ModelMaker

Reference

ModelKinetix.com, a division of
Cherwell Scientific Limited

The Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom 04/00

End User License Agreement

1. LICENSE

   1.1. Cherwell Scientific Limited of The Magdalen Centre, Oxford Science Park, Oxford OX4 4GA ("Cherwell") hereby grants you a non-exclusive license ("License") to use the software accompanying this license ("Software") and the accompanying documentation on the following terms.

   1.2. The copyright and all other rights in the Software and the accompanying documentation remain with Cherwell.

2. ACCEPTANCE

   2.1. You are deemed to accept the terms of this License if you install the Software on a computer.

   2.2. If you do not wish to accept these terms, you must within 30 days of purchase erase any installation file, and any copy of the Software and documentation you have downloaded or stored on any medium and return any copies of the CD, manuals and packaging to Cherwell together with proof of purchase for a full refund.

3. SCOPE OF LICENSE

   3.1. This License permits you to:

      3.1.1. use the Software on a single computer or on a networked computer drive;

      3.1.2. transfer the Software to someone else, provided that you assign all of your rights and obligations under this License to such other person, you erase all copies of the Software under your control and stored on any medium (including the hard disk copy and any backup copy), and the other person agrees to the terms of this License;

      3.1.3. make one copy of the Software for back-up purposes only, which copy must reproduce and include this License in full.

   3.2. You shall not:

      3.2.1. use or copy the Software other than as permitted by this License;

      3.2.2. at any time allow more users than stated in your sales invoice to access or use the Software concurrently;

      3.2.3. modify, adapt, merge, translate, decompile, disassemble, or reverse engineer the Software, except as permitted by law;

      3.2.4. use, sell, assign, rent, sub-license, loan, mortgage, charge or otherwise deal in any way in the Software or the accompanying documentation or any interest in them or under this License except as expressly provided in this License.

4. TERM

4.1. Unless terminated under clause 4.2 this License shall last for as long as you continue to use the Software.

4.2. This License shall terminate automatically if you fail to abide by any of its terms.

4.3. Upon termination of this License you shall destroy the Software and its accompanying documentation and shall erase all copies of the Software under your control and stored on any medium (including the hard disk copy and any backup copy).

5. WARRANTIES AND REMEDIES

5.1. Cherwell warrants that the storage media on which the Software is supplied will be free from defects in materials and workmanship under normal use for a period of 90 days after the date of original purchase. If a defect in any disk occurs during such period you may return it with proof of purchase to Cherwell who will replace it free of charge.

5.2. Cherwell warrants that the Software will perform substantially in accordance with its accompanying documentation (provided that the Software is properly used on the computer and with the operating system for which it was designed) and that the documentation correctly describes the operation of the Software in all material respects. If Cherwell is notified of any material error in the Software during the period of 90 days after the date of original purchase it will correct any such error within a reasonable time or (at its option) refund the price of the Software (against return of the Software and its documentation).

5.3. The warranties set out in clauses 5.1 and 5.2 are your sole warranties and are in place of all warranties conditions or other terms expressed or implied by statute or otherwise, all of which are hereby excluded to the fullest extent permitted by law. Clauses 5.1 and 5.2 also set out your sole remedies for any breach of Cherwell's warranties.

5.4. In particular Cherwell does not warrant that the Software will meet your requirements or that the operation of the Software will be uninterrupted or error free or that all errors in the Software can be corrected. You load and use the Software at your own risk and in no event will Cherwell be liable to you for loss or damage of any kind (except personal injury or death resulting from Cherwell's negligence), including lost profits, consequential or other loss arising from the use of or inability to use the Software or from errors or deficiencies in it whether caused by negligence or otherwise, in excess of 125% of the price paid for the Software, except as expressly provided in this License.

5.5. If you deal as a consumer as defined in the Unfair Contract Terms Act 1977, your statutory rights remain unaffected.

6. SUPPORT

Cherwell's technical support staff will endeavour to answer any queries you may have about the Software. Support contact details are given in the Schedule.

7. LAW

This License constitutes the entire agreement between you and Cherwell relating to the Software and is governed by and construed in accordance with the laws of England. The courts of England shall have exclusive jurisdiction.

SCHEDULE

Technical support is available directly from the ModelKinetix web site at **http://www.modelkinetix.com/support**. Our web support database is constantly updated and we strongly urge you to explore the support options here before calling us.

In the unlikely event that you cannot find your answer on our web support database, please call our UK office directly on +44 (0)1865 784800 between the hours of 9.30 and 17.30 UK time Monday to Friday inclusive (excluding UK public holidays). Telephone calls will be logged in our support system and you will be contacted by e-mail, fax, or telephone with a solution or a request for more information.

E-mail support queries should be sent to support@modelkinetix.com.

Copyright

© 2000 Cherwell Scientific Ltd
All rights reserved. No part of this publication or the program ModelMaker may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language in any form or by any means electronic, mechanical, magnetic, optical, chemical, manual, biological or otherwise, without prior written permission of the publisher.

Publisher

ModelMaker is published by:Cherwell Scientific Limited
The Magdalen Centre
Oxford Science Park, Oxford OX4 4GA

Disclaimer

Cherwell Scientific Ltd make no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

Trademarks

All trademarks and registered trademarks are the property of their respective companies

ModelKinetix.com web site

For technical support, Frequently Asked Questions (FAQs) or product information, please visit our web site at:

http://www.modelkinetix.com

Technical Support

Technical support is available directly from the ModelKinetix.com web site at **http://www.modelkinetix.com -** simply follow the links to Support.

Our Web support database is constantly updated and we strongly urge you to explore the support options here before calling us. In the unlikely event that you cannot find your answer, then please e-mail us (**support@modelkinetix.com**) or call our UK office on **+44 (0)1865 784800** during normal office hours. Telephone calls will be logged in our support system and you will be contacted subsequently with a solution or request for more information by e-mail, fax, or phone.

Contents

1. Introduction

This manual provides a reference guide to using the ModelMaker application. There are two main sections:

Interface guide

The Interface guide (Chapter 2) provides a detailed description of the ModelMaker interface; toolbars, menus, dialogs and available shortcuts. You will find when using the software that there are often different ways of achieving the same result – allowing you to work in the way you choose.

Mathematical reference

The remaining chapters provides background information on the mathematical methods used by ModelMaker. This includes arithmetic, integration and interpolation methods, optimization, minimization and analysis methods. In most cases this is only an introduction and users are referred to standard texts and original papers for detailed descriptions.

2. Interface Reference Guide

ModelMaker window overview

Main window

The ModelMaker window comprises two panels. On the left is the Model Explorer, on the right is the Model View. Selecting a view on the Model Explorer shows that view on the right. It is possible to work on several models at one time, each model being displayed in its own model window. You may view these in the manner common to Windows applications using the Windows menu: cascade and tiled.

Toolbars

Two toolbars are in view at any one time.

- The main toolbar comprises icons for functions that can be used at any time during use
- The view dependent toolbar that changes according to the view currently on display. For example if you are viewing a graph then the Graph toolbar is on display

The View menu includes a 'Toolbar' command with a sub-menu for customizing this view dependent toolbar. You cannot change the appearance of the main toolbar.

Toolbars are dockable and can be rearranged by clicking and dragging on the left hand side. To make toolbars floating windows drag away from the toolbar panel. Re-dock by dragging back to the panel.

Menus

These are as follows:

- File: common file utility commands
- Edit: common editing commands
- View: commands pertaining to model views
- Component: select component type to add to model, Evaluation Order and Format open the relevant dialogs and

the Align sub-menus allow arrangement of selected components

- Model: all commands relating to Model run and analysis functions
- Windows: common window utility commands
- Help: open the Help file, Tip of the day, About box (ModelMaker version number and licensing) and links to the ModelMaker web site

In addition right mouse button (RMB) menus provide commands in context with the currently displayed view or selected model component.

ModelMaker Help

ModelMaker provides comprehensive on-line help and context sensitive help for all views and dialogs. The ?' button provides pop-up help for individual functions in dialogs. To use this

- Click the **?** icon in the main toolbar, or in the top right of the dialog
- Click on the item you wish to know more about

In addition pressing F1 opens the main help file at the appropriate topic.

Quick reference guide

The following sections provide a guide to all of the features of the ModelMaker interface including the use of keyboard shortcuts.

Toolbars - the Main view

Two toolbars are available. The Main Toolbar which is always in view and the View Toolbar which changes depending on the current ModelMaker view on display. You can customize the View Toolbars using the Customize feature (opened from the View|Toolbars|Customize menu command) by adding or removing command buttons from the toolbars. Details of the View toolbars are shown below.

Icon	**Command**
🗋	Open new model.
📂	Open existing model.
💾	Save model to disk.
✂	Cut selected components or text to clipboard.
⎘	Copy selected components to clipboard.
📋	Paste from clipboard.
📈	Create new graph - opens graph selection dialog box.
📊	Create new histogram (for results of Monte Carlo run) – opens the Histogram selection dialog.
▥	Create new table - opens table selection dialog box.
GO	Run active model.
⊞	View parameters. Create new or edit existing parameters.
↖	Select component. Click & drag to select multiple components.
▭	Compartment. A compartment represents an integrator in a model.
▢	Variable. Variables hold values which are calculated as the model is run.

Icon	Command
⬡	Defined Value: a constant value calculated at the start of a model run.
↗	Flow – represents movement between compartments.
⇢	Influence. Influences represent relationships between model components.
▭	Delay. Use to hold the value of another component for a defined period.
▷	DLL. Allows models to be connected to external functions - Windows DLL modules.
▣	Group components accomplishing a particular function as a sub-model.
⇤	Input. Use to transfer data from main model to sub-model.
⇥	Output. Use to transfer data from sub-model to main model.
⊞	Look-up Table. Use to enable real data to help define the model.
⊖	Look-up File. Links model to data in external file for look-up function.
ⓘ	Independent Event. Event triggered by independent variable.
ⓒ	Component Event. Event triggered by values of components.
🅃	Text Box. Use to annotate model diagrams, add graphics or both.

Toolbars - other views

Table View

Icon	Command
	Selection. Use to select component to include on graph or table.
	Series. Edit the attributes of the series displayed in active graph, table / model data view.
	Insert new row or rows into table of experimental model data.
	Delete row or rows from table of experimental model data.
	Insert new column or columns into table of model data.
	Delete column or columns from table of model data.

Graph/Histogram View

Icon	Command
T	Attributes. Title, legend and frame attributes of currently active graph.
X	Change X-axis characteristics for currently active graph.
↑Y	Change Y-axis characteristics for currently active graph.

Parameters View

Adds new parameter to model.

Sample Points View

Adds new sample point to model.

The Diagram - icons

Iconized text box. Double click to open.

Use Model Options (Diagram) dialog to select whether icon opens read-only text box or text box definition dialog.

Dialogs

In this section we list the ModelMaker dialog boxes.

Note that component definition dialogs have many things in common, including:

- A Bitmap panel for importing a graphic to enhance your model display
- An Information panel allowing you to add useful explanatory notes
- A list of available components whose values may be used in a defining equation
- A list of ModelMaker functions that may be used in a defining equation or in the case of event action ModelMaker commands and logical expressions
- A set of filters for selecting the components and functions to be displayed in the lists

If you make a mistake in defining a component, or use the symbol for an as yet undefined component or parameter ModelMaker will warn you but allow you to continue if you wish. You can switch off this warning message by clearing 'Quiet Model Editing' in User Preferences (Edit menu). Several dialogs have two or three tabbed panels. To open the individual panel, click on the tab at the top of the dialog

Dialog.	**To open….**	**Description**
About ModelMaker	Select Help | About menu command.	Displays ModelMaker version and license information.
Add Password Protection	Select File | Password Protection.	Enter a password that must be entered when opening the model.
Change Page	Select Change Page… from RMB menu on data sheet in Model Explorer.	Add a new name for a data sheet.
Component Event Definition (Trigger tab)	Double click Component Event.	Define the event trigger in terms of the value of a model component.
Component Format (Format tab)	Select Component Format from the RMB menu on any component.	Set line color and thickness and text alignment for individual components.
Component Format (Format tab)	Select Component Format from the RMB menu on any component.	Set font attributes for individual components.
Component order	Select Evaluation Order from the Component menu.	Set the order in which Events and DLL functions are evaluated during a model run.
Conditional Compartment Definition	Click Conditional in Unconditional Compartment Definition.	Lists series of conditions and equations for Compartment.

Dialog.	**To open….**	**Description**
Conditional Equation Definition	Click New or Change in Conditional component definition.	Define condition and corresponding equation for Conditional Component.
Debug Information	Select Debug Information from Model menu. (you must have Collect Debug information selected in User preferences dialog and have run the model).	Lists debugging information.
Delay Definition (Definition Tab)	Double click Delay.	Define a delay in terms of a symbol, the component whose value is to be delayed, an initial value and the maximum delay.
Distribution	Click the '… ' button in the Parameters view when Monte Carlo is selected for a parameter.	Select the distribution type and add distribution characteristics for Monte Carlo analysis.
DLL Definition (DLL tab)	Double click a DLL component.	Define a DLL symbol, the path to the DLL file and the name of the function being called. You can also define the function type.

Dialog.	**To open….**	**Description**
DLL Definition (Values tab)	Double click a DLL component.	Lists input values (equations or model component values passed to the DLL) and output values (as symbols that may be accessed by other model components.
DLL Function Input Value Definition	Click New… or Change… on DFF definition Values tab.	Define an Input Value for a DLL as an equation or model component value.
Event Definition (Actions tab)	Double click Independent or Component Event.	Define the actions (using ModelMaker script commands) that will occur when the event is triggered.
Event Definition (Definition tab)	Double click Independent or Component Event.	Define the Event symbol, status and class.
File Format Definition	Click File Format… on Lookup File Definition or Import File dialog.	Configure data file used as lookup or model data.
Find Component	Select Edit|Find Component.	Select component to locate on the model (useful when editing large models).
Global Sensitivity Results	Click Series Statistics in the Histogram series dialog.	Displays Global Sensitivity Results after a Monte Carlo run.
Graph Attributes Configuration (General tab)	Click the Attributes icon on the Graph Toolbar.	Set the background and frame colors for the graph.

Dialog.	**To open….**	**Description**
Graph Attributes Configuration (Legend tab)	Click the Attributes icon on the Graph Toolbar.	Add the legend text, set text characteristics and position.
Graph Attributes Configuration (Title tab)	Click the Attributes icon on the Graph Toolbar.	Add the title and set text characteristics and position.
Graph Selection	Click Graph icon on the Main toolbar (creates a new graph) or click the Selection icon on the Graph toolbar (changes the current graph).	Select components whose values are to be displayed on the graph.
Graph Series	Click the Series icon on the Graph toolbar.	Configure the model and data series (line color / thickness etc.) to be displayed on the graph.
Histogram Selection	Click the Histogram icon on the Main toolbar (create new histogram) or click the Selection icon on the Histogram toolbar (change current histogram).	Select sample points to display on the histogram. Define the number of columns in which to display the data.
Independent Event Definition (Trigger tab)	Double click Independent Event, click Triggers tab.	Define the event as period, non-periodic or both.

Dialog.	**To open….**	**Description**
Insert/Delete	Select Insert or Delete cells… from the RMB in the Model Data view.	Choose how to handle insertion or deletion of cells when data is already present in the view.
License Control Dialog	Click Licensing… in the About ModelMaker dialog.	To fully activate your copy of ModelMaker contact us with the two User Codes and then enter the activation codes we give you.
Lookup File Definition (Definition tab)	Double click Lookup File.	Define the Symbol and the path to the lookup file.
Lookup File Definition (Series tab)	Double click Lookup File.	Define the data series in the lookup table either (as a control or a controlled series) and the method of interpolation.
Lookup Table Definition (Definition Tab)	Double click Lookup Table.	Define the lookup Symbol, create or open the Table View and set the active page if several pages of data are available.
Lookup Table Definition (Series Tab)	Double click Lookup Table.	Define the data series in the lookup table either (as a control or a controlled series) and the method of interpolation.
Minimization Configuration	Select Model|Configure |Minimization or click Advanced on the Start Minimization dialog.	Define constraint ranges and convergence settings for the minimization process.

Dialog.	**To open….**	**Description**
Model Data Series Definition	Click the Series icon in the Model data toolbar.	Defines model data series i.e., relates data to model components.
Model Options (Diagram tab)	Select Model|Options menu command.	Set default line thickness and color for model diagram. Toggle Snap-to-Grid for drawing alignment and size of grid.
Model Options (Font tab)	Select Model|Options menu command.	Select the default model font and style.
Model Options (Symbols tab)	Select Model|Options menu command.	Define Symbols for the independent and iterator variables.
Monte Carlo Run	Select Model|Monte Carlo menu command.	Confirm Monte Carlo set up and start the run. Enter the number of trials to run.
Non-periodic Event Trigger Definition	Click New... or Change... under Non-periodic Triggers in Independent Event Definition, Triggers tab.	Define or edit a non-periodic trigger for an independent event.
Optimization Run	Select Model|Optimize.	Select Optimization method, error weighting and error estimation. Start the optimization.

Dialog.	**To open….**	**Description**
Optimization Settings (Estimation tab)	Select Model|Configure |Optimization menu or click Advanced from the Optimization Run dialog.	Configure initial parameter estimation range and method (Grid Search or Simulated Annealing).
Optimization Settings (Optimize tab)	Select Model|Configure |Optimization menu or click Advanced from the Optimization Run dialog.	Configure optimization convergence and Marquardt settings Define a constraint equation (may include several parameters).
Optimization Settings (Weighting tab)	Select Model|Configure |Optimization menu or click Advanced from the Optimization Run dialog.	Configure data weighting and error ranges for optimization.
Parameter Definition (Constraints tab)	Double click parameter in Parameters view.	Define constraint ranges for parameters (limits values parameters may take during optimization/minimization runs).
Parameter Definition (Definition tab)	Double click parameter in Parameters view.	Define parameter Symbol and value. X, Y and Z panels become active when defining arrays.
Parameter Definition (Estimation tab)	Double click parameter in Parameters view.	Define ranges for parameter estimation during optimization.
Password Entry	Opens automatically.	Enter the correct password to open the model.

Dialog.	**To open….**	**Description**
Reference Definition (parent view)	Double click Reference.	Define the Symbol for the reference.
Reference Definition (sub-model view)	Double click the reference.	Define a Symbol and arrow position for the link in the sub-model view.
Rename model Component	Opens automatically when you paste a model component onto the same model level.	Prompts user to rename a model component.
Run	Select Model|Integrate menu command or click Go.	Confirm Start/Stop values for independent variable, select repeated run and start the model run.
Run Options (Integration tab)	Select Model|Configure |Integration menu command or click Advanced on Run menu.	Select and configure integration method and error scaling.
Run Options (Repeated Run Settings tab)	Select Configure |Integration menu command or click Advanced on Run menu.	Configure the repeated run or sensitivity analysis.
Run Options (Repeated Run tab)	Select Model|Configure |Integration menu command or click Advanced on Run menu.	Set the number of repeated runs to perform and the components to be included.

Dialog.	**To open….**	**Description**
Sample Point Definition	Double click a sample point on the Sample Point view.	Define a Name, Time Point and component for the sample point (used in Monte Carlo and Minimization).
Start Minimization	Select Model|Minimize menu command.	Confirm minimization set up and start the minimization.
Sub-model Definition (Definition tab)	Click sub model with RMB and select Sub-Model definition.	Define Symbol and a description for the Sub-model.
Table Selection	Click the Table icon on the Main toolbar (create new table) or click the Selection icon on the table toolbar (change current table).	Select components whose values are to be displayed on the table.
Table Series	Click the Series icon on the Table toolbar.	Configure the numeric format and the column titles of model series displayed.
Text Box definition	Double click text Box.	Enter text to enhance your model. text box may also be iconized to save space.
Tip of the Day	Select Help|Tip of the Day menu or can open at start up.	Displays a helpful hint for using ModelMaker.
Toolbar Customization	Select View|Toolbar |Customize...	Add or remove command icons from View toolbars.

Dialog.	**To open….**	**Description**
Unconditional Compartment Definition (Definition tab)	Double click Compartment.	Define Symbol, Equation and initial value for Compartment.
Unconditional Defined Value Definition (Definition tab)	Double click Defined Value.	Define symbol and equation for a Defined Value
User Preferences (General tab)	Select User Preferences from the Edit menu.	Toggle a number of options controlling how you work with ModelMaker e.g. quiet editing mode, undo levels and user warnings.
User Preferences (Output tab)	Select User Preferences from the Edit menu.	Define report file names and levels of active tracing when debugging.
Variable Definition	Double click a variable.	Enter an equation to be calculated at every time step during the model run.
View Data	Click View Data on Lookup File Definition or on File Format dialog.	View data or text file imported into ModelMaker.
X or Y-Axis Configuration (Format tab)	Click the X or Y-axis icon on the Graph toolbar.	Configure the X-axis scale and grid lines.
X or Y-Axis Configuration (Title tab)	Click the X or Y-axis icon on the Graph toolbar.	Add the title, position and font style for the axis title. For the Y axis also select to display vertical text.
X-or Y Axis Configuration (Labels tab)	Click the X or Y-axis icon on the Graph toolbar.	Configure the numeric format and font style for the axis labels.

Selecting components

Mouse action	**Effect**
Mouse click on unselected component.	Component selected, all other components de-selected.
SHIFT + mouse click on unselected component.	Component is selected, any existing selection remains.
SHIFT + mouse click on selected component.	Component is de-selected, any existing selection remains.
Drag area.	All components inside the drag area selected, all other components de-selected.
SHIFT + drag area.	All components inside the drag area selected, any existing selection remains.
Mouse click outside component and selection area.	All components de-selected.

Scrolling The Model Diagram

Key	**Effect**
UP-ARROW or DOWN-ARROW	One line up or down.

LEFT-ARROW or RIGHT-ARROW	One column left or right.
PAGE-UP or PAGE-DOWN	One page up or down.
SHIFT-LEFT ARROW or SHIFT-RIGHT ARROW	One page left or right.
HOME or END	To the top or bottom of the diagram.
CTRL-LEFT ARROW or CTRL-RIGHT ARROW	To the extreme left or right of the diagram.
CTRL-HOME	To the top left corner of the diagram.
CTRL-END	To the bottom right corner of the diagram.

Keyboard Shortcuts

Key combination	**Effect**
Ctrl + Z	Undo. Will also undo actions within dialog boxes. Set undo level in Model Options (Diagram) dialog box.
Ctrl + Y	Redo.
Ctrl + X	Cut selection to clipboard.
Ctrl + C	Copy selection to clipboard.

Ctrl + V	Paste from clipboard into current model.
Del	Delete selected component(s).
Ctrl + L	View top level model (main).
Ctrl + N	View parent model - from sub-model view.
Ctrl + G	Create new graph, opens Graph Selection dialog box.
Ctrl + T	Create new table, opens Table Selection dialog box.
Ctrl + D	View definitions.
Ctrl + P	View Parameters.
Ctrl + +	Zoom in.
Ctrl + -	Zoom out.

Selecting Parameters

Before a parameter may be defined or edited it must be selected:

Mouse/Key combination	**Effect**
Mouse click on unselected parameter or parameter array range.	Parameter or range selected; all others de-selected.
SHIFT + mouse click on unselected parameter or parameter array range.	Parameter or range is selected; any existing selection remains.
SHIFT + mouse click on selected parameter or parameter array range.	Parameter or range is de-selected; any existing selection remains.
Mouse click outside parameter or parameter array range.	All Parameter and ranges de-selected.

Model component shading

Shading	**Interpretation**
Blue diagonal lines	Component is empty - model will not run.
Green diagonal lines	The component is global.
Green cross hatching	The component is universal.
Red cross-hatching	Component is in error - model will not run.

Moving around and selecting table cells

Using the Keyboard

Key/combination	**Effect**
Arrow key	Moves in the direction of the arrow by one cell.
SHIFT+ARROW	Extends the selection by one cell.
CTRL+RIGHT ARROW	Moves right one screen.
CTRL+SHIFT+RIGHT ARROW	Extends the selection right one screen.
CTRL+LEFT ARROW	Moves left one screen.
CTRL+SHIFT+LEFT ARROW	Extends the selection left one screen.
HOME	Moves to the beginning of the row.
SHIFT+HOME	Extends the selection to the beginning of the row.
CTRL+HOME	Moves to the top left hand corner of the page.
CTRL+SHIFT+HOME	Extends the selection to top left hand corner of page.
END	Moves to the end of the row.
SHIFT+END	Extends the selection to the end of the row.
CTRL+END	Moves to the bottom right hand corner of the page.
CTRL+SHIFT+END	Extends selection to bottom right hand corner of page.
PAGE DOWN	Moves down one screen.
PAGE UP	Moves up one screen.

Key/combination	Effect
SHIFT+PAGE DOWN	Extends the selection down one screen.
SHIFT+PAGE UP	Extends the selection up one screen.
DELETE	Deletes the contents of the selected cells.

Using the Mouse

Action	Effect
Click on a cell	Moves selection to a cell.
SHIFT + Click on a cell	Extends the selection to the cell.
Drag to another cell	Extends the selection to the cell.
Click on a row marker	Selects the row.
SHIFT + Click on a row marker	Extends the selection to the row.
Drag to another row marker	Extends the selection to the row.
Click on a column header	Selects the column.
SHIFT + Click on a column header	Extends the selection to the column.
Drag to another column header	Extends the selection to the column.

3. ModelMaker Arithmetic

Overview

This chapter defines ModelMaker arithmetic, its numerical accuracy, numerical and logical operations that are allowed and the mathematical functions provided.

Equations

Each model component has an equation defined by the user. ModelMaker equations may contain:

- Numerical values, in either fixed-point format (e.g. 13.567) or scientific notation (e.g. 1.23E+6).
- Symbols representing components within the model. These must be either global or linked via an influence or flow.
- Symbols representing parameters in the model.
- Arithmetic operators: + (addition), **-** (subtraction), * (multiplication), / (division), ^ (raising to a power).
- Mathematical functions defined within ModelMaker (e.g. cos()). See below for details.
- Brackets ( ) to control evaluation precedence in an equation or to specify the arguments of a function.

ModelMaker equations are intended to be as intuitive as possible and follow normal mathematical conventions. Equations are entered on a single line and use a restricted character set typical of those used by computers.

The order in which ModelMaker evaluates an equation follows strict rules of precedence, as follows:

Precedence	Operation
1	Expressions in brackets (........)
2	Functions and negatives (-1)
3	Powers (x^y)
4	Multiplication (*), division (/)
5	Addition (+), subtraction (-)

When two operations of equal precedence appear in an equation they are evaluated from left to right.

Conditions

In addition to normal arithmetical equations ModelMaker supports Boolean relationships i.e., those that produce a TRUE or FALSE result. These are called *conditions* and are used in conjunction with *conditional* components and component events.

Conditions use relational operators which yield a TRUE or FALSE result depending on the relative magnitudes of two numeric values. In the table below, *x* and *y* represent symbols or numerical values.

Operator	Result
$x < y$	TRUE if *x* is less than *y*
$x > y$	TRUE if *x* is greater than *y*
$x <= y$	TRUE if *x* is less than or equal to *y*
$x >= y$	TRUE if *x* is greater than or equal to *y*
$x = y$	TRUE if *x* equals *y*
$x <> y$	TRUE if *x* is not equal to *y*

If a relational operator does not give a TRUE result then the result is FALSE. No other result is possible.

Relational operators *cannot* be used to produce compound conditions. This can only be accomplished using logical operators. These have operands which are not numerical values, but Boolean values, either TRUE or FALSE. The logical operators are given below; op1 and op2 represent relational expressions using the above relational operators and yielding a result which is either TRUE or FALSE.

Operator	**Result**
op1 **and** op2	TRUE if both op1 and op2 are TRUE
op1 **or** op2	TRUE if either op1 or op2, or both are TRUE
op1 **xor** op2	TRUE if either op1 or op2 is TRUE, but FALSE if they are both TRUE
not op1	TRUE if op1 is FALSE

As with the relational operators if the result of a logical expression is not TRUE, it is FALSE.

Initial values of compartments

Compartments are described by differential equations which ModelMaker solves by initial value methods. Consequently compartments require initial values to be defined by the user. Initial value may be an equation, although in many cases the equation is simply a constant numeric value. For example, the default expression for a compartment's initial value is 0.0. This can be changed to a symbolic equation that conforms to the above rules; the only limitation is that the equation must be evaluated *before* a model step is taken. It can therefore only use symbols whose values are known at the beginning of a run. This limits it to parameters, defined values and lookup symbols controlled by the independent variable.

Numerical accuracy

ModelMaker uses 8-byte reals for all its arithmetic operations.

- The smallest value that can be computed is 1.7×10^{-308}. Any operation that results in a smaller number gives a result of zero.
- The largest value that can be computed is 1.7×10^{308}. Any operation that results in a number greater than this produces an error message.
- All calculations maintain numerical accuracy to the first 15 significant digits.

Functions

ModelMaker supports a library of standard mathematical functions. The syntax for using these functions in equations is very straightforward: you simply enter the function name and the function argument(s) in brackets. An argument is a value acted upon by the function. If required this can be an expression, in which case it is evaluated before being passed to the function.

The Available Functions list box in the various component definition dialog boxes lists all the functions that ModelMaker provides along with their syntax.

If the result of a function cannot be calculated then the model run halts and an error message is presented telling you which function has failed.

Mathematical functions in ModelMaker

The following table lists the mathematical functions available when constructing equations and conditions in ModelMaker.

Function	Remarks
abs(x)	Calculates the absolute value of x.

Function	Remarks
arccos(x)	Calculates the principle angle (in radians) whose cosine is x. The argument x must be in the range -1 to +1 otherwise the function fails.
arccosh(x)	Calculates the inverse hyperbolic cosine of x. If x < 1, the function fails.
arcsin(x)	Calculates the principal angle (in radians) whose sine is x. The argument x must be in the range -1 to +1 otherwise the function fails.
arcsinh(x)	Calculates the inverse hyperbolic sine of x.
arctan(x)	Calculates the principle angle (in radians) whose tangent is x. The argument x must be in the range -1 to +1 otherwise the function fails.
arctanh(x)	Calculates the inverse hyperbolic tangent of x. If x <= 1 or x >= -1, the function fails.
cos(x)	Calculates the cosine of x, which is assumed to be an angle in radians.
cosh(x)	Calculates the hyperbolic cosine of x.
deg(x)	Converts the value x in radians to degrees.
x div y	Calculates the number of times x is completely divisible by y.
exp(x)	Calculates the value e^x. If x > 709.7827, the function fails.
fact(x)	Calculates the factorial of the integer part of x.
frac(x)	Returns the fractional part of x i.e. frac(1.2345) = 0.2345.
ln(x)	Calculates the natural logarithm of x. If x < 0, the function fails.
log(x)	Calculates the logarithm to base 10 of x. If x < 0, the function fails.
max(x_1, x_2, ...)	Returns the maximum value from the list x_1, x_2, ...
min(x_1, x_2, ...)	Returns the minimum value from the list x_1, x_2, ...

Function	Remarks
x mod y	Calculates the remainder when x is divided by y.
rad(x)	Converts the value x in degrees to radians.
rand(x, y)	Generates a series of random numbers uniformly distributed between x and y.
randn(x, y)	Generates a series of random numbers with a mean of x and a standard deviation of y.
rande	Generates a series of random numbers exponentially distributed between 0 and 709.7827.
round(x)	Rounds x to the nearest integer. If the fractional part of x equals 0.5, x is rounded up.
sin(x)	Calculates the sine of x, which is assumed to be an angle in radians.
sinh(x)	Calculates the hyperbolic sine of x.
sqrt(x)	Calculates the square root of x. If $x < 0$, the function fails.
tan(x)	Calculates the tangent of x, which is assumed to be an angle in radians. This function is undefined at multiples of $\pi/2$.
tanh(x)	Calculates the hyperbolic tangent of x.
trunc(x)	Calculates the integer part of x i.e. trunc(1.23) = 1.0

4. Integration Methods

Overview

ModelMaker uses numerical methods to solve differential equations. Using ModelMaker you can solve equations that cannot be solved using analytical methods and find solutions to problems that would be beyond even expert mathematicians.

For non-expert mathematicians who nevertheless wish to find solutions to complex problems in the field of their own expertise ModelMaker uses suitable general methods of integration. In some cases these methods may be inappropriate and other methods may be chosen. This chapter gives an insight into these integration methods and when it may be appropriate to use them.

Reference This chapter describes only the bare essentials of the methods used. For a more detailed introduction see:

Press, William H., Flannery, Brian P., Teukolsky, Saul A. and Vetterling, William T., *Numerical Recipes*. Cambridge University Press, Cambridge 1989.

The full story is provided by:

Gear, C. William, *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, N.J. 1971.

Stoer, J. and Bulirsch, R., *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.

For details on Gear's Method3/14/00, see: C. W. Gear, *The Automatic Integration of Ordinary Differential Equations*, Comm ACM 14, 3, pp 176-179 (1971).

Boundary conditions

ModelMaker is designed to solve 'initial boundary' problems, i.e. sets of differential equations where only the initial value of a variable is known. The application does not solve two point boundary problems.

Consequently the remainder of this chapter comprises brief descriptions of how, given an initial value and a differential equation, ModelMaker uses the various methods to calculate 'future' values of the variables. Note that although ModelMaker uses time (symbol t) as the default independent variable you can use any variable and change the symbol if you wish to do so.

Euler's method

The simplest method for solving differential equations numerically is Euler's method. As with all other methods this is based upon the assumption that the rate of change at a point (the differential) also applies over a finite interval.

$$\frac{dy}{dt} \approx \frac{\Delta y}{\Delta t}$$

so that

$$\Delta \mathrm{y} \approx \frac{dy}{dt}\Delta t$$

The differential, which can be calculated from the known equation, is assumed to hold over a finite interval Δt so that a finite change in y, Δy, can be calculated. A new value of y can then be calculated from the previous value plus the change over the time interval:

$$y' = y + \Delta y$$

where y' is the value of the function after a time period of Δt.
This is an approximation because in fact the differential, dy/dt, may change over the finite interval Δt. However the approximation can be a good one if the time interval is small.

In practice this method is not recommended for many problems, as it can be very inaccurate and highly unstable. It is included in ModelMaker primarily for:

1. Educational applications.

2. Occasional models whose differential equations are very well behaved, and where minimizing the computational overhead associated with the other methods is desirable. 'Well behaved' is jargon usually meaning 'smooth', i.e. the function has continuous derivatives.

We recommend that if you are making serious use of the Euler method you should occasionally check that the results are comparable with those obtained using a more sophisticated method, e.g. Runge-Kutta (see later in this chapter).

Mid-point method

The mid-point method is an extension of Euler's method. A standard Euler step is used to estimate the value of the function *half-way* across the interval (i.e. at the mid-point). From this position a new estimate of the derivative is made, and this is applied across the *whole* step from y_n to y_{n+1}.

In equations this is represented as

$$y_{n+1/2} = y_n + \frac{dy_n}{dt}\frac{\Delta t}{2}$$
$$y_{n+1} = y_n + \frac{dy_{n+1/2}}{dt}\Delta t$$

This method has considerable advantages over Euler's method in terms of accuracy, primarily because it is using a value for the derivative that is a better average over the interval than a derivative calculated just at the beginning of the interval.

Runge-Kutta method

The idea behind the mid-point method can be extended almost *ad infinitum* to produce more and more complex ways to cross the interval with more intermediate points. However, such extensions do not necessarily confer greater accuracy, and certainly make the calculations more complicated. A compromise between using more and more intermediate evaluations of the derivatives and a reasonable level of performance is the '4th order Runge-Kutta method'. This is probably the most popular numerical method of all for solving differential equations and is the default method used by ModelMaker.

Runge-Kutta, specifically 4th order Runge-Kutta, involves four evaluations of the derivatives of the function, once at each end-point and twice in the middle (see diagram below).

- First the derivative at the beginning of the interval (position 1) is calculated and used to estimate the function at the midpoint (position 2), mid-point method style.

- The derivative is then calculated at this mid-point and applied from the beginning of the interval to arrive at *another* estimate of the mid-point (position 3).

- The derivative is recalculated at this second estimated mid-point and the new value is applied from the start of the interval to calculate an initial estimate of the function for the end of the interval (position 4).

- Finally the actual estimated value of the function at the end of the interval (y_{n+1}) is calculated by combining the derivatives, taking 1/3 of the value of each of the two mid-point derivatives, and 1/6 of each of the two end ones.

Algebraically:

$$y_{n+1} = y_n + \left(\frac{d_1}{6} + \frac{d_2}{3} + \frac{d_3}{3} + \frac{d_4}{6} \right) \Delta t$$

where d_1 is the derivative evaluated at position 1
d_2 is the derivative evaluated at position 2
d_3 is the derivative evaluated at position 3
d_4 is the derivative evaluated at position 4
Δt is the time step.

Runge-Kutta is a particularly robust method. It is unlikely to fail and can deal with some very difficult systems of differential equations, although it may not always be the quickest method. It is the default method selected by ModelMaker and if you try one of the alternatives you would be wise to compare its results to those of Runge-Kutta before risking your reputation on the rocks of integration noise.

Bulirsch-Stoer method

The Bulirsch-Stoer method differs from those detailed above in that it does not take small steps, it takes wild leaps forward. We do not discuss the fine details here but confine ourselves to the basic concept; readers are referred to the standard texts for a full discussion.

A large step is to be taken. To begin with this is done is two half steps, yielding an estimate of the solution at the end of the large step. It is then attempted in four sub-steps, yielding a new and theoretically more accurate estimate. It is then attempted in six sub-steps, yielding a new and theoretically even more accurate estimate.

This process could be repeated indefinitely, although a practical limit of 96 sub-steps is placed on the implementation used in ModelMaker. Each time, an estimate of the true answer is obtained together with estimates of the extrapolation error. If the error is unsatisfactory (see S*tep length control and accuracy criteria*), another attempt is made to cross the interval

If the error is within the required limits, the step is deemed a success and the process is restarted for the next interval.

This method is very elegant and can be very usefully applied. However, it should really be reserved for 'well behaved' problems, and can be of limited use if many output steps are required (see *Choice of method* later in this chapter).

Gear’s method

Gear's method is an algorithm for integrating a set of ordinary differential equations that works efficiently on stiff sets of equations. These are typically for systems exhibiting extremes of behavior – in some periods the functions are changing slowly and others where they change rapidly - whose solutions remain bounded. For example, simulating a set of chemical reactions often results in a stiff set equations because although a set of reactions may initially occur very rapidly if they tend toward some equilibrium then the timescale decreases greatly.

In Gear's method, when calculating the solution at a point in time, an interpolation polynomial is calculated for the previous points in time and the new point. A set of values, for the solution at the new time point, is acceptable if the derivatives at the time point, calculated from the equations are sufficiently similar to the derivatives of the interpolating polynomial at the time point.

An implementation of Gear's method has to choose when to change the order of the interpolating polynomial (typically orders between 1 and 4 are used) and when to change the size of integration steps. The version of Gear's method implemented in ModelMaker also estimates the integration step size to be used at the start of a simulation (the user can over-ride this).

Step length control and accuracy criteria

All the methods used by ModelMaker to solve differential equations automatically control their own step length to provide you with the answer at a specified level of accuracy as quickly as possible. Estimates are made of the accuracy of the solution and the length of step is adjusted accordingly. If the accuracy is inadequate then the step length is reduced; if the accuracy is greater than required the step length is increased.

Step length: Euler, Mid-point and Runge-Kutta

In order to be able to adjust the length of step taken an estimate is required of the accuracy of each step. In the cases of Euler, mid-point and Runge-Kutta an identical approach to estimating the error is used: step-halving. Having completed a step, the step is redone in two halves. This must produce a more accurate answer, and the assumption is made that the difference between the two estimates is a good estimate of the error in the step.

You also get a more accurate estimate of the function from the two half steps, and this is the value that all the methods actually use. Of course there is no estimate of how accurate this is relative to the 'true' and probably unobtainable analytic solution. All that is really known is how much better it is than the full size step! The only way to find out would be to take even smaller steps, i.e. do more work.

Although step-halving involves a lot more computation this is unavoidable if the advantages of step length control are to be gained, and in general these advantages greatly outweigh the costs. However, the overheads are not as great as might be first thought because for each of the methods some of the derivative evaluations can be used for both the full and the half steps.

The estimate of error, which we shall term ε, is used to adjust the step length by comparison with a desired accuracy, ε_0. The relationship used to make this comparison depends upon the method being used, because the methods vary in the *order* of their error term. Euler has a *second order* error term, meaning that the error is roughly proportional to Δt^2. In other words, its error is only one order less than the solution itself. The mid-point method's error term is *third order*, so the error is much smaller in relative terms. The Runge-Kutta error term is *fifth order*. This all means that the accuracy improvement resulting from the same change in step length is greater for Runge-Kutta than for mid-point, which is in turn greater than for Euler. Consequently the way in which the step lengths are altered to adjust the error for each step varies across the different methods.

For a time step Δt_1 producing an error ε, these arguments suggest that the time step Δt_0 which would produce the desired error ε_0 is given by the following expressions.

For Euler

$$\Delta t_0 = \Delta t_1 \left| \frac{e_0}{e} \right|^{0.5}$$

For mid-point

$$\Delta t_0 = \Delta t_1 \left| \frac{e_0}{e} \right|^{0.333}$$

For Runge-Kutta

$$\Delta t_0 = \Delta t_1 \left| \frac{e_0}{e} \right|^{0.2}$$

These equations are used to control the step length used by ModelMaker. If the error is larger than that desired then these are used to determine a new reduced step size to retry the step. If the error is less than that desired then the equations above are used to determine the step length used for the next step. In practice ModelMaker is a little more conservative than these equations suggest: when a step fails the time steps are reduced rather more than may be required, just to be sure that the next step succeeds.

The description above has been confined to considering just one differential equation. Most ModelMaker applications contain a number of simultaneous differential equations, each with their own errors. These are solved together and the same step length is used for each. Therefore the step length is determined by the *worst* equation.

Another issue raised by the simultaneous solution of numerous equations is that the errors calculated using the step-halving approach described above are absolute. This does not matter provided that all the equations have solutions of roughly the same order, so that the errors are roughly comparable. If they have solutions that are orders of magnitude apart then problems can arise. In this case it may make sense to use a fractional error to normalize between the different scale equations.

The desired accuracy referred to earlier, ε_0, is determined by multiplying the accuracy requirement defined by the user by a scaling factor. If Constant Errors is selected in the method setup, this scaling factor is a constant that the user can set. The default value is 1.0. If Fractional Errors is selected then the scaling factor is the reciprocal of the function value (1/[function value]). There is a tendency for this last option to appeal to most users as a first choice. However, they should be aware that it encounters real problems if the function approaches or passes through zero. In fact, if the solution is zero then the method fails completely. The third scaling method provided by ModelMaker is immune to this problem; it sets the error scaling equal to the magnitude of the solution plus its derivative times the step length ($y_n + \Delta t \times \mathrm{d}y / \mathrm{d}t$).

Step length: Bulirsch-Stoer

Adjusting the step length for the Bulirsch-Stoer method is done rather differently. It is quite rare for a Bulirsch-Stoer step to fail; the whole technique is based around the idea of keeping on trying until you get there! Adjusting the step length is therefore a rather prosaic business. In practice, it is rarely useful to extrapolate beyond the last seven estimates of the solution; the earlier values do not contribute extra information. Therefore if the method has to cross the interval more than seven times before converging satisfactorily then it is wasting some effort, implying that a smaller step is required. Conversely, if fewer than seven trips are required then the step is shorter than strictly needed. The next step length is adjusted accordingly. The step fails completely if it crosses the interval 11 times and still does not converge. Under these circumstances the step length is reduced sharply and the step re-attempted.

Through the use of these techniques ModelMaker constantly adjusts the step lengths it takes to ensure it works not *within* the accuracy you specify, but *at* the accuracy you specify. This means that if the system is highly dynamic then the step lengths are small and the calculation proceeds slowly. If the system only changes slowly then the step lengths become longer and the calculation is quicker. This effect can become apparent if you have a model where some kind of dynamic event suddenly occurs part way through the run. The calculation as monitored by the percentage indicator in the run message bar slows down visibly.

Step length: Gear’s method

When using Gear’s method a different approach is taken in that the end-user specifies a relative error per step. For example, if this is 0.001, then the integration method should ensure that the error on each component in one integration step is less than 0.1%.

Note that for stiff simulations, the ones for which Gear is an appropriate solver, the solution tends to a steady state based on conservation laws. This is independent of the integration errors and hence errors do not accumulate from step to step.

Gear's Method changes the step size and the order of the interpolating polynomial as follows:

If the iterative solution procedure for the (implicit) equations solved by Gear do not converge, then the step size is reduced and the step is re-tried.

If the iterative solution procedure converges, Gear's Method estimates the error on that step. If that is within the specified criterion, Gear's Method estimates the largest step sizes at which the error would be acceptable at the current order, an increased order and at a reduced order. If any of these would mean a significant increase in step size, the step size is changed and (if appropriate) so is the order.

If the error is unacceptable, the step size is reduced and the step is re-tried. If reducing the step size does not cure the problem, the order of the interpolating polynomial is reduced.

Choice of method

There is no *correct* method to solve the differential equations in your model. The best method to use is the one that gets the right answer in the shortest time. The default method in ModelMaker is 4th order Runge-Kutta.

The first question to ask is “Is the system stiff?” A stiff system is one in which the equations for the processes contain significantly different timescales and, at some instants during the simulation, the components values all change more slowly than the fastest process. A typical stiff system is a set of chemical reactions where some of them reaction rates are so fast that some of the components behave, at least for part of the simulation, as if they were in quasi-equilibrium.

If your system is stiff, you should use the Gear solver.

If your system is not stiff, the default (4th order Runge-Kutta) is a good method to try.

This solves most problems and is much more reliable than Euler or mid-point. By this we mean it is more likely to get the right answer. However, even with Runge-Kutta it is always worth occasionally checking that your solutions are accurate by specifying a much higher accuracy level and seeing whether you get the same result.

The Euler method is not very stable and is really only included for its educational value. Mid-point is usually a great deal safer. Both Euler and mid-point can be useful if you are *forcing* ModelMaker to use short step lengths, irrespective of the accuracy criteria. You might do this if for some reason you require lots of output intervals. In this situation step lengths set by the accuracy criteria often take the solution beyond the next output interval, and ModelMaker reduces the step length accordingly. You may therefore inadvertently force the solution to go more slowly than strictly necessary, simply because you want lots of points on your graph! In these circumstances the simplest methods, with their low computational overheads, can be useful.

What about Bulirsch-Stoer? It has accuracy comparable with that of Runge-Kutta and for some kinds of applications is much quicker, especially ones with smooth solutions. However, it does not deal well with discontinuities and sudden changes in the solutions. For example, data derived from lookup tables is often rather discontinuous:

The discontinuities can sometimes be smoothed out by using the more sophisticated interpolation options that exist for lookup tables However, the major limitation of Bulirsch-Stoer is that it is only really useful when it can take giant steps. For many problems it can never get into its stride because it has to keep stopping to generate output.

Starting step length

Above we discussed the adjustments to step length which ModelMaker performs as it goes along. Of course, this assumes that there is a valid first step that it can make. If the first step is so bad that the run fails immediately, giving an error, then the program has no chance to adjust automatically.

The initial step length is calculated by dividing the first output interval by the Starting Steps value set in the Integration Configuration dialog box. By default, the Starting Steps value is 1.0 so ModelMaker assumes that a reasonable first step length is one output step. Smaller first steps can be achieved by increasing the Starting Steps value.

5. Interpolation in Lookup Files and Tables

Overview

Lookup tables provide a way of including tables of numerical data in ModelMaker models. ModelMaker provides a number of methods for estimating values in between the tabulated values. Of these, three are really just 'rules' and three are standard numerical interpolation methods:

'Rules'	**'Interpolations'**
• Interval Start	• Linear
• Interval Center	• Polynomial
• Interval End	• Rational Function

Reference This chapter outlines the bare essentials of the methods used. For a more detailed introduction see:

Press, William H., Flannery, Brian P., Teukolsky, Saul A. and Vetterling, William T., *Numerical Recipes*. Cambridge University Press, Cambridge 1989.

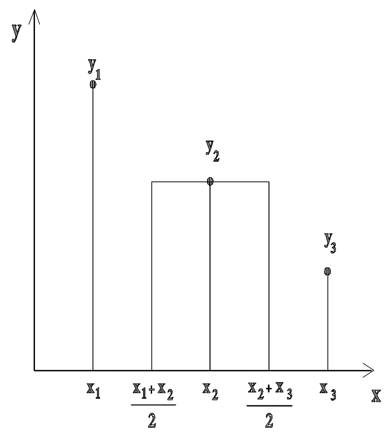
For the full story consult:

Stoer, J. and Bulirsch, R., *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.

Acton, Forman S., *Numerical Methods that Work*. Harper and Row, New York, 1970.

Rule methods

These are very simple methods. No actual interpolation is carried out; instead, simple rules are used to determine which value in the table to use. The rules are explained with reference to the diagram below.



- Interval Start. Suppose we require a value of *y* in the region $x_1 \le x < x_2$. If interval start is used then the value of y_1 applies across the whole interval.

- Interval End. Suppose we require a value of *y* in the region $x_1 < x \le x_2$. Under interval end the value of y_2 applies over the whole interval.

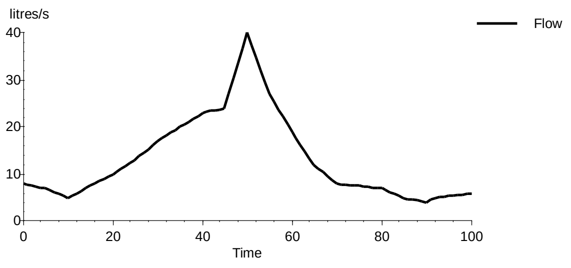
- Interval Center. Under this rule a *y* value is used over the half interval above and the half interval below its corresponding *x* value. For example, the value of y_2 applies from the midpoint between x_1 and x_2 to the midpoint between x_2 and x_3, i.e. over the range

$$\frac{x_1 + x_2}{2} \le x < \frac{x_2 + x_3}{2}$$

Linear interpolation

Linear interpolation uses information from two points to estimate values between them. As the name suggests, it involves taking the straight line between the two values and using this to calculate the value of *y* at any intermediate *x*:

The method is very quick and simple, but has the weakness that it can be rather discontinuous at the tabulated values, as shown in the example below. In some situations this can be a limitation.



Polynomial interpolation

Polynomial interpolation involves using information from points other than those bounding the interval of interest to define a polynomial which then gives an estimate of *y* for intermediate *x*.

For example, a cubic (i.e. third order polynomial) function is completely described by four values in the lookup tables. This function can then be used to estimate the value of *y* for $x_n \leq x < x_{n+1}$. The order of a polynomial is one less than the number of points required to define it, so linear interpolation is in fact a first order polynomial interpolation.

Third or fourth order polynomial interpolation generally produce much smoother results than simple linear interpolation. However you should be cautious of using higher orders as they are prone to yield solutions that oscillate wildly in between the tabulated values.

You can check how an interpolation behaves by plotting its results on a fairly fine output mesh over the range of interest.

Rational function interpolation

A rational function is one polynomial divided by another:

$$r(x) = \frac{n(x)}{d(x)} = \frac{a + bx + cx^2}{g + hx + ix^2}$$

Generally these are as good as polynomials at interpolation, although they are rather more complicated. However, they are useful for interpolating functions which contain 'poles'. A pole is a singularity of some sort, for example a place at which the function approaches positive or negative infinity. The function cannot be evaluated at such a place even by a rational function, but the latter can get a lot *closer* to a solution than can a comparable polynomial function.

For the mathematically minded, such poles can occur (quite commonly) in the complex plane near to the real point where you want to interpolate the function. Such a nearby complex pole can upset polynomial interpolation in a way that rational functions can resist.

Rational function interpolation is useful, then, for interpolating badly behaved functions while retaining the smoothness advantages of conventional polynomials.

Choice of interpolation method

There is no definitive answer to the question of which interpolation method to use. As ever, everything depends upon circumstances. However, the following points are worth bearing in mind:

- Interpolation is about *estimating* intermediate values using completely *arbitrary* functions. Everything therefore depends on the nature of the function you are interpolating.

- Sudden jumps and kinks in interpolated values can, sometimes, upset other numerical methods ModelMaker may be using, particularly for solving your differential equations.

- Higher order polynomials are *smoother* because they are *stiffer*! Like an oil tanker they cannot turn corners quickly. If you want a lot of sharp changes you may find them smoothed out by a 4th order polynomial.

- Very high order polynomials are erratic and can oscillate spectacularly in between your tabulated values whilst still passing through them.

- It is always wise to plot out interpolated variables, preferably using a fine output interval, over the range of interest to check out the method's behavior. This can easily be done in ModelMaker (see *User Manual*). Check to see whether it's doing what you want.

6. Optimization

Overview

In any simulation modeling environment one of the most important questions to ask should be "Is the model correct?" or "Does this model represent the observed system accurately?" A first step in validating a model might be to compare the output with observed (real) data. ModelMaker allows you to do this by adding data to the Model data view.

If the simulated data does not match the observed data to any obvious degree then there may be several reasons:

- The model itself is incorrectly built, i.e., has the wrong components
- The model structure is correct but has the wrong equations
- The model is 'more-or-less' correct but the parameters have the wrong values

ModelMaker includes statistical tools that allow some quantitative judgment to be made on just how close the model predictions are to the observed data. However the user must decide whether the model structure and the equations used are correct. Since most models are imperfect to some extent the decision is really whether the model is good enough. If the output seems to fit, the discrepancies may be reduced using ModelMaker's Optimization tools. Optimization is the adjustment of parameter values to minimize the differences between predicted and observed data.

Although optimization is a very useful tool, if you are interested in the way a model changes when parameter values change then you can also use the Mode Carlo, Minimization and Sensitivity analysis functions.

Reference This chapter outlines the bare essentials of the optimization methods available. For an introduction see:

Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., *Numerical Recipes*. Cambridge University Press, Cambridge, 1989.

For the full story refer to:

Beington, Philip R., *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill, New York, 1969.

Marquardt, D.W. (1963). *J. Soc. Ind. Appl. Math.*, **11**, 431-44.

Optimization defined

Optimization is the adjustment of parameters in the model in order to minimize the difference between the model's predictions and observed data.

As with all simulation methods there are potential disadvantages of which the user should be aware:

- Optimization is an iterative process, and is therefore time-consuming
- It may not find the 'best' parameter set. The minima it finds may be local, not global
- Initial values of all the parameters to be adjusted will be required. For some problems these may need to be good estimates for the method to succeed

Optimization as minimization of deviation

Two methods of minimization are available:

- Marquardt (also known as Levenberg–Marquardt)
- Simplex

The default regression method used by ModelMaker is the Marquardt method. The measure of deviation used by ModelMaker is the residual sum of squares, *RSS*, defined as:

$$RSS = \sum \frac{(m_i - o_i)^2}{e_i^2}$$

where o_i is the value of the *i*th observation
ε_i is the error estimate for that observation, and
m_i is the model prediction for that observation.

ModelMaker offers three main alternatives in the Optimization Run dialog:

- Ordinary least squares where $\varepsilon = 1$
- Weighted least squares where ε is set by the user in the Optimization Settings dialog (opened using the Advanced button in the Optimization Run dialog), Weighting tab
- Extended least squares where the error is assumed to have the form:

$$e^2 = s^2 m_i^z$$

where σ and ζ are values to be found by ModelMaker. When Extended Least Squares method is used values of σ and ζ for each component configured in the model data view are displayed in the Optimization Results view under Results in the Model Explorer.

A fourth option is to check 'Reasonable error estimates' in the Optimization Run dialog. This grays out the individual options and accepts the error weightings chosen by the user (in the Optimization Settings dialog) as being reasonable estimates and takes these values as they stand. Hence this is a weighted least squares with fixed error estimates. If Reasonable error estimates is not checked then it is assumed that the errors are thought to relate to each other but their absolute values are not known. In this case additional information is calculated by assuming a good fit and scaling parameter variances by the Residual Mean Square that is given as the ratio of the Residual Sum of Squares to the Residual Degrees of Freedom:

$$\text{Residual Mean Square} = \frac{\text{Residual Sum of Squares}}{\text{Residual Degrees of Freedom}}$$

Optimization recipe

The iterative approach employed for optimization is as follows:

1. Pick initial starting values for the parameters, a

2. Evaluate RSS

3. Change parameters according to the selected method

4. Evaluate the model with the new parameters, thereby obtaining a new value of *RSS*

5. If *RSS* improves, decrease the estimate a, re-evaluate the model and return to step 3

6. If *RSS* gets worse, increase the estimate a, re-evaluate the model and return to step 3

Convergence criteria

In principle the above recipe could continue indefinitely, producing a marginally better fit from time to time. In practice we should recognize that any measurement error, no matter how small, will prevent the perfect parameter set being found, even if the model is completely correct. Therefore it is sensible to specify when to stop, or in other words to stipulate *convergence criteria.*

In ModelMaker, optimization is deemed to have converged when a user-defined number of *consecutive* convergent steps has occurred. A convergent step occurs when *RSS* improves only marginally and the parameters do not change significantly.

If both of these conditions are true then the step is *convergent*. If a step is not convergent, i.e., *RSS* increases or decreases significantly, or the parameters change significantly then the running total of convergent steps returns to zero.

The default number of consecutive steps required for convergence is 5. This is rather conservative. For many problems a lower value will be sufficient and may save considerable computing time.

Parameter constraints

ModelMaker will attempt to impose constraints on the range of values the parameters to be optimized can take. This is to try and prevent completely unrealistic values being used that would cause the model to fail, thereby terminating the optimization attempt. You can override these by setting your own constraints using the Parameter Definition dialog.

If a parameter hits its constraint, it will usually return to a more reasonable value. If, however, the parameter continues to bump into the constraint for a user defined number of times, the optimization will halt (see *Optimization termination* below).

A constraint range can be defined for each individual parameter which will over-ride the default constraint range defined for the optimization process.

Optimization termination

The optimization process may stop before convergence occurs for a number of reasons:

- A calculation reaches the limit of machine accuracy. This means that the parameters and / or *RSS* are continuing to change, and the changes are so small that they cannot be calculated accurately because the computers limit of numerical accuracy has been reached. Usually this condition can be interpreted as full convergence.

- The value of λ (Marquardt method) becomes too large, so that the gradient solution results in a numerical overflow. This usually indicates a very poor fit, and may be overcome by trying improved starting values for the parameters.

- The model itself fails during one of the required simulations. This can happen if the optimization takes a wild jump, leading to a parameter set that takes the model's equations out of range. This situation may be overcome by reducing the constraint range for the parameters.

- Constraint range violation. The optimization has repeatedly tried to take one or more parameters outside their allowed range. Sometimes this is a realistic situation and the parameters need to be updated and the optimization repeated, perhaps with a wider constraint range. Sometimes, however, a parameter continues to be either decreased[1] with the result that it constantly hits its constraint range. This is often because its true value is zero! This problem is overcome by accepting the very low or zero parameter value, and re-optimizing without that parameter selected for adjustment.

Assumption of normality

Marquardt as implemented in ModelMaker assumes that the errors on your data are normally distributed. In practice you can usually get away with errors that are 'more or less' normally distributed without any dire consequences. In fact the estimates of the parameters are usually reliable even if the data has errors that are not at all normal in their distribution. However this does not apply to information derived from the covariance matrix, such as the uncertainty estimates on the parameter values. In such circumstances these should be viewed with caution.

Optimization statistics

How well does the model fit the data? ModelMaker provides three main ways of assessing this. After performing an optimization run two new views are created under Results in the Model Explorer. These are Parameter Results and Optimization Statistics. Click the latter to view the statistical analysis of the optimization run.

[1] Sometimes the situation occurs where the parameter is continually increased.

The first measure of the 'goodness of fit' can be obtained from the Sum of Squares value. This can be looked up on a χ^2 table with degrees of freedom equal to the number of data values minus the number of adjusted parameters. This gives the Q value – the probability that the difference between the model and the data has occurred by chance. To avoid the use of tables ModelMaker displays Q in the results view. Note that Q is only calculated when you select *Reasonable error estimates* in the Optimization Run dialog.

Interpreting *Q*

If Q is low then there is a problem which could be caused by any of the following:

- The model is wrong! The model does not fit the data, and perhaps never will. You should always be aware of this possibility.
- The errors on the data are underestimated. This can be quite common and the error estimation is usually worth checking. Be suspicious of purely instrumental errors where variation between replicates is likely. If the errors are based on variation between 'replicates', are the replicates genuine according to the scriptures of experimental design? (Consult a good textbook on the subject if you are unsure.) Are there enough replicates to give *good* estimates of the errors?
- The previous points apply only to random errors. If there are systematic errors in the data, they need to accounted for separately.
- Low values of Q can also be due to non-normal errors which often manifest themselves as an unusually high number of outliers.

This last point is quite common and isn't usually a major problem if the distribution is more or less normal. For this reason apparently low values of Q are often accepted.

A truly wrong model, i.e. one that fails to fit the data, will usually generate a very low Q value, say 10^{-10} or less. A Q value of 10^{-3} or so is worrying, but may just be a symptom of errors which are rather non-normal. A value of say, 0.1, is very encouraging. High values of Q, i.e. tending to 1.0, are basically too good to be true! They usually arise through overestimating the errors, or fudging the data! A useful rule of thumb is that χ^2 should be roughly equal to the number of degrees of freedom (the number of data values minus the number of adjustable parameters).

Analysis of variance

The second, very common, way to assess goodness of fit is to use an analysis of variance for the model. This partitions the total variation in the data into the variation explained by the model and the residual variation[2]. If the residual variation is small compared to the total variation, the model is doing a good job of describing the data. The information is summarized in the Optimization Statistics view (displayed under Results in the ModelMaker Explorer).

The columns of this table are the two components of variation (Model and Residual) and their total (the total variation in the data). The statistics presented in the table are:

- Degrees of freedom (DF).
- Weighted sum of squares (WSS). This is the variation attributed to each component.
- The mean square (MS). This is the variation per degree of freedom, i.e. MS = WSS / DF.
- The variance ratio or *F*-Value, where *F*= Model MS / Residual MS.

  The higher the value of the variance ratio the less likely it is that your model has explained variation by chance.
- *P*-Value. This is the probability associated with the variance ratio and degrees of freedom values. It is the probability that the model explained the variation by chance. The smaller it is the better the fit.
- r^2 value. This is the fraction of the total variation explained by the model, i.e. r^2 = Model WSS / Total WSS.

It is worth pointing out that the total sum of squares is calculated as the total squared deviation of the data from the (weighted) mean of

[2] The residual variation is the variation of the data about the simple model, i.e., the mean data value.

the data. In other words what you are really testing with your analysis of variance is whether the model is a (significantly) better representation of the data than its mean value.

Comparison of Q and F

Analysis of variance is therefore answering a different question to the Q value derived from χ^2. Q is the probability that the difference between the model and the data is pure chance, i.e. does not arise through a fault in the model (assuming that the data errors you provide are accurate and normal). This may give a low value for reasons other than the model being wrong. Similarly a low r^2 doesn't necessarily imply a poor fit to the data, merely that the model does not do much better than the mean. This sometimes happens if the data is very flat.

Visual comparison

The final, very popular, way to test goodness of fit is to make a visual comparison of the data overlaid on the model prediction using the graph view.

This gives an immediate indication as to the accuracy of the model and gross errors may be detected this way. However when a quantitative analysis is required you should use ModelMaker's inbuilt functions described above.

Optimization guidelines

Optimization as implemented in ModelMaker is a very powerful facility but it is *not* a panacea. The technique should be applied with care. The following points are worth noting as they are common pitfalls for the over-ambitious:

- Be sensible about how many parameters you attempt to optimize. Every Marquardt iteration has to carry out N+1 model runs (where N is the number of adjustable parameters). Hence increasing the number of parameters will increase the time required.

- You should have more (ideally many more) observed values than adjustable parameters.

- Try to provide reasonable initial values for your parameters. This will certainly increase your chances of success, and probably speed up convergence.

- Be aware of local minima in the χ^2 surface. Marquardt does not always find the best parameters for your model, the global minimum; it may get stuck in a local minimum, perhaps far from the true result. It is usually worth starting with several different initial sets of parameters to check that they all end up at the same convergence point.

Optimization changes the model predictions by adjusting the parameters. If the predictions are not very sensitive to changes in the parameters then it will be difficult to improve the fit. Put another way, ask yourself whether the data has something to say about the value of the parameters. A set of data will be of limited value in determining the value of a parameter which has only a marginal effect on the simulated values of that data. This is particularly true when the data has significant errors associated with it.

Parameter Estimation

Optimization will only give acceptable results if you have good estimates of the values of parameters to begin with. If you do not know the values of parameters then you may choose to use one of the parameter estimation algorithms included with ModelMaker. These attempt to find good initial values of parameters before running the optimization. The two methods are Grid Search and Simulated annealing. You can choose whether or not to use parameter estimation by selecting Initial Parameter Estimation in the Optimization Run dialog. This activates the method options. You can configure the methods in the Estimation tab of the Configure Optimization dialog opened from the *Advanced* button on the Optimization Run dialog or the Configure Optimization command on the Model menu. You can set methods for individual parameters using the Parameter Definition dialog for each parameter. On the Estimation tab select either Range or Fixed to activate the Grid steps option for Grid search.

Grid Search

This method of parameter estimation employs a simple grid within the defined estimation range. The size of the grid is defined in the Grid Steps edit box. As with other optimization routines this will change the parameter set until a minimum is found. the estimates will then be passed to the Marquardt optimization routine.

Simulated Annealing

When using the Marquardt method to find the optimal parameter set of a nonlinear system, you usually want to find those that correspond to the global minimum. In many cases multiple local minima may exist and these can cause problems in optimization algorithms as they cannot distinguish a local minimum from a global one. This is due to the fact that Marquardt finds minima by following the local gradient towards the closest minimum. If you have a parameter set close to a global minimum this is fine but otherwise the best solution may not be found.

Simulated annealing is able to escape these local minima and will find the global minimum more reliably. The downside is that the algorithm may take longer to converge at a solution.

Configuring Simulated Annealing

The Estimation tab of the Optimization Settings dialog comprises a number of configuration options for Simulated Annealing. These include the default search range and those specific to this algorithm.

Maximum number of temperature steps.

Simulated annealing works by starting with an *initial temperature*. It is this value governing how likely the algorithm is to accept worse steps. As the annealing continues, this *temperature* is gradually lowered, meaning that the algorithm is progressively less and less likely to accept worse steps. The *Maximum Number of temperature steps* controls the number of times the algorithm can lower the temperature.

Number of trials per temperature step

At each level of temperature, the algorithm makes a number of *trials*. During each trial the parameters are varied and the model is run. The new value of the objective function is compared to the current value and the change in the parameters is accepted or rejected.

Number of good trials before temperature step

If the algorithm is thought to be accepting too many trials at a particular temperature step, then the temperature should be lowered as it is likely that the temperature is too high, causing too many trials to be accepted.

Initial Temperature

This is the initial value of the temperature of the algorithm. It is an adjustable parameter for the user to set and change as necessary.

Acceptance Probability Distribution.

Trials are accepted or rejected by comparison with a probability distribution. Two functions, Boltzmann and Tsallis distributions are available.

Cooling Schedule

This controls how the temperature is lowered. If a "multiplicative" factor is chosen, then the temperature is reduced by

$$T_{new} = T_{old} * Factor$$

If *Exponential* is chosen, then the temperature is dropped according to the schedule:

$$T_{new} = T_{old} * \left[1 - \left(\frac{current\ step}{maximum\ steps}\right)\right]^N$$

where N is the user-defined factor in the *Exponential* edit box.

Termination Criterion

To stop the annealing, the user may wait for the maximum number of temperature steps to be taken, or specify a method of stopping the process if nothing is changing. User proportion of good steps will stop the optimization if the proportion of good trials falls below a certain level. For example if this value is 0.05, and the number of trials at a temperature step is 100, then the algorithm will stop if the number of good trials at a temperature step falls below 0.05*100 = 5.

7. Handling DLL Functions

ModelMaker DLL function components allow models to call functions located in external DLL modules during a model run. Such external functions receive an array of values from the model and can perform any manipulation of these values before returning a separate array of values to the model. Some possible uses of such functions are to:

- perform a special mathematical calculation not available within ModelMaker
- interface a ModelMaker model to the calculations of an existing model
- display a specially designed dialog box for the model
- query or update a database
- access an external device e.g. play a video or sound clip.

DLL Function Interface

For a function within a DLL to be compatible with a ModelMaker model it must

- conform to the ModelMaker DLL function prototype
- be an exported function.

The ModelMaker DLL function interface also defines the following attributes of the function and DLL resources:

- function parameters
- function return value
- function information string table.

Function Prototype

The ModelMaker DLL function prototype for C and Pascal are shown below.

C and C++

```
extern "C" int function_name (       int     input_length,
                              double*  input_values,
                              int      output_length,
                              double*  output_values);
```

Pascal and Delphi

```
Type

    ArrayDouble    = array [0..100] of Double; {array of arbitrary
                                                 length}
    PArrayDouble  = ^ArrayDouble;

    Function function_name(input_length: Longint;
                           input_values  : PArrayDouble;
                           output_length: Longint;
                           output_values    : PArrayDouble):
Longint; cdecl;
```

Exporting the Function

For C and C++ the function prototype makes the function an exported function. However, by default C and C++ add an underscore (_) to the start of all exported function names. In order to make the function available to ModelMaker as “function_name” rather than “_function_name” the EXPORTS section of the DLL model definition file must be used as follows:

```
EXPORTS
    function_name = _function_name
```

Pascal and Delphi require an export clause to be added to the module file as follows

```
export
    function_name;
```

Function Parameters

The parameters passed to a DLL Function are as follows:

- input_length

  A 4 byte integer containing the number of elements in the input_values array. This is the number of input values defined for the DLL function component plus:

  - 2 for Normal DLL functions
  - 3 for Integrate DLL Functions.

- input_values

  A far pointer to an array of 8 byte floating point values (double) with the following structure depending on the type of the DLL function component:

Normal

Window handle
Independent variable value
First input value
...
Last input value

Integrate

Window handle
Independent variable value at the start of the step
Independent variable value at the end of the step
First input value
...
Last input value

The window handle is the handle of the main ModelMaker window and allows the DLL to display a message or dialog box, if required.

- output_length

  A 4 byte integer containing the number of elements in the input_values array. This is the number of output values defined for the DLL function component.

- output_values

  A far pointer to an array of 8 byte floating point values (double) with the following structure:

First output value
...
Last output value

Function Return Value

The return value of the DLL function is defined as follows

- zero

  The function has completed successfully and the model run should continue.

- non-zero

  The function has failed and the model run should terminate.

A non-zero value returned by a DLL function can refer to a string resource contained within the DLL module. If the indicated string resource exists, ModelMaker displays a message box inserting the string resource text as follows:

“The model run was stopped during step <n> because <text> when evaluating the DLL function <component>.”

where	<n> is the step during which the error occurred
	<text> is the text read from the DLL’s string resource
	<component> is the name of the calling DLL function component

If the indicated string resource does not exist, ModelMaker displays the following default message box before the model run terminates.

"The model run was stopped during step <n> because a call to the ALL function <function_name> in module <module_name> failed with value <value> when evaluating the DLL function <component>."

where	<n> is the step during which the error occurred
	<function_name> is the name of the DLL function
	<module_name> is the name of the DLL module
	<value> is the value returned by the DLL function.

Function Information

The ModelMaker DLL function interface allows a string table resource to be added to the DLL module containing information about the functions provided by that module. This information appears in the DLL Information dialog box (see *Handling DLL Functions*). The string table must be defined
as follows:

String numbers	Description
20000 - 20099	Names of the functions contained in the DLL
20100 - 20199	General description of the functions contained in the DLL
20200 - 20299	Description of the input parameters
20300 - 20399	Description of the output parameters

If there are more than 100 functions in the DLL, the string resource structure repeats from 20400.

The function name strings (20000 - 20099) can also be used to indicate the type of each of the DLL function to ModelMaker as follows:

String format	Function type
function_name, N	Normal (default)
function_name, I	Integrate

The type indication contained in the resource strings is only of information. It does not prevent a normal type DLL function being called as an integrate function and vice versa.

8. Using Event Actions

This section describes the syntax and use of the model event actions. The syntax definition for the actions observe the following conventions:

[, parameter ...]	optional parameter list up to 64 parameters in length.
Value	numeric or model component value or expression. For example, 2.123, 2.2 + 100, (V1 / V2) + C1.
Event	independent or component event.
Component_Event	component event only.
Component	compartment or defined value.
Initial	compartment, defined value or delay.
Lookup	lookup component.
“formatted text”	text optionally containing formatting commands (see below).

Message Text Formatting

ModelMaker uses formatting commands to control the appearance of the values passed to the Message, GetChoice, GetValue, GetFileName and SetFileName actions. Formatting commands may appear in the message text, title text and file name fields of the above actions. All of the command begin with the percent character, %, and have the following format:

%[-][width][.places]<command>

The elements of the formatting command are described below.

-	This left justifies the formatted value within the space defined by the width element.
width	This is the minimum width, in characters, which the formatted text will occupy. If the formatted value requires fewer characters than the minimum with, it will be padded with spaces. If the value requires more space, the minimum number of characters necessary to show the value will be used.
places	This is the number of decimal places which will be used to format fixed point or scientific notation values. If the places element is absent, the default number of decimal places is 5.
command	This controls which notation is to be used to display the value as follows.

f	Fixed point
e	Scientific
i	Rounded to the nearest integer
n	Skip the associated value
%	A percent sign.

If any other character is used or there is no associated value in the parameter list to format, the formatting command will be removed from the message text and ignored.

Examples

Message("A value %f", "", 50.92346);	A value 50.92346
Message("A value %.2e", "", 50.92346);	A value 5.09E+01
Message("A percentage %i%%", "", 50.92346);	A percentage 51%
Message("A weight %8.3fkg", "", 50.92346);	A weight 50.923kg
Message("A weight %-8.3fkg", "", 50.92346);	A weight 50.923kg
Message("Second value %n%i, "", 1, 2);	Second value 2

EvaluateFunction

Purpose: To evaluate the specified list of DLL function.

Syntax: **Evaluate Function** (DLL_Function [,DLL_Function ...]);

Return Value: None.

See Also: EventDeactivate, EventQuery, EventProcess.

Comments: The event which uses this action must be universal, global or be connected to each of the DLL functions in the specified list by an influences.

Example: EvaluateFunction(Query_DataBase, Play_Tune);

EventActivate

Purpose: To activate the specified list of events.

Syntax: **EventActivate** (Event [, Event ...]);

Return Value: None.

See Also: EventDeactivate, EventQuery, EventProcess.

Comments: The event which uses this action must be universal, global or be connected to each of the events in the specified list by an influences.

Activated events are included in the model run calculations.

Example: EventActivate(Fertilizer_Event, Irrigation_Event, Disease_Event);

EventDeactivate

Purpose: To deactivate the specified list of events.

Syntax:	**EventDeactivate** (Event [, Event ...]);
Return Value:	None.
See Also:	EventActivate, EventQuery, EventProcess.
Comments:	The event which uses this action must be universal, global or be connected to each of the events in the specified list by an influences. Deactivated events are not included in the model run calculations.
Example:	EventDeactivate(Infection_Event);

EventProcess

Purpose:	To unconditionally process the actions of the specified list of independent and component events.
Syntax:	**EventProcess** (Event [, Event ...]);
Return Value:	None.
See Also:	EventActivate, EventDeactivate, EventQuery.
Comments:	The event which uses this action must be global or be connected to each of the events in the specified list by an influences. EventQuery will process a specified event even if it is deactivated.
Example:	EventProcess(Harvest_Event);

EventQuery

Purpose:	To evaluate the trigger condition for each the specified list of component events. If any of the events is triggered, its actions are processed.

Syntax: **EventQuery**(Component_Event[,Component_Event ...]);

Return Value: None.

See Also: EventActivate, EventDeactivate, EventProcess.

Comments: All the specified events must be component events.

The event which uses this action must be universal, global or be connected to each of the events in the specified list by an influences.

EventQuery will evaluate a specified event even if it is deactivated.

Example: EventQuery(Lake_Full_Event, Pollution_Event);

GetChoice

Purpose: To prompt the user to make a Yes/No choice using a message box containing the specified formatted text.

Syntax: Component = **GetChoice** (“*Message text*”, “*Title text*” [,Value ...]);

Return Value: Component = 1.0 if Yes is pressed.

Component = 0.0 if No is pressed.

See Also: Message, GetValue, GetFileName.

Comments: The message box displayed contains three buttons: Yes, No and Cancel. Pressing Yes or No will return the values 1.0 or 0.0 respectively and allow the model run to continue. Pressing Cancel stops the model run.

The Message text and Title text fields can contain both ordinary text and formatting commands which define how the following values are displayed (see Message Text Formatting earlier in this chapter).

Example: halt = GetChoice(“Total production has reached %.3f tons on day %i. Do you want to halt production?”,”Question”, produce, t);

GetFileName

Purpose: To prompt the user for a lookup file name using a message box containing the specified formatted text. If the entered file name exists and is a valid lookup file, it is assigned to the defined lookup. The current lookup file name is used as the default value.

Syntax: **GetFileName** (“*Message text*”, “*Title text*”, Lookup_File [,Value ...]);

Return Value: If the entered file name is valid GetFileName assigns the file name to the specified lookup file.

See Also: SetFileName, Message, GetChoice, GetValue, GetPage.

Comments: The message box displayed contains three buttons: OK, Find and Cancel. Pressing OK causes ModelMaker to validate the file. If the file is valid, it is assigned to the specified lookup file and the model run continues. If the file is invalid, an error message is displayed and the user prompted to enter another file name. Pressing Find will invoke the Find File dialog box and pressing Cancel stops the model run.

The Message text and Title text fields can contain both ordinary text and formatting commands which define how the following values are displayed (see Message Text Formatting earlier in this chapter).

Example: GetFileName(“Which data file is to be used for %i?”, “Meteorological Data”, Met_data, 1993);

GetPage

Purpose: To prompt the user for a lookup table page number using a message box containing the specified formatted text. If the entered page number exists, the page becomes the active page defined for the lookup table. The current lookup table page number is used as the default value.

Syntax: **GetPage** ("*Message text*", "*Title text*", Lookup_Table [,Value ...])

Return Value: If the entered page number is valid GetPage make the page the active page of the specified lookup table.

See Also: QueryPage, SetPage.

Comments: The message box displayed contains two buttons: OK and Cancel. Pressing OK causes ModelMaker to validate the entered page number. If the page exists, it become the active page for the specified lookup table and the model run continues. If the page number is invalid, an error message is displayed and the user prompted to enter another page number. Pressing Cancel stops the model run.

The Message text and Title text fields can contain both ordinary text and formatting commands which define how the following values are displayed (see Message Text Formatting earlier in this chapter).

Example: GetPage("Which page of data is to be used for %i?", "Meteorological Data", Met_data, 1993)

GetValue

Purpose: To prompt the user for a value by using a message box containing the specified formatted text. If the entered value satisfies the given validation condition, it is assigned to the defined component. The current value of the component is used as the default value.

Syntax: **GetValue** ("*Message text*", "*Title text*", Component, Condition [,Value ...]);

Return Value: If the entered value is valid, GetValue assigns the value to the component.

See Also: Message, GetChoice, GetFileName.

Comments: The message box displayed contains two buttons: OK and Cancel. Pressing OK will cause ModelMaker to evaluate the condition using the entered value. If the condition is TRUE, the entered value is assigned to the component specified by Value-component and the model run continues. If the condition is FALSE, an error message is displayed and the user prompted to enter another value. Pressing Cancel stops the model run.

The Message text and Title text fields can contain both ordinary text and formatting commands which define how the following values are displayed (see Message Text Formatting earlier in this chapter).

Example: GetValue("The production rate cannot be greater than %.2f. What should the new rate be?", "Production Rate", rate, rate > 0 and rate <= max_rate, max_rate);

Initialize

Purpose: To re-initialize the specified list of compartments, defined values and delays.

Syntax: **Initialize** (Component [, Component ...]);

Return Value: None.

See Also: InitializeAll.

Comments: The equations for the initial value of the specified compartments, defined values and delays will be evaluated and the calculated values assigned to the components. If an initial value equation references the independent variable, the current value of the independent variable will be used in the evaluation.

Example	Initialize(Fuel_load, Distance_traveled)

InitializeAll

Purpose:	To re-initialize all the compartments, defined values and delays for a model.
Syntax:	**InitializeAll;**
Return Value:	None.
See Also:	Initialize.
Comments:	The equations for the initial value of all the model's compartments, defined values and delays will be evaluated and the calculated values assigned to the components. If an initial value equation references the independent variable, the current value of the independent variable will be used in the evaluation.

Message

Purpose:	To display a message box showing the specified formatted text.
Syntax:	**Message** ("*Message text*", "*Title text*" [,Value...]);
Return Value:	None.
See Also:	GetChoice, GetValue, GetFileName.
Comments:	The message box displayed contains two buttons: OK and Cancel. Pressing OK will allow the model run to continue. Pressing Cancel will stop the model run. The Message text and Title text fields can contain both ordinary text and formatting commands which define how the following values are displayed (see Message Text Formatting earlier in this chapter).
Example:	Message("Entering stage %i at time %f", "Warning",

stage, t);

ModelExit

Purpose: To stop the model run generating an error message.

Syntax: **ModelExit** (Value);

Return Value: None.

See Also: ModelStop.

Comments: ModelMaker will stop the run of the model generating an error message showing the run step during which the action occurred, the event responsible and the specified value. If optimization, confidence interval calculation, sensitivity analysis or repeated runs is being performed, the process will be terminated.

Example: ModelExit(1);

ModelStop

Purpose: To stop the model run without generating an error message.

Syntax: **ModelStop**

Return Value: None.

See Also: ModelExit.

Comments: ModelMaker will stop the run of the model without causing an error. This means that if sensitivity analysis or a repeated run is being performed, the process will not be terminated.

ModelStop is not valid during model optimization, minimization or confidence interval calculation. If a ModelStop action is encountered, the process will be terminated with an appropriate error message.

QueryPage

Purpose:	To return the number of the active page of the specified lookup table.
Syntax:	**QueryPage**(Lookup_Table);
Return Value:	The number of the active page of the specified lookup table.
See Also:	GetPage, SetPage.
Example:	page = QueryPage(Met_data);

SetPage

Purpose:	To set the active page for the specified lookup table.
Syntax:	**SetPage**(Lookup_Table, Value);
Return Value:	If the specified page number is valid SetPage sets the page as the active page for the specified lookup table.
See Also:	GetPage, QueryPage.
Comments:	ModelMaker attempts to load the page with resulting number. If an error occurs the model run is terminated with an appropriate error message.
Example:	SetPage(Met_data, page_index);

SetFileName

Purpose:	To set the file name for the specified lookup file.
Syntax:	**SetFileName**(Lookup, *“File name”* [, Value]);
Return Value:	If the specified file name is valid SetFileName assigns

the file name to the specified lookup file.

See Also: GetFileName.

Comments: ModelMaker formats the specified text (see Message Text Formatting earlier in this chapter) and attempts to load the file with resulting name. If an error occurs the model run is terminated with an appropriate error message.

Example: SetFileName(Met_data, "met%i.dat", year);

TraceMessage

Purpose: To output a message containing the specified formatted text to the configured trace file.

Syntax: **TraceMessage** (level, "*Message text*", "*Title text*" [,Value...]);

Return Value: None.

See Also: TraceOn, TraceOff.

Comments: If tracing is enabled and the specified trace level is active, the formatted message text is output to the configured trace file.

The TraceMessage text field can contain both ordinary text and formatting commands which define how the following values are formatted (see Message Text Formatting earlier in this chapter).

Example: TraceMessage(1, "Tital is %f at time %f", running_total, t);

TraceOff

Purpose: To turn the specified trace levels off.

Syntax:	**TraceOff** (level [, level]);
Return Value:	None.
See Also:	TraceMessage, TraceOn.
Comments:	If tracing is enabled ModelMaker turns the specified tracing levels off so that any TraceMessage actions which specify any of the levels will be ignored and not output to the configured trace file .
Example:	TraceOff(1, 3, 5);

TraceOn

Purpose:	To turn the specified trace levels on.
Syntax:	**TraceOn** (level [, level]);
Return Value:	None.
See Also:	TraceMessage, TraceOff.
Comments:	If tracing is enabled ModelMaker turns the specified tracing levels on so that any TraceMessage actions which specify any of the levels will be processed and output to the configured trace file.

Appendix: Introduction to Marquardt and Simplex Theory

Marquardt theory

Marquardt (also known as Levenberg–Marquardt) is the method of first choice for non-linear regression, in the same way that Runge-Kutta is the first choice for solving differential equations numerically.

Before briefly outlining the Marquardt method, it is worth reviewing Taylor's series theory. This says that *any* function $f(x)$ close to a given point, x, can be approximated by the series

$$f(x+\Delta x) \approx f(x) + \frac{df}{dx}\Delta x + \frac{1}{2}\frac{d^2 f}{dx^2}\Delta x^2 + ... \qquad (1)$$

In our case the function of interest is χ^2 as a function of the model's parameters ***a***. There can be more than one adjustable parameter so ***a*** is a vector (and therefore written in bold type) and (1) is re-written as

$$c^2(\boldsymbol{a}) \approx c - \boldsymbol{d} \bullet \boldsymbol{a} + \frac{1}{2}\boldsymbol{a} \bullet \boldsymbol{D} \bullet \boldsymbol{a}$$

(2)

where ***a*** is the vector of current model parameters (N of them)
c is a constant, in fact χ^2 at the minimum
-d is a vector (N-long) representing the derivative of χ^2 with respect to the parameters ***a***
D is an $N \times N$ matrix of the double derivatives of χ^2 with respect to ***a*** (this is sometimes called the curvature matrix).

This is the Taylor expansion of the χ^2 function. It is made a little unpleasant by the vectors and matrices which arise because we can adjust more than one parameter at a time.

Differentiating this we can obtain a relationship for the gradient of the χ^2 function

$$\nabla c^2 = \boldsymbol{a} \bullet \boldsymbol{D} - \boldsymbol{d}$$

At the minimum of the χ^2 function this will be zero

$$\boldsymbol{a}_{min} \bullet \boldsymbol{D} - \boldsymbol{d} = 0$$

and at the current parameter values the gradient is

$$\nabla c^2(\boldsymbol{a}_{cur}) = \boldsymbol{a}_{cur} \bullet \boldsymbol{D} - \boldsymbol{d}$$

Subtracting these two equations eliminates ***d*** and reorganizing gives

$$\boldsymbol{a}_{min} \bullet \boldsymbol{D} \approx \boldsymbol{a}_{cur} \bullet \boldsymbol{D} - \nabla c^2(\boldsymbol{a}_{cur})$$

Multiplying through by the inverse matrix $\boldsymbol{D}^{-1}$ we have

$$\boldsymbol{a}_{min} \approx \boldsymbol{a}_{cur} - \boldsymbol{D}^{-1} \bullet [-\nabla c^2(\boldsymbol{a}_{cur})]$$

(3)

Taking stock, this equation gives the parameters of the model, $\boldsymbol{a}_{\mathrm{min}}$, which have the smallest χ^2, in terms of:

- $\boldsymbol{a}_{\mathrm{cur}}$, the current set of parameters that we have.
- $\boldsymbol{D}^{-1}$, the inverse of the curvature matrix ***D***.
- $\nabla c^2(\boldsymbol{a}_{\mathrm{cur}})$, the slope matrix for χ^2 at the current values of the parameters.

All of these terms are numerically obtainable, albeit after some potentially considerable computing. However, all of this is based upon an approximate Taylor expansion. This is *only* a good approximation if we are near the minimum to begin with. If not then the above expression is quite likely to give us a worse set of parameters than we started with. In this situation an alternative is required. One of the most popular is iterative steepest descent.

$$\boldsymbol{a}_{next} = \boldsymbol{a}_{cur} - constant \times \nabla c^2(\boldsymbol{a}_{cur})$$

(4)

where the constant should be small enough to ensure that the currently estimated slope does not run out.

Marquardt's method elegantly combines these two approaches so that the optimization process can smoothly slide between equations (3) and (4). This is done through the constant in equation (4), which is set to be equal to the appropriate diagonal element of a matrix a (defined below) and an overall constant λ. The change in the *i*th parameter a_i given by equation (4) is then

$$\Delta a_i = \frac{1}{l a_{ii}} \frac{\P c^2}{\P a_i}$$

where a is a matrix closely related to the curvature matrix ***D***,

$$a_{ii} = \frac{1}{2}(1 + l)D_{ii}$$
$$a_{ij} = \frac{1}{2}D_{ij} \qquad j \neq i$$

Equation (3) is re-written substituting a for ***D***.

$$\boldsymbol{a}_{\boldsymbol{min}} \approx \boldsymbol{a}_{\boldsymbol{cur}} - a^{\boldsymbol{-1}} \bullet [-\nabla c^2(\boldsymbol{a}_{\boldsymbol{cur}})]$$

(5)

The key feature of this result is that λ, the scaling constant for the gradient approach of equation (4), is included in the diagonal elements of the matrix a. When λ>>1 a is diagonally dominant, and applying equation (5) with a in place of ***D***, the result is completely equivalent to applying equation (4). Therefore by varying the value of λ the Marquardt method can smoothly move between a direct solution to the minimum and a gradient descent. When performing an optimization run then the user can select the initial value of lambda in the Optimization Settings dialog.

The simplex method

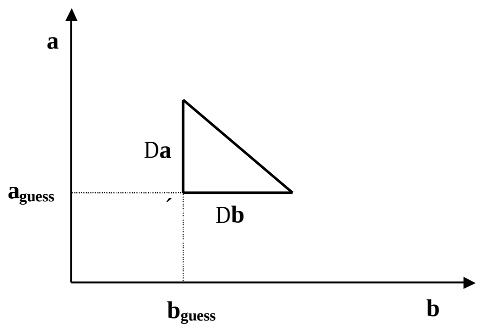
A simplex is a geometrical figure that in N dimensions has N+1 'vertices', so in two dimensions the simplex has 2 + 1 = 3 vertices and is a triangle. In three dimensions it is a tetrahedron. Each dimension represents a parameter whose value is to be adjusted. Simplex is used in two areas in ModelMaker. The first is in optimizing a model and the second is in minimizing a model component. In the former we are minimizing the difference between observed and calculated values of components. In the second we are minimizing the value of a single component. In both cases the result is achieved by changing values of parameters. What follows is a general discussion on the implementation of Simplex minimization.

In minimization the simplex is applied in a very simple fashion. The simplex is positioned by making an initial guess at the parameter values. An initial guess is always required for multi-dimensional minimization and optimization. In addition to an initial position, the 'characteristic scale' of the problem has to be guessed in order to give the sides of the simplex a length. The component or error that is to be minimized is evaluated at each of the simplex's corners. The corner which has the highest value is moved through the opposite side to create a new simplex and the process is repeated by contracting or expanding the simplex as outlined below.

This can be illustrated with an example. Consider the calculation of y which depends upon the values of the parameters a and b:

$$y = f(a,b)$$

We make a guess for initial values of a and b in order to position the first point of the simplex in the two-dimensional space (point x):



To construct the other two points for the simplex, coordinates of ($a + \Delta a$, b) and (a, $b + \Delta b$) are used as shown in the above diagram. In ModelMaker the values of Δa and Δb are calculated using the value in the *fractional change* field of the Advanced tab of the Minimization Configuration dialog box. By default, the value used is 0.001, which means that $\Delta a = 0.001 \times a$.

The function is then evaluated at each corner of the simplex and the worst (i.e. the largest) point identified. This point is then *reflected* through the opposite side of the simplex. Essentially this process is repeated until the minimization is judged to have converged. However, depending upon the value of the new corner the simplex may have expanded or contracted. If the new corner is now the best, the simplex is expanded in that direction by a factor of two. If it is the worst, the simplex is contracted by a factor of two in that direction. If it is still the worst, the simplex is contracted about the best corner.

In ModelMaker the process is judged to have converged when a certain number (by default 5) of consecutive *convergent* steps have occurred. A convergent step is one where the value being minimized gets smaller, but only by a small amount (by default less than 10%).

Index