Berkeley Madonna User's Guide

Version 8.0

March 23, 2000

Robert Macey

George Oster

Tim Zahnley

University of California

Department of Molecular and Cellular Biology

Berkeley, CA 94720

http://www.berkeleymadonna.com

Table of Contents

Introduction

Berkeley Madonna is a program that numerically solves systems of ordinary differential equations (ODEs) and difference equations. It was originally developed to execute models written in STELLA® more quickly. Over time, we added our own unique features which have made Berkeley Madonna into a fast, self-contained, and easy-to-use modeling tool.

Berkeley Madonna is available for both Macintosh and Windows platforms. While these versions are mostly identical, there are some differences which are noted throughout this guide with the notation [Macintosh] or [Windows].

System Requirements

Macintosh: Power Macintosh or compatible computer[1] with a PowerPC processor and least two megabytes of free memory. It may require more memory depending on the size of your models and the amount of data they generate. The flowchart editor requires MRJ 2.1 or later to be installed on your system.[2] You can download the latest version of MRJ from http://www.apple.com/java. Note that Mac OS 9 comes with MRJ 2.1.4. We recommend that you use MRJ 2.2 or later since it offers significantly improved performance over MRJ 2.1.x.

Windows: PC compatible running Microsoft Windows 95 or later, or Windows NT 4.0 or later. It does not work under older versions of Windows NT (such as 3.51), nor does it run under 16-bit versions of Windows (3.1, 3.11, etc) .[3] The flowchart editor required JRE 1.1 to be installed on your system.[4] You can download the latest version of JRE 1.1 from http://java.sun.com/products/jdk/1.1/jre.

Berkeley Madonna Windows

Berkeley Madonna uses various kinds of windows to represent your model including equation, flowchart, parameter, graph, datasets, and notes windows.

The Equation Window

You use the equation window to edit your model's equations. The editor is a simple plain text editor similar to SimpleText [Macintosh] or Notepad [Windows]. The equation window can be displayed at any time by choosing **Equations** from the **Model** menu.

When you create a new model using the **New** command in the **File** menu, Berkeley Madonna creates a new, untitled equation window. Similarly, when you open a plain text file, it creates a new, untitled equation window showing the file's contents.

[1] Berkeley Madonna does not run on "classic" Macintosh models with 68K processors.

[2] MRJ stands for **M**acintosh **R**untime for **J**ava.

[3] It may be possible to run Berkeley Madonna on Windows 3.1 or 3.11 by first installing Microsoft's Win32s extensions. However, this has not been tested and is not supported.

[4] JRE stands for **J**ava **R**untime **E**nvironment.

You don’t need to keep the equation window open if you don’t want to. As long as at least one other window for your model is open (parameters, graphs, etc.), you can close the equation window and your model will remain open.

Changes made to your equations do not take effect until your model has been recompiled. By default, Berkeley Madonna automatically recompiles your equations when you perform any of the following operations:

Model menu:

Modules » Boundary Value ODE

Run

Compute menu:

Check DT/TOLERANCE

Run

Graph menu:

New Graph

Choose Variables

Parameters menu:

Parameter Window

Define Sliders

Show Sliders

Batch Runs

Repeat Batch Runs

Curve Fit

Optimize

Parameter Plot

Sensitivity

However, if you turn off the **Automatically Recompile Equations** option in the **General** page of the **Preferences** dialog, Berkeley Madonna will only recompile your changed equations when you explicitly tell it to do so by choosing **Compile** from the **Model** or **Compute** menus.

By default, Berkeley Madonna will prompt you before recompiling the equations. This gives you the opportunity to either cancel the recompile operation or revert to the equations that were last compiled. You can avoid this prompting by turning off the **Confirm Before Recompiling Equations** option in the **General** page of the **Preferences** dialog.

When Berkeley Madonna recompiles your equations, all runs in memory are discarded. In addition, any settings in your model that depend on particular symbol names (such as variables in graph windows, modified parameter values, sliders, etc.) will be lost if you changed the name of the symbol in your equations.

If Berkeley Madonna finds an error when compiling your equations, an error message is displayed and the suspect text will be selected in the equation window. The Equation Syntax section on page 22 explains how to write properly-formed equations.

The equation window supports several editing features of note:

- Standard clipboard commands (cut/copy/paste) and drag-drop editing are provided.
- Text that would extend past the right edge of the window is automatically wrapped to the next line; there is no need for a horizontal scroll bar and Berkeley Madonna doesn't provide one.
- The **Balance** command in the **Edit** menu selects the text within the innermost matching pair of parentheses or brackets. The search for a matching pair of characters starts at the current insertion point or selection and expands outward. This feature helps you to locate sub-expressions nested within complex equations.
- The **Find…** and **Replace…** commands in the **Edit** menu help you search for and replace text in your equations.
- The **Settings** dialog (**Model** menu) allows you to change the font used to display your model's equations. This font is also used for axis labels in your model's graph windows.
- The **Save Equations As…** command in the **File** menu allows you to save your model's equations as a plain text file. This is useful for transferring your equations to another program. Note that this command is available only when an equation window is active.
- The **Print Equations…** command in the **File** menu allows you to print your model's equations on the current printer. You may want to use the **Print Preview** command first to verify that the margins are appropriate and to determine how many pages would be required to print them. This command is available only when an equation window is active.
- The **Insert Picture** [Macintosh] or **Insert Object** [Windows] command allows you to embed pictures and objects within your equations. See Embedding Pictures and Objects on page 17. Note that embedded objects are treated by the compiler as if they were blanks (space characters). Therefore, they have no effect on the semantics of your model's equations.

By default, newly created models use the Geneva 12 [Macintosh] or Arial 10 [Windows] font for their equations. You can change this default in the **General** page of the **Preferences** dialog.

The Flowchart Window

The flowchart window provides an alternative to the equation window for constructing models. Instead of typing in equations by hand, you build models by dragging icons from a toolbar onto a flowchart and connecting them with arcs to represent dependencies. As you construct your model graphically, Berkeley Madonna generates textual equations representing your model's structure.

To create a new visual model, choose **New Flowchart** from the **File** menu. This opens an empty flowchart window on which you construct your model. Unlike plain-text models (created with the **New** command), a visual model's equations cannot be edited directly in the equation window. Instead, equations are edited within each icon's "icon dialog" which is opened by double-clicking the icon.

The flowchart window can be hidden by choosing **Hide Flowchart** from the **Flowchart** menu. This can be done as long as some other non-flowchart window for this model is open (such as a graph window or the equation window). The command then changes to **Show Flowchart** which makes the flowchart window visible again.

Visual models can be converted into plain-text models by choosing **Discard Flowchart** from the **Flowchart** or **Model** menus. After discarding the flowchart, the model's equations can be edited directly in the equation window. Use caution as the flowchart cannot be recovered if you save a model after its flowchart has been discarded.

Berkeley Madonna cannot convert plain-text models into visual models for you. However, you can create a new visual model that duplicates the behavior of an existing plain-text model.

Refer to "Building a Visual Model" in the *Berkeley Madonna Tutorial* to familiarize yourself with basic techniques for constructing and editing visual models. After you're comfortable with the basics, refer to the Flowchart Reference on page 55 to learn about more sophisticated flowchart features.

The Parameter Window

You can display the parameter window by choosing **Parameter Window** from the **Parameters** menu. This window allows you to change parameters in your model as well as the integration method[5] without recompiling your model.

Parameters are symbols whose values do not depend on any other symbols in your model and do not change over time. Berkeley Madonna defines a set of built-in "system parameters" that exist in every model (STARTTIME, STOPTIME, DT, etc). These parameters always appear at the beginning of the list in the parameter window.

To change the value of a parameter, select it with the mouse and type in a new value.[6] You can enter any real number or mathematical expression in Berkeley Madonna syntax. The resulting value is displayed next to the parameter's name in the list. If you enter an invalid expression, the value in the list won't change.

Berkeley Madonna displays an asterisk (*) next to the name of each parameter that has been changed from its default value as specified in the equations. You can return a parameter to its default value by selecting it and clicking the **Reset** button.[7]

The sliders window provides another convenient way to change parameters and automatically run your model. See Sliders on page 18 for details.

[5] The integration method can also be changed via the **Integration Method** submenu in the **Compute** menu.

[6] In the Windows version, you must double-click the parameter in order to transfer focus to the edit field so that you can change the parameter's value. You can also click the edit field with the mouse or press the tab key to transfer focus.

[7] In the Macintosh version, you can also reset a parameter by double-clicking its list entry while holding down the option key.

The Graph Window

When you run your model for the first time, Berkeley Madonna automatically creates a graph window. The window title indicates the run number and the variables plotted on the X and Y axes. If **Overlay Plots** is off, each subsequent run will replace the data from the previous run.

If you turn on **Overlay Plots**, subsequent runs will be added to the graph window. You can discard the most recent run by choosing **Discard Last Run** from the **Graph** menu. To discard all the runs shown in the window, choose **Discard All Runs** from the **Graph** menu.

Creating Multiple Graph Windows

Berkeley Madonna allows you to create more than one graph window for your model. This feature is useful if you want to view several variables with very different scales. To create another graph window, you have two choices. First, you can create an empty graph window by choosing **New Window** from the **Graph** menu and add variables to it using the **Choose Variables** command. Note that new graph windows don't contain any run data and remain empty until you run your model again.

You can also create additional graph windows by selecting an existing graph window and choosing **Duplicate Window** from **Graph** menu. A graph window created in this way contains the same runs, variables, and other settings as the window from which it was duplicated.

When you run your model, Berkeley Madonna stores the results in all unlocked graph windows. If you want to preserve the contents of a graph window when you run your model, simply depress the **Lock** button. Locked windows are never modified when you run your model.

For example, say you want to run your model twice and display the results of each run in a separate window. This can be accomplished by locking the graph window after performing the first run. Then, when you perform the second run, Berkeley Madonna creates a new graph window because no unlocked graph windows exist in which to store the new run.

The **Discard Last Run** command discards the last run shown in the active graph window. This run will also be discarded from all other unlocked graph windows in which it appears. The **Discard All Runs** command discards all runs shown in the active graph window. These runs will also be discarded from all other unlocked graph windows in which they appear. If a locked graph window is active and you attempt to discard runs, Berkeley Madonna asks you to confirm the operation.

Specifying Which Variables to View

To change which variables are plotted, choose **Choose Variables** from the **Graph** menu.[8] You can specify any number of variables to plot on the Y axes. Each variable's scale and label can be shown on the left-hand or right-hand Y axis. To change which axis is used, select the

[8] You can also open the Choose Variables dialog by double-clicking the plot area within the graph window.

variable and toggle the **Right Axis** check box. To hide or show a variable, select it and toggle the **Visible** check box.

By default, Berkeley Madonna plots TIME on the X axis. You can change this to any scalar variable in your model using the X axis control in the **Choose Variables** dialog. If you're working with arrays, you can plot their final values (i.e., when TIME = STOPTIME) against the element index by choosing **[i]** in the X axis control.

When you specify new variables for your plot, they may not appear after you click OK in the **Choose Variables** dialog. This happens because Berkeley Madonna only stores the variables in your model that appear in the Y Axes list for at least one graph window at the time your model is run. If you add new variables after the run, you'll need to run your model again so they're stored in memory.

For example, say your model has three variables: A, B, and C. Assume that the first time you run your model, you specify variable A on the Y axis. Berkeley Madonna stores values of A for this run, but not the values of B or C. Then, if you change the graph to plot B and C in addition to A, only a curve for A will appear. If you run your model again, A, B and C will be stored and will appear in the plot.

If you find yourself working with a group of variables but don't want to view all of them at once, use the variable buttons at the bottom of the graph window (available when the **Variable Buttons** command in the **Graph** menu is checked). Clicking a variable button shows or hides the variable. Holding the shift key while clicking a variable button moves the variable from one Y axis to the other.

Holding the option key [Macintosh] or control key [Windows] while clicking a variable button sets the X and Y axis scales to optimally display the selected variable. Note that this operation turns off automatic scaling for these axes. To restore autoscaling for all axes, hold down the option or control key while clicking one of the axes.

Changing Axis Settings

By default, Berkeley Madonna automatically chooses the scale for each axis in your plot. You can override automatic scaling by choosing **Axis Settings** from the **Graph** menu and selecting the **Scales** tab.[9] When one of the **Auto** boxes is checked, the settings for that axis are grayed because Berkeley Madonna is choosing them for you. To change them, you must first uncheck the **Auto** box. The **#Div** field specifies the number of divisions the axis is divided into by the grid lines.

Checking a **Log** box in the **Scales** page of the **Axis Settings** dialog changes the scaling for that axis from linear to logarithmic. If the **Auto** box is checked, Berkeley Madonna will choose the axis range for you using exact powers of ten. If you uncheck the **Auto** box, you can specify whatever limits you want. Note that when using logarithmic scales, Berkeley Madonna does not distinguish between positive and negative values. Also, zero values are not plotted at all since the logarithm of zero is infinite.

[9] You can open the Scales page of the Axis Settings dialog in one step by double-clicking any axis in the graph window.

You can quickly "zoom in" to view a portion of your data. To do this, click and drag the mouse to draw a rectangle over the portion you wish to view. When you release the mouse, Berkeley Madonna adjusts the axis limits so that the portion that was in the rectangle now fills the window.

When you zoom in, Berkeley Madonna adjusts the axis scale limits to keep them nice and round. If you want the new scale to be exact (i.e., without rounding), hold down the option key [Macintosh] or control key [Windows] while you release the mouse.

To undo the effect of a zoom-in operation, click the **Zoom Out** button. Each click of this button undoes the effect of the most recent zoom-in operation. Holding down the option key [Macintosh] or control key [Windows] while clicking the **Zoom Out** button undoes the effect of *all* zoom-in operations.

Any changes made in the **Scales** page of the **Axis Settings** dialog while zoomed-in are lost when you zoom out. To prevent this from happening, zoom out all the way before making changes to the axis scales.

By default, Berkeley Madonna chooses the text used to label the axes based on the variables you're plotting. You can override the default by choosing **Labels** tab in the **Axis Settings** dialog.[10] Uncheck the **Default** box for each label you want to change and edit its text.

Creating Multiple Pages [Windows only]

The Windows version can display multiple plots in a single graph window. Each plot is referred to as a "page". Only one page (the active page) is visible at a time. To select a different page, click its tab at the top of the window.

Newly-created graph windows always start out with a single page. You add pages by clicking the **New Page** button in the toolbar. You can delete the active page by clicking the **Delete Page** button. Note that you cannot delete the last page.

Settings for new pages (variables, colors, etc.) are copied from the page that was active when you clicked the **New Page** button. You can change the plotted variables, axis scales, and most other settings independent of other pages.

All pages in a given window display data from the same runs as shown in the window's title bar. When you run your model, the new run will be displayed in all pages. If you discard a run, it will be discarded from all pages. If you lock the window or turn on **Overlay Plots**, all pages in the window will be affected.

Using Graph Buttons

The buttons in the top-left corner of the graph window perform the following functions:

Run **Run.** Runs the model once. Same as choosing **Run** from the **Model** or **Compute** menus, except for parameter plot windows where this button performs a parameter plot run.

[10] You can open the Labels page of the Axis Settings dialog in one step by double-clicking any axis label in the graph window.

New Page. [Windows only] Creates a new page in the graph window.

Delete Page. [Windows only] Removes the active page from the graph window.

Lock. Prevents the window's contents from being changed when running the model or discarding runs in other windows.

Overlay Plots. When enabled, new runs are added to existing runs in the window. Otherwise, existing runs are discarded before new runs are stored.

Table. Displays the contents of the window in tabular form. Note that the TIME column reflects the time for the run shown in the rightmost column of the table. If there are other runs with different time scales, their values will be mapped to the times shown in the TIME column using linear interpolation.

FFT. Displays the contents of the graph window in the frequency domain. See Fast-Fourier Transform on page 15.

Legend. Displays the legend. The legend shows the variable name and run number for each curve. Additional information in parentheses is included for Batch Runs: the parameter value for separate runs and labels for mean and mean±sd runs. You can place the legend anywhere in the graph window by dragging it with the mouse.

Parameters. Displays the parameter list. This list shows the value of each parameter when the model ran. If there is more than one run in memory and a parameter is not the same in all runs, the minimum and maximum values of the parameter are shown. You can position the parameter list with the mouse just like the legend.

Oscilloscope Mode. Displays the data like an oscilloscope. This button is present only if the "trigger" symbol is defined in your model. See Oscilloscope Mode on page 14.

Colors. Displays each curve in a different color.

Dashed Lines. Displays each curve in a different line style (solid or dashed).

Data Points. Draws a solid dot centered over each point in your data. If you have many closely-spaced data points, this has the effect of making the curves very thick. They will also redraw more slowly when this option is enabled. Tip: if you hold the option key [Macintosh] or control key [Windows] while clicking this button, the lines connecting the data points will be hidden.

Grid. Shows the grid lines.

Readout. Displays X and Y values as you drag the mouse over the plot, instead of zooming in. See Readout on page 14.

Initial Conditions. Enables you to set initial conditions by dragging the mouse over the plot. See Initial Conditions on page 15.

Zoom Out. Undoes the effect of the last zoom-in operation. If no zoom-in operations have been performed, this button is grayed.

Some of the graph window options can be turned on or off by default for new graph windows by editing the **Graph Windows** page of the **Preferences** dialog.

The Macintosh version allows you to customize the appearance of these buttons. The **Graph Buttons** page of the **Preferences** dialog allows you to specify whether large (12x12 pixels) or small (8x8 pixels) buttons are used. You can also hide buttons that you don't use to reduce clutter.

Printing Graphs and Tables

You can print a graph or table by activating its window and choosing **Print Graph** or **Print Table** from the **File** menu. Before printing, however, you should choose **Page Setup** from the **File** menu to ensure that your output will appear as desired.

On Windows, you control the position of the output on the printed page by specifying margins in the **Page Setup** dialog. On Macintosh, use the **Margins** dialog (**File** menu) to specify the location and size of the output.

You can preview the appearance of your graph or table before printing by choosing **Print Preview** from the **File** menu. On Macintosh, you can adjust the page setup and margins while print preview is active. On Windows, however, you must close the print preview window before making changes to your page setup.

Exporting Graphs and Tables

You can copy and paste graphs into other applications in PICT format [Macintosh] or bitmap and enhanced metafile formats [Windows]. To copy a graph to the clipboard, activate its window and choose **Copy Graph** from the **Edit** menu.

On Macintosh, you can create a copy of the graph by clicking in any 'inactive' area of the graph window and dragging the copy to the Finder or other drag-aware application. The inactive area of a graph window includes the entire window except for buttons, the legend and parameters box, and the data area of the plot (the area inside the axes).

You can save graphs as picture files by choosing **Save Graph As** from the **File** menu. Note that you must first activate the graph window you want to save. On Macintosh, graphs are saved as PICT files. On Windows, they are saved as bitmap files (.bmp) or enhanced metafiles (.emf). Select the desired format using the **Save as type** box in the **Save Graph As** dialog.

Tables can be saved as tab-delimited text or comma-delimited text (CSV) files by activating the window containing the table and choosing **Save Table As** from the **File** menu. Tables can also be copied to the clipboard using the **Copy Table** command in the **Edit** menu and, on Macintosh, dragged to another application or the desktop. Tables are always copied in tab-delimited text format.

Customizing Background Colors [Macintosh only]

In the Macintosh version, you can change the background color of graph windows and tables by clicking the color swatches on the **Graph Windows** page of the **Preferences** dialog. To restore the default background colors (gray for graph windows, light yellow for tables), click the **Reset** button.

Oscilloscope Mode

Oscilloscope Mode causes the graph to start drawing at zero on the X axis whenever a trigger event occurs. To use this feature, define a variable in your model named “trigger”. Berkeley Madonna recognizes a trigger event whenever this variable changes from zero to any nonzero value. For example, to trigger whenever the voltage Vm is greater than 15 millivolts, add this equation to your model:

trigger = Vm > .015

Compile and run your model. Then, click the **Oscilloscope Mode** T button. The variable(s) you chose will be displayed as one or more curves starting at time zero depending on how many trigger events occurred. To restore normal (non-oscilloscope) mode, click the **Oscilloscope Mode** button again.

Note that oscilloscope mode is temporarily disabled when the Fast Fourier Transform is used.

Run Information

By default, Berkeley Madonna displays the number of steps and elapsed time of the last run in the upper right-hand corner of the graph window. The number of steps reflects how many times your model was stepped forward in time after initialization. This number is one less than the number of data points shown in the graph or table since it doesn’t include the initialization step. However, this is not true if you use DTOUT to reduce the number of steps stored in memory since it doesn’t change the stepsize used to advance your model. See Using DTOUT on page 47.

When performing batch runs with averaging, the run information indicates the total number of steps and elapsed time for <u>all</u> of the individual runs used to create the resulting average. The same totaling also applies to parameter plot runs.

Run information is not shown when either the Readout or Initial Conditions features are active.

Readout

Berkeley Madonna can display the coordinates of any point in the graph window. Activate readout mode by depressing the **Readout** ✣ button, then drag the mouse around inside the data area of the plot. As you drag, the “crosshair” cursor tracks the mouse and the corresponding values are displayed in the information area at the top of the window.

When Fast Fourier Transform mode is active, the readout displays both frequency and period corresponding to the X axis coordinate.

Note that you must deactivate readout mode in order to use the “zoom in” feature.

Initial Conditions

The **Initial Conditions** button provides a convenient way to set the initial values of two reservoirs or difference equations in your model by dragging the mouse over their X-Y phase plane. To use this feature, you must:

- Set the X and Y axis variables to reservoirs or difference equations in your model. The initial value expressions for these variables must be parameters that can be accessed from the parameter window. For example, if you plot reservoir “A” on the X axis and reservoir “B” on the Y axis, both “INIT A = ...” and “INIT B = ...” must appear in the parameter window.
- Manually set the scale for both the X and Y axes using the **Axis Settings** dialog.

Once you’re plotting the proper variables on each axis and have defined the scales, the **Initial Conditions** button will be enabled. Click it and the readout cursor will appear. Drag the cursor to a point in the phase plane and release the mouse button. Berkeley Madonna will set the initial values of the X and Y axis variables and run your model.

Fast-Fourier Transform

When investigating oscillating systems, it is useful to be able to find the period or frequency of a periodic solution. One way to do this is by looking at the Fourier transform of a time series run, which gives the contribution to the data from each periodic component.

To view the Fourier transform of your data, click the **Fast Fourier Transform** button in the graph window. The data will be transformed into the frequency domain. To find the period and frequency, activate readout mode by clicking the **Readout** button, place the mouse pointer over the lowest frequency peak, and press the mouse button. The frequency, period, and amplitude will be displayed in the upper-right corner of the window.

Once you have transformed your data, Berkeley Madonna keeps the frequency data in memory so you can quickly switch between the time and frequency domains. It also keeps track of the axis scales used in the time and frequency domains separately so you don’t have to change the scale every time you switch between domains.

When a run is saved in a graph window where **Fast Fourier Transform** is active, the run will be transformed automatically as it is saved. If you don’t want this to happen, switch back to the time domain or lock the window.

The Datasets Window

You can import numerical data from a text file into your model. The datasets window lists the datasets you’ve imported. Datasets can be used as targets for curve fitting (see Curve Fitter on page 51) and as piecewise-linear functions in your model’s equations (see Dataset Functions on page 40).

Importing Datasets

To import a dataset into your model, choose **Import Dataset** from the **File** menu. Or, open the datasets window by choosing **Datasets** from the **Model** menu and click the **Import** button. Then, choose a text file containing the numerical data you want to import. The data must be in tab-delimited text format or comma-separated values (CSV) format. Data files in this format can be generated by spreadsheet programs such as Microsoft Excel by specifying the appropriate file type in the “Save As” dialog.

Berkeley Madonna assumes that the data file consists of a rectangular array of numerical values (integers and/or decimal numbers). Rows are read sequentially starting at the beginning of the file. Values in each row are read left to right until either the end of the row is reached or a non-numerical value (a letter or symbol, for example) is read. Berkeley Madonna then checks that this row contains the same number of numerical values as the previous rows. If it does not, you’ll get an error message stating that the array is not square and the import operation will be aborted. Otherwise, Berkeley Madonna continues reading rows until the end of the file is reached. Note that rows containing no numerical values are ignored. This makes it possible to have a line of text in your data file (such as a title).

After Berkeley Madonna has finished reading in the data file, it displays the **Import Dataset** dialog where you specify the name by which the dataset is referred to in your model and the type of dataset (vector or matrix) to create.

Vector Datasets

Vector datasets (also knows as one-dimensional datasets) map an input value (X) into an output value (Y). When creating a vector dataset, you specify which columns in your imported data file to use for the input domain (X column) and output range (Y column). The X column must consist of monotonically increasing values (i.e., the value in row n+1 must be greater than the value in row n).

The datasets window lists each vector dataset with its name followed by the number of points in parentheses.

After you’ve created a vector dataset using the **Import Dataset** command, Berkeley Madonna automatically plots the dataset in a graph window. However, if you create the dataset using the **Import** button in the datasets window, the dataset is not automatically plotted. You can manually add the dataset to a graph window using the **Choose Variables** dialog.

Matrix Datasets

Matrix datasets (also known as two-dimensional datasets) map two input values (X and Y) into an output value (Z). Berkeley Madonna uses your *entire* imported data file to construct the dataset using the following scheme. For example, assume you’ve imported a data file with 35 elements in five rows and seven columns:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \end{bmatrix}$$

Berkeley Madonna uses elements in the first column (a_{21} through a_{51}) to define the dataset’s X domain and the elements in the first row (a_{12} through a_{17}) to define the dataset’s Y domain. The remaining elements (a_{22} through a_{57}) define the dataset’s output range (Z) for the corresponding X and Y input values. Note that although element a_{11} is not used, it must still be present in your data file or Berkeley Madonna will refuse to import the file because it is not “square”.

The datasets window lists each matrix dataset with its name followed by the dimensions of the dataset in square brackets.

Note that matrix datasets cannot be plotted directly in graph windows. This is because Berkeley Madonna doesn’t have a 3D plotting capability. However, you can still get an idea of what a matrix dataset looks like using the technique described in Plotting Matrix Datasets on page 40.

Modifying and Removing Datasets

A dataset’s name can be changed by selecting it in the datasets window and clicking the **Rename** button.

A dataset can be removed from your model by selecting it in the datasets window and clicking the **Discard** button. You may want to remove datasets that you are no longer using since they can significantly increase the size of your model file depending on the number of elements.

To modify the contents of a dataset, simply import a new dataset and give it the same name as the existing dataset. Berkeley Madonna will confirm that you want to replace the contents of the dataset.

The Notes Window

The notes window lets you save text and graphic notes in your model without cluttering your equations with comments. Notes can be displayed at any time by choosing **Notes** from the **Edit** or **Model** menus. Like the equation window, the notes window supports drag & drop text editing, find/replace, embedded pictures and objects, etc. You can also print your notes or save them as a text file.

Unlike the equation window, you can mix multiple styles of text within your notes by using the font name, size, and style controls at the top of the notes window.

Embedding Pictures and Objects

Pictures [Macintosh] or objects [Windows] can be embedded in your notes. They can also be embedded directly in your equations.

There are several ways to embed pictures or objects. The **Insert Picture** [Macintosh] or **Insert Object** [Windows] command in the **Edit** menu inserts pictures (PICT files) or objects into the active notes or equation window. You can also paste pictures and objects from the clipboard or drag them from other applications.

Once pictures or objects have been embedded, they can be manipulated like any character in your text. On Windows, objects can be “activated” by double-clicking them. Typically, activating an object opens the application which created it so it can be edited.

Support for embedded objects in the Windows version is not very thorough. For example, clicking the right mouse button on an object doesn’t display a context menu like it should. If you resize an object, the change is lost once you close and re-open your model. These limitations may be addressed in a future release of Berkeley Madonna.

Embedded graphics and objects are stored in platform-dependent formats which are not supported on other platforms. For example, if you open a Windows-created model on the Macintosh, any embedded objects will be replaced by spaces.

Running Models

Single Run

The simplest way to run your model is to click the **Run** button in the equation, parameter, or graph windows. Or you can choose **Run** from the **Model** or **Compute** menu. The result of the run is displayed in a graph window. If no graph exists, Berkeley Madonna will create one and plot the first eight variables[11] in your model.

Each time you run your model, the data from the new run will replace the data from the previous run. If you want to display more than one run at a time, depress the **Overlay Plots** button.

When **Overlay Plots** is on, each new run is added to the previous runs in memory. The title of the graph window indicates how many runs are in memory.

When **Overlay Plots** is off, Berkeley Madonna discards all existing runs in memory when a new run is completed.

Continue Run [Windows only]

The Windows version provides the ability to continue the last single run from where it was stopped. For example, you can stop a run in progress, change a parameter, then resume from where it was stopped. You can also extend a completed run by increasing STOPTIME before continuing. To continue a run, choose **Continue** from the **Compute** menu.

Sliders

Sliders provide a quick way to change a parameter and run the model in one step. To create sliders, choose **Define Sliders** from the **Parameters** menu. In the dialog, select a parameter and add it to the sliders list by clicking **Add**. Select linear or logarithmic scaling and adjust the minimum, maximum, and increment or multiplier for the slider. If you want additional sliders, add them now. Click OK. Berkeley Madonna will display a floating window containing the sliders you defined.

To change a parameter via a slider, drag the indicator or click the arrows. When the mouse is released, the change will be applied to the parameter and the model will automatically run.

The scaling of an individual slider can be quickly changed by double-clicking anywhere on the slider's text (where the parameter name and value are displayed). This action opens the **Define Sliders** dialog and selects the slider you clicked. The **Define Sliders** dialog can also be opened when no sliders have been defined by double-clicking anywhere in the empty sliders window.

[11] Only the first two variables will be visible. To show the other variables, use the variable buttons at the bottom of the graph window.

Each slider has a check box labeled **10x** which reduces the range of the slider by a factor of ten around its current value. This feature makes it easier to zero in on a desired value. Note that this 10x mode is cancelled if you make any changes via the **Define Sliders** dialog or recompile your model.

You can hide the sliders window whenever you want to clicking its close box or choosing **Hide Sliders** from the **Parameters** menu. You can bring the previously-defined sliders back at any time by choosing **Show Sliders** from the **Parameters** menu.

Batch Runs

Berkeley Madonna can automatically run your model multiple times while optionally varying a specified parameter. To use this feature, choose **Batch Runs** from the **Parameters** menu. Berkeley Madonna displays the **Batch Runs** dialog. Set it up as follows:

1. Specify the parameter you want to vary for each run from the **Parameter** control. If you don't want Berkeley Madonna to change a parameter, choose None.
2. Specify the number of runs you want to perform. This number must be between 2 and 1000.
3. Specify the values you want the parameter to take for the first and last run. Enter these values in the **Initial Value** and **Final Value** fields, respectively. Note that these values must be different.
4. Specify whether the parameter should be varied according to an arithmetic or geometric series by clicking one of the **Series Type** radio buttons. Note that the initial and final values for a geometric series must be nonzero and the same sign.
5. Specify how Berkeley Madonna should process the runs. There are three choices:
    - **Keep Runs Separate.** Berkeley Madonna stores each run separately in memory and adds them to the frontmost unlocked graph window as they complete. When all the runs have completed, all unlocked graphs will be redrawn showing each run.
    - **Compute Mean.** Berkeley Madonna computes the mean of the runs and stores the result as a single run.
    - **Compute Mean ± SD.** Berkeley Madonna computes the mean and standard deviation of the runs and stores the result as three runs: the first run is the mean, the second is the mean minus the standard deviation, and the third is the mean plus the standard deviation. In other words, the second and third runs show the bounds of the area extending one standard deviation on each side of the mean.
6. Click OK to start the runs.

Note that Berkeley Madonna will add the resulting run(s) to the graph if **Overlay Plots** is on. If you want any previous runs to be discarded, be sure to turn off **Overlay Plots** before starting the runs.

You can easily repeat the previous Batch Runs command by choosing **Repeat Batch Runs** from the **Parameters** menu.

Parameter Plot

The Parameter Plot feature enables you to plot the result of each run as a single data point over a range of parameter values. Berkeley Madonna provides a variety of methods to convert a variable's value over time into a single value. The initial, final, minimum, maximum, mean, and standard deviation calculations are self-explanatory. The oscillation frequency and amplitude calculations consist of performing an FFT on the run data and selecting the frequency component with the maximum amplitude.

To perform a parameter plot, choose **Parameter Plot** from the **Parameters** menu. This opens the **Parameter Plot** dialog where you specify the parameter to vary and the variables to plot. As part of selecting the parameter to vary, you specify the number of runs (steps), initial value, final value, and series type. This setup is identical to that used in the **Batch Runs** dialog.

The variables portion of the **Parameter Plot** dialog is similar to the **Choose Variables** dialog that you're already familiar with: you select the variables you want to plot and add them to the Y Axes list. However, for each variable, you also must select at least one of type of output conversion to generate a single value from the run data. Simply check the appropriate box (or boxes) to the right of the **Y Axes** list. As each box is checked, the variable name in the list reflects the type of conversion (for example, 'min' for minimum).

Once you've specified the parameter to vary and the variables to plot, click the **Run** button in the dialog. Berkeley Madonna performs the series of runs and plots the result in a new graph window. If you don't want to run the model yet, click **Don't Run** and Berkeley Madonna will create the graph window without performing the runs. You can perform the runs later by clicking the **Run** button in the graph window or the **Parameter Plot** dialog.

Since the independent variable in a parameter plot is a parameter in your model rather than TIME, Berkeley Madonna displays the result of the run in a special graph window known as a parameter plot graph window. Such windows can be identified by the text "vs. XXX" in the title bar, where XXX is the name of the specified parameter (as opposed to TIME for normal windows). Parameter plot graph windows are different from normal graph windows in other ways, as well:

- They can only display results of parameter plot operations based on the parameter shown in the title bar.
- The X axis variable cannot be changed.
- Since the window's contents are not based on TIME, you cannot perform a Fast Fourier Transform on them or use the oscilloscope mode feature.
- The **Run** button performs the last parameter plot operation as specified in the Parameter Plots dialog for this parameter. This makes it easy to perform the Parameter Plots operation again (for example, after changing some other parameter in the parameters window) without invoking the **Parameter Plot** dialog.

Once you have a parameter plot graph window, you can easily change the parameter's range and/or variables plotted by invoking the **Parameter Plot** dialog again. This can be done in several ways:

- Double-click anywhere in the plot area of the graph (the same action that displays the **Choose Variables** dialog in normal graph windows).

- Activate the parameter plot graph window and choose either **Choose Variables** from the **Graph** menu or **Parameter Plot** from the **Parameters** menu.

When the **Parameter Plot** dialog is opened for an existing parameter plot graph window, you'll note that the **Create New Window** box in the lower-left corner is enabled and not checked. This means that any changes you make to the list of variables will be applied to the graph window for which the dialog was invoked. If you would rather not modify this window, check this box and Berkeley Madonna will create a new parameter plot graph window.

If you change the parameter in the Parameter Plot dialog, Berkeley Madonna checks to see if a parameter plot graph window for the selected parameter exists. If such a window is found, the **Create New Window** box remains enabled so that you can modify this window rather than create a new one. If such a window cannot be found, Berkeley Madonna must create a new graph window so the box will be checked and disabled.

If you choose to modify an existing window and that window is locked, the **Run** button in the dialog will be disabled. To get around this, either unlock the graph window before invoking the **Parameter Plot** dialog or check the **Create New Window** box.

Check DT

The **Check DT/TOLERANCE** command in the **Compute** menu helps you verify that you're using a sufficiently small value of DT (when using fixed-stepsize integration methods) or TOLERANCE (when using variable-stepsize integration methods) for accurate results. Berkeley Madonna performs this check by running your model twice: first with the current value of DT or TOLERANCE and second with DT or TOLERANCE reduced by a factor ten. After both runs have been completed, Berkeley Madonna displays the **Check DT Report** or **Check TOLERANCE Report** dialog. These dialogs show the maximum and root-mean-square deviation between the two runs for each stored variable. Both runs are also stored in all unlocked graph windows so they can be compared visually.

Floating-Point Exceptions

If you get a floating-point exception when running your model, it means some numerical operation resulted in an error. Exceptions include division by zero, numerical overflow, taking the square root of a negative number, etc.

When the **Stop On Exception** box in the **Settings** dialog (**Model** menu) is checked (the default), Berkeley Madonna checks for exceptions after each time step has been calculated and its results stored in memory. If an exception has occurred, the run is stopped and the exception is reported.

When the **Stop On Exception** box is not checked, Berkeley Madonna checks for exceptions after the final step has been completed (STOPTIME has been reached). In this case, it may not be possible to determine exactly when the first exception occurred.

Berkeley Madonna usually gives you the option of re-running your model after a floating-point exception occurs.[12] If you choose to retry the failed run, Berkeley Madonna divides DT by two (for fixed-stepsize methods) or TOLERANCE by ten (for variable-stepsize methods) and runs your model again. If an exception occurs during this run, you are again given the option of reducing DT/TOLERANCE and re-running your model. This sequence continues until the run completes without errors or you choose not to retry the failed run.

When you choose to retry a failed multiple run sequence, Berkeley Madonna restarts the sequence with the first run. You can observe this behavior by watching the **Running** dialog as you click **Yes** to retry.

When you choose not to re-run your model after a floating-point exception occurs, Berkeley Madonna will store the failed run. Since errors were detected during the run, its result data may contain non-finite values such as infinity or NAN (not-a-number).

Non-finite values are not displayed in graphs, so to find out where things started going wrong it may help to view the results in table form. Unlike graphs, tables display invalid numerical values and highlight them in red so they can be easily spotted.

Equation Syntax

To display a summary of Berkeley Madonna's equation syntax, choose **Equation Help** from the **Help** menu. You can select text in this window and drag it to your equation window. The equation help window can also be printed or saved as a text file by activating it and choosing **Print Equation Help…** or **Save Equation Help As…** from the **File** menu.

Basic Syntax

Berkeley Madonna models consist of a list of equations. Equations can span more than one line, and you can put multiple equations on a single line. The order in which equations appear is generally unimportant. Most equations are of the form:

variable = expression

Variable names consist of one or more characters. You may use letters (A-Z), digits (0-9), and underscore (_) in variable names. However, variable names must not begin with a digit.

On Macintosh, you can also use the dollar sign ($), at sign (@), and many extended characters outside the normal US ASCII character set. This feature allows you to use special symbols like Greek letters found in specialized fonts.

A variable can be given almost any name as long as it doesn't clash with any of Berkeley Madonna's built-in function or symbol names (see Built-in Functions on page 34 and Built-in Symbols on page 31).

Comparisons between variable names are case-insensitive. For example, the names "Result", "RESULT", and "result" all refer to the same variable.

[12] The curve fitter, optimizer, and boundary value solver do not provide the option to reduce DT/TOLERANCE and retry. If these algorithms detect floating-point exceptions from which they cannot recover, they stop executing and leave the parameters set to the values that caused the exception.

Expressions use standard infix notation. That is, arithmetic and relational operators are placed between their two operands like this:

```
result = a + b * c
```

The usual operator precedence rules apply. For example, in the above equation, b and c are multiplied first, then their product is added to a. Parentheses can be used to force different order of evaluation. For example:

```
result = (a + b) * c
```

Berkeley Madonna provides a number of built-in functions. The name of the built-in is followed by one or more arguments enclosed in parentheses. However, functions which take no arguments (e.g., PI) are not followed by parentheses. For example:

```
result = a * SIN(2 * PI * x)
```

Multiple function arguments are separated by commas like this:

```
result = SUM(a, b, c) / n
```

You can place comments anywhere in the equations by enclosing them in curly brackets. For example:

```
{This is a sine wave}
result = a * SIN(x)
{This is the amplitude}
a = 7.5
```

Curly-bracket comments can span multiple lines and can be nested, like this:

```
a = ...     {define a}
{the following two lines have been disabled by this comment:
b = ...     {define b}
c = ...     {define c}}
d = ...
```

Berkeley Madonna also recognizes single-line comments beginning with the semicolon character. This type of comment extends through the end of the line and doesn't require an "end of comment" character like curly brackets do. For example:

```
a = ...     ;define a
b = ...     ;define b
c = ...     ;define c
```

Differential Equations

Equations like those shown above simply assign the value of the expression on the right-hand side to the variable on the left-hand side. Such equations are referred to as "formulas" by the flowchart editor.

Ordinary differential equations (known as "reservoirs" in the flowchart editor) are defined by two equations: an initializer equation and an integrator equation. The initializer equation determines the initial value of the reservoir. The integrator equation determines how much the reservoir’s value increases or decreases during each time step (also known as the inflow or outflow).

Berkeley Madonna supports different forms of initializer and integrator equations. This flexibility allows Berkeley Madonna to compile equations generated by STELLA. The following three forms of initializer syntax are functionally identical; each initializes reservoir R to an initial value denoted by “…”:

```
R(STARTTIME) = …
INIT R = …
INIT (R) = …
```

The following five forms of integrator syntax are also functionally identical; each defines a net flow into reservoir R denoted by “…”:

```
d/dt(R) = …
R′ = …
FLOW R = …
R(t) = R(t - dt) + (…) * dt
R = R + dt * (…)
```

The last two forms are provided for compatibility with STELLA. We don’t recommend their use since the notation is more error-prone.

Since the different forms of these equations are equivalent, you can use whichever you like in your own equations.

The second integrator syntax shown above (R′ = …) deserves special mention. Unlike other notations, this “prime” notation allows you to define higher-order systems without explicitly writing their equations. For example, you can define a third-order system like this:

```
u′′′ = 2*u′′ - 5*u′ + u - 1
```

Berkeley Madonna internally translates this into a system of three first-order equations like this:

```
u′′′ = 2*u′′ - 5*u′ + u - 1
d/dt(u′′) = u′′′
d/dt(u′) = u′′
d/dt(u) = u′
```

The generated equations, although not shown in the equation window, do exist behind the scenes. This means that you have to provide initializers, like this:

```
INIT u′′ = ...
INIT u′ = ...
INIT u = ...
```

Difference Equations

Berkeley Madonna also supports difference equations. Each difference equation consists of two equations: an initializer equation and a “next” equation. The initializer equation determines the initial value of the difference equation. The next equation determines the value the difference equation will take at the next time step. The syntax of the initializer equation is the same as used by reservoirs (see above). There are two forms of the next equation:

```
v(t + dt) = …
NEXT v = …
```

When Berkeley Madonna computes the next value of a difference equation, it uses the **current** values of any variables that appear in the expression on the right-hand side of the next equation.

Discrete Equations

Berkeley Madonna version 8.0 introduces support for three kinds of discrete objects: conveyors, ovens, and queues. These objects accumulate inputs, hold them for a period of time, and release them. The manner in which this is done differs for each kind of object.

These objects are similar to the conveyor, oven, and queue stocks in STELLA. However, since they use a different syntax, you must modify STELLA-generated equations that use these types of stocks. See Simulating STELLA Conveyors, Ovens, and Queues on page 44 for details.

Note that discrete objects require that you use a fixed-stepsize integration method. Furthermore, since the behavior of these objects is discrete rather than continuous, no additional accuracy is gained by using Runge-Kutta integration methods. Therefore, models which employ these objects should use Euler's method.

Conveyors

Conveyors work like a conveyor belt. Each time the model is stepped, an input is placed on the conveyor along with a transit time. The transit time determines how long this input remains in the conveyor. When its transit time has expired, the input is released from the conveyor.

Conveyors are defined using the following syntax:

```
name = CONVEYOR(input, tt, cap)
```

where *name* is the name of the conveyor, *input* is the input expression, *tt* is the transit time, and *cap* is the capacity. The name can be followed by array dimensions in square brackets to create an array of conveyors. The capacity argument is optional; if omitted, the conveyor has unlimited capacity.

The name of the conveyor is a variable which can be used in other equations or plotted. Its value is the sum of all inputs in the conveyor multiplied by DT.

To determine when inputs have exited the conveyor, use the OUTFLOW built-in:

```
output = OUTFLOW(name)
```

Since transit times for each input can vary, it is possible that more than one input will exit the conveyor at the same time. When this happens, the OUTFLOW built-in reports the sum of all inputs exiting the conveyor at that time.

A conveyor with a finite capacity is said to be capacity-constrained, meaning that the sum of its contents cannot exceed the specified capacity. However, conveyors will accept whatever input you provide even when a capacity is specified. To make the capacity constraint functional, you must explicitly limit the input values to prevent the conveyor's capacity from being exceeded. Use the MAXINFLOW built-in function to determine the maximum input the conveyor can accept, as follows:

```
input = ...
limited_input = MIN(input, MAXINFLOW(name))
name = CONVEYOR(limited_input, tt, cap)
```

Here, the input is prevented from exceeding the value returned by the MAXINFLOW function for this conveyor. This ensures that the conveyor's capacity will not be exceeded.

For a simple example illustrating a conveyor in action, open the "How Do I Use Conveyors" model via the Help menu in the software.

Ovens

Ovens work a lot like real ovens. They accumulate inputs during the filling state, hold them during the cooking state, and release them during the emptying state. The amount of time the oven spends in the filling and cooking states is controlled by the fill time and cook time parameters.

Ovens are defined using the following syntax:

```
name = OVEN(input, ct, ft, cap)
```

where *name* is the name of the oven, *input* is the input expression, *ct* is the cook time, *ft* is the fill time, and *cap* is the capacity. The name can be followed by array dimensions in square brackets to create an array of ovens. As with conveyors, the oven's capacity argument is optional; if omitted, the oven has unlimited capacity.

The name of the oven is a variable which can be used in other equations or plotted. Its value is sum of all the inputs in the oven multiplied by DT.

An oven exists in one of four states at any given time: idle, filling, cooking, or emptying. Initially, an oven is in the idle state. While idle, the oven watches its input for nonzero values. The oven will remain in this state indefinitely as long as its input remains zero. When the oven detects a change in the input, it adds that value to its contents and switches to the filling state.

During the filling state, the oven continues adding inputs to its contents. The oven will stop filling once its fill time (sampled when the oven started filling) has expired or its capacity has been reached, whichever comes first. Once this happens, the oven enters the cooking state.

During the cooking state, the oven holds onto its current contents without accepting any additional inputs. It remains in this state until its cook time (sampled when the oven entered the cooking state) has expired. The oven then enters the emptying state.

The emptying state lasts for just one time step. During this step, it empties its contents. This can be observed using the OUTFLOW built-in function whose value will be the sum of all inputs placed in the oven during the filling state:

```
output = OUTFLOW(name)
```

An oven begins filling again during the emptying state if its input expression is nonzero. When this happens, the oven switches back to the filling state for the next time step. Otherwise, it returns to the idle state.

An oven’s state can be determined using the OSTATE built-in function:

```
state = OSTATE(name)
```

State values are 0 for idle, 1 for filling, 2 for cooking, and 3 for emptying.

Since an oven can’t always accept inputs, you should use the MAXINFLOW built-in function to limit its input expression. For example, say you have a reservoir of material named “R” which feeds into an oven “V” at the rate of 3 units per time step. You might set up the equations like this:

```
rate = 3
INIT R = ...
d/dt(R) = -rate
V = OVEN(rate, ct, ft, cap)
```

The problem here is that the reservoir will lose material during every time step, even when the oven is cooking. To prevent this, use the MAXINFLOW built-in function to define the amount of material flowing from the reservoir to the oven:

```
rate = 3
inflow = min(rate, MAXINFLOW(V))
INIT R = ...
d/dt(R) = -inflow
V = OVEN(inflow, ct, ft, cap)
```

During the cooking phase, the MAXINFLOW function evaluates to zero which stops the flow of material out of the reservoir. And while filling, it will prevent the oven’s capacity from being exceeded.

For a simple example illustrating oven behavior, open the “How Do I Use Ovens” model via the Help menu in the software.

Queues

Queues accept one or more nonzero inputs during each time step and release them in the order they were added. Unlike conveyors and ovens, the removal of items from a queue is controlled by the use of special built-in functions. This makes it possible for a queue to have multiple outputs. For example, a single queue can feed several conveyors.

Queues are defined using the following syntax:

```
name = QUEUE(input1, input2, ..., inputN)
```

where *name* is the name of the queue and *input1* through *inputN* are the inputs to the queue. The name can be followed by array dimensions in square brackets to create an array of queues.

During each time step, a queue’s inputs are evaluated in left-to-right order. If an input is nonzero, it is placed in the queue. Thus, up to N items can be added to a queue during a single time step, where N is the number of inputs.

The name of the queue is a variable which can be used in other equations or plotted. Its value is the sum of all items in the queue multiplied by DT.

By itself, a queue collects inputs but never releases them. To remove items from a queue, use the QPOP and QPOPS built-in functions:

```
output = QPOP(name, maxel, maxamt)
output = QPOPS(name, maxel, maxamt)
```

where *name* is the queue from which elements are removed, *maxel* is the maximum number of items to remove, and *maxamt* is the maximum amount of the items removed. Each time a QPOP or QPOPS built-in is evaluated, it removes as many items as it can from the named queue such that the number of items removed is less than or equal to the *maxel* argument and the sum of the items removed is less than or equal to the *maxamt* argument. Items are always removed in the order they were added.

The QPOP built-in removes only whole items from the queue. If the next item to be removed is greater than *maxamt* argument, no items will be removed and the function's value will be zero.

The QPOPS built-in can remove part of an item from the queue. So, if the next item to be removed is greater than the *maxamt* argument, it will subtract this amount from the item, leaving the remainder in the queue.

Items can be removed from a queue in the same time step that they were added. In other words, material can flow through the queue without spending any time in it. This is in sharp contrast with conveyors and ovens in which inputs are delayed by at least one time step before they exit.

When queues are used to feed conveyors and ovens, the MAXINFLOW built-in function should be used to define the maximum amount of material that can be removed from the queue. For example, a queue feeding an oven would be defined like this:

```
Q = QUEUE(input1, ...)
inflow = QPOP(Q, 1, MAXINFLOW(V))
V = OVEN(inflow, ct, ft, cap)
```

This guarantees that items will be removed from the queue only when the oven is able to accept them. For a more interesting example of the use of queues, open the "How Do I Use Queues" model via the Help menu in the software.

Root Finder Equations

Berkeley Madonna can find roots of one or more expressions automatically during model execution. This feature is useful if certain quantities are defined by zeroing a dependent expression. For example, say you have a quantity q in your model whose value is determined by solving the following equation:[13]

$$aq^2 + bq + c = 0$$

To determine the quantity of q such that this equation is satisfied, define q using a root finder equation:

[13] This is a contrived example: you could solve for q using the standard formula for roots of quadratic equations.

```
ROOTI q = a*q*q + b*q + c
GUESS q = 1.0
LIMIT q >= -10
LIMIT q <= +10
```

The ROOTI equation tells Berkeley Madonna to compute the value of q so that the expression $aq^2 + bq + c$ evaluates to zero. Berkeley Madonna uses the GUESS equation to determine the initial value of q when it starts its search for a root. The LIMIT equations specify the range of values q may take during the search. These initial guess and limit equations are required any time you define a root finder equation.

Initial and Step Root Finders

The use of the ROOTI keyword indicates that this is an "initial root finder". Initial root finders perform the root-finding operation only once during model initialization. In the above example, even if a, b, or c were to change during subsequent time steps, the value of q would not be recomputed; it would remain constant through the model's execution.

If you want the root to be recomputed during each time step, use the ROOTS keywords instead of ROOTI. This defines a "step root finder" which recalculates the root for each time step.

You might think that finding a root during each step of your model would take a long time. However, after the root has been found during initialization, subsequent roots are generally found quickly since the quantities affecting the root (a, b, and c in this example) generally don't change much. Berkeley Madonna uses the previous root as the initial guess when finding the root the next time and usually doesn't have to look too far to find it.

The Root Finder Algorithm

Berkeley Madonna finds roots using a globally-convergent variation of the classic Newton-Raphson method. Put simply, it starts with your initial guess and uses Newton's method to estimate the root's position. If the function is smooth and the root isn't too far away, this algorithm converges very quickly. If it doesn't converge, Berkeley Madonna discards the guess and randomly chooses a new initial guess within the specified limits. While this is somewhat simplistic, it will usually hit upon an initial guess that converges to a root.

How does Berkeley Madonna decide that it has found a root? If the expression on the right-hand side of the root finder equation evaluates to exactly zero, then by definition a root has been found. However, in practice this rarely happens due to the limited precision of floating-point arithmetic on computers. Therefore, Berkeley Madonna also considers a root to have been found when the relative change applied to the variable (as determined by Newton's method) is less than the tolerance specified by the ROOTTOL parameter. In other words, Berkeley Madonna stops when it appears to be close to a root. If it isn't getting close enough for you, try reducing the value of ROOTTOL.

If Berkeley Madonna cannot find a root after a certain number of randomly-chosen initial guesses (currently 1000), it gives up and sets the variable to NAN. This causes a floating-point exception to be reported during model execution. In this example, if no root was found, q would be set to NAN which you could observe by examining its value in a table.

Multidimensional Root Finders

One advantage of using the Newton-Raphson algorithm to find roots is that it scales easily to multiple dimensions. For example, you can set up a series of root finder equations so that the values of x, y, and z are calculated by finding the roots of three separate equations. For example:[14]

```
ROOTI x = a*x*y + b*x + c
ROOTI y = d*x*z + e*y + f
ROOTI z = g*y*y + h*z + i
```

By examining the interdependencies of these equations, Berkeley Madonna determines that the values of x, y, and z must be determined by simultaneously solving all three equations.

Having three root finder equations in your model does not necessarily mean that you have a three-dimensional root-finding problem. For example:

```
ROOTI x = a*x*y + b*x + c
ROOTI y = d*x*x + e*y + f
ROOTI z = g*y*y + h*z + i
```

Here, notice that the roots of the first two equations can be found by varying only x and y. Once these roots have been found (a two-dimensional problem), Berkeley Madonna computes the value of z using the third equation without having to vary x or y.

Limit Equations

Limit equations are used to define upper and/or lower bounds for any variable in your model. Whenever a variable with limit(s) is assigned, the value is forced to be within the specified limits. The syntax is:

```
LIMIT var >= lower-bound
LIMIT var <= upper-bound
```

"var" must be the name of a variable defined elsewhere in your model. "lower-bound" and "upper-bound" may be any valid Berkeley Madonna expression. On Macintosh, you can use ≥ and ≤ instead of >= and <=, respectively.

Other Equation Types

There are three more types of "equations". Unlike other equations, however, they don't directly affect the values of variables in your model, so we call them "statements" instead.

[14] The initial GUESS and LIMIT equations have been omitted for clarity, but they must be included.

METHOD Statement

The "method" statement defines the default integration method that appears in the parameter window when your model is first compiled.[15] The syntax is:

METHOD name

"name" must be Euler, RK2, RK4, Auto, or Stiff.

DISPLAY Statement

The "display" statement allows you to control which symbols in your model are accessible from various windows and dialogs. The syntax is:

DISPLAY name1, name2, ...

"name1", "name2", ... are names of symbols in your model. You can use more than one display statement if you want. If your model uses one or more display statements, any symbols whose names don't appear in one of these statements will not be available in any windows or dialogs. If you don't use any display statements, all your symbols will be available.

RENAME Statement

The "rename" statement enables you to rename the built-in symbols listed in the next section. To rename a built-in symbol, use the following syntax:

RENAME old-name = new-name

For example, to change the name of TIME to X, you would write:

RENAME TIME = X

Note that the new name must not already be in use. For example, you cannot rename TIME to STOPTIME (unless STOPTIME itself has already been renamed to something else).

Built-in Symbols

The following symbols are implicitly defined in all models. You can refer to them in your equations just like any other variables in your model. Their values can be explicitly specified in your equations if the default shown below is not appropriate. Note that the definition of TIME cannot be changed.

Symbol	Default Definition
STARTTIME	STARTTIME = 0
STOPTIME	STOPTIME = 10
DT	DT = 0.02

[15] Berkeley Madonna remembers the current integration method whenever you save your model. So, the METHOD statement only has an effect for models that have never been compiled and saved.

Symbol	Default Definition
DTMIN	DTMIN = 1e-6
DTMAX	DTMAX = 1.0
TOLERANCE	TOLERANCE = 0.01
DTOUT	DTOUT = 0
ROOTTOL	ROOTTOL = 0.001
TIME	INIT TIME = STARTTIME d/dt(TIME) = 1.0

DT is used only by fixed-stepsize integration methods. DTMIN, DTMAX and TOLERANCE are only used by variable-stepsize integration methods. Symbols that are not applicable to the currently-selected integration method are not shown in the parameter window. See Integration Methods on page 45 for more information.

DTOUT controls the interval at which results are stored in memory. See Using DTOUT on page 47.

ROOTTOL appears only when your model uses one or more built-in root finder equations. See Root Finder Equations on page 28.

Operators

The following table lists the arithmetic, logical, and relational operators supported by Berkeley Madonna. They are listed in order of decreasing precedence. The relational operators (=, ≠, etc.) generate a value of 1 if the relation is true or 0 if the relation is false. The logical operators (AND, OR, NOT) interpret zero as false and all other values, even those very close to zero, as true. The conditional operator (IF-THEN-ELSE) returns the value of the "then" expression if the "if" expression is nonzero; otherwise, it returns the value of the "else" expression.

Operator	Name	Example
+ – NOT	unary plus (no effect) negate logical inverse	+a –b NOT c
^, **	raise to power	a ^ b, a ** b
* /	multiply divide	a * b a / b
+ –	add subtract	a + b a – b
= ≠, <> < > ≤, <= ≥, >=	equal to not equal to less than greater than less than or equal to greater than or equal to	a = b a ≠ b, a <> b a < b a > b a ≤ b, a <= b a ≥ b, a >= b
AND	logical and	a AND b
OR	logical or	a OR b
IF-THEN-ELSE	conditional	IF a THEN b ELSE c
INF, ∞	infinity	z = a / INF

Built-in Functions

Berkeley Madonna supports the following built-in functions:

Function	Meaning
ABS(x)	Absolute value of x
ARCCOS(x)	Inverse cosine of x
ARCCOSH(x)	Inverse hyperbolic cosine of x
ARCSIN(x)	Inverse sine of x
ARCSINH(x)	Inverse hyperbolic sine of x
ARCTAN(x)	Inverse tangent of x
ARCTANH(x)	Inverse hyperbolic tangent of x
ARRAYSUM(x[*])	Sum of elements in array x
ARRAYMEAN(x[*])	Average of elements in array x
ARRAYSTDDEV(x[*])	Standard deviation of elements in array x

BINOMIAL(p, n)
BINOMIAL(p, n, s)
	Binomial random number generated from n trials of probability p, optional seed s

COS(x)	Cosine of x
COSH(x)	Hyperbolic cosine of x

DELAY(x, d)
DELAY(x, d, i)
	Value of x delayed by time d, optional initial value i

ERF(x)	Error function of x
ERFC(x)	Complimentary error function of x
EXP(x)	e (2.71828...) raised to x
GAMMALN(x)	Natural log of the gamma function of x
GRAPH(v) (x1,y1) (x2, y2) ...	Graphical function of v, with one or more (x,y) data points
INIT(x)	Value of expression x at intialization.
INT(x)	Integer part of x
LOG10(x)	Log base 10 of x

Function	Meaning
LOGN(x)	Natural log of x
MAX(x1, x2, ...)	Maximum of x1, x2, ...
MAXINFLOW(s)	Maximum allowable inflow into conveyor or oven s
MEAN(x1, x2, ...)	Average of x1, x2, ...
MIN(x1, x2, ...)	Minimum of x1, x2, ...
MOD(x, y)	Floating-point remainder of x / y. This is equal to: x - y * INT(x / y)
NETFLOW(r)	Flow into reservoir r from previous time step

NORMAL(μ, v)
NORMAL(μ, v, s)
	Normally-distributed random number with mean μ and variance v, optional seed s

OSTATE(s)	State of oven s: 0=idle, 1=filling, 2=cooking, 3=emptying
OUTFLOW(s)	Outflow from conveyor or oven s
PI	Value of π (3.14159...)

POISSON(μ)
POISSON(μ, s)
	Random integer drawn from a Poisson distribution with mean μ, optional seed s

PULSE(v, f, r)	Pulse with volume v, first pulse at time f, repeat interval r
QPOP(q, n, m)	Removes up to n items from queue q totaling m or less
QPOPS(q, n, m)	Like QPOP, but can remove part of an item to satisfy m

RANDOM(l, h)
RANDOM(l, h, s)
	Uniformly-distributed random number between l and h, optional seed s

ROUND(x)	x rounded to the nearest integer
SIN(x)	Sine of x
SINH(x)	Hyperbolic sine of x
SQRT(x)	Square root of x
SQUAREPULSE(t, d)	Square pulse of height 1 starting at time t and lasting for duration d
STEP(h, t)	Step of height h at time t
STEPSIZE	Change in TIME from previous step to current step

Function	Meaning
SUM(x1, x2, ...)	Sum of x1, x2, ...
SWITCH(x, y)	Equivalent to IF x > y THEN 1 ELSE 0
TAN(x)	Tangent of x
TANH(x)	Hyperbolic tangent of x

Most of these functions are self-explanatory. A few, however, deserve some explanation.

ARRAYSUM, ARRAYMEAN, ARRAYSTDDEV

These function require a single array argument. To pass a array "v" to one of these functions, use the syntax "v[*]". For example:

```
v_sum = ARRAYSUM(v[*])
v_avg = ARRAYMEAN(v[*])
v_std = ARRAYSTDDEV(v[*])
```

For more information on arrays, see Array Equations on page 37.

BINOMIAL, NORMAL, POISSON, RANDOM

These functions generate various distributions of random numbers using a lagged Fibonacci generator with a very long period (approximately 10^84). This generator is based on the FIBMULT program. Each function comes in short and long forms. The short form uses a different seed each time you run your model. The long form uses a fixed seed value that you supply. When using a fixed seed, you'll get a repeatable sequence of random numbers each time you run your model (as long as you don't change the seed).

DELAY

This function returns its input value delayed by the specified delay time. The delay time expression (2nd argument) is evaluated once during model initialization so the amount of delay remains fixed for the entire run. The function returns a special "initialization" value until TIME has reached the specified delay time. In the two-argument form of the function, the initialization value is the initial value of the input expression. In the three-argument form, the initialization value is specified by the third expression which, of course, is evaluated only once.

The DELAY function works properly for both fixed- and variable-stepsize integration methods. However, if an expression using DELAY is used as a reservoir's flow equation, the net flow contributed by DELAY will not be affected by the integration method. In other words, when integrating expressions using DELAY, the accuracy of those reservoirs will be that of Euler's method no matter what integration method is actually used. The reason is that the internal queue that DELAY uses is not updated during the trial steps used by these higher-order methods.

NETFLOW

This function takes a single reservoir as an argument and returns the amount the reservoir changed from the previous time step to the current step. It returns zero during the initialization step. It is especially useful in conjunction with the Custom DT Method (described on page 47) for controlling the stepsize in terms of reservoir changes. Note that you can't simply multiply a reservoir's flow expression by DT to compute the net change unless you're using Euler's method.

STEPSIZE

This function returns the time difference between the previous step and current step (zero during initialization). This is useful for monitoring the change in stepsize when using variable-stepsize methods: assign the value of STEPSIZE to a variable and then add this variable to a graph window.

MAXINFLOW, OUTFLOW, OSTATE, QPOP, QPOPS

These functions are used in conjunction with conveyors, ovens, and queues. See Discrete Equations on page 25 for details.

Array Equations

Berkeley Madonna supports array equations in addition to scalar equations. This feature is useful if you have many scalar equations with similar right-hand sides. Arrays can have one, two, or three dimensions. To define an array equation, the variable name on the left-hand side includes a range of subscripts enclosed in square brackets like this:

```
v[1..10] = SIN(2*pi*f)
```

In this example, v is defined as a vector (one-dimensional array) of 10 elements with subscripts 1 through 10. Note that you don't need to explicitly define the number of elements in the array; Berkeley Madonna figures out the size based on the subscript range on the left-hand side.

Arrays with two or three dimensions are defined with two or three subscript ranges. For example, the following defines a two-dimensional array of 60 elements (5 rows by 12 columns):

```
a[1..5,1..12] = 10.5
```

And the following defines a three-dimensional array with 24 elements:

```
a[1..2, 0..2, 2..5] = -77
```

This last example is a bit tricky because index for each dimension starts at a different number. The array's dimensions are 2 x 3 x 4 and the elements are a[1,0,2], a[1,0,3], a[1,0,4], a[1,0,5], a[1,1,2], ..., a[1,1,5], ..., a[2,0,2], ..., a[2,2,5].

You can define a single array using more than one equation:

```
k[0..3] = a * COS(x)
k[4..9] = b * SIN(x)
k[10] = 0.5
```

In this example, k is an 11-element vector with subscripts 0 through 10. When multiple equations are used to define an array, they are executed in the order they appear in the equation file. Within a single equation, the elements are assigned from the first subscript to the last. In the previous example, the order of assignment is k[0], k[1], k[2], ..., k[9], k[10]. This execution order can very important as we shall soon see.

Berkeley Madonna defines special indexing variables named **i**, **j**, and **k** which are valid only in the context of an array equation. They refer to the subscript of the element being assigned on the left-hand side. Most often, these variables are used to refer to array elements. For example:

```
a[1..10] = ...
b[1..10] = ...
total[1..10] = a[i] + b[i]
```

In two- or three-dimensional array equations, there are two or three index variables. Each index variable corresponds to one dimension: **i** to the first dimension, **j** to the second, and **k** to the third. For example, the in following three-dimensional array equation, **i** goes from 1 to 3, **j** goes from 1 to 4, and **k** remains unchanged at 5:

```
s[1..3,1..4,5] = f[i,j] * g[j,k]
```

The rightmost index variable (**j** in a two-dimensional equation, **k** in a three dimensional equation) changes the fastest. For example, in the following equation:

```
a[1..3,1..3] = ...
```

the elements are assigned in the following order:

```
a[1,1] = ...
a[1,2] = ...
a[1,3] = ...
a[2,1] = ...
a[2,2] = ...
a[2,3] = ...
a[3,1] = ...
a[3,2] = ...
a[3,3] = ...
```

Berkeley Madonna allows you to refer to elements of the array being assigned in the right-hand side of its equation. This permits you to write equations like this:

```
factorial[1] = 1
factorial[2..30] = factorial[i - 1] * i
```

In this example, the order of evaluation is extremely important. If the two equations were reversed, factorial[2] would be undefined because the value of factorial[1] has not yet been computed. Another situation to avoid is illustrated here:

```
a[10] = 1000
a[1..9] = a[i+1] * 0.97
```

This example is invalid because the computation of a[1] depends on the value of a[2] which hasn't been computed yet. The correct way to write these equations is:

```
a[10] = 1000
a[9..1] = a[i+1] * 0.97
```

Finally, you must watch out for gaps in your arrays. Consider the following:

```
b[0] = 0
b[1..4] = b[i-1] + sqrt(i)
b[10] = 100
b[9..6] = b[i+1] * b[10-i]
```

Everything looks fine here until you realize that b's subscripts go from 0 to 10, but b[5] is not defined anywhere. Berkeley Madonna does not check for such errors, so use care.

Arrays can be used with differential, difference, and discrete equations. Just remember to include the subscript range immediately following the variable name on the left-hand side.[16] For example:

```
y[1](starttime) = 1
y[2..3](starttime) = 1.1 * y[i - 1]
d/dt(y[1..3]) = -0.1 * y[i]

INIT a[1..5] = i
NEXT a[1..5] = a[i] + i

INIT x[1..n] = 0
x[1..n](t) = x(t - dt) + (1 / sqrt(y[i])) * dt

c[1..n] = CONVEYOR(...)
```

Note that you can't use multiple equations to define an array of discrete objects. This restriction is necessary to prevent the creation of schizophrenic arrays such as this:

```
s[1..3] = CONVEYOR(...)
s[4..6] = QUEUE(...)
s[7..9] = OVEN(...)
```

Limits can be applied to arrays, but the limit applies equally to all elements. You cannot apply a limit to only certain elements. For example, to limit each element of vector y in the previous example to values greater than or equal to 0.5, add the following equation:

```
LIMIT y >= 0.5
```

You can use variables in subscript ranges. This capability makes it easy to change the size of arrays without recompiling your model. For example:

```
factorial[1] = 1
factorial[2..n] = factorial[i-1] * i
n = 10
```

Since n is a parameter, you can change its value from the parameter window. Berkeley Madonna dynamically determines the size of the array before each run and allocates the necessary memory.

[16] Berkeley Madonna also permits the subscript range to appear immediately following the variable name on the right-hand side of the integrator equation in STELLA syntax. This exception enables Berkeley Madonna to compile array equations generated by STELLA.

If a subscript range uses an expression whose value changes over time, the size of the array does not change. The expression's value during the initialization step determines the size of the array and this size remains fixed during the run.

Berkeley Madonna does not support STELLA "symbolic indexing" array syntax.

Dataset Functions

You can use imported datasets as piecewise-linear functions in your model's equations. This feature makes it easy to use data from external sources to control your model's behavior. Before you can use a dataset function in your equations, you need to import a dataset. Importing Datasets on page 15 describes how to do this.

Once you've imported a dataset, you can use it just like a built-in function taking one argument (for vector datasets) or two arguments (for matrix datasets). For example, assume you've imported a vector dataset named "#temperature" which maps distances into temperature values. (This could be from taking temperature measurements at various points along a metal bar.) In your model, you could compute the temperature "t" at a point "p" using the following expression:

```
t = #temperature(p)
```

Note that since dataset names always begin with a pound-sign character (#), they can be easily distinguished from regular built-in functions. To evaluate this expression, Berkeley Madonna looks up the value of "p" in the dataset's array of input values (X). It finds the two X values which bracket the input value, then it uses linear interpolation to compute the output value using the corresponding Y values. For values outside the dataset's range of X values, it uses the Y value corresponding to the closest X value (always the first or last point in the dataset).

Matrix datasets are used in a similar fashion except that they take two arguments. For example, assume we have a matrix dataset representing temperatures measured at various points on a metal surface. You can determine the temperature at a point with Cartesian coordinates "x" and "y" like this:

```
t = #temperature(x, y)
```

As with vector datasets, Berkeley Madonna performs linear interpolation to find the output value for the corresponding inputs. This time, the interpolation is performed in two steps since both the X and Y inputs may not correspond exactly to the input values in the dataset.

Plotting Matrix Datasets

As discussed in Matrix Datasets on page 16, matrix datasets cannot be directly plotted in a graph window as vector datasets can. However, you can still get an idea of what a matrix dataset looks like by using an array equation to sample its outputs for various inputs and then plotting the resulting array. To do this, you need to know the range of the dataset's input domains (the X and Y inputs). Then, construct an array equation that takes output values at various inputs over the domain. One of the input domains is covered using the TIME variable and the other is covered by the array index variable **i**.

For example, assume you have a matrix dataset named "#temperature" that maps points in a plane to temperature values. Assume the X coordinates go from 0.0 to 5.0 and the Y coordinates from 0.0 to 10.0. Sample the elements of this dataset using the following equations:

```
STARTTIME = 0
STOPTIME = 10.0
DT = 1.0
T[0..5] = #temperature(i, TIME)
```

Run the model and plot the value of array T. You'll get six curves representing temperatures in the Y direction for six different X values. Of course, you could instead use the following equations:

```
STARTTIME = 0
STOPTIME = 5.0
DT = 1.0
T[0..10] = #temperature(TIME, i)
```

This would result in eleven curves representing temperatures in the X direction for eleven distinct Y values.

Regional Settings [Windows only]

The Windows version allows you to change the characters used for the decimal symbol and list separator to match your language's conventions. By default, Berkeley Madonna uses the period (`.`) as the decimal symbol and the comma (`,`) as the list separator. However, if you check the **Use Regional Settings** option in the **General** page of the **Preferences** dialog, the decimal symbol and list separator are instead taken from the Numbers page of Windows' Regional Settings control panel.

The decimal separator is used to separate the whole and fraction parts of floating-point numbers. This applies to your equations and to numbers displayed in Berkeley Madonna's windows. The list separator is used to separate arguments to built-in functions and data points for the GRAPH built-in function.

For example, take the following equation:

```
y = 0.5 * SUM(a, b)
```

If you check the **Use Regional Settings** option and your system's decimal symbol is set to the comma and list separator to the semicolon, this equation would have to be changed to:

```
y = 0,5 * SUM(a; b)
```

These changes would also have to be made to expressions entered into Berkeley Madonna's parameter window and dialogs.

There are some important limitations you should be aware of if you enable the **Use Regional Settings** feature. First, the decimal symbol and list separator should each consist of only a single character (extra characters are ignored). Second, these two characters should be different. Finally, you should not use characters for decimal symbols or list separators that are already used by Berkeley Madonna for other purposes; these characters include letters A-Z (upper and lower case), digits 0-9, and the following special characters: _ `+ - * / ^ < = > ( ) [ ] { }`. You can use the semicolon (`;`) as the decimal symbol or list separator. However, if you use the semicolon for this purpose, you won't be able to use it to write single-line comments.

Running STELLA Models

If you have a STELLA model, you can try running it in Berkeley Madonna. Follow these steps:

1. Save your STELLA model as an equation (text) file.
2. Drag the equation file to the Berkeley Madonna icon, or open the equation file from within Berkeley Madonna. The equations are displayed in an untitled equation window.
3. Choose **Parameter Window** from the **Parameters** menu. Berkeley Madonna will compile the equations and display the parameter window if the compile is successful.
4. If your model uses Euler's Method or Runge-Kutta 2 instead of Runge-Kutta 4, change the integration method to match.
5. If your model's time specs (STARTTIME, STOPTIME, or DT) are not what they were in STELLA, change them to the correct values.
6. Click the **Run** button. Berkeley Madonna will run your model and display the results in a new graph window.

If an error occurs when you attempt to open the parameter window (step 3) or if the results of the run were not what you expected, the problem is most likely due to an incompatibility between STELLA and Berkeley Madonna.

Although Berkeley Madonna was originally designed to be compatible with STELLA equation files, there are some features in STELLA that Berkeley Madonna does not support. Also, there are some differences in the way certain features are handled by the two programs.

Berkeley Madonna can't handle documentation embedded in STELLA equation files. To avoid this problem, uncheck the **Show Documentation** check box in STELLA's **Equation Prefs** dialog before saving the equations.

Berkeley Madonna does not support all STELLA built-in functions. See Built-in Functions on page 34 for a list of built-ins that Berkeley Madonna does support.

There are four kinds of stocks in STELLA: reservoirs, conveyors, queues, and ovens. Berkeley Madonna only supports reservoirs directly. However, you may be able to simulate the effect of conveyors, ovens, and queues by using Berkeley Madonna's discrete equation support (see Simulating STELLA Conveyors, Ovens, and Queues on page 44).

STELLA equation files don't indicate if reservoirs are non-negative or whether flows are unidirectional or bi-directional. Berkeley Madonna assumes that reservoirs are **not** non-negative (i.e., they can have any real value) and flows are bi-directional. You can change this behavior by using the LIMIT statement. See Non-negative Reservoirs and Unidirectional Flows on page 43.

STELLA equation files don't indicate the integration method used. Berkeley Madonna uses the Runge-Kutta 4 method by default. However, you can override this by using the METHOD statement.

STELLA equation files don't indicate the values of STARTTIME, STOPTIME, and DT in your model. Berkeley Madonna assumes their values are 0, 10, and 0.02, respectively. However, you can specify different values in your equation file.

Tips and Techniques

Creating Periodic Functions

To make any function repeat periodically, use the MOD built-in. If F is a function that you want to repeat over the time interval I, you would write:

```
y = F(MOD(TIME, I))
```

For example, to make a square wave with a period of one time unit and an amplitude of 1, you could write:

```
y = IF MOD(TIME, 1) >= 0.5 THEN 1 ELSE 0
```

or, more succinctly:

```
y = MOD(TIME, 1) >= 0.5
```

Creating Piecewise-Linear Functions

Multiple IF-THEN-ELSE expressions can be used to write piecewise linear differential equations. For example:

```
d/dt(x) = IF x < 3 THEN L1 ELSE IF x <= 4 THEN L2 ELSE L3
```

where L1, L2, and L3 are line segments. Or you can use the GRAPH() built-in which generates the line segments for you:

```
d/dt(x) = GRAPH(x) (0,0) (3,4) (4,6.5) (10,1)
```

For large numbers of line segments, you may find it easier to list the points in a text file and use that file to define a dataset function. See Dataset Functions on page 40 for details.

Non-negative Reservoirs and Unidirectional Flows

By default, reservoirs in STELLA models are limited to non-negative values and flows are limited to one direction. These defaults can be overridden by clearing the non-negative checkbox and choosing Biflow instead of Uniflow in the appropriate STELLA dialogs.

Unfortunately, STELLA doesn’t indicate the style of reservoirs and flows in the equation file. Berkeley Madonna assumes that reservoirs and flows can take any real value.

If your model depends on reservoirs and/or flows being limited to non-negative values, you’ll need to add appropriate LIMIT statements to ensure these conditions are met. For example, the following equation taken from STELLA defines a reservoir named “Resv” which depends on an inflow named “In” and an outflow named “Out”:

```
d/dt(Resv) = In - Out
```

By default, Berkeley Madonna assumes that “Resv”, “In”, and “Out” can be positive or negative. To make “Resv” non-negative and the two flows unidirectional, add the following statements to the equation file:

```
LIMIT Resv >= 0
LIMIT In >= 0
LIMIT Out >= 0
```

If your model depends on STELLA's "outflow prioritization" mechanism, you'll have to rewrite your equations in Berkeley Madonna to duplicate this behavior.

Simulating STELLA Conveyors, Ovens, and Queues

Berkeley Madonna cannot directly interpret STELLA equation files containing conveyors, ovens, or queues. However, you can modify the STELLA equations to use the conveyor, oven, and queue facilities provided by Berkeley Madonna.

The first step is to change each discrete stock into a conveyor, oven, or queue equation as shown in Discrete Equations on page 25. When this is done, an equation for the stock's inflow must also be written. For conveyors and ovens, this should consist of an expression using the MAXINFLOW built-in function to limit the flow into the stock. If the flow to a conveyor or oven S is coming from a reservoir R and the desired flow is D, the actual flow F should be computed as follows:

```
F = MIN(D, MAXINFLOW(S))
```

Then, use F in lieu of D for the reservoir's outflow and for the conveyor/oven's inflow:

```
R(t) = R(t-dt) + (... - F) * dt
S = CONVEYOR/OVEN(F, ...)
```

When conveyor or oven inflows are coming from a queue, the MAXINFLOW built-in is used to limit the amount of material removed from the queue by the QPOP/QPOPS built-in functions:

```
Q = QUEUE(...)
F = QPOP(Q, maxel, MAXINFLOW(S))
S = CONVEYOR/OVEN(F, ...)
```

When inflows from a queue are taken "one at a time", set the QPOP/QPOPS *maxel* argument to one. When taking "as much as possible", set the *maxel* argument to infinity (INF). Use the QPOPS built-in if you are removing items in "split batches" or the QPOP built-in if removing only whole items.

The outflows from conveyors and ovens can be computed using the OUTFLOW built-in function:

```
out = OUTFLOW(S)
```

When these outflows flow into another stock, simply use the outflow expression as the stock's inflow expression.

Some of STELLA's stock features aren't yet supported by Berkeley Madonna. These include arresting the operation of stocks, conveyor leakage, and initialization to nonzero values.

Integration Methods

Fixed-stepsize Methods

Berkeley Madonna provides three fixed-stepsize integration methods: Euler's method, Runge-Kutta 2, and Runge-Kutta 4. These methods step the model in increments of DT. The number of steps taken (not including the initialization step) is equal to:

$$\frac{STOPTIME - STARTTIME}{DT}$$

Variable-stepsize Methods

Berkeley Madonna also provides two variable-stepsize integration methods. These algorithms choose the largest stepsize they can consistent with the tolerance and minimum/maximum stepsize you specify. The DTMIN parameter specifies the minimum allowable stepsize and the size of the first step taken. The DTMAX parameter specifies the maximum allowable stepsize (it also controls the width of impulses generated by the PULSE built-in). The variable stepsize algorithms automatically adjust the stepsize within these limits to minimize the number of steps taken while keeping the estimated error below the threshold specified by the TOLERANCE parameter.

When using a variable-stepsize method, you may want to monitor how the stepsize is changing as the run progresses. Your first inclination might be to simply plot DT. However, you won't find DT among the list of available variables to plot. That's because DT is a parameter whose value by definition is constant throughout the run. When using variable stepsize methods, the value of DT is not used and therefore it is not shown in the parameter window. The actual stepsize is determined using the STEPSIZE built-in function. For example:

```
h = STEPSIZE
```

By assigning the value of the STEPSIZE built-in to a variable, you can monitor it during the run by simply plotting the variable ("h" in this example).

Here is the process used by the variable-stepsize algorithms to advance the model by one time step:

1. Compute flows for each reservoir in the model based on the current values of all variables.
2. Estimate absolute error in each flow computed in step 1.
3. Convert absolute errors computed in step 2 to relative errors by dividing each error by the magnitude of the reservoir's current value. This scaling is not performed if the magnitude of the reservoir is less than 1.
4. If the maximum relative error is greater than TOLERANCE and the stepsize is greater than DTMIN, reduce the stepsize and go back to step 1.
5. Advance the model to the next time step using the current stepsize.
6. If the maximum relative error is less than TOLERANCE and the stepsize is less than DTMAX, increase the stepsize for the next time step. Otherwise, leave the stepsize unchanged.

7. Return to step 1 to do the next time step.

The computation of the estimated error for each reservoir can be stated formally:

> Let y be the value of the reservoir, Δy be the estimated increment (flow) to be added to the reservoir, and ε be the estimate of the error in Δy. The estimated error is:
>
> $$\left\{ \begin{matrix} \left|\dfrac{\varepsilon}{y}\right|, |y| > 1 \\ |\varepsilon|, |y| \le 1 \end{matrix} \right\}$$

The Auto-stepsize and Rosenbrock (stiff) methods differ in the way they compute flows, estimate absolute errors, and adjust the stepsize.

The Auto-stepsize method uses a fifth-order Runge-Kutta algorithm to compute flows. It estimates errors by comparing this flow with another flow computed using a fourth-order algorithm. The difference between these estimated flows is the estimated error.

The Rosenbrock (stiff) method uses a semi-implicit fourth-order Runge-Kutta algorithm to compute flows. It estimates error by comparing this flow with another flow computed using a third-order algorithm. Again, the difference between these estimated flows is the estimated error.

The Auto-stepsize method adjusts the stepsize by multiplying it by the following factor:

$$0.99 \times \sqrt[5]{\frac{tol}{\varepsilon_{max}}},$$

where tol is the TOLERANCE and ε_{max} is the maximum relative error.

This factor is limited to the range 0.1 - 10; thus, the stepsize cannot change by more than a factor of ten from one step to the next. The fifth root is used because, to a first approximation, the change in the error is proportional to the change in stepsize raised to the fifth power for a fourth-order method. The 0.99 is a "safety factor": it's much better to use a slightly smaller stepsize than theoretically possible; otherwise, the algorithm may end up overestimating the stepsize slightly and consequently discarding attempted steps (wasting time) because stepsize was just a little too large.

The Rosenbrock (stiff) method adjusts the stepsize in the same way except that a different factor is used:

$$0.99 \times \sqrt[4]{\frac{tol}{\varepsilon_{max}}}$$

This factor is limited to the range 0.5 - 2; thus, the stepsize cannot change by more than a factor of two from one step to the next. The fourth root is used because the change in error is roughly proportional to the change in stepsize raised to the fourth power for a third-order method.

For stiff systems of equations, the Auto-stepsize method grossly overestimates errors and thus uses much smaller stepsizes than is actually necessary. The Rosenbrock (stiff) method does a much better job for these types of systems.

On the other hand, for smooth, non-stiff systems, the Auto-stepsize method, being of higher order than the Rosenbrock (stiff) method, can take larger steps for a given TOLERANCE. Also, the Rosenbrock (stiff) method does a lot more math (matrix inversion, etc.) to estimate flows. For these systems, the Auto-stepsize algorithm is a better choice.

The Auto-stepsize algorithm is derived from the routine **rkqs()** published in *Numerical Recipes in C*. The Rosenbrock (stiff) method is derived from the routine **stiff()** in the same text.

Using DTOUT

Normally, Berkeley Madonna stores an output point for each time step. For small stepsizes, this can result in large quantities of output data. You can reduce the amount of data by increasing the stepsize, but often this is not possible because a small stepsize is needed to achieve sufficient accuracy. By using DTOUT, you can control how much output data are stored independently of the stepsize used to compute them.

The default value of DTOUT is zero, which causes Berkeley Madonna to store every computed step in memory. To reduce the number of steps stored, set DTOUT to some multiple of the stepsize. For example, if your model uses a fixed-stepsize integration method and a DT of 0.001 and you want to store only one output point for every 100 computed points, set DTOUT to 0.1 (0.001 * 100). The resulting output will appear to have a DT of 0.1, but the data are still computed with a DT of 0.001.

If you are using a fixed stepsize integration method and DTOUT is not an integer multiple of DT, Berkeley Madonna will round DTOUT to the nearest integer multiple of DT and use this rounded value in lieu of the value you specify. For example, if your DT is 0.004 and you set DTOUT to 0.05, Berkeley Madonna will round DTOUT to 0.048 and store 1 output step for every 12 computed steps.

Variable stepsize methods handle DTOUT differently. When each computed step is finished, Berkeley Madonna checks to see if the current time is at least DTOUT greater than the time of the previous output step. If it is, the current step will be stored. Otherwise, it is skipped. Note that Berkeley Madonna will always store the final step when TIME reaches STOPTIME, so the interval between the last two output steps may be less than DTOUT (or DTMIN for that matter).

It’s important to remember that DTOUT affects only the number of steps stored in memory. Berkeley Madonna still uses the specified stepsize to perform all internal calculations, so the accuracy of the solution is not affected as evidenced by the fact that the number of computed steps shown in the upper right-hand corner of the graph window doesn’t change.

Custom DT Method

Berkeley Madonna provides a custom DT method which allows your model to directly control the value of DT used for each step. To use this feature, define DT as a difference equation in the following format:

```
INIT DT = initial_dt
NEXT DT = IF ok_to_step THEN next_dt_value ELSE retry_dt_value
```

When you define DT this way, Berkeley Madonna runs your model using the following steps:

1. Initial values of all reservoirs and difference equations are computed as usual. The value of DT is set to initial_dt.

2. Flows are computed for each reservoir based on the current value of DT.
3. The ok_to_step expression is evaluated. If this expression is true (nonzero), Berkeley Madonna continues with step 4. Otherwise, DT is changed to retry_dt_value and Berkeley Madonna returns to step 2.
4. The model is advanced to the next time step using the current value of DT.
5. DT is changed to next_dt_value and Berkeley Madonna goes to step 2.

Note that DT is not limited to any minimum or maximum value; it is your responsibility to set DT to appropriate value in the next_dt_value and retry_dt_value expressions. DTMAX is used only to determine the width of the PULSE function.

Here is an example problem: Vm is a reservoir representing voltage. You want to step your model as fast as possible with the limitation that the change in the voltage cannot exceed ±10 millivolts per time step. Also, DT must be limited to 8 milliseconds maximum and 1 microsecond minimum. When the proposed change in Vm is too large, you'll try reducing DT by a factor of 2; when the change is within bounds, try increasing DT by 10%.

Here's what you need to put in your model to accomplish the above:

```
INIT DT = .001        {Start with 1 millisecond time steps}
NEXT DT = IF ABS(NETFLOW(Vm)) <= .010 THEN MIN(DT * 1.1, 0.008) ELSE MAX(DT
/ 2, 1e-6)
```

Note that if the minimum DT of 1e-6 produces too large a voltage change, the model will loop infinitely. This should be avoided by changing the ok_to_step expression to:

```
ABS(NETFLOW(Vm)) <= 0.010 OR DT = 1e-6
```

This way, the model is guaranteed to step when trying the minimum allowable DT.

One word of caution: you may be tempted to use intermediate variables to reduce the complexity of your ok_to_step and retry_dt_value expressions. For example:

```
step_ok = ABS(NETFLOW(Vm)) <= 0.010 OR DT = 1e-6
NEXT DT = IF step_ok THEN ... ELSE ...
```

This will not work! Why? Because the ok_to_step expression is evaluated before your model is advanced. So, the step_ok variable above will still have its value from the previous step which depends on the flow from the previous step. To avoid problems like this, you should make sure that your ok_to_step and retry_dt_value expressions do not reference variables that depend on DT or the NETFLOW() function. Instead, references to DT and NETFLOW() should appear directly in these expressions.

Another thing to note: when Berkeley Madonna retries a step because the ok_to_step expression was false (zero), the only thing that changes is DT. Nothing else in your model has changed because no step was taken. Thus, the only things that will be different when the ok_to_step expression is re-evaluated are DT and the flows going into your reservoirs (which depend on DT and are computed using the NETFLOW() function).

Models using the custom DT feature can employ any fixed-stepsize integration method (Euler's method, Runge-Kutta 2, and Runge-Kutta 4). Variable-stepsize algorithms are not supported. When using the custom DT feature, your model is implicitly running with a variable stepsize.

Other Features

Chemical Reactions

The chemical reactions feature enables you to enter equations using ordinary chemical reaction notation. Berkeley Madonna generates the corresponding kinetic equations based on the law of mass action for you. Chemical reactions are represented by special embedded objects in the equation window which makes them easy to identify. You can freely mix these objects with your own equations.

To work with chemical reactions in your model, choose **Chemical Reactions** from the **Modules** submenu in the **Model** menu. Berkeley Madonna displays the **Chemical Reactions** dialog. You use this dialog to add, modify, and delete chemical reactions in your model.

To add a chemical reaction to your model, enter the left- and right-hand sides of a reaction into the **Reactants** and **Products** fields, respectively. Each side consists of a list of substances (reactants or products) separated by plus signs (+). Each substance is prefixed with an optional integer multiplier. For example, the following reaction consumes one unit of substance A and two units of substance B, producing three units of substance C and one unit of substance D:

$$A + 2B \Leftrightarrow 3C + D$$

If desired, enter the forward and reverse rate constants into the **Kf** and **Kr** fields, respectively. If you leave these fields blank, they default to 1.0. Then click the **Add** button. The chemical reaction is added to the **Reactions** list. Note that each reaction is assigned a number so it can be uniquely identified. Also note that the forward and reverse reaction rates are not shown in the list. To see them, select the reaction and examine the **Kf** and **Kr** fields. You can continue adding reactions to the list by repeating the above process.

To modify an existing reaction, select it in the **Reactions** list, edit the **Reactants**, **Products**, **Kf**, and/or **Kr** fields and then click the **Modify** button. Don't forget to click **Modify** or the reaction will not be updated. To remove a reaction, simply select the reaction and click **Remove** (or double-click the reaction).

As you add reactions, you'll notice that Berkeley Madonna adds entries to the **Initial Concentrations** list. Here you can specify the initial value of any substance. To do so, select the substance and enter the value in the box below the list.

When you're done building the list of reactions and setting the initial concentrations for each substance, click the OK button. Berkeley Madonna adds the chemical reactions to your model's equations. To see the results, open the equation window by choosing **Equations** from the **Model** menu.

Each chemical reaction is represented in the equation window by an embedded chemical reaction object. These objects are easily identified by their gray shading. Within each object, Berkeley Madonna shows the chemical reaction in chemical notation followed by the reaction rate equation (RXN1 = ...), forward and reverse rate constants, initial concentrations, and the differential equations for each substance.

You can simplify the display of chemical reaction objects by hiding any of these parts except for the reaction itself by unchecking the appropriate **Show** boxes in the **Chemical Reactions** dialog. Note that these settings apply to all reactions in your model.

When you compile a model with chemical reactions, Berkeley Madonna generates kinetic equations for each reaction as shown in its object. Berkeley Madonna keeps track of which substances are produced and/or consumed by each reaction, keeping the generated equations consistent with each other. To see the actual equations generated, activate the equation window, choose **Save Equations As** from the **File** menu, and examine the resulting text file.

Chemical reaction objects can be manipulated just like other embedded objects such as pictures. They can be cut, copied, and pasted using the clipboard, moved or copied using drag and drop, and removed by selecting them and choosing **Clear** [Macintosh] or **Delete** [Windows] from the **Edit** menu.

To quickly modify a reaction, double-click it in the equation window. Berkeley Madonna will open the **Chemical Reactions** dialog and select the desired reaction for you. All you have to do is enter the changes, click **Modify**, then click OK. Note that the default buttons are set up so that you can just hit enter twice rather than clicking the buttons.

You don't have to keep the equation window open to modify chemical reactions. You can always choose **Chemical Reactions** from the **Modules** submenu in the **Model** menu to display the dialog.

Adding Other Equations

As mentioned earlier, you can freely mix your own equations with chemical reaction objects. This can cause problems if your equations conflict with those generated by the chemical reaction objects. For example, consider the following simple reaction:

$$A \Leftrightarrow B$$

Berkeley Madonna generates the following equations:

```
{ 1: A <--> B }
    RXN1 = K1f*A - K1r*B
    K1f = 1
    K1r = 1
    INIT A = 0
    INIT B = 0
    d/dt(A) = -RXN1
    d/dt(B) = +RXN1
```

Now insert the following equation before the chemical reaction object:

```
B = 2.2
```

Choose **Compile** from the **Model** or **Compute** menu. Berkeley Madonna highlights the chemical reaction object and displays an error message stating that "B is already defined as a converter". The problem is that B is defined twice, first by the line you inserted and then by the equations generated by the chemical reaction object. The chemical reaction object was highlighted because the definition of B within was rejected.[17]

If you enter the above equation after the reaction object, you get a similar error: this time your attempt to define B as a normal equation (converter) is illegal because it has already been defined as a reservoir by the chemical reaction object.

It is possible to override the equations generated from a chemical reaction object. This can be useful if you want to, for example, change the rate of reaction from mass action to some other law. To do this, you need to add a new definition for the rate of reaction variable after the chemical reaction object. It is imperative that you add your overriding definition after the reaction object; otherwise, the object's definition will override your definition which is probably not what you had in mind.

For example, adding the following line after the chemical reaction object shown above changes the rate of reaction to a law that applies on a strange alien world:

```
RXN1 = SQRT(K1f * LOGN(A) - K1r * LOGN(B))
```

Curve Fitter

Berkeley Madonna can automatically find the values of one or more parameters in your model that minimize the deviation between your model's output and a dataset. Before you use this feature, you should be able to run your model and be confident that it is working properly. You should already have determined integration parameters that give accurate results while minimizing execution time.

To perform a curve fit, choose **Curve Fit** from the **Parameters** menu. You will see a fairly involved dialog box. Here's what you do with it:

1. Specify which variable in your model you are trying to fit to the external data.
2. Specify the dataset you want to fit the variable to. If you haven't imported the dataset yet, you can do so by clicking the **Import Dataset** button. See Importing Datasets on page 15 for details.
3. Specify one or more parameters in your model which you want to solve for. To do this, choose a parameter from the available list on the left and click **Add**. If you change your mind, click **Remove**.
4. For each parameter you have chosen, you must provide two initial guesses. To set guesses for a parameter, select its name in the parameter list and edit the values displayed in the **Guess #1** and **Guess #2** boxes.
5. You must also provide a minimum and maximum value for each parameter. Both guesses must be within this range. The curve fitter will not set the parameter outside of this range.

[17] Unfortunately, Berkeley Madonna does not highlight the offending portion of the reaction object; instead, it highlights the entire reaction object. However, the cause of the problem can usually be deduced from the wording of the error message.

6. Specify the fractional tolerance you desire in the solution. For example, if you want the parameter(s) solved to three significant figures, set the tolerance to 0.001. The tolerance must be greater than zero.

7. Once you are satisfied with your setup, click OK to begin the curve fit. Berkeley Madonna will run your model repeatedly until it finds a solution. If you get tired of waiting, click the **Stop** button to abort the fit.

8. When the fit is complete, Berkeley Madonna leaves the parameters set to the values that give the best fit. It then runs your model and plots the fit variable and dataset in a graph window.

While the curve fit is in progress, Berkeley Madonna displays the RMS deviation between the data and best run so far. The deviation is root mean square of the differences between individual data points in the dataset and the corresponding points in the run.

When the curve fit operation complete, Berkeley Madonna immediately closes the **Running** dialog. If you want the dialog to remain open after the curve fit completes, check the **Pause After Curve Fit** option in the **General** page of the **Preferences** dialog. This enables you to observe the RMS deviation of the best fit. To close the dialog, click the **Done** button.

Multiple Fits

By default, the curve fitter fits a single variable in your model to a single imported dataset. However, it can simultaneously fit multiple variables to multiple datasets. To do this, check the **Multiple Fits** box in the **Curve Fit** dialog and add variable-dataset pairs to the list of fits using the **Add** button. Each variable-dataset pair has a relative weight which allows you to control the importance of each pair. The default weight is one, but it can be changed by selecting the pair and entering a new weight in the box below the list of fits. Berkeley Madonna multiplies the deviation of each pair by its weight, then sums these products to compute the overall deviation to minimize.

Note that when the **Multiple Fits** box is checked, the selections in the **Fit Variable** and **To Dataset** controls have no effect on the curve fit operation; they are used simply to define variable-dataset pairs to add to the **Multiple Fits** list.

Specifying Initial Guesses

There are subtleties involved in specifying initial guesses for each parameter. First, the two guesses must be different.[18] If one or more parameter's initial guesses do not meet this criterion, the curve fit will stop before the best solution is found.

---

[18] More specifically, the fractional difference between the two guesses should be much greater than the tolerance you specified, i.e.:

$$\frac{|g_1 - g_2|}{\frac{1}{2}\left(|g_1| + |g_2|\right)} >> tol ,$$

where g_1 is Guess #1, g_2 is Guess #2, and tol is the desired tolerance of the solution.

When you add a parameter to the list, Berkeley Madonna proposes default guesses. Guess #1 is set to 0.5 times the parameter's value. Guess #2 is set to 1.5 times the parameter's value. However, if the parameter's value is zero, the guesses are set to -1 and +1.

Floating-point Exceptions

Some models are very touchy when you vary their parameters. That is, they will not run at all! When this happens, you get a floating-point exception. As the curve fitter runs, it may change the value of a parameter in such a way that the model won't run. The curve fitter can handle this case by backing off, but when it first starts up it must be able to run your model with any combination of your initial guesses. If your model won't run with one of these combinations, the curve fitter won't be able to start. You'll know when this happens if you get an error while the curve fitter is in the initialization phase.

Optimizer

The optimizer feature automatically searches for parameter values that minimize an arbitrary expression. To perform an optimization, choose **Optimize** from the **Parameters** menu. The dialog is set up similar to the Curve Fitter (described on page 51) except that instead you specify an expression to minimize instead of a variable and dataset.

Th expression to minimize is evaluated once after each run when variables have reached their final values (TIME = STOPTIME). It should refer to at least one variable in your model. For example, if your model has a reservoir named "Q" representing charge and you want to minimize the final value of this charge, you would simply enter "Q" in the **Minimize Expression** box.

If you want to minimize something other than the final value of a variable, you'll need to define an auxiliary variable in your model that computes the desired quantity at the final time. For example, say you have a variable "temp" representing temperature and you want to adjust several parameters to get the <u>average</u> temperature over time equal to 20 degrees. To do this, you need to define a variable that computes the average temperature. This can be done using a couple of difference equations:

```
INIT sum_temp = 0
NEXT sum_temp = sum_temp + temp
INIT n = 0
NEXT n = n+1
```

At the final time, the variable sum_temp will contain the sum of temperature values for all steps and n will contain the number of steps. So, to get the average temperature to 20 degrees, you'd enter the following into the **Minimize Expression** box:

```
ABS(sum_temp/n - 20)
```

Note the use of the absolute value function which ensures that this expression takes on its minimum possible value (zero) when the average temperature is exactly 20 degrees.

Boundary Value Solver

Berkeley Madonna can solve boundary value problems by varying one or more parameters such that specified boundary conditions are met. The requirements for using the boundary value solver are similar to those for the curve fitter. However, instead of using imported datasets, the boundary value solver uses one or more boundary conditions that you specify in the **Boundary Value ODE** dialog box.

To solve a boundary value problem, choose **Boundary Value ODE** from the **Modules** submenu in the **Model** menu. Then, complete the dialog box as follows:

1. Specify a boundary condition by: (1) choosing a variable from the **Set** control, (2) entering the target value in the = field, (3) entering the time at which the target value should be reached in the at **X** = field. Then, click the **Add** button. The boundary condition will be displayed in the form var(x)=y in the **Boundary Conditions** list on the right. Repeat this procedure for each boundary condition.

2. To remove a condition you don't want, select it from the list and click **Remove**. To change a condition, remove it, edit the appropriate fields, then add it back again.

3. Specify a parameter to solve for by choosing one from the **Available Parameters** list and clicking the **Add** button. The parameter appears in the **Unknowns** list on the right.

4. Adjust the minimum and maximum values for the parameter so that they are just wide enough to cover the range where you expect the solution to be found. Berkeley Madonna will not search for solutions outside of these limits. If the limits are too large, it will take much longer to find a solution or may not be able to find one at all.

5. Repeat steps 3 and 4 for each parameter you want to solve for. Note that the number of parameters must be the same as the number of boundary conditions specified in step 1.

6. To remove a parameter you don't want, select it from the **Unknowns** list and click **Remove**.

7. Specify the relative accuracy you desire in the **Tolerance** field. The default is 0.001 (one part per thousand).

Once everything is set up, start the solver by clicking OK. The solver will run until it finds a solution that meets all of the specified boundary conditions. If a solution cannot be found, the solver will run indefinitely. You can stop it at any time by clicking the **Stop** button.

When the solver finds a solution, it leaves the parameters set to the values that satisfy the boundary conditions and runs your model, plotting the variables used in the boundary conditions. If you stop the solver before it finds a solution, it leaves the parameters set to the values that came closest to the solution and does not run your model.

Sometimes the solver will complete as if it had found a solution, but when you examine the results of the run it appears that the boundary conditions you specified aren't met. This happens because the solver doesn't actually require all conditions be met exactly (if it did, it would never finish due to the limited precision of floating-point numbers on a computer). Instead, the solver considers its work done when the relative adjustments it makes to each parameter are less than the specified tolerance. If you model doesn't respond smoothly to small parameter changes close to the solution, the solver may think it's close to a solution when in fact it is not. To avoid this problem, try using a lower tolerance in the **Boundary Value ODE** dialog. Or try using a smaller stepsize to improve the stability of your model's output.

Sensitivity

The sensitivity feature computes the sensitivity of variables in your model to changes one or more parameters. The sensitivity of a variable V to a parameter P is computed as follows:

1. Run the model with all parameters at their specified values. We'll call the results of this run V1(t).

2. Adjust parameter P slightly by adding an amount Δ equal to 0.001 * P. Run the model again, calling the results V2(t). Note that if P is zero, Δ is set to 0.001.

3. Compute the sensitivity S(t) by the following formula:

   $$S(t) = (V2(t) - V1(t)) / \Delta$$

When you choose **Sensitivity** from the **Parameters** menu, you can select more than one parameter. Berkeley Madonna will perform one normal run followed by runs where each parameter is perturbed one at a time. It then computes the sensitivity and stores the results as separate runs, one for each parameter you chose. The legend shows the parameter for which sensitivity was measured.

Sensitivity cannot be determined when using variable stepsize algorithms; in order to subtract results of one run from another, Berkeley Madonna needs the step times to match exactly which can only be achieved using fixed stepsize algorithms.

Plug-in Functions and Integration Methods

Berkeley Madonna provides a variety of built-in functions and integration methods. However, you may find that your problem demands one which we have not included in the software. One solution is to write your own "plug-in" functions and integration methods. Plug-ins are written in C or C++ and must conform to Berkeley Madonna's plug-in interface. If you are interested in writing plug-ins for Berkeley Madonna, contact us and we'll send you the necessary documentation.

Flowchart Reference

This section describes advanced techniques for working with visual models. It also describes the various commands and preferences for flowchart windows.

Aliases

As your models grow more complex, you'll encounter situations where you want to make a connection (arc or flow) between two icons that are spaced far apart on the flowchart. If you make the connection between these icons, you'll end up with a long arc or flow line that crosses over a bunch of unrelated icons and connections. Not only does this look messy, but it makes it harder to see the essential structure of your model.

One solution to this problem is to move one of the icons you want to connect closer to the other. However, if both icons already have a lot of connections, moving either one will make your flowchart even more disorganized.

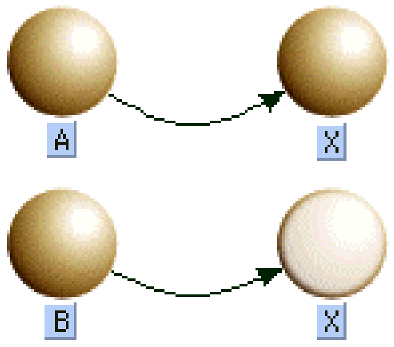
A better solution to this problem is to use an alias. Instead of moving one icon closer to the other, you can create an alias for one of the icons and place it next to the other icon. Then you connect the alias to the other icon with a short, simple connection.

An alias is an icon that provides another way to access an existing "original" icon. The most important thing to remember is that an alias **always** refers to an original. You can locate the original by selecting an alias and choosing **Find Original** from the **Flowchart** menu. Aliases look like hollow versions of their originals.

To create an alias, select one or more existing icons and click the **alias** button in the toolbar. Berkeley Madonna will create one alias for each icon you selected. However, note that aliases will only be created for reservoir, flow, and formula icons. You can create as many aliases for a particular original as you want by repeatedly clicking the alias button. These aliases can be placed anywhere in your flowchart.

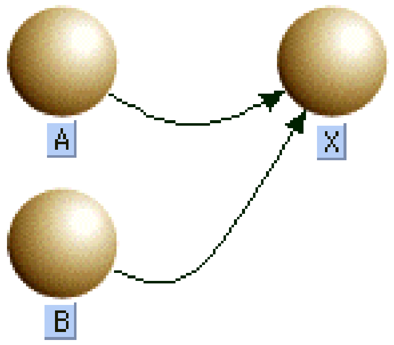
An alias always has the same name as the original to which it refers. If you change the name of an alias, it changes the name of the original and **all** of its aliases. If you double-click an alias, it opens the icon dialog for the original.

If you make connections to an alias, the effect is identical to that resulting from making the same connections to the original. For example, suppose you have an original formula named X and one arc going from another formula A to X. Also, suppose that you have an arc going from another formula B to an alias of X, like this:



In this case, formula X depends on formula A and formula B, just as if an arc had been drawn from formula B to the original formula X.

When you delete an alias, any arcs connected to it are also deleted. And if it is a reservoir, any flows connected to it are detached and replaced by infinite source/sinks. However, it is possible delete an alias while preserving its connections. This is called "alias merging" and is accomplished by dragging the alias you want to remove to its original (or another one of its aliases) and dropping it. Upon dropping the alias, it disappears and its connections are merged into the icon on which it was dropped. For example, in the above flowchart you can drag the alias X to the original X and drop it. Berkeley Madonna deletes the alias and connects the arc from formula B to the original X, like this:



Alias merging is very useful when working with submodels as you will soon see.

Submodels

You've already seen how aliases can help make complex models easier to construct and maintain. But as your models continue to grow, you'll soon realize that there simply isn't enough room in a single flowchart window to contain all of your model's elements. And even if you were able to cram your entire model into one full-screen window, it would difficult to see its overall structure through the forest of arcs, flow lines and icons.

Berkeley Madonna allows you to break your model down into functional parts by placing them in submodels just as you would place related groups of files into a folder or directory on your computer. Submodels appear as icons in your flowchart. The flowchart containing a submodel icon is referred to as the submodel's parent flowchart. When you double-click a submodel icon, a separate flowchart window is opened showing the contents of that submodel.

Creating and Deleting Submodels

To create a new, empty submodel, click the submodel tool on the toolbar, drag the mouse over the flowchart and release it. Berkeley Madonna places a new submodel icon on your flowchart.

Often, you'll want to create a new submodel containing some icons that already exist on your flowchart. To do this, select the icons and choose **Group** from the **Flowchart** menu. Berkeley Madonna creates a submodel icon on your flowchart and places the previously selected icons inside that submodel. Note that when icons are moved into a submodel, aliases may be created in order to preserve the connections that existed before the submodel was created. See Moving Icons Between Submodels on page 59 for details about how this works.

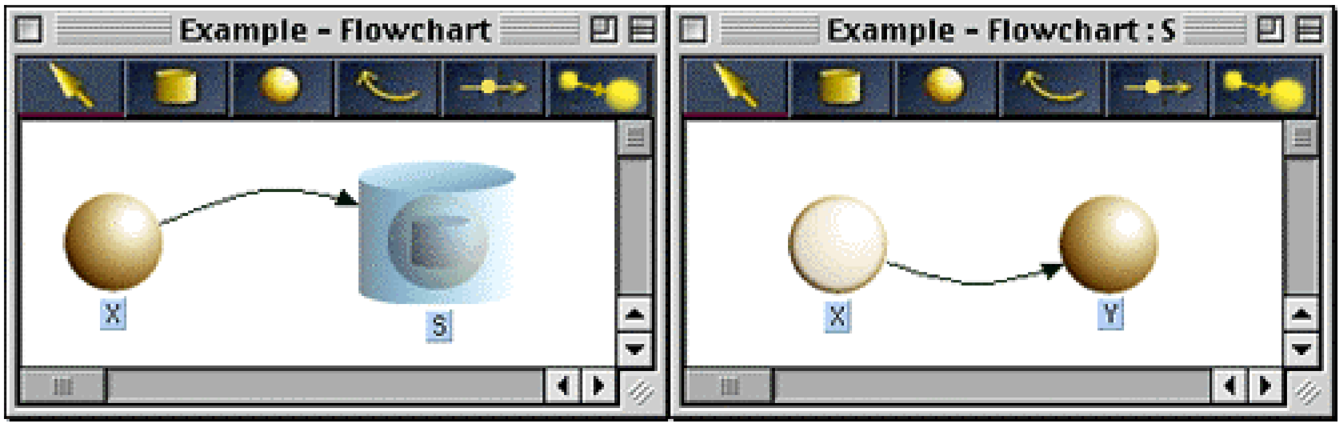
To edit the contents of a submodel, you must first open its flowchart window by double-clicking the submodel's icon. Once opened, you can edit the submodel's flowchart in the same way you edit your model's top-level flowchart.

When you delete a submodel icon, you delete not only the icon but all of the icons it contains. However, you can delete a submodel icon while preserving the icons and connections it contains by selecting one or more submodel icons and choosing **Ungroup** from the **Flowchart** menu. Berkeley Madonna removes the submodel icon and moves all the icons it contained into the parent flowchart.

Making Connections Between Submodels

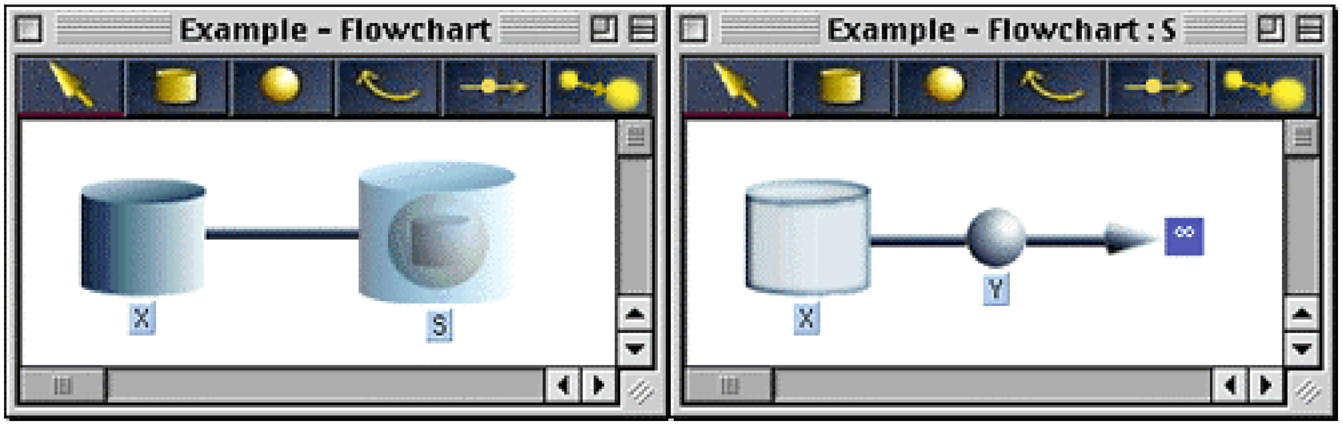
Submodels wouldn't be very useful if you couldn't make connections between icons in different submodels. But simply creating an arc or flow between flowchart windows doesn't work. For example, place a formula icon named X and a submodel icon named S in an empty flowchart window. Open the submodel's flowchart by double-clicking its icon and place a formula icon named Y in its flowchart. Now place the two flowchart windows side-by-side and try creating an arc from X to Y. As you can see, Berkeley Madonna doesn't let you create a connection across two flowchart windows.

So, how do you make a connection between two icons in different submodels? You use an alias. Continuing the above example, create an alias to the X icon and drag it to the submodel icon. This moves the alias from the top-level flowchart into submodel S. Now you can make the connection from X's alias to icon Y in the submodel's flowchart window. Try it. Your flowcharts should end up looking like this:



Notice that when you created the arc from X's alias to Y, another arc was magically created in the top-level flowchart going from X to submodel S. Berkeley Madonna does this to remind you that something in submodel S (in this case, Y) depends on X. This "magic" arc is referred to as a "bundle" since it can represent more than one dependency between X and icons within S. For example, if you create icons A, B, and C within S and connect arcs from X's alias to these icons, the top-level flowchart's appearance will not change. The bundle (arc) from X to S now represents all of the connections from X to icons within S (A, B, C, and Y).

Berkeley Madonna also creates bundles representing one or more flows connected to reservoirs. For example, try this: create a reservoir X in the top-level flowchart. Create an alias to X and move it into S. Then create a flow Y within S going from X's alias to an infinite sink. You should end up with something like this:



When you create the flow Y going from X, Berkeley Madonna creates a bundled flow or "pipe" in the top-level model connecting the original X icon and the submodel. This bundle reminds you that something in S is moving material in or out reservoir X. You could create additional flows in S connected to X and the appearance of the top-level flowchart would not change.

Berkeley Madonna uses a bunch of complex algorithms to ensure that these bundles represent actual flow and arc connections you've made. Explaining these algorithms would be difficult and probably not very helpful. There are a couple of simple rules to remember that will help you avoid confusion:

- Bundles are **always** connected between a reservoir, flow, or formula icon and a submodel icon. Whenever you see an arc connected to or from a submodel icon, it is a bundle created automatically by Berkeley Madonna. The same holds for flow bundles (pipes), but they are easier to distinguish since they have no flow icon or label associated with them.

- You cannot directly create or delete bundles. For example, you can't draw an arc from a submodel to another icon. The only way to delete bundles is to remove the underlying connection(s) they represent.

One final note about bundles. If Berkeley Madonna needs to create a bundle from a submodel icon to another icon that has no representation in the same flowchart window, it will create an alias to that icon. This automatic alias creation is harmless but can be distracting. To minimize the need for additional aliases, try to keep your connections between neighboring submodel levels (from parent to child and vice-versa) as we did in the above examples.

Moving Icons Between Submodels

You've already seen that you can create connections between icons in different submodels by creating connections between aliases. But you can achieve the same effect by moving icons from one submodel to another **after** connections to them have already been created. In some ways, this technique is simpler because it enables you to construct parts of your model within a single flowchart window where all icons are accessible at once. Then, you can isolate parts of your flowchart within their own submodels and let Berkeley Madonna worry about preserving connections between them.

To illustrate how this method of construction differs, let's try to recreate the example model we discussed earlier. Create a new flowchart and place two formula icons named X and Y. Draw an arc from X to Y. Now, you decide that you want to place Y in its own submodel. To so this, select Y and choose **Group** from the **Flowchart** menu. Now Y has been replaced by a submodel icon named "S1". Open S1 and you'll see it has the same structure as when you created the alias to X yourself. If you want to make things look exactly like they did before, you can change the name of the submodel to “S”.

Once you’ve created a submodel or two, you may want to move icons from one submodel to another. Berkeley Madonna supports this kind of rearrangement in two ways. First, you can move a set of icons from a flowchart into one of its submodels (children) by dragging the icons to the submodel icon and dropping them inside. Second, you can move a set of icons from a submodel into its parent flowchart (the flowchart containing that submodel’s icon) by selecting the icons and choosing **Move To Parent** from the **Flowchart** menu.

As you begin to move icons around between submodels, you’ll find that Berkeley Madonna can create a lot of aliases along the way. Or you might find some icons moving that you didn’t want moved. Here are some tips that will help you understand what’s going on:

- When you drag an icon from a parent model into a child, its connections to other icons are also moved. For example, say you have a reservoir connected to a flow and a formula in your top model. If you drag this reservoir to a submodel, the arcs and pipes that connect this reservoir to the flow and formula are also moved to the submodel. After the connections are moved, aliases are created in the submodel as needed so that the other ends of these connections are maintained.
- When you move icons from a submodel into its parent, Berkeley Madonna will create aliases in the parent model to preserve any connections that were moved. This behavior can lead to a proliferation of aliases in the parent model. These aliases can often be moved back into the submodel and then merged with their originals to reduce clutter.
- When deciding where to place original icons in your model, remember that it’s best to put originals high enough in the hierarchy so that all their references come from other icons in the same flowchart (including submodels). Try to avoid putting icons down in a submodel when they are referenced by other icons in the parent model or sibling submodels.

Take a look at the “HH Axon (Submodels)” model in the “Examples” folder. Although it is somewhat involved, each submodel is easy to understand and you can get a good view of the model’s overall structure just by looking at the top-level flowchart.

Flowchart Toolbar

The flowchart toolbar appears at the top of every flowchart window in Berkeley Madonna. The active tool is indicated by a red bar drawn beneath it. When you click the mouse in the flowchart, the function performed depends on the active tool as described in the table below. After using most tools, the **Select** tool is activated. However, you can make certain tools remain active after use, or “sticky”, by holding down the control key when selecting the tool. When the active tool is sticky, it is denoted by a thicker red bar.

Tool	Description
Select	Use the select tool to select one or more icons. Clicking on an icon selects it and deselects any previously-selected icons. Shift-clicking an icon toggles its selection status without deselecting previously selected icons. You can also drag a selection rectangle around a group of icons. To deselect all icons, click an empty space in the flowchart. Normally, you don't need to choose this tool since it is activated after using any of the other tools. However, if you use the "sticky tool" feature described above, you'll need to explicitly activate the Select tool when you're done using the sticky tool.
Reservoir	Use the reservoir tool to create a new reservoir on the flowchart. There are two ways to do this. One is to select the tool by clicking once, then moving over the flowchart and clicking again. The other way is to click the reservoir tool, drag the mouse over the flowchart, then release the mouse. If you want to create many reservoirs rapidly, control-click the reservoir tool to make it sticky, then repeatedly click in the flowchart. Remember to click the select tool when you're done using the sticky reservoir tool.
Formula	Use the formula tool to place formula icons on the flowchart. You can use either the two-step, one-step (drag), or sticky methods as described for reservoir tool.
Arc	Use the arc tool to create a dependency relationship between two icons. After selecting the tool, move the mouse to the source icon, click and drag the mouse to the destination icon, and release the mouse. This creates an arc pointing from the source to the destination icon representing the destination icon's dependency on the source icon. The source icon must be a reservoir, formula, or flow. The destination icon must be a formula or flow. You can use the sticky tool feature if you need to create many arcs quickly.
Flow	Use the flow tool to represent a flow of material between a reservoir and an infinite source/sink or between two reservoirs. After selecting the tool, click the mouse on the source, then drag the mouse to the destination and release it. If you want the source or destination to be an infinite source or sink, respectively, start or end the drag at an empty area of the flowchart and Berkeley Madonna will create the source/sink for you. If a source/sink is created but you intended to connect that end to a reservoir, drag the source/sink to the reservoir to make the connection. You can use the sticky tool feature to create a number of flow connections quickly.
Submodel	Use the submodel tool to organize portions of your model into a hierarchical structure. This feature is described in Submodels on page 57.

Tool	Description
Text	Use the text tool to create text labels on the flowchart. After selecting the tool, click on the flowchart to place an empty text box. You can then type in your label text. Unlike other tools, this tool is always sticky. When this tool is active, you can click on any text label (including those associated with icons) to position the insertion point for editing.
Align	Use the align tool to align the top, bottom, left, or right edges of the selected icons. The direction of alignment depends on where you position the mouse within the tool. Watch the help text as you move the mouse within this tool.
Grid	Use the grid tool to turn the grid on and off. When the grid is on and you move an icon, its position is adjusted so that the edge closest to a gridline coincides with that gridline.
Alias	Use the alias tool to create an alias for the selected icon(s). An alias is a separate representation of the original icon. Alias icons appear as hollow versions of their originals. See Aliases on page 55 for more information.
Show Formulas Hide Formulas	Use the show/hide formulas tool to toggle the visibility of formulas (and their connecting arcs) in the flowchart. Hiding formulas and their arcs can make a complex flowchart easier to understand. This is equivalent to toggling the **Formulas** checkbox in the **Icon Visibility** dialog.

Flowchart Commands

Commands in the **Flowchart** menu apply to the active flowchart window. In addition, there are a few other flowchart-related commands in other menus. Commands shown in the following table appear in the **Flowchart** menu unless otherwise specified.

Command	Description
New Flowchart (File menu)	Creates a new visual model and opens an empty top-level flowchart window.
Cut (Edit menu)	Copies the selected icons to the clipboard and removes them from the flowchart.
Copy (Edit menu)	Copies the selected icons to the clipboard.
Paste (Edit menu)	Pastes the icons on the clipboard to the active flowchart. If the names of any pasted icons clash with existing icons, they are given new names.
Clear/Delete (Edit menu)	Removes the selected icons from the active flowchart.

Command	Description
Select All (Edit menu)	Selects all icons in the active flowchart.
Show Icon (Edit menu)	Selects the icon corresponding to the specified equation in the equation window. The specified equation is indicated by the position of the insertion point or the beginning of the selected text. Available only when the equation window is active.
Copy Image	Copies the entire flowchart to the clipboard as a picture. This picture can then be pasted into other applications.
Icon Info...	Displays the icon dialog for the selected icon. For submodel icons, this command opens the submodel's flowchart window. For text labels created with the text tool, this command displays the text dialog which allows you to change the text style. This command is available only when a single icon is selected and has the same effect as double-clicking an icon.
Find Original	Locates and selects the original icon for the selected alias icon. Available only when a single alias icon is selected.
Group	Places the selected icons into a submodel. See Submodels on page 57 for more information.
Ungroup	Moves icons from the selected submodel into its parent model, then deletes the submodel. See Submodels on page 57 for more information.
Move To Parent	Moves the selected icons to the parent model. See Submodels on page 57 for more information.
Show Parent Window	Activates the flowchart window containing the active flowchart's submodel icon. See Submodels on page 57 for more information.
Redraw	Redraws the flowchart window. Useful when a portion of the flowchart appears corrupted.
Background Color...	Displays a dialog which enables you to change the flowchart's background color.
Import Icon Image...	Replaces the selected icon(s) default image with a custom image from an external file. The image file must be in GIF or JPEG format. This command is available only when one or more icons are selected.
Import Background Image...	Displays an image from a GIF or JPEG file in the active flowchart window. This command is available only when no icons are selected.
Remove Icon Image	Removes the custom image from the selected icon(s). This command is available only when one or more icons with custom images are selected.

Command	Description
Remove Background Image	Removes the background image from the active flowchart window. Available only when the window has a custom background image and no icons are selected.
Icon Visibility...	Displays a dialog which enables you to hide different kinds of icons and connectors in the flowchart.
Tools	This submenu provides another way to select various tools in the toolbar.
Arrange » Front	Makes the selected icons appear in front of all other icons.
Arrange » Back	Makes the selected icons appear behind all other icons.
Arrange » Align	Aligns the top, bottom, left, or right edges of the selected icons. Same as the align tool on the toolbar.
Arrange » Distribute	Distributes the selected icons evenly across in the flowchart in the horizontal or vertical direction.
Arrange » Snap To Grid	Aligns the selected icons with the gridlines. Works even when the grid is not visible.
Show Flowchart	Shows the active model's hidden flowchart windows. If none of the flowchart windows are hidden, it simply opens the top-level flowchart. This command changes to **Hide Flowchart** when at least one flowchart window is open and none of the flowchart windows are hidden.
Hide Flowchart	Hides the active model's flowchart windows. The hidden windows can be quickly restored by choosing **Show Flowchart**. Note that closing a flowchart window by clicking its close box closes the window rather than hiding it. This distinction is important because only hidden flowchart windows are restored when **Show Flowchart** is chosen.
Discard Flowchart (Flowchart and Model menus)	Removes the flowchart from the active model, converting it into a plain text model. Once this is done, the flowchart cannot be recovered. However, you can continue to edit the equations in the equation window.

Flowchart Preferences

The behavior of Berkeley Madonna's flowchart editor can be customized using the **Flowchart** page of the **Preferences** dialog as follows:

Preference	Description
Arc Style	Specifies the type of curve used for dependency arcs between icons. The default is a "single point" Bezier curve whose trajectory is adjusted via a single control point. "Double point" bezier curves provide more control over the trajectory but require twice as much fiddling to position since they have two control points. The "straight line" option makes all arcs into straight lines.

Preference	**Description**
Grid On Initially	If checked, the grid in flowchart windows is turned on when they are first opened.
Drag Outlines	When checked (the default), an icon's outline is shown during dragging and the icon is not moved until you release the mouse. When not checked, the icon itself is moved during a dragging operation. While this looks cool, dragging the entire icon is much slower. Also, certain drag/drop operations such as dragging icons into submodels and alias merging may not function properly unless outline dragging is enabled.
Show Drag Alerts	When checked (the default), error messages are displayed if you attempt to make an invalid connection with a flow or arc, for example, trying to draw an arc from one reservoir to another. Turning off this preference prevents these error messages from being displayed.
Small Icons	If you uncheck this option (it is checked by default), Berkeley Madonna uses larger icons in your flowchart.
Grid Spacing	Specifies the distance between gridlines in the flowchart.
Max. Text-to-Owner Distance	Specifies the maximum distance between and icon and its text label. Berkeley Madonna ensures that the distance between the icon's enclosing rectangle and its text label is not greater than this distance.

Customizing Icons

The **Import Icon Image** command discussed earlier can be used to substitute a custom image (in a GIF or JPEG file) for Berkeley Madonna's default icon images. This mechanism has some drawbacks:

- It is difficult to tell when icon with a custom image has been selected because it doesn't change appearance.
- Custom icons considerably increase the size of the model file. This is especially wasteful when the same images are used in many models.

Berkeley Madonna provides a alternative way to customize the images used for all instances of a particular type of icon. This method makes it possible to change the entire look of your flowcharts.

To customize icons in this way, create a folder named "icons" in the same location as the Berkeley Madonna application file. Place your custom icon image files in this folder. The only requirements are that the image files be in the GIF format and that they adhere to the following naming:

<type><select><alias><size>.gif

where:

- <type> is a three-letter code indicating the type of icon:

con	formula icon (converter)
flo	flow icon
mod	submodel icon
res	reservoir icon

- <select> is a single letter indicating whether the image is for a selected icon or a normal (unselected) icon:

n	Normal icon (not selected)
s	Selected icon

- <alias> is a single letter indicating whether the images is for an alias icon or a normal (non-alias) icon:

n	Normal icon (not an alias)
a	Alias icon

- <size> is a single letter denoting the size of the icon:

s	Small icon
l	Large icon

Here are some examples:

resnas.gif	Reservoir alias icon, not selected, small size
modsnl.gif	Submodel icon, selected, large size

Note that since there is no such thing as an alias to a submodel icon, images named **mod**X**a**X**.gif** are not used.

The customized icon images will be recognized the next time Berkeley Madonna is launched.

Known Bugs and Limitations

1. The flowchart editor cannot generate equations employing some of Berkeley Madonna's more advanced equation features. These include:

    - Array equations.
    - Upper and lower bounds (LIMIT statement).
    - Built-in root finder equations.
    - Renaming system symbols (RENAME statement).
    - Exposing a limited subset of a model's symbols (DISPLAY statement).

    Also, the Chemical Reactions feature cannot be used with visual models.

    We may add support for some of these features in a future release of Berkeley Madonna.

2. Conveyors, ovens, and queues cannot be represented with stock icons and flows like reservoirs. Instead, they must be defined using formula icons.

3. The icon dialog allows you to define graphical functions using Berkeley Madonna's GRAPH() built-in function. However, the Graph Input dialog is fairly buggy and difficult to use.
4. Operations involving the dragging of one or more source icons to a target icon do not work when the **Drag Outlines** preference is turned off. These operations are alias merging, connecting infinite source/sinks to reservoirs, and moving icons to submodels. We recommend leaving this preference turned on when performing these operations.