	no yes auto	no	Determines how the converged solutions are reused in the parameter sweep.
ponerror	stop empty	stop	Determines what the solver does when there is a solver error or when the continuation backtracking fails.
pmaxstep	positive real		Maximum step size for parameter. See paramtuning.
pminstep	positive real		Minimum step size for parameter. See paramtuning.
pname	vector of strings		Parameter names.
porder	constant linear	linear	Predictor order for parameter stepping.
pout	plist psteps	plist	Output either the parameters in plist or the solution at the expansion points.
plot	on off	off	Plot while solving.
plotgroup	String	default	Plot group to use for plot while solving.
probesel	all none manual	all	The probes to compute.
probes	array of strings		Probes to use when probesel=manual.
pwork	integer	1	Maximum number of distributed groups.
rstep	real scalar > 1	10	Restriction for step size update.
sweepstype	sparse filled	sparse	Method for doing the parameter variation. For sweepstype=sparse, the parameter tuples defined by the columns in plistarr are solved for. This method requires equal length for the rows. For sweepstype=filled, all parameter combinations given by plistarr are solved for.

Previous Solution

Compute solutions for previous parametric solution parameters and previous time-dependent solution parameters.

SYNTAX

```
model.sol(sname).create(fname, 'Stationary')
model.sol(sname).feature(fname).create(parname, 'Parametric')
model.sol(sname).feature(fname).feature(parname).create(psname, 'PreviousSolution')
model.sol(sname).feature(fname).feature(parname).feature(psname).set(pname, pvalue)
```

```
model.sol(sname).create(fname, 'Time');
model.sol(sname).feature(fname).create(psname, 'PreviousSolution');
model.sol(sname).feature(fname).feature(psname).set(pname, pvalue);
```

DESCRIPTION

Attribute feature. After the solver has converged for a parameter step or a time step, the previous components are solved for in a separate solver step. These components are held fixed (not solved for) during the normal solver procedure.

Segregated

Handle the segregated solution approach.

SYNTAX

```
model.sol(sname).feature(solv).create(fname, "Segregated")  
model.sol(sname).feature(solv).feature(fname).set(pname, value)  
model.sol(sname).feature(solv).feature(fname).feature(fname2).set(pname, value)
```

DESCRIPTION

This feature can be used as an attribute for the **Time** and **Stationary** features. The approach taken is nonlinear Uzawa iterations in which user-defined groups of variables are solved for separately (a segregated step) while other variables are held fixed. The segregated steps for the segregated solver is handled by subattributes of the sort **SegregatedStep**.

The Segregated attribute supports the following properties;

TABLE 6-53: VALID SEGREGATED PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
maxsegiter	positive integer	100, 25 (Time)	Maximum number of segregated iterations.
ntolfact	positive scalar	1	Tolerance factor.
plot	on off	off	Plot while solving.
plotgroup	String	default	Plot group to use for plot while solving.
probesel	all none manual	all	The probes to compute.
probes	array of strings		Probes to use when probesel>manual.
ratelimit	positive scalar	1	Limit on nonlinear convergence rate.
segaaccdim	positive integer	10	Dimension of Anderson iteration space when segstabacc = segaacc.
segiter	positive integer	1	Fixed number of segregated iterations.
segreserrfact	positive scalar	100	Residual factor for segtermmonres=auto.
segstabacc	none segclfcmp segaacc	none	Stabilization and acceleration: None, pseudo time stepping (for stationary solvers), or Anderson acceleration.
segterm	iter tol itertol	tol	Segregated solver termination technique.
segtermmonres	off on auto	auto	Termination criterion: Solution, residual, solution or residual.
subcfltol	positive scalar	0.1	Target error estimate for pseudo time stepping.
subinitcfl	positive scalar	5.0	Initial CFL number for pseudo time stepping.
subkdpid	positive scalar	0.05	PID regulator-Derivative for pseudo time stepping.
subkipid	positive scalar	0.05	PID regulator-Integrative for pseudo time stepping.
subkppid	positive scalar	0.65	PID regulator-Proportional for pseudo time stepping.
useratelimit	on off	off, on (time)	Use limit on nonlinear convergence rate.

Termination of the segregated solver is controlled by the property **segterm**. The default setting is **tol**, in which case the segregated iterations are terminated when, for each group, the estimated error is below the corresponding tolerance set by the main tolerance for the parent solver multiplied with the nonlinear tolerance factor **ntolfact**. However, a maximum number of allowed segregated iterations is chosen through the property **maxsegiter**; if the maximum is reached, the iterations are terminated and an error message is displayed. Termination after a fixed number of segregated iterations is achieved by instead choosing **iter**. The number of segregated iterations is controlled by the property **segiter**. The third available option for **segterm** is **itertol**, which is a combination of the other two options; the segregated iterations are terminated when one of the two convergence criteria of **tol**

and `iter` is met. The property `maxsegiter` is only supported when `tol` is used for termination. For both the settings `iter` and `itertol`, the number of iterations is controlled by the property `segiter`.

The nonlinear solver uses an adaptive tolerance for termination of iterative linear system solvers. This adaptive tolerance is based on the maximum of `ntol` and `itol`. During the nonlinear iterations, it can, however, be larger or smaller than this number. The segregated solver uses the same tolerance as the linear solver when constant damping is used. However, when automatically adjusted damping is used, the adaptive tolerance of the nonlinear solver is used. The parametric solver uses the same tolerance as the corresponding stationary solver.

The property `segstabacc` enables or disables pseudo time stepping (for stationary problems) or Anderson acceleration. When enabled the pseudo time stepping is controlled by the scalar-valued regulator parameters `subcfltol`, `subinitcfl`, `subkdpid`, `subkipid`, and `subkppid`. For the Anderson acceleration, the parameter `segaaccdim` specifies the dimension of the Anderson iteration space.

The property `segtermmonres` controls the termination criterion for stationary problems when `segterm` is not `iter`. When `segtermmonres=off` the estimated error is solution-based, with `segtermmonres=on` it is based on a relative residual and for `segtermmonres=auto` the estimated error is the minimum of the solution and residual based errors. For `segtermmonres=auto` the property `segreserrfact` is a scalar factor multiplying the relative residual error.

COMPATIBILITY

The property `subusecflcmp` from earlier versions of COMSOL Multiphysics is not used in version 5.0 and later versions. Use the property `segstabacc` instead.

SegregatedStep

Handle a segregated solution step.

SYNTAX

```
model.sol(sname).create(fname, "SegregatedStep")
model.sol(sname).feature(fname).set(pname, value)
model.sol(sname).feature(fname).feature(sname).set(pname, value)
```

DESCRIPTION

This feature controls one segregated solution step.

TABLE 6-54: VALID SEGREGATED STEP PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>maxsubiter</code>	integer	20	Maximum number of substep iterations
<code>segcomp</code>	vector of strings		Field/State components in step if <code>segvarspec=manual</code>
<code>segvar</code>	vector of strings		Fields/States in step
<code>segvarspec</code>	<code>all</code> <code>manual</code>	<code>all</code>	Include all components or specify which manually
<code>subdamp</code>	real	1.0	Substep damping factor
<code>subdtech</code>	<code>const</code> <code>auto</code> <code>hnlin</code> <code>ddog</code>	<code>const</code>	Substep damping technique
<code>subddoginitdamp</code>	non-negative scalar	1	Initial damping factor for <code>subdtech</code> set to <code>ddog</code>
<code>subinitstep</code>	real	1	Substep initial damping factor for <code>subdtech=auto</code>
<code>subinitsteph</code>	real	1e-4	Substep initial damping factor for <code>subdtech=hnlin</code>
<code>subiter</code>	integer	1	Substep iterations
<code>subjtech</code>	<code>minimal</code> <code>once</code> <code>onfirst</code> <code>onevery</code>	see below	Substep Jacobian update technique for <code>subdtech=const</code>
<code>subminstep</code>	real	1e-4	Substep minimum damping factor for <code>subdtech=auto</code>

TABLE 6-54: VALID SEGREGATED STEP PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
subminstep	real	1e-8	Substep minimum damping factor for subdtype=hnl
subntolfact	real	10	Substep tolerance factor
subreserrfact	positive scalar	100	Residual factor for subtermres=auto
subresscale	scalefieldwise scaleuniform	scalefieldwise	Residual scaling technique for subdtype set to ddog
subrstep	real	10	Substep restrictions for step-size update
subtermconst	iter tol itertol	iter	Substep termination technique for subdtype=const
subtermauto	tol itertol	itertol	Substep termination technique for subdtype=auto/hnl
subtermres	off on auto	auto	Termination criterion: Solution, residual, solution or residual.

The fields/states to include in the step is defined through the property `segvar`. The property `segvarspec` controls which components of the fields/states in `segvar` to include in the step. By default `segvarspec` is `all`, in which case all components in the fields/states of `segvar` are included. By setting `segvarspec` to `manual`, a subset of the fields/states of `segvar` can be included in the step. The components to include in the step are then defined through the property `segcomp`.

Analogously, the property `subterm` controls how each substep is terminated through the properties `maxsubiter`, `subiter`, and `subntol/subntolfact` for a stationary or time-dependent problem.

The damping technique used in each substep is controlled by the property `subdtype`. The default setting is `const`, which means that damped Newton iterations with a fixed damping factor is used. The damping factor is set in the property `subdamp`. The other available damping technique is `autodamp` in which case the damping factor is automatically adjusted. For substeps which uses `autodamp`, four other properties are supported: `subhnl`, `subinitstep`, `subminstep`, and `subrstep`. For each substep, these properties set the properties `hnl`, `initstep`, `minstep`, and `rstep` supported by the nonlinear solver, see `FullyCoupled`.

In substeps with `subdtype=const`, the property `subjtech` controls how often the Jacobian is updated. The values `minimal`, `once`, and `onevery` give the same Jacobian update techniques as they do when applied to the coupled solver through the property `jtech`; see `FullyCoupled`. The value `onfirst` makes the solver update the Jacobian of the substep on the first subiteration each time the substep is solved for. Default value is `onevery` for stationary problems and `minimal` for time-dependent problems.

When `subdtype` is set to `ddog` (stationary problems), the double dogleg solver is used. The initial damping factor is controlled by the property `subddoginitdamp` and the property `subresscale` controls the residual scaling. The option `resscale=scalefieldwise` scales the equations based on the field-wise sizes of the initial residual. When the option `subresscale=scaleuniform` is selected the algorithm terminates on the relative residual based on the initial residual.

The property `subtermres` controls the termination criterion for stationary problems when `segterm` is not `iter`. When `subtermres=off` the estimated error is solution-based, with `subtermres=on` it is based on a relative residual, and for `subtermres=auto` the estimated error is the minimum of the solution and residual based errors. For `subtermres=auto` the property `subreserrfact` is a scalar factor multiplying the relative residual error.

Sensitivity

Handle sensitivity solver parameters.

SYNTAX

```
model.sol(sname).feature(solv).create(fname, "Sensitivity")
model.sol(sname).feature(solv).feature(fname).set(pname, value)
```

DESCRIPTION

Attribute feature. This feature can be used to make analytic forward or backward sensitivity analysis. This analysis is done after the main problem has converged. The solution approach (coupled or segregated, Jacobians, and so on) for the main problem is reused.

TABLE 6-55: VALID SENSITIVITY PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sensfunc	String		Sensitivity functional variable name
sensmethod	none adjoint forward	none	Sensitivity analysis

StateSpace

Assemble and store matrices that describe a model as a dynamic system.

SYNTAX

```
model.sol(sname).create(fname, "StateSpace")
model.sol(sname).feature(fname).set(pname, value)
model.sol(sname).feature(fname).getSparseMatrixVal(mname)
model.sol(sname).feature(fname).getSparseMatrixValImag(mname)
model.sol(sname).feature(fname).getSparseMatrixRow(mname)
model.sol(sname).feature(fname).getSparseMatrixCol(mname)
model.sol(sname).feature(fname).getVector(vname)
model.sol(sname).feature(fname).getVectorImag(vname)
model.sol(sname).feature(fname).getSparseMatrixValBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixValImagBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixRowBlock(mname, start, stop)
model.sol(sname).feature(fname).getSparseMatrixColBlock(mname, start, stop)
model.sol(sname).feature(fname).getVectorBlock(vname, start, stop)
model.sol(sname).feature(fname).getVectorImagBlock(vname, start, stop)
model.sol(sname).feature(fname).isReal(mname)
model.sol(sname).feature(fname).getM(mname)
model.sol(sname).feature(fname).getN(mname)
```

DESCRIPTION

State-space feature.

TABLE 6-56: VALID GENERAL PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
input	String array		The input parameters that affect the model
output	String array		The output expressions
static	on off	on	Static linearized model
Mc	on off	off	Assemble the Mc matrix
MA	on off	off	Assemble the McA matrix
MB	on off	off	Assemble the McB matrix
C	on off	off	Assemble the C matrix
D	on off	off	Assemble the D matrix
Null	on off	off	Assemble the Null matrix
ud	on off	off	Assemble the ud vector
x0	on off	off	Assemble the initial data

The state-space feature assembles matrices that describe a model as a dynamic system when `Static` is `off`

$$\begin{aligned} Mc\dot{x} &= McAx + McBu \\ y &= Cx + Du \end{aligned}$$

In the case when `Static` is `on` a static linearized model of the system is described by

$$y = (D - C(McA)^{-1}McB)u$$

Let *Null* be the PDE constraint null-space matrix and *ud* a particular solution fulfilling the constraints. The solution vector *U* for the problem can then be written

$$U = \text{Null}x + ud + u0$$

where *u0* is the linearization point, which is the solution stored in the sequence once the state-space export feature is run. The input linearization point is stored in the sequence after the state-space feature is run.

The input parameters `input` should contain all parameters that are of interest as input to the model. The output expressions `output` should contain a list of all expressions that are to be evaluated as output from the model.

Stationary

Solve a stationary problem with or without parameters, mesh adaptation, sensitivity, or optimization.

SYNTAX

```
model.sol(sname).create(fname, "Stationary")
model.sol(sname).feature(fname).set(pname, value)
```

DESCRIPTION

Operation feature.

The following (intrinsic) properties are available.

TABLE 6-57: STATIONARY PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>const</code>	String array		Provide constants as input parameters using an N-by-2 string array (for example, <code>.set("const", "a 1.0 b 1.0");</code>).
<code>control</code>	String	<code>user</code>	Name of controlling study.
<code>keepnotsolstatic</code>	<code>on</code> <code>off</code>	<code>off</code>	Used in Fatigue analysis to store all solnums from the source solution in the Fatigue solution object (default value is <code>on</code> for Fatigue analysis).
<code>linpmethod</code>	<code>init</code> <code>sol</code>	<code>init</code>	Method used for linearization point.
<code>linpsol</code>	<code>zero</code> solution object	<code>zero</code>	Linearization point solution.
<code>linpsoluse</code>	<code>current</code> solution store	<code>current</code>	
<code>lumpedflux</code>	<code>on</code> <code>off</code>	<code>off</code>	Use lumping when computing fluxes.
<code>nonlin</code>	<code>auto</code> <code>on</code> <code>off</code> <code>linper</code>	<code>auto</code>	Use the nonlinear solver.
<code>message</code>	String		The log message from the last solution process.
<code>outsollinear</code>	<code>du</code> <code>u</code>	<code>u</code>	Store the total solution (<i>u</i>) or deviation and linearization point (<i>du</i>), when <code>nonlin=off</code> and <code>storelinpoint=off</code> .

TABLE 6-57: STATIONARY PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
outsollinearized	du u	du	Store the total solution (u) or deviation and linearization point (du), when <code>nonlin=linper</code> and <code>storelinpoint=off</code> .
pdistrib	on off	off	Distribute parametric sweep.
reacf	on off	on	Compute reaction forces.
solnum	auto all interp first last manual positive integer	auto	Which solnums from another solution to use as linearization point.
stol	positive real	1e-3	Relative tolerance.
storelinpoint	on off	off	Whether to store the linearization point.
t	real	0	Interpolation time for linearization point from another solution, when <code>solnum=interp</code> .

This solver uses a nonlinear solver if `nonlin` is `on`, and it uses the linear solver if `nonlin` is `on` or `linper`. If `nonlin` is set to `auto` an analysis is performed to automatically detect if the problem can be solved with a linear solver approach. For a description of the nonlinear solver see the entry under `coupling`.

The automatic nonlinear/linear detection works in the following way. The linear solver is called if the residual Jacobian matrix (the stiffness matrix, K) and the constraint Jacobian matrix (the constraint matrix, N) are both found not solution dependent and if these matrices are detected as complete. In all other situations the nonlinear solver is used. The analysis is performed by a symbolic analysis of the expressions contributing to these matrices. Complete here means that in the residual and constraint vectors, only expressions were found for which COMSOL Multiphysics computes the correct Jacobian contribution.

Therefore, if you want to solve a linearized (nonlinear) problem, you must set `nonlin` to `off` or `linper`. The `off` option uses the linearization point for both the residual computation and for the Jacobian and the solution to the liner problem is added to the linearization point. This corresponds to one step in the Newton method. For `linper`, the linearization point is used for the Jacobian, the zero solution is used for the assembly of the residual and the solution to the linear problem is returned as the solution. Furthermore, the residual assembled for `linper` is computed using loads marked with the `linper` operator.

There are variables for which COMSOL Multiphysics is conservative and therefore flags these, and their Jacobian contribution, as solution dependent even though they not always are. For these situations, the nonlinear solver is used even though the linear solver could be used. This should only result in some extra computational effort, and should not influence the result. The opposite situation however, where the linear solver is used for a nonlinear problem is more dangerous. So, select `nonlin` to `off` with great care.

The property `reacf` controls the computation and storage of constraint reaction forces. The value `reacf=on` (default) means that the solver stores the FEM residual vector L in the solution object `model.sol`. Because $L = N_F \Lambda$ for a converged solution, the residual is the same as the constraint force. Only the components of L that correspond to nonzero rows of N_F are stored. The value `reacf=off` gives no computation or storage of the reaction force and saves some memory.

The linear solver uses the property `itol` for termination of iterative linear system solvers and for error checking for direct solvers (if enabled).

StopCondition

Handle stop conditions for time-dependent and parametric solver processes.

SYNTAX

```
model.sol(sname).create(fname, "Stationary")
model.sol(sname).feature(fname).create(pname, Parametric)
model.sol(sname).feature(fname).feature(pname).create(ocname, StopCondition)
model.sol(sname).feature(fname).feature(pname).feature(ocname).set(pname, pvalue)
```

DESCRIPTION

Attribute feature. Use the property `Stopcond` feature to make sure the solver stops when a specified condition is fulfilled. When you provide a scalar expression, then the expression is evaluated after each time or parameter step. The stepping is stopped if the real part of the expression is evaluated to something negative. The corresponding solution, for which the expression is negative is not returned. When you provide an integer the solver stops when the corresponding implicit event is triggered.

TABLE 6-58: VALID STOPCONDITION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
stopcondition	String		Stop condition expression

StoreSolution

A placeholder for a solver sequence that is used to store a computed solution.

SYNTAX

```
model.sol(sname).create(fname, 'StoreSolution')
model.sol(sname).feature(fname).getString('sol')
```

DESCRIPTION

The store solution feature stores a reference to a computed solution. Use the `sol` property to find out the name of the referenced solution.

TABLE 6-59: VALID STORESOLUTION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
sol	String	A solver sequence	Name of solver sequence that stores the solution

In the case of a parametric sweep you can use the `StoreSolution` features to find the solutions created during the sweep. You first find out the solver sequence that holds the stored solutions

```
model.batch(pname).feature(fname).getString('psol')
```

where `pname` is the name of the parametric sweep feature that ran and `fname` is the name of the solution feature that stored the solutions. Use

```
model.sol(sname).feature().tags()
```

to find out the tags of the stored solutions. Use

```
model.sol(sname).feature(fname).getString('sol')
```

to find the solver sequence for a parameter. Use

```
model.sol(sname).getParamNames()
```

and

```
model.sol(sname).getParamVals()
```

to find the parameter values that created the solution object.

StudyStep

Specify which problem to use for subsequent solver operations.

SYNTAX

```
model.sol(sname).create(fname, "StudyStep")
model.sol(sname).feature(fname).set(pname, pvalue)
```

DESCRIPTION

Utility feature. This feature determines which problem to use for subsequent solver operations. It contains a reference to a study and a reference to a study step within that study. When run, the corresponding low-level equation representation is compiled.

TABLE 6-60: VALID CONFIGURATION PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
study	String		Name of study to use
studystep	String		Name of study step to use
splitcomplex	on off	off	Represent complex variables by separate degrees of freedom for real and imaginary parts

Time

Solve a time-dependent problem.

SYNTAX

```
model.sol(sname).create(fname, 'Time')
model.sol(sname).feature(fname).set(pname, value)
```

DESCRIPTION

Operation feature.

The time interval and possible intermediate time values are given in the property Tlist. The output times are controlled by the property Tout.

The feature Time accepts the following property/values:

TABLE 6-61: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
atol	String	empty	Absolute tolerance per field. See below.
atolmethod	String	empty	How to interpret the atolfields value. See below.
atolglobal	positive scalar	1e-3	Global absolute tolerance.
atolglobalmethod	scaled unscaled	scaled	How to interpret the atolglobal value.
atoludot	String	empty	Absolute tolerance for time derivatives per field. Only applicable if atoludotactive is on. See below.
atoludotactive	String	empty	Used to activate manual specification of absolute tolerance for time derivatives. See below.
complex	on off	off	Allow complex numbers.
consistent	off on bweuler	bweuler	Consistent initialization of DAE systems.

TABLE 6-61: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");)
control	String	user	Name of controlling study.
dopriGrowmax	positive scalar	10	Maximum step size growth ratio for Dormand-Prince 5.
dopriGrowmin	positive scalar	0.2	Minimum step size growth ratio for Dormand-Prince 5.
dopriPicontrol	smooth quick disabled	smooth	Control behavior of the proportional-integral controller that adds damping on step size changes for Dormand-Prince 5.
dopriSafe	positive scalar	0.9	Step size safety factor for Dormand-Prince 5.
estrat	include exclude	include	Error estimation strategy.
ewtrescale	on off	on	Update scaled absolute tolerance for BDF.
incrDelay	positive integer	15	Number of time steps to delay a time step increase.
incrDelayactive	on off	off	Use delay in time step increase.
initialStepbdf	positive scalar	1e-3	Initial time step for BDF.
initialStepbdfactive	on off	off	Use an initial time step for BDF.
initialStepck5	positive scalar	1e-3	Initial time step for Cash-Karp 5.
initialStepdopri5	positive scalar	1e-3	Initial time step for Dormand-Prince 5.
initialStepprk34	positive scalar	1e-3	Initial time step for RK34.
initialStepck5active	on off	off	Use an initial time step for Cash-Karp 5.
initialStepdopri5active	on off	off	Use an initial time step for Dormand-Prince 5.
initialStepprk34active	on off	off	Use an initial time step for RK34.
initialStepgenalpha	positive scalar	1e-3	Initial time step for generalized alpha.
initialStepgenalphaactive	on off	off	Use an initial time step for generalized alpha.
lumpedflux	on off	off	Use lumping when computing fluxes.
masssingular	yes maybe	maybe	Singular mass matrix.
maxorder	integer between 1 and 5	5	Maximum BDF order.
maxStepbdf	positive scalar	1e-1	Maximum time step BDF.
maxStepbdfactive	on off	off	Use a maximum time step for BDF.
maxStepdopri5	positive scalar	1e-1	Maximum time step Dormand-Prince 5.
maxStepdopri5active	on off	off	Use a maximum time step for Dormand-Prince 5.
maxStepgenalpha	positive scalar	1e-1	Maximum time step generalized alpha.
maxStepgenalphaactive	on off	off	Use a maximum time step for generalized alpha.
message	String		The log message from the last solution process.
minorder	1 2	1	Minimum BDF order.
nlsolver	automatic manual	manual	Nonlinear solver settings.

TABLE 6-61: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
plot	on off	off	Plot while solving.
plotfreq	tsteps tout	tout	Times to update plot.
plotgroup	String		Name of plot group for plot while solving.
probefreq	tsteps tout	tsteps	Times to update probe.
probes	vector of strings		Probed to use if probesel=manual.
probesel	all none manual	all	The probes to compute.
predictor	linear constant	linear	Predictor type to use (linear or constant),
reactf	on off	on	Compute reaction forces.
rhoinf	numeric	0.75	Amplification factor for high frequencies.
rkmeth	rk34 ck5 dopri5	rk34	Runge-Kutta method: RK34, Cash-Karp 5, or Dormand-Prince 5. Only available when timemethod is set to rk.
rkstiffcheck	on off	on	Check for and stop if problem becomes numerically stiff for Runge-Kutta solvers.
rtol	numeric	0.01	Relative tolerance.
solfile	on off	off	Store solution on file.
solfileblock	positive scalar	16	Max size of solution block (MB).
storeudot	on off	on	Store time derivatives.
timemethod	bdf rk genalpha init	bdf	Time-stepping method.
timestepgenalpha	numeric scalar numeric vector string with expression	0.01	Time step when manual time stepping.
tlist	numeric vector		Time list.
tout	tlist tsteps	tlist	Output times.
tstepsbdf	free intermediate strict	free	Time-stepping mode when rkmeth is set to bdf.
tstepsdopri5	free intermediate strict manual	free	Time-stepping mode when rkmeth is set to dopri5.
tstepsgenalpha	free intermediate strict manual	free	Time-stepping mode when rkmeth is set to genalpha.

By default, you can control the process of solving the linear or nonlinear system of equations in each time step manually. For a coupled problem, this is done through the properties `Damp`, `Dtech`, `Hnlin`, `Initstep`, `Jtech`, `Maxiter`, `Minstep`, and `Rstep` listed under `femnlin`. For a segregated problem, the properties listed under `femstatic` that are related to the segregated solver are available. When `Timemethod` is set to `bdf` it is possible to use the internal nonlinear solver of the time integrator. This can be achieved by setting `Nlsolver` to `automatic`.

The properties `atol`, `atolmethod`, `atolglobal`, `atolglobalmethod`, `atoludot`, and `atoludotactive` require some additional explanation. The default value of the absolute tolerance for all fields is given by the property `atolglobal`. The modifier `atolglobalmethod` specifies whether the given value of `atolglobal` should be applied to scaled or unscaled variables. For variables where the automatic scaling does a good job, or where a manual scaling has been used, specifying the absolute tolerance in scaled variables is much easier. If either a different absolute value or scaling method than dictated by `atolglobal` and `atolglobalmethod` is wanted for one or several variables you can use the properties `atol` and `atolmethod`. Enter `atol` as a space separated string with alternating field names

and tolerances (for example, “u 1e-3 v 1e-6”). Enter `atolmethod` as a space-separated string with alternating field names and one of the strings `global`, `scaled`, or `unscaled` (for example, “u unscaled v scaled”). By default `atolmethod` is equal to `global` for all fields. The lists `atol` and `atolmethod` do not have to contain all fields. The ones not present get absolute tolerances as specified by `atolglobal` and `atolglobalmethod`. When solving wave-type equations with `timemethod` set to `bdf` the time-derivatives of all fields are also treated as unknowns and therefore absolute tolerances have to be specified also for these components. By default these tolerances are chosen automatically. In some situations it might be necessary to specify them manually with the properties `atoludot` and `atoludotactive`. To turn on manual specification for, say, the two fields `u` and `v` set the property `atoludotactive` to the string “u on v on”. If `atoludot` is not specified these two time-derivatives get the default absolute tolerance `1e-3`. To specify other absolute tolerances set `atoludot` to, for instance, the string “u 1e-4 v 1e-7”. The absolute tolerance method for all time derivatives is the same as the method specified for the field itself.

The maximum allowed relative error in each time step (the local error) is specified using `rtol`. However, for small components of the solution vector U , the algorithm tries only to reduce the absolute local error in U below the given absolute tolerance.

There is no guarantee that the error tolerances are met strictly; that is, for hard problems they can be exceeded.

For the tolerance parameter in the convergence criterion for linear systems, the maximum of the numbers `rtol` and `itol` is used.

Use `complex=on` if complex numbers occur in the solution process.

The property `Consistent` controls the consistent initialization of a *differential algebraic equation* (DAE) system. The value `Consistent=off` means that the initial values are consistent (this is seldom the case because the initial value of the time derivative is 0). Otherwise, the solver tries to modify the initial values so that they become consistent. The value `consistent=on` can be used (when `timemethod=bdf` and `nlsolver=automatic`) for index-1 DAEs. Then the solver fixes the values of the differential DOFs and solves for the initial values of the algebraic DOFs and the time derivative of the differential DOFs. The value `Consistent=bweuler` can be used for both index-1 and index-2 DAEs. Then the solver perturbs the initial values of all DOFs by taking a backward Euler step.

For a DAE system, if `Estrat=exclude`, then the algebraic DOFs are excluded from the error norm of the time discretization error.

You can suggest a size of the initial time step using the property `initialstepbdf` when `timemethod` is set to `bdf` the property `initialstepdopri5` when `timemethod` is set to `dopri5`, and the property `initialstepgenalpha` when `timemethod` is set to `genalpha`. You also have to set one of the properties `initialstepbdfactive`, `initialstepdopri5active`, or `initialstepgenalphaactive` to `on` for the specified initial step to be active.

By default, the solver determines whether the system is differential-algebraic by looking after zero rows or columns in the mass matrix. If you have a DAE where the mass matrix has no zero rows or columns, put `masssingular=yes`.

The property `maxorder` gives the maximum degree of the interpolating polynomial in the BDF method (when `timemethod=bdf`).

The properties `maxstepbdf` (for `timemethod=bdf`), `maxstepdopri5` (for `timemethod=dopri5`), and `maxstepgenalpha` (for `timemethod=genalpha`) put an upper limit on the time step size (this property is not allowed when `tstepsgenalpha>manual`). You also have to set one of the properties `maxstepbdfactive`, `maxstepdopri5active`, or `maxstepgenalphaactive` to `on` for the specified maximal step to be active.

The property `timemethod` is used to select which time-stepping method to use:

- With `timemethod=bdf`, the IDA solver (which uses variable order backward differentiation formula) is used.

- With `timemethod=dopri5`, the Dormand-Prince 5 (DOPRI5) method for solving ODEs is used (a method of the Runge-Kutta type).
- With `timemethod=genalpha`, the generalized- α method is used. With generalized- α , the numerical damping can be controlled by giving a value, $0 \leq \rho_\infty \leq 1$, by which the amplitude of the highest possible frequency is multiplied each time step (hence, a small value corresponds to large damping while a value close to 1 corresponds to little damping). This is done through the property `rhoinf`. Also, the initial guess for the solution at the next time step (needed by the nonlinear solver) can be controlled through the property `predictor` when generalized- α is used. With `predictor=linear`, linear extrapolation using the current solution and time-derivative is used. With `predictor=constant`, the current solution is used as initial guess.
- When `timemethod` is set to `init` the solver computes consistent initial values (for the start time, as defined by the property `tlist`) for the system and then stop. Time derivatives of algebraic variables and indicator functions might still be uninitialized after this operation. Such uninitialized quantities are represented by NaN (not a number) in the solution object.

The property `reactf` controls the computation and storage of the constraint reaction force. The value `reactf=on` (default) means that the solver stores the FEM residual vector L in the solution object. Because $L = N_F \Lambda$ for a converged solution, the residual is the same as the constraint force. Only the components of L that correspond to nonzero rows of N_F are stored. For each time for which the solution is requested an extra residual vector assembly is performed. The value `reactf=off` gives no computation or storage of the reaction force and can therefore save some computational time.



The property `tlist` must be a strictly monotone vector of real numbers. Commonly, the vector consists of a start time and a stop time. If more than two numbers are given, the intermediate times can be used as output times, or to control the size of the time steps (see below). If just a single number is given, it represents the stop time, and the start time is 0.

The property `tout` determines the times that occur in the output. If `tout=tsteps`, then the output contains the time steps actually taken by the solver. If `tout=tlist`, then the output contains interpolated solutions for the times in the `tlist` property. The default is `tout=tlist`.

The properties `tstepsbdf` (applicable when `timemethod=bdf`), `tstepsdopri5` (applicable when `timemethod=dopri5`), and `tstepsgenalpha` (applicable when `timemethod=genalpha`) control the selection of time steps. If either of these properties is set to `free`, the solver selects the time steps according to its own logic, disregarding the intermediate times in the `tlist` vector. If either of the properties is set to `strict`, then time steps taken by the solver contain the times in `tlist`. If either of the properties is set to `intermediate`, then there is at least one time step in each interval of the `tlist` vector. If `tstepsgenalpha` has been set to `manual`, the solver follows the time step specified in the property `timestepgenalpha`. If `timestepgenalpha` is a scalar value, this time step is taken in the entire simulation. When `timestepgenalpha` is a (strictly monotone) numeric vector, the solver computes the solution at the times in the vector. The start time and stop time is still obtained from `tlist`; the vector given in `timestepgenalpha` is truncated and/or expanded using the first and/or last time step in the vector so that the start time and stop time agrees with the values in `tlist`. Finally, an expression using variables with global scope and which results in a scalar can be used as `timestepgenalpha`.

For problems of wave type, the logic by which the solver selects the time step can sometimes result in a time step that oscillates in an inefficient manner. When `timemethod=genalpha` (the solver typically used for wave-type problems), you can avoid such oscillations in the time step using the properties `incrdelay` and `incrdelayactive`. When `incrdelayactive=on`, a counter keeps track of the number of consecutive time steps for which a time step

increase has been warranted. When this counter exceeds the number given in the property `incrdelay`, the time step is increased and the counter is set to zero.

	In structural mechanics models, the displacements are often quite small, and it is critical that the absolute tolerance is chosen to be smaller than the actual displacements.
	For more information about the Time-Dependent solver; see Time-Dependent Solver in the <i>COMSOL Multiphysics Reference Manual</i> .

TimeDiscrete

Solve a time-discretized problem.

SYNTAX

```
model.sol(sname).create(fname, "TimeDiscrete")
model.sol(sname).feature(fname).set(pname, value)
```

DESCRIPTION

Operation feature.

The time interval and possible intermediate time values are given in the property `tlist`. The output times are controlled by the property `tout`.

The feature `TimeDiscrete` accepts the following property/values:

TABLE 6-62: VALID PROPERTY/VALUE PAIRS FOR THE TIME-DISCRETE SOLVER

PROPERTY	VALUES	DEFAULT	DESCRIPTION
<code>atol</code>	String	empty	Absolute tolerance per field. See below.
<code>atolmethod</code>	String	empty	How to interpret the <code>atolfields</code> value. See below.
<code>atolglobal</code>	positive scalar	1e-3	Global absolute tolerance.
<code>atolglobalmethod</code>	scaled unscaled	scaled	How to interpret the <code>atolglobal</code> value.
<code>const</code>	String array		Provide constants as input parameters using an N-by-2 string array (for example, <code>.set("const", "a 1.0 b 1.0");</code>).
<code>control</code>	String	user	Name of controlling study.
<code>message</code>	String		The log message from the last solution process.
<code>plot</code>	on off	off	Plot while solving.
<code>plotfreq</code>	tsteps tout	tout	Times to update plot.
<code>plotgroup</code>	String		Name of plot group for plot while solving.
<code>prevlevels</code>	positive integer	2	Number of previous time levels to store.
<code>probefreq</code>	tsteps tout	tsteps	Times to update probe.
<code>probes</code>	vector of strings		Probed to use if <code>probesel=manual</code> .
<code>probesel</code>	all none manual	all	The probes to compute.
<code>rtol</code>	numeric	0.01	Relative tolerance.
<code>solfile</code>	on off	off	Store solution on file.
<code>solfileblock</code>	positive scalar	16	Max size of solution block (MB).

TABLE 6-62: VALID PROPERTY/VALUE PAIRS FOR THE TIME-DISCRETE SOLVER

PROPERTY	VALUES	DEFAULT	DESCRIPTION
timestepdiscrete	numeric scalar numeric vector string with expression	0.01	Time step when manual time stepping.
tlist	numeric vector		Time list.
tout	tlist tsteps	tlist	Output times.

The `TimeDiscrete` solver is used for solving time-dependent PDEs that have already been discretized in time using, for example, the `prev` operator or the `bdf` operator. Such discretization requires the solution at previous time steps. Different discretizations require different number of previous time steps. For example, the first order accurate `bdf` method requires the solution at the previous time step, while the second-order accurate `bdf`-method requires the solution at the two preceding time steps. How many previous time steps should be accessible to the solver is controlled through the property `prevlevels`.

You can control the process of solving the linear or nonlinear system of equations in each time step manually. For a coupled problem, this is done through the properties `Damp`, `Dtech`, `Hnlin`, `Initstep`, `Jtech`, `Maxiter`, `Minstep`, and `Rstep` listed under `femnlin`. For a segregated problem, the properties listed under `femstatic` that are related to the segregated solver are available.

Because only manual time stepping is available, there is no estimation of the error made in a time step. However, the tolerances, specified through the properties `rtol`, `atol`, `atolmethod`, `atolglobal`, and `atolglobalmethod` are still important as tolerances when solving the nonlinear system of equations in each time step. For a description of these properties, see [Time](#). They should in general be set to the desired accuracy in the final solution.

The property `tlist` must be a strictly monotone vector of real numbers. Commonly, the vector consists of a start time and a stop time. If more than two numbers are given, the intermediate times can be used as output times, or to control the size of the time-steps (see below). If just a single number is given, it represents the stop time, and the start time is 0.

The property `tout` determines the times that occur in the output. If `tout=tsteps`, then the output contains the time steps actually taken by the solver. If `tout=tlist`, then the output contains interpolated solutions for the times in the `tlist` property. The default is `tout=tlist`.

The size of the time step is controlled through the property `timestepdiscrete`. If `timestepdiscrete` is a scalar value, this time step is taken in the entire simulation. When `timestepdiscrete` is a (strictly monotone) numeric vector, the solver computes the solution at the times in the vector. The start time and stop time is still obtained from `tlist`; the vector given in `timestepdiscrete` is truncated and/or expanded using the first and/or last time step in the vector so that the start time and stop time agrees with the values in `tlist`. Finally, an expression using variables with global scope and which results in a scalar can be used as `timestepdiscrete`.



For more information about the time discrete solver; see [Time Discrete Solver](#) in the *COMSOL Multiphysics Reference Manual*.

TimeExplicit

Solve time-dependent problems with explicit time stepping.

SYNTAX

```
model.sol(sname).create(fname,"TimeExplicit")
model.sol(sname).feature(fname).set(pname,pvalue)
```

DESCRIPTION

Operation feature. The `TimeExplicit` solver is used for solving time-dependent PDEs using the classic Runge-Kutta or the Adams-Bashforth 3 explicit time-stepping schemes

TABLE 6-63: VALID TIMEEXPLICIT PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
<code>const</code>	String array		Provide constants as input parameters using an N-by-2 string array (for example, <code>.set("const", "a 1.0 b 1.0");</code>).
<code>control</code>	String	<code>user</code>	Name of controlling study.
<code>erkorder</code>	integer between 1 and 4	4	Runge-Kutta order.
<code>exprs</code>	String		Expression for time stepping when <code>tstepping=elemexprs</code> .
<code>linsolver</code>			Linear solver
<code>message</code>	String		The log message from the last solution process.
<code>odesolver</code>	<code>erk</code> <code>ab3</code> <code>ab3loc</code>	<code>erk</code>	Time explicit solver.
<code>plot</code>	<code>on</code> <code>off</code>	<code>off</code>	Plot while solving.
<code>plotfreq</code>	<code>tsteps</code> <code>tout</code>	<code>tout</code>	Times to update plot.
<code>plotgroup</code>	String		Name of plot group for plot while solving.
<code>probefreq</code>	<code>tsteps</code> <code>tout</code>	<code>tsteps</code>	Times to update probe.
<code>probes</code>	vector of strings		Probed to use if <code>probesel>manual</code> .
<code>probesel</code>	<code>all</code> <code>none</code> <code>manual</code>	<code>all</code>	The probes to compute.
<code>rktimestep</code>	positive scalar	<code>1e-3</code>	Time step.
<code>solfile</code>	<code>on</code> <code>off</code>	<code>on</code>	Store solution out-of-core.
<code>storeudot</code>	<code>on</code> <code>off</code>	<code>on</code>	Store time-derivatives.
<code>tlist</code>	vector of strings		Specified time list.
<code>tout</code>	<code>tsteps</code> <code>tlist</code>	<code>tlist</code>	Times to store in solution.
<code>tstepping</code>	<code>manual</code> <code>elemexprs</code>	<code>manual</code>	Manual or from expressions time stepping.

The order of the Runge-Kutta method can be set by the `erkorder` property. The size of the time step is controlled through the property `rktimestep` and can be given as a single scalar value, a (strictly monotone) numeric vector, or an expression using variables with global scope, which results in a scalar. For Adams-Bashforth 3 only a scalar constant value of the time step is allowed. Time stepping from expressions `tstepping=elemexprs` is useful for the Wave form PDE interface. A local time stepping version of Adams-Bashforth 3 is available for the Wave form PDE interface by `odesolver=ab3loc`.

TimeParametric

Handle properties for parameter stepping for a time-dependent problem.

SYNTAX

```
model.sol(sname).create(fname, 'Time')
model.sol(sname).feature(fname).create(parname, 'TimeParametric')
model.sol(sname).feature(fname).feature(parname).set(pname, pvalue)
```

DESCRIPTION

Attribute feature.

TABLE 6-64: PARAMETRIC PROPERTIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
control	String	user	Name of controlling study
pdistrib	on off	off	If the solver should distribute the parameter sweep
plist	real array		List of parameter values. Obsolete, use plistarr instead.
plistarr	real matrix		Lists of parameter values. One row of values for each parameter name.
pname	vector of strings		Parameter names
pwork	integer	1	Maximum number of distributed groups
sweepstype	sparse filled	sparse	Method for doing the parameter variation. For sweepstype=sparse, the parameter tuples defined by the columns in plistarr are solved for. This method requires equal length for the rows. For sweepstype=filled, all parameter combinations given by plistarr are solved for.

Variables

Handle initial data and scaling for variables solved, as well as how variables not solved for are computed. The methods are applied for the dependent variables present as Field or State subattributes. These attributes are automatically created and updated by this feature. So, if the Analysis (for the solver sequence) is altered or if a different Analysis is used in the sequence, then the Field attributes are changed accordingly.

SYNTAX

```
model.sol(sname).create(fname, 'Variables')
model.sol(sname).feature(fname).set(pname, pval)
model.sol(sname).feature(fname).feature(varname).set(pname, pval)
```

DESCRIPTION

Operation feature. Computes the initial values for the variables that are solved for and how the variables not solved for are computed. The variables handled are the ones present as Field or State attributes. The feature also handles scaling and which variables to store in output.

Attribute feature.

TABLE 6-65: VALID VARIABLE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
const	String array		Provide constants as input parameters using an N-by-2 string array (for example, .set("const", "a 1.0 b 1.0");).
control	String	user	Name of controlling study.
initmethod	init sol	init	Method used for initial value computation.
initsol	zero solution object	zero	Initial value solution object.
initsoluse	current solution store	current	Use current or stored values from the initial value solution object.
notsol	zero solution object	zero	Solution object for variables not solved for.
notsolmethod	init sol	init	Method used for variables not solved for.
notsolnum	auto all positive integer	auto	Which solnums from other solution to use for variables not solved for.

TABLE 6-65: VALID VARIABLE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
notsoluse	current solution store	current	Use current or stored values from the solution object for variables not solved for.
scalemethod	auto init none manual	auto	Method used for scaling of variables.
scaleval	scalar	1	Global scaling value.
solnum	auto first last positive integer	auto	The solnum from other solution to use for initial values of variables solved for.
useinitsol	on off	off	Use user-controlled initial values of variables solved for.
usesol	on off	off	Use user-controlled values of variables not solved for.

TABLE 6-66: VALID VARIABLE SUBATTRIBUTE FIELD/STATE PROPERTY/VALUE PAIRS

PROPERTY	VALUES	DEFAULT	DESCRIPTION
comp	vector of strings		Field/State components
out	on off	on	Store Field/State in output
scalemethod	auto init none manual parent	parent	Method used for scaling of variables
scaleval	scalar	1	Scaling value
solvefor	on off	on	Solve for this Field/State
storesel	Array of selection strings		Selections defining which field data to store.
usestoresel	all selection	all	Store all field data or field data defined in selections in storesel.

The properties `initmethod`, `initsol`, and `initsoluse` determine the initial value for the solution components you solve for when `useinitsol` is set to on.

The properties `notsolmethod`, `notsol`, `notsoluse`, and `notsolnum` determine the value of solution components you do not solve when `usesol` is set to on.

Which variables to solve for, and which variables to store in the solution is controlled by the properties `solvefor` and `out` in the Field or State attributes.

The properties `scalemethod` and `scaleval` determine a scaling of the degrees of freedom that is applied in order to get a more well-conditioned system. The possible values of `scalemethod` are:

TABLE 6-67: VALUES FOR THE PROPERTY SCALEMETHOD

VALUE	MEANING
auto	The scaling is automatically determined.
init	The scaling is determined from the initial value. Use this if the sizes of the components of the initial value give a good estimate of the order of magnitude of the solution.
none	No scaling is applied.
manual	The scaling is user controlled by setting the property <code>scaleval</code> .
parent	Scaling method is inherited from the parent Variables feature (only for subattributes).

XmeshInfo

Get extended mesh information.

SYNTAX

```
SolverFeature step = model.sol(seqTag).feature(studyStepTag);
XmeshInfo xmi = step.xmeshInfo();
XmeshInfo xmi = step.xmeshInfo(meshCase);
SolverFeature var = model.sol(seqTag).feature(variablesTag);
XmeshInfo xmi = var.xmeshInfo();
XmeshInfo xmi = var.xmeshInfo(meshCase);
XmeshInfo xmi = model.sol(seqTag).xmeshInfo();
XmeshInfo xmi = model.sol(seqTag).xmeshInfo(meshCase);

String[] mcases = xmi.meshCases();

int nDofs = xmi.nDofs();
String[] fieldNames = xmi.fieldNames();
int[] fieldNDofs = xmi.fieldNDofs();
String[] geomTags = xmi.geoms();
String[] meshTypes = xmi.meshTypes();
String[] meshTypes = xmi.meshTypes(geomTag);

XmeshInfoDofs dofs = xmi.dofs();
int[] dofs.geomNums()
int[] dofs.nodes()
double[][] dofs.coords()
String[] dofs.dofNames()
int[] dofs.nameInds()
int[] dofs.solVectorInds()

XmeshInfoNodes nodes = xmi.nodes();
XmeshInfoNodes nodes = xmi.nodes(geomTag);
double[][] nodes.coords()
String[] nodes.dofNames()
int[][] nodes.dofs()

XmeshInfoElements elems = xmi.elements(meshType);
XmeshInfoElements elems = xmi.elements(meshType,geomTag);
double[][] elems.localCoords()
int[][] elems.nodes()
String[] elems.localDofNames()
double[][] elems.localDofCoords()
int[][] elems.dofs()

model.sol(seqTag).feature(studyStepTag).clearXmesh();
model.sol(seqTag).feature(variablesTag).clearXmesh();
```

DESCRIPTION

The Xmesh information methods provide information about the numbering of elements, nodes, and degrees of freedom (DOFs) in the extended mesh and in the matrices returned by `Assemble` and the solvers.

```
SolverFeature step = model.sol(seqTag).feature(studyStepTag);
XmeshInfo xmi = step.xmeshInfo();
```

returns information about all degrees of freedom in the given study step, for the main mesh case. This includes information about internal degrees of freedom.

```
XmeshInfo xmi = step.xmeshInfo(meshCase);
```

returns information about the given mesh case. The string `meshCase` can be `main`, `adaptionresidual`, or a multigrid level tag.

```
SolverFeature var = model.sol(seqTag).feature(variablesTag);
XmeshInfo xmi = var.xmeshInfo();
XmeshInfo xmi = var.xmeshInfo(meshCase);
```

returns information about the degrees of freedom solved for in the given variables feature. That is, the numbering of the degrees of freedom span over the DOFs solved for, which is the indexing used in the matrices and vectors assembled by the solver. Internal degrees of freedom are not included.

```
XmeshInfo xmi = model.sol(seqTag).xmeshInfo();
XmeshInfo xmi = model.sol(seqTag).xmeshInfo(meshCase);
```

is equivalent to calling `xmeshInfo` on the last study step feature in the sequence.

```
model.sol(seqTag).feature(studyStepTag).clearXmesh();
model.sol(seqTag).feature(variablesTag).clearXmes();
```

clears out the Xmesh object created by the call to `xmeshInfo`. After the required information has been obtained from the XmeshInfo object, this function should be called to release memory. When `xmeshInfo` is called on a solver sequence, an already existing Xmesh object is used, so there is no need to call `clearXmesh`.

General Information

`String[] mcases = xmi.meshCases()` returns a string vector containing tags of all mesh cases.

`int nDofs = xmi.nDofs()` returns the total number of DOFs.

`String[] fieldNames = xmi.fieldNames()` returns the field names, or the field names solved for.

`int[] fieldNDofs = xmi.fieldNDofs()` returns the number of DOFs for each field.

`String[] geomTags = xmi.geoms()` returns the tags of all geometries that exist in the xmesh.

`String[] meshTypes = xmi.meshTypes()` returns all mesh types.

`String[] meshTypes = xmi.meshTypes(geomTag)` returns all mesh types in geometry `geomTag` (a string). Possible mesh types are `vtx`, `edg`, `tri`, `quad`, `tet`, `hex`, `prism`, and `pyr`.

Information About Each DOF

`XmeshInfoDofs dofs = xmi.dofs()` returns information about each DOF.

The class `XmeshInfoDofs` has the following methods:

TABLE 6-68: XMESHINFODOFS METHODS

FIELD	CONTENTS
<code>int[] geomNums()</code>	1-based geometry numbers for all DOFs
<code>int[] nodes()</code>	0-based node numbers for all DOFs.
<code>double[][] coords()</code>	Global coordinates for all DOFs in the model length unit. The kth column of this matrix contains the coordinates of DOF number k.
<code>double[][] coords(xdGeomTag)</code>	For DOFs in an extra dimension product, return the coordinates of each DOF in the extra dimension geometry <code>xdGeomTag</code> . For DOFs that are not located in an extra dimension product containing <code>xdGeomTag</code> , the value NaN is returned.
<code>double[][] gCoords()</code>	Same as <code>coords()</code> except that coordinates are given in the geometry length unit. If there is more than one geometry, the coordinates of each DOF are given in the length unit of the geometry of that DOF.
<code>String[] dofNames()</code>	DOF names
<code>int[] nameInds()</code>	0-based indices into <code>dofNames()</code> for all DOFs.
<code>int[] solVectorInds()</code>	0-based indices into solution vector for all DOFs.

Information About Each Node Point

`XmeshInfoNodes nodes = xmesh.nodes()` returns information about nodes. This method throws an error if there is more than one geometry.

`XmeshInfoNodes nodes = xmesh.nodes(geomTag)` returns information about nodes in geometry `geomTag` (a string).

The class `XmeshInfoNodes` has the following methods:

TABLE 6-69: XMeshInfoNodes CLASS METHODS

FIELD	CONTENTS
<code>double[][] coords()</code>	Global coordinates for all nodes. The <i>n</i> th column of the matrix <code>coords</code> contains the coordinates of node point number <i>n</i>
<code>double[][] gCoords()</code>	Same as <code>coords()</code> except that coordinates are given in the geometry length unit.
<code>String[] dofNames()</code>	DOF names in this geometry
<code>int[][] dofs()</code>	0-based DOF numbers for all nodes in this geometry. <code>dofs()[k][n]</code> is the DOF number for DOF name <code>dofNames()[k]</code> at node point <i>n</i> . A value of -1 means that there is no DOF with this name at the node. Note: If there is a slit, only one of the DOFs is given for each node point.

Information about Each Mesh Element

`XmeshInfoElements[] elems = xmesh.elements(meshType)` returns information about mesh elements of type `meshType` (a string). This method throws an error if there is more than one geometry.

`XmeshInfoElements[] elems = xmesh.elements(meshType,geomTag)` returns information about mesh elements of type `meshType` in geometry `geomTag`.

The `XmeshInfoElements` class has the following methods:

TABLE 6-70: XMeshInfoElements CLASS METHODS

FIELD	CONTENTS
<code>double[][] localNodes()</code>	Local coordinates of nodes. The <i>k</i> th column of the matrix <code>lnodes()</code> contains the coordinates of local node point number <i>k</i> .
<code>int[][] nodes()</code>	0-based node point indices for all mesh elements of type <code>type()</code> . <code>nodes()[k][e1]</code> is the node point number within geometry <code>geomNum()</code> (see the output <code>xmi.nodes()</code>) for local node point <i>k</i> within mesh element <i>e1</i> . A value -1 means that there is no node point at this location.
<code>String[] localDofNames()</code>	The name for each local DOF.
<code>double[][] localDofCoords()</code>	The local coordinates for each local DOF (one column for each local DOF).
<code>int[][] dofs()</code>	0-based DOF numbers for all mesh elements of type <code>type()</code> . <code>dofs()[k][e1]</code> is the DOF number for local DOF <i>k</i> within mesh element <i>e1</i> . A value -1 means that there is no DOF at this location.

Results

Detailed COMSOL API reference information is included for the result features and utility methods.

In this chapter:

- [About Results Commands](#)
- [Use of Data Sets](#)
- [Extracting Data](#)
- [Solution Selection](#)

About Results Commands

The commands in this chapter provide numerical results and plots for visualizing results. Note that `result` for the model object corresponds to Results in the Model Tree.

The following list includes the available commands for results evaluation and visualization (listed in alphabetical order):

Results Commands (A to L)

- `Animation`
- `Annotation`
- `Array2D`, `Array3D`
- `ArrowVolume`, `ArrowSurface`, `ArrowLine`
- `AvVolume`, `AvSurface`, `AvLine`
- `Average`, `Integral`, `Maximum`, `Minimum`
- `Color`
- `Contour`
- `Contour (data set)`
- `CoordSysLine`, `CoordSysSurface`, `CoordSysVolume`
- `CutLine2D`, `CutLine3D`
- `CutPlane`
- `CutPoint1D`, `CutPoint2D`, `CutPoint3D`
- `Data`
- `Deform`
- `Edge2D`, `Edge3D`
- `Eval`
- `EvalGlobal`
- `EvalGlobalMatrix`
- `EvalPoint`
- `EvalPointMatrix`
- `FarField`
- `Filter`
- `Filter (Particle Tracing)`
- `Filter (Point Trajectories)`
- `Filter (Ray Tracing)`
- `Global (numerical)`
- `Global (plot)`
- `Grid1D`, `Grid2D`, `Grid3D`
- `Height`
- `Histogram`
- `HistogramHeight`
- `Image1D`, `Image2D`, `Image3D`
- `Interp`
- `InterferencePattern`
- `IntVolume`, `IntSurface`, `IntLine`
- `Isosurface`
- `Isosurface (data set)`
- `Join`
- `Line`
- `LineGraph`

Results Commands (M to Z)

- [MatrixHistogram](#)
- [MaxMinVolume](#), [MaxMinSurface](#), [MaxMinLine](#)
- [MaxVolume](#), [MaxSurface](#), [MaxLine](#),
[MinVolume](#), [MinSurface](#), [MinLine](#)
- [Mesh](#)
- [Mesh \(data set\)](#)
- [Mesh \(export\)](#)
- [Mirror2D](#), [Mirror3D](#)
- [Multislice](#)
- [Nyquist](#)
- [OctaveBand](#)
- [Parametric1D](#), [Parametric2D](#)
- [ParCurve2D](#), [ParCurve3D](#)
- [ParSurface](#)
- [Particle](#)
- [Particle \(data set\)](#)
- [Particle \(Evaluation\)](#)
- [ParticleMass](#)
- [ParticleTrajectories](#)
- [PhasePortrait](#)
- [Plot](#)
- [PlotGroup1D](#), [PlotGroup2D](#), [PlotGroup3D](#)
- [PoincareMap](#)
- [PointGraph](#)
- [PointTrajectories](#)
- [PolarGroup](#)
- [PrincipalSurface](#), [PrincipalVolume](#)
- [ReflectionGraph](#), [ImpedanceGraph](#),
[AdmittanceGraph](#)
- [Revolve1D](#), [Revolve2D](#)
- [ScatterVolume](#), [ScatterSurface](#)
- [Sector2D](#), [Sector3D](#)
- [Slice](#)
- [SmithGroup](#)
- [Solution](#)
- [Streamline](#)
- [Surface](#)
- [Surface \(data set\)](#)
- [SystemMatrix](#)
- [Table](#)
- [Table \(export\)](#)
- [Table \(plot\)](#)
- [TableHeight](#)
- [TableSurface](#)
- [TimeAverage](#), [TimeIntegral](#)
- [Volume](#)

Commands Grouped by Function

ATTRIBUTES

FUNCTION	PURPOSE
Color	Color expression
Deform	Deformation
Filter	Filter
Height	Height
HistogramHeight	Height for histogram plots
TableHeight	Height for table plots

DATA SETS

FUNCTION	PURPOSE
Array2D , Array3D	Array data sets
Average , Integral , Maximum , Minimum	Evaluation data sets

FUNCTION	PURPOSE
Contour (data set)	Contour data set
CutLine2D, CutLine3D	Cut line data sets
CutPlane	Cut plane data set
CutPoint1D, CutPoint2D, CutPoint3D	Cut point data sets
Edge2D, Edge3D	Edge data sets
Grid1D, Grid2D, Grid3D	Grid data sets
Isosurface (data set)	Isosurface data set
Join	Join data set
Mesh (data set)	Mesh data set
Mirror2D, Mirror3D	Mirror data set
Parametric1D, Parametric2D	Extends another data set by using a parameter as dimension
ParCurve2D, ParCurve3D	Parameterized curve data set
ParSurface	Parameterized face data set
Particle (data set)	Particle data set
Revolve1D, Revolve2D	Revolution data set
Sector2D, Sector3D	Sector symmetry data set
Solution	Solution data set
Surface (data set)	Surface data set
TimeAverage, TimeIntegral	Time average and time integral data sets

EXPORT FEATURES

FUNCTION	PURPOSE
Animation	Animation export or player
Data	Data export from data sets
Mesh (export)	Mesh export from data sets
Image1D, Image2D, Image3D	Image export
Plot	Plot export
Table (export)	Table export

NUMERICAL RESULTS

FUNCTION	PURPOSE
AvVolume, AvSurface, AvLine	Average
Eval	Generic evaluation
EvalGlobal	Global evaluation
EvalGlobalMatrix	Global matrix evaluation
EvalPoint	Evaluation at points
EvalPointMatrix	Matrix evaluation at points
Global (numerical)	Generic evaluation
Interp	Interpolation
IntVolume, IntSurface, IntLine	Integration
MaxVolume, MaxSurface, MaxLine, MinVolume, MinSurface, MinLine	Maximum and minimum
Particle (Evaluation)	Evaluation on particle trajectories
Ray (Evaluation)	Evaluation on ray trajectories
SystemMatrix	Evaluation of system matrix.

PLOTS

FUNCTION	PURPOSE
ArrowVolume, ArrowSurface, ArrowLine	Arrow plots
Contour	Contour plot
CoordSysLine, CoordSysSurface, CoordSysVolume	Coordinate system plot
FarField	Far-field plot
Global (plot)	Global plot
Histogram	Histogram plot
InterferencePattern	Interference pattern plot
Isosurface	Isosurface plot
Line	Line plot
LineGraph	Line graph plot
MatrixHistogram	Matrix histogram plot
MaxMinVolume, MaxMinSurface, MaxMinLine	Max/min marker plots
Mesh	Mesh plot
Multislice	Multislice plot
Nyquist	Nyquist plot
OctaveBand	Octave band plot
Particle	Massless particle tracing plot
Particle (1D Plot)	Particle plot
ParticleMass	Particle tracing plot with mass
ParticleTrajectories	Plot the data from a particle data set
PhasePortrait	Phase portrait plot
PoincareMap	Plot a Poincaré map
PointGraph	Point graph plot
PointTrajectories	Point trajectories plot

FUNCTION	PURPOSE
PrincipalSurface , PrincipalVolume	Principal stress/strain plots
Ray (1D Plot)	Ray plot
Ray Trajectories	Plot the data from a ray data set
ScatterVolume , ScatterSurface	Scatter plots
Slice	Slice plot
Streamline	Streamline plot
Surface	Surface plot
Table (plot)	Table plot
TableSurface	Table surface plot
Volume	Volume plot

TABLES

FUNCTION	PURPOSE
Table	Table containing real and imaginary data.

	Results Analysis and Plots in the <i>COMSOL Multiphysics Reference Manual</i>
-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------

Use of Data Sets

Table 7-1 shows applicability of data set output as data sets input. Rows correspond to data set output and columns to data set input. Check marks appear if the row data set output is accepted as input to the column data set. *evaluation* indicates Average, Integral, Maximum, and Minimum data sets

TABLE 7-1: DATA SET INPUT VERSUS OUTPUT

DATA SETS Input versus output	ARRAY2D	ARRAY3D	CONTOUR	CUTLINE2D	CUTLINE3D	CUTPLANE	CUTPOINT1D	CUTPOINT2D	CUTPOINT3D	EVALUATION	ISOSURFACE	JOIN	MESH	MIRROR2D	MIRROR3D	PARAMETRIC1D	PARAMETRIC2D	PARCURVE2D	PARCURVE3D	PARSURFACE	PARTICLE	RAY	REVOLVE1D	REVOLVE2D	SOLUTION
Array2D	√		√	√				√		√		√		√										√	
Array3D		√								√	√	√			√										
Contour										√															
CutLine2D	√						√			√				√									√		
CutLine3D		√					√			√					√										
CutPlane	√	√	√	√				√		√				√	√			√						√	
CutPoint1D										√															
CutPoint2D	√									√				√											
CutPoint3D		√								√					√										
Edge2D	√													√											√
Edge3D		√													√										√
<i>evaluation</i>										√															
Isosurface		√	√							√					√										
Join																									
Mesh	√	√	√	√	√	√	√	√	√	√	√			√	√			√	√	√			√	√	
Parametric1D																									√
Parametric2D																									√
ParCurve2D	√					√	√			√	√			√											
ParCurve3D		√					√			√					√										
ParSurface	√	√	√	√				√		√				√	√			√							
Particle						√				√															
Ray						√				√															
Revolve1D	√		√	√				√		√				√				√							
Revolve2D		√			√	√			√	√	√				√				√	√					
Sector2D	√			√		√						√		√									√		√
Sector3D		√			√	√					√	√			√									√	√
Solution	√	√	√	√	√	√	√	√	√	√	√	√		√	√	√	√	√	√	√	√	√	√	√	
Surface	√		√											√											

Table 7-2 shows applicability of data set output as input to plot features. Rows correspond to data set output and columns to plot features. Check marks appear if the data set output is accepted by the plot feature.

TABLE 7-2: DATA SET OUTPUT AS INPUT TO PLOT FEATURES

DATA SET OUTPUT As input to plots	ARROW	CONTOUR	FAR FIELD	GLOBAL	INTERFERENCE PATTERN	ISOSURFACE	LINE	LINE GRAPH	MESH	MULTISLICE	PARTICLE	PARTICLE TRACING	PARTICLE TRAJECTORIES	POINT GRAPH	POINT TRAJECTORIES	RAY	RAY TRAJECTORIES	SCATTER	SLICE	STREAMLINE	SURFACE	VOLUME
Array2D	✓	✓					✓											✓		✓	✓	
Array3D	✓					✓	✓			✓								✓	✓	✓	✓	✓
Contour	✓						✓															
CutLine2D	✓						✓	✓										✓				
CutLine3D	✓						✓	✓										✓				
CutPlane	✓	✓			✓		✓											✓		✓	✓	
CutPoint1D														✓	✓							
CutPoint2D														✓	✓							
CutPoint3D														✓	✓							
Edge2D							✓	✓														
Edge3D							✓	✓														
<i>evaluation</i>				✓										✓								
Isosurface	✓	✓					✓														✓	
Join	✓	✓				✓	✓	✓		✓				✓				✓	✓	✓	✓	✓
Mesh	✓	✓				✓	✓	✓	✓	✓				✓				✓	✓	✓	✓	✓
Mirror2D	✓	✓					✓											✓		✓	✓	
Mirror3D	✓					✓	✓			✓								✓	✓	✓	✓	✓
Parametric1D	✓	✓					✓											✓		✓	✓	
Parametric2D	✓	✓				✓	✓			✓								✓		✓	✓	✓
Particle				✓							✓		✓									
ParCurve2D	✓						✓	✓														
ParCurve3D	✓						✓	✓														
ParSurface	✓	✓					✓											✓		✓	✓	
Ray				✓												✓	✓					
Revolve1D	✓	✓					✓											✓		✓	✓	
Revolve2D	✓	✓				✓	✓			✓								✓	✓	✓	✓	✓
Sector2D	✓	✓					✓											✓		✓	✓	
Sector3D	✓	✓				✓	✓			✓								✓		✓	✓	✓
Solution	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓		✓	✓			✓	✓	✓	✓	✓
Surface	✓	✓					✓											✓	✓		✓	

Extracting Data

This section describes how to retrieve results or plot data directly through the COMSOL API.

- [Retrieving Plot Data](#)
- [Retrieving Numerical Results](#)

Retrieving Plot Data

Each plot is made up of one or more *groups* or parts. Most plots contain only one group, such as [Surface](#) or [Line](#) plots, whereas for example [Slice](#) plots contain one group per slice. Retrieve the number of groups with the method

```
plot.getGroups(<renderIndex>)
```

where `plot` is any plot feature.

Retrieving the groups requires that you specify a *render index* for which you want the number of groups. Each plot is made up of one or more internal, or rendering, plot types. For example, particle tracing can contain both lines and spheres, which are separate rendering types. Most plots have only a single rendering type, in which case you can safely use 0 as the render index. The *group index* property indicates groups or parts within the rendering group index.

Extract the data contained in any plot feature using the following methods:

`p = plot.getVertices(<renderIndex>, <groupIndex>)` returns a float matrix containing one row for each dimension and one column for each vertex.

`t = getElements(<renderIndex>, <groupIndex>)` returns the indices to columns in `p` of a simplex mesh, each column in `t` representing a simplex.

`d = getData(<renderIndex>, <groupIndex>, <dataType>)` returns a float array containing the data for each point in `p`. The type of data to retrieve is given by the String `<dataType>: "Color", "Height",` and so forth.

To retrieve the available data types, use:

```
types = getDataTypes(<renderIndex>)
```

which returns a string array with the data types currently present in the plot.

For plot features, you can retrieve the point normals using the following method:

```
n = getNormals(<renderIndex>, <groupIndex>)
```

which returns a floating-point matrix of point normals with each column corresponding to a single vertex.

EXAMPLE

```
int renderDataGroups = plot.getRenderGroups();

for (int ri = 0; ri < renderDataGroups; ri++) {

    String[] dataTypes = plot.getDataTypes(ri);
    int plotDataGroups = plot.getGroups(ri);
    for (int gi = 0; gi < plotDataGroups; gi++) {
        p = plot.getVertices(ri, gi);
        t = plot.getElements(ri, gi);
        //d = plot.getData(ri, gi, "Color");
        d = plot.getData(ri, gi, dataTypes[0]);
    }
}
```

Retrieving Numerical Results

It is possible to retrieve numerical results from any numerical feature, but there are a number of features especially designed for the COMSOL API that are not present in the COMSOL Desktop. These functions support multiple expressions, as well as some additional advanced properties, and a more convenient way to extract interpolated data. These two groups have slightly different access methods. Note that `numerical` in the model object corresponds to Derived Values in the Model Tree. In the API, code like `model.result().numerical(<ftag>)` returns objects of type `NumericalFeature`.

API-ONLY NUMERICAL FEATURES

TABLE 7-3: API-ONLY NUMERICAL FEATURE

FUNCTION	PURPOSE
Eval	Generic evaluation
Global (numerical)	Generic evaluation
Interp	Interpolation

`result = model.result().numerical(<ftag>).getData()` returns the real part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getData(<expressionIndex>)` returns the real part of the result for one expression, equivalent to `result[expressionIndex]`.

`result = model.result().numerical(<ftag>).getImagData()` returns the imaginary part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getImagData(<expressionIndex>)` returns the imaginary part of the result for one expression, equivalent to `result[expressionIndex]`.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex. For models with outer solutions, `isComplex()` returns true if the result for the current value of `outersolnum` is complex.

`model.result().numerical(<ftag>).isComplex(<outersolnum>)` returns true if the result is complex for the given outer solution.

`model.result().numerical(<ftag>).getNData()` returns the number of data vectors in the result.

`model.result().numerical(<ftag>).getCoordinates()` returns the coordinates of the evaluation or interpolation.

`model.result().numerical(<ftag>).getElements()` returns indices to columns in `p` of a simplex mesh.

`model.result().numerical(<ftag>).getVertexElements()` returns indices to mesh elements for each point.

More details can be found in the documentation for each feature type.

STANDARD NUMERICAL FEATURES

TABLE 7-4: STANDARD NUMERICAL FEATURES, ALSO PRESENT IN THE COMSOL MULTIPHYSICS GUI

FUNCTION	PURPOSE
EvalGlobal	Global evaluation
EvalPoint	Evaluation in points
IntVolume , IntSurface , IntLine	Integration

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to (`<columnwise>`) when `<columnwise>` is `false`. If `<columnwise>` is `true`, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If `<allocate>` is `true`, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `<allocate>` `true` and `<columnwise>` `false`.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex. For models with outer solutions, `isComplex()` returns true if the result is complex for any of the currently set `outersolnum`.

`model.result().numerical(<ftag>).isComplex(<outersolnum>)` returns true if the result is complex for the given outer solution.

You can find more details in the documentation for each feature type.

Solution Selection

In this section:

- [About Selecting Solutions](#)
- [Selecting Solutions by Solution Number](#)
- [Selecting Solutions by Solution Level](#)
- [Choosing Solution Selection Method](#)

About Selecting Solutions

During the solution of, for example, time-dependent models or parametric sweep models, more than one solution is generated, and in order to do relevant postprocessing the correct set of solutions needs to be selected among them. How many solutions the result features can display vary, the main difference being between single-select and multi-select features. Single solution selection is available when only one solution at a time can be visualized, for example in 2D and 3D plots. In numerical features and 1D plots multi-select is used to be able to display plots relating to more than one parameter value at a time.

There are two ways to select solutions—By solution number and by solution level.

COMSOL Multiphysics keeps them synchronized as far as possible.

Selecting solutions by solution number is a method that is tightly linked to the solution representation used by the solvers. The solution level method is structurally similar to the set up of the studies. Setting any one property belonging to either method triggers a synchronization in which all properties of that method are mapped to those of the other method. Mapping to the solution number method always works, since that case is the more general. If, on the other hand, mapping to the solution level method fails, its properties are set to their default settings.

The `solrepresentation` property is set to "solnum" automatically if the solution number method is the only one able to represent the selection made. On solving it is set to "solutioninfo" if the solution level method can be used to represent the previous selection. The `solutionrepresentation` property also decides which method is visible in the GUI.

Selecting Solutions by Solution Number

In the solution number method, the solutions are selected as a collection of inner and outer solutions. Inner solutions are generated by the solvers in one step. Outer solutions are generated by parametric sweeps wrapping the inner solutions. If there are more than one inner or outer parameter, they are represented and selected as tuples—that is, combinations of values of the different parameters. Depending on the setup of the studies the parameters can be either filled (all combinations) or sparse (a selection of combinations). In the single-selection case the properties used are `solnum`, `t` and `outersolnum`. In the multi-selection case the valid properties are `innerinput`, `solnum`, `solnumindices`, `t`, `outerinput`, `outersolnum`, and `outerindices`.

Selecting Solutions by Solution Level

In contrast, the solution level method handles all filled parameters—that is, parameters defined individually in the studies—in one level each, up to a maximum of three levels. If more levels than that exist in the study the outermost levels are combined in the third level. Parameters that together form sparse tuples in the studies are always presented on the same level. In the single-selection case the properties used are `looplevel` and `interp`. In the multi-selection case the valid properties are `looplevelinput`, `looplevel`, `looplevelindices`, and `interp`.

Choosing Solution Selection Method

Whether selection is made using the solution level method or the solution number method, the ways of selecting solutions are the same. In the single-selection case, solutions are selected by parameter index or, if the solution is time-dependent, by value. In the multiselection case there are a number of selection input options all, first, last, by index from existing values, by manual indices, or, in time-dependent cases, by time.

Which method to use is not self-evident. The solution level method is perhaps the more intuitive of the two because it attempts to treat each individual level separately. The solution number method, however, is more general, and can pinpoint which tuples to use with more precision. The recommendation is to use the solution level method wherever possible, and resort to the solution number method only if its special characteristics are needed.

Animation

Create animations of plots, saved to a file or shown in a player in the COMSOL Desktop main window.

SYNTAX

```
model.result().export().create(<ftag>, "Animation");  
model.result().export(<ftag>).set(property, <value>);  
model.result().export(<ftag>).run();
```

DESCRIPTION

`model.result().export().create(<ftag>, "Animation")` creates an animation feature with the name `<ftag>`.

`result().export(<ftag>).set("plotgroup", <ptag>)` changes the source of the animation to the plot group named `<ptag>`.

The following properties are available:

TABLE 7-5: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
allarray	integer array	All solutions	The solutions to use, chosen from all combinations of outer and inner. Applicable when both inner and outer solutions exist and solnumtype is all.
alwaysask	on off	off	Always ask for filename when saving. This property is ignored when running without a GUI.
antialias	on off	on	Enable/disable antialiasing.
avifilename	String		The name of the AVI file produced if movietype is avi.
aviqual	double in [0,1]	0.75	The quality of the AVI file if movietype is avi; higher is better.
axes	on off	on	If options is on; enable/disable display of the coordinate axes. Used for 1D and 2D animations.
axisorientation	on off	on	If options is on; enable/disable display of the axis orientation indicator. Used for 3D animations.
background	current color	color	The background color.
bmpfilename	String		The name of the output file if imagetype is bmp.

TABLE 7-5: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customcolor	double array	{1, 1, 1}	If background is color; the red, green, and blue components of the background color.
cycletype	fullharm halfharm linear	fullharm	The transformation of the solution that is performed if sweeptype is dde.
flashfilename	String		The name of the Flash file produced if movietype is flash.
flashinterp	on off	on	Enable/disable interpolation between frames in the generated Flash file if movietype is flash.
flashopen	on off	off	Enable/disable opening of the generated Flash file in a browser if movietype is flash.
fontsize	integer	9	The font size.
fps	positive integer	10	The number of frames per second.
framesel	all number	number	When sweeptype is solutions, framesel controls whether to look through all solutions or a specified number of frames.
frametime	positive double	0.1	The time in seconds that each frame is displayed when the target is set to player.
giffilename	String		The name of the GIF file produced if movietype is gif.
gifopen	on off	off	Enable/disable opening of the generated GIF file in a browser if movietype is gif.
grid	on off	on	If options is on; enable/disable display of the coordinate grid. Used for 3D animations.
height	integer	480	The height in pixels of the frames of the animation.
imagefilename	String		The base name of generated images if type is movieseq; the generated images have names of the form base1.ext, base2.ext, ...
interp	double array	Empty	The times to use, for transient levels, that the multilooplevel property points to. Available when the underlying data is transient, looplevelinput is interp.
legend	on off	on	If options is on; enable/disable display of the legend.
logo	on off	on	If options is on; enable/disable display of the logotype.
looplevel	integer array	All solutions	The solutions to use. Applicable to the level that the multilooplevel property points to, when looplevelinput is manual.
looplevelindices	integer array or String	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable to the level that the multilooplevel property points to, when looplevelinput is manualindices.

TABLE 7-5: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
looplevelinput	all manual manualindices interp	all	How to input the solution to use, for the level that multilooplevel points to. manual indicates that looplevel is used for that level. manualindices indicates that looplevelindices is used for that level. interp indicates that interp is used for that level.
maxframes	integer >= 2	25	The number of frames.
movietype	gif flash avi	gif	The movie format to use if type is movie.
multilooplevel	1<=integer<=3	1	The loop level to animate over. Only applied if solnumtype is pointing to the same level and solrepresentation is solutioninfo
options	on off	off	Enable/disable optional components of the image.
outerinnertype	first last all	last	When solnumtype is outer, outerinnertype controls whether to plot all, the first or the last inner solution. Applicable only for parametric sweep models.
outersolnumarray	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models when solnumtype is outer.
outersolnumsingle	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models when solnumtype is inner.
parameter	String		The model parameter that varies if sweeptype is parameter.
plotgroup	String		The name of the plot group to animate.
pstart	double	0	The start parameter if sweeptype is parameter.
pstop	double	1	The stop parameter if sweeptype is parameter.
repeat	Boolean	false	Whether to repeat from the beginning when the end is reached when playing.
resolution	integer	96	The frame resolution in dots per inch.
reverse	on off	off	Enable/disable reversal in time of the animation.
showframe	positive integer	1	The shown frame when playing,
singleinterp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels with the level pointed to by multilooplevel exempt. Available when the underlying data is transient.
singlelooplevel	array of non-negative integers and strings	First solution for each level.	The index of the solution to use, per level with the level pointed to by multilooplevel exempt, or interp, but only when the underlying data is transient.
solnum	integer array	all	The solutions to animate if sweeptype is solutions

TABLE 7-5: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solnumtype	level1 level2 level3 all inner outer	all	Whether to sweep over a selected loop level or over all, inner or outer solutions, in a parametric sweep model.
sweepstype	solutions parameter dde	solutions	The parameter that varies during animation.
target	file player	file	Save the animation in a file or show it in a player in a COMSOL graphics window.
timeinterp	on off	off	Enable/disable sweep over interpolated times if sweepstype is solutions.
tint	double array		The interpolated times to animate if timeinterp is on.
title	on off	on	If options is on; enable/disable display of the title.
type	imageseq movie	movie	The kind of output that is produced.
width	integer	640	The width in pixels of the frames of the animation.

EXAMPLE

Create a 2D animation of a surface and contour plot:

Code for use with Java

```
ExportFeature amim = model.result().export().create("c2", "pg1", "Animation");
anim.set("plotgroup", "pg1");
anim.run();
```

Code for use with MATLAB

```
anim = model.result.export.create('c2','pg1','Animation');
anim.set('plotgroup','pg1');
anim.export('c2').run;
```

SEE ALSO

[Image1D](#), [Image2D](#), [Image3D](#)

Annotation

Add an annotation to a 2D or 3D plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Annotation");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Annotation")` creates an annotation feature named `<ftag>` belonging to the plot group `<pgtag>`.

Annotations add text, including optional position coordinates, to 2D or 3D plots.

The following properties are available:

TABLE 7-6: VALID PROPERTY/VALUE PAIRS FOR ANNOTATIONS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
allowexprs	on off	off	Allow evaluation of expressions in the annotation's text.
color	custom black blue cyan green magenta red white yellow	red	The color to use for the point where the annotation appears.
data	none parent data set name	parent	The data set this feature refers to.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
exprprecision	positive integer	6	Precision for the expressions used in annotations.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that color is inherited from.
level	fromdataset global line point surface volume	fromdataset	The geometry level for evaluating expressions. The default is to take it from the data set. volume is only available in 3D.
posexpr	String array	{"0", "0"} (2D)	Position coordinates for the annotation (2 in 2D; 3 in 3D).
posprecision	positive integer	6	Precision for the position values.
prependcoords	on off	off	Include coordinates of annotation position before the annotation text.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
showpoint	on off	on	Whether to show the point for the position of the annotation or not.
text	String	Empty	The text to display as an annotation.
titletype	auto custom manual none	none	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

ATTRIBUTES

None.

SEE ALSO

[MaxMinVolume](#), [MaxMinSurface](#), [MaxMinLine](#)

Array2D, Array3D

Create 2D and 3D array data sets.

SYNTAX

```
model.result().dataset().create(<dtag>,"Array2D");
model.result().dataset().create(<dtag>,"Array3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Array2D")` creates a 2D array data set feature named `<dtag>`.

`model.result().dataset().create(<dtag>,"Array3D")` creates a 3D array data set feature named `<dtag>`.

This data set takes data from another data set and adds an array of copies in cells as a linear, rectangular (2D), or three-dimensional (3D) array of cells.

The following properties are available for Array 2D and Array 3D:

TABLE 7-7: VALID PROPERTY/VALUE PAIRS FOR ARRAY DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
base	corner center	corner	Base point for the position of the input cell if inputmethod is set to manual.
celllinearvar	String	Depends on the feature's tag	If hasvar is true: an integer variable that identify the current cell in the array.
cellvars	String array	Depends on the feature's tag	If hasvar is true: an integer array that identify the current cell in the array.
data	none data set name	First compatible data set	The data set this feature refers to.
displ	array (double or string)	{0, 0} (2D); {0, 0, 0} (3D)	The displacement used to move each cell in the array if displmethod is set to manual.
displmethod	auto manual	auto	Use automatic or manual definition of the displacement for the cells in the array.
floquetper	on off	off	Include Floquet periodicity.
fullsize	array (double or string)	{1, 1} (2D); {1, 1, 1} (3D)	Size of the array (number of cells in each direction) for a rectangular or three-dimensional array.
hasvars	Boolean	false	If true, space and cell indicator variables are defined.
inputmethod	auto manual	auto	Use automatic or manual definition of the part of space that the array is created from,
linearsize	positive integer	1	Size of the linear array in number of cells.
spacevars	String array	Depends on the feature's tag	If hasvar is true: The name of space variables, which evaluate to the coordinates in the data set's coordinate system.

TABLE 7-7: VALID PROPERTY/VALUE PAIRS FOR ARRAY DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
type	linear rectangular three-dimensional	rectangular (2D); three-dimensional (3D)	The type of array: Linear, rectangular (2D only), or three-dimensional (3D only).
wavevector	String array		The wave vector for Floquet periodicity as a 2-element or 3-element string array.

ArrowVolume, ArrowSurface, ArrowLine

Create volume, surface, or line arrow plots.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"ArrowVolume");
model.result(<pgtag>).create(<ftag>,"ArrowSurface");
model.result(<pgtag>).create(<ftag>,"ArrowLine");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Arrow...")` creates an arrow plot feature named `<ftag>`.

Arrow plots are available in 2D and 3D. `ArrowLine` plots arrows on lines in 2D or 3D such as geometry boundaries in 2D or cut plane edges in 3D. `ArrowSurface` plots arrows on surfaces in 2D or 3D. `ArrowVolume` plots arrows in a 3D volume.

The following properties are available:

TABLE 7-8: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the arrow position.
arrowcount	positive integer	200	If placement is uniform: The approximate number of arrows. Available for <code>ArrowEdge</code> in 2D and 3D and for <code>ArrowSurface</code> in 2D and 3D.
arrowlength	normalized proportional logarithmic	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowtype	arrow cone	arrow	The type of arrow to draw.
arrowxmethod	number coord	number	Indicates whether the x-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for <code>ArrowSurface</code> in 2D and <code>ArrowVolume</code> in 3D.
arrowymethod	number coord	number	Indicates whether the y-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for <code>ArrowSurface</code> in 2D and <code>ArrowVolume</code> in 3D.
arrowzmethod	number coord	number	Indicates whether the z-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for <code>ArrowVolume</code> in 3D.

TABLE 7-8: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The color to use.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent	The description of the expressions in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array of length 2 in 2D and 3 in 3D	Mode-dependent	The components to plot.
inheritarrowsscale	Boolean	true	If inheritplot is not none: Determines if arrow scale is inherited.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that arrow scale, color, and deformation scale is inherited from.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.

TABLE 7-8: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
placement	elements uniform uniformanis	uniform	If uniform, then the arrows are distributed uniformly. If uniformanisotropic, then the arrows are distributed uniformly with weights taken from the weight property. If elements, then arrows are drawn in the mesh nodes for each element. Available for ArrowEdge in 2D and 3D and for ArrowSurface in 3D.
prefixintitle	String	Empty	Added prefix to contribution to title.
scale	positive double	1	If scaleactive is true: The length scale factor.
scaleactive	Boolean	false	Whether to use manual scaling.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to contribution to title
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
weight	double array	1	If placement is uniformani: The weights given to the different axis directions.
xnumber	non-negative integer	15 (3D), 7 (2D)	Number of points in the x-direction. Active when arrowxmethod is set to number. Available for ArrowSurface in 2D and ArrowVolume in 3D.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when arrowxmethod is set to coord. Available for ArrowSurface in 2D and ArrowVolume in 3D.

TABLE 7-8: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ynumber	integer	15 (3D), 7 (2D)	Number of points in the y-direction. Active when arrowmethod is set to number. Available for ArrowSurface in 2D and ArrowVolume in 3D.
ycoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowmethod is set to coord. Available for ArrowSurface in 2D and ArrowVolume in 3D.
znumber	integer	15 (3D), 7 (2D)	Number of points in the z-direction. Active when arrowzmethod is set to number. Available for ArrowVolume in 3D.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowzmethod is set to coord. Available for ArrowVolume in 3D.

ATTRIBUTES

[Color](#), [Deform](#), [Filter](#)

SEE ALSO

[Line](#), [ScatterVolume](#), [ScatterSurface](#), [Surface](#), [Volume](#)

AvVolume, *AvSurface*, *AvLine*

Compute a volume average, surface average, or line average.

SYNTAX

```

model.result().numerical().create(<ftag>,"AvVolume");
model.result().numerical().create(<ftag>,"AvSurface");
model.result().numerical().create(<ftag>,"AvLine");
model.result().numerical(<ftag>).selection(...);
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();

```

DESCRIPTION

model.result().numerical().create(<ftag>,"AvVolume") creates a volume average feature with the name <ftag>.

model.result().numerical().create(<ftag>,"AvSurface") creates a surface average feature with the name <ftag>.

model.result().numerical().create(<ftag>,"AvLine") creates a line average feature with the name <ftag>.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to `(columnwise)` when `columnwise` is false. If `columnwise` is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(allocate, columnwise)` returns the imaginary part of complex result, recomputing the feature if necessary. If `allocate` is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `allocate` true and `columnwise` false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).selection()` returns the selection of the geometry for the computation of the volume average, surface average, or line average. See [Selections](#) for more information about the available selection methods.

`model.result().numerical(<ftag>).set(property, <value>)` sets the value of a property of the volume average, surface average, or line average.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-9: VALID PROPERTY/VALUE PAIRS FOR AVERAGES

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
dataseries	average integral maximum minimum none rms stddev variance	none	The operation that is applied to the data series formed by the evaluation.
dataseriesmethod	auto integration summation	auto	The method to use for the data series: automatic, integration, or summation.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot

TABLE 7-9: VALID PROPERTY/VALUE PAIRS FOR AVERAGES

NAME	VALUE	DEFAULT	DESCRIPTION
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
intorder	positive integer	4	The integration order
intorderactive	on off	off	Whether to use manually specified integration order
intsurface	on off	off	Compute surface integral. Available for line integration features of axisymmetric models.
intvolume	on off	off	Compute volume integral. Available for surface integration features of axisymmetric models.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range (1, 1, 20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
maximumobj	abs real	real	The value being maximized if dataseries is maximum.
method	auto integration summation	auto	The integration method
minimumobj	abs real	real	The value being minimized if dataseries is minimum
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.

TABLE 7-9: VALID PROPERTY/VALUE PAIRS FOR AVERAGES

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	non-negative integer array	All solutions	The solutions to use
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models. The level values (level1, level2, and so on) are the levels in the parametric sweep.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[IntVolume](#), [IntSurface](#), [IntLine](#)

Average, Integral, Maximum, Minimum

Create average, integral, maximum, and minimum evaluation data sets.

SYNTAX

```
model.result().dataset().create(<dtag>,"Average");
model.result().dataset().create(<dtag>,"Integral");
model.result().dataset().create(<dtag>,"Maximum");
model.result().dataset().create(<dtag>,"Minimum");
model.result().dataset(<dtag>).set(property, <value>);
model.result().dataset(<dtag>).selection(...);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"Average") creates a data set named <dtag> that computes the average of another data set.

`model.result().dataset().create(<dtag>,"Integral")` creates a data set named <dtag> that computes the integral of another data set.

`model.result().dataset().create(<dtag>,"Maximum")` creates a data set named <dtag> that computes the maximum of another data set.

`model.result().dataset().create(<dtag>,"Minimum")` creates a data set named <dtag> that computes the minimum of another data set.

`model.result().dataset(<dtag>).selection()` returns the selection of the geometry for the data set, which by default is the selected geometry in the data set that this data set refers to. See [Selections](#) for more information about the available selection methods.

`model.result().dataset(<dtag>).set(property,<value>)` sets the value of a property of the data set.

The following properties are available:

TABLE 7-10: COMMON PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
name	String	None	The name of this feature
data	none data set name	First compatible data set	The data set this feature refers to.
level	fromdataset volume surface edge point	fromdataset	The element dimension; if fromdataset, then the highest dimension supported by the data set is used.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.

The following properties are available:

TABLE 7-11: ADDITIONAL PROPERTY/VALUE PAIRS FOR AVERAGE AND INTEGRAL

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intmethod	auto integration summation	auto	For Average and Integral: The integration method.
intorderactive	on off	off	For Average and Integral: Whether to use manually specified integration order.
intorder	Positive integer	4	For Average and Integral: The manually set integration order.
intsurface	Boolean	false	Compute surface integral for 1D axisymmetric data

TABLE 7-11: ADDITIONAL PROPERTY/VALUE PAIRS FOR AVERAGE AND INTEGRAL

PROPERTY	VALUE	DEFAULT	DESCRIPTION
intvolume	Boolean	false	Compute volume integral for 2D axisymmetric data
refine	non-negative integer	1	For Maximum and Minimum: The element refinement to use. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.

Color

Add a color expression attribute to add a coloring to the shapes defined by a plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"Color");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"Color")` creates a color expression attribute named `<atag>` belonging to the plot feature `<ftag>`.

Use color expressions to add a coloring to the shapes defined by a plot. Color expressions can be added to surface plots, line plots, volume plots, arrow plots, contour plots, isosurface plots, particle tracing plots, and streamline plots.

The following properties are available:

TABLE 7-12: VALID PROPERTY/VALUE PAIRS FOR COLOR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color the parent plot.
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.

TABLE 7-12: VALID PROPERTY/VALUE PAIRS FOR COLOR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
prefixintitle	String	Empty	Added prefix to contribution to title.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range are not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
suffixintitle	String	Empty	Added suffix to contribution to title.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

COLOR TABLES

The following color tables are available for use with the `colortable` property here and in other types of plots: Cyclic, Disco, DiscoLight, GrayPrint, GrayScale, Rainbow, RainbowLight, Spectrum, Thermal, ThermalEquidistant, ThermalLight, Traffic, TrafficLight, Wave, WaveLight. See the *COMSOL Multiphysics Reference Guide* for more information about the color tables.

SEE ALSO

[Deform](#), [Height](#), [ArrowVolume](#), [ArrowSurface](#), [ArrowLine](#), [Contour](#), [Isosurface](#), [Particle](#), [ParticleMass](#), [Streamline](#)

Contour

Create a contour plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Contour");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Contour")` creates a contour plot feature named `<ftag>` belonging to the plot group `<pgtag>`. Contour plots are available in 2D and 3D plot groups. Contour lines are colored by their level. Add a color expression attribute to color them by an arbitrary expression.

The following properties are available:

TABLE 7-13: VALID PROPERTY/VALUE PAIRS FOR CONTOUR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color the contours.
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
contourlabels	on off	off	Whether to display labels next to the contour lines. Only applicable if contourtype is lines.
contourtype	lines filled tubes	lines	Whether to display lines for each level or a surface with bands of color.

TABLE 7-13: VALID PROPERTY/VALUE PAIRS FOR CONTOUR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
customlabelcolor	RGB-triplet	{1,0,0} or last used color.	If contourlabels is on and labelcolor is custom: The label color to use.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color and deformation scale is inherited from.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
levelmethod	number levels	number	How to specify contour levels.
labelcolor	custom black blue cyan green magenta red white yellow	red	If contourlabels is on: The label color to use.
labelprec	positive	integer	If contourlabels is on: The number of significant digits in the labels.
levels	double array		The specific levels to plot. Active when levelmethod equals levels.

TABLE 7-13: VALID PROPERTY/VALUE PAIRS FOR CONTOUR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
number	positive integer	20	Total number of contour levels. Active when levelmethod equals number.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title.
radiusexpr	String	1	Expression for the tube radius, when the contour type is set to tubes.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range are not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
smooth	none internal everywhere	internal	Smoothing settings.

TABLE 7-13: VALID PROPERTY/VALUE PAIRS FOR CONTOUR

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solnum	Non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
tuberadiusscale	double	Computed automatically	The scale factor for the tube radius when the contour type is set to tubes.
tuberadiusscaleactive	Boolean	false	Whether the tube radius scaling is active or not. By default, the tube radius scaling is automatic.
tubescale	Boolean	true	Whether the style inheritance of tube radius scale factor is active or not.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
wireframe	on off	off	Whether to plot filled elements or only their edges.

ATTRIBUTES[Color](#), [Deform](#), [Filter](#)**SEE ALSO**[Contour \(data set\)](#), [Surface](#)

Contour (data set)

Create a contour data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Contour");  
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Contour")` creates a contour data set feature named `<dtag>`.

Contour lines are in general not parametrizable, and this limits the set of plots and transformations that can be applied to them.

The following properties are available:

TABLE 7-14: VALID PROPERTY/VALUE PAIRS FOR CONTOUR DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
descr	String		Description of expr
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	u	Expression to find constant level for
level	double	0	Level that defines the data set. Active when levelmethod is level.
levelmethod	number levels	number	How to specify contour levels
number	positive integer	20	Total number of contour levels. Active when levelmethod is number.
unit	String		Unit of expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Contour](#)

CoordSysLine, CoordSysSurface, CoordSysVolume

Create a coordinate system line, surface, or volume plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"CoordSysVolume");  
model.result(<pgtag>).create(<ftag>,"CoordSysSurface");  
model.result(<pgtag>).create(<ftag>,"CoordSysLine");  
  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"CoordSysVolume")` creates a coordinate system plot named `<ftag>` belonging to the plot group `<pgtag>`.

The following properties are available:

TABLE 7-15: VALID PROPERTY/VALUE PAIRS FOR COORDINATE SYSTEM PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowcount	positive integer	200	If placement is uniform: The approximate number of points to display coordinate systems in. Available for CoordSysLine in 2D and 3D and for CoordSysSurface 3D.
arrowlength	normalized proportional logarithmic	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowtype	arrow cone	arrow	The type of arrow to draw
arrowxmethod	number coord	number	Indicates whether the x-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
arrowymethod	number coord	number	Indicates whether the y-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
arrowzmethod	number coord	number	Indicates whether the z-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for CoordSysVolume in 3D.
data	none parent data set name	parent	The data set this feature refers to.
descr	String		If mode is matrix: The description of the matrix expression to plot. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String		If mode is matrix: The matrix expression to plot. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.

TABLE 7-15: VALID PROPERTY/VALUE PAIRS FOR COORDINATE SYSTEM PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
inheritarrowsscale	Boolean	true	If inheritplot is not none: Determines if arrow scale is inherited.
inherit color	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that arrow scale, color, and deformation scale is inherited from.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
mode	system matrix	system	Chooses whether what's plotted is taken from a coordinate system or a matrix variable. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
placement	elements uniform uniformani	uniform	If uniform, then the arrows are distributed uniformly. If uniformanisotropic, then the arrows are distributed uniformly with weights taken from the weight property. If elements, then arrows are drawn in each element. Available for CoordSysLine in 2D and 3D and for CoordSysSurface 3D.
scale	positive double	1	If scaleactive is true: The length scale factor.
scaleactive	Boolean	false	Whether to use manual scaling.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
sys	none coordinate system name	none	The coordinate system to plot.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.

TABLE 7-15: VALID PROPERTY/VALUE PAIRS FOR COORDINATE SYSTEM PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title should be used (the title property). none if no title should be displayed.
type	stress strain	stress	Selection between principal stress and principal strain plot.
weight	double array	1	If placement is uniform: The weights given to the different axis directions.
xnumber	non-negative integer	15 (3D), 7 (2D)	Number of points in the x-direction. Active when arrowxmethod is set to number. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when arrowxmethod is set to coord. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
ynumber	integer	15 (3D), 7 (2D)	Number of points in the y-direction. Active when arrowymethod is set to number. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
ycoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowymethod is set to coord. Available for CoordSysSurface in 2D and CoordSysVolume in 3D.
znumber	integer	15 (3D), 7 (2D)	Number of points in the z-direction. Active when arrowzmethod is set to number. Available for CoordSysVolume in 3D.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowzmethod is set to coord. Available for CoordSysVolume in 3D.

CutLine2D, CutLine3D

Create a 2D or 3D cut line data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"CutLine2D");
model.result().dataset().create(<dtag>,"CutLine3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"CutLine2D") creates a 2D cut line data set feature named <dtag>.

model.result().dataset().create(<dtag>,"CutLine3D") creates a 3D cut line data set feature named <dtag>.

The following properties are available:

TABLE 7-16: VALID PROPERTY/VALUE PAIRS FOR CUTLINE DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
bndsnap	Boolean	false	If true, each point is snapped to the closest boundary.
bounded	on off	on	Active when method equals twopoint, this property specifies if the line should be bounded by the two points or extend infinitely.
data	none data set name	First compatible data set	The data set this feature refers to.
genparaactive	on off	off	Decides if parallel lines should be drawn
genparadist	double array	{}	Active when genparaactive is on, this property contains the distances from the extra parallel lines to the base line.
genpoints	double matrix	{{0, 0, 0}, {1,0,0}}	Active when method equals twopoint, this property contains the coordinates of the two points in the two rows of the matrix.
method	twopoint pointdir	twopoint	Decides if the line should be specified by two points or through one point and a direction.
orthvec	double array of length 3	{0, 1, 0}	Used in 3D when genparaactive is true: The direction that together with the cut line's direction spans the plane that contains the parallel lines.
pddir	double array	Zero vector	Active when method equals pointdir, this property contains the direction.
pdpoint	double array	Zero vector	Active when method equals pointdir, this property contains the coordinates of the point.
spacevars	String array of length 1	Created from the feature tag.	The name of the variable that evaluates to the line's parameterization

SEE ALSO

[LineGraph](#)

CutPlane

Create a cut plane data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"CutPlane");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"CutPlane") creates a cut plane data set with the name <dtag>.

The following properties are available:

TABLE 7-17: VALID PROPERTY/VALUE PAIRS FOR CUT PLANE DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
genparaactive	on off	off	Decides if parallel lines should be drawn.

TABLE 7-17: VALID PROPERTY/VALUE PAIRS FOR CUT PLANE DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
genparadist	double array	{}	Active when genparaactive is on, this property contains the distances from the extra parallel planes to the base plane.
genmethod	threepoint pointnormal	threepoint	Active when planetype equals general, this property indicates whether the plane should be specified by three points or through one point and a normal.
genpoints	double matrix	{{0, 0, 0}, {1,0,0}, {0,1,0}}	Active when method equals threepoint, this property contains the coordinates of the three points in the rows of the matrix.
genpnpoint	double array of length three	Zero vector	Active when genmethod equals pointnormal, this property contains the coordinates of the point.
hasnormal	on off	off	Decides if variables for the plane's normal component are defined.
genpnvec	double array of length three	Zero vector	Active when genmethod equals pointnormal, this property contains the normal vector.
normal	String array of length three	Created from the feature tag.	If hasnormal is on: The names of the variables that can be used to evaluate the plane's normal.
planetype	quick general	quick	Specify plane type
quickplane	xy yz zx yx zy xz	yz	Specify quick plane type. Active when planetype is quick.
quickx	double	0	x-coordinate if planetype is yz or zy.
quicky	double	0	y-coordinate if planetype is zx or xz.
quickz	double	0	z-coordinate if planetype is xy or yx.
spacevars	String array of length 2	Created from the feature tag.	The names of the variables that evaluate to the plane's parameterization.

EXAMPLES

This example creates a cut plane and plots a surface plot and a contour plot on it.

Create a solution data set and set it to point to the named solution Sol1 from a solver sequence:

Code for use with Java

```
DatasetFeature ds = model.result().dataset().create("dset1", "Solution");
ds.set("solution", "sol1");
```

Code for use with MATLAB

```
ds = model.result.dataset.create('dset1', 'Solution');
ds.set('solution', 'sol1');
```

Create 3D plot group:

Code for use with Java

```
ResultFeature pg = result().create("pg1",3);
DatasetFeature ds = result().dataset().create("cutp1","CutPlane");
ds.set("data", "dset1");

pg.create("surf1","Surface");
pg.feature("surf1").set("data","cutp1");

pg.create("cont1","Contour");
pg.feature("cont1").set("data","cutp1");
```


Code for use with MATLAB

```
pg = result.create('pg1',3);
ds = result.dataset.create('cutp1','CutPlane');
ds.set('data','dset1');

pg.create('surf1','Surface');
pg.feature('surf1').set('data','cutp1');

pg.create('cont1','Contour');
pg.feature('cont1').set('data','cutp1');
```

SEE ALSO

[Surface](#)

CutPoint1D, CutPoint2D, CutPoint3D

Create a 1D, 2D, or 3D cut point data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"CutPoint1D");
model.result().dataset().create(<dtag>,"CutPoint2D");
model.result().dataset().create(<dtag>,"CutPoint3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"CutPoint1D")` creates a 1D cut point data set with the name `<dtag>`.

`model.result().dataset().create(<dtag>,"CutPoint2D")` creates a 2D cut point data set with the name `<dtag>`.

`model.result().dataset().create(<dtag>,"CutPoint3D")` creates a 3D cut point data set with the name `<dtag>`.

A cut point is the 0D analog of cut lines and cut planes. A difference compared to cut lines and cut planes is that a cut point feature can contain an arbitrary number of points. Cut points can exist in 1D, 2D, and 3D.

The following properties are available:

TABLE 7-18: VALID PROPERTY/VALUE PAIRS FOR CUT POINT DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
bndsnap	Boolean	false	If true, each point is snapped to the closest boundary.
data	none data set name	First compatible data set	The data set this feature refers to.
gridx	double array		If method is grid: The grid x-coordinates
filename	String		If method is file: The file that contains the point coordinates in a spreadsheet format.
gridy	double array		If method is grid: The grid y-coordinates. (Only for CutPoint2D and CutPoint3D)
gridz	double array		If method is grid: The grid z-coordinates. (Only for CutPoint3D)
method	coords file grid regulargrid	coords	The method used for defining the points
pointx	double array	[]	If method is coords: The point x-coordinates.

TABLE 7-18: VALID PROPERTY/VALUE PAIRS FOR CUT POINT DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
pointy	double array	[]	If method is coords: The point y-coordinates. (Only for CutPoint2D and CutPoint3D)
pointz	double array	[]	If method is coords: The point z-coordinates. (Only for CutPoint3D)
regulargridx	integer	10	If method is regulargrid: The number of grid points in the x-direction.
regulargridy	integer	10	If method is regulargrid: The number of grid points in the y-direction. (Only for CutPoint2D and CutPoint3D)
regulargridz	integer	10	If method is regulargrid: The number of grid points in the z-direction. (Only for CutPoint3D)

SEE ALSO[PointGraph](#)*Data*

Export data to a file.

SYNTAX

```
model.result().export().create(<ftag>, "Data");
model.result().export().create(<ftag>, <dtag>, "Data");
model.result().export(<ftag>).set(property, <value>);
model.result().export(<ftag>).run();
```

DESCRIPTION

`model.result().export().create(<ftag>, "Data")` creates a data export feature with the name `<ftag>`.

`model.result().export().create(<ftag>, <dtag>, "Data")` creates a data export feature with the name `<ftag>` for the data set `<dtag>`.

The following properties are available:

TABLE 7-19: VALID PROPERTY/VALUE PAIRS FOR DATA EXPORT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
alwaysask	on off	off	Always ask for filename when saving. This property is ignored when running without a GUI.
coordfilename	String		If location is file: The file that contains coordinates to evaluate in.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	String		The name of the data set to export.
descr	String array		Descriptions of the expressions in expr.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.

TABLE 7-19: VALID PROPERTY/VALUE PAIRS FOR DATA EXPORT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
expr	String array		The expressions to evaluate and export.
filename	String		The output file.
fullprec	on off	on	If on, floating-point numbers are written in full precision, otherwise they are written with six significant digits.
gporder	positive integer	1	If pattern is gauss: The Gauss point order.
gridstruct	grid spreadsheet	spreadsheet	If location is grid or regulargrid: The format of the exported data.
gridx1	double array		If location is grid and the data set is 1D: x-coordinates to evaluate in.
gridx2	double array		If location is grid and the data set is 2D: x-coordinates to evaluate in.
gridy2	double array		If location is grid and the data set is 2D: y-coordinates to evaluate in.
gridx3	double array		If location is grid and the data set is 3D: x-coordinates to evaluate in.
gridy3	double array		If location is grid and the data set is 3D: y-coordinates to evaluate in.
gridz3	double array		If location is grid and the data set is 3D: z-coordinates to evaluate in.
header	on off	off	Enable/disable a data header in the output file.
ifexists	append overwrite	overwrite	If the file exists, append to or overwrite the file contents.
lagorder	integer	1	The Lagrange point order to use if resolution is custom.
level	fromdataset volume surface line point	fromdataset	The geometry level to evaluate on.
location	fromdataset file grid regulargrid	fromdataset	The points evaluated in.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
pattern	gauss lagrange	lagrange	If the data set is a solution: Specifies if evaluation takes place in Lagrange points or in Gauss points.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
regulargridx1	positive integer	10	If location is regulargrid and the data set is 1D: The number of grid points along the x-axis.
regulargridx2	positive integer	10	If location is regulargrid and the data set is 2D: The number of grid points along the x-axis.
regulargridy2	positive integer	10	If location is regulargrid and the data set is 2D: The number of grid points along the y-axis.
regulargridx3	positive integer	10	If location is regulargrid and the data set is 3D: The number of grid points along the x-axis.
regulargridy3	positive integer	10	If location is regulargrid and the data set is 3D: The number of grid points along the y-axis.

TABLE 7-19: VALID PROPERTY/VALUE PAIRS FOR DATA EXPORT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
regulargridz3	positive integer	10	If location is regulargrid and the data set is 3D: The number of grid points along the z-axis.
resolution	normal fine finer custom		The evaluation resolution.
sdim	fromdataset 0 1 2 3	fromdataset	The space dimension to evaluate the underlying data set in.
solnum	integer array		Solution number to take values from.
sort	on off	off	Enable/disable sorting of the points with respect to the coordinates.
struct	sectionwise spreadsheet		If location is fromdataset: Format of the exported data.
t	String		Time to evaluate in if timeinterp is on.
timeinterp	on off	off	Enable/disable explicit time to evaluate in.

SEE ALSO

[HistogramHeight](#)

Deform

Add a deformation attribute to a plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>, "Deform");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>, "Deform")` creates a deformation attribute feature with the name `<atag>`, belonging to the feature `<ftag>`.

Deformation attributes deform the coordinates of a plot feature according to an expression (typically the displacement in a structural mechanics model). The deformation attribute can be added to arrow plots, contour plots, isosurface plots, particle plots, slice plots, streamline plots, surface plots, volume plots, and mesh plots.

The following properties are available:

TABLE 7-20: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot

TABLE 7-20: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
prefixintitle	String	Empty	Added prefix to contribution to title
scale	positive double	Model-dependent	The scale factor for the deformation
scaleactive	on off	off	Whether to use the automatically computed scale, or the scale in the scale property.
suffixintitle	String	Empty	Added suffix to contribution to title
title	String	The auto-title	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Color](#), [Height](#), [ArrowVolume](#), [ArrowSurface](#), [ArrowLine](#), [Contour](#), [Contour](#), [Isosurface](#), [Line](#), [Mesh](#), [Particle](#), [ParticleMass](#), [ScatterVolume](#), [ScatterSurface](#), [Slice](#), [Streamline](#), [Surface](#), [Volume](#).

Edge2D, Edge3D

Create a 2D or 3D edge data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Edge2D");
model.result().dataset().create(<dtag>,"Edge3D");

model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Edge2D")` creates a 2D edge data set with the name `<dtag>`.

The edge data set makes it possible to evaluate edge of a model in 1D or the model's dimension.

The following properties are available:

TABLE 7-21: VALID PROPERTY/VALUE PAIRS FOR EDGE DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The solution data set this feature refers to.

Eval

Evaluate expressions in domains.

SYNTAX

```
model.result().numerical().create(<ftag>,"Eval");
model.result().numerical(<ftag>).selection(...);
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getData();
model.result().numerical(<ftag>).getData(<expressionIndex>);
model.result().numerical(<ftag>).getImagData();
model.result().numerical(<ftag>).getImagData(<expressionIndex>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).getNData();
model.result().numerical(<ftag>).getCoordinates();
model.result().numerical(<ftag>).getElements();
model.result().numerical(<ftag>).getVertexElements();
model.result().numerical(<ftag>).run();
```

DESCRIPTION

`model.result().numerical().create(<ftag>,"Eval")` creates an evaluation feature with the name `<ftag>`.

Eval is a feature made specifically for users of the COMSOL API and does not appear in the COMSOL Multiphysics GUI. It is used to evaluate results directly on the solution.

`result = model.result().numerical(<ftag>).getData()` returns the real part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getData(<expressionIndex>)` returns the real part of the result for one expression, equivalent to `result[expressionIndex]`.

`result = model.result().numerical(<ftag>).getImagData()` returns the imaginary part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getImagData(<expressionIndex>)` returns the imaginary result for one expression, equivalent to `result[expressionIndex]`.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).getNData()` returns the number of data vectors in the result.

`model.result().numerical(<ftag>).getCoordinates()` returns node point coordinates.

`model.result().numerical(<ftag>).getElements()` returns indices to columns in `p` of a simplex mesh.

`model.result().numerical(<ftag>).getVertexElements()` returns indices to mesh elements for each point.

The following properties are available:

TABLE 7-22: VALID PROPERTY/VALUE PAIRS FOR EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	none data set name	First compatible data set	The data set this feature refers to. Only solution and particle data sets are supported.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic

TABLE 7-22: VALID PROPERTY/VALUE PAIRS FOR EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent	The expressions to evaluate
complexfun	on off	on	Use complex-valued functions with real input
gporder	poitive integer	1	If pattern is gauss: The Gauss point order.
matherr	on off	off	Error for undefined operation or variable
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
refine	auto integer	auto	Refinement of elements for evaluation points.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
pattern	gauss lagrange	lagrange	Specifies if evaluation takes place in Lagrange points or in Gauss points.
phase	double	0	Evaluate solution at this angle, given in degrees.
smooth	none internal everywhere	internal	Smoothing settings
solnum	non-negative integer array	All solutions	The solutions to use
t	double array	Empty	The times to use. Available when the underlying solution is transient.
phase	double	0	Evaluate solution at this angle, given in degrees.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Interp](#), [Global \(numerical\)](#), [EvalPoint](#), [EvalGlobal](#)

EvalGlobal

Evaluate global quantities.

SYNTAX

```
model.result().numerical().create(<ftag>,"EvalGlobal");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();
```

DESCRIPTION

`model.result().numerical().create(<ftag>,"EvalGlobal")` creates an evaluation feature with the name `<ftag>`.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to (`<columnwise>`) when `columnwise` is false. If `columnwise` is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If `allocate` is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `<allocate>` true and `<columnwise>` false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluate the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-23: VALID PROPERTY/VALUE PAIRS FOR GLOBAL EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	none data set name	First compatible data set	The data set this feature refers to.
dataserries	average integral maximum minimum none rms stddev variance	none	The operation that is applied to the data series formed by the evaluation.
dataserriesmethod	auto integration summation	auto	The method to use for the data series: automatic, integration, or summation.

TABLE 7-23: VALID PROPERTY/VALUE PAIRS FOR GLOBAL EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
maximumobj	abs real	real	The value being maximized if dataseries is maximum.
minimumobj	abs real	real	The value being minimized if dataseries is minimum.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.

TABLE 7-23: VALID PROPERTY/VALUE PAIRS FOR GLOBAL EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	nonnegative integer array	All solutions	The solutions to use.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models. The level values (level1, level2, and so on) are the levels in the parametric sweep.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[EvalGlobalMatrix](#), [Eval](#), [EvalPoint](#)

EvalGlobalMatrix

Evaluate global matrix quantities.

SYNTAX

```

model.result().numerical().create(<ftag>,"EvalGlobalMatrix");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();

```

DESCRIPTION

`model.result().numerical().create(<ftag>, "EvalGlobalMatrix")` creates an evaluation feature with the name `<ftag>`.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to (`<columnwise>`) when `columnwise` is false. If `columnwise` is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If `allocate` is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `<allocate>` true and `<columnwise>` false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-24: VALID PROPERTY/VALUE PAIRS FOR GLOBAL MATRIX EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
data	none data set name	First compatible data set	The data set this feature refers to.
dataseries	average none sum	none	The operation that is applied to the data series formed by the evaluation over inner solutions.
descr	String	Model-dependent	The description of the expression in <code>expr</code> . Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if <code>evalmethod</code> is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The matrix variable to plot.
ignorenan	on off	on	When on, NaN is ignored when taking the average in the inner or outer data series. If all entries are NaN for a given component, however, you get NaN as output.

TABLE 7-24: VALID PROPERTY/VALUE PAIRS FOR GLOBAL MATRIX EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
outerdataseries	average none sum	none	The operation that is applied to the data series formed by the evaluation over outer solutions.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	nonnegative integer array	All solutions	The solutions to use.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double array	Empty	The times to use. Available when the underlying solution is transient.

TABLE 7-24: VALID PROPERTY/VALUE PAIRS FOR GLOBAL MATRIX EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
table	new table name	new	The table to use when calling <code>setResult()</code> or <code>appendResult()</code> . <code>new</code> indicates that a new table is created.
tablerows	inner outer	inner	Whether to use inner or outer solutions as rows in the table when calling <code>setResult()</code> or <code>appendResult()</code> . Applicable only for parametric sweep models.
trans	none inverse sy sz ys yz zs zy	none	The transformation to apply to the matrix.
unit	String	Model-dependent	The unit to use for the expression in <code>expr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.
y0	double array	Taken from the physics interfaces.	If <code>trans</code> is <code>sy</code> or <code>ys</code> : The characteristic admittance.
z0	double array	Taken from the physics interfaces.	If <code>trans</code> is <code>sz</code> or <code>zs</code> : The characteristic impedance.

SEE ALSO

[EvalGlobal](#), [Eval](#), [EvalPoint](#), [EvalPointMatrix](#)

EvalPoint

Evaluate quantities in points.

SYNTAX

```

model.result().numerical().create(<ftag>,"EvalPoint");
model.result().numerical(<ftag>).selection(...);
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();

```

DESCRIPTION

`model.result().numerical().create(<ftag>,"EvalPoint")` creates a point evaluation feature with the name `<ftag>`. Point evaluations can be performed both on points in a geometry or on cut points.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that there is one row per point, with one row containing data for all solution numbers. This is identical to `(<columnwise>)` when `<columnwise>` is `false`. If `<columnwise>` is `true`, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If `<allocate>` is `true`, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `<allocate>` `true` and `<columnwise>` `false`.

`model.result().numerical(<ftag>).isComplex()` returns `true` if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-25: VALID PROPERTY/VALUE PAIRS FOR EVALUATION IN POINTS

NAME	VALUE	DEFAULT	DESCRIPTION
<code>const</code>	String array of property/value pairs	Empty	Parameters to use in the expressions.
<code>data</code>	<code>none</code> data set name	First compatible data set	The data set this feature refers to.
<code>dataseries</code>	<code>average</code> <code>integral</code> <code>maximum</code> <code>minimum</code> <code>none</code> <code>rms</code> <code>stddev</code> <code>variance</code>	<code>none</code>	The operation that is applied to the data series formed by the evaluation.
<code>dataseriesmethod</code>	<code>auto</code> <code>integration</code> <code>summation</code>	<code>auto</code>	The method to use for the data series: automatic, integration, or summation.
<code>descr</code>	String	Model-dependent	The description of the expression in <code>expr</code> . Is used in the automatic title.
<code>differential</code>	<code>on</code> <code>off</code>	<code>on</code>	Whether the expression should be linearized at the linearization point. Applicable only if <code>evalmethod</code> is <code>harmonic</code>
<code>evalmethod</code>	<code>linpoint</code> <code>harmonic</code> <code>lintotal</code> <code>lintotalavg</code> <code>lintotalrms</code> <code>lintotalpeak</code>	<code>harmonic</code>	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
<code>expr</code>	String	Model-dependent	The expression to plot.
<code>innerinput</code>	<code>all</code> <code>first</code> <code>last</code> <code>manual</code> <code>manualindices</code> <code>interp</code>	<code>all</code>	How to input the solution to use. <code>manual</code> indicates that <code>solnum</code> is used. <code>manualindices</code> indicates that <code>solnumindices</code> is used. <code>interp</code> indicates that <code>t</code> is used.
<code>interp</code>	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
<code>looplevel</code>	integer row matrix	All solutions on all levels	The solutions to use, per level.

TABLE 7-25: VALID PROPERTY/VALUE PAIRS FOR EVALUATION IN POINTS

NAME	VALUE	DEFAULT	DESCRIPTION
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
maximumobj	abs real	real	The value being maximized if dataseries is maximum.
minimumobj	abs real	real	The value being minimized if dataseries is minimum.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	nonnegative integer array	All solutions	The solutions to use.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.

TABLE 7-25: VALID PROPERTY/VALUE PAIRS FOR EVALUATION IN POINTS

NAME	VALUE	DEFAULT	DESCRIPTION
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models. The level values (level1, level2, and so on) are the levels in the parametric sweep.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO[EvalGlobal](#)*EvalPointMatrix*

Evaluate matrix quantities at points in the geometry.

SYNTAX

```

model.result().numerical().create(<ftag>,"EvalPointMatrix");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();
model.result().numerical(<ftag>).selection();

```

DESCRIPTION

model.result().numerical().create(<ftag>,"EvalPointMatrix") creates an evaluation feature with the name <ftag>.

model.result().numerical(<ftag>).getReal() returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to (<columnwise>) when columnwise is false. If columnwise is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>) returns the imaginary part of complex result, recomputing the feature if necessary. If allocate is true, a zero-valued matrix is allocated even when the result is real. getImag() uses <allocate> true and <columnwise> false.

model.result().numerical(<ftag>).isComplex() returns true if the result is complex.

<outersolnum> is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

`model.result().numerical(<ftag>).selection()` selects the points where the matrix quantity should be evaluated. `model.result().numerical(<ftag>).selection().all()`; selects all points in the geometry.

The following properties are available:

TABLE 7-26: VALID PROPERTY/VALUE PAIRS FOR GLOBAL MATRIX EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
<code>const</code>	String array of property/value pairs	Empty	Parameters to use in the expressions.
<code>data</code>	<code>none</code> data set name	First compatible data set	The data set this feature refers to.
<code>dataseries</code>	<code>average</code> <code>none</code> <code>sum</code>	<code>none</code>	The operation that is applied to the data series formed by the evaluation over inner solutions.
<code>descr</code>	String	Model-dependent	The description of the expression in <code>expr</code> . Is used in the automatic title.
<code>differential</code>	<code>on</code> <code>off</code>	<code>on</code>	Whether the expression should be linearized at the linearization point. Applicable only if <code>evalmethod</code> is <code>harmonic</code>
<code>evalmethod</code>	<code>linpoint</code> <code>harmonic</code> <code>lintotal</code> <code>lintotalavg</code> <code>lintotalrms</code> <code>lintotalpeak</code>	<code>harmonic</code>	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
<code>expr</code>	String	Model-dependent	The matrix variable to plot.
<code>ignorenan</code>	<code>on</code> <code>off</code>	<code>on</code>	When <code>on</code> , NaN is ignored when taking the average in the inner or outer data series. If all entries are NaN for a given component, however, you get NaN as output.
<code>innerinput</code>	<code>all</code> <code>first</code> <code>last</code> <code>manual</code> <code>manualindices</code> <code>interp</code>	<code>all</code>	How to input the solution to use. <code>manual</code> indicates that <code>solnum</code> is used. <code>manualindices</code> indicates that <code>solnumindices</code> is used. <code>interp</code> indicates that <code>t</code> is used.
<code>interp</code>	double row matrix	Empty on all levels	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
<code>looplevel</code>	integer row matrix	All solutions on all levels	The solutions to use, per level.
<code>looplevelindices</code>	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1,1,20)</code> . Applicable when <code>looplevelinput</code> is <code>manualindices</code> on a level.

TABLE 7-26: VALID PROPERTY/VALUE PAIRS FOR GLOBAL MATRIX EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
outerdataseries	average none sum	none	The operation that is applied to the data series formed by the evaluation over outer solutions.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	nonnegative integer array	All solutions	The solutions to use.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.
tablerows	inner outer	inner	Whether to use inner or outer solutions as rows in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[EvalGlobal](#), [Eval](#), [EvalPoint](#), [EvalGlobalMatrix](#)

Create a far-field plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"FarField");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"FarField")` creates a plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Far-field plots are used to visualize radiation patterns in the far-field for antennas, for example. They can be added to all types of plot groups.

The following properties are available:

TABLE 7-27: VALID PROPERTY/VALUE PAIRS FOR FAR-FIELD PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
anglerestr	manual none	none	Whether there is a restriction on the angle range that is evaluated.
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions. Available in 1D.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions. Available in 1D.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line. Available in 1D.
center	double array of length 3	{0, 0, 0}	If circle is manual: The center of the circle to evaluate on. Available in 2D and 3D.
center2	double array of length 2	{0, 0}	If circle is manual and the data set is 2D: The center of the circle to evaluate on. Available in 1D.
center3	double array of length 3	{0, 0, 0}	If circle is manual and the data set is 3D: The center of the circle to evaluate on. Available in 1D.
circle	manual unit	unit	The circle to evaluate. Available in 1D.
colorexpr	String	Model-dependent.	The color of the far-field surface. Available in 3D.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customgridcolor	RGB-triplet	{0,0,1}	f grid is fine, normal, or coarse and gridcolor is custom: The grid's color. Available in 3D.
customlinecolor	RGB-triplet	{0,0,1} or last used edgcolor.	The color to use for the lines. Active when linecolor is set to custom. Available in 1D.
data	none parent data set name	parent	The data set this feature refers to.

TABLE 7-27: VALID PROPERTY/VALUE PAIRS FOR FAR-FIELD PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
directivity	on off	off	Whether to compute the maximum directivity and the direction where it is attained. Available in 2D and 3D.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
grid	fine normal coarse none	none	The type of grid to display on the far-field surface. Available in 3D.
gridcolor	custom black blue cyan green magenta red white yellow	black	If grid is fine, normal, or coarse: The grid's color. Available in 3D.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited. Available in 2D and 3D.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends. Available in 1D.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property. Available in 1D.
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual. Available in 1D.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line. Available in 1D.
linewidth	double	0.5	The line width. Available in 1D.

TABLE 7-27: VALID PROPERTY/VALUE PAIRS FOR FAR-FIELD PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line. Available in ID.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line. Available in ID.
looplevel	integer row matrix	all on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set. Available in ID.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set. Available in ID.
normal	double array of length 3	{0, 0, 1}	If circle is manual and the data set is 3D: The normal of the circle to evaluate on. Available in ID.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.
phidisc	integer ≥ 2	50	The angular discretization in the phi direction. Available in ID.
phimin	double	0	If anglerestr is manual: The minimum phi angle in degrees.

TABLE 7-27: VALID PROPERTY/VALUE PAIRS FOR FAR-FIELD PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
phirange	double	360	If <code>anglerestr</code> is manual: The phi angle's range in degrees.
prefixintitle	String	Empty	Added prefix to contribution to title.
radius	double	1	If <code>circle</code> is manual: The radius of the circle to evaluate on.
rangecoloractive	on off	off	Whether to use the manual color range specified in <code>rangecolormin</code> and <code>rangecolormax</code> . The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when <code>rangecoloractive</code> is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when <code>rangecoloractive</code> is on.
rangedataactive	on off	off	Whether to use the manual data range specified in <code>rangedatamin</code> and <code>rangedatamax</code> . Values outside the data range are not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when <code>rangedataactive</code> is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when <code>rangedataactive</code> is on.
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>innerinput</code> is manualindices.
solrepresentation	<code>solnum</code> <code>solutioninfo</code>	<code>solutioninfo</code>	Indicates which method of selecting solutions is active.
sphere	manual unit	unit	The sphere to evaluate on. Available in 2D and 3D.
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
thetadisc	integer ≥ 2	10	The angular discretization in the theta direction. Available in 2D and 3D.
thetamin	double	0	If <code>anglerestr</code> is manual: The minimum theta angle in degrees. Available in 2D and 3D.
thetarange	double	360	If <code>anglerestr</code> is manual: The phi angle's range in degrees. Available in 2D and 3D.
threshold	double		Threshold value for the evaluated radius maps to the plotted radius 0 if <code>thresholdactive</code> is on (3D only).
thresholdactive	on off	off	Use a manual threshold value (3D only).

TABLE 7-27: VALID PROPERTY/VALUE PAIRS FOR FAR-FIELD PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

ATTRIBUTES

None.

Filter

Add a filter attribute to a plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"Filter");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

model.result(<pgtag>).feature(<ftag>).create(<atag>,"Filter") creates an element filter attribute named <atag> belonging to the plot feature <ftag>.

Use filters to limit plots to elements satisfying a condition.

Filters can be added to arrow plots, contour plots, isosurface plots, line plots, mesh plots, slice plots, surface plots, and volume plots.

The following properties are available:

TABLE 7-28: VALID PROPERTY/VALUE PAIRS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
expr	String	1	The logical expression that must be satisfied for an element to be included in the plot.
nodespec	all any xor smooth	smooth	The set of nodes in an element that have to satisfy the logical expression for it to be included in the plot. The smooth option creates smooth edges by "cutting" through elements.

Filter (Particle Tracing)

Add a filter attribute to a particle trajectories or particle plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"ParticleTrajectoriesFilter");  
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"ParticleTrajectoriesFilter")` creates a filter attribute named `<atag>` belonging to the plot feature `<ftag>`.

Use filters to plot a subset of particles in a data set.

Filters can be added to particle trajectories plots and particle plots.

The following properties are available:

TABLE 7-29: VALID PROPERTY/VALUE PAIRS FOR THE FILTER FOR PARTICLE TRAJECTORIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
evaluate	all fraction number	all	Determines which particles are rendered when creating the plot.
fraction	Double	1	Fraction of particles that are rendered. Active when evaluate is set to fraction.
logical	Double	1	Logical statement that must be true for a particle to be included in the plot. Active when type is set to logical.
number	Integer	100	Number of particles that are rendered. Active when evaluate is set to number.
type	all primary secondary logical	all	Determines which particles are included when creating the plot.

Filter (Point Trajectories)

Add a filter attribute to a point trajectories plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"PointTrajectoriesFilter");  
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"PointTrajectoriesFilter")` creates a filter attribute named `<atag>` belonging to the plot feature `<ftag>`.

Use filters to plot a subset of points in a data set.

Filters can be added to point trajectories plots.

The following properties are available:

TABLE 7-30: VALID PROPERTY/VALUE PAIRS FOR THE FILTER FOR POINT TRAJECTORIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
evaluate	all fraction number	all	Determines which points are rendered when creating the plot.
fraction	Double	1	Fraction of points that are rendered. Active when evaluate is set to fraction.
logical	Double	1	Logical statement that must be true for a point to be included in the plot. Active when type is set to logical.

TABLE 7-30: VALID PROPERTY/VALUE PAIRS FOR THE FILTER FOR POINT TRAJECTORIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
number	Integer	100	Number of points that are rendered. Active when evaluate is set to number.
type	all logical	all	Determines which points are included when creating the plot.

Filter (Ray Tracing)

Add a filter attribute to a ray trajectories or ray plot.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"RayTrajectoriesFilter");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"RayTrajectoriesFilter")` creates a filter attribute named `<atag>` belonging to the plot feature `<ftag>`.

Use filters to plot a subset of rays in a data set.

Filters can be added to ray trajectories plots and ray plots.

The following properties are available:

TABLE 7-31: VALID PROPERTY/VALUE PAIRS FOR THE FILTER FOR RAY TRAJECTORIES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
evaluate	all fraction number	all	Determines which rays are rendered when creating the plot.
fraction	Double	1	Fraction of rays that are rendered. Active when evaluate is set to fraction.
logical	Double	1	Logical statement that must be true for a ray to be included in the plot. Active when type is set to logical.
number	Integer	100	Number of rays that are rendered. Active when evaluate is set to number.
type	all primary secondary logical	all	Determines which rays are included when creating the plot.

Global (numerical)

Evaluate global quantities.

SYNTAX

```
model.result().numerical().create(<ftag>,"Global");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getData();
model.result().numerical(<ftag>).getData(<expressionIndex>);
model.result().numerical(<ftag>).getImagData();
model.result().numerical(<ftag>).getImagData(<expressionIndex>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).getNData();
model.result().numerical(<ftag>).run();
```

DESCRIPTION

`model.result().numerical().create(<ftag>, "Global")` creates an evaluation feature with the name `<ftag>`.

Global is a feature made specifically for users of the COMSOL API and does not appear in the COMSOL Desktop. Global features support multiple expressions as well as some additional advanced properties not available in [EvalGlobal](#).

`result = model.result().numerical(<ftag>).getData()` returns the real part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][value]`.

`model.result().numerical(<ftag>).getData(<expressionIndex>)` returns the real part of the result for one expression, equivalent to `result[expressionIndex]`.

`result = model.result().numerical(<ftag>).getImagData()` returns the imaginary part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][value]`.

`model.result().numerical(<ftag>).getImagData(<expressionIndex>)` returns the imaginary part of the result for one expression, equivalent to `result[expressionIndex]`.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).getNData()` returns the number of data vectors in the result.

The following properties are available:

TABLE 7-32: VALID PROPERTY/VALUE PAIRS FOR GLOBAL EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	none data set name	First compatible data set	The data set this feature refers to.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent	The expression to evaluate.
complexfun	on off	on	Use complex-valued functions with real input.
matherr	on off	off	Error for undefined operation or variable.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
phase	double	0	Evaluate solution at this angle, given in degrees.
solnum	nonnegative integer array	All solutions	The solutions to use.

TABLE 7-32: VALID PROPERTY/VALUE PAIRS FOR GLOBAL EVALUATION

NAME	VALUE	DEFAULT	DESCRIPTION
t	double array	Empty	The times to use. Available when the underlying solution is transient.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Eval](#), [Interp](#), [EvalPoint](#), [EvalGlobal](#)

Global (plot)

Create a global plot for 1D plot groups and polar plot groups.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Global");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Global")` creates a plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Global plot is used to visualize defined variables, such as solutions to ODEs or coupling operators with a destination. Global plots can be added to 1D and polar plot groups.

The following properties are available:

TABLE 7-33: VALID PROPERTY/VALUE PAIRS FOR GLOBAL PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line.
autounit	on off	off	Whether the automatic legends should include the unit.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.

TABLE 7-33: VALID PROPERTY/VALUE PAIRS FOR GLOBAL PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
descriptionintitle	on off	on	Whether the title contribution should contain the description when <code>titletype</code> is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if <code>evalmethod</code> is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when <code>titletype</code> is custom.
freqmax	integer		If <code>xaxisdata</code> is spectrum and <code>freqrangeactive</code> is true: The upper frequency bound.
freqmin	integer		If <code>xaxisdata</code> is spectrum and <code>freqrangeactive</code> is true: The lower frequency bound.
freqrangeactive	on off	false	If <code>xaxisdata</code> is spectrum: Controls whether a manual frequency range is used.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. <code>manual</code> indicates that <code>solnum</code> is used. <code>manualindices</code> indicates that <code>solnumindices</code> is used. <code>interp</code> indicates that <code>t</code> is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends.	Manual legends active when <code>legendmethod</code> is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.

TABLE 7-33: VALID PROPERTY/VALUE PAIRS FOR GLOBAL PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
nfreqs	integer	1	If xaxisdata is spectrum and nfreqsactive is true: The number of frequencies to plot.
nfreqsactive	Boolean	false	If xaxisdata is spectrum: Controls whether the number of frequencies is set manually.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title.
scale	Boolean	false	If xaxisdata is spectrum: The frequency spectrum is transformed so that it has the same scale as the original data.

TABLE 7-33: VALID PROPERTY/VALUE PAIRS FOR GLOBAL PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
xdata	expr solution spectrum phase	solution	x-axis data. expr uses the expression in xdataexpr. solution uses the available solutions in the underlying data set, such as time steps. phase uses a phase range, and rearranged phase plots.
xdataexpr	String	Model-dependent	Expression for x-axis data.
xdatadescr	String	Model-dependent	Description of expression in xdataexpr.
xdataphaserange	double array	range(0, 0.5, 2 π)	The phases for which the expressions should be evaluated when xdata is phase.
xdataphaseunit	String	rad	The unit in which xdataphaserange is described.

TABLE 7-33: VALID PROPERTY/VALUE PAIRS FOR GLOBAL PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
xdatasolnumtype	all inner outer valid level	outer	Whether the expression should be evaluated for every inner or every outer solution, or for a specific level (level1, level2, and so on). Applicable only for models containing multiple levels.
xdataunit	String	Model-dependent	The unit to use for the expression in xdataexpr. If the old unit is not valid when the expression changes, the unit property is reset to default.

ATTRIBUTES

None.

SEE ALSO

[LineGraph](#)

Grid1D, Grid2D, Grid3D

Create a data set that can evaluate 1D, 2D, or 3D functions or other data sets on a domain with a grid. For example, you can use these data sets to evaluate BEM and far-field operators.

SYNTAX

```
model.result().dataset().create(<dtag>,"Grid1D");
model.result().dataset().create(<dtag>,"Grid2D");
model.result().dataset().create(<dtag>,"Grid3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Grid1D")` creates a 1D grid data set feature named `<dtag>`.

This data set provides support for evaluation of functions or other data sets on a domain with a grid. All functions in the same function list as the selected function can also be evaluated. The domain is an interval for `Grid1D`, a rectangle for `Grid2D`, and a block for `Grid3D`. The domain does not need to have the same dimension as the number of arguments to the function.

The following properties are available:

TABLE 7-34: VALID PROPERTY/VALUE PAIRS FOR GRID DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set tag	none	The data set this feature refers to.
function	none all function tag	none	The function this feature refers to.
par <i>i</i>	String	Model-dependent	The name of the parameter. par1, par2, and par3 for Grid3D, for example.
parmin1	double	0	Lower bound for the first dimension of the domain.
parmax1	double	1	Upper bound for the first dimension of the domain.
parmin2	double	0	Lower bound for the second dimension of the domain. (For Grid2D and Grid3D.)

TABLE 7-34: VALID PROPERTY/VALUE PAIRS FOR GRID DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
parmax2	double	1	Upper bound for the second dimension of the domain. (For Grid2D and Grid3D.)
parmin3	double	0	Lower bound for the third dimension of the domain. (For Grid3D.)
parmax3	double	1	Upper bound for the third dimension of the domain. (For Grid3D.)
resi	integer >= 2	1000 for 1D, 100 for 2D, and 30 for 3D	The number of points into which each dimension is discretized. res1, res2, and res3 for Grid 3D, for example.
source	function data	function	Use a function or a data set as the source.

Height

Add a height attribute for 2D line and surface plots.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>, "Height");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>, "Height")` creates a Height attribute feature with the name `<atag>`, belonging to the feature `<ftag>`.

Height attributes are available for 2D line and surface plots. Adding a height attribute changes the rendered view to 3D. The plot group, however, still remains a 2D plot group. Any other plots in the group that do not have a height attribute are rendered at $z = 0$.

The following properties are available:

TABLE 7-35: VALID PROPERTY/VALUE PAIRS FOR HEIGHT ATTRIBUTES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when <code>titletype</code> is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if <code>evalmethod</code> is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when <code>titletype</code> is custom.

TABLE 7-35: VALID PROPERTY/VALUE PAIRS FOR HEIGHT ATTRIBUTES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
heightdata	parent expr	parent	parent uses the expression and description in the plot the height feature has been added to. expr uses the expr and descr properties.
offset	double	0	The offset along the z-axis where to place the height plot.
prefixintitle	String	Empty	Added prefix to contribution to title.
scale	positive double	1	If scaleactive is true: The scale factor for the height.
scaleactive	Boolean	false	Whether to use manual scaling.
suffixintitle	String	Empty	Added suffix to contribution to title.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

SEE ALSO[Color](#), [Deform](#)*Histogram*

Create a histogram plot for 1D and 2D plot groups.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Histogram");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Histogram") creates a histogram plot feature named <ftag> belonging to the plot group <pgtag>.

Histograms are used to visualize the distribution of the range of an expression. The result is a plot with the expression's range on the *x*-axis and element length, area, or volume on the *y*-axis. Histogram plots can be added to 1D plot groups.

The following properties are available:

TABLE 7-36: VALID PROPERTY/VALUE PAIRS FOR HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
cumulative	Boolean	false	Plot a cumulative histogram.
customlinecolor	RGB-triplet	{0,0,1} or last used edgcolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
function	continuous discrete	continuous	Use a continuous function or a discrete function (that is, using a constant level in each bin) for the histogram.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
level	fromdataset volume surface line point	fromdataset	The geometry level to evaluate on.
limits	double array	0 1	The bin limits if method is limits.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.

TABLE 7-36: VALID PROPERTY/VALUE PAIRS FOR HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
method	limit number	number	The method used to specify the bins in which the expression's range is split
normalization	integral none peak	none	The normalization of the values plot on the y-axis.
number	nonnegative integer	10	The number of bins if method is number.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title

TABLE 7-36: VALID PROPERTY/VALUE PAIRS FOR HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
refine	nonnegative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
sdim	fromdataset 0 1 2 3	fromdataset	The space dimension to evaluate the underlying data set in
solnum	integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
type	curve solid	curve	Use a filled histogram (solid) or a histogram drawn using lines (curve). This property is only available when function is discrete.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.

TABLE 7-36: VALID PROPERTY/VALUE PAIRS FOR HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

SEE ALSO[MatrixHistogram](#)*HistogramHeight*

Add a height attribute for 2D histogram plots.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"HistogramHeight");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"HistogramHeight")` creates a `HistogramHeight` attribute feature with the name `<atag>`, belonging to the feature `<ftag>`.

Adding a height attribute to a 2D histogram plot changes the rendered view to 3D. The plot group, however, still remains a 2D plot group. Any other plots in the group that do not have heights attribute is rendered at `z=0`.

The following properties are available:

TABLE 7-37: VALID PROPERTY/VALUE PAIRS FOR HISTOGRAM HEIGHT ATTRIBUTES

PROPERTY	VALUE	DEFAULT	DESCRIPTION
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
offset	double	0	The offset along the z-axis where to place the height plot
prefixintitle	String	Empty	Added prefix to contribution to title.
scale	positive double	1	If <code>scaleactive</code> is true: The scale factor for the height.
scaleactive	Boolean	false	Whether to use manual scaling
suffixintitle	String	Empty	Added suffix to contribution to title
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.

SEE ALSO

[Height](#), [Histogram](#)

Image1D, Image2D, Image3D

Export an image.

SYNTAX

```
model.result().export().create(<ftag>, "Image1D");
model.result().export().create(<ftag>, "Image2D");
model.result().export().create(<ftag>, "Image3D");
model.result().export().create(<ftag>, <pgtag>, "Image1D");
model.result().export().create(<ftag>, <pgtag>, "Image2D");
model.result().export().create(<ftag>, <pgtag>, "Image3D");
model.result().export(<ftag>).set(property, <value>);
model.result().export(<ftag>).run();
```

DESCRIPTION

`model.result().export().create(<ftag>, "Image1D")` creates a 1D image feature with the name `<ftag>`.

`model.result().export().create(<ftag>, <pgtag>, "Image1D")` creates a 1D image feature with the name `<ftag>` for the plot group `<pgtag>`.

`result().export(<ftag>).set("plotgroup", <ptag>)` changes the source of the image to the plot group named `<ptag>`.

Image features can be used both to export images and to have ready-made views of plot groups.

The following properties are available:

TABLE 7-38: VALID PROPERTY/VALUE PAIRS FOR IMAGE EXPORT

NAME	VALUE	DEFAULT	DESCRIPTION
alwaysask	on off	off	Always ask for filename when saving. This property is ignored when running without a GUI.
antialias	on off	on	Enable/disable antialiasing.
axes	on off	on	If options is on; enable/disable display of the coordinate axes. Used for 1D and 2D plots.
axisorientation	on off	on	If options is on; enable/disable display of the axis orientation indicator. Used for 3D plots.
background	current color transparent	color	The background color.
bmpfilename	String		The name of the output file if imagetype is bmp.
customcolor	double array	{1, 1, 1}	If background is color; the red, green, and blue components of the background color.
epsfilename	String		The name of the output file if imagetype is eps.
fontsize	integer	9	The font size.
grid	on off	on	If options is on; enable/disable display of the coordinate grid. Used for 3D plots.
height	double	600	The height of the image.

TABLE 7-38: VALID PROPERTY/VALUE PAIRS FOR IMAGE EXPORT

NAME	VALUE	DEFAULT	DESCRIPTION
imagetype	png eps jpeg bmp	png	The type of image to export; eps can only be used for 1D plots.
jpegfilename	String		The name of the output file if imagetype is jpeg.
legend	on off	on	If options is on; enable/disable display of the legend.
lockratio	Boolean	false	If true, then the aspect ratio of the image is preserved when the width or the height is changed.
logo	on off	on	If options is on; enable/disable display of the logotype.
options	on off	off	Enable/disable optional components of the image.
pngfilename	String		The name of the output file if imagetype is png.
resolution	integer	96	The frame resolution in dots per inch.
plotgroup	String		The plot group to export.
title	on off	on	If options is on; enable/disable display of the title.
unit	px mm in	mm	The unit for the dimensions of the image.
width	double	800	The width of the image.

The default values listed are valid before any image has been exported successfully. After that, the settings from the last successful image export are used as default values the next time an image export feature is created.

Changing plot group after creation does not reset plot group-dependent default settings (title, colorlegend).

SEE ALSO

[Animation](#)

InterferencePattern

Plot the interference pattern resulting from the intersection of rays with a cut plane.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"InterferencePattern");
model.result(<pgtag>).feature(<ftag>).selection(...);
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"InterferencePattern")` creates an interference pattern plot feature named `<ftag>` belonging to the plot group `<pgtag>`. The interference pattern plot is used to visualize the interference of multiple rays as they intersect a cut plane. Interference pattern plots can be added to 2D plot groups.

The following properties are available:

TABLE 7-39: VALID PROPERTY/VALUE PAIRS FOR INTERFERENCE PATTERN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this surface
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
originspec	manual index intensity	manual	Determines how the origin of the interference plot is defined.
originx	double	0	If originspec is manual, determines the x-coordinate of the origin.
originy	double	0	If originspec is manual, determines the y-coordinate of the origin.
originz	double	0	If originspec is manual, determines the z-coordinate of the origin.

TABLE 7-39: VALID PROPERTY/VALUE PAIRS FOR INTERFERENCE PATTERN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
rangexactive	on off	off	Whether to use a manual range of x-coordinates for the grid points.
rangexmax	double	Plot-dependent	The maximum x-coordinate in the local coordinate system of the interference pattern plot. Active when rangexactive is on.
rangexmin	double	Plot-dependent	The minimum x-coordinate in the local coordinate system of the interference pattern plot. Active when rangexactive is on.
rangeyactive	on off	off	Whether to use a manual range of y-coordinates for the grid points.
rangeymax	double	Plot-dependent	The maximum y-coordinate in the local coordinate system of the interference pattern plot. Active when rangeyactive is on.
rangeymin	double	Plot-dependent	The minimum y-coordinate in the local coordinate system of the interference pattern plot. Active when rangeyactive is on.
rayindex	int	1	The index of the ray used to define the location of the origin of the interference pattern plot. The origin is located at the intersection of the specified ray with the cut plane. Active when originspec is index.

TABLE 7-39: VALID PROPERTY/VALUE PAIRS FOR INTERFERENCE PATTERN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
xyresolution	int	200	The number of grid points in each direction to use when rendering the interference pattern. Cannot be greater than 1000.

Interp

Evaluate expressions in arbitrary points or data sets using interpolation.

SYNTAX

```

model.result().numerical().create(<ftag>,"Interp");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getCoordinates();
model.result().numerical(<ftag>).getData();
model.result().numerical(<ftag>).getData(<expressionIndex>);
model.result().numerical(<ftag>).getImagData();
model.result().numerical(<ftag>).getImagData(<expressionIndex>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).getElements();
model.result().numerical(<ftag>).getNData();
model.result().numerical(<ftag>).run();
model.result().numerical(<ftag>).setInterpolationCoordinates(<value>);

```

DESCRIPTION

`model.result().numerical().create(<ftag>,"Interp")` creates an interpolation feature with the name `<ftag>`.

Interp is a feature made specifically for users of the COMSOL API and does not appear in the COMSOL Multiphysics GUI. Interp combines cut points and evaluation features, as well as allowing evaluation of arbitrary data sets. It supports multiple expressions and some additional advanced properties not available when using cut points and [EvalPoint](#) features.

`result = model.result().numerical(<ftag>).getData()` returns the real part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getData(<expressionIndex>)` returns the real part of the result for one expression, equivalent to `result[expressionIndex]`.

`result = model.result().numerical(<ftag>).getImagData()` returns the imaginary part of the result, recomputing the feature if necessary. `result` is a three-dimensional double matrix ordered `result[expression][solnum][coordinates]`.

`model.result().numerical(<ftag>).getImagData(<expressionIndex>)` returns the imaginary part of the result for one expression, equivalent to `result[expressionIndex]`.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).getElements()` returns indices to columns in `p` of a simplex mesh.

`model.result().numerical(<ftag>).getNData()` returns the number of data vectors in the result.

`model.result().numerical(<ftag>).getCoordinates()` returns the coordinates of the interpolation.

`model.result().numerical(<ftag>).setInterpolationCoordinates(<value>)` is identical to `model.result().numerical(<ftag>).set(coord, <value>)`, when `value` is of type `double[][]`.

The columns of the `coord` property are the coordinates for the evaluation points. If the number of rows in `coord` equals the space dimension, then `coord` are global coordinates, and the selection (or the property `edim`) determines the dimension in which the expressions are evaluated. For instance, dimension 2 means that the expressions are evaluated on boundaries in a 3D model. If the dimension to evaluate on is less than the space dimension, then the points in `coord` are projected onto the closest point on a domain of that dimension. It is further possible to select a single geometric entity of a given dimension. If so, then the closest point on that domain in the given dimension is used.

If the number of rows in `coord` is less than the space dimension, then these coordinates are parameter values on a geometry face or edge. In that case, the domain number for that face or edge must be specified in the selection.

For data sets that do not support selections, the `edim` property must always be used, and no selections are possible.

The following properties are available:

TABLE 7-40: VALID PROPERTY/VALUE PAIRS FOR INTERP

NAME	VALUE	DEFAULT	DESCRIPTION
<code>complexfun</code>	<code>on</code> <code>off</code>	<code>on</code>	Use complex-valued functions with real input
<code>const</code>	String array of property/value pairs	Empty	Parameters to use in the expressions.
<code>coord</code>	double matrix	Empty	The coordinates to evaluate in. Data sets that define an interpolation in themselves, such as cut planes, revolve and so forth, can be evaluated directly in the coordinates that define their shapes, without specifying <code>coord</code> .
<code>coorderr</code>	<code>on</code> <code>off</code>	<code>on</code>	If <code>on</code> , evaluating for a set of points that all fall outside the geometry results in an error being reported.
<code>data</code>	<code>none</code> data set name	First compatible data set	The data set this feature refers to.

TABLE 7-40: VALID PROPERTY/VALUE PAIRS FOR INTERP

NAME	VALUE	DEFAULT	DESCRIPTION
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
edim	auto 0 1 2 3	auto	Element dimension. This is used for arbitrary data sets. Solution data sets use selections as usual to specify where to perform the evaluation, and ignore the value of the edim property.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent	The expressions to evaluate.
ext	double between 0 and 1	0.1	Extrapolation distance: How much outside the mesh that the interpolation searches. The scale is in terms of the local element size.
matherr	on off	off	Error for undefined operation or variable
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
phase	double	0	Evaluate solution at this angle, given in degrees.
refine	auto integer	auto	Refinement of elements for evaluation points.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
solnum	non-negative integer array	All solutions	The solutions to use.
t	double array	Empty	The times to use. Available when the underlying solution is transient.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Eval](#), [Global \(numerical\)](#), [EvalPoint](#), [EvalGlobal](#)

IntVolume, *IntSurface*, *IntLine*

Add a volume, surface, or line integration to evaluate numerical results.

SYNTAX

```
model.result().numerical().create(<ftag>,"IntVolume");
model.result().numerical().create(<ftag>,"IntSurface");
model.result().numerical().create(<ftag>,"IntLine");
model.result().numerical(<ftag>).selection(...);
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();
```

DESCRIPTION

`model.result().numerical().create(<ftag>,"IntVolume")` creates a volume integral feature with the name `<ftag>`.

`model.result().numerical().create(<ftag>,"IntSurface")` creates a surface integral feature with the name `<ftag>`.

`model.result().numerical().create(<ftag>,"IntLine")` creates a line integral feature with the name `<ftag>`.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to `(columnwise)` when `columnwise` is false. If `columnwise` is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(allocate, columnwise)` returns the imaginary part of complex result, recomputing the feature if necessary. If `allocate` is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `allocate` true and `columnwise` false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-41: VALID PROPERTY/VALUE PAIRS FOR INTEGRATION

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/ value pairs	Empty	Parameters to use in the expressions
data	none data set name	First compatible data set	The data set this feature refers to

TABLE 7-41: VALID PROPERTY/VALUE PAIRS FOR INTEGRATION

NAME	VALUE	DEFAULT	DESCRIPTION
dataseries	average integral maximum minimum none rms stddev variance	none	The operation that is applied to the data series formed by the evaluation
dataseriesmethod	auto integration summation	auto	The method to use for the data series: automatic, integration, or summation.
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
intorder	positive integer	4	The integration order
intorderactive	on off	off	Whether to use manually specified integration order
intsurface	on off	off	Compute surface integral. Available for line integration features of axisymmetric models.
intvolume	on off	off	Compute volume integral. Available for surface integration features of axisymmetric models.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.

TABLE 7-41: VALID PROPERTY/VALUE PAIRS FOR INTEGRATION

NAME	VALUE	DEFAULT	DESCRIPTION
maximumobj	abs real	real	The value being maximized if dataseries is maximum
method	auto integration summation	auto	The integration method
minimumobj	abs real	real	The value being minimized if dataseries is minimum
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
solnum	non-negative integer array	All solutions	The solutions to use
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models. The level values (level1, level2, and so on) are the levels in the parametric sweep.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[EvalGlobal](#), [EvalPoint](#), [Global \(numerical\)](#)

Isosurface

Create an isosurface plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Isosurface");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Isosurface")` creates an isosurface plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

The following properties are available:

TABLE 7-42: VALID PROPERTY/VALUE PAIRS FOR ISOSURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this isosurface.
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic

TABLE 7-42: VALID PROPERTY/VALUE PAIRS FOR ISOSURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color and deformation scale is inherited from.
interactive	on off	off	If true, the isosurfaces can be moved interactively after the plot has been made.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
levelmethod	number levels	number	How to enter isosurface levels.
levels	double array		The levels to plot. Active when levelmethod equals levels.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
number	positive integer	5	Total number of isosurface levels. Active when levelmethod equals number.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.

TABLE 7-42: VALID PROPERTY/VALUE PAIRS FOR ISOSURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
smooth	none internal everywhere	internal	Smoothing settings.
shift	double	0	If interactive is on: The shift that is applied to the level values.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
userder	Boolean	true	Only for isosurface plots: If true, space derivatives of the isosurface expression are used to produce smoother plots.

ATTRIBUTES

[Color](#), [Deform](#), [Filter](#)

SEE ALSO

[Isosurface \(data set\)](#), [Slice](#), [Volume](#)

Isosurface (data set)

Create an isosurface data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Isosurface");  
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Isosurface")` creates an isosurface data set feature named `<dtag>`.

The following properties are available:

TABLE 7-43: VALID PROPERTY/VALUE PAIRS FOR ISOSURFACE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
descr	String		Description of expr.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	u	Expression to find constant level for.
level	double array	{0}	Levels that defines the data set if levelmethod is levels.
levelmethod	number levels	number	How to specify levels.
number	positive integer	5	Total number of contour levels. Active when levelmethod is number.
unit	String		Unit of expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[Isosurface](#)

Join

Join data sets for joining two data sets to form the difference, for example.

SYNTAX

```
model.result().dataset().create(<dtag>,"Join");  
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Join")` creates a join data set feature named `<dtag>`.

The following properties are available:

TABLE 7-44: VALID PROPERTY/VALUE PAIRS FOR JOIN DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	None	The first source data set.
data2	none data set name	None	The second source data set.
expr	String	data1-data2	When method is general: The way the evaluations in the first and second data sets is combined. data1 and data2 can be used as symbols for the values from the two data sets, respectively.
method	difference differencenorm explicit general product quotient sum	difference	How to combine the results from evaluating in the two source data sets.
solutions	all one	all	Whether to expose one or all solutions from the first data set.
solutions2	all one	all	Whether to expose one or all solutions from the second data set.

Line

Create a line plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Line");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Line")` creates a line plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Line plots display a quantity on lines, curves and edges in 2D or 3D.

The following properties are available:

TABLE 7-45: VALID PROPERTY/VALUE PAIRS FOR LINE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this line plot.
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.

TABLE 7-45: VALID PROPERTY/VALUE PAIRS FOR LINE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, tube scale, and deformation scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
inheritrangescale	Boolean	true	If inheritplot is not none and linetype is tube: Determines if the tube scale is inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
linetype	line tube	line	Plot lines or tubes.

TABLE 7-45: VALID PROPERTY/VALUE PAIRS FOR LINE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
looplevel	array of nonnegative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
outersolnum	nonnegative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title
radiusexpr	String	1	The tube radius. Active when linetype is tubes.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
smooth	none internal everywhere	internal	Smoothing settings

TABLE 7-45: VALID PROPERTY/VALUE PAIRS FOR LINE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
tuberadiusscale	double	1	The scale factor applied to the tube radii if tuberadiusscaleactive is true.
tuberadiusscaleactive	Boolean	false	If true, tuberadiusscale is used, otherwise the scale factor is computed automatically.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
wireframe	on off	off	Whether to plot filled elements or only their edges.

ATTRIBUTES

Deform, [Filter](#), [Height](#)

EXAMPLES

Line plot on 2D solution:

Code for use with Java

```
DatasetFeature ds = model.result().dataset().create("dset1", "Solution");
ds.set("solution", "sol1");

ResultFeature pg = model.result().create("pg1",2);
```

```
pg.set("data", "dset1");
pg.create("line1", "Line");
pg.feature("line1").set("expr", "3*u");
```

Code for use with MATLAB

```
ds = model.result.dataset.create('dset1', 'Solution');
ds.set('solution', 'sol1');

pg = model.result.create('pg1', 2);
pg.set('data', 'dset1');
pg.create('line1', 'Line');
pg.feature('line1').set('expr', '3*u');
```

Line plot on cut plane in 3D, using the Thermal color table:

Code for use with Java

```
DatasetFeature ds2 = model.result().dataset().create("cp1", "CutPlane");
ds2.set("data", "dset2");

ResultFeature pg2 = model.result().create("pg2", 3);
pg2.create("line2", "Line");
pg2.feature("line2").set("data", "cp1");
pg2.feature("line2").set("expr", "2*u");
pg2.feature("line2").set("colortable", "Thermal");

pg2.run();
```

Code for use with MATLAB

```
ds2 = model.result.dataset.create('cp1', 'CutPlane');
ds2.set('data', 'dset2');

pg2 = model.result.create('pg2', 3);
pg2.create('line2', 'Line');
pg2.feature('line2').set('data', 'cp1');
pg2.feature('line2').set('expr', '2*u');
pg2.feature('line2').set('colortable', 'Thermal');

pg2.run;
```

SEE ALSO

[LineGraph](#)

LineGraph

Create a line graph plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>, "LineGraph");
model.result(<pgtag>).feature(<ftag>).selection(...);
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>, "LineGraph")` creates a line graph plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Line graph plot is used to visualize quantities on lines, either cut lines or boundaries (2D) and edges (3D) in a geometry. Line graph plots can be added to 1D plot groups.

The following properties are available:

TABLE 7-46: VALID PROPERTY/VALUE PAIRS FOR LINE GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
autlegends	on off	on	Whether to use the automatically generated legends or the legends defined in the legends property.
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customlinecolor	RGB-triplet	{0,0,1} or last used edgcolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.

TABLE 7-46: VALID PROPERTY/VALUE PAIRS FOR LINE GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title.
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.

TABLE 7-46: VALID PROPERTY/VALUE PAIRS FOR LINE GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
solnum	integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
xdata	arc reversedarc expr	solution	x-axis data. expr uses the expression in xdataexpr. arc uses the curve's arc length, and reversedarc uses the arc length measured from the curve's endpoint.
xdataexpr	String	Model-dependent	Expression for x-axis data
xdatadescr	String	Model-dependent	Description of expression in xdataexpr.
xdataunit	String	Model-dependent	The unit to use for the expression in xdataexpr. If the old unit is not valid when the expression changes, the unit property is reset to default.

ATTRIBUTES

None

SEE ALSO

[Line](#)

MatrixHistogram

Create a matrix histogram plot for 2D plot groups.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"MatrixHistogram");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"MatrixHistogram")` creates a histogram plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Use a matrix histogram if you have a precomputed matrix that you want to visualize as a 2D histogram. For example, in a fatigue analysis, you can use it for rainflow counting to visualize the distribution of stress amplitudes and mean stresses. The matrix then contains data points in the xy -plane, where the x - and y -values are stresses. Matrix histogram plots can be added to 2D plot groups.

The following properties are available:

TABLE 7-47: VALID PROPERTY/VALUE PAIRS FOR MATRIX HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
axisunit	String	Empty	Unit for the histogram axes.
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color the slices
colorlegend	on off	on	Whether to show color legend when coloring is set to colortable
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic

TABLE 7-47: VALID PROPERTY/VALUE PAIRS FOR MATRIX HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	all on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title
solnum	integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions

TABLE 7-47: VALID PROPERTY/VALUE PAIRS FOR MATRIX HISTOGRAM PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1,1,20)</code> . Applicable when <code>innerinput</code> is <code>manualindices</code> .
solrepresentation	<code>solnum</code> <code>solutioninfo</code>	<code>solutioninfo</code>	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when <code>titletype</code> is <code>manual</code>
titletype	<code>auto</code> <code>custom</code> <code>manual</code> <code>none</code>	<code>auto</code>	<code>auto</code> if the title contribution should be computed automatically, possibly using the group's customization. <code>custom</code> if the title contribution should be computed automatically, but customized. <code>manual</code> if the manual title contribution should be used (the <code>title</code> property). <code>none</code> if no title contribution should be used.
typeintitle	<code>on</code> <code>off</code>	<code>on</code>	Whether the title contribution should contain the type when <code>titletype</code> is <code>custom</code> .
unit	String	Model-dependent	The unit to use for the expression in <code>expr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	<code>on</code> <code>off</code>	<code>on</code>	Whether the title contribution should contain the unit when <code>titletype</code> is <code>custom</code> .
xdata	String	Empty	Data for the histogram's x-axis.
ydata	String	Empty	Data for the histogram's y-axis.

ATTRIBUTES[Height](#)**SEE ALSO**[Histogram](#)*MaxMinVolume, MaxMinSurface, MaxMinLine*

Create max/min marker plots.

SYNTAX

```

model.result(<pgtag>).create(<ftag>,"MaxMinVolume");
model.result(<pgtag>).create(<ftag>,"MaxMinSurface");
model.result(<pgtag>).create(<ftag>,"MaxMinLine");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();

```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"MaxMinVolume")` creates a max/min marker plot feature named <ftag> belonging to the plot group <pgtag>.

This plot type displays the maximum and minimum of an expression and the points there they are attained.

The following properties are available:

TABLE 7-48: VALID PROPERTY/VALUE PAIRS FOR MAX/MIN PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	black	The color with which markers are plotted.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customcolor	RGB-triplet	{1,0,0} or last used color	The color to use when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
display	minmax min max	minmax	The selection determines which markers are shown.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String		The expression to compute the maximum and minimum for
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that deformation scale is inherited from.
precision	integer > 0	6	The number of decimals displayed in the labels in the GUI.
prefixintitle	String	Empty	Added prefix to contribution to title.
refine	integer > 0	2	The number of refinements of each mesh element when computing the maximum and minimum.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).

TABLE 7-48: VALID PROPERTY/VALUE PAIRS FOR MAX/MIN PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
suffixintitle	String	Empty	Added suffix to contribution to title.
title	String	The auto-title	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

ATTRIBUTES[Deform](#)*MaxVolume, MaxSurface, MaxLine, MinVolume, MinSurface, MinLine*

Find extremal values (maximum or minimum).

SYNTAX

```

model.result().numerical().create(<ftag>,"MaxVolume");
model.result().numerical().create(<ftag>,"MinVolume");
model.result().numerical().create(<ftag>,"MaxSurface");
model.result().numerical().create(<ftag>,"MinSurface");
model.result().numerical().create(<ftag>,"MaxLine");
model.result().numerical().create(<ftag>,"MinLine");
model.result().numerical(<ftag>).selection(...);
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();

```

DESCRIPTION

model.result().numerical().create(<ftag>,"MaxVolume") creates a volume maximum feature with the name <ftag>, and similarly for "MinVolume".

model.result().numerical().create(<ftag>,"MaxSurface") creates a surface maximum feature with the name <ftag>, and similarly for "MinSurface".

`model.result().numerical().create(<ftag>, "MaxLine")` creates a line maximum feature with the name `<ftag>`, and similarly for "MinLine".

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that one row contains data for all solution numbers. This is identical to `(columnwise)` when `columnwise` is false. If `columnwise` is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(allocate, columnwise)` returns the imaginary part of complex result, recomputing the feature if necessary. If `allocate` is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `allocate` true and `columnwise` false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-49: VALID PROPERTY/VALUE PAIRS FOR MAXIMUM AND MINIMUM

NAME	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
data	none data set name	First compatible data set	The data set this feature refers to.
dataseries	average integral maximum minimum none rms stddev variance	none	The operation that is applied to the data series formed by the evaluation.
dataseriesmethod	auto integration summation	auto	The method to use for the data series: automatic, integration, or summation.
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent	The expression to plot.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.

TABLE 7-49: VALID PROPERTY/VALUE PAIRS FOR MAXIMUM AND MINIMUM

NAME	VALUE	DEFAULT	DESCRIPTION
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
obj	abs real	real	The function of each real or complex number that is maximized/minimized.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
refine	nonnegative integer	1	The element refinement to use. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
solnum	nonnegative integer array	All solutions	The solutions to use.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.

TABLE 7-49: VALID PROPERTY/VALUE PAIRS FOR MAXIMUM AND MINIMUM

NAME	VALUE	DEFAULT	DESCRIPTION
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling setResult() or appendResult(). new indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling setResult() or appendResult(). Applicable only for parametric sweep models. The level values (level1, level2, and so on) are the levels in the parametric sweep.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.

SEE ALSO

[IntVolume](#), [IntSurface](#), [IntLine](#)

Mesh

Create a mesh plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Mesh");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Mesh") creates a mesh plot feature named <ftag> belonging to the plot group <pgtag>.

This plot type provides visualization of meshes.

The following properties are available:

TABLE 7-50: VALID PROPERTY/VALUE PAIRS FOR MESH PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
colortable	color table name	Rainbow	The element color table to use when elemcolor is set to quality. See Color Tables for a list of color tables. 2D and 3D only.
data	none parent data set name	parent	The data set this feature refers to
elemcolor	quality color	quality	How to color the elements 2D and 3D only.

TABLE 7-50: VALID PROPERTY/VALUE PAIRS FOR MESH PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
elemfilter	random quality qualityrev size expression logicalexpression	random	If filteractive is on: The expression to use for filtering when only a subset of the elements are shown. 2D and 3D only.
elemscale	double in [0,1]	1	The factor with which the elements are scaled before display. 2D and 3D only.
elemtype2	all tri quad	all	2D mesh element types to plot
elemtype3	all tet prism hex	all	3D mesh element types to plot
filteractive	on off	off	Whether to use element filtering
filterexpr	String	x	The expression to use for filtering when elemfilter is set to expression. 2D and 3D only.
logfilterexpr	String	'1'	The logical expression to use for filtering when elemfilter is set to 'logicalexpression'. 2D and 3D only.
meshdomain	all point edge surface volume (if 3D)	all	Mesh domain level(s) to plot. 2D and 3D only.
tetkeep	Double in [0, 1]	1	The fraction of the elements to display. 2D and 3D only.
title	String	The auto-title	The title to use when titletype is manual. 2D and 3D only.
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title should be used (the title property). none if no title should be displayed. 2D and 3D only.
wireframecolor	none color	black	How to color the wireframe mesh; 'none' means that it is not displayed at all. 2D and 3D only.
1dfilter	size none	none	Plot the mesh size as the y-axis data. 1D only.

ATTRIBUTES[Filter](#)**SEE ALSO**[Mesh \(data set\)](#)*Mesh (data set)*

Create a mesh data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Mesh");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>, "Mesh")` creates a mesh data set feature named <dtag>.

This data set provides support for evaluation of spatial coordinates and mesh variables on a mesh.

The following properties are available:

TABLE 7-51: VALID PROPERTY/VALUE PAIRS FOR MESH DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
mesh	String	Empty	The mesh that this data set refers to.

SEE ALSO

[Mesh](#)

Mesh (export)

Export a mesh.

SYNTAX

```
model.result().export().create(<ftag>, "Mesh");  
model.result().export().create(<ftag>, <dtag>, "Mesh");  
model.result().export(<ftag>).set(property, <value>);  
model.result().export(<ftag>).run();
```

DESCRIPTION

`model.result().export().create(<ftag>, "Mesh")` creates a mesh export feature with the name <ftag>.

`model.result().export().create(<ftag>, <dtag>, "Mesh")` creates a mesh export feature with the name <ftag> for the data set <dtag>.

The following properties are available:

TABLE 7-52: VALID PROPERTY/VALUE PAIRS FOR MESH EXPORT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
alwaysask	on off	off	Always ask for filename when saving. This property is ignored when running without a GUI.
data	String		The name of the data set to export.
filename	String		The output file.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
solnum	integer array		Solution number to take values from.
stlformat	binary text	binary	STL file format.
t	String		Time to evaluate in if timeinterp is on.
timeinterp	on off	off	Enable/disable explicit time to evaluate in.

SEE ALSO

[Data](#)

Mirror2D, Mirror3D

Create 2D and 3D mirror data sets.

SYNTAX

```
model.result().dataset().create(<dtag>,"Mirror2D");  
model.result().dataset().create(<dtag>,"Mirror3D");  
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Mirror2D")` creates a 2D mirror data set feature named `<dtag>`.

`model.result().dataset().create(<dtag>,"Mirror3D")` creates a 3D mirror data set feature named `<dtag>`.

This data set takes data from another data set and adds a mirror copy with respect to an axis or plane of reflection (for 2D and 3D, respectively).

The following properties are available for Mirror 2D and Mirror 3D:

TABLE 7-53: VALID PROPERTY/VALUE PAIRS FOR MIRROR DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to
genpoints	double matrix	{{0, 0, 0}, {1,0,0}}	Active when method equals twopoint, this property contains the coordinates of the two points in the two rows of the matrix.
hasvar	Boolean	false	If true, space and axis indicator variables are defined
method	twopoint pointdir	twopoint	Decides if the line should be specified by two points or through one point and a direction
pddir	double array	{0, 1}	Active when method equals pointdir, this property contains the direction.
pdpoint	double array	{0, 0}	Active when method equals pointdir, this property contains the coordinates of the point.
sidevar	String		If hasvar is true: The name of the positive side variable, which is 1 on the side where the original data resides and 0 on the other side.
spacevars	String array	Depends on the feature's tag	If hasvar is true: The name of space variables, which evaluate to the coordinates after the transformation performed by the mirror data set.

The following properties are available for Mirror 3D:

TABLE 7-54: VALID PROPERTY/VALUE PAIRS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to
genmethod	threepoint pointnormal	threepoint	Active when planetype equals general, this property indicates whether the plane should be specified by three points or through one point and a normal.
genpoints	double matrix	{{0, 0, 0}, {1,0,0}, {0,1,0}}	Active when method equals threepoint, this property contains the coordinates of the three points in the rows of the matrix.

TABLE 7-54: VALID PROPERTY/VALUE PAIRS

NAME	VALUE	DEFAULT	DESCRIPTION
genpnpoint	double array of length three	Zero vector	Active when genmethod equals pointnormal, this property contains the coordinates of the point.
genpnvec	double array of length three	Zero vector	Active when genmethod equals pointnormal, this property contains the normal vector.
hasvar	Boolean	false	If true, space and axis indicator variables are defined
planetype	quick general	yz	Specify plane type
quickplane	xy yz zx	yz	Specify quick plane type. Active when planetype is quick.
quickx	double	0	x-coordinate if planetype is yz
quicky	double	0	y-coordinate if planetype is zx
quickz	double	0	z-coordinate if planetype is xy
sidevar	String		If hasvar is true: The name of the positive side variable, which is 1 on the side where the original data resides and 0 on the other side.
spacevars	String array	Depends on the feature's tag	If hasvar is true: The names of space variables, which evaluate to the coordinates after the transformation performed by the mirror data set.

Multislice

Create a slice plot in multiple directions at once.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Multislice");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Multislice") creates a slice feature in multiple directions named <ftag> belonging to the plot group <pgtag>.

The following properties are available:

TABLE 7-55: VALID PROPERTY/VALUE PAIRS FOR MULTISLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color the slices
colorlegend	on off	on	Whether to show color legend when coloring is set to colortable
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.

TABLE 7-55: VALID PROPERTY/VALUE PAIRS FOR MULTISLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.

TABLE 7-55: VALID PROPERTY/VALUE PAIRS FOR MULTISLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
multiplanexmethod	number coord	number	Indicates whether the x-coordinates of the x-planes should be specified by number of planes to distribute evenly across the data or by coordinates.
multiplaneymethod	number coord	number	Indicates whether the y-coordinates of the y-planes should be specified by number of planes to distribute evenly across the data or by coordinates.
multiplanezmethod	number coord	number	Indicates whether the z-coordinates of the z-planes should be specified by number of planes to distribute evenly across the data or by coordinates.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.

TABLE 7-55: VALID PROPERTY/VALUE PAIRS FOR MULTISLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
smooth	none internal everywhere	internal	Smoothing settings
suffixintitle	String	Empty	Added suffix to contribution to title
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom
xnumber	non-negative integer	1	Number of planes in the x-direction. Active when multiplanexmethod is set to number.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when multiplanexmethod is set to coord

TABLE 7-55: VALID PROPERTY/VALUE PAIRS FOR MULTISLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
ynumber	integer	1	Number of planes in the y-direction. Active when multiplanemethod is set to number
ycoord	double array	Empty	Absolute coordinates in the z-direction, active when multiplanemethod is set to coord
znumber	integer	1	Number of planes in the z-direction. Active when multiplanezmethod is set to number
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when multiplanezmethod is set to coord.

ATTRIBUTES

[Deform](#), [Filter](#)

SEE ALSO

[Slice](#), [Volume](#)

Nyquist

Create a Nyquist plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Nyquist");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Nyquist")` creates a Nyquist plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Nyquist plots are used to visualize complex-valued variables. Nyquist plots can be added to 1D plot groups.

The following properties are available:

TABLE 7-56: VALID PROPERTY/VALUE PAIRS FOR NYQUIST PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line.
autounit	on off	off	Whether the automatic legends should include the unit.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.

TABLE 7-56: VALID PROPERTY/VALUE PAIRS FOR NYQUIST PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
legend	on off	off	Whether to show legends.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.

TABLE 7-56: VALID PROPERTY/VALUE PAIRS FOR NYQUIST PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title
solnum	non-negative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String array	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitcircle	Boolean	false	Whether to plot the unit circle along with the axis.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.

OctaveBand

Create an Octave Band Plot.



The Octave Band Plot requires a license for the Acoustics Module.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"OctaveBand");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"OctaveBand")` creates an octave band plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

The octave band plots a weighted frequency response in bands (octave or 1/3 octave) or as a continuous frequency response. The frequency can be weighted.

The following properties are available:

TABLE 7-57: VALID PROPERTY/VALUE PAIRS FOR OCTAVE BAND PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
amplref	double	1	Reference level for the amplitude expression type.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{0,0,1} or last used edgcolor.	The color to use for the lines. Active when <code>linecolor</code> is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent.	The description of the expression in <code>expr</code> . Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when <code>titletype</code> is custom.
expr	String	Model-dependent.	The expression used to transform the input to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when <code>titletype</code> is custom.
exprtype	amplitude power transfer	amplitude	Type of expression: amplitude, power, or transfer function.
inband	on off	on	Whether to use in-band data only.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. <code>manual</code> indicates that <code>solnum</code> is used. <code>manualindices</code> indicates that <code>solnumindices</code> is used. <code>interp</code> indicates that <code>t</code> is used.
legend	on off	off	Whether to show legends.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the <code>legends</code> property.
legends	String array	The last computed automatic legends.	Manual legends active when <code>legendmethod</code> is set to manual.
levelref	double	0	Reference level for the transfer function expression type.

TABLE 7-57: VALID PROPERTY/VALUE PAIRS FOR OCTAVE BAND PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
powerref	double	1	Reference level for the power expression type.

TABLE 7-57: VALID PROPERTY/VALUE PAIRS FOR OCTAVE BAND PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
prefixintitle	String	Empty	Added prefix to contribution to title
sdim	fromdataset 0 1 2 3	fromdataset	The space dimension to evaluate the underlying data set in
solnum	integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
style	continuous octave octave3	octave	Octave band style: continuous, octave bands, or 1/3 octave bands.
suffixintitle	String	Empty	Added suffix to contribution to title
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
type	curve solid	curve	Use a filled histogram (solid) or a histogram drawn using lines (curve). This property is only available when function is discrete.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
weightexpr	expression		Value or expression to use as weighting.
weighting	a c z expression	z	Weighting function for octave band plot: A-weighting, C-weighting, Z-weighting, or a user-defined expression.

Parametric1D, Parametric2D

Extend a 1D or 2D data set by using a parameter as dimension.

SYNTAX

```
model.result().dataset().create(<dtag>,"Parametric1D");
model.result().dataset().create(<dtag>,"Parametric2D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"Parametric2D") creates a parametric data set feature named <dtag>. This data set extends a data set by using a parameter, such as time, as a dimension.

The following properties are available:

TABLE 7-58: VALID PROPERTY/VALUE PAIRS FOR THE PARAMETRIC1D AND PARAMETRIC2D DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
levelscale	double	1	The scaling factor applied to the levels if levelscaleactive is true.
levelscaleactive	Boolean	false	If true, levelscale is used to scale the levels; otherwise the scale factor is computed automatically.
leveltrans	none expression	none	Use an expression in terms of the level variable to define a level transformation.
solnum	Integer array	All solutions	The solutions to use for extrapolation.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
t	double array	Empty	The times to use as intermediate layers in the extrusion.
transexpr	String	level	Level transformation expression (as a function of level; the default value means no transformation).

ParCurve2D, ParCurve3D

Create a 2D or 3D parameterized curve data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"ParCurve2D");
model.result().dataset().create(<dtag>,"ParCurve3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"ParCurve2D") creates a 2D parameterized curve data set feature named <dtag>.

model.result().dataset().create(<dtag>,"ParCurve3D") creates a 3D parameterized curve data set feature named <dtag>.

Evaluation is made along an arbitrary parameterized curve in 2D or 3D.

The following properties are available: Add option for evaluating outside the mesh:

TABLE 7-59: VALID PROPERTY/VALUE PAIRS FOR PARAMETERIZED CURVE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
name	String	none	The name of this feature
data	none data set name	First compatible data set	The data set this feature refers to

TABLE 7-59: VALID PROPERTY/VALUE PAIRS FOR PARAMETERIZED CURVE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
global	boolean	false	If set to true, the data set only evaluates globally defined expressions
par1	String	t	The curve parameter
parmin1	double	0	The minimum value for par
parmax1	double	1	The maximum value for par
exprx	String	0	The expression for x(par)
expry	String	0	The expression for y(par)
exprz	String	0	The expression for z(par)
res	integer	1000	Resolution (number of discretization points along the curve)

The `global` property can be useful for BEM models, for example, to be able to evaluate globally defined expressions outside the mesh.

SEE ALSO

[ParSurface](#)

ParSurface

Create a parameterized surface data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"ParSurface");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"ParSurface")` creates a parameterized surface data set feature named `<dtag>`.

Evaluation is made along an arbitrary parameterized surface in 3D.

The following properties are available:

TABLE 7-60: VALID PROPERTY/VALUE PAIRS FOR A PARAMETERIZED CURVE DATA SET

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to
exprx	String	0	The expression for x(par1, par2)
expry	String	0	The expression for y(par1, par2)
exprz	String	0	The expression for z(par1, par2)
global	boolean	false	If set to true, the data set only evaluates globally defined expressions
par1	String	s	The first surface parameter
par2	String	t	The second surface parameter
parmin1	double	0	The minimum value for par1.
parmax1	double	1	The maximum value for par1
parmin2	double	0	The minimum value for par2
parmax2	double	1	The maximum value for par2
res	integer	200	Resolution (the number of discretization points for each parameter)

The `global` property can be useful for BEM models and for far-field postprocessing, for example, to be able to evaluate globally defined expressions outside the mesh.

SEE ALSO

[ParCurve2D](#), [ParCurve3D](#)

Particle

Create a massless particle tracing plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Particle");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Particle")` creates a massless particle tracing plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

With massless particle tracing, you can visualize path lines, that is, trajectories of particles released in a flow field, which can be time-dependent or static. For time-dependent flows you can also use a snapshot in time of the flow field as a static field. The motion of the particles does not affect the flow field. Particle tracing is available in 2D and 3D plot groups.

The following properties are available:

TABLE 7-61: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the particle position.
arrowexpr	string array		Active when pointtype is set to arrow. Determines the direction in which the arrow points.
arrowlength	logarithmic normalized proportional	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowscale	positive double	1	If arrowscalefactor is true: the length scale factor
arrowscaleactive	boolean	false	If true, arrowscale is used; otherwise the scale factor is computed automatically.
arrowtype	Arrow cone	Arrow	The type of arrow to draw.
atol	manual automatic	automatic	Whether to specify a manual absolute tolerance for the position. automatic indicates that the absolute tolerance for the position is the mean of the lengths of the bounding box of the geometry multiplied by the relative tolerance.
atolpos	positive double	0.001	Absolute tolerance for the position. Active when atol is set to manual.
bndcoord	double array	0	Vector of values from 0 to 1 to describe the start points. Active when bndmethod is set to coord. Available in 2D only.

TABLE 7-61: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
bndmethod	number coord	number	Active when posmethod is set to bnd. Available in 2D only.
bndnumber	positive integer	10	The number of equidistant start points along the selected boundaries. Available in 2D only.
bndselection	Reference to a named selection on boundaries	First available named selection	The boundaries to start from. Available in 2D only.
comettailexpr	String array		Active when pointtype is set to cometail. Determines the direction in which the comet tail points.
const	String array of property/value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use for lines. Active when linecolor is set to custom.
custompointcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use for points. Active when pointcolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String	Model-dependent	The description of the expressions in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom
dropfreq	positive double	1	Particles are released at these intervals when dropmethod is set to freq.
dropmethod	once freq times	once	Method to use when releasing particles.
droptimes	double array	1	The specific times when to release particles. Used when dropmethod is set to times.
edgetol	positive double	0.001	The absolute tolerance controlling how close to the geometry boundary the path lines are cut when they exit the geometry. A lower value cuts the line closer to the geometry boundary.
expr	String array of length 2 in 2D and 3 in 3D	Model-dependent	Expressions for the components to plot.
hmax	double	0.1	The maximum time step. Active when stepsize is set to manual.
hstart	double	0.1	The initial time step. Active when stepsize is set to manual.
inheritarrowsscale	Boolean	true	If inheritplot is not none: Determines if the arrow scale is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that sphere scale, tube scale, and deformation scale is inherited from.

TABLE 7-61: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the sphere scale is inherited.
inherit tubescale	Boolean	true	If inheritplot is not none and linetype is tube: Determines if the tube scale is inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
linecolor	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use for lines.
linetype	none line tube	line	Plot particle traces as lines or tubes, or not at all.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
maxsteps	positive integer	1000	The maximum number of steps to use in the particle simulation. Active when maxstepsactive is set to on.
maxstepsactive	on off	off	Whether to manually specify the maximum number of steps used in particle simulation. off indicates that the limit varies in this way: for static flow fields, the algorithm uses the value 1000; for time-dependent flows, there is no upper limit of the number of steps, and the particle simulation goes on until it reaches the end time.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points.
pointdom	disappear stick	stick	stick plots the points on the boundary at the exit point, and disappear does not render these points at all.
pointlineanim	on off	off	When animating particle tracing plots with points, this property indicates whether the points at the current time should appear on top of the lines.

TABLE 7-61: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
pointautoscale	on off	on	If enabled, the point radii are scaled so that the maximum radius is 0.01 of the plot scale.
pointradiusexpr	String	1	The point radius expression.
pointtype	none point comettail	none	Selects between plotting the path lines' endpoints as points, comet tails, or not at all.
posmethod	start bnd	start	The type of particle tracing positioning. Choose specific start points or a boundary to start from. Available in 2D only.
prefixintitle	String	Empty	Added prefix to contribution to title.
radiusexpr	String	1	The tube radius.
resolution	coarser coarse normal fine finer extrafine		Affects the number of output points by adding points between each time step taken in the ODE solver, using a 4th-order interpolation, to produce a smoother output.
rtol	positive double	0.001	The relative error tolerance that the ODE solver uses.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
static	on off	off	Only affects time-dependent problems. When set to on, this property freezes the time and considers this a static flow field.
statictend	double	1	The maximum time at which to end the particle-tracing simulation for static flow fields. Active when statictendactive is set to on.
statictendactive	on off	off	Whether to specify the end time for static flow fields automatically. (For time-dependent flows, the automatic end time is the last time that the time-dependent solver returns.) When set to off, there is no upper limit of the time. In this case, the particle simulation goes on until all particles exit the geometry or the simulation reaches the maximum number of steps.
stepsize	automatic manual	automatic	Whether to specify initial (hstart) and maximum (hmax) time step manually.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
tailscale	positive double	1	Active when pointtype is set to comettail and tailscaleactive is on. Specifies the manual scale factor with which the comet tail expression is multiplied.

TABLE 7-61: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
tailscaleactive	on off	off	Active when pointtype is set to comettail. Specifies whether manual tail scaling is enabled.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
tstart	double	0	Manual start time. Active when tstartactive is set to on.
tstartactive	on off	off	on indicates a manual start time (set in the tstart property). Off indicates that the start time becomes either the first time value that the time-dependent solver returns or 0 for stationary flows. Available when dropmethod is set to once or freq.
tubeautoscale	on off	on	If enabled, the tube radii are scaled so that the maximum tube radius is 0.02 of the plot scale.
tvar	String	partt	The name of the variable for time. Normally there is no need to change the default name. You can use this names in expressions as well as for the color when coloring the path lines according to an expression.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when posmethod is set to start.
ycoord	double array	Empty	Absolute coordinates in the y-direction, active when postmethod is set to start.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when posmethod is set to coord. Available in 3D only.

ATTRIBUTES

[Color](#) (applies to lines), [Deform](#)

SEE ALSO

[ParticleMass](#), [Streamline](#)

Particle (1D Plot)

Create a particle plot, which plots particle properties over time or compares particle properties against each other at a set of time steps.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Particle1D");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Particle1D")` creates a particle plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

With the particle plot, you can plot properties of all particles in a data set versus time, or plot two particle properties against each other at a set of selected times. When plotting particle properties versus time, it is possible to apply data series operations to the particle data. The particle plot is available in 1D plot groups.

The following properties are available:

TABLE 7-62: VALID PROPERTY/VALUE PAIRS FOR PARTICLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line.
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
dataseries	none average sum maximum minimum rms stddev variance	sum	The data series operation that is applied to all particles. Effective only when xdata is not expr.
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.

TABLE 7-62: VALID PROPERTY/VALUE PAIRS FOR PARTICLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the color is different for each line.
linewidth	double	0.5	The line width
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.

TABLE 7-62: VALID PROPERTY/VALUE PAIRS FOR PARTICLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title.
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
xdata	expr solution	solution	x-axis data. expr uses the expression in xdataexpr. solution uses the available solutions in the underlying data set, such as time steps.
xdataexpr	String	Model-dependent	Expression for x-axis data

TABLE 7-62: VALID PROPERTY/VALUE PAIRS FOR PARTICLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
xdatadescr	String	Model-dependent	Description of expression in <code>xdataexpr</code>
xdataphaseunit	String	rad	The unit in which <code>xdataphaserange</code> is described
xdatasolnumtype	all inner outer valid level	outer	Whether the expression should be evaluated for every inner or every outer solution, or for a specific level (<code>level1</code> , <code>level2</code> , and so on). Applicable only for models containing multiple levels.
xdataunit	String	Model-dependent	The unit to use for the expression in <code>xdataexpr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.

ATTRIBUTES

[Color](#) (applies to lines), [Filter \(Particle Tracing\)](#).

SEE ALSO

[Particle \(Evaluation\)](#), [ParticleTrajectories](#).

Particle (data set)

Create a particle data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Particle");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Particle")` creates a particle data set.

A particle data set is required to plot particle trajectories. The particle data set identifies the geometry in which the particle data is stored and the degrees of freedom which determine the position of each particle.

The following properties are available:

TABLE 7-63: VALID PROPERTY/VALUE PAIRS FOR PARTICLE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
solution	String	first available solution	The solution this data set refers to.
pgeomspec	manual fromphysics	manual	The method of specifying the particle geometry and the names of the particle position degrees of freedom.
physicsinterface	none or the tag of a valid physics interface	none	The physics interface from which the particle geometry and the names of the particle position degrees of freedom are defined.

TABLE 7-63: VALID PROPERTY/VALUE PAIRS FOR PARTICLE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
pgeom	String	pgeom	The geometry in which the particle degrees of freedom are defined. The correct name of this particle geometry is pgeom_<tag>, where <tag> is the name for the Particle Tracing interface node.
posdof	String array	mod1.qx, mod1.qy	The position degrees of freedom of the particles.

SEE ALSO

[Particle \(Evaluation\)](#)

Particle (Evaluation)

Particle evaluations to evaluate quantities on particle trajectories.

SYNTAX

```

model.result().numerical().create(<ftag>,"Particle");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>,<outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>,<outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();

```

DESCRIPTION

`model.result().numerical().create(<ftag>,"Particle")` creates a particle evaluation feature with the name <ftag>. Particle evaluation can be performed on trajectories accessed through a particle data set.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that there is one row per point, with one row containing data for all solution numbers. This is identical to (<columnwise>) when <columnwise> is false. If <columnwise> is true, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If <allocate> is true, a zero-valued matrix is allocated even when the result is real. `getImag()` uses <allocate> true and <columnwise> false.

`model.result().numerical(<ftag>).isComplex()` returns true if the result is complex.

<outersolnum> is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-64: VALID PROPERTY/VALUE PAIRS FOR PARTICLE EVALUATIONS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First particle compatible data set	The particle data set this feature refers to
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
evaluate	all fraction number	al	What particles to evaluate
expr	String	Model-dependent	The expression to plot
fraction	double	1	If evaluate is fraction: The fraction of particles to evaluate.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
number	double	100	If evaluate is number: The number of particles to evaluate.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.

TABLE 7-64: VALID PROPERTY/VALUE PAIRS FOR PARTICLE EVALUATIONS

NAME	VALUE	DEFAULT	DESCRIPTION
solnum	non-negative integer array	All solutions	The solutions to use
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>innerinput</code> is <code>manualindices</code> .
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
t	double array	Empty	The times to use. Available when the underlying solution is transient.
table	new table name	new	The table to use when calling <code>setResult()</code> or <code>appendResult()</code> . <code>new</code> indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling <code>setResult()</code> or <code>appendResult()</code> . Applicable only for parametric sweep models. The level values (<code>level1</code> , <code>level2</code> , and so on) are the levels in the parametric sweep.
unit	String	Model-dependent	The unit to use for the expression in <code>expr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.

ParticleMass

Create a particle tracing plot with mass.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"ParticleMass");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"ParticleMass")` creates a particle tracing plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

With particle tracing, you can visualize path lines (that is, trajectories of particles released in a flow field), which can be time dependent or static. For time-dependent flows you can also use a snapshot in time of the flow field as a static field. The motion of the particles does not affect the flow field. Particle tracing is available in 2D and 3D plot groups.

The following properties are available:

TABLE 7-65: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS WITH MASS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the particle position.
arrowexpr	string array		Active when pointtype is set to arrow. Determines the direction in which the arrow points.
arrowlength	logarithmic normalized proportional	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowscale	positive double	1	If arrowscalefactor is true: the length scale factor.
arrowscaleactive	boolean	false	If true, arrowscale is used; otherwise the scale factor is computed automatically.
arrowtype	Arrow cone	Arrow	The type of arrow to draw
atol	manual automatic	automatic	Whether to specify a manual absolute tolerance for the position and velocity. automatic indicates that the absolute tolerance for the position is the mean of the lengths of the bounding box of the geometry multiplied by the relative tolerance.
atolpos	positive double	0.001	Absolute tolerance for the position. Active when atol is set to manual.
atolvel	positive double	0.001	Absolute tolerance for the velocity. Active when atol is set to manual.
bndcoord	double array	0	Vector of values from 0 to 1 to describe the start points. Active when bndmethod is set to coord. Available in 2D only.
bndmethod	number coord	number	Active when posmethod is set to bnd. Available in 2D only.
bndnumber	positive integer	10	The number of equidistant start points along the selected boundaries. Available in 2D only.
bndselection	Reference to a named selection on boundaries	First available named selection	The boundaries to start from. Available in 2D only.
comettailexpr	String array		Active when pointtype is set to comettail. Determines the direction in which the comet tail points.
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customlinecolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use for lines. Active when linecolor is set to custom.
custompointcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use for points. Active when pointcolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to

TABLE 7-65: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS WITH MASS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
descr	String	Model-dependent	Description of forces to plot
dropfreq	positive double	1	Particles are released at these intervals when dropmethod is set to freq.
dropmethod	once freq times	once	Method to use when releasing particles.
droptimes	double array	1	The specific times when to release particles. Used when dropmethod is set to times.
edgetol	positive double	0.001	The absolute tolerance controlling how close to the geometry boundary the path lines are cut when they exit the geometry. A lower value cuts the line closer to the geometry boundary.
fx	String	Mode-dependent	x-component for the force expression
fy	String	Mode-dependent	y-component for the force expression
fz	String	Model-dependent	z-component for the force expression. In 2D, this property is only used if the underlying geometry is axisymmetric.
hmax	double	0.1	The maximum time step. Active when stepsize is set to manual.
hstart	double	0.1	The initial time step. Active when stepsize is set to manual.
inheritarrowsscale	Boolean	true	If inheritplot is not none: Determines if the arrow scale is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that sphere scale, tube scale, and deformation scale is inherited from.
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the sphere scale is inherited.
inherit tubescale	Boolean	true	If inheritplot is not none and linetype is tube: Determines if the tube scale is inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
linecolor	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use for lines
linetype	none line tube	line	Plot particle tracing lines as lines or tubes, or not at all.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.

TABLE 7-65: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS WITH MASS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
mass	String	1	The particle mass
maxsteps	positive integer	1000	The maximum number of steps to use in the particle simulation. Active when maxstepsactive is set to on.
maxstepsactive	on off	off	Whether to manually specify the maximum number of steps used in particle simulation. off indicates that the limit varies in this way: for static flow fields, the algorithm uses the value 1000; for time-dependent flows, there is no upper limit of the number of steps, and the particle simulation goes on until it reaches the end time.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points
pointdom	disappear stick	stick	stick plots the points on the boundary at the exit point, and disappear does not render these points at all.
pointlineanim	on off	off	When animating particle tracing plots with points, this property indicates whether the points at the current time should appear on top of the lines.
pointtype	none point comettail	none	Selects between plotting the path lines' endpoints as points, comet tails, or not at all.
posmethod	start bnd (in 2D)	start	The type of particle tracing positioning. Choose specific start points or a boundary to start from. Available in 2D only.
radiusexpr	String	1	The tube radius
pointautoscale	on off	on	If enabled, the point radii are scaled so that the maximum radius is 0.01 of the plot scale.
pointradiusexpr	String	1	The point radius expression.
resolution	coarser coarse normal fine finer extrafine		Affects the number of output points by adding points between each time step taken in the ODE solver, using a 4th-order interpolation, to produce a smoother output
rtol	positive double	0.001	The relative error tolerance that the ODE solver uses

TABLE 7-65: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS WITH MASS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
static	on off	off	Only affects time-dependent problems. When set to on, this property freezes the time and considers this a static flow field.
statictend	double	1	The maximum time at which to end the particle-tracing simulation for static flow fields. Active when statictendactive is set to on.
statictendactive	on off	off	Whether to specify the end time for static flow fields automatically. (For time-dependent flows, the automatic end time is the last time that the time-dependent solver returns.) When set to off, there is no upper limit of the time. In this case, the particle simulation goes on until all particles exit the geometry or the simulation reaches the maximum number of steps.
stepsize	automatic manual	automatic	Whether to specify initial (hstart) and maximum (hmax) time step manually.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
tailscale	positive double	1	Active when pointtype is set to comettail and tailscaleactive is on. Specifies the manual scale factor with which the comet tail expression is multiplied.
tailscaleactive	on off	off	Active when pointtype is set to comettail. Specifies whether manual tail scaling is enabled.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title should be used (the title property). none if no title should be displayed.
tstart	double	0	Manual start time. Active when tstartactive is set to on.
tubeautoscale	on off	on	If enabled, the tube radii are scaled so that the maximum tube radius is 0.02 of the plot scale.

TABLE 7-65: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRACING PLOTS WITH MASS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
tstartactive	on off	off	On indicates a manual start time (set in the tstart property). Off indicates that the start time becomes either the first time value that the time-dependent solver returns or 0 for stationary flows. Available when dropmethod is set to once or freq.
tvar	String	partt	The name of the variable for time. Normally there is no need to change the default name. You can use this names in expressions as well as for the color when coloring the path lines according to an expression.
velvarx	String	partu	The name of the variable for the x-component of the velocity
velvary	String	partv	The name of the variable for the y-component of the velocity
velvarz	String	partw	The name of the variable for the z-component of the velocity. In 2D, this property is only used if the underlying geometry is axisymmetric.
velstartx	String	0	x-component for initial velocity
velstarty	String	0	y-component for initial velocity
velstartz	String	0	z-component for initial velocity. In 2D, this property is only used if the underlying geometry is axisymmetric.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when posmethod is set to start.
ycoord	double array	Empty	Absolute coordinates in the y-direction, active when posmethod is set to start.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when posmethod is set to coord. Available in 3D only.

ATTRIBUTES

[Color](#) (applies to lines), [Deform](#)

SEE ALSO

[Particle](#), [PointTrajectories](#), [Streamline](#)

ParticleTrajectories

Create a particle trajectories plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"ParticleTrajectories");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"ParticleTrajectories") creates a particle trajectories plot feature named <ftag> belonging to plot group <pgtag>.

A particle data set is required to plot particle trajectories. The particle data set identifies the geometry in which the particle data is stored and the degrees of freedom which determine the position of each particle.

The following properties are available:

TABLE 7-66: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the particle position.
arrowexpr	string array		Active when pointtype is set to arrow. Determines the direction in which the arrow points.
arrowlength	logarithmic normalized proportional	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowscale	positive double	1	If arrowscaleactive is true: the length scale factor.
arrowscaleactive	boolean	false	If true, arrowscale is used, otherwise the scale factor is computed automatically.
arrowtype	arrow cone	arrow	The type of arrow to draw.
comettailexpr	String array	inverse of velocity vector of the first available particle tracing interface	Active when pointtype is set to comettail. Determines the direction in which the comet tail points and the relative sizes of the tails of different particles.
data	none parent data set name	parent	The data set this feature refers to.
ellipsearrowbase	head tail	head	The base for the arrow to draw for ellipses.
ellipsearrowtype	arrow cone	arrow	The type of arrow to draw for ellipses.
ellipsecount	positive integer	10	Maximum number of ellipses.
ellipsetimes	String array		Collection of times for polarization ellipses.
inheritarrowscale	Boolean	true	If inheritplot is not none: Determines if the arrow scale is inherited.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritellipsecale	Boolean	true	If inheritplot is not none: Determines if the ellipse scale factor is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale are inherited from.

TABLE 7-66: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the radius scale factor is inherited.
inherittailscale	Boolean	true	If inheritplot is not none: Determines if the tail scale factor is inherited.
inherit tubescale	Boolean	true	If inheritplot is not none: Determines if the tube radius scale factor is inherited.
interpolation	none uniform	none	The type of interpolation used to plot lines when linetype is line or tube.
linecolor	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use for lines
linetype	none line tube	line	Plot particle traces as lines or tubes, or not at all.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
pointautoscale	on off	on	If enabled, the point radii are scaled based on the size of the geometry.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points.
pointlineanim	on off	off	When animating particle tracing plots with points, this property indicates whether the points at the current time should appear on top of the lines.
pointradiusexpr	String	0.001	The point radius expression.
pointtype	none point comettail arrow ellipse	none	Plot particles as points, with comet tails, with arrows, with polarization ellipses, or not at all.
radiusexpr	String	1	The tube radius expression
semimajorexpr	String array		Expressions for the polarization ellipses' semi-major axis,
semiminorexpr	String array		Expressions for the polarization ellipses' semi-minor axis,
sphereradiusscale	positive double		The scale factor applied to the point radii if sphereradiusscaleactive is on.
sphereradiusscaleactive	on off	off	If on, sphereradiusscale is used; otherwise the scale factor is computed automatically.

TABLE 7-66: VALID PROPERTY/VALUE PAIRS FOR PARTICLE TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
tailscale	positive double	1	Active when pointtype is set to comettail and tailscaleactive is on. Specifies the manual scale factor with which the comet tail expression is multiplied.
tailscaleactive	on off	off	Active when pointtype is set to comettail. Specifies whether manual tail scaling is enabled.
title	String		Active when title is set to manual. Specifies the plot title.
titletype	automatic manual none	automatic	Determines how the plot title is specified.
tuberadiusscale	String	1	The scale factor applied to the tube radii if tuberadiusscaleactive is on.
tuberadiusscaleactive	on off	off	If on, tuberadiusscale is used, otherwise the scale factor is computed automatically

SEE ALSO

[Particle \(Evaluation\)](#)

ATTRIBUTES

[Color](#), [Filter \(Particle Tracing\)](#).

PhasePortrait

Create a phase portrait plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"PhasePortrait");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"PhasePortrait") creates a plot feature named <ftag> belonging to the plot group <pgtag>.

You can use phase portraits in 2D and 2D axisymmetric models to visualize large data sets of particle trajectories.

The following properties are available:

TABLE 7-67: VALID PROPERTY/VALUE PAIRS FOR PHASE PORTRAIT

NAME	VALUE	DEFAULT	DESCRIPTION
data	none parent data set name	parent	The data set this feature refers to.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited. Available in 2D and 3D.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.

TABLE 7-67: VALID PROPERTY/VALUE PAIRS FOR PHASE PORTRAIT

NAME	VALUE	DEFAULT	DESCRIPTION
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the radius scale factor is inherited.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	all on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points.
pointradiusexpr	String	1	The point radius expression.
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.

TABLE 7-67: VALID PROPERTY/VALUE PAIRS FOR PHASE PORTRAIT

NAME	VALUE	DEFAULT	DESCRIPTION
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
sphereradiusscale	positive double		The scale factor applied to the point radii if sphereradiusscaleactive is on.
sphereradiusscaleactive	Boolean	false	If true, sphereradiusscale is used; otherwise the scale factor is computed automatically.
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom
xdata	position manual	position	Whether to use the particle position or user-defined x-axis data.
xmanual	String	Empty	User-defined x-axis data used if xdata is set to manual.
ydata	position manual	position	Whether to use the particle position or user-defined y-axis data.
ymanual	String	Empty	User-defined y-axis data used if ydata is set to manual.

ATTRIBUTES*Color**Plot*

Export plots to files.

SYNTAX

```

model.result().export().create(<ftag>,"Plot");
model.result().export().create(<ftag>,<pgtag>,<plottag>,"Plot");
model.result().export(<ftag>).set(property,<value>);
model.result().export(<ftag>).run();

```

DESCRIPTION

model.result().export().create(<ftag>,"Plot") creates a plot export feature with the name <ftag>.

model.result().export().create(<ftag>,<pgtag>,<plottag>,"Plot") creates a plot export feature with the name <ftag> for the plot <plottag> in the plot group <pgtag>.

To export a 3D surface from a plot to an STL file, use `stl` as filename extension.

The following properties are available:

TABLE 7-68: VALID PROPERTY/VALUE PAIRS FOR PLOT EXPORTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
<code>alwaysask</code>	<code>on off</code>	<code>off</code>	Always ask for filename when saving. This property is ignored when running without a GUI.
<code>ifexists</code>	<code>append overwrite</code>	<code>overwrite</code>	If the file exists, append to or overwrite the file contents.
<code>plotgroup</code>	String		The name of the plot group containing the plot to export
<code>plot</code>	String		The name of the plot (in <code>plotgroup</code>) to export
<code>compact</code>	Boolean	<code>false</code>	Only applicable for streamline and particle tracing plots. If true, the exported data contains one line per streamline or particle rather than one line per point on a streamline or particle trajectory.
<code>filename</code>	String		The output file
<code>fullprec</code>	<code>on off</code>	<code>on</code>	If on, floating-point numbers are written in full precision, otherwise they are written with six significant digits.
<code>header</code>	<code>on off</code>	<code>off</code>	Enable/disable a data header in the output file.
<code>repeat</code>	<code>on off</code>	<code>off</code>	Enable/disable repeated playing of the animation.
<code>sort</code>	<code>on off</code>	<code>off</code>	Enable/disable sorting of the points with respect to the coordinates.
<code>struct</code>	<code>sectionwise spreadsheet</code>	<code>spreadsheet</code>	Format of the exported data
<code>stlformat</code>	<code>binary text</code>	<code>binary</code>	Format of the STL file

SEE ALSO

[HistogramHeight](#)

PlotGroup1D, PlotGroup2D, PlotGroup3D

Create a 1D, 2D, or 3D plot group.

SYNTAX

```
model.result().create(<pgtag>, dim);  
model.result(<pgtag>).set(property, <value>);  
model.result(<pgtag>).run();
```

DESCRIPTION

`model.result().create(<pgtag>, dim)` creates a plot group named `<pgtag>` of view dimension `dim`. A plot group is a group of plots that are shown together in a graphics or plot window.

1D plot groups contain graph plots.

The following properties are available for 1D plot groups:

TABLE 7-69: VALID PROPERTY/VALUE PAIRS FOR 1D PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
axisactive	on off	off	Use manual axis formatting settings.
axiscommonexp	on off	on	Use a common exponent for all axis values.
axisnotation	automatic engineering scientific	automatic	Use an automatic notation (decimal or scientific) for axis values or always use engineering or scientific notation.
axisprecision	positive integer	4	Precision for number format (number of digits).
axistrailingzeros	on off	off	Show trailing zeros for axis values.
data	none data set name	First compatible data set	The data set this feature refers to. This is the default data set for all plots in the group.
datasetintitle	on off	off	Whether the title should contain the data set when titletype is custom.
descriptionintitle	on off	on	Whether the title contribution of plots in this group should contain the description when titletype is custom.
expressionintitle	on off	off	Whether the title contribution of plots in this group should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels	The times to use, for transient levels. Available when data is not none and the underlying data is transient.
legendpos	upperright lowerright upperleft lowerleft middleright middleleft	upperright	The position of the legends for all plots in this group.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.

TABLE 7-69: VALID PROPERTY/VALUE PAIRS FOR ID PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
manualgrid	on off	off	Whether to use the automatic grid spacing or the grid settings specified in xspacing and yspacing.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
phaseintitle	on off	off	Whether the title should contain the phase when titletype is custom.
prefixintitle	String	Empty	Added prefix to group contribution to title.
preserveaspect	on off	off	Whether the x- and y-axis should have equal scale.
showlegendsmaxmin	on off	off	Show maximum and minimum values for color legends (when a Color Expression has been added).
solnum	Integer array	All solutions	The solutions to plot. Available when the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to group contribution to title.
t	double array	Empty	The times to plot. Available when the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual.
titlecolor	custom black blue cyan green magenta red white yellow	black	The color of the title.
titletype	auto custom manual none	auto	auto if the title should be computed automatically. custom if the title should be computed automatically, but customized. manual if the manual title should be used (the title property). none if no title should be displayed.

TABLE 7-69: VALID PROPERTY/VALUE PAIRS FOR 1D PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
typeintitle	on off	on	Whether the title contribution of plots in this group should contain the type when titletype is custom.
unitintitle	on off	on	Whether the title contribution of plots in this group should contain the unit when titletype is custom.
xextra	double array	Empty	Extra grid points to include along the x-axis
xlabel	String	Empty	The label on the x-axis.
xlabelactive	on off	off	on if a manual x-axis label should be used, off if it should be computed automatically.
xlog	on off	off	Whether to use a logarithmic scale along the x-axis.
xmax	double	Computed automatically	The maximum value of the x-axis.
xmin	double	Computed automatically	The minimum value of the x-axis.
xspacing	double	Computed automatically	The grid spacing on the x-axis. Used if manualgrid is set to on.
yextra	double array	Empty	Extra grid points to include along the y-axis
ylabel	String	Empty	The label on the y-axis.
ylabelactive	on off	off	on if a manual y-axis label should be used, off if it should be computed automatically.
ylog	on off	off	Whether to use a logarithmic scale along the y-axis.
ymax	double	Computed automatically	The maximum value of the y-axis.
ymin	double	Computed automatically	The minimum value of the y-axis.
yspacing	double	Computed automatically	The grid spacing on the y-axis. Used if manualgrid is set to on.
window	graphics new windowX, where X is an integer	graphics	The window where the plot group is displayed. When plotting on a graphics server, graphics is equivalent to new.
windowtitle	String	Computed automatically	The title to use for the window where the plot group is displayed. It is not possible to change the title of the graphics window.
windowtitleactive	on off	off	Set windowtitleactive to on to enter a manual window title in windowtitle.

The following properties are available for 2D and 3D plot groups:

TABLE 7-70: VALID PROPERTY/VALUE PAIRS FOR 2D AND 3D PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
axisactive	on off	off	Use manual axis formatting settings.
axiscommonexp	on off	on	Use a common exponent for all axis values.
axisnotation	automatic engineering scientific	automatic	Use an automatic notation (decimal or scientific) for axis values or always use engineering or scientific notation.

TABLE 7-70: VALID PROPERTY/VALUE PAIRS FOR 2D AND 3DPLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
axisprecision	positive integer	4	Precision for number format (number of digits).
axistrailingzeros	on off	off	Show trailing zeros for axis values.
customedgecolor	RGB-triplet	{1,1,1} or last used edgecolor	The color to use when plotting data set edges. Active when edgecolor is set to custom.
data	none data set name	parent	The data set this feature refers to. This is the default data set for all plots in the group.
datasetintitle	on off	off	Whether the title should contain the data set when titletype is custom.
edgecolor	custom black blue cyan green magenta red white yellow	black	The color to use when plotting data set edges.
edges	on off	on	Whether to plot data set edges.
expressionintitle	on off	off	Whether the title contribution of plots in this group should contain the expression when titletype is custom.
frametype	mesh material spatial geometry	spatial	The frame used when data set edges are plotted.
interp	double array	Time corresponding to last selected solnum for transient levels	The time to use, for transient levels. Available when data is not none and the underlying data is transient.
legendactive	on off	off	Use manual color legend formatting settings.
legendcolor	custom black blue cyan green magenta red white yellow	black	How to color the values in the color legend.
legendcommonexp	on off	on	Use a common exponent for all color legend values.
legendnotation	automatic engineering scientific	automatic	Use an automatic notation (decimal or scientific) for color legend values or always use engineering or scientific notation.
legendprecision	positive integer	3	Precision for number format (number of digits).
legendpos	alternating bottom left leftdouble right rightdouble	right	The position of the legends for all plots in this group.
legendtrailingzeros	on off	off	Show trailing zeros for axis values.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not none and the underlying data has multiple solutions.

TABLE 7-70: VALID PROPERTY/VALUE PAIRS FOR 2D AND 3DPLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
phaseintitle	on off	off	Whether the title should contain the phase when titletype is custom.
prefixintitle	String	Empty	Added prefix to group contribution to title.
showhiddenobjects	on off	off	Whether objects hidden in geometry sequence should be visible in plots.
showlegends	on off	on	Show color legends.
showlegendsmaxmin	on off	off	Show maximum and minimum values in color legends.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
solutionintitle	on off	on	Whether the title should contain the solution when titletype is custom.
suffixintitle	String	Empty	Added suffix to group contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. If it is not available in any solnum, the time is interpolated in the solution.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title should be computed automatically. custom if the title should be computed automatically, but customized. manual if the manual title should be used (the title property). none if no title should be displayed.
typeintitle	on off	on	Whether the title contribution of plots in this group should contain the type when titletype is custom.
unitintitle	on off	on	Whether the title contribution of plots in this group should contain the unit when titletype is custom.

TABLE 7-70: VALID PROPERTY/VALUE PAIRS FOR 2D AND 3DPLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
view	auto view name	auto	The view settings to use when displaying this plot group. auto indicates that the view is selected automatically, or be created if there is none. Typically geometry-based data sets such as solution or mesh data sets use the current geometry view. Surface plots with height attributes do not use the view setting.
window	graphics new windowX, where X is an integer	graphics	The window where the plot group is displayed. When plotting on a graphics server, graphics is equivalent to new.
windowtitle	String	Computed automatically	The title to use for the window where the plot group is displayed. It is not possible to change the title of the graphics window.
windowtitleactive	on off	off	Set windowtitleactive to on to enter a manual window title in windowtitle.
xlabel	String	Empty	The label on the x-axis. Available for 2D plot groups.
xlabelactive	on off	off	on if a manual x-axis label should be used, off if it should be computed automatically. Available for 2D plot groups.
ylabel	String	Empty	The label on the y-axis. Available for 2D plot groups.
ylabelactive	on off	off	on if a manual y-axis label should be used, off if it should be computed automatically. Available for 2D plot groups.

SEE ALSO

[PolarGroup](#), [Solution](#), [SmithGroup](#)

PoincareMap

Create a plot of a Poincaré map.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"PoincareMap");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"PoincareMap") creates a plot feature named <ftag> belonging to the plot group <pgtag>.

Plots of a Poincaré map are used to visualize where particle trajectories intersect a given plane, for example to illustrate where a ray passes or where particles are deposited. They can be added to 2D and 3D plot groups.

The following properties are available:

TABLE 7-71: VALID PROPERTY/VALUE PAIRS FOR POINCARÉ MAP

NAME	VALUE	DEFAULT	DESCRIPTION
data	none parent data set name	parent	The data set this feature refers to.
descriptionintitle	on off	on	Whether the title contribution should contain the description when <code>titletype</code> is <code>custom</code> .
inheritcolor	Boolean	true	If <code>inheritplot</code> is not none: Determines if the color is inherited. Available in 2D and 3D.
inheritrange	Boolean	true	If <code>inheritplot</code> is not none: Determines if the color and data ranges are inherited.
inheritspherescale	Boolean	true	If <code>inheritplot</code> is not none: Determines if the radius scale factor is inherited.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. <code>manual</code> indicates that <code>solnum</code> is used. <code>manualindices</code> indicates that <code>solnumindices</code> is used. <code>interp</code> indicates that <code>t</code> is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	all on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>looplevelinput</code> is <code>manualindices</code> on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. <code>manual</code> on a level indicates that <code>looplevel</code> is used on that level. <code>manualindices</code> on a level indicates that <code>looplevelindices</code> is used on that level. <code>interp</code> on a level indicates that <code>interp</code> is used on that level.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. <code>manual</code> indicates that <code>outersolnum</code> is used. <code>manualindices</code> indicates that <code>outersolnumindices</code> is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of <code>solnum</code> , the <code>solnum</code> property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>outerinput</code> is <code>manualindices</code> .
prefixintitle	String	Empty	Added prefix to contribution to title.
pointradiusexpr	String	1	The point radius expression.

TABLE 7-71: VALID PROPERTY/VALUE PAIRS FOR POINCARÉ MAP

NAME	VALUE	DEFAULT	DESCRIPTION
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>innerinput</code> is <code>manualindices</code> .
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
sphereradiusscale	positive double		The scale factor applied to the point radii if <code>sphereradiusscaleactive</code> is on.
sphereradiusscaleactive	Boolean	false	If true, <code>sphereradiusscale</code> is used; otherwise the scale factor is computed automatically.
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when <code>titletype</code> is <code>manual</code>
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when <code>titletype</code> is <code>custom</code> .
unitintitle	on off	on	Whether the title contribution should contain the unit when <code>titletype</code> is <code>custom</code>

ATTRIBUTES[Color](#)*PointGraph*

Create a point graph plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"PointGraph");
model.result(<pgtag>).feature(<ftag>).selection(...);
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"PointGraph")` creates a graph point plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

Point plot is used to visualize quantities on points, either cut points or points in a geometry. Point plots can be added to 1D plot groups.

The following properties are available:

TABLE 7-72: VALID PROPERTY/VALUE PAIRS FOR POINT GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the point numbers of the plotted points for geometry-based point plots. For cut point plots, the legend displays the point coordinates.
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom
freqmax	integer		If xaxisdata is spectrum and freqrangeactive is true: The upper frequency bound.
freqmin	integer		If xaxisdata is spectrum and freqrangeactive is true: The lower frequency bound.
freqrangeactive	on off	false	If xaxisdata is spectrum: Controls whether a manual frequency range is used.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends

TABLE 7-72: VALID PROPERTY/VALUE PAIRS FOR POINT GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
nfreqs	integer	1	If xaxisdata is spectrum and nfreqsactive is true: The number of frequencies to plot.
nfreqsactive	Boolean	false	If xaxisdata is spectrum: Controls whether the number of frequencies is set manually.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.

TABLE 7-72: VALID PROPERTY/VALUE PAIRS FOR POINT GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title
scale	Boolean	false	If xdata is spectrum: The frequency spectrum is transformed so that it has the same scale as the original data.
solnum	integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to contribution to title
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom
xdata	expr solution spectrum phase	solution	x-axis data. expr uses the expression in xdataexpr. solution uses the available solutions in the underlying data set, for example, time steps. phase uses a phase range, and rearranged phase plots.

TABLE 7-72: VALID PROPERTY/VALUE PAIRS FOR POINT GRAPHS

NAME	VALUE	DEFAULT	DESCRIPTION
xdataexpr	String	Model-dependent	Expression for x-axis data
xdatadescr	String	Model-dependent	Description of expression in xdataexpr
xdataphaserange	double array	range(0, 0.5, 2 π)	The phases for which the expression should be evaluated when xdata is phase.
xdataphaseunit	String	rad	The unit in which xdataphaserange is described
xdatasolnumtype	all inner outer valid level	outer	Whether the expression should be evaluated for every inner or every outer solution, or for a specific level (level1, level2, and so on). Applicable only for models containing multiple levels.
xdataunit	String	Model-dependent	The unit to use for the expression in xdataexpr. If the old unit is not valid when the expression changes, the unit property is reset to default.

ATTRIBUTES

None.

SEE ALSO

[LineGraph](#)

PointTrajectories

Create a point trajectories plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"PointTrajectories");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).selection(...);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"PointTrajectories")` creates a point trajectories plot feature named `<ftag>` belonging to plot group `<pgtag>`.

`model.result(<pgtag>).feature(<ftag>).selection()` returns the selection of points for which to plot trajectories. Selections are only available when you use points as the plot data. See [Selections](#) for more information about the available selection methods.

A solution or cut point data set is required to plot point trajectories.

The following properties are available:

TABLE 7-73: VALID PROPERTY/VALUE PAIRS FOR POINT TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the arrow position.
arrowexpr	string array		Active when pointtype is set to arrow. Determines the direction in which the arrow points.
arrowlength	logarithmic normalized proportional	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowscale	positive double	1	If arrowscalefactor is true: the length scale factor.
arrowscaleactive	boolean	false	If true, arrowscale is used, otherwise the scale factor is computed automatically.
arrowtype	Arrow cone	Arrow	The type of arrow to draw.
comettailexpr	String array	inverse of velocity vector of the first available particle tracing interface	Active when pointtype is set to comettail. Determines the direction in which the comet tail points and the relative sizes of the tails of different point trajectories.
data	none parent data set name	parent	The data set this feature refers to.
inheritarrowscale	Boolean	true	If inheritplot is not none: Determines if the arrow scale is inherited.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale are inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the radius scale factor is inherited.
inherittailscale	Boolean	true	If inheritplot is not none: Determines if the tail scale factor is inherited.
inherit tubescale	Boolean	true	If inheritplot is not none: Determines if the tube radius scale factor is inherited.
interpolation	none uniform	none	The type of interpolation used to plot lines when linetype is line or tube.

TABLE 7-73: VALID PROPERTY/VALUE PAIRS FOR POINT TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
linecolor	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use for lines.
linetype	none line tube	line	Plot particle traces as lines or tubes, or not at all.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
plotdata	coordinates points	coordinates	The plot data used for the point trajectories: points in the geometry (points) or a user-defined point (coordinate).
pointautoscale	on off	on	If enabled, the point radii are scaled based on the size of the geometry.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points.
pointlineanim	on off	off	When animating particle tracing plots with points, this property indicates whether the points at the current time should appear on top of the lines.
pointradiusexpr	String	0.001	The point radius expression.
pointtype	none point comettail	none	Plot the points in the trajectory plot as points or with comet tails, or not at all.
radiusexpr	String	1	The tube radius expression
sphereradiusscale	positive double		The scale factor applied to the point radii if sphereradiusscaleactive is on.
sphereradiusscaleactive	on off	off	If on, sphereradiusscale is used; otherwise the scale factor is computed automatically.
tailscale	positive double	1	Active when pointtype is set to comettail and tailscaleactive is on. Specifies the manual scale factor with which the comet tail expression is multiplied.
tailscaleactive	on off	off	Active when pointtype is set to comettail. Specifies whether manual tail scaling is enabled.
title	String		Active when title is set to manual. Specifies the plot title.
titletype	automatic manual none	automatic	Determines how the plot title is specified.
tuberadiusscale	String	1	The scale factor applied to the tube radii if tuberadiusscaleactive is on.

TABLE 7-73: VALID PROPERTY/VALUE PAIRS FOR POINT TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
tuberadiusscaleactive	on off	off	If on, tuberadiusscale is used, otherwise the scale factor is computed automatically
xcoordsexpr	Scalar expression		The expression for the x-coordinate when plot data are the coordinates of a user-defined point.
xpointexpr	Scalar expression		The expression for the x-coordinate when plot data are points.
ycoordsexpr	Scalar expression		The expression for the y-coordinate when plot data are the coordinates of a user-defined point.
ypointexpr	Scalar expression		The expression for the y-coordinate when plot data are points.
zcoordsexpr	Scalar expression		The expression for the z-coordinate when plot data are the coordinates of a user-defined point.
zpointexpr	Scalar expression		The expression for the z-coordinate when plot data are points.

SEE ALSO[ParticleTrajectories](#)**ATTRIBUTES**[Color](#), [Deform](#), [Filter \(Point Trajectories\)](#)*PolarGroup*

Create a polar plot group.

SYNTAX

```
model.result().create(<pgtag>, "PolarGroup");
model.result(<pgtag>).set(property, <value>);
model.result(<pgtag>).run();
```

DESCRIPTION

`model.result().create(<pgtag>, "PolarGroup")` creates a polar plot group named `<pgtag>`. A polar plot group displays the containing graph plots in a polar coordinate system.

The following properties are available for polar plot groups:

TABLE 7-74: VALID PROPERTY/VALUE PAIRS FOR POLAR PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to. This is the default data set for all plots in the group.
datasetintitle	on off	off	Whether the title should contain the data set when <code>titletype</code> is custom
expressionintitle	on off	off	Whether the title contribution of plots in this group should contain the expression when <code>titletype</code> is custom

TABLE 7-74: VALID PROPERTY/VALUE PAIRS FOR POLAR PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not none and the underlying data is transient.
legendpos	upperright lowerright upperleft lowerleft	upperright	The position of the legends for all plots in this group
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
manualgrid	on off	off	Whether to use the automatic grid spacing or the grid settings specified in rspacing and tspacing.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
phaseintitle	on off	off	Whether the title should contain the phase when titletype is custom.
prefixintitle	String	Empty	Added prefix to group contribution to title
rextra	double array	Empty	Extra grid points to include along the r-axis
rmax	double	Computed automatically	The maximum value of the r-axis

TABLE 7-74: VALID PROPERTY/VALUE PAIRS FOR POLAR PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
rmin	double	Computed automatically	The minimum value of the r-axis
rspacing	double	Computed automatically	The grid spacing on the r-axis. Used if manualgrid is set to on.
showlegendsmaxmin	on off	off	Show maximum and minimum values for color legends (when a Color Expression has been added).
solnum	Integer array	All solutions	The solutions to plot. Available when the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to group contribution to title
t	double array	Empty	The times to plot. Available when the underlying solution is transient.
textra	double array	Empty	Extra grid points to include along the θ -axis, in degrees
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title should be computed automatically. custom if the title should be computed automatically, but customized. manual if the manual title should be used (the title property). none if no title should be displayed.
tspacing	double	Computed automatically	The grid spacing on the θ -axis. Used if manualgrid is set to on.
typeintitle	on off	on	Whether the title contribution of plots in this group should contain the type when titletype is custom
unitintitle	on off	on	Whether the title contribution of plots in this group should contain the unit when titletype is custom
window	graphics new windowX, where X is an integer	graphics	The window where the plot group is displayed. When plotting on a graphics server, graphics is equivalent to new.
windowtitle	String	Computed automatically	The title to use for the window where the plot group is displayed. It is not possible to change the title of the graphics window.
windowtitleactive	on off	off	Set windowtitleactive to on to enter a manual window title in windowtitle

SEE ALSO

[PlotGroup1D](#), [PlotGroup2D](#), [PlotGroup3D](#), [SmithGroup](#)

PrincipalSurface, PrincipalVolume

Create a principal stress or principal strain plot in volumes or on surfaces.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"PrincipalVolume");
model.result(<pgtag>).create(<ftag>,"PrincipalSurface");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"PrincipalVolume")` create a volume principal stress or principal strain plot named `<ftag>` belonging to the plot group `<pgtag>`.

The following properties are available:

TABLE 7-75: VALID PROPERTY/VALUE PAIRS FOR PRINCIPAL STRESS/STRAIN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowcount	positive integer	200	The number of arrows. Available for PrincipalSurface in 3D.
arrowlength	normalized proportional logarithmic	proportional	The arrow scaling: Proportional uses the actual arrow length, normalized a unit length, and logarithmic a length proportional to the logarithm of the arrow length.
arrowtype	arrow cone	arrow	The type of arrow to draw
arrowxmethod	number coord	number	Indicates whether the x-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
arrowymethod	number coord	number	Indicates whether the y-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
arrowzmethod	number coord	number	Indicates whether the z-coordinates of the arrows should be specified by number of points to distribute evenly across the data or by coordinates. Available for PrincipalVolume in 3D.
color	custom black blue cyan green magenta red white yellow	red	The color to use
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
inheritarrowyscale	Boolean	true	If inheritplot is not none: Determines if arrow scale is inherited.

TABLE 7-75: VALID PROPERTY/VALUE PAIRS FOR PRINCIPAL STRESS/STRAIN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
inherit color	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that arrow scale, color, and deformation scale is inherited from
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
placement	elements uniform uniformani	uniform	If uniform, then the arrows are distributed uniformly. If uniformanisotropic, then the arrows are distributed uniformly with weights taken from the weight property. If elements, then arrows are drawn in each element. Available for PrincipalSurface in 3D.
prncdirstrainexpr	3-by-3 string matrix	Physics-dependent	If type is strain: The expressions for the principal directions.
prncdirstressexpr	3-by-3 string matrix	Physics-dependent	If type is stress: The expressions for the principal directions.
prncvalstrainexpr	1-by-3 string matrix	Physics-dependent	If type is strain: The expressions for the principal values.
prncvalstressexpr	1-by-3 string matrix	Physics-dependent	If type is stress: The expressions for the principal values.
scale	positive double	1	If scaleactive is true: The length scale factor.
scaleactive	Boolean	false	Whether to use manual scaling.
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise, first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.

TABLE 7-75: VALID PROPERTY/VALUE PAIRS FOR PRINCIPAL STRESS/STRAIN PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
t	double	Time corresponding to last selected solnum	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title should be used (the title property). none if no title should be displayed.
type	stress strain	stress	Selection between principal stress and principal strain plot.
weight	double array	1	If placement is uniform: The weights given to the different axis directions.
xnumber	non-negative integer	15 (3D), 7 (2D)	Number of points in the x-direction. Active when arrowxmethod is set to number. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when arrowxmethod is set to coord. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
ynumber	integer	15 (3D), 7 (2D)	Number of points in the y-direction. Active when arrowymethod is set to number. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
ycoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowymethod is set to coord. Available for PrincipalSurface in 2D and PrincipalVolume in 3D.
znumber	integer	15 (3D), 7 (2D)	Number of points in the z-direction. Active when arrowzmethod is set to number. Available for PrincipalVolume in 3D.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowzmethod is set to coord. Available for PrincipalVolume in 3D.

Ray (1D Plot)

Create a ray plot, which plots ray properties over time or compares ray properties against each other at a set of time steps.

SYNTAX

```
model.result(<pgtag>).create(<ftag>, "Ray1D");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>, "Ray1D")` creates a ray plot feature named `<ftag>` belonging to the plot group `<pgtag>`.

With the ray plot, you can plot properties of all rays in a data set versus time, or plot two ray properties against each other at a set of selected times. When plotting ray properties versus time, it is possible to apply data series operations to the ray data. The ray plot is available in 1D plot groups.

The following properties are available:

TABLE 7-76: VALID PROPERTY/VALUE PAIRS FOR RAY PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line.
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
dataseries	none average sum maximum minimum rms stddev variance	sum	The data series operation that is applied to all rays. Effective only when xdata is not expr.
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends.
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.

TABLE 7-76: VALID PROPERTY/VALUE PAIRS FOR RAY PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the color is different for each line.
linewidth	double	0.5	The line width
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.

TABLE 7-76: VALID PROPERTY/VALUE PAIRS FOR RAY PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>outerinput</code> is <code>manualindices</code> .
prefixintitle	String	Empty	Added prefix to contribution to title.
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, <code>range(1, 1, 20)</code> . Applicable when <code>innerinput</code> is <code>manualindices</code> .
solrepresentation	<code>solnum</code> <code>solutioninfo</code>	<code>solutioninfo</code>	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when <code>titletype</code> is <code>manual</code> .
titletype	<code>auto</code> <code>custom</code> <code>manual</code> <code>none</code>	<code>auto</code>	<code>auto</code> if the title contribution should be computed automatically, possibly using the group's customization. <code>custom</code> if the title contribution should be computed automatically, but customized. <code>manual</code> if the manual title contribution should be used (the <code>title</code> property). <code>none</code> if no title contribution should be used.
typeintitle	<code>on</code> <code>off</code>	<code>on</code>	Whether the title contribution should contain the type when <code>titletype</code> is <code>custom</code> .
unit	String	Model-dependent	The unit to use for the expression in <code>expr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	<code>on</code> <code>off</code>	<code>on</code>	Whether the title contribution should contain the unit when <code>titletype</code> is <code>custom</code> .
xdata	<code>expr</code> <code>solution</code>	<code>solution</code>	x-axis data. <code>expr</code> uses the expression in <code>xdataexpr</code> . <code>solution</code> uses the available solutions in the underlying data set, such as time steps.
xdataexpr	String	Model-dependent	Expression for x-axis data
xdatadescr	String	Model-dependent	Description of expression in <code>xdataexpr</code>
xdataphaseunit	String	<code>rad</code>	The unit in which <code>xdataphaserange</code> is described

TABLE 7-76: VALID PROPERTY/VALUE PAIRS FOR RAY PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
xdatasolnumtype	all inner outer valid level	outer	Whether the expression should be evaluated for every inner or every outer solution, or for a specific level (level1, level2, and so on). Applicable only for models containing multiple levels.
xdataunit	String	Model-dependent	The unit to use for the expression in xdataexpr. If the old unit is not valid when the expression changes, the unit property is reset to default.

ATTRIBUTES

[Color](#) (applies to lines), [Filter \(Ray Tracing\)](#)

SEE ALSO

[Ray \(Evaluation\)](#), [Ray Trajectories](#).

Ray (Data Set)

Create a ray data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Ray");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

model.result().dataset().create(<dtag>,"Ray") creates a ray data set.

A ray data set is required to plot ray trajectories. The ray data set identifies the geometry in which the ray data is stored and the degrees of freedom which determine the position of each ray.

The following properties are available:

TABLE 7-77: VALID PROPERTY/VALUE PAIRS FOR RAY DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
solution	String	first available solution	The solution this data set refers to
pgeomspec	manual fromphysics	manual	The method of specifying the ray geometry and the names of the ray position degrees of freedom.
physicsinterface	none or the tag of a valid physics interface	none	The physics interface from which the ray geometry and the names of the ray position degrees of freedom are defined.
rgeom	String	rgeom	The geometry in which the ray degrees of freedom are defined. The correct name of this ray geometry is pgeom_<name>, where <name> is the name for the Geometrical Optics or Ray Acoustics interface node.
posdof	String array	mod1.qx, mod1.qy	The position degrees of freedom of the rays

Ray (Evaluation)

Ray evaluations to evaluate quantities on ray trajectories.

SYNTAX

```
model.result().numerical().create(<ftag>, "Ray");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getReal();
model.result().numerical(<ftag>).getReal(<columnwise>);
model.result().numerical(<ftag>).getReal(<outersolnum>);
model.result().numerical(<ftag>).getReal(<columnwise>, <outersolnum>);
model.result().numerical(<ftag>).getImag();
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>);
model.result().numerical(<ftag>).getImag(<outersolnum>);
model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>, <outersolnum>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).computeResult();
model.result().numerical(<ftag>).setResult();
model.result().numerical(<ftag>).appendResult();
```

DESCRIPTION

`model.result().numerical().create(<ftag>, "Ray")` creates a ray evaluation feature with the name `<ftag>`. Ray evaluations can be performed on trajectories accessed through a ray data set.

`model.result().numerical(<ftag>).getReal()` returns the real result, recomputing the feature if necessary. Data is ordered such that there is one row per point, with one row containing data for all solution numbers. This is identical to (`<columnwise>`) when `<columnwise>` is `false`. If `<columnwise>` is `true`, the ordering is the opposite: each *column* contains the values for all solution numbers.

`model.result().numerical(<ftag>).getImag(<allocate>, <columnwise>)` returns the imaginary part of complex result, recomputing the feature if necessary. If `<allocate>` is `true`, a zero-valued matrix is allocated even when the result is real. `getImag()` uses `<allocate>` `true` and `<columnwise>` `false`.

`model.result().numerical(<ftag>).isComplex()` returns `true` if the result is complex.

`<outersolnum>` is applicable only for parametric sweep solutions and is the index of each outer parametric solution.

`model.result().numerical(<ftag>).computeResult()` returns the matrix of data that the `setResult` method adds to a table. The matrix includes data only, not the parameter columns, and it does not use any table-specific settings.

`model.result().numerical(<ftag>).setResult()` and `model.result().numerical(<ftag>).appendResult()` evaluates the feature and set or append the result in the table indicated by the `table` property.

The following properties are available:

TABLE 7-78: VALID PROPERTY/VALUE PAIRS FOR RAY EVALUATIONS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First ray compatible data set	The ray data set this feature refers to
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
evaluate	all fraction number	all	What ray to evaluate
expr	String	Model-dependent	The expression to plot
fraction	double	1	If evaluate is fraction: The fraction of rays to evaluate.

TABLE 7-78: VALID PROPERTY/VALUE PAIRS FOR RAY EVALUATIONS

NAME	VALUE	DEFAULT	DESCRIPTION
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
number	double	100	If evaluate is number: The number of particles to evaluate.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when outerinput is manualindices.
solnum	non-negative integer array	All solutions	The solutions to use
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1, 1, 20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
t	double array	Empty	The times to use. Available when the underlying solution is transient.

TABLE 7-78: VALID PROPERTY/VALUE PAIRS FOR RAY EVALUATIONS

NAME	VALUE	DEFAULT	DESCRIPTION
table	new table name	new	The table to use when calling <code>setResult()</code> or <code>appendResult()</code> . <code>new</code> indicates that a new table is created.
tablecols	inner outer data level1	data	Whether to use inner or outer solutions as columns in the table when calling <code>setResult()</code> or <code>appendResult()</code> . Applicable only for parametric sweep models. The level values (<code>level1</code> , <code>level2</code> , and so on) are the levels in the parametric sweep.
unit	String	Model-dependent	The unit to use for the expression in <code>expr</code> . If the old unit is not valid when the expression changes, the unit property is reset to default.

Ray Trajectories

Create a ray trajectories plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"RayTrajectories");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"RayTrajectories")` creates a ray trajectories plot feature named `<ftag>` belonging to plot group `<pgtag>`.

A ray data set is required to plot ray trajectories. The ray data set identifies the geometry in which the ray data is stored and the degrees of freedom which determine the position of each ray.

The following properties are available:

TABLE 7-79: VALID PROPERTY/VALUE PAIRS FOR RAY TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
arrowbase	head tail center	head	Whether the head, tail, or center of the arrow is located at the ray position.
arrowexpr	string array		Active when <code>pointtype</code> is set to <code>arrow</code> . Determines the direction in which the arrow points.
arrowlength	logarithmic normalized proportional	proportional	The arrow scaling: <code>Proportional</code> uses the actual arrow length, <code>normalized</code> a unit length, and <code>logarithmic</code> a length proportional to the logarithm of the arrow length.
arrowscale	positive double	1	If <code>arrowscalefactor</code> is true: the length scale factor.
arrowscaleactive	boolean	false	If true, <code>arrowscale</code> is used, otherwise the scale factor is computed automatically.
arrowtype	Arrow cone	Arrow	The type of arrow to draw

TABLE 7-79: VALID PROPERTY/VALUE PAIRS FOR RAY TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
comettailexpr	String array	inverse of velocity vector of the first available ray tracing interface	Active when pointtype is set to comettail. Determines the direction in which the comet tail points and the relative sizes of the tails of different rays.
data	none parent data set name	parent	The data set this feature refers to
inheritarrowsscale	Boolean	true	If inheritplot is not none: Determines if the arrow scale is inherited.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritellipsscale	Boolean	true	If inheritplot is not none: Determines if the ellipse scale factor is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale are inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the radius scale factor is inherited.
inherittailscale	Boolean	true	If inheritplot is not none: Determines if the tail scale factor is inherited.
inherit tubescale	Boolean	true	If inheritplot is not none: Determines if the tube radius scale factor is inherited.
interpolation	none uniform	none	The type of interpolation used to plot lines when linetype is line or tube.
linecolor	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use for lines
linetype	none line tube	line	Plot ray traces as lines or tubes, or not at all
logrange	double > 1	100	If arrowlength is logarithmic: The ratio between the maximum arrow length and the arrow length below which no arrow is drawn.
pointautoscale	on off	on	If enabled, the point radii are scaled based on the size of the geometry.
pointcolor	custom black blue cyan green magenta red white yellow	red	The uniform color to use for points.
pointlineanim	on off	off	When animating ray trajectories plots with points, this property indicates whether the points at the current time should appear on top of the lines.
pointradiusexpr	String	0.001	The point radius expression

TABLE 7-79: VALID PROPERTY/VALUE PAIRS FOR RAY TRAJECTORIES PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
pointtype	none point comettail	none	Plot rays as points or with comet tails, or not at all.
radiusexpr	String	1	The tube radius expression
sphereradiussscale	positive double		The scale factor applied to the point radii if sphereradiussscaleactive is on.
sphereradiussscaleactive	on off	off	If on, sphereradiussscale is used; otherwise the scale factor is computed automatically.
tailscale	positive double	1	Active when pointtype is set to comettail and tailscaleactive is on. Specifies the manual scale factor with which the comet tail expression is multiplied.
tailscaleactive	on off	off	Active when pointtype is set to comettail. Specifies whether manual tail scaling is enabled.
title	String		Active when title is set to manual. Specifies the plot title.
titletype	automatic manual none	automatic	Determines how the plot title is specified
tuberadiussscale	String	1	The scale factor applied to the tube radii if tuberadiussscaleactive is on.
tuberadiussscaleactive	on off	off	If on, tuberadiussscale is used, otherwise the scale factor is computed automatically

SEE ALSO

[Ray \(1D Plot\)](#), [Ray \(Evaluation\)](#)

ReflectionGraph, ImpedanceGraph, AdmittanceGraph

Create a reflection graph, impedance graph, or admittance graph in a Smith plot. You can add a Reflection Graph, Impedance Graph, and Admittance Graph to a [SmithGroup](#).

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"ReflectionGraph");
model.result(<pgtag>).create(<ftag>,"ImpedanceGraph");
model.result(<pgtag>).create(<ftag>,"AdmittanceGraph");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

The reflection graph, impedance graph, and admittance graph are all graphs that appear in a Smith plot. The expressions to use can be S-parameters from an RF simulation, for example. For an impedance plot you can also provide a reference impedance; for an admittance plot you can also provide a reference admittance.

The following properties are available:

The following properties are available:

TABLE 7-80: VALID PROPERTY/VALUE PAIRS FOR REFLECTION GRAPH, IMPEDANCE GRAPH, AND ADMITTANCE GRAPH.

NAME	VALUE	DEFAULT	DESCRIPTION
autodescr	on off	Model dependent	Whether the automatic legends should include the expression descriptions.
autoexpr	on off	Model dependent	Whether the automatic legends should include the expressions.
autolegends	on off	on	Whether to use the automatically computed legends or the legends defined in the legends property. The automatic legends display the description and expression for each line.
autounit	on off	off	Whether the automatic legends should include the unit.
const	String array of property/ value pairs	Empty	Parameters to use in the expressions.
customlinecolor	RGB-triplet	{0,0,1} or last used edgecolor.	The color to use for the lines. Active when linecolor is set to custom.
data	none parent data set name	parent	The data set this feature refers to.
descr	String array	Model-dependent.	The description of the expressions in expr. Is used in the automatic legends.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array	Model-dependent.	The expressions to plot.
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not parent and the underlying data is transient.
legend	on off	off	Whether to show legends.

TABLE 7-80: VALID PROPERTY/VALUE PAIRS FOR REFLECTION GRAPH, IMPEDANCE GRAPH, AND ADMITTANCE GRAPH.

NAME	VALUE	DEFAULT	DESCRIPTION
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width.
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level.
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	nonnegative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.

TABLE 7-80: VALID PROPERTY/VALUE PAIRS FOR REFLECTION GRAPH, IMPEDANCE GRAPH, AND ADMITTANCE GRAPH.

NAME	VALUE	DEFAULT	DESCRIPTION
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
prefixintitle	String	Empty	Added prefix to contribution to title.
solnum	nonnegative integer array	All solutions	The solutions to plot. Available when data is not parent and the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double array	Empty	The times to plot. Available when data is not parent and the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
y0	double	50	Reference impedance (Impedance Graph only).
z0	double	0.02	Reference admittance (Admittance Graph only).

ATTRIBUTE[Color](#)**SEE ALSO**[SmithGroup](#)

Revolve1D, Revolve2D

Create a 1D or 2D revolve data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Revolve1D");
model.result().dataset().create(<dtag>,"Revolve2D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Revolve1D")` creates a 1D revolved data set feature named `<dtag>`.

`model.result().dataset().create(<dtag>,"Revolve2D")` creates a 2D revolved data set feature named `<dtag>`.

The revolved data set needs to supply variables that cannot be expressed in terms of the coordinates of the underlying data set: The spatial coordinates and the rotation angle. Therefore the xmesh interpolation/evaluation has to be extended with support for supplying the values of these variables in a given set of points. The names of the spatial coordinates need not be exposed to the user, but the revolution angle should be available as `revphi`.

The source data sets for the revolutions can be any 1D and 2D data sets although the feature is primarily intended for axisymmetric coordinate systems.

The following properties are available for Revolve1D:

TABLE 7-81: VALID PROPERTY/VALUE PAIRS FOR REVOLVE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
anglevar	String	Derived from feature name	The name of the angle variable in the revolved coordinate system
data	none data set name	First compatible data set	The data set this feature refers to
hasspacevars	Boolean	false	If true, spacevars and anglevars are made available for evaluation
planemap	xy xz	xz	The 3D plane that the 2D xy plane is mapped to
point	double	0	The point that is the axis of revolution
revangle	double	360	The revolution angle (degrees)
revlayers	integer >= 3	50	The number of revolution layers
spacevars	String array	Derived from feature name	The names of the space variables in the revolved coordinate system
startangle	double	0	The angle where the revolved sector begins (degrees)

The following properties are available for Revolve2D:

TABLE 7-82: VALID PROPERTY/VALUE PAIRS FOR 2D REVOLVE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to
method	twopoint pointdir	twopoint	Decides if the line should be specified by two points or through one point and a direction

TABLE 7-82: VALID PROPERTY/VALUE PAIRS FOR 2D REVOLVE DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
genpoints	double matrix	{{0, 0, 0}, {1,0,0}}	Active when method equals twopoint, this property contains the coordinates of the two points in the two rows of the matrix
layermethod	coarse custom fine normal	normal	The method used for choosing how many layers to use
pdpoint	double array	{0,0}	Active when method equals pointdir, this property contains the coordinates of the point.
pddir	double array	{0, 1}	Active when method equals pointdir, this property contains the direction
revlayers	integer >= 3	50	The number of revolution layers if layermethod is custom

SEE ALSO[Solution](#)*ScatterVolume, ScatterSurface*

Create a scatter plot on surfaces or in volumes.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"ScatterVolume");
model.result(<pgtag>).create(<ftag>,"ScatterSurface");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Scatter...") creates a scatter plot feature named <ftag> belonging to the plot group <pgtag>.

Scatter plots are available in 2D and 3D.

The following properties are available:

TABLE 7-83: VALID PROPERTY/VALUE PAIRS FOR SCATTER PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
arrowxmethod	number coord	number	Indicates whether the x-coordinates of the spheres should be specified by number of points to distribute evenly across the data or by coordinates
arrowymethod	number coord	number	Indicates whether the y-coordinates of the spheres should be specified by number of points to distribute evenly across the data or by coordinates
arrowzmethod	number coord	number	Indicates whether the z-coordinates of the spheres should be specified by number of points to distribute evenly across the data or by coordinates. Available for ScatterVolume.

TABLE 7-83: VALID PROPERTY/VALUE PAIRS FOR SCATTER PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The color to use
colorexpr	String		The expression for the colors of the spheres
colordescr	String		A description of colorexpr
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent	The description of the expressions in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array of length 2 in 2D and 3 in 3D	Mode-dependent	The components to plot.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritplot	none plot name	none	The plot that color and sphere scale is inherited from
inheritspherescale	Boolean	true	If inheritplot is not none: Determines if the sphere scale is inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.

TABLE 7-83: VALID PROPERTY/VALUE PAIRS FOR SCATTER PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title
radiusexpr	String		The expression for the radii of the spheres
radiusdescr	String		A description of radiusexpr
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
sphereradiusscale	double	1	The scale factor applied to the sphere radii if sphereradiusscaleactive is true.
sphereradiusscaleactive	Boolean	false	If true, sphereradiusscale is used; otherwise the scale factor is computed automatically.
suffixintitle	String	Empty	Added suffix to contribution to title
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom
xnumber	non-negative integer	15 (3D), 7 (2D)	Number of points in the x-direction. Active when arrowxmethod is set to number.
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when arrowxmethod is set to coord
ynumber	integer	15 (3D), 7 (2D)	Number of points in the y-direction. Active when arrowymethod is set to number.

TABLE 7-83: VALID PROPERTY/VALUE PAIRS FOR SCATTER PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
ycoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowmethod is set to coord
znumber	integer	15 (3D), 7 (2D)	Number of points in the z-direction. Active when arrowzmethod is set to number. Available for ScatterVolume.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when arrowzmethod is set to coord. Available for ScatterVolume.

SEE ALSO

[Surface](#), [Volume](#)

Sector2D, Sector3D

Create a 2D or 3D sector symmetry data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Sector2D");
model.result().dataset().create(<dtag>,"Sector3D");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Sector2D")` creates a 2D sector symmetry data set feature named `<dtag>`.

`model.result().dataset().create(<dtag>,"Sector3D")` creates a 3D sector symmetry feature named `<dtag>`.

This data set takes data from another data sets and expands it using sector symmetry around an axis.

The following properties are available for Sector 2D and Sector 3D:

TABLE 7-84: VALID PROPERTY/VALUE PAIRS FOR SECTOR SYMMETRY DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to
genpoints	double matrix	{{0, 0, 0}, {1,0,0}}	When method is twopoint: The coordinates of two points on the symmetry axis.Available for Sector3D.
hasspacevars	Boolean	false	If true, spacevars are made available for evaluation
hasvar	Boolean	false	If true, an axis indicator variable is defined
include	all manual	all	Use all sectors or specify manually which ones to include.
method	twopoint pointdir	twopoint	Decides if the line should be specified by two points or through one point and a direction. Available for Sector3D.

TABLE 7-84: VALID PROPERTY/VALUE PAIRS FOR SECTOR SYMMETRY DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
mode	integer	0	If sectors is odd or trans is rot: The azimuthal mode number use to transform the solution's phase between sectors.
pddir	double array	{0,1}	When method is pointdir: The direction in which the symmetry axis points. Available for Sector3D.
pdpoint	double array	{0,0}	When method is pointdir: The coordinates of a point on the symmetry axis. Available for Sector3D.
point	double array of length 2	{0, 0}	The coordinates of the symmetry axis. Available for Sector2D.
reflaxis	double array of length 2 in 2D and length 3 in 3D	{1, 0} in 2D, {1, 0, 0} in 3D	If trans is rotnrefl: The direction in which the reflection axis/plane (for 2D and 3D respectively) points.
sectors	integer >= 2	2	The number of sectors.
sectorsinclude	integer >= 2	sectors	The number of sectors to include (when include = manual).
sectorvar	String	created from feature's tag	Name of variable that evaluates to the sector number, which starts at 0 for the first sector.
spacevarst	String array of length 2 in 2D and length 3 in 3D	created from feature's tag	Names of variables that evaluate to the space coordinates in the coordinate system after sector symmetry expansion.
startsector	integer >= 0	0	Starting sector for manual selection of sectors to include (when include = manual).
trans	rot rotnrefl	rot	If sectors is even: Controls whether the transformation performed between consecutive sectors is rotation or rotation and reflection.

Slice

Create a slice plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Slice");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Slice") creates a slice feature named <ftag> belonging to the plot group <pgtag>.



Only one direction of slices is possible in one plot. Use different slice plots together to accomplish several slices in different directions.

The following properties are available:

TABLE 7-85: VALID PROPERTY/VALUE PAIRS FOR SLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color the slices
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
const	String array of property/ value pairs	Empty	Parameters to use in the expressions
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom
gendistance	double array	Empty	Vector of distances from base plane, indicating position of additional parallel planes. Active when genparaactive equals on and genpara equals number.
genmethod	threepoint pointnormal	threepoint	Active when planetype equals general, this property indicates whether the plane should be specified by three points or through one point and a normal

TABLE 7-85: VALID PROPERTY/VALUE PAIRS FOR SLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
gennumber	positive integer	4	Number of additional parallel planes. Active when genparaactive equals on and genpara equals number.
genpara	gennumber gendistances	number	Active when planetype equals general, and genparaactive equals on, this property indicates whether the additional parallel planes should be specify by a number or by a vector of relative distances.
genparaactive	on off	off	Active when planetype equals general, this property indicates whether there should be additional parallel planes created.
genpnpoint	double array of length three	{0,0,0}	Active when genmethod equals pointnormal, this property contains the coordinates of the point.
genpnvec	double array of length three	{0,0,0}	Active when genmethod equals pointnormal, this property contains the normal vector.
genpoints	double matrix of size 3x3	{{0,0,0}, {1,0,0}, {0,1,0}}	Active when genmethod equals threepoint, this property contains the coordinates of the three points, with one row per point.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale is inherited from
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
interactive	on off	off	If true, the isosurfaces can be moved interactively after the plot has been made.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
planetype	quick general	yz	Specify plane type
prefixintitle	String	Empty	Added prefix to contribution to title

TABLE 7-85: VALID PROPERTY/VALUE PAIRS FOR SLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
quickplane	xy yz zx	yz	Specify quick plane type. Active when planetype is quick.
quickx	double array	0	Absolute x-coordinates for planes, active when quickxmethod is set to coord
quickxmethod	number coord	number	Active when planetype equals quick and quickplane equals yz, this property indicates whether the planes should be specified by one or more absolute coordinates along the x-axis, or a number indicating the number of planes to distribute evenly.
quickxnumber	positive integer	5	The number of planes when quickxmethod is set to number
quicky	double array	0	Absolute y-coordinates for planes, active when quickymethod is set to coord.
quickymethod	number coord	number	Active when planetype equals quick and quickplane equals zx, this property indicates whether the planes should be specified by one or more absolute coordinates along the y-axis, or a number indicating the number of planes to distribute evenly.
quickynumber	positive integer	5	The number of planes when quickymethod is set to number
quickz	double array	0	Absolute z-coordinates for planes, active when quickzmethod is set to coord.
quickzmethod	number coord	number	Active when planetype equals quick and quickplane equals xy, this property indicates whether the planes should be specified by one or more absolute coordinates along the z-axis, or a number indicating the number of planes to distribute evenly.
quickznumber	positive integer	5	The number of planes when quickzmethod is set to number
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.

TABLE 7-85: VALID PROPERTY/VALUE PAIRS FOR SLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
shift	double	0	If interactive is on: The shift that is applied to the level values.
smooth	none internal everywhere	internal	Smoothing settings
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to contribution to title
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.

TABLE 7-85: VALID PROPERTY/VALUE PAIRS FOR SLICE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom

ATTRIBUTES

[Deform](#), [Filter](#)

SEE ALSO

[Volume](#)

SmithGroup

Create a Smith plot group..



The Smith Plot Group requires a license for the AC/DC Module, RF Module, or Plasma Module.

SYNTAX

```
model.result().create(<pgtag>, "SmithGroup");
model.result(<pgtag>).set(property, <value>);
model.result(<pgtag>).run();
```

DESCRIPTION

`model.result().create(<pgtag>, "SmithGroup")` creates a Smith plot group named `<pgtag>`. A Smith plot group displays the containing graph plots as Smith plots.

The following properties are available for Smith plot groups:

TABLE 7-86: VALID PROPERTY/VALUE PAIRS FOR SMITH PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
admittance	on off	off	Add grid for an admittance Smith plot.
data	none data set name	First compatible data set	The data set this feature refers to. This is the default data set for all plots in the group.
datasetintitle	on off	off	Whether the title should contain the data set when titletype is custom

TABLE 7-86: VALID PROPERTY/VALUE PAIRS FOR SMITH PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
expressionintitle	on off	off	Whether the title contribution of plots in this group should contain the expression when titletype is custom
impedance	on off	on	Add grid for an impedance Smith plot.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
interp	double row matrix	Empty on all levels.	The times to use, for transient levels. Available when data is not none and the underlying data is transient.
legendpos	upperright lowerright upperleft lowerleft	upperright	The position of the legends for all plots in this group
looplevel	integer row matrix	All solutions on all levels	The solutions to use, per level
looplevelindices	integer row matrix	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when looplevelinput is manualindices on a level.
looplevelinput	String array with all first last manual manualindices interp on each level	all on all levels	How to input the solution to use, per level. manual on a level indicates that looplevel is used on that level. manualindices on a level indicates that looplevelindices is used on that level. interp on a level indicates that interp is used on that level.
outerinput	all first last manual manualindices	all	How to input the outer solutions to use. Applicable only for parametric sweep models. manual indicates that outersolnum is used. manualindices indicates that outersolnumindices is used.
outersolnum	non-negative integer array	1	The index of the outer solutions to use. Applicable only for parametric sweep models. Since the various outer solutions can have different number of solnum, the solnum property is not used.
outersolnumindices	String or integer array	Empty	An alternate way to specify the outer solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when outerinput is manualindices.
phaseintitle	on off	off	Whether the title should contain the phase when titletype is custom.
prefixintitle	String	Empty	Added prefix to group contribution to title
resolution	fine normal coarse	normal	Grid resolution for Smith plot coordinate grid.

TABLE 7-86: VALID PROPERTY/VALUE PAIRS FOR SMITH PLOT GROUPS

NAME	VALUE	DEFAULT	DESCRIPTION
showlegendsmaxmin	on off	off	Show maximum and minimum values for color legends (when a Color Expression has been added).
solnum	Integer array	All solutions	The solutions to plot. Available when the underlying data has multiple solutions.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to group contribution to title
t	double array	Empty	The times to plot. Available when the underlying solution is transient.
title	String	The auto-title.	The title to use when titletype is manual
titletype	auto custom manual none	auto	auto if the title should be computed automatically. custom if the title should be computed automatically, but customized. manual if the manual title should be used (the title property). none if no title should be displayed.
typeintitle	on off	on	Whether the title contribution of plots in this group should contain the type when titletype is custom
unitintitle	on off	on	Whether the title contribution of plots in this group should contain the unit when titletype is custom
window	graphics new windowX, where X is an integer	graphics	The window where the plot group is displayed. When plotting on a graphics server, graphics is equivalent to new.
windowtitle	String	Computed automatically	The title to use for the window where the plot group is displayed. It is not possible to change the title of the graphics window.
windowtitleactive	on off	off	Set windowtitleactive to on to enter a manual window title in windowtitle

SEE ALSO

[PlotGroup1D](#), [PlotGroup2D](#), [PlotGroup3D](#), [PolarGroup](#)

Solution

Create a solution data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Solution");
model.result().dataset(<dtag>).set(property, <value>);
model.result().dataset(<dtag>).selection(...);
model.result().dataset(<dtag>).createDeformedConfig(<gtag>,<mtag>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>, "Solution")` creates a solution data set.

Solution data sets refer to a solution in a solver sequence. All result features that perform evaluations on a solution refer to these data sets rather than directly to the solution itself. The solution data sets specify which geometry and frame to use for evaluation and whether to evaluate only on a specific selection, among other things. It is possible to have multiple data set solutions referring to the same solution.

The following properties are available:

TABLE 7-87: VALID PROPERTY/VALUE PAIRS FOR SOLUTION DATA SETS

NAME	VALUE	DEFAULT	DESCRIPTION
frametype	mesh material spatial geometry	material or spatial	The frame type that this data set uses for evaluation of spatial coordinates.
geom	String	First available geometry, if any.	The geometry this data set refers to. Only applicable for solutions that refer to a geometry.
phase	double	0	Evaluate solution at this angle, given in degrees
scalefactor	double	1	Multiply solution by this scaling factor
solution	String	First compatible solution	The solution this data set refers to

`model.result().dataset(<dtag>).createDeformedConfig(<gtag>, <mtag>)` creates a new geometry sequence (deformed configuration) tagged <gtag> together with a mesh sequence tagged <mtag> to use for remeshing of a deformed mesh. The created geometry sequence only contains a FromMesh feature.

EXAMPLE

Create a solution data set and set it to point to the second geometry in the solver sequence sol1:

Code for use with Java

```
DatasetFeature ds = model.result().dataset().create("dset1", "Solution");
ds.set("solution", "sol1");
ds.set("geom", "geom2");
```

Code for use with MATLAB

```
ds = model.result.dataset.create('dset1', 'Solution');
ds.set('solution', 'sol1');
ds.set('geom', 'geom2');
```

SEE ALSO

[PlotGroup1D](#), [PlotGroup2D](#), [PlotGroup3D](#)

Streamline

Create a streamline plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>, "Streamline");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>, "Streamline")` creates a streamline plot feature named <ftag> belonging to the plot group <pgtag>.

Streamlines visualize a vector quantity on domains. A streamline is a curve everywhere tangent to an instantaneous vector field. Streamlines are available in 2D and 3D plot groups.

The following properties are available:

TABLE 7-88: VALID PROPERTY/VALUE PAIRS FOR STREAMLINE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
back	on off	on	Allow backward time integration. Only available for time-dependent data sets.
color	custom black blue cyan green magenta red white yellow	black (2D), red (3D)	The uniform color to use
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String array of length 2 in 2D and 3 in 3D	Model-dependent	The components to plot
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color, tube scale, and deformation scale is inherited from.
inherit tubescale	Boolean	true	If inheritplot is not none and linetype is tube or ribbon: Determines if the tube or ribbon scale is inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.

TABLE 7-88: VALID PROPERTY/VALUE PAIRS FOR STREAMLINE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
inttol	positive double	0.001 (2D), 0.01 (3D)	Integration tolerance
linetype	line tube ribbon	line	Plot streamlines as lines, tubes, or ribbons
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
looptol	positive double	0.01	Loop tolerance
madv	manual automatic	automatic	Whether to specify advanced parameters when positioning method is magnitude.
maxsteps	positive integer	5000	The maximum number of integration steps
maxtime	positive double	Inf	Maximum integration time
mdensity	non-negative integer	20	The density of the streamlines. Active when posmethod is set to magnitude. Only available in 2D.
mdist	double array of length two containing positive doubles	{0.05, 0.15}	Minimum and maximum distance relative to the size of the geometry. Active when posmethod is set to magnitude. Only available in 3D.
minitref	positive integer	1	Boundary element refinement. Active when madv is set to manual.
msatfactor	non-negative double	1.3	Starting distance factor. Active when madv is set to manual. Only available in 3D.
mseed	double array of length 2 or 3, depending on space dimension.	{0,0} (2D), {0,0,0} (3D)	The first start point. Active when mseedactive is set to on.
mseedactive	on off	off	Whether to manually specify first start point. Active when madv is set to manual.
normal	on off	off	on when vector field should be normalized
number	integer	20	Approximate number of streamlines. Active when startmethod is set to number.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
posmethod	magnitude start uniform selection	selection	The type of streamline positioning. Either start point controlled, uniform density, magnitude controlled, or starting from selected lines (2D) or surfaces (3D).
prefixintitle	String	Empty	Added prefix to contribution to title

TABLE 7-88: VALID PROPERTY/VALUE PAIRS FOR STREAMLINE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
radiusexpr	String	1	The tube radius
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
selnumber	non-negative integer	20	If posmethod is selection: The approximate number of streamlines placed on the selected edges (2D) or surfaces (3D).
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active
startdata	none cut line or cut plane name	none	Active when posmethod is start: The line or plane where the start points are distributed evenly.
startmethod	number coord	number	Active when posmethod is start. Indicates whether the start points should be specified by number of points to distribute evenly or by coordinates.
stattol	positive double	0.01	Stationary point stop tolerance
suffixintitle	String	Empty	Added suffix to contribution to title
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.

TABLE 7-88: VALID PROPERTY/VALUE PAIRS FOR STREAMLINE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
tuberadiussscale	double	1	The scale factor applied to the tube radii if tuberadiussscaleactive is true.
tuberadiussscaleactive	Boolean	false	If true, tuberadiussscale is used, otherwise the scale factor is computed automatically
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
uadv	manual automatic	automatic	Whether to specify advanced parameters when positioning method is uniform.
udist	positive double	0.15 (3D) , 0.05 (2D)	Separating distance relative to the size of the geometry. Active when posmethod is set to uniform.
udistend	non-negative double	0.5	Terminating distance factor. Active when uadv is set to manual.
uignoredist	non-negative double	0.5	Fraction of streamline length to ignore. Active when uadv is set to manual.
unitref	positive integer	1	Boundary element refinement. Active when uadv is set to manual.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
usatfactor	non-negative double	1.3	Starting distance factor. Active when uadv is set to manual.
useed	double array of length 2 or 3, depending on space dimension.	{0,0} (2D) or {0,0,0} (3D)	The first start point. Active when useedactive is set to on.
useedactive	on off	off	Whether to manually specify first start point. Active when uadv is set to manual.
widthexpr	String	1	The ribbon width
widthscale	double	1	The scale factor applied to the ribbon widths if widthscaleactive is true.
widthscaleactive	Boolean	false	If true, widthscale is used, otherwise the scale factor is computed automatically

TABLE 7-88: VALID PROPERTY/VALUE PAIRS FOR STREAMLINE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
xcoord	double array	Empty	Absolute coordinates in the x-direction, active when startmethod is set to coord.
ycoord	double array	Empty	Absolute coordinates in the y-direction, active when startmethod is set to coord.
zcoord	double array	Empty	Absolute coordinates in the z-direction, active when startmethod is set to coord. Available in 3D only.

ATTRIBUTES

[Color](#), [Deform](#), [Filter](#)

SEE ALSO

[Particle](#), [ParticleMass](#)

Surface

Create a surface plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>, "Surface");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>, "Surface") creates a surface plot feature named <ftag> belonging to the plot group <pgtag>.

Surface plots display a quantity as a colored 2D or 3D surface.

The following properties are available:

TABLE 7-89: VALID PROPERTY/VALUE PAIRS FOR SURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this surface
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.

TABLE 7-89: VALID PROPERTY/VALUE PAIRS FOR SURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic.
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritheightscale	Boolean	true	If inheritplot is not none: Determines if height scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.

TABLE 7-89: VALID PROPERTY/VALUE PAIRS FOR SURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
smooth	none internal everywhere	internal	Smoothing settings
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.
solrepresentation	solnum solutioninfo	solutioninfo 0	Indicates which method of selecting solutions is active
suffixintitle	String	Empty	Added suffix to contribution to title

TABLE 7-89: VALID PROPERTY/VALUE PAIRS FOR SURFACE PLOTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
wireframe	on off	off	Whether to plot filled elements or only their edges.

ATTRIBUTES

Deform, Filter, Height

EXAMPLES

Surface plot on 2D solution:

Code for use with Java

```
DatasetFeature ds = model.result().dataset().create("dset1", "Solution");
ds.set("Solution", "sol1");

ResultFeature pg = result().create("pg1",2);
pg.set("data","dset1");
pg.create("surf1","Surface");
pg.feature("surf1").set("expr", "3*u");
```

Code for use with MATLAB

```
ds = model.result.dataset.create('dset1', 'Solution');
ds.set('Solution', 'sol1');

ResultFeature pg = result().create('pg1',2);
pg.set('data','dset1');
pg.create('surf1','Surface');
pg.feature('surf1').set('expr', '3*u');
```

Surface plot on cut plane in 3D using a Thermal color table:

Code for use with Java

```
result().dataset().create("cp1", "CutPlane");
```

```
ResultFeature pg2 = result().create("pg2",3);
pg2.create("surf2","Surface");
pg2.feature("surf2").set("data", "cp1");
pg2.feature("surf1").set("expr", "2*u");
pg2.feature("surf1").set("colortable", "Thermal");

pg2.run();
```

Code for use with MATLAB

```
result.dataset.create('cp1', 'CutPlane');

pg2 = result.create('pg2',3);
pg2.create('surf2','Surface');
pg2.feature('surf2').set('data', 'cp1');
pg2.feature('surf1').set('expr', '2*u');
pg2.feature('surf1').set('colortable', 'Thermal');

pg2.run;
```

SEE ALSO

[Volume](#)

Surface (data set)

Create a surface data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"Surface");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"Surface")` creates a surface data set with the name `<dtag>`.

The surface data set makes it possible to evaluate surfaces of a 3D model in 2D or 3D. For evaluation in 2D, different parameterizations of the 2D projection are available.

The following properties are available:

TABLE 7-90: VALID PROPERTY/VALUE PAIRS FOR SURFACE DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The solution data set this feature refers to.
param	face xy yz zx yx zy xz expr	face	The projection used when evaluating in 2D.
exprx	String		When param is expr: The x-axis expression when evaluating in 2D.
expy	String		When param is expr: The y-axis expression when evaluating in 2D.

SystemMatrix

Create a numerical evaluation of system matrix.

SYNTAX

```
model.result().numerical().create(<ftag>,"SystemMatrix");
model.result().numerical(<ftag>).set(property, <value>);
model.result().numerical(<ftag>).getData();
model.result().numerical(<ftag>).getData(<expressionIndex>);
model.result().numerical(<ftag>).getImagData();
model.result().numerical(<ftag>).getImagData(<expressionIndex>);
model.result().numerical(<ftag>).isComplex();
model.result().numerical(<ftag>).isComplex(<outersolnum>);
model.result().numerical(<ftag>).getNData();
model.result().numerical(<ftag>).run();
```

DESCRIPTION

`model.result().numerical().create(<ftag>,"SystemMatrix")` create a system matrix evaluation feature with the name `<ftag>`.

The system matrix evaluation makes it possible to retrieve matrices directly from Assemble and Modal features in a solver sequence.

The following properties are available:

TABLE 7-91: VALID PROPERTY/VALUE PAIRS FOR SYSTEM MATRIX EVALUATIONS.

PROPERTY	VALUE	DEFAULT	DESCRIPTION
feature	String	The first Assemble or Modal feature in the chosen solution.	The solver feature from which the matrix is retrieved.
matrixassem	constraint constraintforcejac constraintforcenullbasis constraintjac constraintnullbasis damping elimdamping elimload elimmass elimstiffness load lowerboundconstraint mass optconstraint optconstraintjac partsol stiffness upperboundconstraint uscale	stiffness	The matrix to retrieve if the solver feature is an Assemble feature.
matrixmodal	constload damping dampingratio load mass pertload projection stiffness	stiffness	The matrix to retrieve if the solver feature is a Modal feature.
solution	String	First compatible solution	The solution this data set refers to.

Table

Create a feature containing a table of data.

SYNTAX

```
model.result().table().create(<ftag>, "Table");
model.result().table(<ftag>).setColumnHeaders(<headers>);
model.result().table(<ftag>).setTableData(<realData>);
model.result().table(<ftag>).setTableData(<realData>, <imagData>);
model.result().table(<ftag>).addColumnns(<headers>, <realData>);
model.result().table(<ftag>).addColumnns(<headers>, <realData>, <imagData>);
model.result().table(<ftag>).addRow(<realData>);
model.result().table(<ftag>).addRow(<realData>, <imagData>);
model.result().table(<ftag>).removeRow(<index>);
model.result().table(<ftag>).getColumnHeaders();
model.result().table(<ftag>).getReal();
model.result().table(<ftag>).getImag();
model.result().table(<ftag>).Row(<index>);
model.result().table(<ftag>).getImagRow(<index>);
model.result().table(<ftag>).getTableData(<fullPrecision>);
model.result().table(<ftag>).getTableRow(<index>, <fullPrecision>);
model.result().table(<ftag>).getNRows();
model.result().table(<ftag>).isComplex();
model.result().table(<ftag>).clearTableData();
model.result().table(<ftag>).set(property, <value>);
model.result().table(<ftag>).save(<filename>);
model.result().table(<ftag>).save(<filename>, <fullPrecision>);
model.result().table(<ftag>).loadFile(tempFile, ...);
model.result().table(<ftag>).saveFile(tempFile, ...);
```

DESCRIPTION

`model.result().table().create(<ftag>, "Table")` creates a table feature named `<ftag>`. Tables support two data formats, *all* or *filled*. Filled data is typically produced from parametric sweeps and makes it possible to retrieve data for a pair of parameters on a matrix format. Filled tables can be used to make response surface plots. (See [TableSurface](#).)

`model.result().table(<ftag>).setColumnHeaders(<headers>)` sets the table's column headers from the string array `<headers>`.

`model.result().table(<ftag>).setTableData(<realData>)` sets the table content from a double matrix containing real data. Any previous real or imaginary data is removed.

`model.result().table(<ftag>).setTableData(<realData>, <imagData>)` sets both real and imaginary data from the double matrices `<realData>` and `<imagData>`, which must be of the same size. `<imagData>` can be null to indicate that there is no imaginary data.

`model.result().table(<ftag>).getColumnHeaders()` retrieves the column headers.

`model.result().table(<ftag>).addColumnns(<headers>, <realData>)` adds one or more columns and associated real data to the table.

`model.result().table(<ftag>).addColumnns(<headers>, <realData>, <imagData>)` adds one or more columns and associated real data and imaginary data to the table.

`model.result().table(<ftag>).addRow(<realData>)` adds one row of real data to the table.

`model.result().table(<ftag>).addRow(<realData>, <imagData>)` adds one row of real and imaginary data to the table.

`model.result().table(<ftag>).removeRow(<index>)` removes the row with a given index from the table. If the row index is out of bounds, nothing happens.

`model.result().table(<ftag>).getColumnHeaders()` returns the column headers in the table.

`model.result().table(<ftag>).getReal()` returns the real part of the table content.

`model.result().table(<ftag>).getImag()` returns the imaginary part of the table content. Note: this method allocates imaginary data if there was none. Check for imaginary content with the `isComplex` method before calling this method if you want to avoid this.

`model.result().table(<ftag>).getRealRow(<index>)` returns the real data in one row.

`model.result().table(<ftag>).getImagRow(<index>)` returns the imaginary data in one row.

`model.result().table(<ftag>).getFilledReal()` returns the real part of the table content on a filled format, when available.

`model.result().table(<ftag>).getFilledImag()` returns the imaginary part of the table content on a filled format, when available.

`model.result().table(<ftag>).getTableData(<fullPrecision>)` returns the table data as a string matrix, with limited or full precision as specified by the Boolean `<fullPrecision>`.

`model.result().table(<ftag>).getTableRow(<index>, <fullPrecision>)` returns the table data for one row as a string array, with limited or full precision.

`model.result().table(<ftag>).getNRows()` returns the number of rows in the table.

`model.result().table(<ftag>).isComplex()` returns true if the table contains imaginary data. This method checks whether imaginary data has been allocated, not if it is different from 0.

`model.result().table(<ftag>).clearTableData()` removes all table data and column headers.

`model.result().table(<ftag>).save(<filename>)` saves the table `<ftag>` content to the text file `<filename>` in full precision.

For `model.result().table(<ftag>).loadFile()` and `model.result().table(<ftag>).saveFile()`, see [The loadFile and saveFile Methods](#).

The following properties are available:

TABLE 7-92: VALID PROPERTY/VALUE PAIRS FOR TABLES

NAME	VALUE	DEFAULT	DESCRIPTION
cols	Parameters in table none	none	For tables with filled data, cols controls which parameter is used on the columns. Cannot be the same parameter as in rows.
datacol	Positive integer	1	For tables with filled data, datacol controls which data column should be used to populate the filled table.
descr	none data manual	data	For tables with filled data, descr controls which description to use for the columns, in addition to the parameter values. none gives no description, manual uses the description in descrmanual, and data uses the description in the table data.
descrmanual	String	Empty	Manual column header for tables with filled data, used when descr is set to manual.
filename	String	Empty	The name of the file on which the table is stored, used when storetable is set to inmodelandonfile or onfile.

TABLE 7-92: VALID PROPERTY/VALUE PAIRS FOR TABLES

NAME	VALUE	DEFAULT	DESCRIPTION
filterstringdata	on off	off	For tables with filled data, filterstringdata controls whether to filter the input column using showrowstep.
format	all filled	all	Controls whether you retrieve all or filled table data when calling the getTableData method.
param	Positive integer	1	For tables with filled data containing more than two parameters, param is the index indicating which values to use for fixing additional parameters. Additional parameters are ordered from left to right in the original table data.
rows	Parameters in table none	none	For tables with filled data, rows controls which parameter is used on the rows.
showrowstep	Positive integer	1	When retrieving filled data using the getTableData method, showrowstep controls whether to leave all but every showrowstep input cells empty, for example, every third cell in the first column. This refers to the input column only. Only applied when filterstringdata is on.
storetable	inmodel inmodelandonfile onfile	inmodel	Controls whether table is stored in model(inmodel) or on file(onfile), or both(inmodelandonfile).
tablebuffersize	Positive integer	Taken from buffer size preference. The default of which is 10000.	The size of the in memory buffer size table is stored, used when storetable is set to inmodel or inmodelandonfile. In the second case only the last tablebuffersize rows are kept in the model; the rest is read from file when necessary.

SEE ALSO

[Table \(plot\)](#), [TableSurface](#)

Table (export)

Export data from tables to files.

SYNTAX

```
model.result().export().create(<ftag>,"Table");
model.result().export(<ftag>).set(property, <value>);
model.result().export(<ftag>).run();
```

DESCRIPTION

model.result().export().create(<ftag>,"Table") creates a table export feature with the name <ftag>.

The following properties are available:

TABLE 7-93: VALID PROPERTY/VALUE PAIRS FOR TABLE EXPORTS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
alwaysask	on off	off	Always ask for filename when saving. This property is ignored when running without a GUI.
ifexists	append overwrite	overwrite	If the file exists, append to or overwrite the file contents.
filename	String		The output file.
fullprec	on off	on	If on, floating-point numbers are written in full precision, otherwise they are written with six significant digits.
header	on off	off	Enable/disable a data header in the output file.
sort	on off	off	Enable/disable sorting of the points with respect to the coordinates.
table	none table name	First table	The table this feature refers to.

SEE ALSO

[Data, Table](#)

Table (plot)

Plot data from a table in a graph plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Table");  
model.result(<pgtag>).feature(<ftag>).set(property, <value>);  
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Table")` creates a graph table plot feature named `<ftag>`.

Table graphs can be added to 1D and polar plot groups and displays data from a table.

The following properties are available:

TABLE 7-94: VALID PROPERTY/VALUE PAIRS FOR TABLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
autolegends	on off	on	Whether to use the automatically generated legends or the legends defined in the legends property.
customlinecolor	RGB-triplet	{0,0,1} or last used edgcolor.	The color to use for the lines. Active when linecolor is set to custom.
freqmax	integer		If transform is spectrum and freqrangeactive is true: The upper frequency bound.
freqmin	integer		If transform is spectrum and freqrangeactive is true: The lower frequency bound.
freqrangeactive	on off	false	If transform is spectrum: Controls whether a manual frequency range is used.

TABLE 7-94: VALID PROPERTY/VALUE PAIRS FOR TABLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
imagplot	on off	off	If transform is none: When on, the imaginary part of the data in the table is plotted.
legend	on off	off	Whether to show legends
legendmethod	automatic manual	automatic	Whether to use the automatic legends or the legends supplied in the legends property.
legends	String array	The last computed automatic legends.	Manual legends active when legendmethod is set to manual.
linecolor	custom cycle black blue cyan green magenta red white yellow	cycle	How to color the lines in the graph. Cycle indicates that the colors is different for each line.
linewidth	double	0.5	The line width
linemarker	none cycle asterisk circle diamond plus point square star triangle	none	The line markers, if any. Cycle indicates that the marker is different for each line.
linestyle	none cycle solid dotted dashed dashdot	solid	The line style, if any. Cycle indicates that the line style is different for each line.
markerpos	interp datapoints	interp	Controls whether the positions of the markers are in the data points of the plot, or interpolated depending on the number (set in markers). Markers are visible when linemarker is set.
markers	integer	8	The number of markers to show. Markers are visible when linemarker is set.
nfreqs	integer	1	If transform is spectrum and nfreqsactive is true: The number of frequencies to plot.
nfreqsactive	Boolean	false	If transform is spectrum: Controls whether the number of frequencies is set manually.
plotcolumninput	all manual	all	The columns to plot. all indicates all columns excluding those used in xaxisdata, manual indicates the columns specified in plotcolumns.
plotcolumns	positive integer array	Empty	The columns to plot when plotcolumninput is manual.
preproc{x y}	none linear	none	Use a linear preprocessing of the table data values, which you specify using the scaling and shift properties.
scale	Boolean	false	If transform ffs is spectrum: The frequency spectrum is transformed so that it has the same scale as the original data.
scaling{x y}	double	1.0	Real scaling value for data preprocessing
shift{x y}	double	0.0	Real shift value for data preprocessing
table	none table feature name	First available table	The table feature this plot refers to.

TABLE 7-94: VALID PROPERTY/VALUE PAIRS FOR TABLE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
transform	none spectrum	none	The transformation to apply to the data before plotting.
xaxisdata	auto rowindex positive integer	auto	The column supplying x-axis data. auto determines input based on evaluated result, for example, using an index if the table contains output from a parametric sweep with multiple parameters. rowindex means that all columns are used as data and are plotted against the row number in the table.

SEE ALSO

[Table](#)

TableHeight

Create a height attribute for 2D table plots.

SYNTAX

```
model.result(<pgtag>).feature(<ftag>).create(<atag>,"TableHeight");
model.result(<pgtag>).feature(<ftag>).feature(<atag>).set(property, <value>);
```

DESCRIPTION

`model.result(<pgtag>).feature(<ftag>).create(<atag>,"TableHeight")` creates a `TableHeight` attribute feature with the name `<atag>`, belonging to the feature `<ftag>`.

Table height attributes are available for 2D table surface plots. Adding a height attribute changes the rendered view to 3D. The plot group, however, still remains a 2D plot group. Plots in the plot group that do not have height attributes are rendered at $z = 0$.

The following properties are available:

TABLE 7-95: VALID PROPERTY/VALUE PAIRS FOR TABLE HEIGHT

PROPERTY	VALUE	DEFAULT	DESCRIPTION
datacol	Positive integer	1	For tables with filled data, datacol controls which data column should be used to populate the filled table. Used when heightdata is data.
heightdata	parent data	parent	parent uses the data from the surface plot the height feature has been added to. data uses the data specified by the height attribute properties.
offset	double	0	The offset along the z-axis where to place the height plot.
scale	positive double	1	If scaleactive is true: The scale factor for the height.
scaleactive	Boolean	false	Whether to use manual scaling.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.

SEE ALSO

[TableSurface](#), [Table](#)

TableSurface

Plot data from a table in a surface plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"TableSurface");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

`model.result(<pgtag>).create(<ftag>,"Table")` creates a table surface plot feature named `<ftag>`.

Table surface plots can only be used with tables containing filled data, produced from a parametric sweep. They can be added to 2D plot groups.

The following properties are available:

TABLE 7-96: VALID PROPERTY/VALUE PAIRS FOR TABLE SURFACE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this surface
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
datacol	Positive integer	1	For tables with filled data, datacol controls which data column should be used to populate the filled table.
descr	none data manual	data	For tables with filled data, descr controls which description to use for the columns, in addition to the parameter values. none gives no description, manual uses the description in descrmanual, and data uses the description in the table data.
descrmanual	String	Empty	Manual column header for tables with filled data, used when descr is set to manual.

TABLE 7-96: VALID PROPERTY/VALUE PAIRS FOR TABLE SURFACE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
filterstringdata	on off	off	For tables with filled data, filterstringdata controls whether to filter the input column using showrowstep.
format	all filled	all	Controls whether you retrieve all or filled table data when calling the getTableData method.
imagplot	on off	off	If transform is none: When on, the imaginary part of the data in the table is plotted.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritheightscale	Boolean	true	If inheritplot is not none: Determines if height scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, and height scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
param	Positive integer	1	For tables with filled data containing more than two parameters, param is the index indicating which values to use for fixing additional parameters. Additional parameters are ordered from left to right in the original table data.
plotdata	table manual	table	For tables with filled data containing more than two parameters, plotdata controls whether to use manual settings in the plot (table row, column, and so forth), or the filled settings in the table feature. In the latter case, the table's properties are automatically synchronized with the plot's properties.
preproc{x y data}	none linear	none	Use a linear preprocessing of the table data values, which you specify using the scaling and shift properties.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.

TABLE 7-96: VALID PROPERTY/VALUE PAIRS FOR TABLE SURFACE PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
scaling{x y data}	double	1.0	Real scaling value for data preprocessing
shift{x y data}	double	0.0	Real shift value for data preprocessing
showrowstep	Positive integer	1	When retrieving filled data using the <code>getTableData</code> method, <code>showrowstep</code> controls whether to leave all but every <code>showrowstep</code> input cells empty, for example, every third cell in the first column. This refers to the input column only. Only applied when <code>filterstringdata</code> is on.
smooth	none internal everywhere	internal	Smoothing settings
table	none table feature name	First available table	The table feature this plot refers to.
title	String	The auto-title.	The title to use when <code>titletype</code> is manual.
titletype	auto manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. manual if the manual title contribution should be used (the <code>title</code> property). none if no title contribution should be used.
xaxisdata	Parameters in table none	none	For tables with filled data, <code>xaxisdata</code> controls which parameter is used on the x-axis.
yaxisdata	Parameters in table none	none	For tables with filled data, <code>yaxisdata</code> controls which parameter is used on the y-axis. Cannot be the same parameter as in <code>xaxisdata</code> .
wireframe	on off	off	Whether to plot filled elements or only their edges,

SEE ALSO

[Table](#), [Table \(plot\)](#)

TimeAverage, TimeIntegral

Create a time average or time integral data set.

SYNTAX

```
model.result().dataset().create(<dtag>,"TimeAverage");
model.result().dataset().create(<dtag>,"TimeIntegral");
model.result().dataset(<dtag>).set(property, <value>);
```

DESCRIPTION

`model.result().dataset().create(<dtag>,"TimeAverage")` creates a time average data set with the name `<dtag>`.

`model.result().dataset().create(<dtag>,"TimeIntegral")` creates a time integral data set with the name `<dtag>`.

The time average and time integral data sets make it possible to compute time averages and time integrals of some time-dependent data in another data set, for example.

The following properties are available:

TABLE 7-97: VALID PROPERTY/VALUE PAIRS FOR TIME AVERAGE AND TIME INTEGRAL DATA SETS

PROPERTY	VALUE	DEFAULT	DESCRIPTION
data	none data set name	First compatible data set	The data set this feature refers to.
innerinput	all first last manual manualindices interp	all	How to input the solution to use. manual indicates that solnum is used. manualindices indicates that solnumindices is used. interp indicates that t is used.
minlen	nonnegative double	1e-4	Minimum interval length relative to the length of the time interval.
rtol	nonnegative double	1e-3	Relative tolerance.
solnum	Integer array	All solutions	The solutions to use for extrapolation.
solnumindices	String or integer array	Empty	An alternate way to specify the solutions to use, allowing you to enter, for example, range(1,1,20). Applicable when innerinput is manualindices.
t	double array	Empty	The times to use as intermediate layers in the extrusion.

Volume

Create a volume plot.

SYNTAX

```
model.result(<pgtag>).create(<ftag>,"Volume");
model.result(<pgtag>).feature(<ftag>).set(property, <value>);
model.result(<pgtag>).feature(<ftag>).run();
```

DESCRIPTION

model.result(<pgtag>).create(<ftag>,"Volume") creates a volume plot feature named <ftag> belonging to the plot group <pgtag>.

A volume plot shows an expression evaluated in all elements of a 3D volume.

The following properties are available:

TABLE 7-98: VALID PROPERTY/VALUE PAIRS FOR VOLUME PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
color	custom black blue cyan green magenta red white yellow	red	The uniform color to use. Active when coloring is uniform.
coloring	colortable uniform	colortable	How to color this surface
colorlegend	on off	on	Whether to show color legend, when coloring is set to colortable.
colortable	color table name	Rainbow	The color table to use when colormethod is set to colortable. See Color Tables for a list of color tables.
colortablerev	on off	off	Whether to reverse to color table when colormethod is set to colortable.

TABLE 7-98: VALID PROPERTY/VALUE PAIRS FOR VOLUME PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
colortablesym	on off	off	Whether to symmetrize the color range around 0 when colormethod is set to colortable.
const	String array of property/value pairs	Empty	Parameters to use in the expressions
customcolor	RGB-triplet	{1,0,0} or last used color.	The uniform color to use. Active when color is set to custom.
data	none parent data set name	parent	The data set this feature refers to
descr	String	Model-dependent.	The description of the expression in expr. Is used in the automatic title.
descriptionintitle	on off	on	Whether the title contribution should contain the description when titletype is custom.
elemfilter	random expression	random	If filteractive is on: The expression to use for filtering when only a subset of the elements are shown.
elemscale	double in [0,1]	1	The scale factor with which each element is scaled.
differential	on off	on	Whether the expression should be linearized at the linearization point. Applicable only if evalmethod is harmonic
evalmethod	linpoint harmonic lintotal lintotalavg lintotalrms lintotalpeak	harmonic	Applicable only for solutions with a stored linearization point. Controls if the linearization point, the perturbation, or a combination should be used when evaluating the expression.
expr	String	Model-dependent.	The expression to plot
expressionintitle	on off	off	Whether the title contribution should contain the expression when titletype is custom.
filteractive	on off	off	Whether to use element filtering
filterexpr	String	x	The expression to use for filtering when elemfilter is set to expression.
inheritcolor	Boolean	true	If inheritplot is not none: Determines if the color is inherited.
inheritdeformscale	Boolean	true	If inheritplot is not none: Determines if the deformation scale is inherited.
inheritplot	none plot name	none	The plot that color, color range, and deformation scale is inherited from.
inheritrange	Boolean	true	If inheritplot is not none: Determines if the color and data ranges are inherited.
interp	double array	Time corresponding to last selected solnum for transient levels.	The time to use, for transient levels. Available when data is not parent and the underlying data is transient.

TABLE 7-98: VALID PROPERTY/VALUE PAIRS FOR VOLUME PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
looplevel	array of non-negative integers and strings	Last solution for transient or parametric solution for each level. Otherwise first solution for each level.	The index of the solution to use, per level, or interp, but only for transient solutions. Available when data is not parent and the underlying data has multiple solutions.
outersolnum	non-negative integer	1	The index of the outer solutions to use. Applicable only for parametric sweep models.
prefixintitle	String	Empty	Added prefix to contribution to title.
rangecoloractive	on off	off	Whether to use the manual color range specified in rangecolormin and rangecolormax. The color range specifies the minimum and maximum value in the plotted colors. Default is the minimum and maximum data values.
rangecolormax	double	Plot-dependent	The maximum color range value. Active when rangecoloractive is on.
rangecolormin	double	Plot-dependent	The minimum color range value. Active when rangecoloractive is on.
rangedataactive	on off	off	Whether to use the manual data range specified in rangedatamin and rangedatamax. Values outside the data range is not plotted.
rangedatamax	double	Plot-dependent	The maximum data value. Active when rangedataactive is on.
rangedatamin	double	Plot-dependent	The minimum data value. Active when rangedataactive is on.
recover	off pprint ppr	off	The derivative recovery method (off, within domains, or everywhere).
refine	non-negative integer	1	The element refinement to use, if resolution is set to manual. Bear in mind that this is the refinement used for the base data set, so the number of elements in the model can increase radically if the plot uses, for example, a revolve data set.
resolution	norefine coarse normal fine finer extrafine custom	normal	Controls the plot's resolution. A finer setting results in a higher resolution by modifying the internally computed default refinement. Use custom to enter your own refinement in the refine property.
tetkeep	double in [0, 1]	1	The fraction of the elements to display
smooth	none internal everywhere	internal	Smoothing settings
solnum	non-negative integer	Last solution for transient or parametric solution. Otherwise first solution.	The index of the solution to use. Available when data is not parent and the underlying data has multiple solutions.

TABLE 7-98: VALID PROPERTY/VALUE PAIRS FOR VOLUME PLOTS

NAME	VALUE	DEFAULT	DESCRIPTION
solrepresentation	solnum solutioninfo	solutioninfo	Indicates which method of selecting solutions is active.
suffixintitle	String	Empty	Added suffix to contribution to title.
t	double	Time corresponding to last selected solnum.	The time to use, for transient problems. Available when data is not parent and the underlying data is transient.
timeinterp	on off	off	on if t is used to determine time steps, off if solnum is used.
title	String	The auto-title.	The title to use when titletype is manual.
titletype	auto custom manual none	auto	auto if the title contribution should be computed automatically, possibly using the group's customization. custom if the title contribution should be computed automatically, but customized. manual if the manual title contribution should be used (the title property). none if no title contribution should be used.
typeintitle	on off	on	Whether the title contribution should contain the type when titletype is custom.
unit	String	Model-dependent	The unit to use for the expression in expr. If the old unit is not valid when the expression changes, the unit property is reset to default.
unitintitle	on off	on	Whether the title contribution should contain the unit when titletype is custom.
wireframe	on off	off	Whether to plot filled elements or only their lines

Graphical User Interfaces

In this chapter you find reference information about functionality that is useful if you want to create your own Graphical User Interface (GUI) that calls the COMSOL[®] API. The Application Builder is an alternative tool for creating custom applications and user interfaces based on COMSOL models with the possibility to use the COMSOL API and Java[®] for enhanced and extended functionality. For more information, see the documentation for the Application Builder.

Using the functionality described in this guide, you can create custom GUIs that utilize COMSOL's modeling and analysis capabilities and create 1D, 2D, and 3D plots directly in your applications.

In this chapter:

- [Getting Started](#)
- [Example Graphical User Interface](#)
- [GUI Classes](#)

Getting Started

The COMSOL API is based on Java[®]. You must have access to a Java compiler to create and compile programs that can utilize the functionality provided by the COMSOL API.

You can either use a standard Java compiler or you can use an integrated development environment.

You can get a compiler from www.oracle.com. Go to that website and download a Java JDK that suits you. The version for Java SE is suitable for most uses. The Oracle website also has plenty of background information on the Java language. Especially the Java Tutorial can be recommended for users that have little or no prior experience in using the Java language.

It is highly recommended that you use an integrated development environment (IDE) for writing and compiling the Java programs. An IDE helps when writing the source code because it provides the user with code completion, syntax highlighting, and access to help text (JavaDoc). An IDE usually also provides an integrated debugger that makes it possible to run the program line by line in order to easier find any bugs in the programs while running.

One such IDE is Eclipse. Eclipse is free and can be downloaded from www.eclipse.org. On the download page, download the *Eclipse IDE for Java Developers*.

The following sections assume that you have basic knowledge of the Java language and knows how to, for example, compile a Java program.



See www.comsol.com/system-requirements for the supported Java versions.

Example Graphical User Interface

In this section:

- [Introduction](#)
- [Downloading Extra Material](#)
- [Creating the Code for the Model](#)
- [Construction of the Initial GUI with Graphics](#)
- [Handling of Progress Information](#)
- [Setting Up Inputs From the GUI to the Model](#)
- [Displaying Results in the GUI](#)
- [Other Details](#)

Introduction

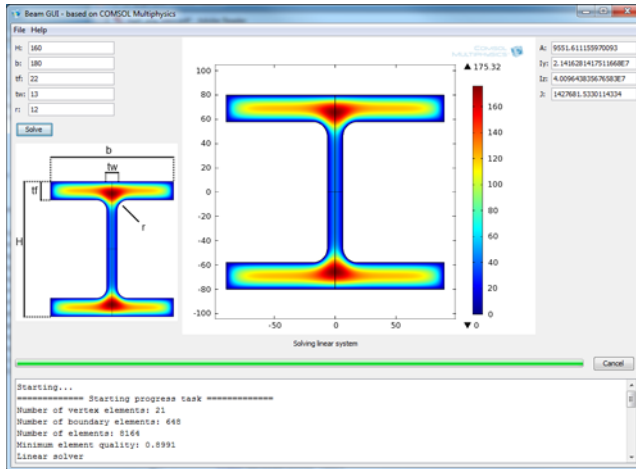
In the following sections a GUI is built based on a model of a cross section of a beam. Beam models are used within the field of structural mechanics, but the application area is not important to the given example. The example model itself solves a simple Poisson type of equation (heat transfer) to calculate the area and moment of inertia for the beam. These calculated values can then later be used as input parameters in a real beam model. A reference for the modeling method in this example can be found in *Foundations of Solid Mechanics* by Y.C. Fung.

The example includes:

- Save a model file for Java from an existing COMSOL Multiphysics model
- Integrate the Java file into a GUI built in Java[®]
- How to handle graphics in your own application
- How to handle progress information and, for example, stopping a solver
- How to hook up inputs in the GUI to the simulation
- How to show output from the simulation in the GUI
- Additional details such as adding a menu and icon to the application window

The example is based on the model stored in the `BeamModel1.mph` file. The windows includes a lot of detail. The goal of creating a user-defined GUI is to be able to reduce the amount of clutter on the screen by only allowing suitable settings to be viewed and changed. This allows users who are not experienced in modeling to benefit from models created by others.

The final user interface looks like this



In this window only the necessary settings are provided on the left side. These settings are used to change the dimensions of the geometry. After the model has been solved the results are shown in the right side and can be copied to another application or report for further use.

Downloading Extra Material

Layout managers are used in Java[®] applications to create appealing GUIs that support automatic resizing. This demo uses the MIG Layout Manager. It is a free Java layout manager that removes a lot of the pain of using layout managers in Java.

The MIG Layout manager is free and can be downloaded from www.miglayout.com.

Creating the Code for the Model

In the `demo` folder under the COMSOL Multiphysics installation directory there are some files that can be used for creating the demonstration example. The example models for this demonstration is placed in the `demo\api\beammodel` directory. There is a model file called `BeamModel.mph`. This file contains the model to use for the GUI. Open COMSOL and open the model.

Spend some time familiarizing yourself with the model. Note especially that the model has a set of parameters under **Global Definitions** that are used to update the dimensions of the geometry, and that there is a set of variables defined under **Model>Definitions>Variables** that are used to define the outputs from the simulations.

Although any parameter or setting in the model can be changed using the COMSOL API it is recommended that input and output data are well defined as shown in this model. Such definitions make it easier to follow the data flow in the model.

You can export a model file for Java from the COMSOL Desktop. Before exporting, it is worthwhile to make the model history compact. Doing so makes sure that the exported Java file only contains the necessary steps that are needed to reproduce the model.

Use **Compact History** to make the model history compact. Then save the Java file using **Save As**. Choose **Model file for Java (*.java)** from the **Save as type** list, and name the file `BeamModel.java`. Then click **Save**.



[Compacting the History](#) and [Creating a Copy Using Save As](#) in the *COMSOL Multiphysics Reference Manual*

Perform the following steps to set up Eclipse for handling your exported Java file and create a GUI:

- 1 Start Eclipse.
- 2 Create a new Java Project. Enter `BeamModelDemo` as the project name and click **Next**.
- 3 Go to the Libraries tab and click **Add External JARs**. Add all the JAR files placed in the `plugins` directory under the COMSOL Multiphysics installation directory (typically `C:\Program Files\COMSOL\COMSOL52\Multiphysics\plugins`). This allows Eclipse to find the definitions of the classes used by the COMSOL API and to run the code. In addition add the external Jar file `miglayout-4.0-swing.jar` (the numbers can be different for the file you downloaded). Click **Finish**.
- 4 Drag and drop your exported Java file to the `src` folder of your Eclipse project. Choose Copy files when Eclipse asks you and click OK.
- 5 Open the copied Java file by double-clicking it and navigate to the Main method. Remove the line in the Main method. A main method in this file is not required.
- 6 Navigate to the run method. The run method contains all settings necessary to set up the model and solve it. It even contains the definition of the Plot that is displayed in the application.
- 7 Remove this line that says `model.sol("sol1").runAll()`; The model does not require solving when setting it up.
- 8 In order to be able to extract numerical results from the model add some Global nodes to the model. The Global nodes are only available in the COMSOL API and thus cannot be added using the COMSOL Desktop. Add the following lines to the bottom of the `run()` method; just above the `return` statement.

```
model.result().numerical().create("glA", "Global");
model.result().numerical("glA").set("expr", "A");
model.result().numerical().create("glIy", "Global");
model.result().numerical("glIy").set("expr", "Iy");
model.result().numerical().create("glIz", "Global");
model.result().numerical("glIz").set("expr", "Iz");
model.result().numerical().create("glJ", "Global");
model.result().numerical("glJ").set("expr", "J");
```

The file `BeamModel.java` now contains all the necessary settings to set up a model. This file is not changed again.

Add the `main()` method of the program that opens the graphics window (frame) and shows the model.

- 1 Select **New Java Class**. Name the new class `BeamModelDemo`. Select that this class should have a `public static void main(String[] args)` method. Click **Finish**.
- 2 Add a Model field to the `BeamModelDemo` class by adding these lines to the top of the class

```
private JFrame frame;
private Model model;
```

Eclipse might complain that `JFrame` and `Model` are unknown at this time, and you have to add import statements in order to resolve the names. For the `Model` variable it is the `com.comsol.model` package that you should import. Eclipse is helpful and can provide such import statements automatically if you point to the offending new class name and press `Ctrl+I`. Throughout this demonstration example, add new classes for which such import statements have to be added.

- 3 Navigate to the main method and edit it such that it contains these lines

```
public static void main(String[] args) {
    BeamModelDemo demo = new BeamModelDemo();
    demo.init();
    demo.start();}
```

- 4 Create an `init` method in `BeamModelDemo`. It should only contain a single line, and the method should look like this:

```

public void init() {
    ModelUtil.initStandalone(true);
}

```

- 5 Create the start method. It is the main method to set up a model and the GUI used to display it. This method is updated frequently when setting up this demonstration model.

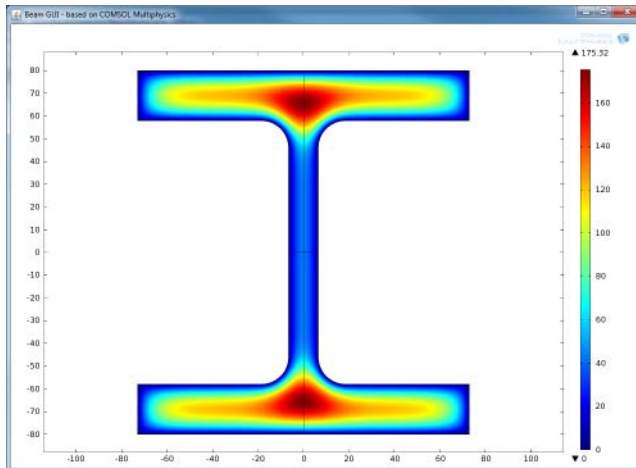
```

public void start() {
    frame = new JFrame("Beam GUI - based on COMSOL Multiphysics");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(1000, 730);
    JPanel mainPanel = new JPanel();
    frame.getContentPane().add(mainPanel);
    mainPanel.setLayout(new BorderLayout());
    SwingGraphicsPanel graphicsPanel =
        new SwingGraphicsPanel("window1", "Window1");
    mainPanel.add(graphicsPanel, BorderLayout.CENTER);
    frame.setVisible(true);

    model = BeamModel.run();
    model.sol("sol1").runAll();
    model.result("pg1").set("window", "window1");
    model.result("pg1").run();
}

```

- 6 Right-click the BeamModelDemo.java file in the **Package Explorer** and select **Run as>Run configuration**.
- 7 Select the **Arguments** tab. Add the following line to the **VM arguments** text box:
-XX:MaxPermSize=256m
- 8 Select the **Environment** tab. Click the **New** button. Use the **Name** PATH (on Windows), LD_LIBRARY_PATH (on Linux) or DYLD_LIBRARY_PATH (on Mac) and enter the following text in **Value**: <comsolinstalldir>/lib/<platformname> where <comsolinstalldir> is the directory where COMSOL Multiphysics is installed and <platformname> is one of win64/glnxa64/maci64 depending on your platform. Click **Apply**.
- 9 Click **Run**. The application window opens and a COMSOL graphics panel displays. After several seconds, the model is solved and the 2D graphics with the result are presented.



The application window is missing some information and some ways to control the simulation.

Handling of Progress Information

It is possible to create a monitor for the progress of the solver. This monitor can also be used to cancel long running simulations if desired.

The progress information is made available using two different classes: `SwingProgressPanel` is used to display the progress in the GUI, and `SwingDemoProgressContext` is used to handle the communication between COMSOL Multiphysics and the areas the application that needs information about progress.

`SwingDemoProgressContext` extends `SwingProgressContext`, which is described in the reference section at the end of this chapter.

Both classes are added by copying two files instead of writing them from scratch.

- 1 Use the mouse to drag and drop these files to the src folder shown in the **Package Explorer** in Eclipse: `SwingDemoProgressContext.java` and `SwingProgressPanel.java`.

- 2 Open the `BeamModelDemo.java` file and navigate to the start method.

- 3 Add these lines before the call to `frame.setVisible`

```
SwingProgressPanel progressPanel = new SwingProgressPanel();
mainPanel.add(progressPanel, BorderLayout.PAGE_END);
progressPanel.updateProgressLog("Starting\n");
ProgressWorker.setContext(new SwingDemoProgressContext(progressPanel));
```

- 4 These lines are added to the progress panel to the bottom of the main window and the context is set up between COMSOL and the user application.

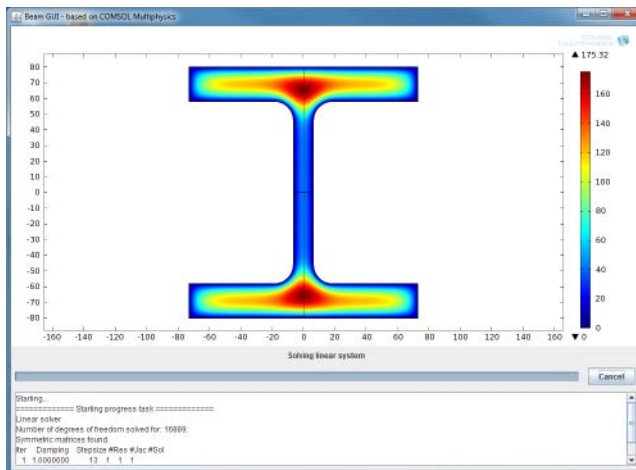
- 5 In order to show any progress in the GUI, the solver must not run in the main thread. This means that a separate thread for the solver is created. Create the following solve method by removing the corresponding lines from the start method:

```
private void solve() {
    ProgressWorker.run(new Runnable() {
        public void run() {
            model.sol("sol1").runAll();
            model.result("pg1").set("window", "window1");
            model.result("pg1").run();
        }
    });
}
```

- 6 Remember to add a call to the solve method at the end of the start method.

- 7 Run the application by right-clicking `BeamModelDemo.java` in the Package Explorer and select **Run as>Java Application**.

- 8 The application window opens and shows progress information while starting and solving.



Setting Up Inputs From the GUI to the Model

To add some more dynamics to the application, add a method of inputting parameters to the model and create a Solve button.

The input data is provided by fields in the right side of the window.

Add these fields to the top of the class definition of the `BeamModelDemo` class:

```
    JTextField editH;  
    JTextField editb;  
    JTextField edittf;  
    JTextField editr;  
    JTextField edittw;
```

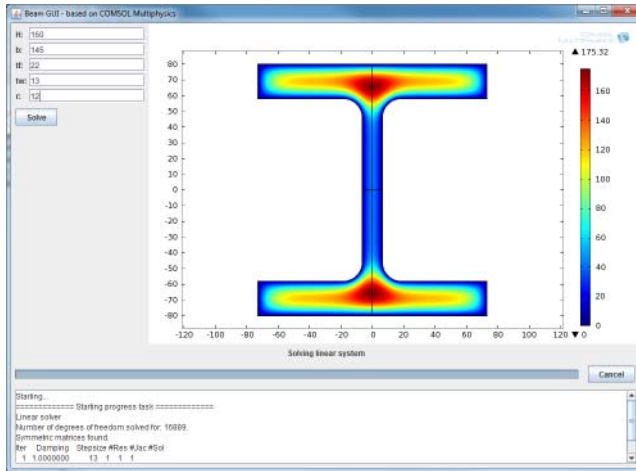
A method `leftPanel` is created that sets up the various components.

```
private JPanel leftPanel() {  
    MigLayout layout = new MigLayout("wrap 2");  
    JPanel panel = new JPanel(layout);  
  
    JLabel label = new JLabel("H:");  
    panel.add(label);  
    editH = new JTextField(16);  
    editH.setText("160");  
    panel.add(editH);  
  
    label = new JLabel("b:");  
    panel.add(label);  
    editb = new JTextField(16);  
    editb.setText("145");  
    panel.add(editb);  
  
    label = new JLabel("tf:");  
    panel.add(label);  
    edittf = new JTextField(16);  
    edittf.setText("22");  
    panel.add(edittf);  
  
    label = new JLabel("tw:");  
    panel.add(label);  
    edittw = new JTextField(16);  
    edittw.setText("13");  
    panel.add(edittw);  
  
    label = new JLabel("r:");  
    panel.add(label);  
    editr = new JTextField(16);  
    editr.setText("12");  
    panel.add(editr, "wrap 10px");  
  
    JButton solveButton = new JButton("Solve");  
    panel.add(solveButton, "span, wrap 10px");  
  
    solveButton.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            solve();  
        }  
    });  
  
    return panel;  
}
```

Add a call to `leftPanel` in the start method just before the definition of the `graphicsPanel` variable. Also comment out the call to the solve method at the end of the start method. Solving is handled manually by clicking the Solve button.

```
JPanel panel = leftPanel();
mainPanel.add(panel, BorderLayout.LINE_START);
```

Run the application by, for example, pressing Ctrl+F11



It is now possible to re-solve the model by pressing the **Solve** button, but the fields still needs to be hooked into the model.

- 1 Navigate to the solve method in the `BeamModelDemo.java` file. Add the following lines to the top of the method:

```
model.param().set("H", editH.getText()+"[mm]");
model.param().set("b", editb.getText()+"[mm]");
model.param().set("tw", edittw.getText()+"[mm]");
model.param().set("tf", edittf.getText()+"[mm]");
model.param().set("r", editr.getText()+"[mm]");
```

- 2 Save the file and start the application again.
- 3 Try to change the H parameter to, for example, 200 and press the **Solve** button.
- 4 The new model geometry and new simulation results display.

Displaying Results in the GUI

The aim of the GUI is to provide calculations of areas and moments of inertia based on the model. These results are added in a panel to the right of the graphics panel.

- 1 Add these fields to the top of the `BeamModelDemo` class file:

```
TextField editA;
TextField editIy;
TextField editIz;
TextField editJ;
```

- 2 Add a method `rightPanel` at the end of the `BeamModelDeom.java` that contains the output. Choose `TextFields` to display the result. It is possible to copy text from these fields for use in other applications.

```
private JPanel rightPanel() {
    MigLayout layout = new MigLayout("wrap 2");
    JPanel panel = new JPanel(layout);

    JLabel label = new JLabel("A:");
```

```

panel.add(label);
editA = new JTextField(16);
panel.add(editA);

label = new JLabel("Iy:");
panel.add(label);
editIy = new JTextField(16);
panel.add(editIy);
label = new JLabel("Iz:");
panel.add(label);
editIz = new JTextField(16);
panel.add(editIz);
label = new JLabel("J:");
panel.add(label);
editJ = new JTextField(16);
panel.add(editJ, "wrap 10px");

return panel;
}

```

- 3 A small utility method is required in order to extract numerical data from the Global nodes. Add this method after the `rightPanel` method:

```

private String getScalar(NumericalFeature num) {
    double[][] array = num.getData(0);
    double A = array[0][0];
    return Double.toString(A);
}

```

- 4 Add a few lines to extract the numerical results from the model after is being solved and show the numbers in the main window. Add the following lines to the solve method just after the call to the run method:

```

editA.setText(getScalar(model.result().numerical("glA")));
editIy.setText(getScalar(model.result().numerical("glIy")));
editIz.setText(getScalar(model.result().numerical("glIz")));
editJ.setText(getScalar(model.result().numerical("glJ")));

```

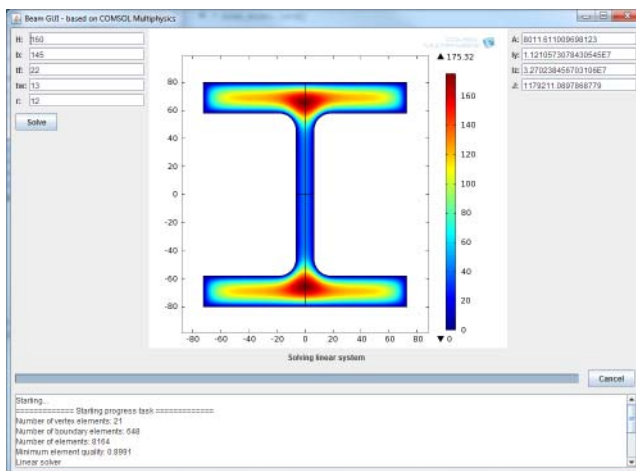
- 5 Add code for producing the `rightPanel` to the `start` method just below the addition of the `graphicsPanel`:

```

panel = rightPanel();
mainPanel.add(panel, BorderLayout.LINE_END);

```

- 6 Run the application again.



The application by now is able to accept input from the user, simulate a model, and display results graphically as well as numerical results.

In order to finalize the model, add some additional features to the application. At the end, there is a short description of things that remains to be done.

ADDING A MENU

A menu is usually added.

- 1 Add implements `ActionListener` to the definition of the `BeamModelDemo` class such that the first line of the class definition reads

```
public class BeamModelDemo implements ActionListener {
```

- 2 Add a method that handles the event when the user performs actions with the menus. Here an action is added that can be used to exit the application and an about box is added that utilizes one of the `JOptionPane` dialog boxes.

```
public void actionPerformed(ActionEvent e) {
    String ac = e.getActionCommand();
    if (ac.equals("exit")) {
        System.exit(0);
    }
    else if (ac.equals("about")) {
        JOptionPane.showMessageDialog(frame,
            "Beam GUI Example\n"+
            "Simple DEMO example\n"+
            "Copyright 2011-2013",
            "About",
            JOptionPane.DEFAULT_OPTION);
    }
}
```

- 3 Add a method that defines the menu and menu items. The code adds a **File** and a **Help** menu where the exit and about actions are placed as menu items.

```
private JMenuBar menu() {
    JMenuBar menubar = new JMenuBar();

    JMenu menu = new JMenu("File");
    menubar.add(menu);

    JMenuItem item = new JMenuItem("Exit");
    item.setActionCommand("exit");
    item.addActionListener(this);
    menu.add(item);

    menu = new JMenu("Help");
    menubar.add(menu);

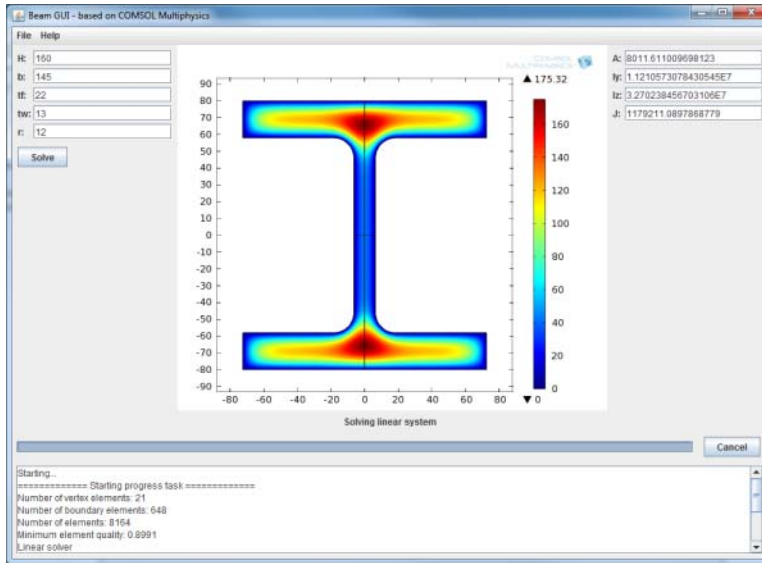
    item = new JMenuItem("About");
    item.setActionCommand("about");
    item.addActionListener(this);
    menu.add(item);

    return menubar;
}
```

- 4 Add a call to `menu` in the start method just before the call to `setVisible`:

```
JMenuBar menubar = menu();
frame.setJMenuBar(menubar);
```

5 Start the application



ADDING AN ICON AND AN IMAGE

An application that has to be used by other people should have appealing appearance and graphics and a suitable icon. For this application add the COMSOL logo to the window. Choose any icon you want to use for your own application.

- 1 Add a `setIcon` method to the `BeamModelDemo` class

```
private void setIcon(String filename) {
    BufferedImage img;
    img = null;
    try {
        img = ImageIO.read(new File(filename));
    } catch (IOException e) {
        return;
    }
    frame.setIconImage(img);
}
```

- 2 Add this line to the top of the start method (right after the call to `setSize`)

```
setIcon("comsolicon.png");
```

- 3 Place the icon file `comsolicon.png` in your workspace directory for this project. Right-click `BeamModelDemo.java` in the **Package Explorer** and choose **Properties** in order to determine the location of this directory.

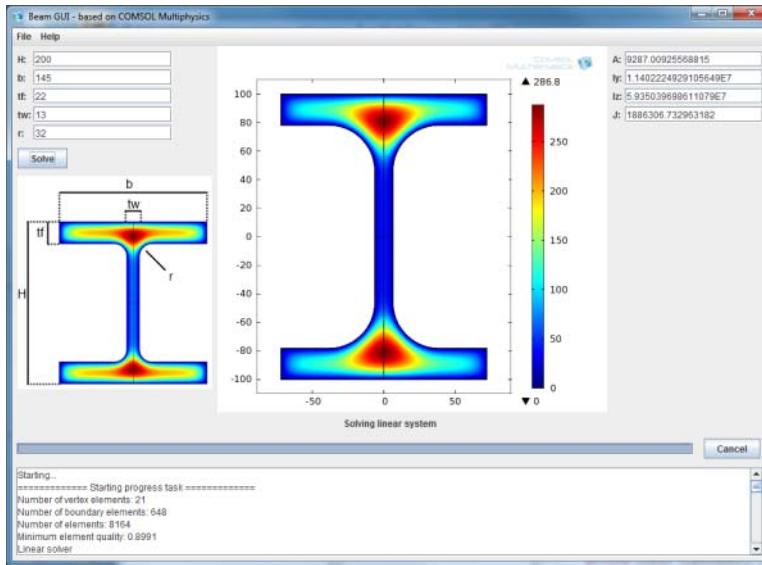
- 4 It is convenient to have an image that describes the parameters used in the panel on the left side of the window. Start by adding the `ImageComponent.java` file to the project by dragging it to the `src` folder in the **Package Explorer**.

- 5 Add these lines to the `leftPanel` method just before the line with `return panel;`

```
ImageComponent img =
    new ImageComponent("beam_dim_small.png");
panel.add(img, "span");
```

- 6 Place the `beam_dim_small.png` file in the workspace directory for this project.

7 Start the application.



LOOK AND FEEL

The appearance of a Java[®] Swing application by default does not look like other applications that are written specifically for the platform you are running on. You can improve the appearance by setting the Look and Feel for the Java application:

- 1 Add a `lookandfeel` method to the `BeamModelDemo` class:

```
private void lookandfeel() {
    try {
        UIManager.setLookAndFeel(
            UIManager.getSystemLookAndFeelClassName());
    }
    catch (Exception e) {
    }
}
```

- 2 Add a call to `lookandfeel` at the very top of the `start` method

```
lookandfeel();
```

- 3 Run the application. The application demonstration example is now complete.

FINISHING NOTES

The demonstration example is not an application that is ready to be sent to customers. For further development of the application, consider adding the following functionality:

- Error handling such that the application becomes insensitive to user's incorrect input and such that proper error messages are displayed in the event of an error or malfunction.
- For long running jobs, show an hour glass mouse pointer in order to show the user that a time consuming task has started.
- The model object keeps the history that is recorded every time the model object is changed such that a model file for Java later can be saved that includes all actions on the object. If that is not required the history generation can be switched off, which saves memory.
- It is possible to have more than one model in an application. For example, it is possible to add a 3D beam model that utilizes the values calculated by the demonstration application.

- The graphics window can show 1D, 2D, and 3D graphics, so any results obtained in COMSOL Multiphysics can be shown in your own applications.
- Code can be added that remembers the choices made in the program. For example, to remember the location of the main window and the values entered in the fields.
- On-line help is missing.
- Calculated values could be copied to the clipboard or saved as a text file for easy sharing of the values.

GUI Classes

The following classes are available for working with graphical user interfaces:

- [ProgressContext](#)
- [ProgressWorker](#)
- [SWTGraphicsPanel](#)
- [SwingGraphicsPanel](#)

ProgressContext

Receive progress and log information from lengthy tasks and cancel them.

SYNTAX

```
progressUpdated(double progress);
progressDescriptionUpdated(String description);
progressLogUpdated(String message);
started();
finished(Throwable t);
cancel();
isCanceled();
```

DESCRIPTION

`ProgressContext` is the base class that you can extend to create a class that handles progress and log information. There are 5 different methods that you can override to receive calls when various events happen. The calls to these methods display on the background thread that the task is running on. The default implementation in `ProgressContext` for these methods does nothing.

If you are creating a GUI in SWT the class `SWTProgressContext` is also available to extend from. It receives the method calls for the methods overridden on the SWT event dispatching thread. This is convenient because calls to update SWT widgets must be made from that thread.

`progressUpdated(progress)` is the method to override if you want to receive information when the current progress is updated. `progress` is a value between 0 and 1.

`progressDescriptionUpdated(description)` is called when the description for what progress task that is currently running is changed.

`progressLogUpdated(message)` is called when a new line is added to the log of messages. This is mostly used for log information from the solvers.

`started()` is called when the progress task is about to start.

`finished(t)` is called when the progress task is finished. If an exception occurred while the progress task was running it is non-null. You can use the `isCanceled` method to check if the progress task was canceled.

`cancel()` is the method to call if you want to request cancellation of the currently running progress task. You typically call it from a listener for a cancel button in your GUI.

`isCanceled()` returns true if the `cancel` method has been called.

ProgressWorker

Run lengthy tasks on a separate thread and report progress and log information.

SYNTAX

```
ProgressWorker.setContext(ProgressContext context);  
ProgressWorker.run(Runnable run);  
ProgressWorker.run(Runnable run, ProgressContext context);
```

DESCRIPTION

`ProgressWorker.setContext(context)` sets that the `ProgressContext` context should receive progress information when the `run` method in `ProgressWorker` is called.

`ProgressWorker.run(runnable)` calls the `run` method on `runnable` on a separate thread and reports progress back to the registered `ProgressContext`.

`ProgressWorker.run(runnable, context)` calls the `run` method on `runnable` on a separate thread and reports progress back to the `ProgressContext` context.

SWTGraphicsPanel

Create an SWT Composite (panel) that can be used to plot COMSOL Multiphysics graphics into a custom GUI.

SYNTAX

```
SWTGraphicsPanel(Composite parent, String tag, String description);
```

DESCRIPTION

`SWTGraphicsPanel(parent, tag, description)` creates an SWT Composite that can be used in a GUI created using the Standard Widget Toolkit (SWT). `parent` is the SWT composite that is the parent of the panel in the GUI. `tag` is a unique tag used to identify the panel. The "window" property of a plot group in the model object should be set to this tag to plot into the panel.

SwingGraphicsPanel

Create an Swing JPanel that can be used to plot COMSOL Multiphysics graphics into a custom GUI.

SYNTAX

```
SwingGraphicsPanel(String tag, String description);
```

DESCRIPTION

`SWTGraphicsPanel(tag, description)` creates a Swing JPanel that can be used in a GUI created using Swing. `tag` is a unique tag used to identify the panel. The "window" property of a plot group in the model object should be set to this tag to plot into the panel.

This panel supports 1D, 2D, and 3D graphics.

The COMSOL File Formats

This chapter describes the COMSOL file formats including COMSOL Multiphysics[®] files in binary format and text format.

In this chapter:

- [File Formats](#)
- [Data Formats](#)
- [Color Tables](#)
- [Binary Data Files and Text Data Files](#)
- [Serializable Types](#)
- [Examples of the Serialization Format](#)

File Formats

The following table shows the file types that COMSOL Multiphysics can read and write and provides pointers to documentation:

TABLE 9-1: FILE FORMATS SUMMARY

FILE FORMAT	EXTENSION	PRODUCT	READ	WRITE	DOCUMENTATION
COMSOL Model File	mph	MPH	Yes	Yes	N/A
Model files for Java	java	MPH	No	Yes	About the COMSOL Model File Formats
Compiled model files for Java (class files)	class	MPH	Yes	No	N/A
Model files for MATLAB (M-files)	m	LLML	No	Yes	About the COMSOL Model File Formats
Binary Data File	mphbin	MPH	Yes	Yes	Binary Data Files and Text Data Files
Text Data File	mphtxt	MPH	Yes	Yes	Binary Data Files and Text Data Files
Spreadsheet file	txt	MPH	Yes	Yes	Spreadsheet Data Format
Grid file	txt	MPH	Yes	Yes	Grid Data Format
Sectionwise file	txt	MPH	Yes	Yes	Sectionwise Data Format
Continuous color tables		MPH	Yes	No	Continuous Color Tables
Discrete color tables		MPH	Yes	No	Discrete Color Tables
Microsoft Excel [®] files	xlsx, xls, xlsb, xlsxm	LLEXCEL	Yes	Yes	Supported Microsoft Excel File Types

Product keys:

- MPH: COMSOL Multiphysics
- LLML: COMSOL LiveLink[™] for MATLAB[®]
- LLEXCEL: COMSOL LiveLink[™] for Excel[®]

Data Formats

The data formats in COMSOL Multiphysics are used for exporting results data to file as well as representing input to interpolation functions in COMSOL. For all data formats, the exported file can contain a number of header rows starting with %, which contain information about the model and the exported data.

In this section:

- [Spreadsheet Data Format](#)
- [Grid Data Format](#)
- [Sectionwise Data Format](#)



In the *COMSOL Multiphysics Reference Manual*:

- [Examples of Spreadsheet, Sectionwise, and Grid File Formats](#)
- [Exporting Data and Images](#)

Spreadsheet Data Format

This data format is used for importing unstructured data and exporting results data:

TABLE 9-2: DATA

SECTION	DESCRIPTION
Data	Data values separated by spaces

Each row of the file defines the coordinates and data values for the function in one point. A file used to define a function of three variables can begin as follows:

```
0 0 0.12 0.34
0 1 0.52 1.50
1 0 0.67 0.91
...
```

The first variable (input argument) appears in the first (leftmost) column, the second variable in the second column, the third variable in the third column, and the function values in the fourth (rightmost) column. The variables can be any function inputs. For space-dependent functions $f(x, y, z)$ they are the x -, y -, and z -coordinates.

It is possible to define several functions in one file by providing more than one data column after the input variables (which, in many cases, are coordinates). When exporting several expressions to a file, COMSOL Multiphysics generates files with this structure.



You can use the % character to indicate that a row contains comments and not data.

Grid Data Format



The grid data format can only be used for import of data.

The following table shows the format for results data stored as grid points and corresponding data values:

TABLE 9-3: GRID, DATA

SECTION	NUMBER OF ROWS	DESCRIPTION
% Grid	1–3	x grid points separated by spaces y grid points separated by spaces (optional) z grid points separated by spaces (optional)
% Data	Number of y grid points (2D) or number of y grid points times number of z grid points (3)	Data values separated by spaces

Each row contains values for different x grid points for fixed values of y and z . The rows first increase the y grid value and then the z grid value. The grid points can also represent another independent variable that the data values depend on. For example, the “grid points” can be temperature values and the data values the thermal conductivity at these temperatures.



It is important to use a comment line starting with % to separate the grid points or other interpolation points and the data values that are associated with these coordinates or interpolation points.

It is possible to include more than one function in the file as long as a % Data header separates them one from the other.



In the *COMSOL Multiphysics Reference Manual*:

- [Interpolation](#)
- [Examples of Spreadsheet, Sectionwise, and Grid File Formats](#)
- [Exporting Data and Images](#)



Rock Fracture Flow: Application Library path **COMSOL_Multiphysics/Geophysics/rock_fracture_flow**

Sectionwise Data Format

The following table shows the format for results data stored as node coordinates, elements, and corresponding data values:



TABLE 9-4: NODES, ELEMENTS, DATA

SECTION	NO. COLUMNS	DESCRIPTION
%Coordinates	1–3	One to three columns containing x , y (optional), and z (optional)

TABLE 9-4: NODES, ELEMENTS, DATA

SECTION	NO. COLUMNS	DESCRIPTION
%Elements	3 (2D), 4 (3D)	Triangulation where each row contains the row indices of the points in the Coordinates section that make up a single element—triangular in 2D, tetrahedral in 3D
%Data	1	Column of data values

This format can also be used to import data for unstructured interpolation. It has the advantage over the [Spreadsheet Data Format](#) in that it also contains the exact mesh used to perform the interpolation.

	Duplicate values (that is, evaluations with the same coordinates and the same values of the evaluated expressions) are removed before the sectionwise data is exported to file.
	<p>In the <i>COMSOL Multiphysics Reference Manual</i>:</p> <ul style="list-style-type: none"> • Examples of Spreadsheet, Sectionwise, and Grid File Formats • Exporting Data and Images

Supported Microsoft Excel File Types

If your license includes the LiveLink™ for Excel®, you can import data to COMSOL from files with the following Excel® formats:

- For a client with Excel: Excel, Excel 97, Excel binary, and Excel macro
- For a client without Excel: Excel and Excel macro

Color Tables

Color table files are used when plotting. When COMSOL Multiphysics is loaded, it reads all files in the following directories:

- `data/colortables/` in the directory where COMSOL is installed.
- The user settings directory `.comsol/v51` under your local home directory.

The files that adhere to the format specified in this section are made available as color tables.

In this section:

- [Continuous Color Tables](#)
- [Discrete Color Tables](#)

Continuous Color Tables

An example of a color table that defines a transition from blue to red:

```
% Continuous
0 0 1
1 0 0
```

Each line, from left to right, contains the red, green, and blue components (RGB) of a color. The components are floating-point values between 0 and 1. The table can contain an arbitrary number of colors.

Each color can contain an optional fourth component, the length of the interval between the two colors. For N colors, there are $N-1$ intervals. An example of a color scale that defines a sharp transition from blue to magenta followed by a slow transition from magenta to red:

```
% Continuous
0 0 1 1
0.5 0 0.5 10
1 0 0
```

Discrete Color Tables

An example of a color table for which the lower half of the legend is blue and the upper half is red:

```
% Discrete
0 0 1
1 0 0
```

Each color can contain an optional fourth component, the length of the interval occupied by that color. For N colors, there are N intervals. An example of a color scale that defines a small blue interval, a long green interval, and a medium blue interval:

```
% Discrete
0 0 1 1
0 1 0 10
1 0 0 5
```

Binary Data Files and Text Data Files

A COMSOL Multiphysics file is used to store COMSOL data. The format is suitable for exchange of mesh or CAD data between COMSOL and other software systems. It is possible to save a COMSOL Multiphysics file in a text file format, using the extension .mphtxt, or a binary file format, using the extension .mphbin. The file formats contain the same data in the same order.

In this section:

- [File Structure](#)
- [Records](#)
- [Terminology](#)
- [Text File Format](#)
- [Binary File Format](#)

File Structure

The COMSOL Multiphysics file format has a global version number, so that it is possible to revise the whole structure. The first entry in each file is the file format, indicated by two integers. The first integer is the major file version and the second is referred to as the minor file version. For the current version, the first two entries in a file is 0 1.

The following sections describe the file structure of the supported version.

FILE VERSION 0.1

After the file version, the file contains three groups of data:

- A number of *tags* stored as strings, which gives an identification for each *record* stored in the file.
- A number of *types*, which are strings that can be used in serializing the object. The tag should be used as an indication of the contents of the file, but can also be an empty string.
- The *records* containing the serialized data in the file.

Example When using `model.mesh(<tag>).export(<filename>)` or **Mesh>Export to File** to save a COMSOL mesh, the tag equals the variable name (m1), the type is set to obj (but this is not used), and the record contains the serialization of the mesh object, including point coordinates and element data of the mesh. See [Examples of the Serialization Format](#) for more examples of COMSOL Multiphysics text files.

```
# Created by COMSOL Multiphysics Fri Aug 26 14:19:54 2005

# Major & minor version
0 1
##### Tags
1 # number of tags
2 m1

##### Types
1 # number of types
3 obj

##### Records

# A planar face object

0 0 1
```

```
4 Mesh # class
...
```

Records

The record contains the serialization data in the file and additional information on how to process the serialized data. It also has a version number.

The record is a wrapper for *serializable types* stored in the file. The reason for having this wrapper is to be able to use a version number, so that the serialization can be revised in the future while maintaining backward compatibility.

The following sections describe the format of the supported version:

RECORD VERSION 0

This record is a wrapper for *serializable types* stored in the file. The following table contains the attributes of the records:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
Integer		Version
Integer		Not used
Integer	type	Serialization type, 1 for Serializable
Serializable	obj	If type equals 1, this field follows

Serialization type 1 indicates that the following field is a subtype to [Serializable](#). COMSOL uses type equal to 0 internally, but such files are only used for temporary purposes.

Terminology

The following data types are used in the serialization:

- *Boolean* refers to an 8-bit signed character which must be 0 or 1.
- *Character* refers to an 8-bit signed character.
- *Integer* refers to a 32-bit signed integer.
- *Double* refers to a 64-bit double.

Matrices are stored in row-major order. In this documentation brackets are used to indicate a matrix. Hence, `integer[3][4]` means that 12 integers representing a matrix are store in the file. The first three entries corresponds to the integers in the first row of the matrix, and so on.

Text File Format

COMSOL Multiphysics text file, using the file extension `.mptxt`, are text files where values are stored as text separated by whitespace characters.

Lexical conventions:

- Strings are serialized as the length of the string followed by a space and then the characters of the string, for example, “6 COMSOL”. This is the only place where whitespace matters.
- The software ignores everything following a `#` on a line except when reading a string. This makes it possible to store comments in the file.

Binary File Format

COMSOL Multiphysics binary file, using the extension .mphbin, are binary files with the following data representation:

- Integers and doubles are stored in little-endian byte order.
- Strings are stored as the length of the string (integer) followed by the characters of the string (integers).

Serializable Types

In this section:

- [Attribute](#)
- [BezierCurve](#)
- [BezierMfd](#)
- [BezierSurf](#)
- [BezierTri](#)
- [BSplineCurve](#)
- [BSplineMfd](#)
- [BSplineSurf](#)
- [Ellipse](#)
- [Geom0](#)
- [Geom1](#)
- [Geom2](#)
- [Geom3](#)
- [Manifold](#)
- [Mesh](#)
- [MeshCurve](#)
- [MeshSurf](#)
- [Plane](#)
- [PolChain](#)
- [Serializable](#)
- [Solution](#)
- [Straight](#)
- [Transform](#)
- [VectorDouble](#)
- [VectorInt](#)
- [VectorString](#)

Attribute

SUPPORTED VERSIONS

0

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	n	Number of attribute fields
Serializable[n]		Each entity field is stored as a serializable

DESCRIPTION

An `Attribute` is a general purpose class from which different subtypes can be derived. Each such subtype should then be serialized using the serialization of the `Attribute` class, which means that all that it should add to the serialization is the version number.

Attributes are used in COMSOL Multiphysics for internal purposes, and these attributes are not documented. However, because `Attribute` has a strict serialization structure, the serialization of these attributes is well documented.

EXAMPLE

This is a serialization of an attribute used internally in COMSOL Multiphysics. It is serialized as a vector of integers.


```

11 AssocAttrib # class
0 0 # version
1 # nof attribute fields
9 VectorInt # class
18 3 2 2 2 1 0 1 1 1 1 1 1 0 1 1 1 1 0

```

BezierCurve

SUPPORTED VERSIONS

0

SUBTYPE OF

[BezierMfd](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	DESCRIPTION
integer	Version
BezierMfd	Parent class containing common data

DESCRIPTION

A rational Bézier curve is a parameterized curve of the form

$$\mathbf{b}(t) = \frac{\sum_{i=0}^p \mathbf{b}_i w_i B_i^p(t)}{\sum_{i=0}^p w_i B_i^p(t)}, \quad 0 \leq t \leq 1$$

where the functions

$$B_i^p(t) = \binom{p}{i} t^i (1-t)^{p-i}$$

are the *Bernstein basis* functions of *degree* p , $\mathbf{b}_i = (x_1, \dots, x_n)$ are the control vertices of the n -dimensional space, and w_i are the weights, which should always be positive real numbers to get a properly defined rational Bézier curve. A rational Bézier curve has a direction defined by the parameter t .

EXAMPLE

The following illustrates a linear Bézier curve.

```

11 BezierCurve # class
0 0 # version
2 # sdim
0 2 1 # transformation
1 0 # degrees
2 # number of control points
# control point coords and weights
0 0 1
1 1 1

```

SEE ALSO

[BSplineCurve](#)

SUPPORTED VERSIONS

0

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation class
integer	m	Degree in first parameter
integer	n	Degree in second parameter
integer	k	Number of control points
double[k][d+1]	P	Matrix of control points with the weights in the last column

DESCRIPTION

The **BezierMfd** type is the abstract base class for the different type of Bézier surfaces and curves that are supported. These can all be represented in using the generalized equation

$$\mathbf{S}(s, t) = \frac{\sum_{i=0}^m \sum_{j=0}^n \mathbf{b}_{i,j} w_{i,j} B(s, t)}{\sum_{i=0}^m \sum_{j=0}^n w_{i,j} B(s, t)}$$

where B are functions as described in the respective entry, and \mathbf{b} are the control point coordinates in **P** and w are the weights stored in the last column of **P**.

SEE ALSO

[BSplineMfd](#)

SUPPORTED VERSIONS

0

SUBTYPE OF

[BezierMfd](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	DESCRIPTION
integer	Version
BezierMfd	Parent class containing common data

DESCRIPTION

A rectangular rational Bézier patch of degree p -by- q is described by

$$\mathbf{S}(s, t) = \frac{\sum_{i=0}^p \sum_{j=0}^q \mathbf{b}_{i,j} w_{i,j} B_i^p(s) B_j^q(t)}{\sum_{i=0}^p \sum_{j=0}^q w_{i,j} B_i^p(s) B_j^q(t)}, \quad 0 \leq s, t \leq 1,$$

where B_i^p and B_j^q are the Bernstein basis functions of degree p and q , respectively, as described in the entry of [BezierCurve](#). This surface description is called rectangular because the parameter domain is rectangular; that is, the two parameters s and t can vary freely in given intervals.

SEE ALSO

[BSplineSurf](#), [BezierTri](#)

BezierTri

SUPPORTED VERSIONS

0

SUBTYPE OF

[BezierMfd](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	DESCRIPTION
integer	Version
BezierMfd	Parent class containing common data

DESCRIPTION

Another form of surface description is the triangular patch, also called a *Bézier triangle*. A triangular rational Bézier patch is defined as

$$\mathbf{S}(s, t) = \frac{\sum_{i+j \leq p} \mathbf{b}_{i,j} w_{i,j} B_{i,j}^p(s, t)}{\sum_{i+j \leq p} w_{i,j} B_{i,j}^p(s, t)}, \quad 0 \leq s, t \leq 1$$

which differs from the Bézier curve description only by the use of *bivariate* Bernstein polynomials instead of *univariate*, for the curve case. The bivariate Bernstein polynomials of degree p are defined as

$$B_{i,j}^p(s, t) = \frac{p!}{i!j!(p-i-j)!} s^i t^j (1-s-t)^{p-i-j}, \quad i+j \leq p$$

where the parameters s and t must fulfill the conditions

$$\begin{cases} 0 \leq s, t \\ s + t \leq 1 \end{cases}$$

which form a triangular domain in the parameter space, therefore the name of this surface description.

SEE ALSO

[BezierSurf](#)

BSplineCurve

SUPPORTED VERSIONS

0

SUBTYPE OF

[BSplineMfd](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	DESCRIPTION
integer	Version
BSplineMfd	Parent class containing common data

DESCRIPTION

The [BSplineCurve](#), describes a general spline curve, using B-spline basis functions, as defined in [BSplineMfd](#). Splines on this form are often referred to as B-splines.

A p th-degree spline curve is defined by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^n N_i^p(u) w_i \mathbf{P}_i}{\sum_{i=0}^p N_i^p(u) w_i}, \quad a \leq u \leq b$$

where \mathbf{P}_i are the control points., the w_i are the weights and the N_i^p are the p th degree B-spline basis functions defined in the nonperiodic and nonuniform knot vector

$$U = \{a, \dots, a, u_{p+1}, \dots, u_{m-p-1}, b, \dots, b\}$$

For non-rational B-splines, all weights are equal to 1 and the curve can be expressed as

$$\mathbf{C}(u) = \sum_{i=0}^n N_i^p(u) w_i \mathbf{P}_i, a \leq u \leq b$$

SEE ALSO

[BezierCurve](#)

BSplineMfd

SUPPORTED VERSIONS

0

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation class
integer		Dimension (1 if curve, 2 if surface)
integer	p, q	Degree in each dimension (1 or 2 integers)
Boolean		If rational equal to 1
integer		Number of knot vectors (1 for curves, 2 for surfaces)
integer	m1	Length of first knot vector
double[m1]	U	First knot vector
integer	m2	Length of second knot vector (not for curves)
double[m2]	V	Second knot vector (not for curves)
integer	n1	Number of control points in first parameter direction
integer	n2	Number of control points in second parameter dimension
integer	n3	Number of coordinates per control point
double [n1][n2][n3]	P	Matrix of coordinates where the last dimension is increased by 1 to store the weights if the manifold is rational

DESCRIPTION

The `BSplineMfd` type is the abstract base type for [BSplineCurve](#) and [BSplineSurf](#), which represent general spline curves and surfaces, respectively.

They are represented using *B-spline basis functions*. Let $U = \{u_0, \dots, u_m\}$ be a nondecreasing sequence of real numbers; U is called the *knot vector* and the elements u_i of U are called *knots*. The i th B-spline basis function of p -degree, $N_i^p(u)$, is defined as

$$N_i^0(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u)$$

A general B-spline can be described by

$$\mathbf{S}(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m N_i^p(u) N_j^q(v) w_{i,j} \mathbf{b}_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m N_i^p(u) N_j^q(v) w_{i,j}}, a \leq u \leq b, c \leq v \leq d$$

where

$$U = \{a, \dots, a, u_{p+1}, \dots, u_{m-p-1}, b, \dots, b\}$$

and

$$V = \{c, \dots, c, v_{p+1}, \dots, v_{m-p-1}, d, \dots, d\}$$

are the two knot vectors stored in the entry, and \mathbf{b} and w are the control points coordinates and weights stored in \mathbf{P} .

For periodic splines, the first and last parameter values in the knot vectors are not duplicated.

BSplineSurf

SUPPORTED VERSIONS

0

SUBTYPE OF

[BSplineMfd](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	DESCRIPTION
integer	Version
BSplineMfd	Parent class containing common data

DESCRIPTION

The generalization of B-spline curves to surfaces is a tensor product surfaces given by

$$\mathbf{S}(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m N_i^p(u) N_j^q(v) w_{i,j} \mathbf{P}_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m N_i^p(u) N_j^q(v) w_{i,j}}, a \leq u, v \leq b$$

SEE ALSO

[BezierSurf](#)

Ellipse

SUPPORTED VERSIONS

0

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation class
double[d]	center	Center coordinate
Boolean		Equal to 1 if clockwise rotation (only if d==2)
double[d]	normal	Normal vector coordinates,
double[d]	M	Major axis
double	rat	Ratio of minor axis length to major axis length
double	offset	Parameter at the end of major axis
Boolean		Equal to 1 if ellipse is degenerated

DESCRIPTION

This manifold defines an ellipse in the two or three dimensional space.

In 2D, an ellipse is defined by a center point `center`, a vector defining the major axis `M` of the ellipse (including the magnitude of the major axis), the radius ratio of the minor axis length to the major axis length `rat`, the direction of the ellipse, and the parameter offset at the major axis `offset`.

In 3D, an ellipse is defined by a center point `center`, a unit vector normal to the plane of the ellipse `normal`, a vector defining the major axis of the ellipse `M` (including the magnitude of the major axis), the radius ratio, and the parameter offset at the major axis `offset`. The direction of the ellipse is defined by the right hand rule using the normal vector.

An ellipse is a closed curve that has a period of 2π . It is parameterized as:

$$\text{point} = \text{center} + M \cos(t - \text{offset}) + N \sin(t - \text{offset})$$

where `M` and `N` are the major and minor axes, respectively.

EXAMPLE

```
7 Ellipse # class
0 0 # version
2 # sdim
0 2 1 # transformation
0 0 # center
0 # reverse?
2 0 # major axis
0.5 # minor axis length / major axis length
```

```

0 # parameter value at end of major axis
0 # degenerated?
# Attributes
0 # nof attributes

```

Geom0

SUPPORTED VERSIONS

1

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	type	Geometry type
double		Relative geometry tolerance
integer	p	Number of points (0 or 1)
integer	na	Number of attributes
Attribute[na]		Vector of attributes

DESCRIPTION

The type represent a 0D geometry class.

The type can be 0 for solid or -1 for general object.

EXAMPLE

A solid 0D geometry object.

```

5 Geom0 # class
1 0 1e-010 1
0 # nof attributes

```

Geom1

SUPPORTED VERSIONS

1, 2

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	type	Geometry type
integer		1 if void regions are labeled, 0 otherwise. Not present in version 1.

ENTITY/OBJECT	VARIABLE	DESCRIPTION
double		Geometry tolerance
integer	nv	Number of vertices
double[nv]	vtx	Vector of vertex coordinates
integer[nv][2]	ud	Matrix of integers giving domains on up and down side of each vertex

DESCRIPTION

The type represent a 1D geometry class.

The type can be 0, 1, or -1 for point, solid, or general objects, respectively.

EXAMPLE

A solid 1D object.

```

5 Geom1 # class
2 # version
1 # type
1 # voidsLabeled
1e-010 # gtol
3 # number of vertices
# Vertex coordinates
0
1
3
# Vertex up/down
1 0
2 1
0 2
# Attributes
0 # nof attributes

```

Geom2

SUPPORTED VERSIONS

1, 2

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer		Type
integer		1 if void regions are labeled, 0 otherwise. Not present in version 1.
double		Relative geometry tolerance
integer	nv	Number of vertices
integers/doubles [nv][4]	vertex	Matrix of vertex data
integer	ne	Number of edges

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integers/doubles [ne][8]	edge	Matrix of edge data
integer	nc	Number of curves
Manifold[nc]	curve	An array of Manifold objects

DESCRIPTION

The type represent a 2D geometry class.

The type can be 0, 1, 2, or -1 for point, curve, solid or general objects, respectively.

EXAMPLE

```

5 Geom2 # class
2 # version
1 # type
1 # voidsLabeled
1e-010 # gtol
0.0001 # resTol
2 # number of vertices
# Vertices
# X Y dom tol
0 0 -1 NAN
1 2.2999999999999998 -1 NAN

1 # number of edges
# Edges
# vtx1 vtx2 s1 s2 up down mfd tol
1 2 0 1 0 0 1 NAN
1 # number of manifolds
11 BezierCurve # class
0 0 # version
2 # sdim
0 2 1 # transformation
1 0 # degrees
2 # number of control points
0 0 1
1 2.2999999999999998 1
0 # nof attributes

```

Geom3

SUPPORTED VERSIONS

1,2

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	type	Type
integer		1 if void regions are labeled, 0 otherwise. Not present in version 1.
double		Relative geometry tolerance

ENTITY/OBJECT	VARIABLE	DESCRIPTION
double		Relative resolution tolerance
integer	nv	Number of vertices
integers/doubles [nv][5]	vertex	Matrix of vertex data
integer	npv	Number of parameter vertices
integers/doubles [npv][6]	pvertex	Matrix of parameter vertex data
integer	ne	Number of edges
integers/doubles [ne][7]	edge	Matrix of edge data
integer	npe	Number of parameter edges
integers/doubles [nep][10]	pedge	Matrix of parameter edge data
integer	nf	Number of faces
integers/doubles [nf][4]	face	Matrix of face data
integer	nm	Number of 3D manifolds, curves, and surfaces
Manifold[nmfd]	mfd	Vector of manifolds
integer	npc	Number of parameter curves
Manifold[npc]	pcurve	Vector of parameter curves

DESCRIPTION

The type represent a 3D geometry class.

The type can be 0, 1, 2, 3, or -1 for point, curve, shell, solid, or general objects, respectively.

Manifold

SUPPORTED VERSIONS

0

SUBTYPE OF

[Serializable](#)

FIELDS

This is an abstract class with no fields.

DESCRIPTION

A manifold is the common supertype for curve and surface types. It is used by the geometry types.

SEE ALSO

[Geom0](#), [Geom1](#), [Geom2](#), [Geom3](#)

Mesh

SUPPORTED VERSIONS

1

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension (if equal to 0 no more fields)
integer	np	Number of mesh points
integer		Lowest mesh point index
double[d][np]	p	Mesh points
integer	nt	Number of element types (fives the number of repeats of the following fields)
string		Element type
integer	nep	Number of nodes per element
integer	ne	Number of elements
integer[ne][nep]	elem	Matrix of point indices for each element
integer	ndom	Number of geometric entity values
integer[ndom]	dom	Vector of geometric entity labels for each element

DESCRIPTION

The geometric entity numbering for points, edges, and boundaries must start from 0 and the geometric entity numbering for domains must start from 1 when defining a mesh through a COMSOL Multiphysics mesh file.

EXAMPLE

The following displays a mesh with triangular elements on a unit square. Neither point nor edge elements are present.

```
4 Mesh # class
2 # sdim
5 # number of mesh points
0 # lowest mesh point index
# Mesh point coordinates on unit square
0 0
1 0
1 1
0 1
0.5 0.5
1 # number of element types
3 tri # type name
3 # number of nodes per element
4 # number of elements
# Elements, 4 triangular elements
0 1 4
3 0 4
2 3 4
1 2 4
4 # number of domains
# Domains
1
1
2
2
```

MeshCurve

SUPPORTED VERSIONS

1

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer		Space dimension
Transform		Transformation
integer	np	Number of points
double[np][d]	p	Matrix of point coordinates
double[np]	par	Vector of point parameters
Manifold	intcurve	Interpolating curve

DESCRIPTION

Mesh structures can also be used to define manifolds. Because meshes contain a number of nodes and, in the case of COMSOL Multiphysics, corresponding parameter values, a good geometric representation can be obtained using a suitable interpolation method for evaluating the values of the manifolds and its derivatives on parameter values that are inside the intervals between the given nodes. Mesh curves are handled by cubic spline interpolation.

The matrix `p` and the vector `par` corresponds to the structures corresponding structures in an edge mesh representation. For the `MeshCurve`, they serve as the interpolation data to obtain `intcurve`.

SEE ALSO

[BSplineCurve](#)

MeshSurf

SUPPORTED VERSIONS

1

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer		Space dimension
Transform		Transformation
integer	nv	Number of vertices
double[nv][3]	p	Matrix of mesh vertex coordinates
double[nv][2]		Matrix of mesh vertex parameters
integer	nt	Number of triangles
integers[nv][3]	elem	Matrix of vertex indices for each element

DESCRIPTION

Mesh structures for surface meshes can be used to make a geometric definition of unstructured data. Since a mesh type in COMSOL Multiphysics have coordinates as well as parameter values for each element, interpolation can be employed to create smooth surfaces.

A quadratic interpolation is used to define a parametric surface from a surface mesh.

The matrix `p` of coordinates and the triangles `elem` with indices into `p` are used as the interpolation data.

SEE ALSO

[Mesh](#)

Plane

SUPPORTED VERSIONS

1

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation
double[d]	p	The point in the plane with parameter value (0,0)
double[d]	n	Normal vector
double[d]	b	Direction of first parameter axis

DESCRIPTION

This manifold defines a plane in the three dimensional space. It is represented by a point, a unit vector normal to the plane, and the vector of the `u` derivative.

A plane is open in both parameter directions and neither periodic nor singular at any point. It is parameterized as:

$$\text{pos} = \text{p} + \text{u} * \text{b} + \text{v} * (\text{n} \times \text{b})$$

EXAMPLE

```
5 Plane # class
0 0 # version
3 # sdim
0 3 1 # transformation
1.3 0.80000000000000004 1.6000000000000001 # root point
-6.1257422745431001e-017 0 1 # normal
-1 0 -6.1257422745431001e-017 # derivatives
0 # degenerated?
```

PolChain

SUPPORTED VERSIONS

1

SUBTYPE OF[Manifold](#)**FIELDS**

The class is defined by the following fields:

ENTITY/ OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation
integer	n1	First dimension of matrix of point coordinates (equal to d)
integer	n2	Second dimension of matrix point coordinates, number of points
doubles	pol	n1-by-n2 matrix of point coordinates

DESCRIPTION

Polygon chains are piecewise linear curves that are used as approximations of curves in the decomposition algorithm. They have an implicit parameter representation; that is, $[(i-1)(p-1), i(p-1)]$ on the i th interval in a polygon chain with p points. This is not a suitable representation because the derivatives can vary substantially along the curve.

SEE ALSO[MeshCurve](#)*Serializable***SUPPORTED VERSIONS**

0

FIELDS

This is the abstract base type of all other types. It has no fields.

DESCRIPTION

Serializable is the abstract base type. It is used to indicate that a field can contain all supported types, as is the case for the Attribute type.

SEE ALSO[Attribute](#)*Solution***SUPPORTED VERSIONS**

1

SUBTYPE OF[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Solution object version
integer		Solution type
integer		Equal to one if it is ok to interpolate between solution vectors. Zero otherwise
string[]		Parameter names
integer		Number of degrees of freedom
integer		Mesh case number
integer		Size of parameter list
integer	nU	Number of data types in object
integer[nU]	iU	Number of solution vectors of each type
dcmplx[nU][iU]		Solution data identifier
integer[3]		DOF indices
Vector		Values of static DOFs
double		Parameter values
Vector		Values of dynamic DOFs without stored time derivatives
Vector		Values of dynamic DOFs with stored time derivatives
Vector		Time derivatives of corresponding DOFs

The first section of solution object specific data contains the following parts and in the following order:

- Internal solution object version number. At the time of writing the version number is 5.
- Solution type. The solution type is one of stationary solution(=0), parametric solution(=1), time-dependent solution(=2), and eigenvalue solution(=3).
- Whether or not interpolation between solution vectors is ok.
- Parameter names. Text file specific information: The first integer gives the number of parameter names (a single empty parameter name also registers as a 1 here). Thereafter each parameter is stored preceded by its length.
- Total number of degrees of freedom. This number includes both variables solved for and variables not solved for.
- Mesh case number.
- Size of the parameter list. This also becomes the number of stored dynamic solution vectors described below (if any such components are present).

The second section of solution specific data stores information about the type of each DOF and the values of any static DOFs. The data is stored in the following order:

- Internal solution object version number. At the time of writing the version number is 5.
- Number of different data types used in the present version of the solution object. At the time of writing the data types are: ordinary solution data, reaction forces data, adjoint sensitivity data, forward sensitivity data, functional sensitivity data, and linearization point data (and they are stored in this order).
- Number of stored solutions of each type. This number specifies the number of solutions per parameter value and not the total number of solution vectors of a certain type. Text file specific information: Space-separated vector of integers with information about the number of stored solutions of each type (ordinary solution data, reaction forces data, and so on).
- Solution data identifiers. Text file specific information: Space-separated vector of complex numbers (stored as real part followed by a space and the imaginary part) containing solution data identifiers. First, the identifiers for

all solutions of the first type is stored (where the number of solutions of the first type is given by the first integer of the preceding line). Then the identifiers for all solutions of the second type follows and so forth.

- Each DOF can be one of three types: static, dynamic without stored time derivatives, and dynamic with stored time derivatives. Information about which DOFs belong to which type is stored in three integer vectors. Text file specific information: The first vector contains the number of static DOFs followed by the indices of these DOFs. The second vector contains the number of dynamic DOFs stored without time derivatives followed by the indices of these DOFs. The third vector contains the number of dynamic DOFs stored with time derivatives followed by the indices of these DOFs.
- Vectors containing values of the static DOFs for one data type. Text file specific information: The first number gives the number of static vectors stored for the current data type. Thereafter the actual vectors are stored preceded by their lengths.

One DOF index data structure and one static DOFs values data structure is needed for each data type. The DOF indices and static vectors sections are therefore repeated n_U times.

The third section of solution specific data stores information about the dynamic part of the solution. In total this section contains as many subsections as the size of the parameter list. Each subsection contains the following information and in the following order:

- Internal solution object version number. At the time of writing the version number is 5.
- Number of different data types used in the present version of the solution object. At the time of writing the data types are: ordinary solution data, reaction forces data, adjoint sensitivity data, forward sensitivity data, functional sensitivity data, and linearization point data (and they are stored in this order).
- Parameter values for the data in the current subsection. The number of parameter values here is the same as the number of parameter names.
- Vector containing values of dynamic DOFs stored without time derivatives for one data type and the corresponding parameter values. Text file specific information: The first number gives the number of vectors stored for the current data type and DOF type. Thereafter the actual vectors are stored preceded by their lengths.
- Vector containing values of dynamic DOFs stored with time derivatives for one data type and the corresponding parameter values. Text file specific information: The first number gives the number of vectors stored for the current data type and DOF type. Thereafter the actual vectors are stored preceded by their lengths.
- Vector containing values of time derivatives for one data type and the corresponding parameter values. Text file specific information: The first number gives the number of vectors stored for the current data type and DOF type. Thereafter the actual vectors are stored preceded by their lengths.

Within each subsection dynamic DOF value data are stored for one data type at a time.

Example of text file output of a simple stationary solution object. The underlying setup is a simple one-dimensional geometry containing a single second order Lagrange element with two uncoupled unknown variables. Both variables are solved for:

```
# Created by COMSOL Multiphysics Tue Jun 30 13:08:00 2009

# Major & minor version
0 1
1 # number of tags
# Tags
4 sol1
1 # number of types
# Types
3 obj

# ----- Object 0 -----
```



```
# time derivatives
1 0
```

The object only contains static solution data and that the only data types that are present are ordinary solution data and reaction forces data. The reaction forces data vector contains four entries since the setup has two unknown variables and both have been solved for in the example.

A second example uses the same setup as the previous one with the difference that a parametric solve with the parameters p1 and p2 has been performed. The parameter list was selected as 0 0 0.1 0.2 and only one of the two variables is solved for:

```
# Created by COMSOL Multiphysics Tue Jun 30 13:08:29 2009
```

```
# Major & minor version
0 1
1 # number of tags
# Tags
4 sol1
1 # number of types
# Types
3 obj

# ----- Object 0 -----

0 0 1
8 Solution # class

5 # solution object version
1 # solution type
0 # interpolation between solnums is ok
2 2 p1 2 p2 # parameter names
6 # number of DOFs
0 # mesh case number
2 # size of parameter list

5 # solution object version
6 # number of data types
# number of solutions
1 1 0 0 0 0
# solution data identifier
NAN NAN NAN NAN
# DOF indices
3 1 3 5
3 0 2 4
0
# static vectors
1 3 0 0 0
# DOF indices
0
2 0 4
0
# static vectors
1 0
# DOF indices
0
0
0
# static vectors
# DOF indices
0
0
0
# static vectors
```

```

# DOF indices
0
0
0
# static vectors
# DOF indices
0
0
0
# static vectors

5 # solution object version
6 # number of data types
1 2 0 0 # parameter values
# DOFs without time derivatives
1 3 1 1.375 1
# DOFs with time derivatives
1 0
# time derivatives
1 0
# DOFs without time derivatives
1 2 -1.5 -1.500
# DOFs with time derivatives
1 0
# time derivatives
1 0

5 # solution object version
6 # number of data types
1 2 0.100 0.200 # parameter values
# DOFs without time derivatives
1 3 1 1.375 1
# DOFs with time derivatives
1 0
# time derivatives
1 0
# DOFs without time derivatives
1 2 -1.5 -1.500
# DOFs with time derivatives
1 0
# time derivatives
1 0

```

Half the DOFs end up in the static part and half in the dynamic part since only one variable is being solved for. This also has the consequence that the only appearing reaction force data in the stored solution object is in the dynamic part and that these vectors only have two components.

Straight

SUPPORTED VERSIONS

1

SUBTYPE OF

[Manifold](#)

FIELDS

The class is defined by the following fields:

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer		Version
integer	d	Space dimension
Transform		Transformation
double[d]	root	The point from which the ray starts
double[d]	dir	The direction in which the ray points
double	pscale	Parameter scale

DESCRIPTION

This manifold defines an infinite straight line in the two or three dimensional space. It is represented by a point and a unit vector specifying the direction. A straight also has a scale factor for the parameterization, so the parameter values can be made invariant under transformation. If not specified the value of this parameter is set to 1.0.

A straight line is an open curve that is not periodic. It is parameterized as:

$$\text{pos} = \text{root} + u * \text{pscale} * \text{dir}$$

where u is the parameter.

EXAMPLE

```
8 Straight # class
0 0 # version
3 # sdim
0 3 1 # transformation
1.3 0.8 0.0 # root point
-1 0 0 # direction
1 # parameter scale
```

SEE ALSO

[Plane](#)

Transform

SUPPORTED VERSIONS

1

SUBTYPE OF

[Serializable](#)

FIELDS

The class is

ENTITY/OBJECT	VARIABLE	DESCRIPTION
integer	d	Space dimension
Boolean		1 if transformation is a unit transformation, 0 otherwise. If the value is 1, no more fields are present
double [d+1][d+1]	M	Values in transformation matrix
Boolean		1 if determinant is positive, 0 otherwise
Boolean		1 if matrix is isotropic, 0 otherwise

DESCRIPTION

The transformation class is defined by the transformation matrix, which operates as a premultiplier on column vectors containing homogeneous coordinates thus

$$\begin{bmatrix} x' & y' & z' & s' \end{bmatrix} = M \cdot \begin{bmatrix} x & y & z & s \end{bmatrix}'$$

where the conventional 3D coordinates are

$$\begin{bmatrix} x & y & z \\ s & s & s \end{bmatrix}$$

The matrix thus consists of

$$\begin{bmatrix} & & & T_x \\ & \mathbf{R} & & T_y \\ & & & T_z \\ 0 & 0 & 0 & S \end{bmatrix}$$

where **R** is a nonsingular transformation matrix, containing the rotation, reflection, nonuniform scaling, and shearing components; **T** is a translation vector; and *S* is a global scaling factor greater than zero.

VectorDouble

SUPPORTED VERSIONS

SUBTYPE OF
[Serializable](#)

FIELDS
The class is defined by the following fields:

ENTITY/ OBJECT	VARIABLE	DESCRIPTION
integer	n	Number of elements
double[n]	d	Elements

DESCRIPTION
This is a wrapper for a vector of doubles that can be used to store fields in the `Attribute` class.

SEE ALSO
[Attribute](#)

VectorInt

SUBTYPE OF
[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	VARIABLE	DESCRIPTION
integer	n	Number of elements
integer[n]	d	Elements

DESCRIPTION

This is a wrapper for a vector of integers that can be used to store fields in the `Attribute` class.

SEE ALSO

[Attribute](#)

VectorString

SUBTYPE OF

[Serializable](#)

FIELDS

The class is defined by the following fields:

ENTITY/ OBJECT	VARIABLE	DESCRIPTION
integer	n	Number of elements
string[n]	d	Elements

DESCRIPTION

This is a wrapper for a vector of strings that can be used to store fields in the `Attribute` class.

SEE ALSO

[Attribute](#)

Examples of the Serialization Format

To illustrate the use of the serialization format, some text files are listed in this section.

- [A Mesh with Mixed Element Types](#)
- [A Planar Face](#)

A Mesh with Mixed Element Types

For an example of a file containing a 3D mesh with tetrahedral and prism elements, see `mesh_example_4.mph.txt` in `\models\COMSOL\Multiphysics\Meshing_Tutorials`.

A Planar Face

The following listing is a complete representation of a planar 3D face.

```
# Created by COMSOL Multiphysics Fri Aug 26 14:19:54 2005

# Major & minor version
0 1
##### Tags
1 # number of tags
# Tags
2 b1

##### Types
1 # number of types
# Types
3 obj

##### Records

# A planar face object

0 0 1
5 Geom3 # class
1 # version
2 # type
1e-010 # gtol
0.0001 # resTol
4 # number of vertices
# Vertices
# X Y Z sub tol
0 0 0 -1 1e-010
1 0 6.1257422745431001e-017 -1 1e-010
0 1 0 -1 1e-010
1 1 6.1257422745431001e-017 -1 1e-010
4 # number of parameter vertices
# Parameter vertices
# vtx s t fac mfd tol
1 0 0 -1 1 NAN
2 1 0 -1 1 NAN
3 0 1 -1 1 NAN
4 1 1 -1 1 NAN

4 # number of edges
# Edges
# vtx1 vtx2 s1 s2 sub mfd tol
1 2 0 1 -1 2 NAN
2 4 0 1 -1 3 NAN
4 3 0 1 -1 4 NAN
3 1 0 1 -1 5 NAN
4 # number of parameter edges
# Parameter edges
```



```

# edg v1 v2 s1 s2 up down mdfac mfd tol
1 1 2 0 1 1 0 1 1 NAN
2 2 4 0 1 1 0 2 1 NAN
3 4 3 0 1 1 0 3 1 NAN
4 3 1 0 1 1 0 4 1 NAN

1 # number of faces
# Faces
# up down mfd tol
0 0 1 NAN

5 # number of 3D manifolds
# Manifold #0
5 Plane # class
0 0 # version
3 # sdim
0 3 1 # transformation
0 0 0 # root point
-6.1257422745431001e-017 0 1 # normal
1 0 6.1257422745431001e-017 # derivatives
0 # degenerated?

# Manifold #1
8 Straight # class
0 0 # version
3 # sdim
0 3 1 # transformation
0 0 0 # root point
1 0 6.1257422745431001e-017 # direction
1 # parameter scale

# Manifold #2
8 Straight # class
0 0 # version
3 # sdim
0 3 1 # transformation
1 0 6.1257422745431001e-017 # root point

```


I n d e x

ID

- asymmetric geometry 81
- component couplings 56
- cut point data set 457
- cut point data sets 457
- geometric entity counter methods 165
- geometric primitives 147
- global plots 483
- grid data sets 487
- histogram plots 489
- images, exporting 494
- intervals 223
- line graphs 512
- mesh refinements 368
- nvtx matrix 167
- plot groups 561, 565
- point graphs 569
- points, reflecting 228
- revolved data sets 594
- table plots 621

2D

- asymmetric geometry 81
- Bézier polygons 181
- boundary similarity 60
- chamfers 184
- circles 186
- component couplings 56
- cut point data set 457
- cut point data sets 457
- DXF files, importing 217
- ellipses 203
- fillets 208
- geometric entity counter methods 165
- getVertex 167
- grid data sets 487
- height attribute 488
- images, exporting 494
- lines, reflecting 228
- matrix histogram plots 516
- objects, rotating 245
- parameterized curve data sets 537
- parametric curves 231
- plot groups 561
- Polygons 240
- rectangles 243
- rectangular arrays 177
- revolved data sets 594

- squares 250
- tangents 254
- work planes 260

3D

- Bézier polygons 181
- block shaped arrays 177
- blocks 183
- boundary similarity 59
- component couplings 56
- cut point data set 457
- cut point data sets 457
- cylinders 196
- ellipsoids 204
- extruding planar objects 207
- geometric entity counter methods 165
- getVertex 167
- grid data sets 487
- helix 210
- images, exporting 494
- objects, rotating 245
- parameterized curve data sets 537
- parametric curves 231
- parametric data sets 538
- parametric surfaces 232
- planar objects, revolving 243
- planes, reflecting 228
- plot groups 561
- Polygons 240
- pyramids 241
- rectangular arrays 177
- spheres 248
- sweeps 252
- tetrahedrons 256
- torus 257
- work planes 260

A

- absolute repair tolerance 158
- access methods 23
- Adaption (meshes) 367
- adding
 - geometry 152
 - height attributes to 2D plots 488
 - meshing sequences 270
- adjacency information, geometry 165
- Advanced (solvers) 370
- analytic functions 72
- angular units 82, 157
- Animation (plots) 431

- Annotation (plots) 434
- application library examples
 - grid data formats 650
- Argyris elements 338
- Array 177
- arrays, geometry 167
- ArrowLine (plots) 437
- ArrowSurface (plots) 437
- ArrowVolume (plots) 437
- Assemble (solvers) 372
- asymptotic waveform evaluation (AWE) 374
- Atolglobalmethod (solvers) 407
- attribute feature, meshes 270
- attribute, file format 656
- automatic rescaling of linear equations 371
- Average (data sets) 443
- average coupling operator 62
- AvVolume, AvSurface, AvLine 440
- AWE (solvers) 374
- axisymmetric geometries 81
- B**
 - Ball (meshes) 288
 - ball, meshes 288
 - base-vector coordinate system 52
 - basis functions 332
 - Batch (job type) 37
 - batch job 35
 - batch jobs
 - running compiled model files 18
 - Bernstein basis 657
 - Bernstein polynomials
 - bivariate and univariate 660
 - Bézier curve, file format 657
 - Bézier surface rectangular, file format 659
 - Bézier triangles, file format 659
 - BezierPolygon 182
 - binary file formats 653
 - bivariate Bernstein polynomials 660
 - Block 183
 - block shaped arrays 177
 - block size, assembly 371
 - block versions, meshes 282
 - BndLayer (meshes) 289
 - BndLayerProp (meshes) 292
 - Boolean operations 189
 - boundary coordinate systems 53
 - boundary layers, meshes 289
 - boundary probes 116
 - boundary similarity coupling 59
 - Box (meshes) 293
 - box, meshes 292
 - B-spline basis functions 661
 - B-spline curve 660
 - B-spline surface 662
 - bubble element 340
 - build status, meshes 100, 272
 - building
 - geometry 83, 154
 - meshes 100, 271
 - Bunch-Kaufman pivoting 387
- C**
 - C linkage 75
 - C matrix 401
 - CAD Import Module
 - 3D kernels 158
 - arrays 167
 - cadps 82
 - defeaturing 84
 - geometry, exporting 171
 - geometry, importing 221
 - hasCadRep 150
 - cadps 82
 - Chamfer 185
 - changing units 157
 - Circle 186
 - circleperpendicular, plane type 261
 - class files 14, 18
 - Cluster (job type) 38
 - coarse grid solver types 385
 - coefficient form 47
 - coefficient form equations 47
 - Color 445
 - color table files 652
 - compiled model files
 - running as batch jobs 18
 - compiled model files for Java
 - running from Desktop 18
 - compiling model files for Java 16
 - component couplings 56
 - Compose (geometry) 188
 - COMSOL Desktop 16
 - COMSOL Multiphysics files 653
 - COMSOL server 31
 - Cone 192
 - constraint Jacobian
 - assemble 372
 - input matrix 384
 - constraint vector
 - assemble 372
 - input matrix 384

- constraints 49
- contact pairs 108
- continuing operations, meshes 284
- Contour 447
- control vertices 657
- Convert (meshes) 293
- converting objects 194
- coordinate names 70
- coordinate system 51
- coordinates, mesh vertices 279
- coordintes, plane type 261
- CoordSysLine (plots) 451
- CoordSysSurface (plots) 451
- CoordSysVolume (plots) 451
- Copy 299
- Copy (geometry) 229
- CopyDomain 298
- CopyEdge 295
- CopyFace 296
- cpl_BoundarySimilarity 59
- CreateVertex (meshes) 301
- cumulative selection 162
- curl element 340
- curved mesh elements 333
- curves, creating 181, 240
- CutLine2D (data sets) 454
- CutLine3D (data sets) 454
- CutPlane (data sets) 455
- CutPoint (data sets) 457
- Cylinder 196
- Cylinder (meshes) 302
- cylinder, meshes 302
- cylindrical coordinate systems 53
- D**
 - D matrix 401
 - DAE 104, 408
 - damped Newton method 381
 - damping matrix
 - assemble 372
 - input matrix 384
 - damping ratios 393
 - Data (plots, exporting) 458
 - data job type 40
 - data set 119
 - data types 26
 - data, mesh 279
 - defeaturing, in CAD 84
 - Deform (plots) 460
 - degree, rational Bézier curve and 657
 - degrees of freedom 332
 - Delete (geometry) 198
 - Delete (meshes) 303
 - DeleteEntities (meshes) 304
 - deleting
 - geometry 155
 - mesh sequences 99
 - meshes 272
 - density element 348
 - dependent variables 86
 - derived values 428
 - destination map
 - extrusion coupling 57
 - projection coupling 60
 - DetectFaces (meshes) 305
 - Difference (geometry) 188
 - differential algebraic equation 408
 - Direct (solvers) 385
 - direct properties 386
 - Dirichlet boundary conditions 393
 - disabling
 - geometry 155
 - mesh features 272
 - discontinuous element 347
 - Distribution (meshes) 305
 - divergence elements 349
 - DOF. See degrees of freedom.
 - domain mesh, copying 298
 - domains, deleting 198
 - drop tolerance 387
 - duplicates, sectionwise data and 651
 - DXF, CAD drawing 217
- E**
 - eccentric oblique cones, creating 199
 - Eclipse 632
 - setting up for compiling Java files 18
 - ECone 199
 - Edge (data sets) 461
 - Edge (meshes) 307
 - edge elements 340
 - edge elements, meshes 282
 - edge evaluation, geometry 166
 - edge map 59
 - edge meshes, copying 295
 - edgeangle, plane type 261
 - EdgeGroup 307
 - EdgeMap 308
 - edgeparallel, plane type 261
 - edges, deleting 198
 - edited status, meshes 272
 - editing

- geometry 153
 - meshes 271
- eigenmodal method 392
- eigenpairs 393
- Eigenvalue 376
- Eigenvalue (solvers) 367
- element 62
- element coordinates 332
- element sets 65
- element types
 - Argyris elements 338
 - divergence elements 349
 - Hermite elements 338
 - Lagrange elements 334
 - vector elements 340
- elements, deleting from meshes 302
- elevation functions 77
- Ellipse 203
- ellipse, file format 663
- Ellipsoid 204
- emailing COMSOL 14
- error estimation function 369
- error features 154
- error status, meshes 272, 284
- Euler step 408
- Eval 461
- EvalGlobal 463
- EvalPoint 469
- evaluate expressions 462
- event 132
- Excel formats 651
- expand (meshes) 319
- explicit element distribution 306
- explicit time stepping 412
- exploding objects 249
- export feature 119
- exporting
 - data 649
 - geometry 84, 171
 - meshes 100, 286
- external functions 75
- extrapolation tolerance 57
- Extrude 207
- extruding, work planes 160
- extrusion coupling operator 57
- F**
 - face evaluation, geometry 166
 - face mesh, copying 296
 - faceparallel, plane type 261
 - faces, deleting 198
 - fast Fourier transform 377
 - feature status 154
 - FFT 377
 - field 68
 - file formats
 - attribute 656
 - binary 653
 - text 653–654
 - file records 653
 - Fillet 208
 - Filter (Particle) (plot attribute) 479–480
 - Filter (plot attribute) 479
 - Filter (Ray) (plot attribute) 481
 - finalized geometry 152
 - finite element space 332
 - finite elements 332
 - flattened corners, creating 184
 - fonts 32
 - force null-space basis 373
 - frame feature 70
 - frames 69
 - FreeQuad (meshes) 309
 - FreeTet (meshes) 310
 - FreeTri (meshes) 312
 - FromMesh 210
 - frustums, creating 199, 241
 - FullyCoupled (solvers) 381
 - function switch 71
 - functional, as error estimator 369
 - functions 71
- G**
 - gaussian pulse functions 73
 - general extrusion couplings 58
 - general matrix information 365
 - general projection couplings 61
 - geometric entities 28, 165
 - geometrical shape order 70
 - geometry 81
 - arrays 167
 - exporting 171
 - name 70
 - object names 155
 - object selection 84
 - objects, moving and copying 229
 - representation 158
 - selection 27
 - geometry features 81
 - geometry modeling kernels 158
 - geometry objects
 - importing 221

- geometry sequences
 - constructing 152
 - importing 218
- geometry shape order 69, 102, 333
- getType 23
- global
 - attribute features 270
 - selections 27
 - variable probes 116
- Global (plot) 483
- global equations 104
- Global numerical) 481
- gradient/Jacobian evaluation method 395
 - SNOPT 394
- Grid (data sets) 487
- grid data formats 650
- groups 86
- growth rate
 - mesh 278
- H** Height 488, 493, 623
- Helix 210
- Hermite elements 338
- hexahedral elements, numbering 283
- Hexahedron 212
- highly nonlinear 382
- Histogram (plot) 489
- host name 31
- I** identity mapping couplings 60
- identity pairs 108
- IFFT 377
- Image (exporting) 494
- Import (geometry) 217
- Import (meshes) 313
- importing
 - data 649
 - geometry objects 221
 - geometry sequences 218
 - meshes 270, 313
 - mphbin 221
- imprints, creating 210
- infinite elements 55
- initial values 86
- installation path 18
- Integral (data sets) 443
- integrated development environment 632
- integration coupling operator 61
- integration rules 47, 87
- intermediate space 57, 60
- Internet resources 14
- Interp (data sets) 498
- interpolation functions 72
- Intersection (geometry) 188
- Interval (geometry) 223
- IntLine (results feature) 440, 501, 520
- IntSurface (results feature) 440, 501, 520
- IntVolume (results feature) 440, 501, 520
- inverse fast Fourier transform 377
- inverted elements 277
- isoparametric elements 333, 337
- Isosurface (data sets) 507
- Isosurface (plots) 504
- Iterative (solvers) 385
- iterative properties 387
- J** Jacobian update technique
 - fully coupled 381
 - segregated step 400
- Java
 - development kit (JDK) 16
 - memory, meshing limitations 282
 - syntax 25
- Java IDE 632
- job type 35
- join, meshes 315
- JoinEntities (meshes) 315
- K** kernels 158
- knots 661
- knowledge base, COMSOL 15
- Krylov space 377
- L** Lagrange element 334
- Lagrange points 334
- length units 82, 157
- Line (plots) 508
- Linear (solvers) 384
- linear elements 332, 334
- linear extrusion couplings 58
- linear extrusion map properties 58
- linear projection coupling operator 61
- linearization point method 377
- LineGraph 512
- load group 86
- load vector
 - assemble 372
 - input matrix 384
- local
 - coordinates 332
- local attribute feature 270
- LogicalExpression (meshes) 316
- lower bound constraint vector 373

- M** Map (meshes) 316
- mapped coordinate systems 51
- mass matrix
 - assemble 372
 - input matrix 384
- mass properties 88
- material frames 69
- material link 91
- material models 90
- material switch 91
- materials 91
- MATLAB**
 - functions, general 78
 - LiveLink for 14
 - syntax 25
 - syntax for data types 25
- matrix assembly 372
- matrix creation 366
- matrix data 365
- Matrix Histogram (plot) 516
- matrix properties 26
- Maximum (data sets) 443
- maximum and minimum couplings 62
- MaxMinLine (plots) 518
- MaxMinSurface (plots) 518
- MaxMinVolume (plots) 518
- Mc, McA, McB matrices 401
- measuring
 - geometric entities 169
 - geometry 83
 - objects 169
- mesh
 - growth rate 278
 - importing 313
- Mesh (datasets) 524
- Mesh (exporting) 525
- Mesh (plots) 523
- mesh component 272
- mesh control entity 163
- mesh element sets 65
- mesh parts 272
- mesh, copying
 - copying meshes 299
- MeshError 284
- meshes
 - frames 69
 - importing 313
 - refining 320, 367
 - sequences 99
 - setting data 100
- meshing sequences 99
- MeshWarning 284
- MIG Layout Manager** 634
- Minimum (data sets) 443
- Mirror (data sets) 525
- Mirror (geometry) 228
- Modal (solvers) 392
- model
 - 32
 - .attr() 33
 - .attr(<tag>) 34
 - .batch 35
 - .bem 42
 - .capeopen 44
 - .coeff 46
 - .constr 48
 - .coordSystem 51
 - .cpl 55
 - .elem 62
 - .elementSet 65
 - .field 67
 - .frame 68
 - .func 71
 - .geom 79
 - .init 86
 - .intRule 87
 - .material 89
 - .mesh 98
 - .modelName 102
 - .multiphysics 103
 - .ode 104
 - .opt 105
 - .pair 107
 - .param 110
 - .physics 111
 - .probe 115
 - .result 118
 - .savePointt 121
 - .selection 122
 - .shape 128
 - .sol 130
 - .solverEvent 132
 - .study 134
 - .unitSystem 138
 - .variable 139
 - .view 141
 - .weak 144
- Model Builder 16
- model entity 34
- model entity list 34

- model files for Java
 - compiling 16
 - structure of 17
- model nodes 102
- model object 16
- model tree 102
- models, adding geometry 152
- ModelUtil 30
- ModelUtil.connect 31
- Move (geometry) 229
- mphbin 221
- mphbin files 655
- mphtxt 221
- mphtxt files 654
- Multigrid (solvers) 385
- multigrid levels 135
- multigrid properties 389
- MUMPS 387
- N**
 - named selection 122
 - naming, geometry objects 155
 - NASTRAN bulk data format 315
 - NASTRAN files 286
 - Nédélec's edge element 340
 - needs rebuild status, meshes 272
 - Neumann boundary conditions 393
 - Newton iterations 381
 - node points 332
 - nonlinear solver 381
 - null matrix 401
 - null-space basis 373
 - null-space function 371
 - number conventions, meshes 282
 - number of elements, meshes 277
 - numerical results feature 119
 - NURBS curve 660
 - NURBS surface 662
 - Nyquist plots 531
- O**
 - objective functions 106
 - objects, deleting 156, 198
 - OctaveBand 533
 - ODE 104
 - one point map 59
 - OnePointMap (meshes) 317
 - operation features, geometry 152
 - operation features, meshes 270
 - Optimization 394
 - optimization 105
 - Optimization (study node) 36
 - optimization constraint 373
- order
 - of Lagrange elements 334
- orthonormal system 51
- out-of-core solvers 387
- P**
 - pair 108
 - parameter 110
 - parameter checks 230
 - Parametric (data sets) 536
 - Parametric (job type) 36
 - Parametric (solvers) 396
 - ParametricCurve (geometry) 231
 - Parasolid kernel 164
 - ParCurve (data sets) 537
 - PARDISO 387
 - ParSurface (data sets) 538
 - Particle (1D) (plots) 543
 - Particle (data sets) 547
 - Particle (plots) 539
 - ParticleMass (plots) 550
 - ParticleTrajectories (plots) 555
 - PartInstance (geometry) 233
 - password protection 31
 - perfectly matched layer 54
 - physical quantities, list of 93
 - physics 113
 - physics interface 113
 - physics-controlled meshing 275
 - piecewise functions 73
 - pivot perturbation threshold 387
 - planar objects 207
 - Player (plots) 431
 - Plot (exporting) 560
 - plot group 119
 - PlotGroup 561
 - plots, color attribute 445
 - PML 54
 - Point (geometry) 239
 - Point (meshes) 319
 - point normals 427
 - PointGraph 569
 - PointTrajectories (plots) 573
 - polar coordinate systems 576
 - PolarGroup 576
 - Polygon 240
 - port number 31
 - preconditioner feature types 385
 - preordering algorithm 387
 - PreviousSolution (solvers) 397
 - PrincipalSurface (plots) 579

- PrincipalVolume (plots) 579
- prism elements, numbering 283
- probe 116
- processes 40
- projection coupling operator 60
- properties, boundary layers 292
- Pyramid 241
- pyramid elements, numbering 283
- Q**
 - quadratic elements 332, 334
 - quadrilateral elements, numbering 283
 - quality of elements, meshes 277
 - quality of elements, statistics 276
 - quick, plane type 261
- R**
 - ramp functions 74
 - random functions 74
 - Ray (1D) (plots) 581
 - Ray (data sets) 585
 - RayTrajectories (plots) 588
 - reaction forces
 - computation and storage of, stationary 403
 - computation and storage of, time dependent 409
 - records, file structure 653
 - Rectangle (geometry) 243
 - rectangle functions 74
 - rectangular arrays 177
 - Reference (meshes) 319
 - Refine (meshes) 320
 - reflecting objects 228
 - registry key 18
 - relative repair tolerance 82, 158
 - relative tolerance 377
 - relaxation factor 388
 - render index 427
 - report 120
 - results 420
 - Revolve (data sets) 594
 - Revolve (geometry) 244
 - revolving, work planes 160
 - RGB color values 652
 - Rotate (geometry) 245
 - rotated coordinate systems 52
 - Runge-Kutta explicit time stepping 412
- S**
 - SAI preconditioner 386
 - save point model 121
 - saving
 - COMSOL Multiphysics files 653
 - scalar plane wave basis function 350
 - Scale (geometry) 247
 - Scale (meshes) 322
 - scale vector 373
 - scaling coordinate systems 54
 - scaling values 157
 - ScatterSurface (plots) 595
 - ScatterVolume (plots) 595
 - sectionwise data formats 650
 - Segregated (solvers) 397
 - SegregatedStep (solvers) 399
 - selection
 - named 122
 - selection 27
 - selections
 - editing 153
 - frames and 70
 - meshes, and 273
 - sensitivity 105
 - Sensitivity (solvers) 400
 - sequences of operations 16
 - sequences, meshing 275
 - serialization
 - records 653
 - terminology 654
 - types 653
 - serialization, tags 653
 - servers, connecting 31
 - set (assign parameters) 24
 - setIndex(data types) 26
 - shape function 128
 - shape functions 333
 - shape functions, constraints and 49
 - sharg_2_5 Argyris element 338
 - shbub bubble element 340
 - shcurl 346
 - shdisc discontinuous element 347–348
 - shdiv divergence element 349
 - shgp Gauss point data elements 349
 - shherm Hermite element 339
 - shlag Lagrange element 337
 - shuwhelm scalar plane wave basis function 350
 - simplex meshes, converting 293
 - Size (meshes) 324
 - Slice (plots) 599
 - SmithGroup 604
 - solid polygons, creating 181, 240
 - Solution (data sets) 606
 - solutions
 - creating 364
 - object data 358
 - solver sequence 130
 - solver sequences 358

- source map, extrusion coupling 57
 - source map, projection coupling 60
 - sparse approximate inverse preconditioner 386
 - spatial coordinates 70
 - spatial frames 69
 - Sphere 248
 - spherical coordinate systems 53
 - spline curve 660
 - spline surface 662
 - Split (geometry) 249
 - splitmethod (meshes) 293
 - SPOOLES 387
 - spreadsheet data formats 649
 - Square 250
 - standard simplex 332
 - StateSpace 401
 - static linearized model 401
 - Stationary (solvers) 367, 402
 - statistics, meshes 100, 276
 - status, meshes 272, 278
 - step functions 75
 - stiffness matrix
 - assemble 372
 - input matrix 384
 - StopCondition (solvers) 403
 - stopping operations, meshes 284
 - storing, mesh data 279
 - Streamline (plots) 607
 - structured quadrilateral mesh, creating 316
 - study sequence 135
 - study types 135
 - StudyStep (solvers) 130, 405
 - summary of commands 420
 - Surface (data sets) 616
 - Surface (plots) 612
 - Sweep (geometry) 252
 - Sweep (meshes) 326
 - symmetric matrices 371
- T**
- Table (data) 617
 - Table (exporting) 620
 - Table (plots) 621, 624
 - table feature 119
 - tags, file structure and 653
 - tags, naming 23
 - Tangent (geometry) 254
 - tangential derivative variables
 - for Argyris elements 338
 - for Bubble elements 340
 - for curl elements 347
 - for discontinuous elements 348
 - for divergence elements 350
 - for Hermite elements 339
 - for Lagrange elements 337
 - task 39
 - task type 39
 - task type property 40
 - technical support, COMSOL 14
 - tetrahedral elements, numbering 283
 - tetrahedral mesh, creating 310
 - Tetrahedron (geometry) 256
 - text file formats 653
 - text files 654
 - Time (solvers) 405
 - Time Average (data set) 626
 - Time Integral (data set) 626
 - TimeDiscrete (solvers) 410
 - TimeParametric (solvers) 412
 - Torus 257
 - transferring, mesh data 280
 - transformed, plane type 261
 - transposed form 388
 - triangle functions 75
 - triangular Bézier surface 659
 - triangular elements, numbering 283
 - triangular meshes, unstructured 312
 - two point map 59
 - TwoPointMap (meshes) 328
- U**
- ultraweak variational formulation 350
 - undefined numerical values, checking 372
 - Union (geometry) 188
 - unit system entity 138
 - unit systems 138
 - unit systems, model objects 32
 - univariate Bernstein polynomials 660
 - unstructured quadrilateral mesh, creating 309
 - upper bound constraint vector 373
 - Uzawa iterations 398
- V**
- Vanka method 388
 - variable name suffix 70
 - variables 139
 - file formats and 649
 - Variables (solvers) 413
 - vector elements
 - curls elements and 340
 - vertices, deleting 198
 - vertices, plane type 261
 - views 141
 - virtual composite edge, face, domain, and entity 163

- virtual operations 152, 163
- Volume (plots) 627
- volume of elements, meshes 278
- vtx, meshes 301
- W**
 - warning status, meshes 272, 284
 - warnings, build operations 154
 - wave functions 78
 - weak equations 104
 - weak expressions 144
 - weak expressions, global equations and 104
 - weak form 144
 - weak form equations 144
 - websites, COMSOL 15
 - work planes 160, 260
 - WorkPlane (geometry) 260
- X**
 - XmeshInfo (solvers) 414