Agilent Technologies

Genesys 2010
2010
Users Guide

Acknowledgments
Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries. Microsoft®, Windows®, MS Windows®, Windows NT®, and MS-DOS® are U.S. registered trademarks of Microsoft Corporation. Pentium® is a U.S. registered trademark of Intel Corporation. PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated. UNIX® is a registered trademark of the Open Group. Java™ is a U.S. trademark of Sun Microsystems, Inc. SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission. MATLAB® is a U.S. registered trademark of The Math Works, Inc.. HiSIM2 source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code in its entirety, is owned by Hiroshima University and STARC. Drawing Interchange file (DXF) is a trademark of Auto Desk, Inc. EMPOWER/ML, Genesys, SPECTRASYS, HARBEC, and TESTLINK are trademarks of Agilent Technologies, Inc. GDSII is a trademark of Calma Company. Sonnet is a registered trademark of Sonnet Software, Inc.

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license. Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at http://systemc.org/ . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special, incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.
With respect to the portion of the Licensed Materials that describes the software and provides instructions concerning its operation and related matters, "use" includes the right to download and print such materials solely for the purpose described above.

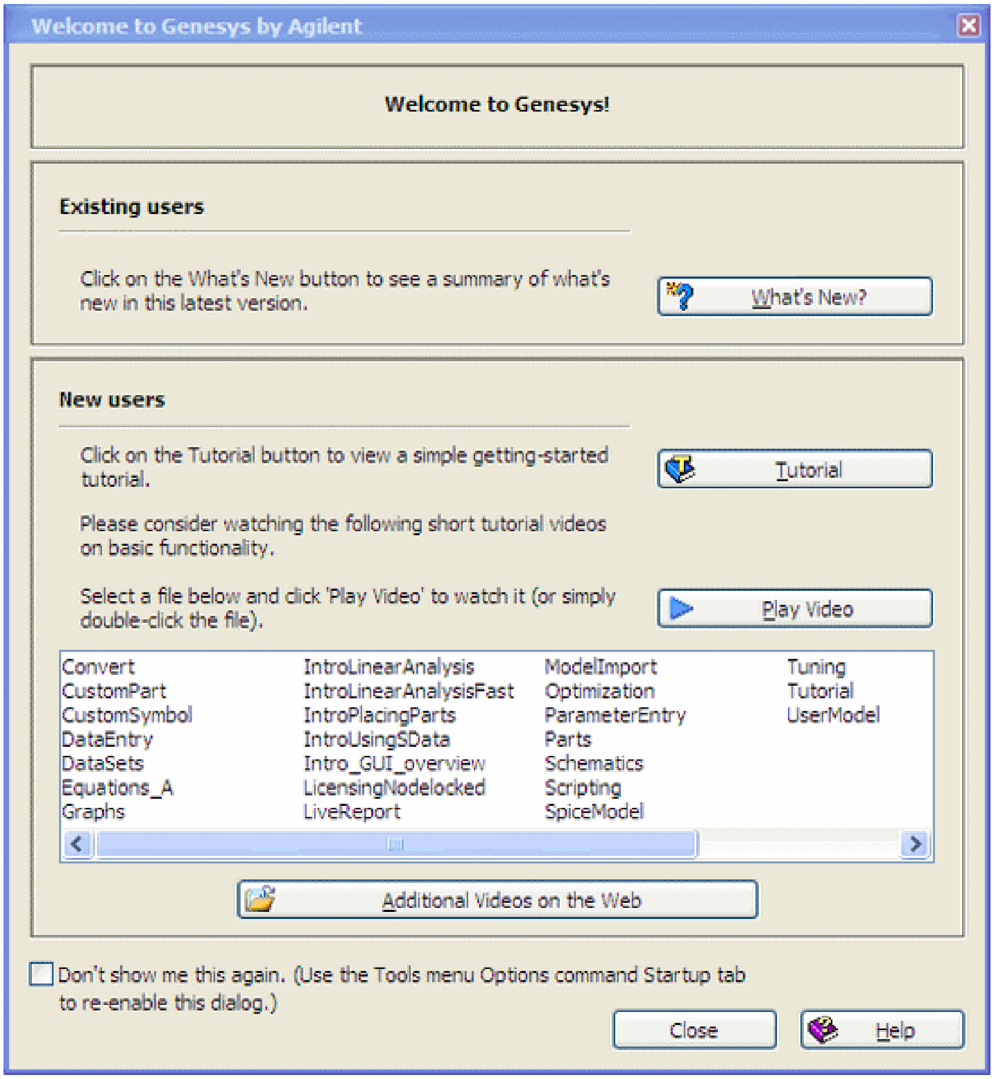
The Genesys Environment

This section will familiarize you with the user interface of Genesys. These sections include introductions of various components of the Genesys software screen. To get more detailed information about using these features, read *Using Genesys* (users) section.

Contents

Starting Genesys

To start Genesys:
Click **Start > Genesys**. Loading Genesys screen opens. After a while Genesys main window opens launching the welcome dialog:



Welcome to Genesys by Agilent

Welcome to Genesys!

Existing users

Click on the What's New button to see a summary of what's new in this latest version.

New users

Click on the Tutorial button to view a simple getting-started tutorial.

Please consider watching the following short tutorial videos on basic functionality.

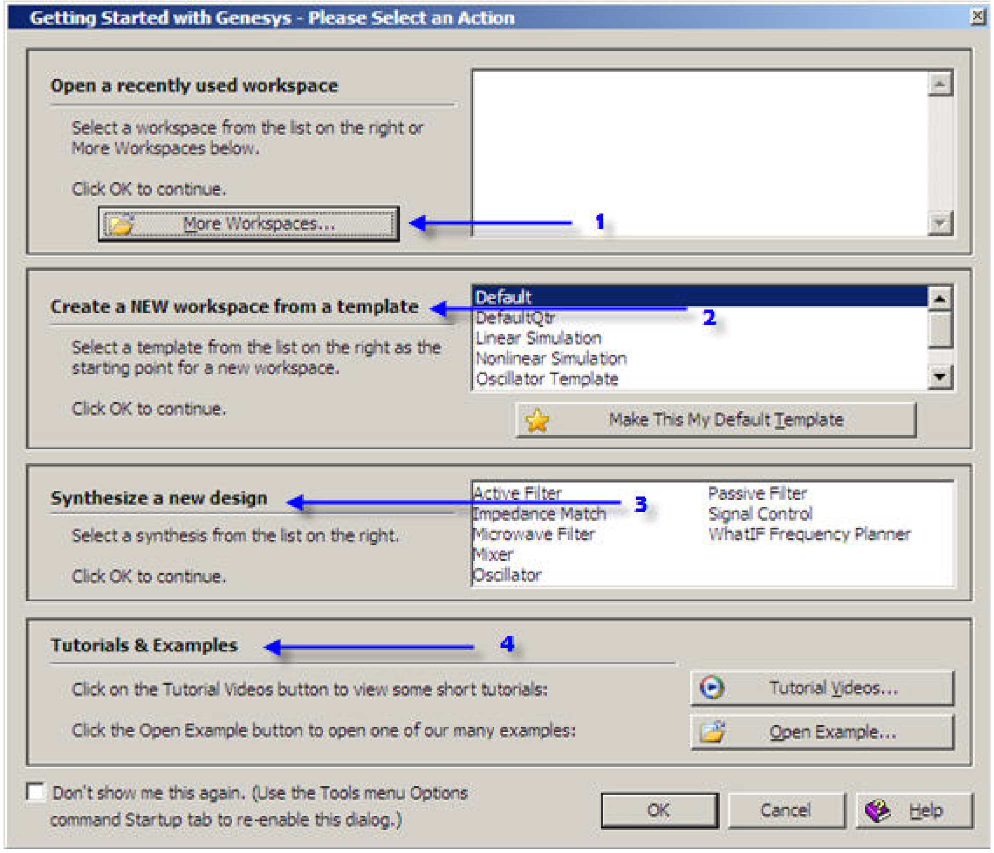
Select a file below and click 'Play Video' to watch it (or simply double-click the file).

□ Don't show me this again. (Use the Tools menu Options command Startup tab to re-enable this dialog.)

This window is also called the **splash screen**.

Note
You might want to view some of the videos listed in the **New Users** section, even if you're an experienced user. They are typically short and cover some of the most useful convenience and quick-start topics.

To close the window, click **Close**. The control automatically shifts to the **Getting started** screen with the title - **Getting Started with Genesys - Please Select an Action**.



Getting Started with Genesys - Please Select an Action

Open a recently used workspace

Select a workspace from the list on the right or More Workspaces below.

Click OK to continue.

Create a NEW workspace from a template

Select a template from the list on the right as the starting point for a new workspace.

Click OK to continue.

Synthesize a new design

Select a synthesis from the list on the right.

Click OK to continue.

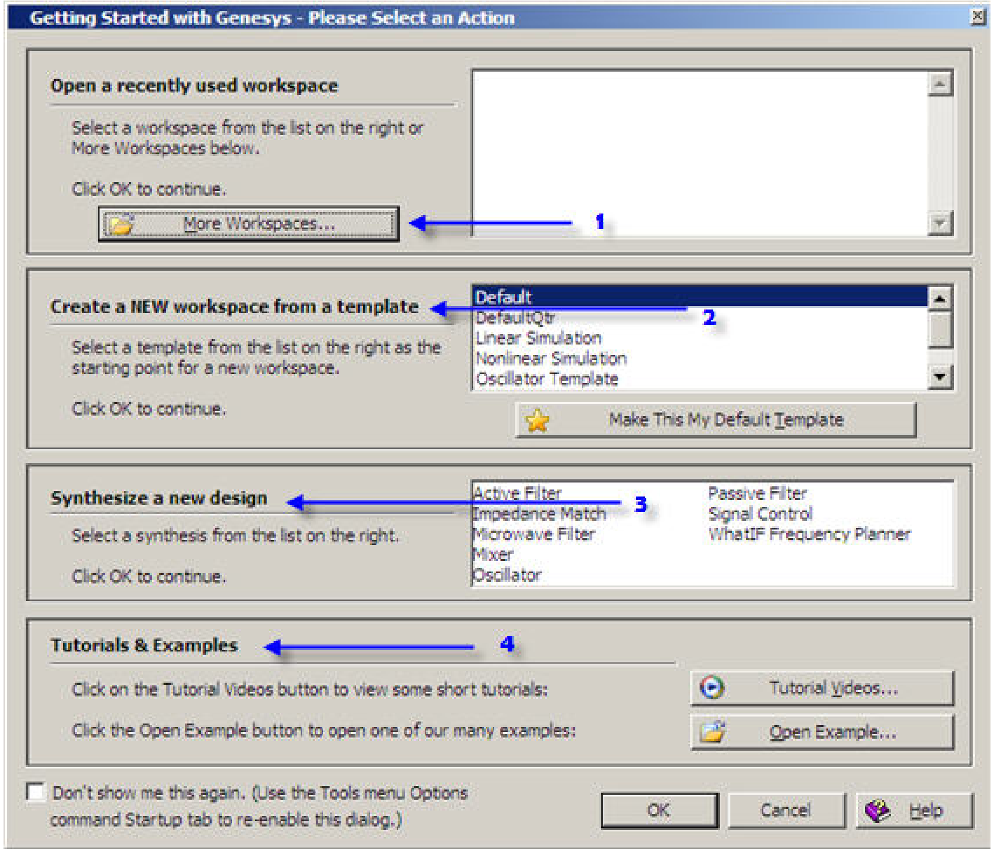
Tutorials & Examples

Click on the Tutorial Videos button to view some short tutorials:

Click the Open Example button to open one of our many examples:

□ Don't show me this again. (Use the Tools menu Options command Startup tab to re-enable this dialog.)

Getting Started with Genesys Dialog



Getting Started with Genesys - Please Select an Action

Open a recently used workspace

Select a workspace from the list on the right or More Workspaces below.

Click OK to continue.

Create a NEW workspace from a template

Select a template from the list on the right as the starting point for a new workspace.

Click OK to continue.

Synthesize a new design

Select a synthesis from the list on the right.

Click OK to continue.

Tutorials & Examples

Click on the Tutorial Videos button to view some short tutorials:

Click the Open Example button to open one of our many examples:

□ Don't show me this again. (Use the Tools menu Options command Startup tab to re-enable this dialog.)

The following are the options that you can exercise with your Getting Started dialog box:

1. **Open a recently used workspace**
   You can select a recently used workspace from the list on the right. If the recent space is not listed, click **More Workspaces** to locate a workspace on your computer system.
2. **Create a NEW Workspace from a template**
   You can select a template fr0om the list on the right. Select **Default** to start withe the default template or you can select any template and click **Make This My Default Template** to make the selected template as default one.
3. **Synthesize a new design**
   Select a synthesis from the list on right to synthecize a new design.
4. **Tutorials and Examples**
   There are two buttons available to access help for videos and examples.

There is "Don't show me again" button on this page as well, but it is recommended not to select it (for new users).

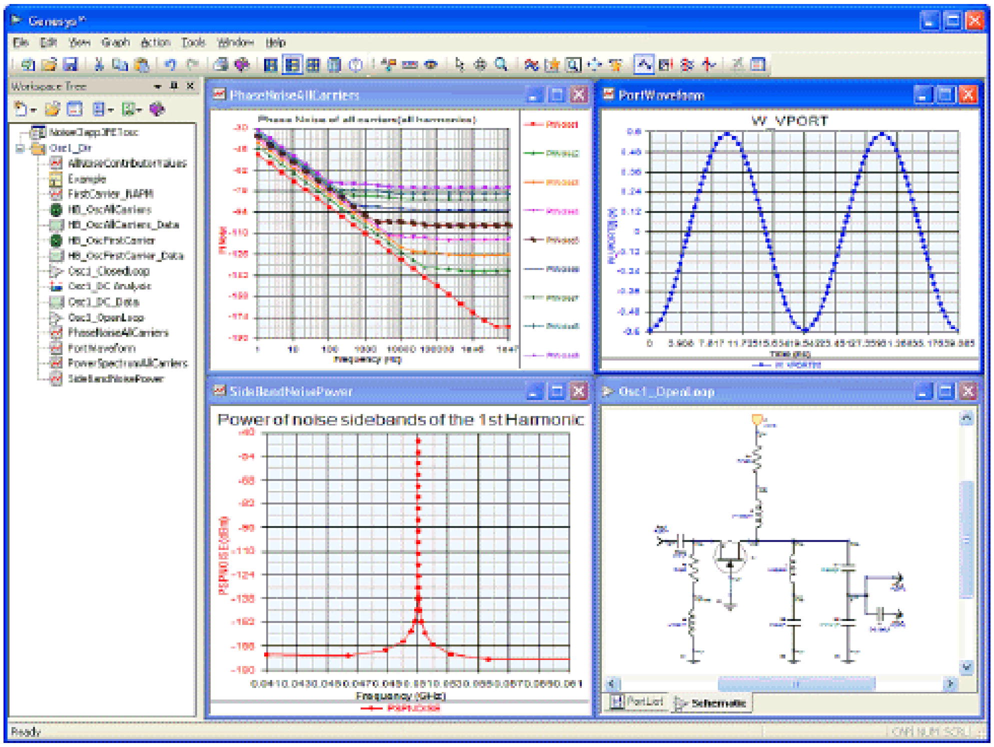
If you simply close this window, you get to the default workspace.

> **Note**
> Click the Start Page button , (the first button in the main toolbar) to open this dialog anytime.

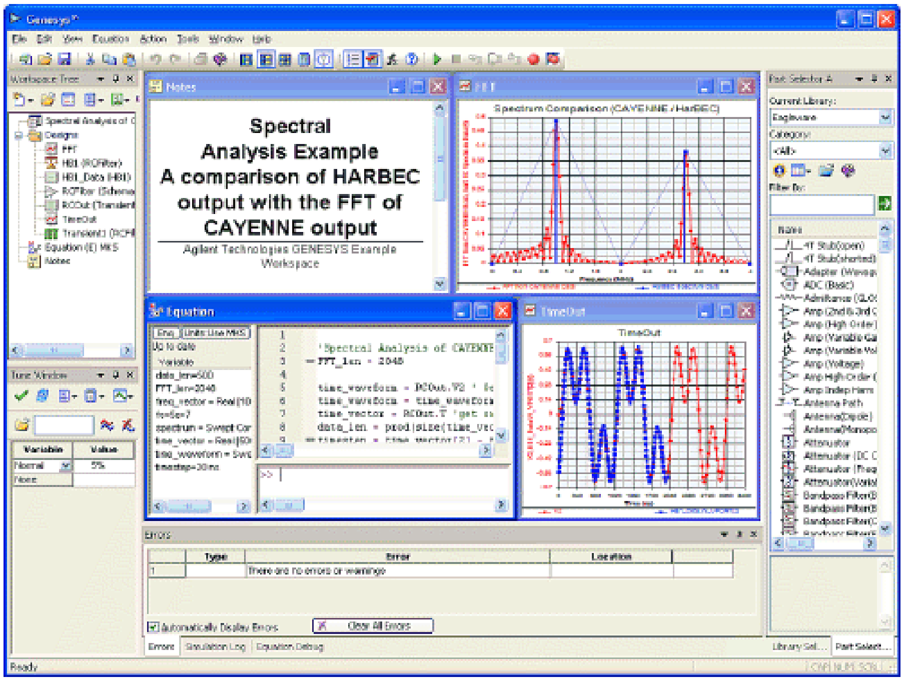
Genesys Design Environment (User Interface)

The Genesys design environment consists of menus, windows, toolbars, and standard editing options. It is easily integrated with other programs, and you can use it to view multiple projects, schematics, and simulations at the same time.

The environment is versatile. It can look like this...



or like this...



Spectral
Analysis Example
A comparison of HARBEC
output with the FFT of
CAYENNE output

Agilent Technologies GENESYS Example Workspace

```
'Spectral Analysis of CAYENNE
FFT_len = 2048

time_waveform = RCOut.V2 ' Sc
time_waveform = time_wavefora
time_vector = RCOut.T 'get si
data_len = prod(size(time_vec
timestep = time_vector[2] - t
```

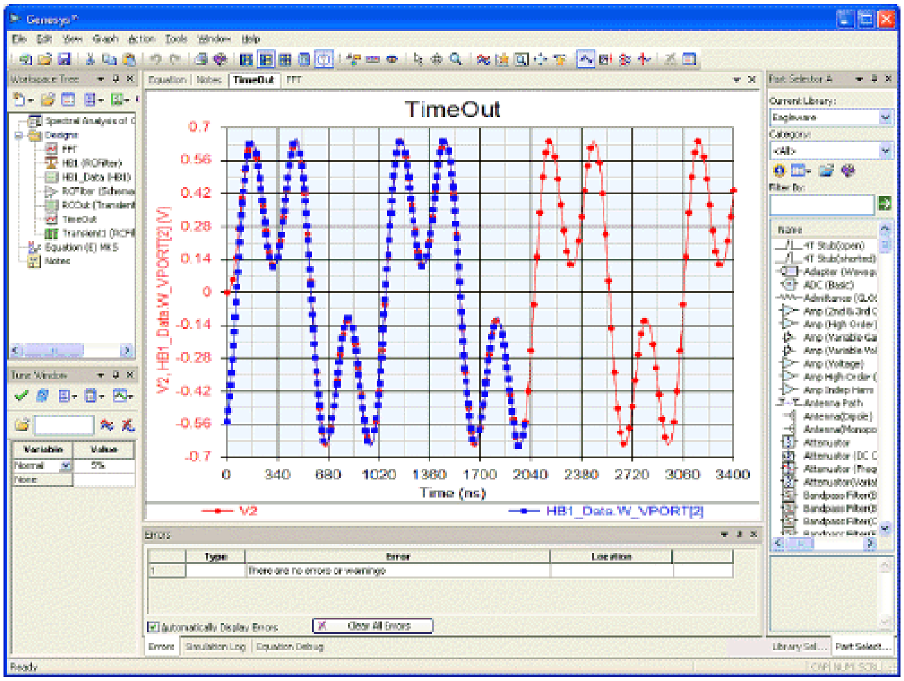
	Type	Error	Location
1		There are no errors or warnings	

☑ Automatically Display Errors   Clear All Errors

Errors | Simulation Log | Equation Debug

We show you the second environment by **default** so you know what is available and what it looks like. In the design area, an un-maximized window can be re-sized by clicking on a boundary and dragging it. Three buttons on the right of the title bar control a design window's viewability. Click on ▁ to minimize the window. Click on □ to maximize the window. Click on ☒ to remove the window.

Also, it's easy to switch to a tabbed window view, by using the *Window Menu* (users). Note the view window tabs at the top of the graph:



1 There are no errors or warnings

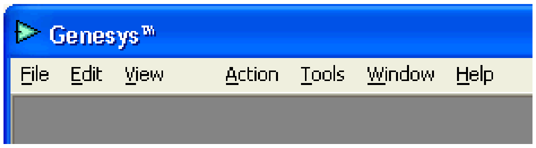
The default Genesys design environment consists of the following:

- ***Menu* (users)** – Contains all of the commands used in Genesys.
- ***Toolbars* (users)** – Contains buttons that are shortcuts for commonly used commands.
- ***Workspace Tree* (users)** – Displays a hierarchical list of items in your project.
- ***Part Selector* (users)** – Lists the electrical parts in a specific library.
- ***Library Selector* (users)** – Lists all of the designs in a specific library.
- ***Tune Window* (users)** – Contains settings that let you modify variables for a circuit in your design.
- ***Simulation Status Window* (users)** – Displays the status of the running simulation.
- ***Error Log* (users)** – Displays error information.
- ***Status Bar* (users)** – Displays useful information at the bottom of the Genesys window.
- ***Design Windows* (users)** – where all the real work takes place, are placed within the gray workspace area.
- ***Simulation Log* (users)** – Information regarding any running simulation.

Click on the page to read more.

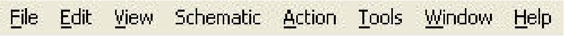
Menus

The Genesys menus are located on the menu bar at the top of the Genesys window. There are several menus that appear automatically whenever Genesys is started. These are called default menus.

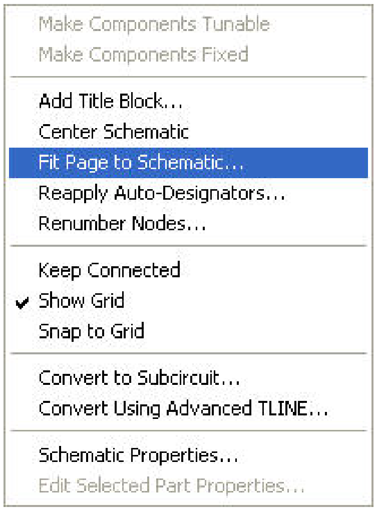


The other Genesys menus are called object menus. They are specific to the windows in a design and appear only when that window is active. For example, the Schematic menu is visible only when the Schematic window is active.

The top is the main Genesys menu like the Windows menu. It contains the basic menus alongwith a **Schematic** menu which is dedicated to Genesys schematic.

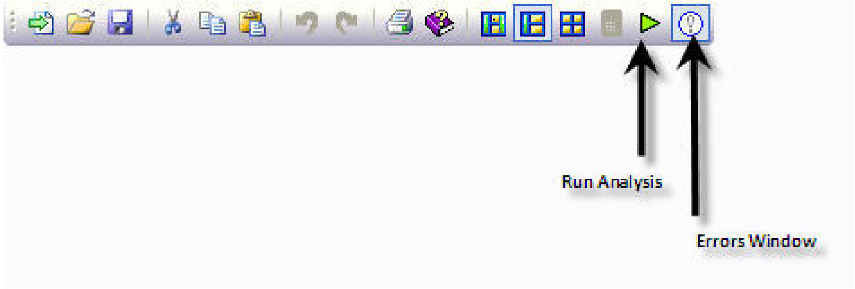


The schematic menu is used for modifying the schematics in the workspace.



List of common icons

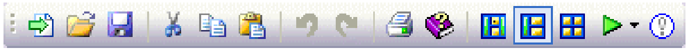
Most of these are common Windows icons itself.



For more information about all of the Genesys menus, see *Appendix B Menus* (users).

Toolbars

There are many toolbars in Genesys. The main Genesys toolbar is referred to as a *default toolbar* (users). The main Genesys toolbar is shown below:



Genesys also has a number of other toolbars called object toolbars. They are specific to the windows in a design and appear only when that window is active. For example, the Schematic toolbar is visible only when the Schematic window is active.

To reposition a toolbar:

- Drag the toolbar to the new location.

To re-size a toolbar:

- Drag a corner of the toolbar until it changes to a different size.

To create a floating toolbar:

- Drag the toolbar to the desktop.

> **Note**
> If you do not want the toolbar to dock to the sides or top of the Genesys window, hold down the CTRL key while dragging.

Using a Default Toolbar

You can use the main Genesys toolbar to perform basic editing commands, such as opening, saving, or printing designs.

To show or hide a default toolbar:

- Click **View** on the Genesys menu and select the toolbar you want to show or hide from the Toolbars menu.

> **Note**
> Toolbars that are currently open have a check mark next to them.

To display the default toolbars on startup:

1. Click **Tools** on the Genesys menu and select **Options**.
2. Click the **Startup** tab.
3. Click the **Use Default Toolbar Settings on Startup** button.
4. Click **OK**.

Using an Object Toolbar

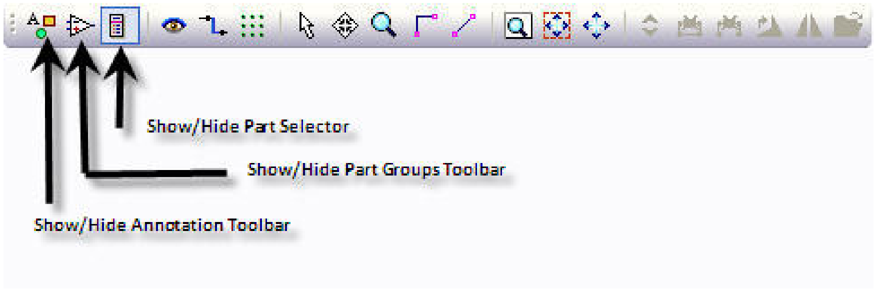
The object toolbars let you perform actions for specific windows.

To show or hide an object toolbar:

- Click **View** on the Genesys menu and select either **Show All Object Toolbars** or **Hide All Object Toolbars** from the Toolbars menu.

Example of schematic toolbar

This is specific to the window that is opened in the workspace area. The icons specific to a schematic window are:

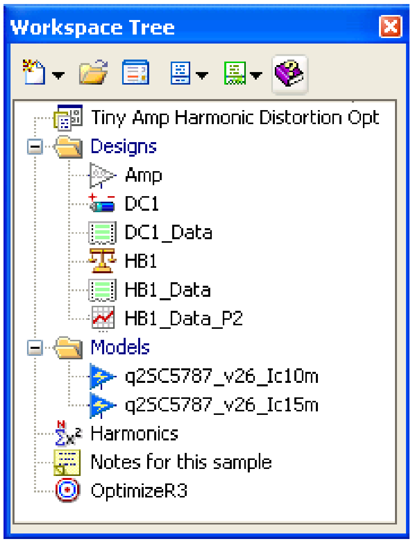


✓ **Tip**
Dock your toolbars in 2 rows, so that the inner window area doesn't change size every time you switch active windows (from a graph to a schematic, etc.).

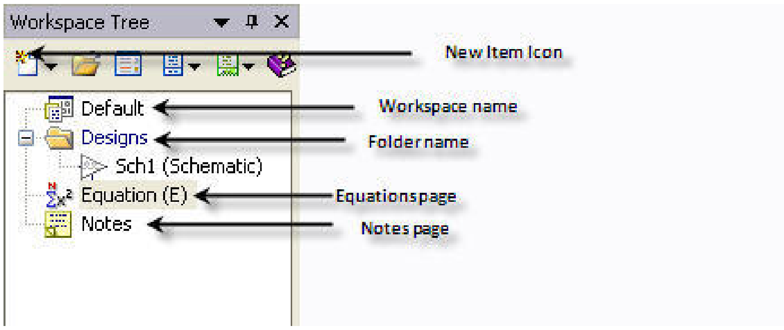
For more information about all of the Genesys toolbars, see *Appendix C Toolbars* (users).

Workspace Tree

The Genesys Workspace Tree displays a hierarchical list of items in your project such as designs, analysis, data sets, and graphs. With it, you can add, delete, or rename items. To use an item right-click the item and select from the menu or click and highlight the item and then click the item menu button shown below.



Default Workspace Tree



The **default workspace tree** has one schematics page, one equations page and one notes page. Workspace name can be changed using the **Save As** option from the main menu. Folder name can also be changed in the same way.
Very important icon is the **New items icon** which is used to add anything you wish to add to your workspace.

You can use the Workspace Tree toolbar to perform the following tasks:

Click this button	To do this
📄▾	Add a new item such as an analysis, design, or graph. Or, add an item from a library.
📂	Open the currently selected item.
🗔	Open the properties window for the currently selected item.
☰▾	Pull down the menu of the currently selected item.
▦▾	Pull down the Workspace Tree menu to adjust the Tree appearance by letting you show/hide datasets, change the sorting order, show additional information, etc.
📖	Get Help.

To add an item to the Workspace Tree:

1. Click the New Item button (📄▾) and select the item you want to add.
2. Type a name in the Name box.
3. Type a description in the Description box, if any.
4. Enter any other parameters in the properties window.
5. Click **OK**.

To delete an item from the Workspace Tree:

1. Right-click the item you want to delete and select **Delete** from the menu.
2. Click **Yes**.

To rename an item in the Workspace Tree:

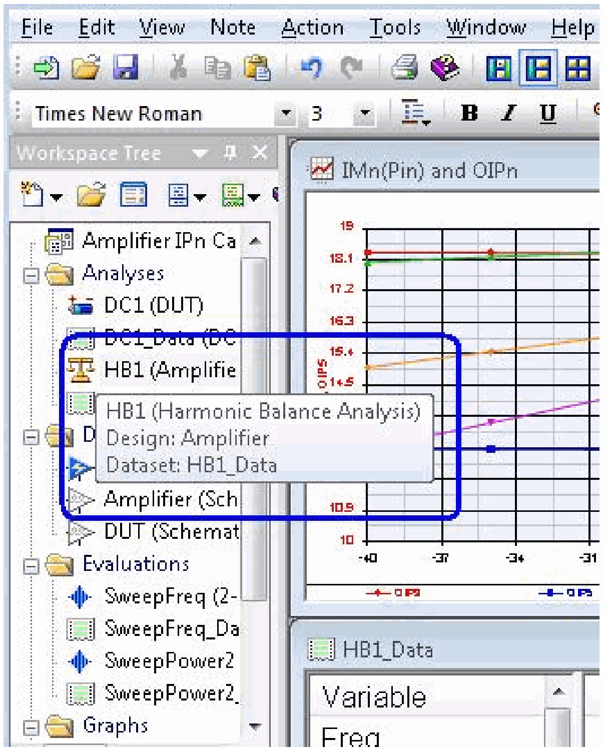
1. Right-click the item you want to rename and select **Rename** from the menu.
2. Delete the current name, and then type a new name in the box.
3. Click **OK**.

or slow double-click and type then click elsewhere when done

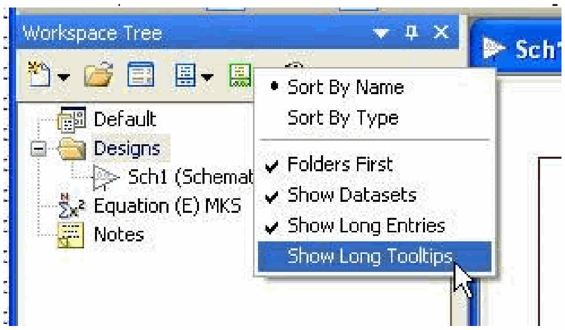
To copy an item to a library:

1. Right-click the item and select the Copy To sub-menu. Pick a library to copy to or use New Library to create a new library.

To Remove Information Pop Up Bubbles From Workspace Tree (shown below)



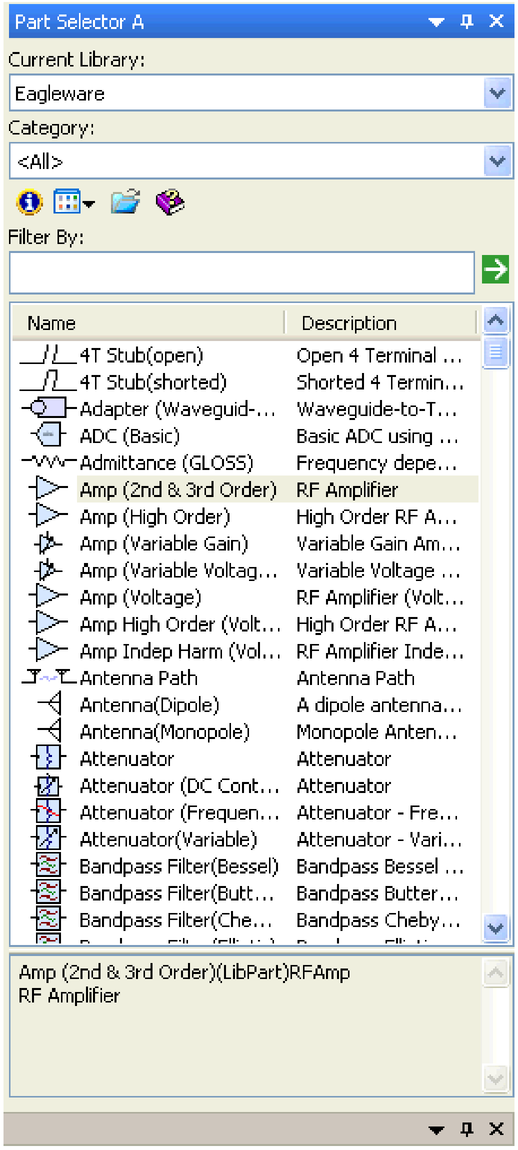
1. Click the **Options** button and uncheck *Show Long Tooltips*



Part Selector

The Part Selector is a toolbar that lets you add **parts** to a design. It displays a list of parts from the currently selected library. A library is a collection of objects that can be used in Genesys. The Part Selector only displays libraries of parts. The Library Selector is used to display libraries of other types. Eagleware is the default library. You can use the Category and Filter By features to display a subset of parts from the current library. When you select a part, detailed information about it is displayed in the information window at the bottom.

Genesys provides two part selectors: A and B. Part Selector A is the default, but you can display both part selectors at the same time. The options for viewing either part selector are found in the *View Menu* (users). Having both Part Selectors open lets you work with two libraries at once. Building a custom library of parts is easier with both Part Selectors open, because you can set one to view the custom library of parts as you build it.



Current Library: Eagleware

Category: <All>

Filter By:

Name	Description
4T Stub(open)	Open 4 Terminal ...
4T Stub(shorted)	Shorted 4 Termin...
Adapter (Waveguid-...	Waveguide-to-T...
ADC (Basic)	Basic ADC using ...
Admittance (GLOSS)	Frequency depe...
Amp (2nd & 3rd Order)	RF Amplifier
Amp (High Order)	High Order RF A...
Amp (Variable Gain)	Variable Gain Am...
Amp (Variable Voltag...	Variable Voltage ...
Amp (Voltage)	RF Amplifier (Volt...
Amp High Order (Volt...	High Order RF A...
Amp Indep Harm (Vol...	RF Amplifier Inde...
Antenna Path	Antenna Path
Antenna(Dipole)	A dipole antenna...
Antenna(Monopole)	Monopole Anten...
Attenuator	Attenuator
Attenuator (DC Cont...	Attenuator
Attenuator (Frequen...	Attenuator - Fre...
Attenuator(Variable)	Attenuator - Vari...
Bandpass Filter(Bessel)	Bandpass Bessel ...
Bandpass Filter(Butt...	Bandpass Butter...
Bandpass Filter(Che...	Bandpass Cheby...

Amp (2nd & 3rd Order)(LibPart)RFAmp
RF Amplifier

You can use the Part Selector toolbar to perform the following tasks.

Click this button	To do this
ℹ	Get reference information for the currently selected part.
▦▾	Select options to change the way parts display in the Part Selector window.
📂	Manage the part libraries. Click this button to open the Library Manager.
📘	Get online Help.

To place a part:

1. Click a part in the Part Selector list. Notice the part details that display in the information window.
2. Click in the Schematic window to place the part.

To view a subset of a part library:

1. Select a subset of parts to view from the Category list.

> ℹ **Note**
> The **All** category displays all available parts in the selected library.

To change part libraries:
1. Click the *Current Library* pull-down and select a library name to display all of the parts in that library.

To add a part library:
1. Click the Library Manager button (📂) and select Library Manager to get a dialog allowing you to add existing libraries.

To search for specific parts:
1. Type the text for the parts you want in the **Filter By** box. For example, type **cap** to get capacitors or any parts whose names or descriptions contains that text.
2. Click the Go button (➔) to display the parts in the Part Selector window.

To copy parts to a library:
1. Right-click the part you want to copy, and then select the name of a library from the Copy To menu. A copy of the part is automatically placed in the new library.

To change which columns are displayed:
1. Right-click the column heading and check on or off the columns you want to see. Click a column heading to sort by that column.

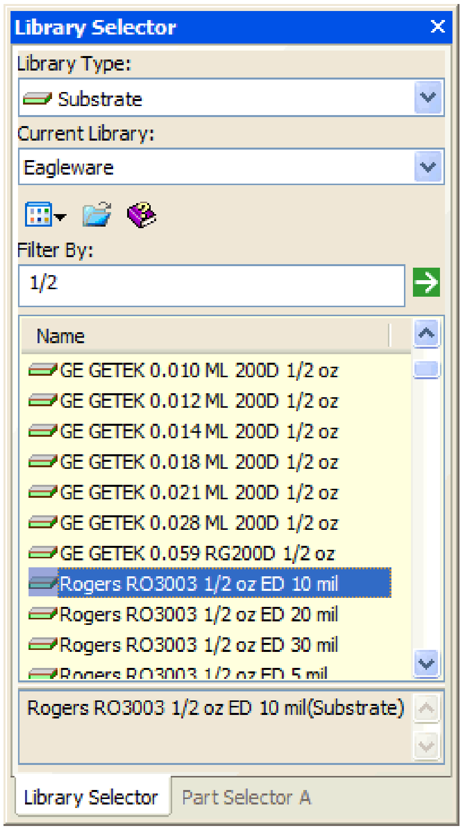
To change the way the selector shows parts:
1. Right-click the white area in the selector and select from the View sub-menu.

Library Selector

The Library Selector is a toolbar that gives you quick access to libraries of archived workspace items(datasets, designs, equations, substrates, etc.). The light-yellow background of the Library Selector distinguishes it from the Part Selector and other toolbars. It is used to display libraries of item types such as datasets, designs, or equations.

A library is a collection of one type of object found in Genesys. For example, a library of equations contains a collection of Equation Blocks, and only other Equations Blocks can be added to this library of equations. Schematic, Models, and Symbols are all considered to be a design, and so a library of designs can contain all three of these. Libraries of parts cannot be displayed in the Library Selector and must be viewed in the Part Selector.

The Library Selector operates for other objects much like the Part Selector operates for parts. The Library Type sets the type of object library you want to view and the Current Library sets to the particular library you want to display. The Filter By feature can be used to display a subset of the Current Library. When you select an item, detailed information about it is displayed in the information window at the bottom.



Library Selector

Library Type: Substrate

Current Library: Eagleware

Filter By: 1/2

Name

GE GETEK 0.010 ML 200D 1/2 oz
GE GETEK 0.012 ML 200D 1/2 oz
GE GETEK 0.014 ML 200D 1/2 oz
GE GETEK 0.018 ML 200D 1/2 oz
GE GETEK 0.021 ML 200D 1/2 oz
GE GETEK 0.028 ML 200D 1/2 oz
GE GETEK 0.059 RG200D 1/2 oz
Rogers RO3003 1/2 oz ED 10 mil
Rogers RO3003 1/2 oz ED 20 mil
Rogers RO3003 1/2 oz ED 30 mil
Rogers RO3003 1/2 oz ED 5 mil

Rogers RO3003 1/2 oz ED 10 mil(Substrate)

To edit a workspace item:
1. Double-click an item in the Library Selector list. The object is placed into your workspace and available for editing. Note that if this is a model or symbol you are currently using in your workspace then the in-workspace version of the model/symbol will override the library version.

✓ **Hint**
This is a great way to send self-contained workspaces to your coworkers, by embedding any custom models or symbols (or vendor models or symbols) into the workspace itself.

To set the library type:
1. Click the Library Type pulldown and select a library type to find.

To change libraries:
1. Click the Current Library pulldown and select a library to switch to. The Current Library pulldown only contains libraries of the type set in Library Type.

To search for specific objects:
1. Type the text for the items you want in the **Filter By** box. For example, in the library shown above type **1/2** to find the 1/2 oz substrates or any objects whose name or description contains that text. The filter is applied to the part name and description.

2. Click the Go button (➔) to display the updated list.

To copy an object into a library
1. Right-click the item in the workspace tree and select the **Copy To** menu to copy the object to a library.

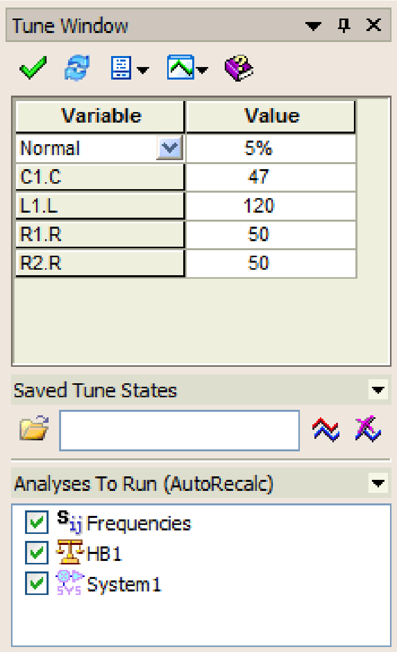
To change which columns are displayed:
1. Right-click the column heading and check on or off the columns you want to see. Click a column heading to sort by that column.

To change the way the selector shows parts:

1. Right-click the white area in the selector and select from the View submenu.

Tune Window

It is a tool used for *Tuning Variables* (users). It is one of the most powerful features of Genesys. You can use tuned variables(real time) almost anywhere in Genesys, including part parameters.



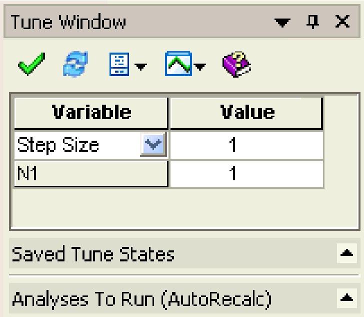
Variable	Value
Normal	5%
C1.C	47
L1.L	120
R1.R	50
R2.R	50

☑ Frequencies
☑ HB1
☑ System1

Any numeric parameter in a part can be made tunable. To get more information on how to tune variables, see *Tuning Variables* (users).

Tune Window Component	**Purpose**
✓ **Accept Tuned Settings**	Applies the current Tune settings to the graphs, etc.
Refresh	Scans for currently tunable variables
Variable Options	Sets Tune Window Variable settings
	- Hide Name Prefix - Omits the name prefix, so the name is as short as possible (overrides Long Names too). Duplicate variable names are common in this mode, which is confusing, so the recommended setting is OFF.
	- Long Names - Display the full name of the tunable variables
	- Select Variables - Displays a window which allows several variables to be selected at once.
Graph Checkpoints	Enables graph checkpoints
Help	Brings up help on tuning. (This page of documentation)
Variable Grid	Contains the tunable variables
	Variable Tuning Mode (dropdown)
	- **Normal** - tune (increment / decrement) by a percentage value, usually 5 or 10%
	- **Step Size** - tune by adding or subtracting the step size
	- **Standard** - Use Standard part values. (Limits tuning to specified "standard" values, which is useful for physical "lumped" parts i.e. resistors, capacitors, etc.)
Tuning Value	Amount to tune variables by (in conjunction with tuning mode)
	Variable - The name of a tunable variable, with optional info as set by the Item Menu above.
	Value - The value of a tunable variable. Click grid cell to activate tuning this variable.
Saved Tune States	Caches the current variable settings
	- Use These Settings - Opens saved settings
	- Settings Name - Name of the current settings
	- Checkpoint the Graphs - Places checkpoint traces on the graphs
	- Remove All Graph Checkpoints - Removes checkpoint traces from all the graphs (but does not delete named settings)
Analysis To Run (AutoRecalc)	Provides easy access to the Automatic Recalc settings of all the Analysis in your workspace
	- **Check** an analysis to enable its AutoRecalc mode, so that the analysis will run when a variable is tuned
	- **Uncheck** an analysis to disable its AutoRecalc setting, so the analysis will NOT automatically run

The Tune Window is collapsible so as to reduce screen clutter, the **Saved Tune States** and **Analysis To Run** panels can be hidden, via the "Fold" ▾ button on the right of each panels titlebar. (Click the "Unfold" ▴ button to restore the panels to full height.)



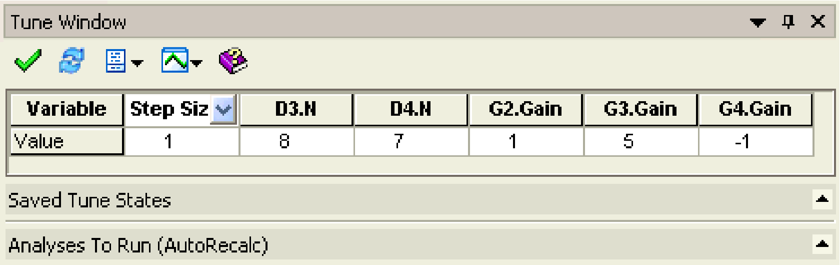
Tune Window

Variable	Value
Step Size	1
N1	1

Saved Tune States

Analyses To Run (AutoRecalc)

The Tune Window also has a horizontal display mode, which is automatically triggered when the Tune Window is wide:



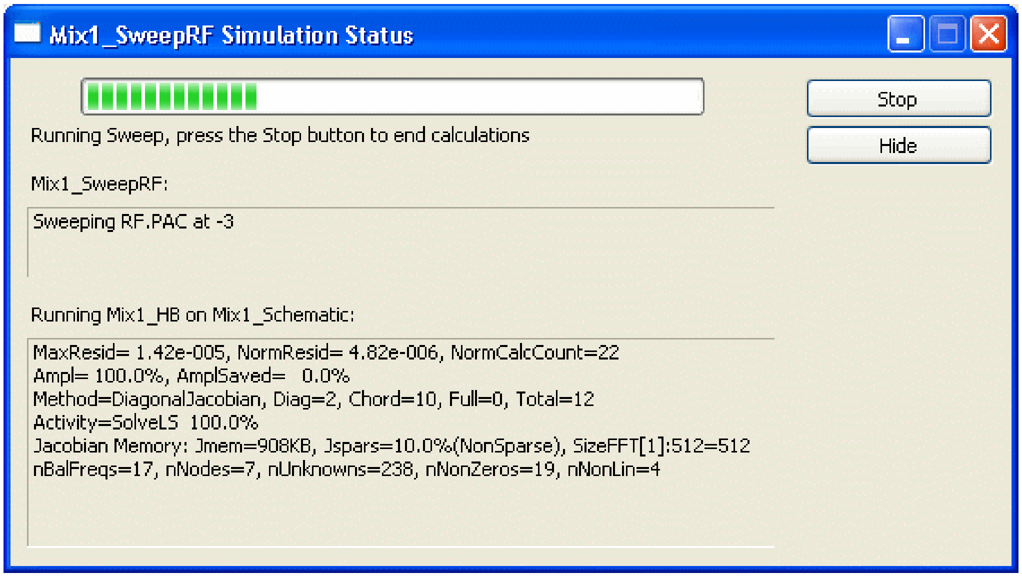
Variable	Step Siz	D3.N	D4.N	G2.Gain	G3.Gain	G4.Gain
Value	1	8	7	1	5	-1

If you are tuning more than one variable which have the same name, you may notice duplicate names in the list if you have selected the option to "Hide Variable Prefix" which shows shortened variable names. The "Hide Variable Prefix" option is available by clicking the **Variable Options** ▤▾ toolbar button.

Simulation Status Window

When a simulation is running, various output will be shown in this window, including the type of simulation being run and the status of the simulation. You can press the Stop button to stop the calculations at anytime. You can also press the Hide button to hide the status window, this will hide the status window and continue running the simulation. The details of the active simulation are shown in the main box of the simulation status window.

An example sweep



Running Sweep, press the Stop button to end calculations

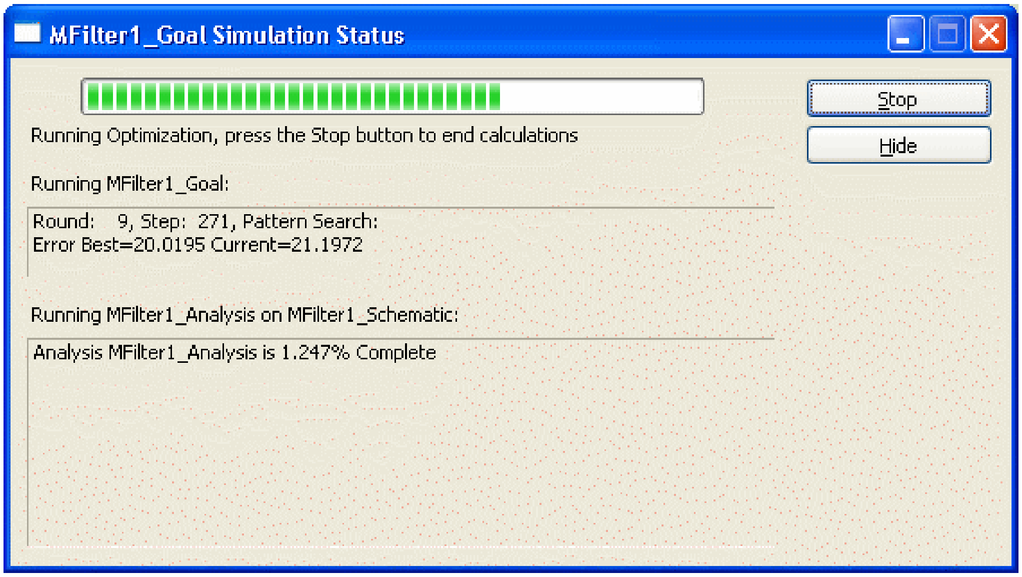
Mix1_SweepRF:

```
Sweeping RF.PAC at -3
```

Running Mix1_HB on Mix1_Schematic:

```
MaxResid= 1.42e-005, NormResid= 4.82e-006, NormCalcCount=22
Ampl= 100.0%, AmplSaved=  0.0%
Method=DiagonalJacobian, Diag=2, Chord=10, Full=0, Total=12
Activity=SolveLS  100.0%
Jacobian Memory: Jmem=908KB, Jspars=10.0%(NonSparse), SizeFFT[1]:512=512
nBalFreqs=17, nNodes=7, nUnknowns=238, nNonZeros=19, nNonLin=4
```

An example optimization



Running Optimization, press the Stop button to end calculations

Running MFilter1_Goal:

```
Round:   9, Step:  271, Pattern Search:
Error Best=20.0195 Current=21.1972
```

Running MFilter1_Analysis on MFilter1_Schematic:

Analysis MFilter1_Analysis is 1.247% Complete

Click the Stop button to stop the simulation run.

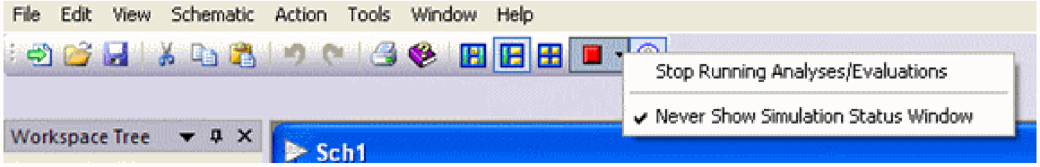
Click the Hide button to hide the status window but continue the simulation. There is also a Global option that always hides the status window. See the global options section.

Hiding the Simulation Status Window

The Hide button on the Simulation Status Window allows you to hide the currently running simulation's status window. There is also a global option that can be set to never show the simulation status window. There are two ways to turn the "Never Show Simulation Status Window" option on or off:

- Use the Global Options Page. See *General Global Options* (users) for information on making the setting this way.
- Use the toolbar start ▶ or stop ■ button drop down.

To set the "Never Show Simulation Status Window" option from the toolbar:

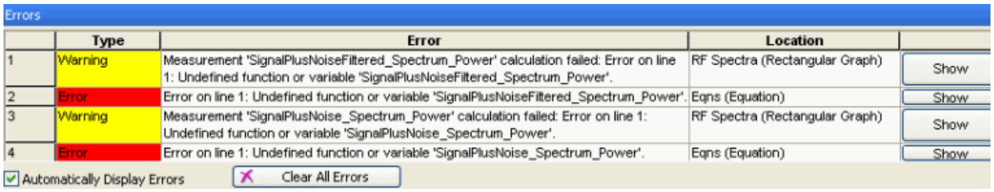

File
Edit
View
Schematic
Action
Tools
Window
Help
Stop Running Analyses/Evaluations
✓ Never Show Simulation Status Window
Workspace Tree
Sch1

1. Click the drop down arrow beside the start ▶ or stop ■ button on the main toolbar. The stop button will only be visible when a simulation is running.
2. Toggle the "Never Show Simulation Status Window" menu entry.

When toggled on from the toolbar when a simulation is running the status window will immediately be shown.

Error Log

The Error Log displays near the bottom of the Genesys window and alerts you to potential problems in your design. You can display the Error Log whenever you open Genesys. Or, you can have Genesys display the Errors window only when higher-level error messages are generated.



Errors

	Type	Error	Location	
1	Warning	Measurement 'SignalPlusNoiseFiltered_Spectrum_Power' calculation failed: Error on line 1: Undefined function or variable 'SignalPlusNoiseFiltered_Spectrum_Power'.	RF Spectra (Rectangular Graph)	Show
2	Error	Error on line 1: Undefined function or variable 'SignalPlusNoiseFiltered_Spectrum_Power'.	Eqns (Equation)	Show
3	Warning	Measurement 'SignalPlusNoise_Spectrum_Power' calculation failed: Error on line 1: Undefined function or variable 'SignalPlusNoise_Spectrum_Power'.	RF Spectra (Rectangular Graph)	Show
4	Error	Error on line 1: Undefined function or variable 'SignalPlusNoise_Spectrum_Power'.	Eqns (Equation)	Show

☑ Automatically Display Errors

click figure to enlarge

To open or close the Error Log:
1. Click **View** on the Genesys menu and select **Error Log**, *or*
2. If the window is open, click the close button (the x on the upper left) of the Error Log to close the window, *or*
3. Click the Errors Button ⓘ in the main toolbar.

To automatically display the Error Log for higher-level errors:
1. Click the **Automatically Display Errors** check box.

To clear out the messages in the error window
1. Click the **Clear All Errors** button.

Reviewing Error MessagesThe Error Log displays informational, warning, error, and critical messages. The messages are color-coded by message type.

- **Informational Message** – Green indicate a potential problem in your design.
- **Warning Message** – Yellow indicate a minor problem in your design.
- **Error Message** – Red indicate a problem in your design.
- **Critical Message** – Black indicate a critical problem in your design.

Messages always have a Show button. Click to bring up a schematic showing the highlighted error or a dialog showing the error line. If you have an undefined error, the Show button may do nothing.

More than one error message can come from the same part. Look at the last error in the list (the first to get thrown) to view the root error.

An instantiation error in a model during a simulation usually means that a parameter was bad (invalid or out of range). It might also imply the model couldn't be found or has changed since it was last used.

Using the Errors Window Button

You can also check for messages by viewing the Errors Window button on the Genesys toolbar. The color and image on the Errors Window button show the highest-level message in the Error Log.

Button	**Symbol**	**Meaning**
White	ⓘ	Indicates there is no message.
Green	?	Indicates an information message.
Yellow	⚠	Indicates a warning message.
Red	❗	Indicates an error message.
Black	❗	Indicates a critical message.

Using the Status Bar

The status bar is located at the bottom of the Genesys window.

Ready

It spans the width of the window and contains useful information or messages regarding your current task. If there is no information, the default message is **Ready**. When an action successfully completes, the default message is **Done**.

You should read the information in the status bar on a regular basis for assistance in using the program.

Design Windows

All working windows in the workspace area are called **Design Windows**.

To show or hide any of the windows:

- Click **View** on the Genesys menu and select the window.

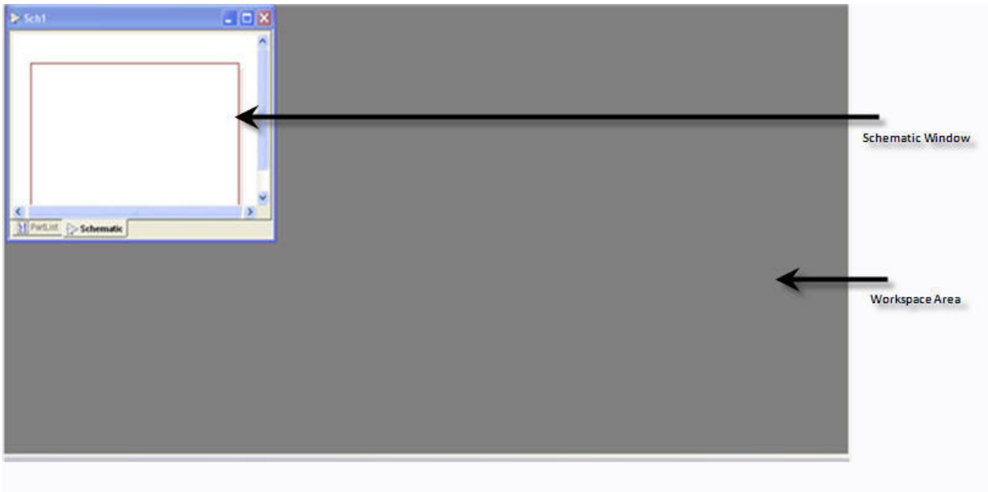
As you click inside each design window, the toolbar for that window will appear and may replace toolbars from the previous design window. You can move or dock toolbars anywhere in Genesys by grabbing the bar on the left and moving it (if docked) or grabbing the title bar (if floating). If a design window is active (selected) among several windows, its title bar becomes dark blue. The above examples show different toolbars to the left of the main toolbar.

If you re-size or maximize a window, the contents will grow (or shrink) adjusting to the new window size. Notes will reformat. Graphs print full-page and schematics print according to their defined physical sizes (using shrink to fit if the page will not fit on the paper).

Design windows are special in that they can show multiple views of itself, so you can see the partlist in one tab, the schematic in another and the layout in yet a third tab. Right click on the window and select the tab option to get another view of the design.

Workspace Area

The background of this area is gray and it holds any windows that are opened i.e. schematic window, graph, notes window,smith chart etc.



You can open as many windows as you want in this area. The default schematic menu icons are listed because that is the default window opened for any new workspace. This menu icons list is changed with the window that is selected. If you open up a graph window, the "schematics icons list" will give way to the "graph icons list".

Special Operations

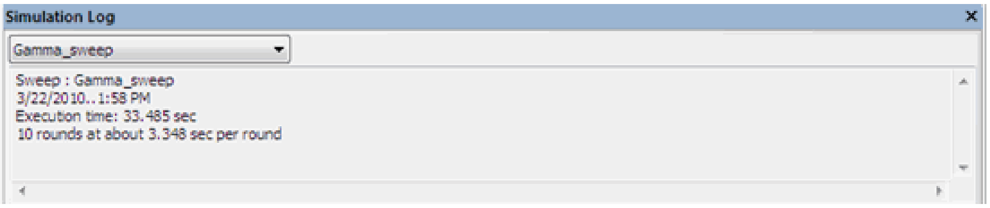
Several operations that you can work out on the windows in your workspace area:

1. Tile them vertically or horizontally
2. Close all the open windows
3. Make the windows opened as **tabbed**
4. Show/Hide all other docking windows (so that only the opened window is visible)
5. Show all output windows

Simulation Log

By default, the Simulation Log shows near the bottom of the Genesys window. However, it is a docking window that can float or be docked in the Genesys window.

The content of the simulation log will depend on the simulation or evaluation that is run. Each will show different information that ranges from a date and time run with execution time to an output for each frequency simulated.



```
Sweep : Gamma_sweep
3/22/2010..1:58 PM
Execution time: 33.485 sec
10 rounds at about 3.348 sec per round
```

To open or close the Simulation Log:
Click View on the Genesys menu and select Simulation Log

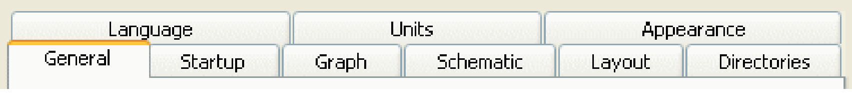
OR

If the window is open click the close button (the X on the upper left) of the Simulation Log to close the window.

To select the analysis or evaluation you want to view:
1. Click the pull-down and select the desired analysis or evaluation.

Setting Global Options for Genesys

Customize your working environment to best suit your needs using the global application options. You can set options for things such as how numbers are formatted for, which windows to display at startup, and which directories to use. The global options are saved when the application ends and are restored the next time you run it.

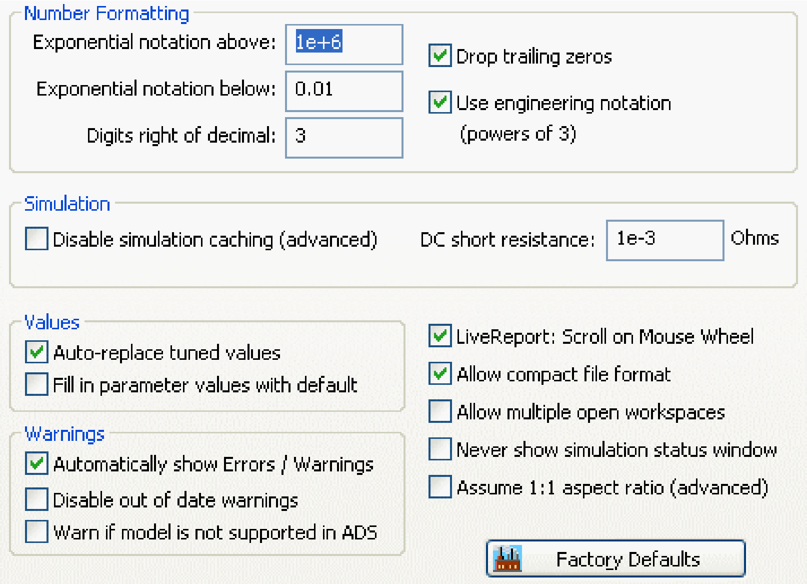


To set Global Options

1. Click **Tools** on the Genesys menu and select **Options**.
2. Click any of the following option tabs:
   General (users)
   Startup (users)
   Graph (users)
   Schematic (users)
   Layout (users)
   Directories (users)
   Language (users)
   Default Units (users)
   Appearance (users)
3. Select the options you want.
4. Click **OK**.

General Options Tab

Use the Global Options General window to select general environment options not specific to any one area of the program.



Number Formatting

Exponential notation above: 1e+6

Exponential notation below: 0.01

Digits right of decimal: 3

☑ Drop trailing zeros

☑ Use engineering notation (powers of 3)

Simulation

☐ Disable simulation caching (advanced)

DC short resistance: 1e-3 Ohms

Values

☑ Auto-replace tuned values

☐ Fill in parameter values with default

Warnings

☑ Automatically show Errors / Warnings

☐ Disable out of date warnings

☐ Warn if model is not supported in ADS

☑ LiveReport: Scroll on Mouse Wheel

☑ Allow compact file format

☐ Allow multiple open workspaces

☐ Never show simulation status window

☐ Assume 1:1 aspect ratio (advanced)

To change general global options:

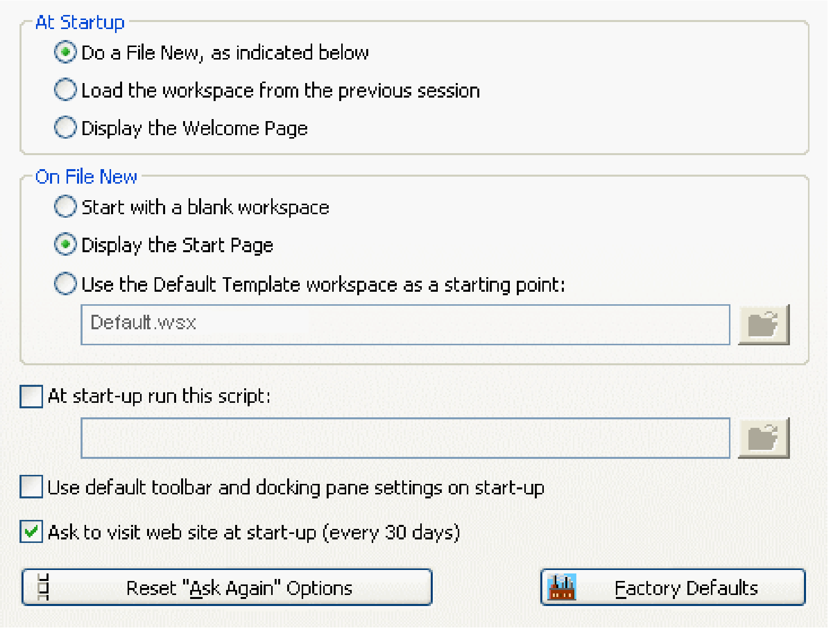
1. Click **Tools** on the menu and select **Options**.
2. Click the **General** tab.
3. Adjust the settings:

- **Number Formatting** – Specifies how the program should display numbers. This format is used uniformly throughout (tables, graph axes, dataset displays).
- **Simulation** – These settings control the simulation engines.
    - Disable simulation caching: Turns off caching of simulation data (runs slower when disabled).
    - DC short resistance: Sets the DC resistance of a short.
- **Values** – These settings control parameter values
    - Auto-replace tuned values keeps the tuned values up-to-date.
    - Fill in parameter values copies default values into blank parameter settings (instead of leaving them empty).
- **Warnings** – These settings control the display of Errors / Warnings
    - Automatically show: Instructs the program to show the Errors window when there are errors in the workspace and to hide the window when there are no errors remaining.
    - Disable out of date warnings turns off those warnings.
    - Warn if model is not supported in ADS: When checked, an error will be issued when a non-ADS model is placed on a schematic.
- **LiveReport: Scroll on Mouse Wheel** – Option to control the mouse wheel behavior on a Live Report. "Ctrl-Mouse Wheel" will zoom and "Shift-Mouse Wheel" pans right or left. If this is not checked, it will zoom on scroll wheel.
- **Allow compact file format** – Allows you to save compressed data files, which are not compatible with version 2005.11 and earlier.
- **Allow multiple open workspaces** – Allows more that one workspace (at a time) to be open, so that items may be easily copied from one workspace to another.
- **Never show simulation status window** – Never show simulation status window during simulations or evaluations.
- **Assume 1:1 aspect ratio** – Ignores incorrect video device information and assumes that the video display has square pixels. Enable this setting if Smith charts are oval, instead of circular.
- **Factory Defaults** – When clicked, this button resets all of the settings on this page of the dialog box.

4. Click **OK**.

Startup Options Tab

Use the Global Options Startup window to customize start up.



At Startup
● Do a File New, as indicated below
○ Load the workspace from the previous session
○ Display the Welcome Page

On File New
○ Start with a blank workspace
● Display the Start Page
○ Use the Default Template workspace as a starting point:
Default.wsx

□ At start-up run this script:

□ Use default toolbar and docking pane settings on start-up

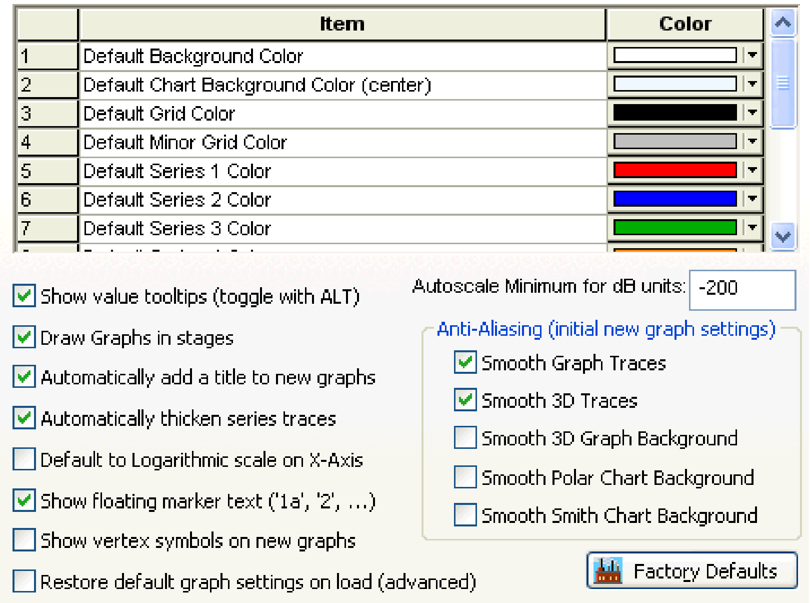
☑ Ask to visit web site at start-up (every 30 days)

To change the startup global options:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Startup** tab.
3. Adjust the settings:
    - **At Startup** – Specifies the action taken each time the program is run.
    - **On File New** – Specifies the action taken whenever a File / New action is initiated.
    - **At startup run this script** – Allows a custom startup action.
    - **Ask to visit web site at start-up** – Will cause a dialog box to be shown every 30 days asking if the user wants to check the web for updates.
    - **Use default toolbar settings on startup** – Forces the program to reinitialize the toolbars at startup.
    - **Factory Defaults** – When clicked, this button resets all of the settings on this page of the dialog box.
4. Click **OK**.

Graph Options Tab

Use the Global Options Graph window to set global (shared) options for graphs.



	Item	Color
1	Default Background Color	⬜
2	Default Chart Background Color (center)	⬜
3	Default Grid Color	⬛
4	Default Minor Grid Color	▨
5	Default Series 1 Color	🟥
6	Default Series 2 Color	🟦
7	Default Series 3 Color	🟩

☑ Show value tooltips (toggle with ALT)

☑ Draw Graphs in stages

☑ Automatically add a title to new graphs

☑ Automatically thicken series traces

□ Default to Logarithmic scale on X-Axis

☑ Show floating marker text ('1a', '2', ...)

□ Show vertex symbols on new graphs

□ Restore default graph settings on load (advanced)

Autoscale Minimum for dB units: -200

Anti-Aliasing (initial new graph settings)

☑ Smooth Graph Traces

☑ Smooth 3D Traces

□ Smooth 3D Graph Background

□ Smooth Polar Chart Background

□ Smooth Smith Chart Background

To change the graph global options:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Graph** tab.
3. Adjust the settings:
    - **Item colors** – These colors are used whenever a new graph or series is created. To apply these colors to an existing graph, right-click inside the graph window and select "Set All Colors To Defaults".
    - **Show value tooltips** – Shows the data value in a tool tip window when the cursor is placed over a trace data point.
    - **Draw Graphs in stages** – graphs can draw in stages. A simple graph is drawn first and details are progressively added. This will help with optimizations and sweeps where graphs redraw over and over.
    - **Automatically add a title** – Places a simple title at the top of each new graph.
    - **Automatically thicken series traces** – This setting will widen the lines used to

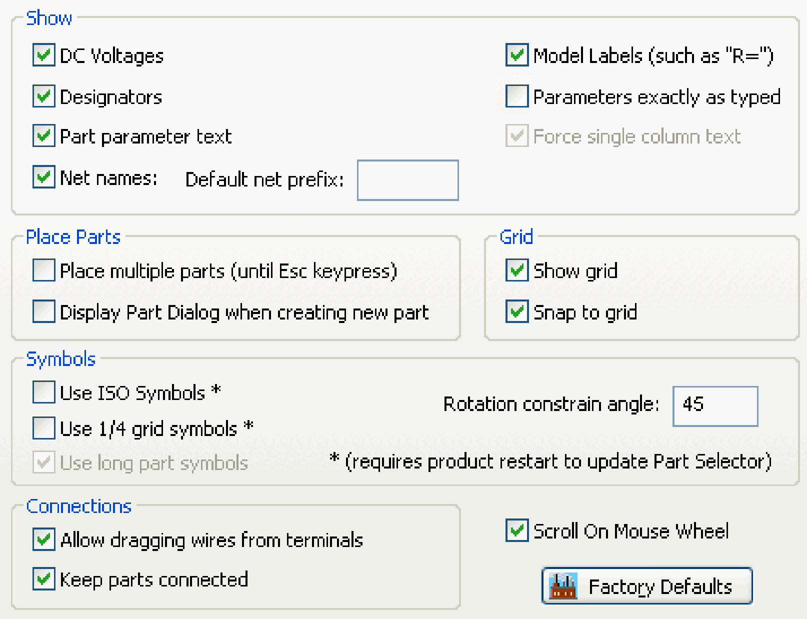
draw the series (trace) line, when a graph is fairly large.

- **Default to Logarithmic scale** – Switches the X-axis from linear to logarithmic scale.
- **Show floating marker text** – Enables short marker labels of the form '1a' or '2'. If not checked, no floating marker text will be shown, when graph markers are drawn in the margin on the right.
- **Show vertex symbols** – Marks series trace vertices with a dot or other symbol (to help distinguish traces on a black & white printout).
- **Restore default graph settings** – This option is rarely used, but can recover a graph from a damaged workspace file.
- **Autoscale Minumum** – The lower auto-scale boundry (prevents scaling all the way down to -600dB).
- **Anti-Aliasing** – These check-boxes enable a smoothing effect to be used when drawing graphs. This gets rid of the stair-stepped, jagged edges when graphs are drawn. When enabled, the graph is drawn with a slightly fuzzy look, which is actually sub-pixel accurate and can accentuate the slight ripples in a trace.
- **Factory Defaults** – When clicked, this button resets all of the settings on this page of the dialog box.

4. Click **OK**.

Schematic Options Tab

Use the Global Options Schematic window to set options for all schematics.



Show
☑ DC Voltages
☑ Model Labels (such as "R=")
☑ Designators
☐ Parameters exactly as typed
☑ Part parameter text
☑ Force single column text
☑ Net names: Default net prefix:

Place Parts
☐ Place multiple parts (until Esc keypress)
☐ Display Part Dialog when creating new part

Grid
☑ Show grid
☑ Snap to grid

Symbols
☐ Use ISO Symbols *
☐ Use 1/4 grid symbols *
☑ Use long part symbols
Rotation constrain angle: 45
* (requires product restart to update Part Selector)

Connections
☑ Allow dragging wires from terminals
☑ Keep parts connected

☑ Scroll On Mouse Wheel

To change the schematic global options:

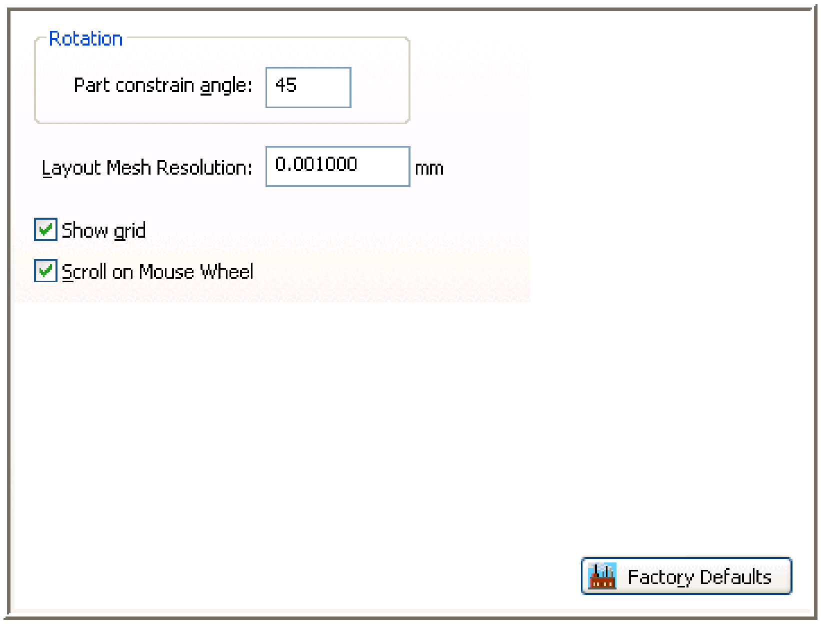
1. Click **Tools** on the menu and select **Options**.
2. Click the **Schematic** tab.
3. Adjust the settings:
   - **Show** – DC Voltages, Designators, Part Parameter Text, etc. Check to enable the specified information to be displayed on a schematic.
   - **Place Parts** – Multiple parts allows parts to be placed each time you left-click on the schematic. Press the Esc key to stop dropping parts. Display Part Dialog will bring up the part dialog each time a part is placed on a schematic, so that the parameters may be entered.
   - **Grid** – Show the background grid and snap the mouse cursor to the grid (if enabled).

- **Symbols** – Use ISO symbols: When checked, ISO standard symbols will be placed. (The ISO standard resistor is a box, instead of a zigzag.) Use ¼ grid symbols: When checked, it will place ADS-compatible parts that have terminals spaced on ¼ of the standard part length (which i the length of a resistor). standard parts are on a 1/6th grid spacing. These settings will not take full effect until you have exited and restarted. Rotation constrain angle: Sets the F3-key rotation increment (usually 45 or 90 degrees).
- **Connections** – Allow dragging wires enables schematic parts to be easily connected; just place the mouse cursor over a part terminal, press the left-button, and drag the newly-created connector to another node. Keep parts connected ensures that schematic parts retain their electrical connections, by inserting new wires (as necessary) when dragging parts. The Alt-key acts as a toggle for the keep connect setting.
- **Scroll On Mouse Wheel** – When checked, the mouse center wheel scrolls the schematic window; when unchecked, the wheel zooms the window instead.
- **Factory Defaults** – When clicked, this button resets all of the settings on this page of the dialog box.

4. Click **OK**.

Layout Options Tab

Use the Global Layout Options window to specify options for every layout you create.



Rotation

Part constrain angle: 45

Layout Mesh Resolution: 0.001000 mm

☑ Show grid

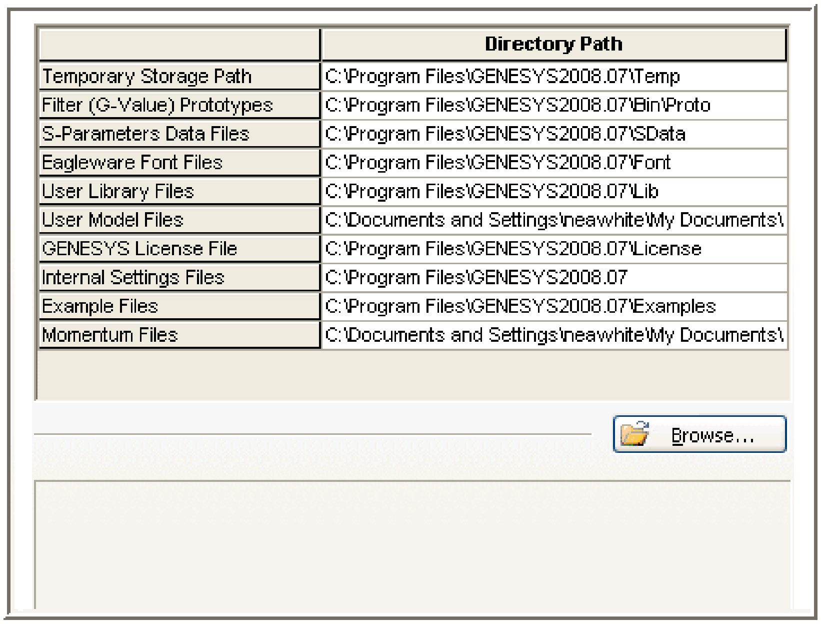
☑ Scroll on Mouse Wheel

To change the layout global options:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Layout** tab.
3. Adjust the settings:
    - **Rotation / Part Constrain Angle** – Specifies the allowable incrementatal rotations (usually 45 or 90) for parts on a layout.
    - **Layout Mesh Resolution** – The default mesh resolution, which is used by MomentumGX.
    - **Show Grid** – When checked, a grid will be displayed in all layout windows.
    - **Scroll On Mouse Wheel** – When checked, the mouse center wheel scrolls the layout window; when unchecked, the wheel zooms the window instead.
4. Click **OK**.

Directories Options Tab

Use the global options directories window to set the default directory paths.



	Directory Path
Temporary Storage Path	C:\Program Files\GENESYS2008.07\Temp
Filter (G-Value) Prototypes	C:\Program Files\GENESYS2008.07\Bin\Proto
S-Parameters Data Files	C:\Program Files\GENESYS2008.07\SData
Eagleware Font Files	C:\Program Files\GENESYS2008.07\Font
User Library Files	C:\Program Files\GENESYS2008.07\Lib
User Model Files	C:\Documents and Settings\neawhite\My Documents\
GENESYS License File	C:\Program Files\GENESYS2008.07\License
Internal Settings Files	C:\Program Files\GENESYS2008.07
Example Files	C:\Program Files\GENESYS2008.07\Examples
Momentum Files	C:\Documents and Settings\neawhite\My Documents\

To set the global directory paths:

1. Click **Tools** on the menu and Select **Options.**
2. Click the **Directories** tab.
3. Click on **Directory Path** or label to see a description of what it's used for.

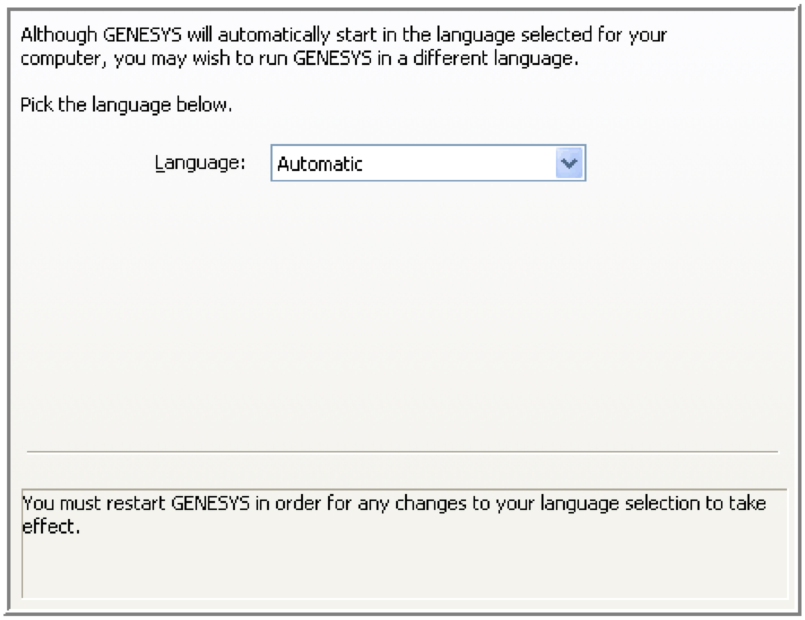
To change a path:

1. Click a **Directory Path**
2. Click **Browse**
3. Select the correct path
4. Click **OK**

Note
You can edit the path directly.

Language Options Tab

Use the Global Options Language window to select a different language in which to run. The default language is pre-selected for your computer and is listed as Automatic in this window. Other choices include Chinese, Korean, and Japanese. You must restart your computer before any language changes can take effect.



Although GENESYS will automatically start in the language selected for your computer, you may wish to run GENESYS in a different language.

Pick the language below.

Language: Automatic

You must restart GENESYS in order for any changes to your language selection to take effect.

To change the language global options:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Language** tab.
3. Select a language from the Language list.
4. Click **OK**.

Default Units Options Tab

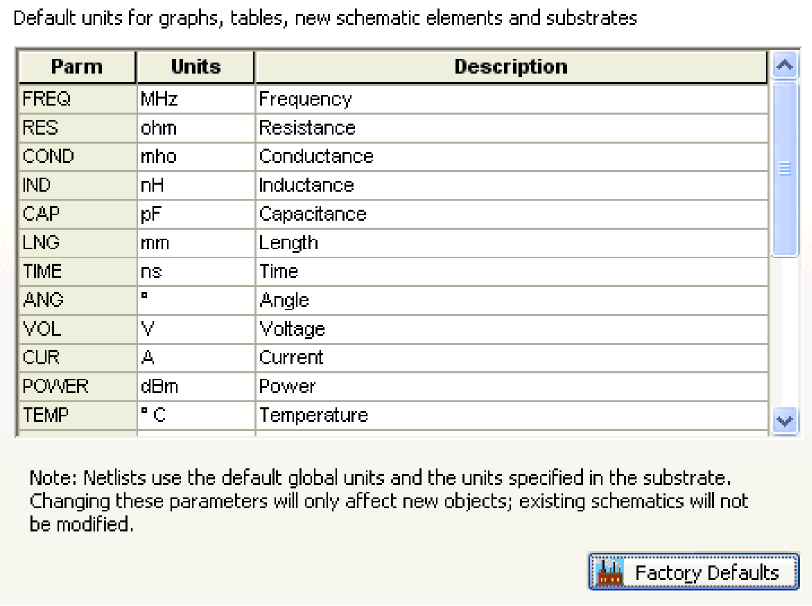
Use the Global Options Units window to make global changes to the default units in a schematic. Changing the default units has no bearing on any of the parts that are in the schematic. Only the initial units of parts placed after the default unit changes are affected.

Note
Physical length is unique. For layout dimensions, the units are specified in the General tab of the Layout Properties window for each layout.

The global default units used are listed in the table below.

Quantity	**Units**
Angle	Degrees
Capacitance	pF (picofarads)
Conductance	mhos (1/ohms or Siemens)
Current	Amps
Frequency	MHz (Megahertz)
Inductance	nH (nanohenries)
Physical Length, Width, Height	mm (millimeters), or based on substrate for netlist
Power	dBm (referenced to a milliwatt)
Resistance	ohms
Temperature	C (Celsius)
Time	ns (nanoseconds)
Voltage	V (volts)

Note
Additional default units are available for non-linear models. They all begin with NL_. The default units for these parameters are the industry standard units, and you should not need to change them.



Default units for graphs, tables, new schematic elements and substrates

Parm	Units	Description
FREQ	MHz	Frequency
RES	ohm	Resistance
COND	mho	Conductance
IND	nH	Inductance
CAP	pF	Capacitance
LNG	mm	Length
TIME	ns	Time
ANG	°	Angle
VOL	V	Voltage
CUR	A	Current
POWER	dBm	Power
TEMP	° C	Temperature

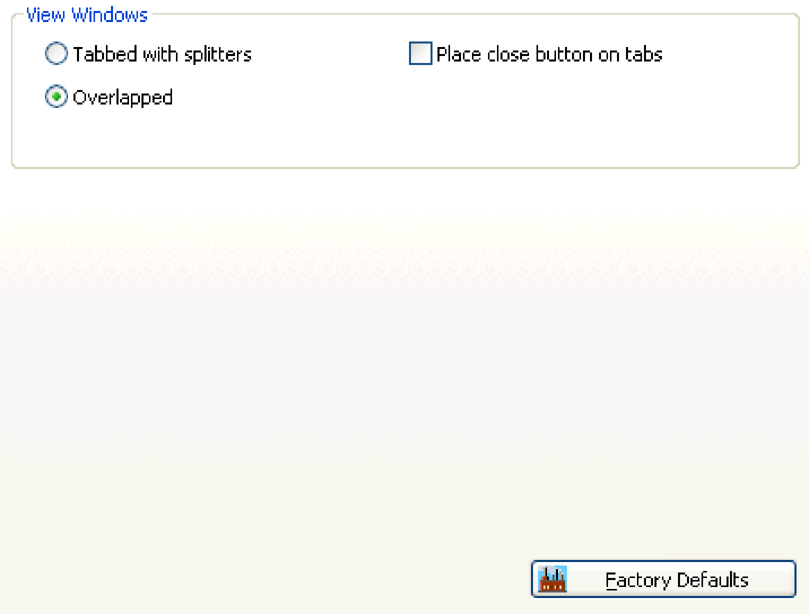
Note: Netlists use the default global units and the units specified in the substrate. Changing these parameters will only affect new objects; existing schematics will not be modified.

To change the global default units:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Units** tab.
3. Change the units you want by clicking the **Units** grid cell next the parameter type and selecting the desired unit from the pop-up combo box.
4. Click **OK**.

Appearance Options Tab

Use the appearance options window to see the default directory paths.



View Windows

○ Tabbed with splitters

● Overlapped

□ Place close button on tabs

To change the appearance global options:

1. Click **Tools** on the menu and select **Options**.
2. Click the **Appearance** tab.
3. Adjust the settings:
    - **Tabbed with splitters** – Specifies the use of tabbed view windows, with splitter bars.
    - **Overlapped** – Specifies the use of multiple document interface (MDI) overlapping windows.
    - **Place close button on tabs** – When checked, the tab close button will be placed on the tab button itself (instead of being placed on the right).
    - **Factory Defaults** – Restores the original factory values to these settings.
4. Click **OK**.

Using Genesys

Contents

Analysis

Circuits and systems can be analyzed in many different ways. When you simulate a circuit, the settings for the analysis determine how the simulation runs. The analysis creates a dataset with the simulation results. If an analysis is set to *automatically recalculate*, it will re-simulate each time you make a change to the schematic design and then click a graph or table dependent on the analysis.

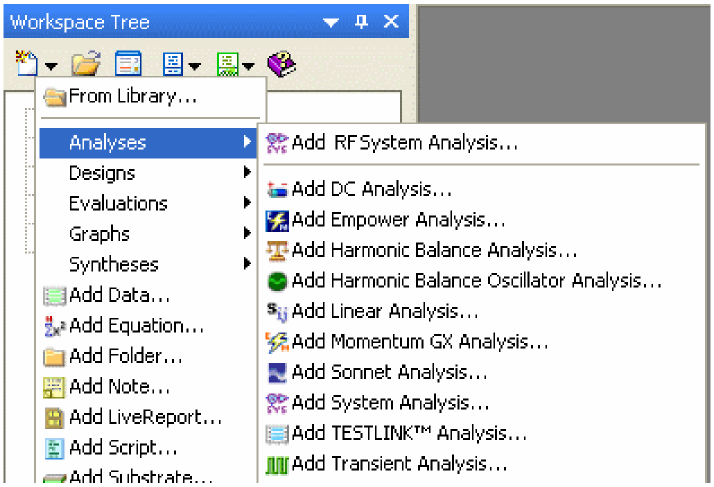
Genesys provides the following analysis engines:

- *DC Analysis* (sim) - Determines the circuit DC operating point.
- *Harbec Harmonic Balance Analysis* (sim) - Checks the steady-state performance of nonlinear circuits and oscillators with excellent frequency resolution.
- *Linear Analysis* (sim) - Calculates S-parameters and noise parameters of a circuit, substituting resistive ports for sources.
- *Empower Planar 3D EM Analysis* (sim) - Performs a linear analysis based on a layout using an electromagnetic simulation.
- *RF Design Kit Spectrasys* (sim) - Performs a system-block-level non-linear analysis on the entire system to determine if all system-level requirements are met.
- *Testlink* (users) - Imports data from instruments to allow measurement comparison with models used to develop the network being tested.
- *Cayenne Transient Analysis* (sim) - Calculates a circuit response using the SPICE time domain method.

For more info on theses analyses see the *Getting Started with Genesys Simulation* (sim)

To add an analysis

1. Click the **New Item button** ( ) on the Workspace Tree toolbar and select an analysis from the **Analyses** menu. A new analysis of the selected type will be created.



2. Fill in the desired analysis parameters.
3. When you click **OK or Calculate**, the analysis will run and create a data set.

Annotations

Annotations include text boxes, arrows, shapes, and controls (widgets) that can be placed on a schematic, graph, or LiveReport to help document a workspace, highlight items of interest, etc.

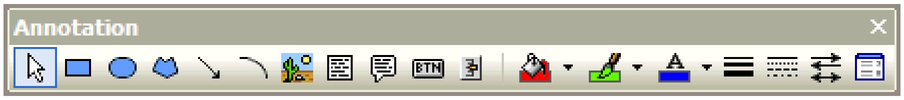
Tools	**Purpose**
Rectangle	Draw a square or rectangle.
Ellipse	Draw a circle or ellipse
Polygon	Draws a filled polygon or unfilled polyline.
Arrow/Line	Draw a line or arrow. Change the arrow style by selecting a line and picking an arrow type from Arrows button menu.
Arc	Draw a circular arc.
Picture	Insert a picture. Use this annotation to add a company logo to a graph, for example. Double-click the new object and select a JPG, GIF, or BMP image file to be displayed. (To allow all users to see the image, the bitmap file should reside on a network server.)
Text	Place text. Text has a number of settings. Double-click a text annotation to set the horizontal and vertical justification (text alignment). The name of the text item can be changed and shown on-screen, which simplifies building a schematic title block.
Text Balloon	Draw a text balloon. This annotation has a "tail" which can be anchored to a data point on a graph, to the page, or not anchored (using the right-button menu).
Button	Draw a user button. This annotation can be "clicked" to run a custom script, which is specified by double-clicking the outer EDGE of the button control. (The middle of the button runs the script.)
Slider	Draw a slider control. This annotation is linked to a tunable parameter and functions much like the *Tuning Window* (users).

Settings	**Purpose**
Fill Color	Sets the annotation fill color. Use the 3 color buttons to change the colors of the selected annotation(s). New annotations will be created using the current colors. The bottom-right color swatch (with a diagonal slash) is transparent, which specifies an unfilled object.
Line Color	Sets the annotation line / border color. The bottom-right color swatch (with a diagonal slash) is transparent, which specifies a object with no outline.
Text Color	Sets the annotation text color.
Line Thickness	Set the width of borders and lines.
Line Style	Set the drawing style of borders and lines (dash pattern, etc.).
Arrows	Set the arrow style of lines.
Properties	Display the properties window for the selected annotation.

Contents

Creating Annotations

The Annotation button on the Schematic, LiveReport, and Graph toolbars toggles the display of the Annotation toolbar.



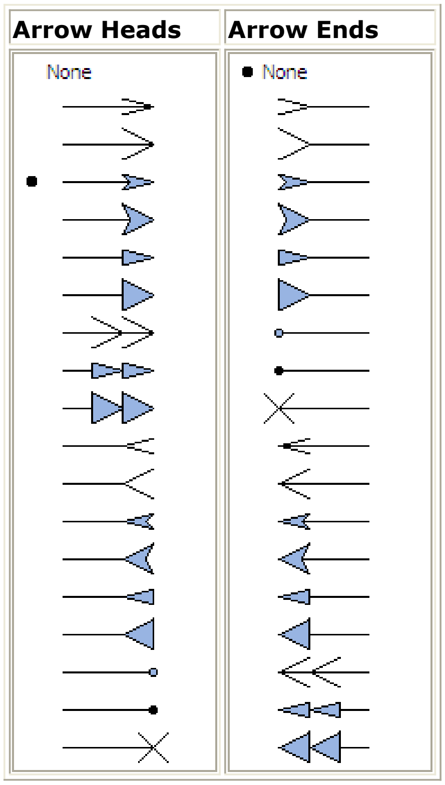
The toolbar provides tools like lines, circles, and text that you can use to point out details of interest on a schematic, draw a box around a group of components, etc.

To place an annotation:

1. Click the various settings buttons (colors, line style, etc.) to adjust the settings for newly created annotations
2. Click an annotation tool button (box, arc, text, etc.) on the Annotation Toolbar.
3. Click in a schematic, LiveReport, or graph window to place the new annotation.
4. Use the annotation setting buttons to change existing, selected annotations. (More than 1 annotation can be adjusted at a time).
5. To set the Font for annotations with text, right-click the object and pick **Font...** from the pop-up menu.

Line Annotations

Lines have many drawing options: Line Thickness, style, color, arrowheads, etc., which are controlled via the Annotation toolbar and by the object's right button menu. Lines can have arrowheads and ends. Simply select a line and pick an arrow type:

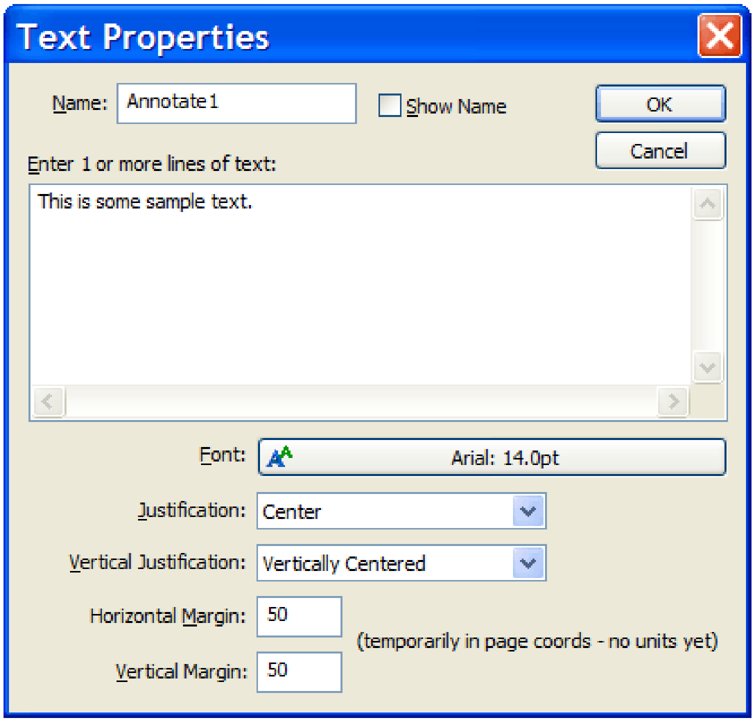


Text Annotations

A text annotation is a filled rectangular box with text inside.

To change the properties of a text annotation:

1. Double-click any text object.
2. Make the changes you want.
3. Click **OK**.



Name: Annotate1

□ Show Name

Enter 1 or more lines of text:

This is some sample text.

Font: Arial: 14.0pt

Justification: Center

Vertical Justification: Vertically Centered

Horizontal Margin: 50

Vertical Margin: 50

(temporarily in page coords - no units yet)

Property	**Purpose**
Name	The name of the Text object.
Show Name	Displays the name of the text item, which simplifies building a adding a title block or other "labeled text".
Enter Lines of Text	Specifies the test to be displayed.
Font	Click the button to set the font.
Justification	Sets the horizontal justification (alignment) of the text: Left, Right, or Center.
Vertical Justification	Sets the vertical alignment of the text: Top, Bottom, or Vertically Centered.
Horizontal and Vertical Margins	Sets the margins (border gap) of the text. Specified in page coordinates (1/1000ths of an inch).

Tips for advanced users

Text annotations can use equations. For example, if your workspace contains an equation block with a text variable named CompanyName, you can place =CompanyName in the Text field. (The leading = sign indicates that the text string is actually an expression.) When the annotation is drawn, the equation will be evaluated and the result displayed.

Text annotations can display model and parameter info when used *within a* ***custom symbol*** . This is implemented via macro-text-substitution. When symbol text is drawn on a schematic, the displayed text is modified prior to output. For example, Name=%Model% would be displayed as "Name=Resistor" on a symbol using a resistor model. The recognized macro strings are:

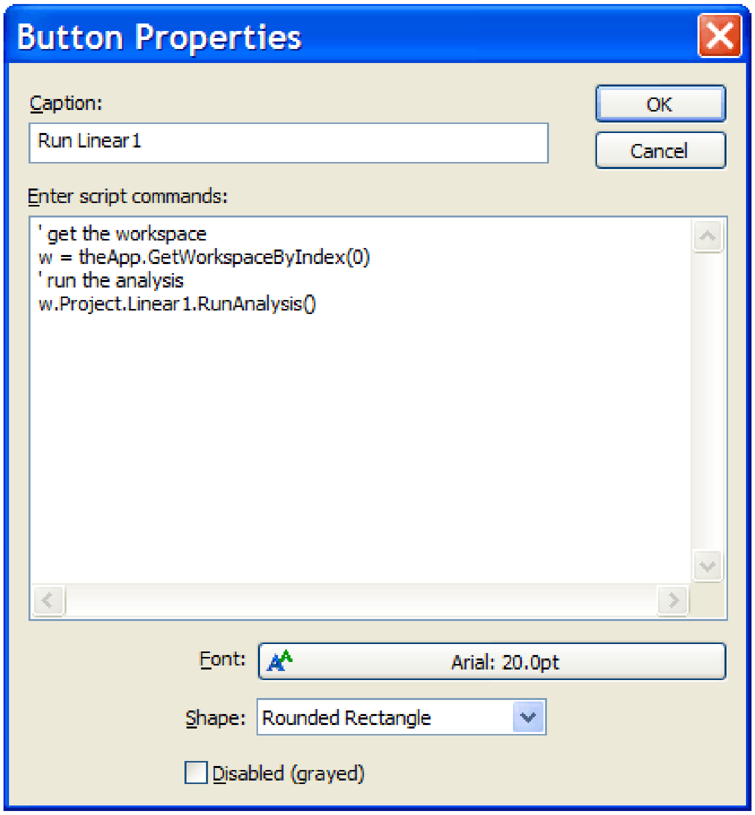
1. **%Des%** - Displays the part's designator.
2. **%Model%** - Displays the name of the model attached to the part.
3. **%MODEL%** - Displays the model name in UPPERCASE.
4. **%ParameterName%** - Displays the value of the specified model parameter attached to the part. E.g. R, C, L, QL, MODE, etc.

Button Annotations (Widgets)

A button annotation is a control which runs a script when clicked. Buttons and other widgets are initially created using "stock Windows colors"; the controls' colors can easily be changed using the Annotation toolbar, as can line thickness, etc.

To change the properties of a button:

1. Double-click the *EDGE* of any button object.
2. Make the changes you want.
3. Click **OK**.



Caption: Run Linear1

Enter script commands:

```
' get the workspace
w = theApp.GetWorkspaceByIndex(0)
' run the analysis
w.Project.Linear1.RunAnalysis()
```

Font: Arial: 20.0pt

Shape: Rounded Rectangle

□ Disabled (grayed)

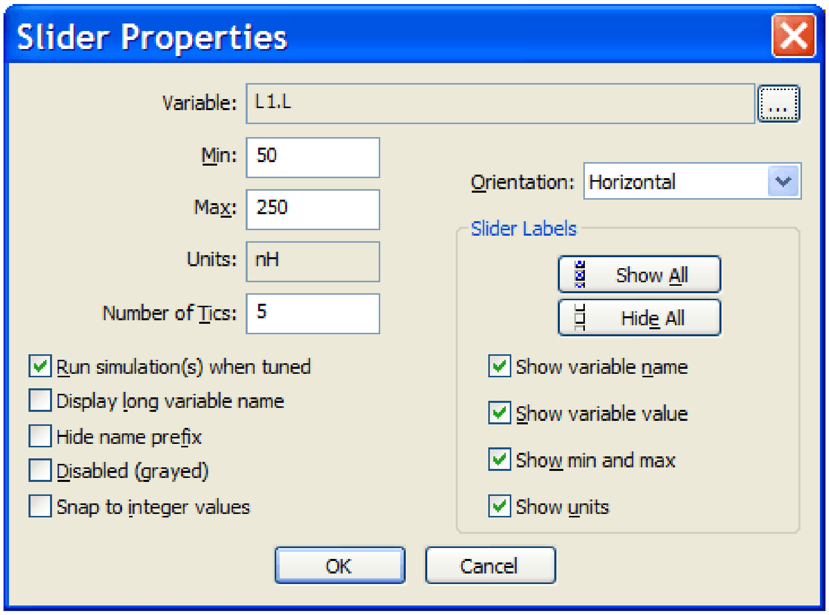
Property	**Purpose**
Caption	The title text displayed on the button.
Script Commands	Specifies the script to be run, when the button is clicked.
Font	To set the font.
Shape	Buttons can be a rectangle, a rounded rectangle, or an ellipse.
Disabled	Grays and inactivates the button.

Slider Annotations (Widgets)

A slider annotation is a control which adjusts a tunable equation variable, part parameter, etc. Sliders and other widgets are initially created using "stock Windows colors"; the controls' colors can easily be changed using the Annotation toolbar, as can line thickness, etc.

To change the properties of a slider:

1. Double-click the EDGE of any slider object.
2. Make the changes you want.
3. Click **OK**.



Slider Properties

Variable: L1.L

Min: 50

Max: 250

Units: nH

Number of Tics: 5

☑ Run simulation(s) when tuned

□ Display long variable name

□ Hide name prefix

□ Disabled (grayed)

□ Snap to integer values

Orientation: Horizontal

Slider Labels

☑ Show variable name

☑ Show variable value

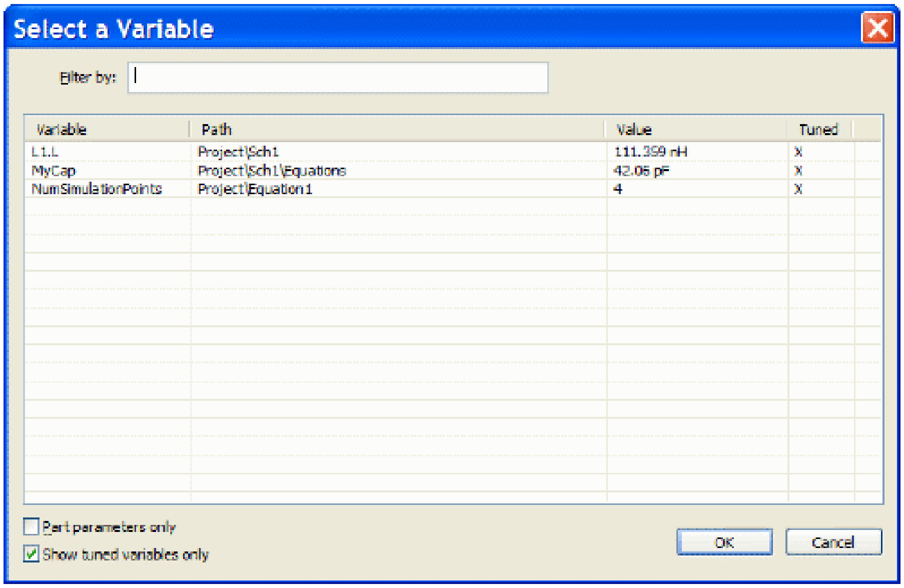
☑ Show min and max

☑ Show units

Property	Purpose
Variable	The variable to be tuned.
'...' Button	Brings up a selector to pick the variable.
Min and Max	Specifies the limits of the tuning range. These can be set to the names of equation variables, for adjustable limits.
Units	Displays the units of the tune variable.
Number of Tics	The number of slider division marks.
Orientation	Sliders can be horizontal or vertical.
Slider Labels	Specifies what text (if any) to display on a slider.
Show / Hide All	Check or uncheck all the Show checkboxes.
Run simulations	Runs enabled simulations on left-button up.
Display long variable name	Displays the tune variable's long name (Eg: Project\Sch1\L1.L).
Hide name prefix	Omits the part or equtaion name (Eg: L instead of L1.L).
Disabled	Grays and inactivates the slider.
Snap to integer values	Limits the tuning to integer values.

Variable Selector

This is displayed via the **'...'** button.



Variable	Path	Value	Tuned
L1.L	Project\Sch1	111.399 nH	X
MyCap	Project\Sch1\Equations	42.06 pF	X
NumSimulationPoints	Project\Equation1	4	X

□ Part parameters only

☑ Show tuned variables only

Property	Purpose
Filter by	Limits the variables displayed to only those that include the specified text.
Variable	Displays the variable's name.
Path	Displays the full pathname of the variable.
Value	Displays the current value of the variable.
Tuned	Displays an X, if the variable is tunable.
Part parameters only	Limits the variables displayed to only part parameters.
Show tuned variables only	Limits the variables displayed to only tunable variables.

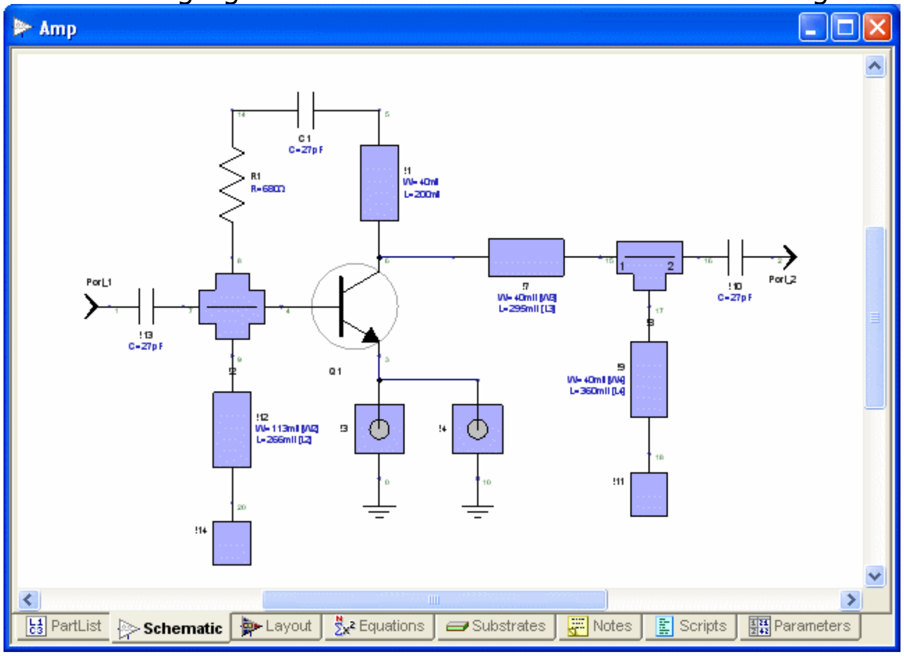
Designs

A design is an abstract term used to define a collection of related items that fully characterize a simulatable circuit. A design generally contains a schematic and parts list. However, notes, equations, scripts, user defined parameters, substrate, and a layout can also be added to any given design. The schematic is a visual representation of the parts being simulated and their connectivity with each other. The schematic is generally the heart of the design. Each tab at the bottom of the design window contains its characteristics that complement and influence the design. Many of these tabs and their added affects are optional. A design is the generic simulatable object. A schematic is a design as well as a schematic symbol, model, user model, layout, footprint, sub-circuit, etc. Designs are often contained in designs. The most common names for a design are schematics, models, or parts. Throughout the documentation the term **Schematic** will be generally synonymous with **Design**.

Designs contain the following characteristics:

- Parts List (this is required)
- Schematic (optional but is generally the preferred method of entering part connectivity)
- Notes (optional)
- Equations (optional)
- Scripts (optional)
- Parameters (optional)
- Substrate (optional)
- Layout(optional)

The following figure has all attributes available to a design:



Specific Types of Designs

A design is an abstract term used to define a collection of related items that fully characterize a simulatable circuit. Specific types of designs contain a specific set of these related items. All of the following items are specific types of designs:

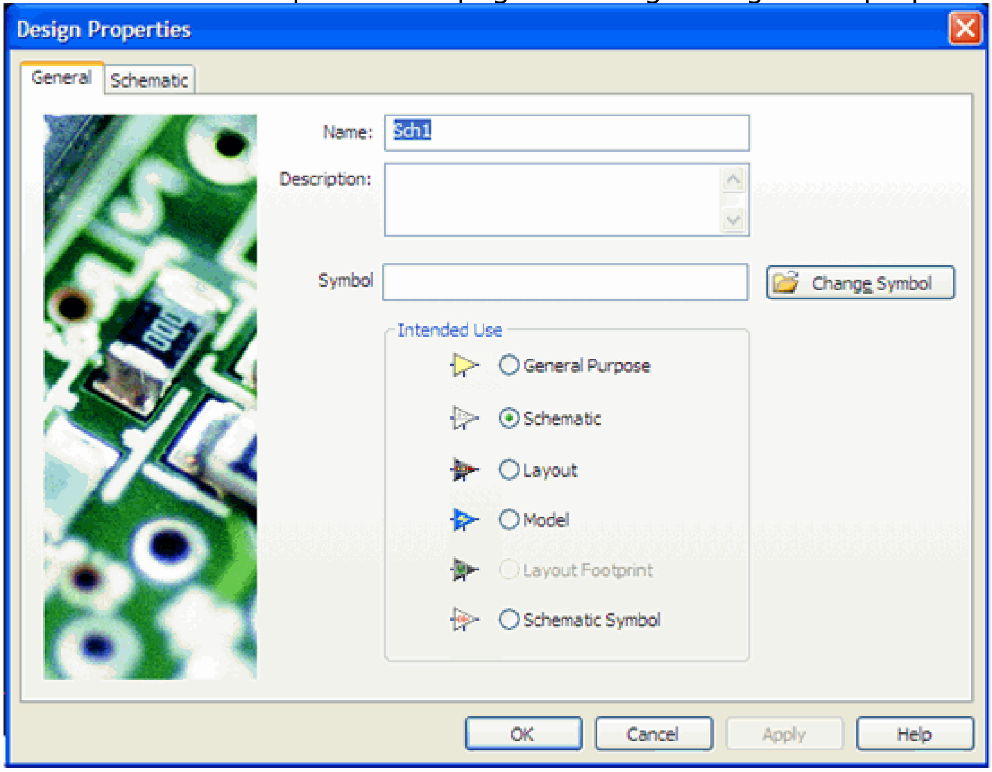
- Schematic
- Schematic Symbol
- User Model
- Footprint
- Layout

Contents

Design Properties

General Tab

Use the General Properties tab page to change the general properties of a Design.



Name: Sch1

Description:

Symbol

Intended Use

○ General Purpose

● Schematic

○ Layout

○ Model

○ Layout Footprint

○ Schematic Symbol

- **Name** - The name of the Design.
- **Description** - The Design description (optional).
- **Intended Use** - What kind of Design is it? This setting controls the Design's icon on the workspace tree and Genesys' interpretation of how the design is intended to be used. (If it's a symbol, you can select it for a schematic part's symbol, if it's a model, you will be able to select it as a model, etc.)

Modifying a Design

The following attributes can be added to a design:

- Notes
- Equations
- Scripts
- Parameters

- Substrate
- Layout

  Here is a brief overview of these attributes.

Notes

A note can be added to help document different aspects of the design.

Equations

An equation block can be added to a design so that variables can be based on equations. These variables can be used for various design parameters. An equation block is generally the link between the parameters a user would see and the parameters used in models.

> **Note**
> These equations are local to the design. Other parts of your workspace cannot access the variables in this equation set.

For more information see *Equations* (users).

Scripts

Scripts control Genesys operations. Add a script to your design to load files, save files, save data sets, and change object parameters.

> **Note**
> To run this script, copy the text and paste it into the Script Processor, then select Run.

For more information see *Running Scripts* (users).

Parameters

Parameters are added to a design when the implementation details are generally hidden from the user as is the case of a user model. When a design contains parameters this design can be used as a user model and these parameters will be exposed as model parameters in the part that uses this design as its model.

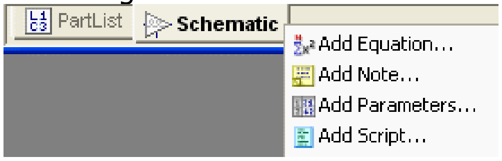
For more information see *User Defined Parameters* (users).

Substrate

Designs can contain any number of local substrates. This is used mainly when creating models so that a model can have an underlying substrate. When your model looks for a substrate it will look *first* in the Substrates container in the design and then in the workspace. Other designs can not access substrates in this design's local container.

Adding a Design Attribute

To add one of these attributes to a design right click on one of the tabs at the bottom of the design and select the desired attribute.



Deleting a Design Attribute

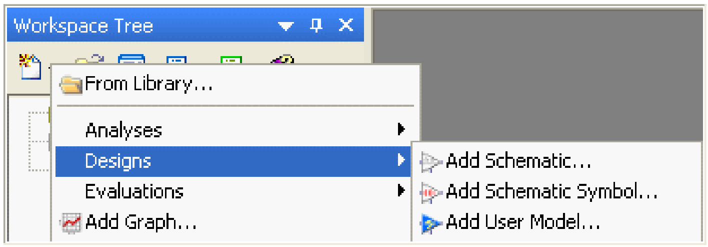
A tab can also be deleted from a design my right clicking on it and selecting the 'Delete' menu entry.

Creating a Design

There are two different ways to to create a design in Genesys. One is the by clicking on the **New Item button** ( ) on the Workspace Tree toolbar or by right clicking on a folder in the workspace tree.

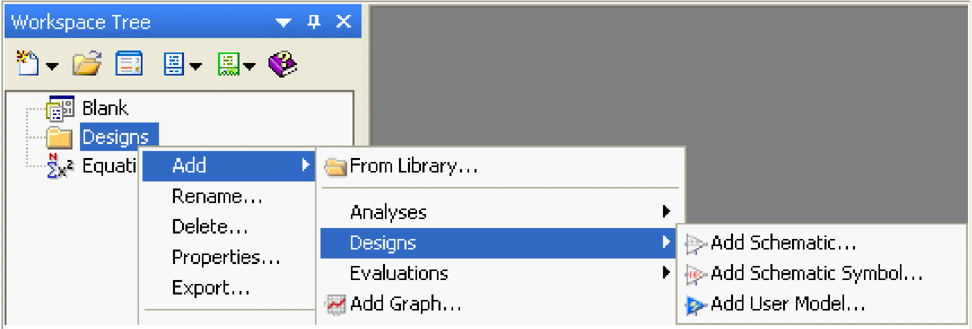
Method 1 - Clicking on the New Item Button

1. Click the New Item button ( ) on the Workspace Tree toolbar
2. Select the **'Designs >'** submenu
3. Now select the design of interest
4. The design will added under the folder that last selected in the workspace tree



Method 2 - Right Clicking on a Workspace Folder

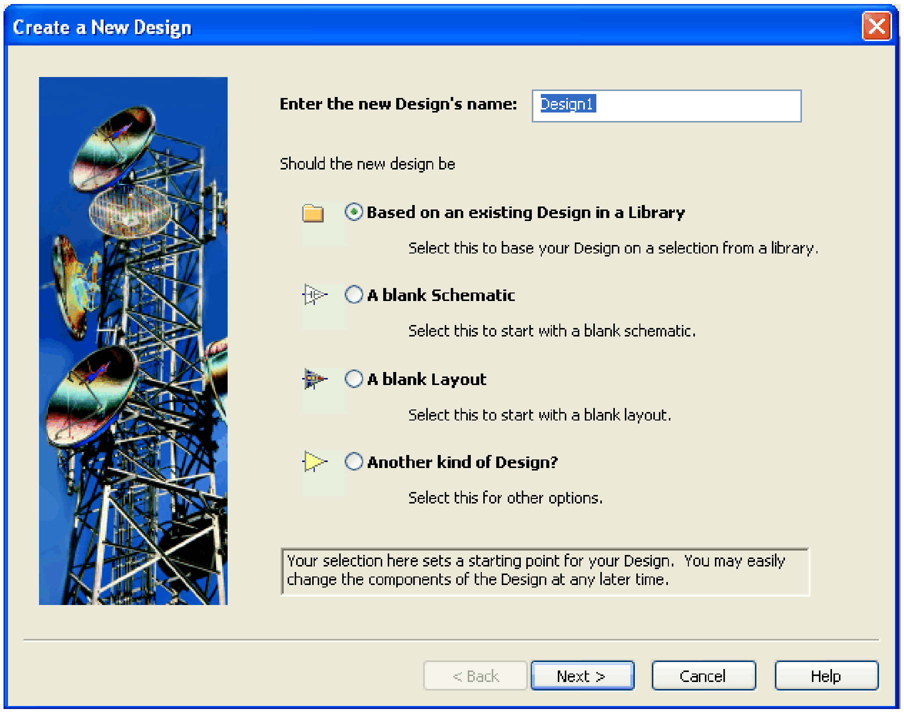
1. Right click on a folder in the workspace tree to bring up the right click menu.
2. Select the 'Add >' submenu.
3. Select the 'Designs >' submenu
4. Now select the design of interest
5. The design will added under the folder that was initially right clicked



Note
If you create the new design in the wrong folder, simply drag it to the folder of interest.

Using the Design Wizard

The Design Wizard makes designing easier if you are new to Genesys or if you want to automate the design process.



Create a New Design

Enter the new Design's name: Design1

Should the new design be

● Based on an existing Design in a Library

Select this to base your Design on a selection from a library.

○ A blank Schematic

Select this to start with a blank schematic.

○ A blank Layout

Select this to start with a blank layout.

○ Another kind of Design?

Select this for other options.

Your selection here sets a starting point for your Design. You may easily change the components of the Design at any later time.

To create a design using the Design Wizard:

1. Click the New Item button on the Workspace Tree toolbar and select **Design Wizard** from the Designs menu.
    1. Type a name for the design in the **Enter the New Design's Name** box.
    2. Click a button to specify whether you want to base your design on an existing design, a blank schematic, a blank layout, or another kind of design.
    3. Do one of the following:
        - Click **Finish** if you selected to base your design on a blank schematic or a blank layout.
        - Click **Next** if you selected to base your design on an existing design or another type design.
    4. If you selected Based on a Design in an Existing Library, do the following:
        - Select the name of the library from the Select the Library Which Contains the Prototype list.
        - Select a prototype from the Select a Prototype list.
    5. If you selected Another Kind of Design, do the following:
        - Click one of the buttons to indicate your intended use for the design.
        - Click **Next**.
        - Click any of the boxes to select initial components to add to your design.
        - Click **Finish**.

Equations

Equations are a powerful tool that enable post processing of data, control over inputs to simulations, and definition of user-defined custom models. Two languages are available for defining equations: Engineering Language (EngLang) and Mathematics Language (Math Language).

Contents

Using Engineering Language

Engineering Language is a simple programming language with structured control statements. Variable types are defined in context and matrices and arrays can be easily manipulated with equations.

- For a full description see *Using Engineering Language* (users).
- For a complete function reference see *Engineering Language Function Reference* (users)

Using Math Language

Math Language, along with most of its built-in functions, was designed to be compatible with m-file script syntax.

- For a full description see *Using Math Language* (users).
- For a complete function reference see *Math Language Function Reference* (users)

Comparing Engineering Language and Mathematics Languages

There are two languages available to use in Equations. Which one should you choose?

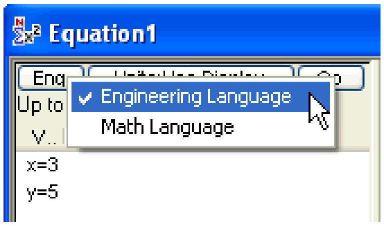
Engineering Language is well suited for interacting with the workspace and for ease of use, while Mathematics Language gives you full control over array processing. Mathematics language is the more powerful and flexible language, and those familiar with MATLAB m-file syntax will prefer to use Mathematics Language. In Engineering Language, it is simpler to access dataset variables (*Dataset.Var* is how you would access a variable Var in a dataset called Dataset), and it understands Swept variables and provides some flexibility in dealing with them.

Mathematics language is syntactically compatible with M-files. It has a lot of power for numeric processing. In some common cases, Mathematics Language code can be more verbose than Engineering Language. For example, if a Linear Analysis produces a swept 2x2 S-matrix of 100 frequencies, EngLang would understand that S[2,1] means S21 for all frequencies. In Math Language, you must explicitly specify that you want all frequencies by using the syntax S(:,2,1). However, there are some places where Mathematics

Language is more compact, allowing things like 1+3j for a complex number, or 1/Var to invert a matrix. Also, Mathematics Language includes support for file I/O and tcp/ip communication functions, both of which are not supported in Engineering Language.

The two equation languages can talk to each other's variables (and the Math Language may retain units and dependencies when you use an EngLang variable). They do not share functions because they are syntactically different (Math Language can return multiple results, for example).

Select which language is being used by an equation set by picking from the Math or Eng button pulldown in the Equation window.



x=3
y=5

Here is a simple comparison of Engineering Language vs. Mathematics Language:

Feature	**EngLang**	**MathLang**
Shortcuts for common vars (eg. S21)	X	
Read/write to/from Datasets	X	X
Run Analysis from a script	X	X
Array-assignments		X
File I/O functions		X
TCP/IP Comm (Instrument connectivity)		X
Signal Processing functions		X
Multiple return values for functions		X

Engineering Language Function Reference

To go directly to entries that start with a specific letter, select one of the following: A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, Z.

Function Name	**Description**
abs (users)	Magnitude of a number x.
abssum (users)	Sums the magnitude of each part in x.
ang (users)	Phase of a complex number x in radians.
ang360 (users)	Phase of a complex number x in radians.
acos (users)	Inverse cosine of x.
acosh (users)	Inverse hyperbolic cosine of x.
array (users)	builds an array with dimensions x by y by ... (can take in any number of arguments, but at least 1).
asin (users)	Inverse sine of x.
asinh (users)	Inverse hyperbolic sine of x.
atan (users)	Inverse tangent of x.
atanh (users)	Inverse hyperbolic tangent of x.
beta (users)	Evaluates the Beta function for corresponding parts of z and w, which must be non-negative real values.
betacdf (users)	Returns the cumulative distribution function of the Beta distribution with parameters a

	and b evaluated at the parts of x.
betainc (users)	Evaluates the incomplete Beta function for corresponding parts of x, z, and w. x must be between 0 and 1. z and w must be non-negative real values.
betainv (users)	Returns the inverse of the Beta cumulative distribution function with parameters a and b evaluated at the parts of p.
betaln (users)	Evaluates the natural logarithm of the Beta function for corresponding parts of z and w, which must be non-negative real values.
betapdf (users)	Returns the probability distribution function of the Beta distribution with parameters a and b evaluated at the parts of x.
ceil (users)	Returns the smallest integer greater than or equal to x. If x is complex, only the real part is used.
column (users)	Force a row or column vector x to be a column vector.
complex (users)	Returns the complex number x + jy.
concatcolumns (users)	Concatenates matrices or vectors x and y. The number of rows of x and y must be equal.
conj (users)	Complex conjugate of x.
cos (users)	Cosine of x, where x is in radians.
cosh (users)	Hyperbolic cosine of x.
cp (users)	Process capability index.
cpk (users)	Process capability index accounting for the mean.
cplower (users)	Lower process capability index.
cpupper (users)	Upper process capability index.
db (users)	x expressed in decibels as 20*log(x).
db10 (users)	x expressed in decibels as 10*log(x).
dbm (users)	Returns the dBm equivalent of x Watts.
dbmtow (users)	Returns the Watts equivalent of x dBm.
dbpolar (users)	Returns the complex value of a voltage in dB and the angle in degrees.
derivative (users)	returns a vector that is the derivative at each point of the dependent with respect to independent vector.
dev_lin_phase (users)	returns a vector in radians of deviation from linear phase; this function is a clone of the ADS function by the same name.
diag (users)	If the argument is a matrix, this returns a vector with the diagonal parts. Otherwise, the argument must be a vector, and diag creates a matrix with the vector on the diagonal and all other parts zero.
erf (users)	Computes the error function of each part x.
erfc (users)	Computes the complementary error function of each part of x.
exp (users)	e to the power of x, where e = 2.7182817...
eye (users)	Function to build an eye diagram from transient data. symbolRate is in Hz, numCycles is the number of cycles to plot before wrapping (optional), and delay is the number of samples of delay (optional).
fft (users)	Discrete Fourier Transform (DFT) of data, computed with FFT algorithm when possible. The len argument (the FFT Length) is optional.
floor (users)	Returns the largest integer less than or equal to x. If x is complex, only the real part is used.
gamma (users)	Evaluates the Gamma function for each part of x, which must be real.
gammainc (users)	Evaluates the incomplete Gamma function for each part of x, which must be real.
gammaln (users)	Evaluates the natural logarithm of the Gamma function for each part of x without computing the Gamma function itself.
gammatoz (users)	Returns the complex impedance given the normalizing impedance and the voltage reflection coefficient (gamma).

getdbunit (users)	Returns the appropriate dB unit: RelDB10 if the input units are power or RelDB20 if the input units are voltage/current.
getindep (users)	Returns the string property containing the path to the independent value of a variable x. (ie. the reference to the independent variable)
getindepvalue (users)	Returns the value of the independent variable of a variable x. x must have only one independent value.
getunits (users)	Returns an integer corresponding to the units of a variable x. This integer may be used by setunits.
hb_gain (users)	Calculates the complex gain for selected spectral component.
hb_getspcomp (users)	Returns the complex amplitude of the spectral component in Watts.
hb_getspcompdbm (users)	Returns the amplitude of the spectral component in dBm.
hb_iipn (users)	Returns the input intercept point of two components IndexS1 and IndexS2 of the power spectrum SpectrPout.
hb_ipn (users)	Calculates the common part of the input and output intercept point functions (hb_iipn and hb_oipn).
hb_larges (users)	Calculates the Large Signal S-parameter for a 1-tone HB analysis.
hb_largesmix (users)	Calculates the LargeSignal S-parameter for multi-tone HB analyses.
hb_oipn (users)	Calculates the output intercept of the two components IndexS1 and IndexS2 of the power Spectrum SpectrPout.
hb_spurious (users)	Calculates the max amplitude of spurious spectral components in the spectrum SpectrVout.
hb_totalsp (users)	Calculates total amplitude for spectral components having the same frequency and sets all of them to the same total amplitude.
hb_transgain (users)	Calculates the transducer gain.
hb_transgaindb (users)	Calculates the transducer gain in dB.
histogram (users)	creates bins for the data in x and returns the number of parts in each container.
hermitian (users)	conjugate transpose of a matrix or swept matrix x
identity (users)	Returns an nxn identity matrix. (Datatype is Complex)
iff (users)	Returns value1 if *boolean expression* is true, otherwise value2.
ifft (users)	Inverse Discrete Fourier Transform (IDFT) of data, computed with IFFT algorithm when possible. The len argument (IFFT length) is optional.
iftrue (users)	If x is true, return y. x and y may be vectors.
imag (users)	Imaginary part of a (complex) number x.
im (users)	Same as imag(x).
integrate (users)	Numerically integrate the dependent with respect to independent, from index = start to index = stop.
interpolate (users)	Produces an array where the independent value of var is changed to new_indep, interpolating as necessary.
intersect (users)	Produces a vector representing a set which is the intersection of two sets (vectors) x and y.
inverse (users)	Returns the inverse of a non-singular matrix x.
kurtosis (users)	Returns the sample Kurtosis of x.
largeS (users)	returns the largeSij for this HB dataset
lininterp (users)	Returns the linear interpolation of the output values given the data values of indep and dep.
ln (users)	Natural logarithm of x.
log (users)	Logarithm (base 10) of x.
mag (users)	Same as abs(x).
matrix (users)	Builds an x by y matrix of complex numbers. parts are initialized to zero.

max (users)	Returns the maximum (magnitude) part of any array x on the dimension iDim.
mean (users)	Returns average of all parts in a vector or an array on the dimension iDim.
median (users)	Returns the median of all parts in a vector or an array on the dimension iDim.
min (users)	Returns the minimum (magnitude) part of any array x on the dimension iDim.
mode (users)	Returns the mode of all parts in a vector or an array on the dimension iDim.
moment (users)	Returns the central moment of order *order* of a vector x.
ndim (users)	Number of dimensions of an array x.
nftovni (users)	Converts noise factor into an equivalent input noise resistance.
norm (users)	Magnitude-squared of a complex number x.
normcdf (users)	Returns the cumulative distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x.
norminv (users)	Returns the inverse cumulative distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x.
normpdf (users)	Returns the probability distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x.
numcols (users)	Returns the number of columns in a matrix x.
numrows (users)	Returns the number of rows in a matrix x.
ones (users)	if n argument is omitted: generate a vector of m 1's, otherwise generate an mxn matrix of 1's.
posterror (users)	Converts the argument to a string (if necessary) and posts the string to the error log. The error stays on the log until the equations are recalculated. This causes the equations to error out and not calculate, as well as posting the error.
postwarning (users)	Converts the argument to a string (if necessary) and posts the string to the error log as a warning. The warning stays on the log until the equations are recalculated. This does not cause the equations to error out. The equations will calculate.
pow (users)	Returns x to the power of y.
prctile (users)	Returns the p'th percentiles of a vector x (p can be a scalar or a vector of percent values).
prod (users)	Calculates the product of all the parts in x.
quantile (users)	Returns the q'th quantiles of a vector x (q can be a scalar or a vector of quantile values).
rand (users)	Generate uniformly distributed random numbers on the interval [0, 1). Both arguments are optional. If both arguments are omitted, the function returns a random scalar. If the second argument is omitted, generate a vector of m random numbers. Otherwise, generate an mxn matrix of random numbers.
randn (users)	Generate Gaussian distributed random numbers with mean 0 and variance 1. Both arguments are optional. If both arguments are omitted, the function returns a random scalar. If the second argument is omitted, generate a vector of m random numbers. Otherwise, generate an mxn matrix of random numbers.
real (users)	Real part of a (complex) number x
re (users)	Same as real(x)
reshape (users)	sets the dimensions of the variable x to be that described by newshape. Swept-dimensions are NOT counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, reshape(S, [4;1]) would return a variable containing S, but having dimensions 100x4x1)
resize (users)	sets the dimensions of the variable x to be that described by newshape. Swept-dimensions are counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, resize(S, [100;4;1]) would

	return a variable containing S, but having dimensions 100x4x1)
reverse (users)	returns the reverse of a vector, so [1,2,3,4] becomes [4,3,2,1]; for matrices this will reverse the columns
rltogamma (users)	Returns the voltage reflection coefficient (gamma) based on the return loss in dB and the angle in radians
rltoz (users)	Returns the complex impedance based on the normalizing impedance, return loss in dB and the angle in radians
rotate (users)	rotates a vector by the integer rotate; if vector = [1,2,3,4], rotate(vector,2) returns [3,4,1,2]; for matrices this will rotate the rows.

RsCondToThick (users)	Returns metal thickness in defined by string parameter Tunits units;
	RsCondToThick(Rs,Cond,Tunits);
	Rs - sheet resistance *Ohm/Sq*, Cond - conductivity *Ohm*M*
	Examples: T_mil=RsCondToThick(50,5.8e7,"mil");
	T_mm=RsCondToThick(50,5.8e7,"mm")

RsResToThick (users)	Returns metal thickness in defined by string parameter Tunits units; RsResToThick(Rs,Res,Tunits); Rs - sheet resistance *Ohm/Sq*, Res - resistivity *Ohm*M* Examples: T_mil=RsResToThick(50,1.724e-8,"mil"); T_mm=RsResToThick(50,1.724e-8,"mm")

RsRhoToThick (users)	Returns metal thickness in defined by string parameter Tunits units;
	RsRhoToThick(Rs,Rho,Tunits);
	Rs - sheet resistance *Ohm/Square*, Rho=Res/ResCop - relative to copper resistivity Res, where copper resistivity ResCop=1.724e-8 *Ohm*M*
	Examples: T_mil=RsRhoToThick(50,1000,"mil");
	T_mm=RsRhoToThick(50,1000,"mm")

runanalysis (users)	Run an analysis in the workspace tree
setindep (users)	set the independent reference for a swept dependent variable to indepvar(s). A minimum of two arguments is required. This function can be used to remove all independent values of a variable by passing in a blank string for the second argument.
setplottype (users)	sets the plot-type property of a variable. Valid plot types include "" (Empty string), "Discrete", "Spectrum", "Level", "Spur", "SpurFree", "OutOfRange", "ValidIF", "Contour", "PointPlot", and "Histogram".
setunits (users)	sets a variable named varname to have units specified by unit. unit may be an integer or a string. Example setunits("totaltime","msec") or setunits("frqsweep", "MHz"). The units are used to by graphs to determine the axis labels and values. They can also by used by the Tune window. Use UseMKS if you are settings units of variables manually to avoid confusion.
shape (users)	Returns a vector containing the number of parts in each dimension of x. Swept-dimensions are NOT counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, shape(S) returns the vector [2;2]).
sin (users)	Sine of x, where x is in radians.
sinc (users)	Sin(x) / x, or 1 if x = 0.
sinh (users)	Hyperbolic sine of x.
size (users)	Returns a vector containing the number of parts in each dimension of x. Swept-dimensions are counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, size(S) returns the vector [100;2;2]).
skewness (users)	Returns the sample skewness of a vector x.
sort (users)	Sorts a vector x in ascending order.
std (users)	Calculates the standard deviation of x.
sqr (users)	Square-root of x.
sqrt (users)	Same as sqr(x).

stoy (users)	convert S-parameters to Y-parameters
stoz (users)	convert S-parameters to Z-parameters
stos (users)	renormalize S-parameters to a different impedance
substrateer (users)	return the dielectric constant (Er) of the substrated named SubstName.
substratetand (users)	return the loss tangent of substrate named SubstName.
substraterho (users)	return the relative resistivity of substrate named SubstName.
substratetmet (users)	return the metal thickness of substrate named SubstName.
substraterough (users)	return the surface roughness of substrate named SubstName.
substrateh (users)	return the height of substrate named SubstName.
sum (users)	Calculates the sum of all the parts of x on the dimension iDim.
tan (users)	Tangent of x, where x is in radians.
tanh (users)	Hyperbolic tangent of x.
tcdf (users)	Returns the cumulative distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of x.
tinv (users)	Returns the inverse of the cumulative distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of p.
time (users)	constructs a time-domain waveform from complex spectra at arbitrary frequencies. (Eg: time(V1, Freq, Time) where V1, Freq, and Time are variables in a HarBEC Dataset)
times (users)	Returns the part-by-part product of the two arguments. If one entry is a scalar and one is an array, the return value is the scalar multiplied by every part in the array. This is in contrast to the * operator that does matrix multiplication.
timevector (users)	Creates a vector of times from start to stop with a specified step size.
tpdf (users)	Returns the probability distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of x.
transpose (users)	Transposes a matrix or swept matrices x
union (users)	Produces a vector representing a set which is the union of two sets (vectors) x and y
englang_unwrap (users)	returns a vector of unwrapped phase; both input and output of the function are radians (note that ang() returns radians so this is already self-consistent; z = unwrap(ang(Linear1.S[2,1])) is an example.
using (users)	sets the current context in an equation block to the dataset called Dataset.
var (users)	Calculates the variance of x.
vector (users)	Builds a vector of complex numbers of length x. parts are initialized to zero.
zeros (users)	if n argument is omitted: generate a vector of m 0's, otherwise generate an mxn matrix of 0's

abs

Syntax

y = abs(x)

Definition

abs takes the absolute value of a real variable or the magnitude of a complex variable. Same as mag function

Examples:

Formula	Result
abs(-1.5)	1.5
abs(complex(1,1))	1.414
abs([-1;-2;3])	[1;2;3]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
mag (users)

abssum

Syntax

y = abssum(x)

Definition
abssum calculates the absolute value of each part in x and returns the sum of those parts. If an part in x is a complex number then the magnitude of the part is used in the summation.

Examples:

Formula	Result

x = [-2.1 , 3.4 , 5.6]
y = abssum(x)
	11.1

x = [complex(1,-.4), complex(-1,1)]
y = abssum(b)
	2.491

Compatibility
Numeric scalars, Vectors, Arrays

See Also
abs (users)

acos

Syntax
y = acos(x)

Definition
acos returns the inverse cosine of the number, in radians (MKS) between 0 <= r <= PI

Examples:

Formula	Result	or
acos(0)	1.571	PI/2
acos(1)	0	0
acos(-1)	3.141	PI
acos(.707)	0.786	PI/4
acos(-.707)	2.356	3*PI/4

Compatibility
Numeric scalars, Vectors, Arrays

See Also

asin (users)

acosh

Syntax
y = acosh(x)

Definition
acosh returns the inverse hyperbolic cosine of the number, or log(x + sqrt(x^2 - 1)). acosh takes magnitude of a complex argument before evaluating.

Examples:

Formula	Result
acosh(1)	0
acosh(10)	2.993
acosh(0)	undefined

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *asinh* (users)
- *atanh* (users)

ang

Syntax
y = ang(x)

Definition
ang finds the angle of a complex number. The range of y is -180 to 180 degrees; internally, the number is stored in radians (MKS), but since the unit is ANGLE, the value can be displayed in degrees or radians. (This differs from **ang360**, which returns a **unitless** value.)

Examples:

Formula	Result
ang(1)	0
ang(complex(sqr(3) / 2 , .5))	0.524
ang(complex(1,1))	.785

Compatibility
Numeric scalars, Vectors, Arrays

See Also
ang360 (users)

ang360

Syntax
y = ang360(x)

Definition
ang360 finds the angle of a complex number in degrees. The range of y is 0 to 360; ang360 specifically returns a **unitless** value (unlike **ang**, which returns a value with ANGLE units).

Examples:

Formula	**Result**
ang360(1)	0
ang360(complex(sqr(3)/2 , .5))	30
ang360(complex(1,1))	45

Compatibility
Numeric scalars, Vectors, Arrays

See Also
ang (users)

array

Syntax
y = array(a, b, ...)

Definition
array creates a multidimensional complex array. The argument list must have at least one integer (to produce a vector). All arguments are set to zero.

Examples:

Formula	**Result**
array(2)	a 2 part vector
array(3, 4, 5)	a 3 dimensional array of size 3x4x5

Compatibility
Variable length integer list

See Also

- *reshape* (users)
- *shape* (users)

asin

Syntax
y = asin(x)

Definition
asin returns the inverse sine of the number, in radians (MKS) between -PI / 2 <= r <= PI / 2

Examples:

Formula	Result	or
asin (0)	0	0
asin (1)	1.571	PI/2
asin(-1)	-1.571	-PI/2
asin (.707)	0.786	PI/4
asin (-.707)	-0.786	-PI/4

Compatibility
Numeric scalars, vectors, arrays

See Also
acos (users)

asinh

Syntax
y = asinh(x)

Definition
asinh returns the inverse hyperbolic sine of the number, or log(x + sqrt(x^2 + 1)). asinh takes magnitude of a complex argument before evaluating.

Examples:

Formula	Result
asinh(1)	0.881
asinh(10)	2.998
asinh(0)	0

Compatibility
Numeric scalars, vectors, arrays

See Also

- *acosh* (users)
- *atanh* (users)

atan

Syntax
y = atan(x)

Definition
atan returns the inverse tangent of the number, in radians (MKS) between -PI/2 < r < PI/2

Examples:

Formula	Result	or
atan(0)	0	
atan(1)	0.785	PI/4
atan(-1)	-0.785	-PI/4
atan(.5)	0.464	
atan(-.5)	-0.464	

Compatibility

Numeric scalars, Vectors, Arrays

See Also

tan (users)

atanh

Syntax

y = atanh(x)

Definition

atanh returns the inverse hyperbolic tangent of the number, or 0.5 * log((1 + x) / (1 - x)). atanh takes magnitude of a complex argument before evaluating.

Examples:

Formula	Result
atanh(1)	undefined
atanh(.5)	0.549
atanh(-.5)	-0.549
atanh(0)	0

Compatibility

Numeric scalars, Vectors, Arrays

See Also

tanh (users)

beta

Syntax

y = beta(z, w)

Definition

Evaluates the Beta function for corresponding parts of z and w, which must be non-negative real values. Both must be of equal dimensions, or either one can be a scalar.

The Beta function is defined as the integral from 0 to 1 of t^(z-1)*(1-t)^(w-1) dt.

Examples:

Formula	Result
beta(1.5, 2)	0.2667
beta([3, 4, 5], 3)	[0.0333, 0.0167, 0.0095]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

- *gamma* (users)
- *gammaln* (users)
- *gammainc* (users)
- *betaln* (users)
- *betainc* (users)

betacdf

Syntax
y = betacdf(x, a, b)

Definition
Returns the cumulative distribution function of the Beta distribution with parameters a and b evaluated at the parts of x. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	Result
betacdf(0.5, 1, 1.5)	0.6464
betacdf(0.8, [1, 2], 3)	[0.992, 0.9728]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *betapdf* (users)
- *betainv* (users)

betainc

Syntax
y = betainc(x, z, w)

Definition
Evaluates the incomplete Beta function for corresponding parts of x (which must be between 0 and 1), z, and w, which must be non-negative real values. The arguments must be of equal dimensions, or any one can be a scalar.

The incomplete Beta function is defined as (1 / beta(z, w)) times the integral from 0 to x of t^(z-1)*(1-t)^(w-1) dt.

Examples:

Formula	Result
betainc(0.5, 1.5, 2)	0.6187
betainc(0.8, [3, 4, 5], 3)	[0.9421, 0.9011, 0.852]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *gamma* (users)
- *gammaln* (users)
- *gammainc* (users)
- *beta* (users)
- *betaln* (users)

betainv

Syntax
x = betainv(p, a, b)

Definition
Returns the inverse of the Beta cumulative distribution function with parameters a and b evaluated at the parts of p. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	**Result**
betainv(0.5, 1, 1.5)	0.37
betainv(0.8, [1, 2], 3)	[0.4152, 0.5825]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *betapdf* (users)
- [betacdf]

betaln

Syntax
y = betaln(z, w)

Definition
Evaluates the natural logarithm of the Beta function for corresponding parts of z and w, which must be non-negative real values. Both must be of equal dimensions, or either one can be a scalar.

Since the Beta function can range over huge values, it is sometimes preferable to use its logarithm.

Examples:

Formula	**Result**
betaln(1.5, 2)	-1.3218
betaln([3, 4, 5], 3)	[-3.4012, -4.0943, -4.654]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *gamma* (users)
- *gammaln* (users)
- *gammainc* (users)
- *beta* (users)
- *betainc* (users)

betapdf

Syntax
y = betapdf(x, a, b)

Definition
Returns the probability distribution function of the Beta distribution with parameters a and b evaluated at the parts of x. All inputs must match in dimensions or be scalars. Scalars

are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	**Result**
betapdf(0.5, 1, 1.5)	1.0607
betapdf(0.8, [1, 2], 3)	[0.12, 0.384]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *betacdf* (users)
- *betainv* (users)

ceil

Syntax
y = ceil(x)

Definition
ceil returns the smallest integer greater than or equal to x. If x is complex, only the real part is used.

Examples:

Formula	**Result**
ceil(10)	10
ceil(complex (1.5 , 6))	2
ceil([-0.5, 0.5])	[0 , 1]

Compatibility
Numeric scalars, vectors, arrays

See Also
floor (users)

column

Syntax
y = column(x)

Definition
Creates a column vector y from a row or column vector x. A column vector has one column and one or more rows.

Examples:

Formula	**Result**
x = [1,2,3,4,5]y = column(x)	y = [1;2;3;4;5] (vector of 5 rows and 1 column)

Compatibility
Vectors

See Also
transpose (users)

complex

Syntax
z = complex(x, y)

Definition
Returns a complex number in the form x + jy. The complex number(s) are stored as z = (x2 + y2)(1/2). By default the magnitude of the complex number(s) will be displayed in tables and datasets. complex(x, y) can be used on vectors or matrices of same dimensions.

Examples:

Formula	**Result**
complex(1 , 1)	1 + j = 1.414
complex(1 , -4)	1 - j4 = 4.123
complex([0 , 5] , [3 , 0])	[3 , 5]
complex(5 , 0)	5

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
mag (users)

concatcolumns

Syntax
z = concatcolumns(x, y)

Definition
Concatenates x and y. Variables x and y must have the same number of rows, be compatible data types, and have no more that two dimensions.

Examples:

Formula	**Result**

x = [1 , 2]
y = [3 , 4]
z = concatcolumns(x , y)
	z = [1 , 2 , 3 , 4]

x = [1 ; 3 ; 5]
y = [2 ; 4 ; 6]
z = concatcolumns(x , y)
	z = [1, 2 ; 3 , 4 ; 5 , 6]

Compatibility
*. Numeric Scalars
*. *array* (users)
*. *vector* (users)

conj

Syntax
y = conj(x)

Definition
Calculates the complex conjugate of x, where x is a complex number. The conjugate of x + jy is x - jy.

Examples:

Formula	Result
conj(complex(1 , 2))	1 - j2
y = conj(complex([1 ; 2] , [3 ; -4]))	y[1] = 1 - j3 y[2] = 2 + j4

Compatibility

*. Numeric Scalars
*. *array* (users)
*. *vector* (users)

cos

Syntax

y = cos(x)

Definition

cos returns the cosine of the number, in radians (MKS) between -1 <= r < 1

Examples:

Formula	Result
cos(0)	1
cos(PI)	-1
cos(PI / 2)	0
cos(PI / 4)	0.707
cos(2.094) or cos(2 * PI / 3)	- 0.5

Compatibility

Numeric scalars, Vectors, Arrays

See Also

sin (users)

cosh

Syntax

y = cosh(x)

Definition

cosh returns the hyperbolic cosine of the number, or (ex + e-x) / 2.

Examples:

Formula	Result
cosh(1)	1.543
cosh(5)	74.21
cosh(PI / 3)	1.6
cosh(PI / 6)	1.14
cosh(0)	1

Compatibility

Numeric scalars, Vectors, Arrays

See Also

acosh (users)

Cp

Syntax
y = Cp(USL, LSL, DataArray)

Definition
Determines the process capability index of a numeric array given both upper and lower specification limits.

The Cp function is defined as (USL - LSL) / (6 * std(DataArray)) where USL is the Upper Specification Limit, LSL is the Lower Specification Limit, and DataArray is a numeric array of data points for which the standard deviation will be derived.

Examples:

Formula	**Result**
Cp(9, 5, [6, 7.5, 6.2, 8.1, 8.5])	0.596
Cp([16, 12], [8, 6], [6.3, 7.2, 6.4, 8.4, 7.9])	[1.452, 1.089]

Compatibility
Numeric arrays

Library
Process Capability in MoreFunctions.xml.

ℹ **NOTE**: This library is not automatically loaded at startup.

See Also
cpk, cplower, cpupper

Cpk

Syntax
y = Cpk(USL, LSL, DataArray)

Definition
Determines the process capability index of a numeric array given both upper and lower specification limits. This index also includes the affects of the mean value of the data.

The Cpk function is defined as min(CpUpper(USL, DataArray), CpLower(LSL, DataArray)) where USL is the Upper Specification Limit, LSL is the Lower Specification Limit, and DataArray is a numeric array of data points for which the standard deviation will be derived.

Examples:

Formula	**Result**
Cpk(9, 5, [6, 7.5, 6.2, 8.1, 8.5])	0.518

Compatibility
Numeric arrays

Library
Process Capability in MoreFunctions.xml.

NOTE: This library is not automatically loaded at startup.

See Also

- *cp* (users)
- *cplower* (users)
- *cpupper* (users)

CpkLower

Syntax
y = CpkLower(LSL, DataArray)

Definition
Determines the lower process capability index of a numeric array given the lower specification limit.

The CpkLower function is defined as (mean(DataArray) - LSL) / (3 * std(DataArray)) where LSL is the Lower Specification Limit and DataArray is a numeric array of data points for which the standard deviation will be derived.

Examples:

Formula	Result
CpkLower(5, [6, 7.5, 6.2, 8.1, 8.5])	0.673
CpkLower([5, 4], [5.2, 6.9, 6.2])	[0.429, 0.819]

Compatibility
Numeric arrays

Library
Process Capability in MoreFunctions.xml.

NOTE: This library is not automatically loaded at startup.

See Also

- *cp* (users)
- *cpk* (users)
- *cpupper* (users)

CpkUpper

Syntax
y = CpkUpper(USL, DataArray)

Definition
Determines the upper process capability index of a numeric array given the upper specification limit.

The CpkUpper function is defined as (USL - mean(DataArray)) / (3 * std(DataArray)) where USL is the Upper Specification Limit and DataArray is a numeric array of data points for which the standard deviation will be derived.

Examples:

Formula	**Result**
CpkUpper(9, [6, 7.5, 6.2, 8.1, 8.5])	0.518
CpkUpper([9, 8], [5.2, 6.9, 6.2])	[1.131, 0.741]

Compatibility
Numeric arrays

Library
Process Capability in MoreFunctions.xml.

ℹ **NOTE**: This library is not automatically loaded at startup.

See Also
cp (users)
cpk (users)
cplower (users)

db

Syntax
y = db(x)

Definition
db returns x expressed in decibels as 20 * log(x).

Examples:

Formula	**Result**
db(1)	0
db(100)	40
db(0.001)	-60

Compatibility
Numeric scalars, Vectors, Arrays

See Also
db10 (users)

db10

Syntax
y = db10(x)

Definition
db10 returns x expressed in decibels as 10 * log(x).

Examples:

Formula	**Result**
db10(1)	0
db10(100)	20
db10(0.001)	-30

Compatibility
Numeric scalars, vectors, arrays

See Also

db (users)

dbm

Syntax
y = dbm(x)

Definition
dbm returns the dBm equivalent of x Watts, or 10 * log (x * 1000) where x is measured in Watts.

Examples:

Formula	Result
dbm(1000)	60
dbm(1)	30
dbm(0.5)	26.99

Compatibility
Numeric scalars, Vectors, Arrays

See Also
dbmtow (users)

dbmtow

Syntax
y = dbmtow(x)

Definition
dbmtow returns the Watt equivalent of x dBm, or (1 / 1000) * 10 (x / 10) where x is measured in dBm.

Examples:

Formula	Result
dbmtow(60)	1000
dbmtow(30)	1
dbmtow(27)	0.501

Compatibility
Numeric scalars, Vectors, Arrays

See Also
dbm (users)

dbpolar

Syntax
y = dbpolar(dB, ang)

Definition
dbpolar returns the complex value of a voltage dB and an angle ang. The voltage is measured in dB and the angle is measured in degrees. Can only take in numbers and square matrices.

Examples:

Formula	Result
dbpolar(12, 45)	2.815 + j2.815
dbpolar(0 , 60)	0.5 + j0.886
dbpolar(100 , 15)	96592.6 + j25881.9

Compatibility

- Numeric scalars
- *array* (users)

derivative

Syntax
z = derivative(dep, indep)

Definition
Returns a vector that is the derivative at each point of the dependent with respect to the independent. Parameter dep is the dependent vector and indep is the independent vector. Parameters must be column vectors.

Examples:

Formula	Result

x = [1 ; 2 ; 3 ; 4 ; 5]
y = x^2
z = derivative(y , x)
	z = [3 ; 3 ; 5 ; 7 ; 9]

x = [4 ; 0 ; 4 ; 0 ; 4]
y = x^(1 / 2)
z = derivative(x , y)
	z = [2 ; 2 ; 2 ; 2 ; 2]

Compatibility
Vectors

See Also
integrate (users)

dev_lin_phase

Syntax
z = dev_lin_phase(x , y)

Definition
Returns a vector in radians of deviation from the linear phase. Clone of the ADS function. Parameter x is the dependent vector and parameter y is the independent vector. Parameters must be vectors.

Examples:

Formula	Result
dev_lin_phase([5 , 10] , [10 , 20])	[6.283 , 3.142]
dev_lin_phase([0 , 2 , 5] , [4 , 5 , 6])	[2.169 , 0.608 , 1.953]

Compatibility
vector (users)

diag

Syntax
y = diag(x)

Definition

Returns a vector of the diagonal parts from a square matrix. The diagonal starts from the entry in the first row and the first column. Given a row vector of n number of columns, the function will return a n by n matrix with the values of the vector on the diagonal.
Parameter x must be a square matrix or a row vector.

Examples:

Formula	**Result**
diag([1 , 2 , 3 , 4])	[1, 4]
diag([1 , 1 , 1])	[1 , 0 , 0 ; 0 , 1 , 0 ; 0 , 0 , 1]

Compatibility

- *vector* (users)
- *array* (users)

iftrue

Syntax

z = iftrue(x , y)

Definition

iftrue returns y if x is true or 0 if x is false. The bool value x is either 1 (true) or 0 (false).
Parameters x and y can also be vectors or arrays of the same size.

Examples:

Formula	**Result**
z = iftrue(1 , 10)	z =10
z = iftrue(0 , 10)	z = 0
z = iftrue([1 , 0 ; 1 , 1] , [6 , 8 ; 7 , 5])	z = [6 , 0 ; 7 , 5]
z = iftrue([1 , 0 , 1 , 1 , 0] , [1 , 2 , 3 , 4 , 5])	z = [1 , 0 , 3 , 4 , 0]

Compatibility

Numeric scalars, Vectors, Arrays

unwrap

Syntax

z = unwrap(wrappedPhase)

Definition

unwrap returns a vector of unwrapped phase; both input and output of the function are in radians

ℹ Note that ang() returns a value in radians.

Examples:

Formula
z = unwrap(ang(Linear1.S[2 , 1]))

Compatibility

Numeric scalars, Vectors, Matrices

erf

Syntax
y = erf(x)

Definition
erf computes the error function of each part of x. The parts of x must be real.

Examples:

Formula	**Result**
erf(-1.5)	-0.9661
erf(2)	0.9953
erf([-1;-2;1.1])	[-0.8427; -0.9953; 0.8802]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
erfc (users)

erfc

Syntax
y = erfc(x)

Definition
erfc computes the complementary error function of each part of x. The parts of x must be real.

Examples:

Formula	**Result**
erfc(-1.5)	1.9661
erfc(2)	0.0047
erfc([-1;-2;1.1])	[1.8427; 1.9953, 0.1198]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
erf (users)

exp

Syntax
y = exp(x)

Definition
Returns the exponential of x. Calculated as e to the power of x, where e = 2.7182817...

Examples:

Formula	**Result**
exp(1)	2.718
exp([0 , 1.5])	[1 , 4.482]
exp([-0.5 , 0.5 ; -2 , 2])	[0.607 , 1.649 ; 0.135 , 7.389]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

ln (users)

eye

Syntax

eye(data, symbolRate, numCycles, delay)

Definition

Function builds an eye diagram from a transient data set. Parameter symbolRate is measured in Hz, numCycles is the number of cycles to plot before wrapping (optional, default is 1), and delay is the number of samples from the beginning of the data to ignore (optional, default is 0).

Examples:

For an example using the eye function to generate an eye diagram, see the example workspace "Eye Diagram Example" located in the *Equations* subdirectory of your *Examples* folder.

fft

Syntax

fft(data, len)

Definition

Discrete Fourier Transform (DFT) of data. Computed with FFT algorithm when possible. The parameter len is the FFT length and is optional.

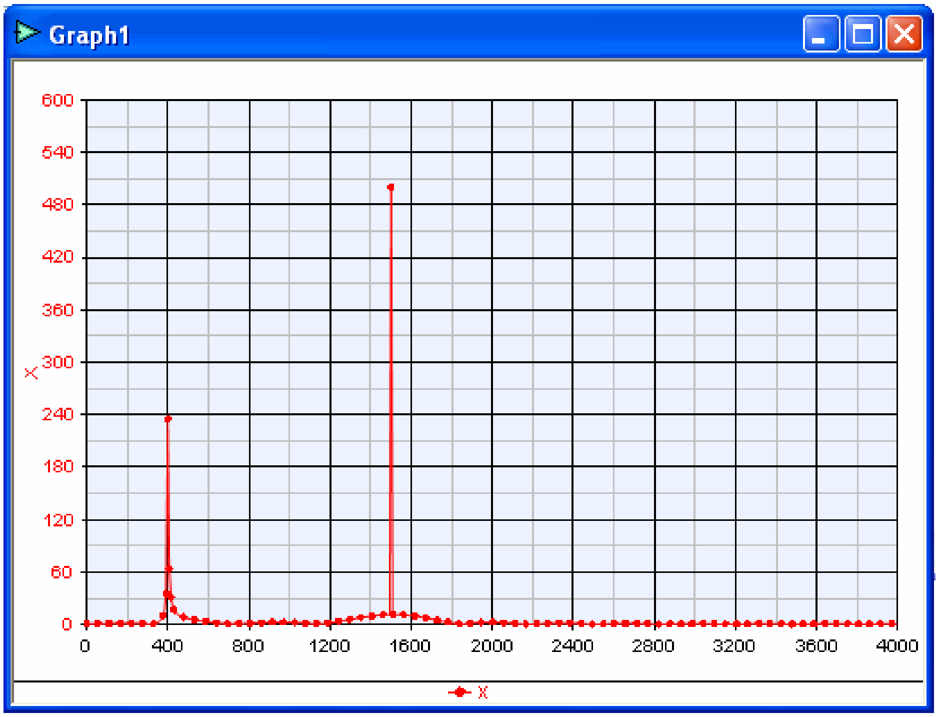
Examples:

The following example generates a signal consisting of the sum of two sinusoids: one at 400 Hz, and one at 1500 Hz. The fft function is then used to compute the spectrum of the signal.

```
fft_len = 1024    ' length of the FFT

fs = 8000         ' 8000 Hz sampling rate

T = 1/fs          ' sample time

L = 1000          ' length of signal

t = (0:(L-1))*T   ' time vector

' x will be the sum of two sinusoids:

' one at 400 Hz and one at 1500 Hz

x = 0.5*cos(2*PI*400*t) + cos(2*PI*1500*t)

X = fft(x, fft_len)  ' spectrum of x

X = X[1:(fft_len/2)] ' we only care about single side-band (the rest is redundant)

f = fs/2 * (0:(2/fft_len):1)
```

setindep("X","f")

The following graph displays the magnitude of X, the spectrum of x.



Compatibility
Dataset

See Also
ifft (users)

floor

Syntax
y = floor(x)

Definition
Returns the largest integer less than or equal to x. If x is a complex number, only the real part is used.

Examples:

Formula	**Result**
floor(10)	10
floor(complex (1.5 , 6))	1
floor([-0.5, 0.5])	[-1 , 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
ceil (users)

gamma

Syntax
y = gamma(x)

Definition
Evaluates the Gamma function for each part of x, which must be real.

The Gamma function is defined as the integral from 0 to infinity of t^(x-1)*exp(-t) dt.

Examples:

Formula	Result
gamma(-1.5)	2.3633
gamma([3, 4, 5])	[2, 6, 24]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *gammaln* (users)
- *gammainc* (users)
- *beta* (users)
- *betaln* (users)
- *betainc* (users)

gammainc

Syntax
y = gammainc(x, a)

Definition
Evaluates the incomplete Gamma function for each part of x, which must be real. The arguments must be arrays of equal dimensions, or either one can be a scalar.

The incomplete Gamma function is defined as (1/gamma(a)) times the integral from 0 to x of t^(a-1)*exp(-t) dt.

Examples:

Formula	Result
gammainc(1.5, 2)	0.4422
gammainc([3, 4, 5], 3)	[0.5768, 0.7619, 0.8753]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *gamma* (users)
- *gammaln* (users)
- *beta* (users)
- *betaln* (users)
- *betainc* (users)

gammaln

Syntax
y = gammaln(x)

Definition
Evaluates the natural logarithm of the Gamma function for each part of x without computing the Gamma function itself.

Since the Gamma function can range over huge values, it is sometimes preferable to use its logarithm.

Examples:

Formula	Result
gammaln(1.5)	-0.1208
gammaln([3, 4, 5])	[0.6931, 1.7918, 3.1781]

Compatibility

- Numeric scalars
- *vector* (users)
- *array* (users)

See Also
gamma, gammainc, beta, betaln, betainc

gammatoz

Syntax
x = gammatoz(zo, gamma)

Definition
Returns the complex impedance given the normalizing impedance (zo) and the voltage reflection coefficient (gamma). Parameter zo must be an array and gamma can be a numeric scalar or a complex number.

Examples:

Formula	Result
gammatoz([-0.4, -0.2, 0.2] , 5)	[0.6, 0.3, -0.3]

x = gammatoz([0.1, 0.2, 0.3], complex(1, 1))	x[1] = -0.1 + j0.2
	x[2] = -0.2 + j0.4
	x[3] = -0.3 + j0.6

Compatibility

- Numeric scalars
- *vector* (users)
- *array* (users)

getdbunit

Syntax
y = getdbunits(x)

Definition
Returns the appropriate dB unit as an integer value. The unit for *db10* (users) is returned if the input units are power and the unit for *db* (users) is returned if the input units are

voltage/current.

Examples:

Formula	Result

x=100
setunits("x","W")
y=getdbunits(x)
	y=15000

x=[100,200]
setunits("x","V")
y=getdbunits(x)
	y=13000

Compatibility
scalar, vector, arrays

See Also
getunits (users)

getindep

Syntax
y = getindep(x)

Definition
Returns a string with the name(s) of the independent variable(s). x is the variable to check.

Examples:

Formula	Result
n=getindep(S)	if S is a linear analysis result this will usually return "Linear_Data\Eqns\VarBlock\F" (the longname of F)
n=getindep(VPORT)	in a HARBEC analysis this will return "HbData\Eqns\VarBlock\Freq" - the Frequency vector

Compatibility
Swept vectors, arrays

See Also
setindep (users)

getindepvalue

Syntax
y = getindepvalue(x)

Definition
Returns the value of the independent variable of a variable x. x must have only one independent value.

Examples:

Formula	Result
y=getindepvalue(S)	If S is a variable from a linear analysis, then y will be a vector of the frequences used.

Compatibility
swept vectors, arrays

See Also

getindep (users)

getunits

Syntax
y = getunits(x)

Definition
Returns an integer corresponding to the units of a variable x. This integer may be used by setunits.

Examples:

Formula	Result

z = 1
setunits("z" , "V")
y = getunits(z)
	y = 9001

z = 1
setunits("z" , "mil")
y = getunits(z)
	y = 6002

z = 1
setunits("z" , "H")
y = getunits(z)
	y = 4003

Compatibility
Numeric scalars, vectors, arrays

See Also
setunits (users)

hb_gain

Syntax
hb_gain(Pin, SpectrOut, FreqIndexIM, IndexS)

Definition
hb_gain calculates the complex gain.

Pin is the voltage (or power) amplitude of the input signal.

SpectrOut is the voltage (or power) output spectrum.

FreqIndexIM is a HB-analysis dataset variable and contains a table of the intermodulation (IM) indexes.

IndexS is the IM-index of the output spectral component.

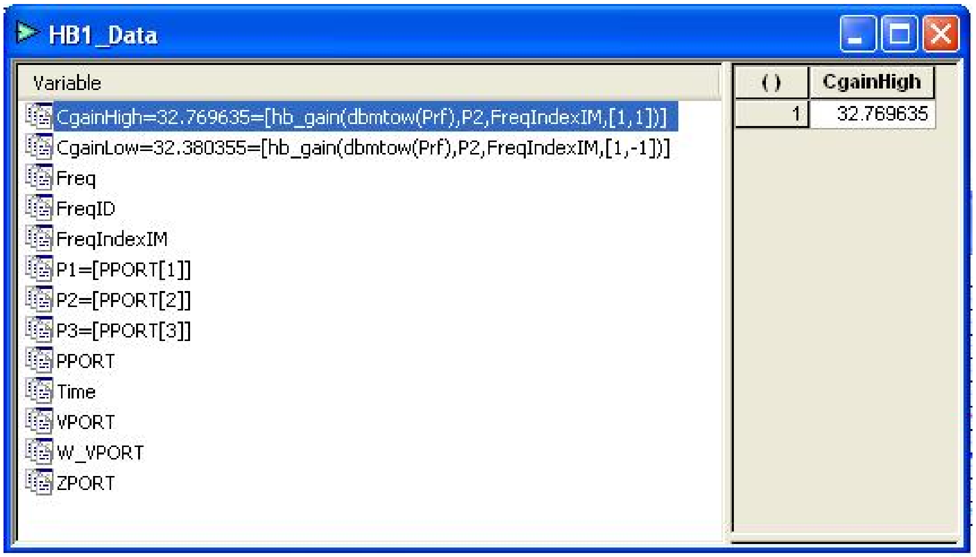
This function is the same as doing:
hb_getspcomp(SpectrOut, FreqIndexIM, IndexS) / Pin

> ℹ Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
The hb_gain function is demonstrated in the following example workspaces: Double Balance Gilbert cell mixer.wsx and Gilbert cell BJT mixer.wsx. Both files are located in the Examples\Mixers folder of your Genesys directory. The functions are used in the Harmonic

Balance dataset.

This dataset can be found in the Double Balance Gilbert cell mixer.wsx example:



Variable

- CgainHigh=32.769635=[hb_gain(dbmtow(Prf),P2,FreqIndexIM,[1,1])]
- CgainLow=32.380355=[hb_gain(dbmtow(Prf),P2,FreqIndexIM,[1,-1])]
- Freq
- FreqID
- FreqIndexIM
- P1=[PPORT[1]]
- P2=[PPORT[2]]
- P3=[PPORT[3]]
- PPORT
- Time
- VPORT
- W_VPORT
- ZPORT

()	CgainHigh
1	32.769635

Compatibility
Arrays

See Also
hb_getspcomp (users)

hb_getspcomp

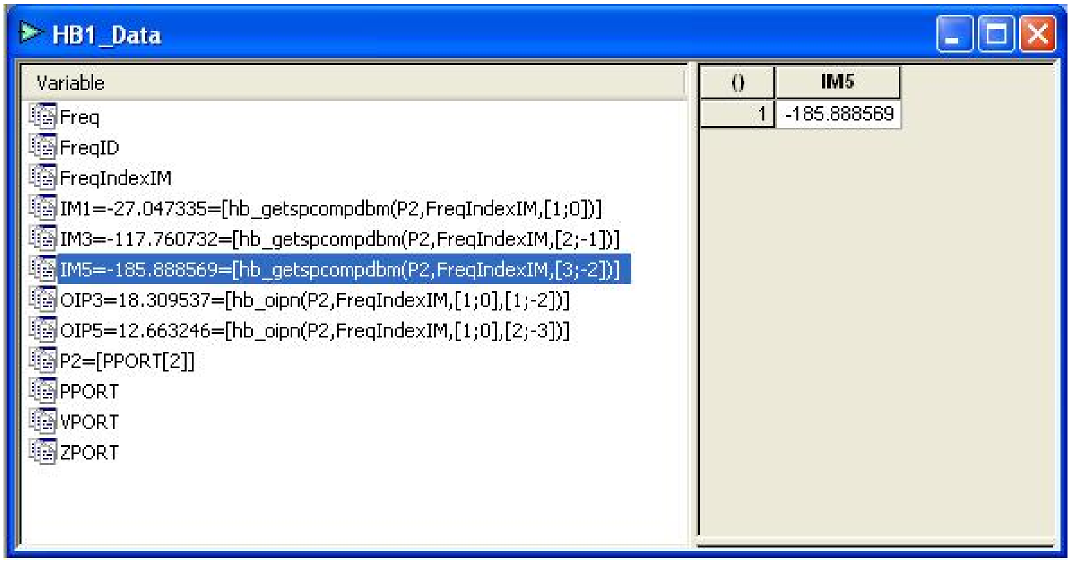
Syntax
hb_getspcomp(Spectr, FreqIndexIM, IndexS)

Definition
hb_getspcomp returns the complex amplitude of the spectral component with IM-Index *IndexS* from spectrum *Spectr* . *FreqIndexIM* is a HB-analysis dataset variable and contains a table of the intermodulation (IM) indexes. Returned value is in Watts.

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
This example uses hp_getspcompdbm, however, hb_getspcompdbm is the same as dbm(hb_getspcomp(Spectr, FreqIndexIM, IndexS)).
Which just calculates the return value in dBm instead of watts.

This dataset can be found in the Amplifier IPn Calculation.wsx workspace located in the Examples\Amplifiers folder of your Genesys directory.



Variable

- Freq
- FreqID
- FreqIndexIM
- IM1=-27.047335=[hb_getspcompdbm(P2,FreqIndexIM,[1;0])]
- IM3=-117.760732=[hb_getspcompdbm(P2,FreqIndexIM,[2;-1])]
- IM5=-185.888569=[hb_getspcompdbm(P2,FreqIndexIM,[3;-2])]
- OIP3=18.309537=[hb_oipn(P2,FreqIndexIM,[1;0],[1;-2])]
- OIP5=12.663246=[hb_oipn(P2,FreqIndexIM,[1;0],[2;-3])]
- P2=[PPORT[2]]
- PPORT
- VPORT
- ZPORT

()	IM5
1	-185.888569

To get the amplitude, look at the FreqIndexIM data to find a row which multi-index (or its conjugate) is equal to the multi index of the spectral component.

For the example referenced above. If the component multi-index *IndexS* = [3;-2], then it corresponds to row # 4 of the FreqIndexIM. So the one dimensional index of the spectral component is 4. Then the value of the spectral component corresponds to the 4th index of the spectrum array P2 and is passed in as *Spectr* .

Compatibility
Arrays

See Also
hb_getspcompdbm (users)

hb_getspcompdbm

Syntax
hb_getspcompdbm(Spectr, FreqIndexIM, IndexS)

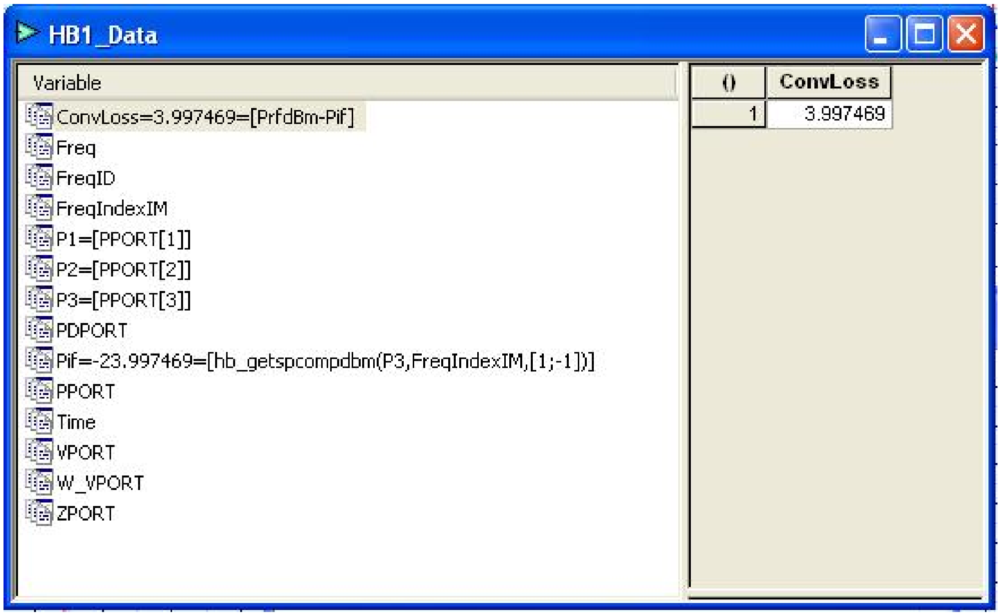
Definition
hb_getspcompdbm returns the complex amplitude of the spectral component with IM-Index *IndexS* from spectrum *Spectr* . *FreqIndexIM* is a HB-analysis dataset variable and contains a table of the intermodulation (IM) indexes. Returned value is in dBm

This function is the same as doing:
dbm(hb_getspcomp(Spectr, FreqIndexIM, IndexS))

> Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
The hb_getspcompdbm function is demonstrated in the following example workspaces: Amplifier IPn Calculations.wsx, SiGe BFP620 Amp.wsx, Diode Ring Mixer.wsx, Gilbert cell BJT mixer.wsx, and Noise Diode Ring Mixer.wsx. These files are found in Examples\Amplifers, Examples\Mixers, or Examples\Nonlinear Noise folders of your Genesys Directory. The functions are used in the Harmonic Balance dataset.

This dataset can be found in the Diode Ring Mixer.wsx.

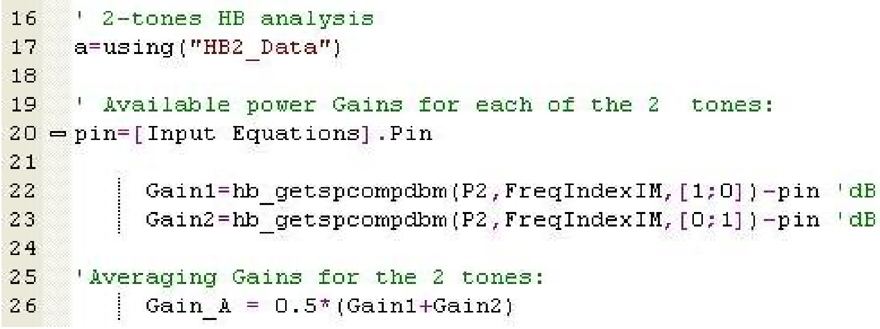


Variable

- ConvLoss=3.997469=[PrfdBm-Pif]
- Freq
- FreqID
- FreqIndexIM
- P1=[PPORT[1]]
- P2=[PPORT[2]]
- P3=[PPORT[3]]
- PDPORT
- Pif=-23.997469=[hb_getspcompdbm(P3,FreqIndexIM,[1;-1])]
- PPORT
- Time
- VPORT
- W_VPORT
- ZPORT

()	ConvLoss
1	3.997469

SiGe BFP620 Amp.wsx uses the function inside the equation called Output Equations.



```
' 2-tones HB analysis
a=using("HB2_Data")

' Available power Gains for each of the 2  tones:
pin=[Input Equations].Pin

    Gain1=hb_getspcompdbm(P2,FreqIndexIM,[1;0])-pin 'dB
    Gain2=hb_getspcompdbm(P2,FreqIndexIM,[0;1])-pin 'dB

'Averaging Gains for the 2 tones:
    Gain_A = 0.5*(Gain1+Gain2)
```

Compatibility
Arrays

See Also

- *hb_getspcomp* (users)
- *dbm* (users)

hb_iipn

Syntax
hb_iipn(SpectrPout, FreqIndexIM, IndexS1, IndexS2, PindBm)

Definition
hb_iipn calculates the input intercept point of the two components *IndexS1* and *IndexS2* of the power spectrum *SpectrPout. FreqIndexIM* contains a table of the intermodulation (IM) indexes. *PindBm* is the input signal power in dBm.

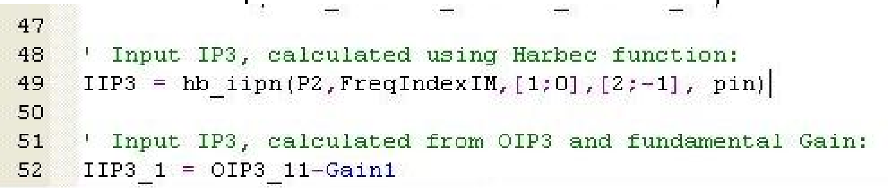
This function is the same as doing:
hb_ipn(SpectrPout, FreqIndexIM, IndexS1, IndexS2) + PindBm

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:

The hb_iipn function is demonstrated in the following workspace: SiGe BFP620 Amp.wsx. This file can be found in the Examples\Amplifiers folder of your Genesys directory.

SiGe BFP620 Amp.wsx uses the function in the equation called Output Equations:



```

' Input IP3, calculated using Harbec function:
IIP3 = hb_iipn(P2,FreqIndexIM,[1;0],[2;-1], pin)

' Input IP3, calculated from OIP3 and fundamental Gain:
IIP3_1 = OIP3_11-Gain1
```

Compatibility

Arrays

See Also

hb_ipn (users)

hb_ipn

Syntax

hb_ipn(SpectrPout, FreqIndexIM, IndexS, IndexIM)

Definition

hb_ipn is the common part of the input an output IPn functions (IIPn = hb_iipn and OIPn = hb_oipn). hb_ipn is not usually used for measurement, it's an internal function for hb_iipn and hb_oipn.

hb_ipn calculates the value:
IPn = (Psig-Pim)/(Nim-Nsig)

where

Nim, Nsig are the harmonic orders of the intermodulation (IM) spectral component, and the harmonic order of the signal spectral component.

Nsig = abssum(IndexS)

Nim = abssum(IndexIM)

Pim, Psig are the power values of the IM and signal spectral components calculated in dBm..

Psig = hb_getspcompdbm(SpectrPout, FreqIndexIM, IndexS)

Pim = hb_getspcompdbm(SpectrPout, FreqIndexIM, IndexIM)

The output and input IPn functions differ only by a constant value.

IIPn = Pin + IPn

OIPn = Pout + IPn = IIPn + Gain[dB]

where Gain is the power gain in dBm. In both cases IPn are calculated for the same spectrum, the output spectrum of the DUT (device under test).

> ℹ Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:

See examples for hb_iipn or hb_oipn.

Compatibility
Arrays

See Also

- *hb_iipn* (users)
- *hb_oipn* (users)
- *hb_getspcompdbm* (users)
- *abssum* (users)

hb_LargeS

> ℹ Note. The obsolete function. Not recommended for new projects. Instead of of this function, please use larges.

Syntax
hb_LargeS(Vin, Vout, sameport)

Definition
hb_LargeS returns the calculated Large Signal S-parameter (LS-parameter) for 1-tone HB-analysis.

Vin is the voltage amplitude of the input port.

Vout is the complex amplitude of voltage for the selected harmonic at the output port. It is the complex amplitude of the output spectral component.

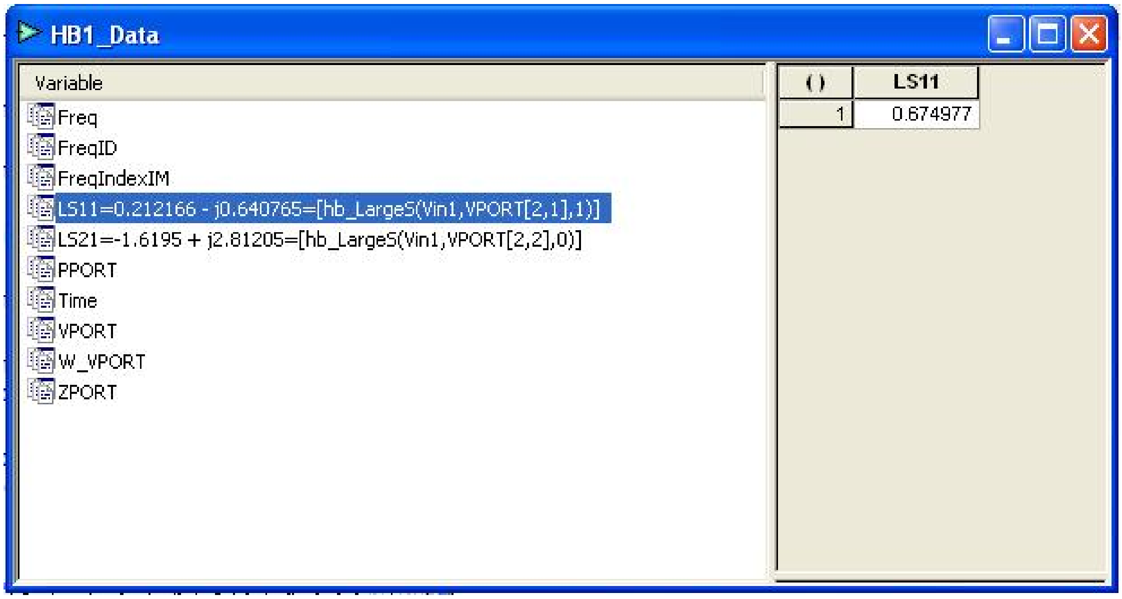
Vout = hb_getspcomp(Spectr, FreqIndexIM, IndexS)

sameport = 1 if the input and output ports are the same, and *sameport* = 0 if the input and output ports are not the same.

> ℹ Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
The hb_LargeS function is demonstrated in the following workspaces: Large Signal S Parameters.wsx and Large Signal S Param LInear Test.wsx. Both workspaces are located in the Examples\Amplifiers folder of your Genesys directory. The functions are used in the HB datasets of the workspaces.

This is the dataset from the Large Signal S Parameters.wsx:



Variable

- Freq
- FreqID
- FreqIndexIM
- LS11=0.212166 - j0.640765=[hb_LargeS(Vin1,VPORT[2,1],1)]
- LS21=-1.6195 + j2.81205=[hb_LargeS(Vin1,VPORT[2,2],0)]
- PPORT
- Time
- VPORT
- W_VPORT
- ZPORT

( )	LS11
1	0.674977

Compatibility
Arrays

See Also

- *hb_largesmix* (users)
- *hb_getspcomp* (users)

hb_LargeSmix

Note. This is an obsolete function. Not recommended for new projects. Instead of this function, please use largesmix.

Syntax
hb_LargeSmix(Vin, SpectrVout, sameport, FreqIndexIM, IndexOut)

Definition
hb_LargeSmix returns the Large Signal S-parameter (LS-parameter). This function is used for multi-tone HB-analysis, or for frequency conversion. hb_LargeSmix calculates the complex spectrum and then finds the LS-parameter.

This function is the same as doing:

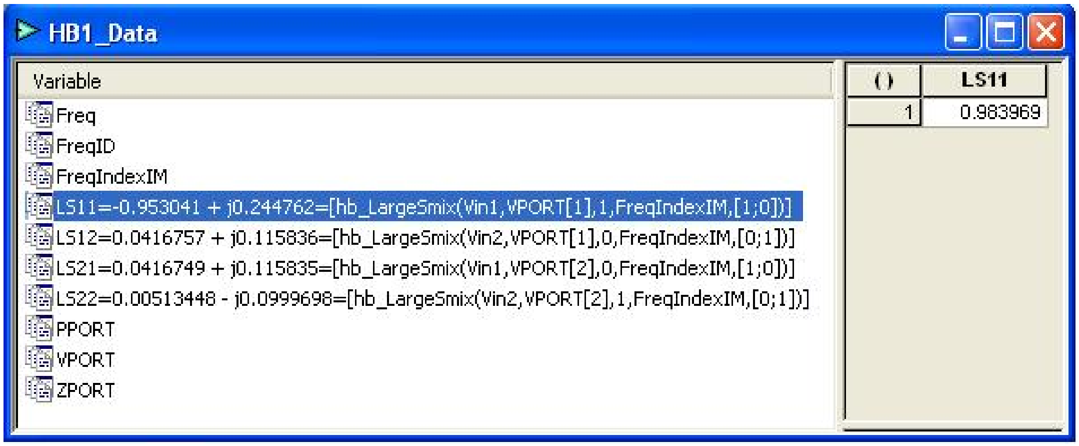
Vout = hb_getspcomp(SpectrVout, FreqIndexIM, IndexOut)

hb_LargeS(Vin, Vout, sameport)

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
The hb_LargeSmix function is demonstrated in the following workspace: Large Signal S Param Linear Test(2 tones).wsx. This files is located in the Examples\Amplifiers folder of your Genesys directory.

The function is used inside the HB dataset:



```
Variable
Freq
FreqID
FreqIndexIM
LS11=-0.953041 + j0.244762=[hb_LargeSmix(Vin1,VPORT[1],1,FreqIndexIM,[1;0])]
LS12=0.0416757 + j0.115836=[hb_LargeSmix(Vin2,VPORT[1],0,FreqIndexIM,[0;1])]
LS21=0.0416749 + j0.115835=[hb_LargeSmix(Vin1,VPORT[2],0,FreqIndexIM,[1;0])]
LS22=0.00513448 - j0.0999698=[hb_LargeSmix(Vin2,VPORT[2],1,FreqIndexIM,[0;1])]
PPORT
VPORT
ZPORT
```

()	LS11
1	0.983969

See Also

hb_larges (users)

hb_oipn

Syntax

hb_oipn(SpectrPout, FreqIndexIM, IndexS1, IndexS2)

Definition

hb_oipn calculates the output intercept point of the two components IndexS1 and IndexS2 of the power spectrum SpectrPout.

This function is the same as doing:

hb_ipn(SpectrPout, FreqIndexIM, IndexS1, IndexS2) + PoutdBm

'PoutdBm is the output signal power in dBm.

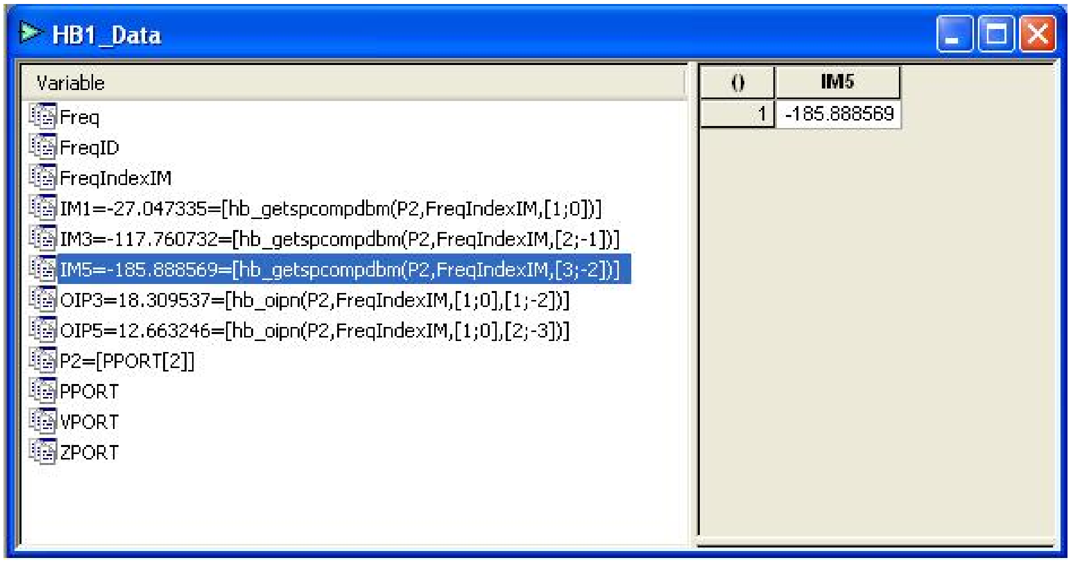
PoutdBm = hb_getspcompdbm(SpectrPout, FreqIndexIM, IndexS1)

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:

The hb_oipn function is demonstrated in the following workspaces: Amplifier IPn Calculation.wsx and SiGe BFP620 Amp.wsx. Both files are located in the Examples\Amplifiers folder of your Genesys directory.

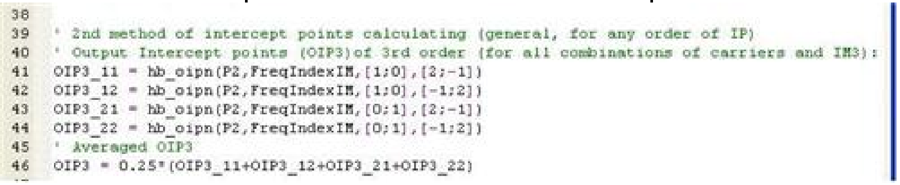
Amplifier IPn Calculation.wsx uses the function in the HB dataset:



Variable
Freq
FreqID
FreqIndexIM
IM1=-27.047335=[hb_getspcompdbm(P2,FreqIndexIM,[1;0])]
IM3=-117.760732=[hb_getspcompdbm(P2,FreqIndexIM,[2;-1])]
IM5=-185.888569=[hb_getspcompdbm(P2,FreqIndexIM,[3;-2])]
OIP3=18.309537=[hb_oipn(P2,FreqIndexIM,[1;0],[1;-2])]
OIP5=12.663246=[hb_oipn(P2,FreqIndexIM,[1;0],[2;-3])]
P2=[PPORT[2]]
PPORT
VPORT
ZPORT

()	IM5
1	-185.888569

SiGe BFP620 Amp uses the function in the equation called Output Equations:



```
' 2nd method of intercept points calculating (general, for any order of IP)
' Output Intercept points (OIP3)of 3rd order (for all combinations of carriers and IM3):
OIP3_11 = hb_oipn(P2,FreqIndexIM,[1;0],[2;-1])
OIP3_12 = hb_oipn(P2,FreqIndexIM,[1;0],[-1;2])
OIP3_21 = hb_oipn(P2,FreqIndexIM,[0;1],[2;-1])
OIP3_22 = hb_oipn(P2,FreqIndexIM,[0;1],[-1;2])
' Averaged OIP3
OIP3 = 0.25*(OIP3_11+OIP3_12+OIP3_21+OIP3_22)
```

Compatibility
Arrays

See Also

- *hb_ipn* (users)
- *hb_getspcompdbm* (users)

hb_spurious

Syntax
hb_spurious(SpectrVout, FreqIndexIM, ExcludeSignals)

Definition
hb_spurious calculates and returns the max amplitude of spurious spectral components in spectrum *SpectrVout*. Used for n-Tone Harmonic Balance Analysis, where n > 0.

SpectrVout is the complex spectrum array.

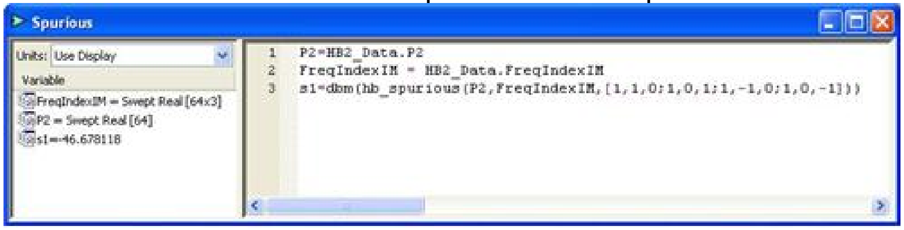
FreqIndexIM is the intermodulation (IM) index matrix.

ExcludeSignals is an integer index matrix of index vectors. This matrix defines the signal components in the spectrum.

> Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:
The hb_spurious function can be found in the following workspace: Double Balance Gilbert cell mixer.wsx. This file is located in the Examples\Mixers folder of your Genesys directory.

The function is used in the Equation called Spurious:



Units: Use Display

Variable

FreqIndexIM = Swept Real [64x3]

P2 = Swept Real [64]

s1=-46.678118

```
P2=HB2_Data.P2
FreqIndexIM = HB2_Data.FreqIndexIM
s1=dbm(hb_spurious(P2,FreqIndexIM,[1,1,0;1,0,1;1,-1,0;1,0,-1]))
```

Compatibility

Arrays

hb_totalsp

Syntax

hb_totalsp(Sin, Freq, eps)

Definition

The hb_totalsp calculates total amplitude for spectral components having the same frequency and sets all of them to the same total amplitude. It creates a spectrum without multiple points at the same frequency point where the amplitude is what the spectrum analyzer measures.

If a circuit has a signal with two frequencies F1 = 100 and F2 = 200 MHz, and the analysis uses multi-dimensional (n-tones) FFt, then the solution spectrum may have multiple components having the same frequency.

(F1) , (F2 - F1) , (2 * F2 - 3 * F1), (3 * F2 - 5 * F1), etc.

All the spectral components have the frequency 100 MHz, but are calculated as independent spectral components.

hb_totalsp is used when Harbec calculates spectrum using n-dim FFT. In this case spectral components with different harmonic index vectors may produce spectral component with the same frequency.

For example, if the input signal has 2 tones 10 and 11 MHz, then the 2 different spectral components created by different order nonlinear products: [2;-1] and [9;-9] have the same frequency 9 MHz:

|2*10 - 1*11| = 9

|9*10 - 9*11| = 9

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping of the Parameter Sweep Properties dialog window must be set.

Examples:

The function hb_totalsp is demonstrated in the following workspace: TotalSpectrum.wsx. This workspace is located in the Examples\Amplifiers folder of your Genesys directory.

This use of the function can be found in the Table1.

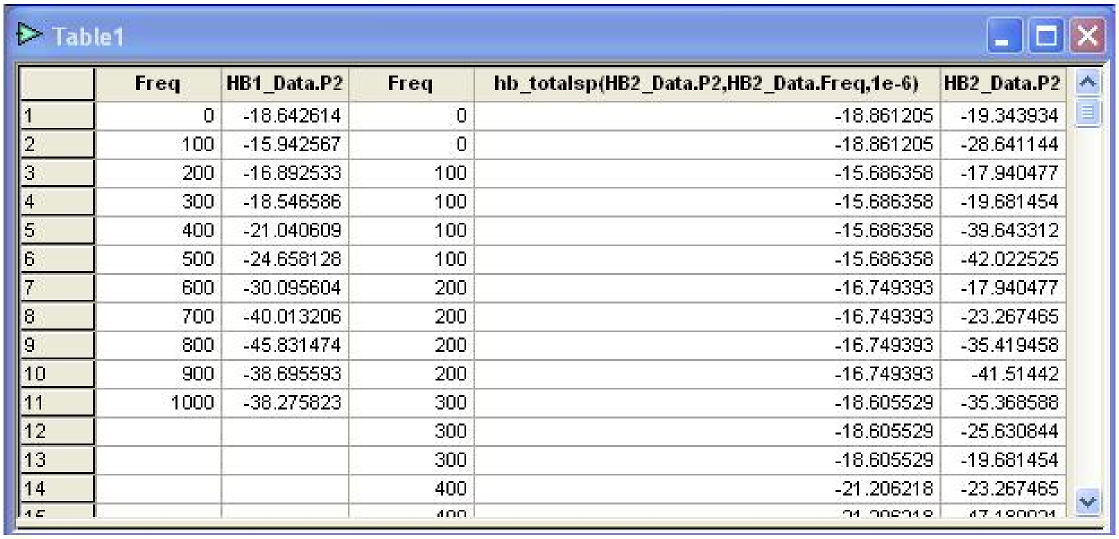
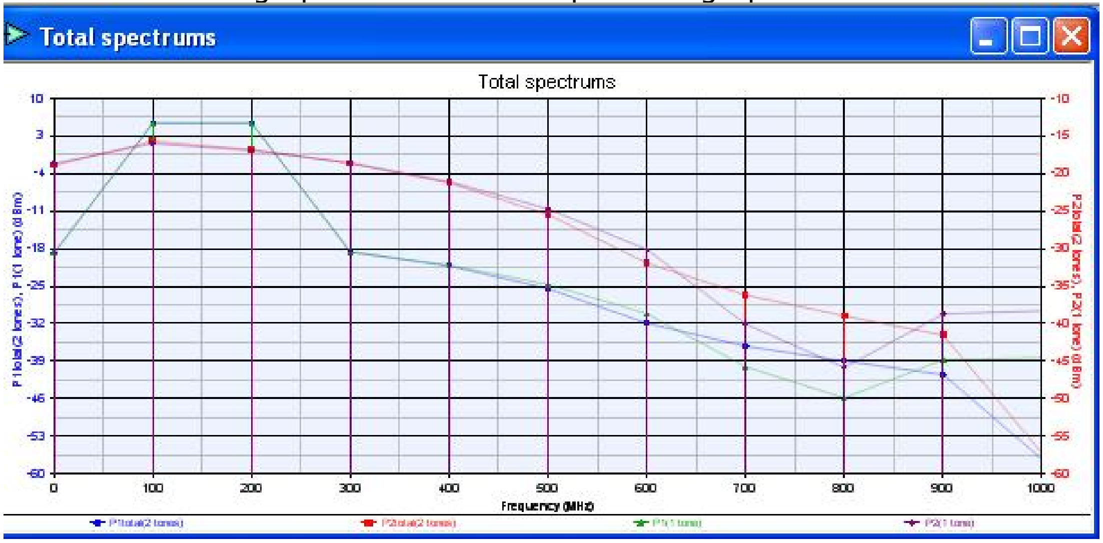


Table1

	Freq	HB1_Data.P2	Freq	hb_totalsp(HB2_Data.P2,HB2_Data.Freq,1e-6)	HB2_Data.P2
1	0	-18.642614	0	-18.861205	-19.343934
2	100	-15.942567	0	-18.861205	-28.641144
3	200	-16.892533	100	-15.686358	-17.940477
4	300	-18.546586	100	-15.686358	-19.681454
5	400	-21.040609	100	-15.686358	-39.643312
6	500	-24.658128	100	-15.686358	-42.022525
7	600	-30.095604	200	-16.749393	-17.940477
8	700	-40.013206	200	-16.749393	-23.267465
9	800	-45.831474	200	-16.749393	-35.419458
10	900	-38.695593	200	-16.749393	-41.51442
11	1000	-38.275823	300	-18.605529	-35.368588
12			300	-18.605529	-25.630844
13			300	-18.605529	-19.681454
14			400	-21.206218	-23.267465
15			400	-21.206218	-47.180021

The results were graphed in the Total spectrum graph.



See the workspace for more detail.

Compatibility

Arrays

hb_transgain

Syntax

hb_transgain(SpectrIn, SpectrOut, FreqIndexIM, IndexIn, IndexOut)

Definition

hb_transgain calculates and returns the transducer gain from the input spectrum component *SpectrIn* with a intermodulation index (IM-index) *IndexIn* to the output spectrum component *SpectrOut* with IM-index *IndexOut*

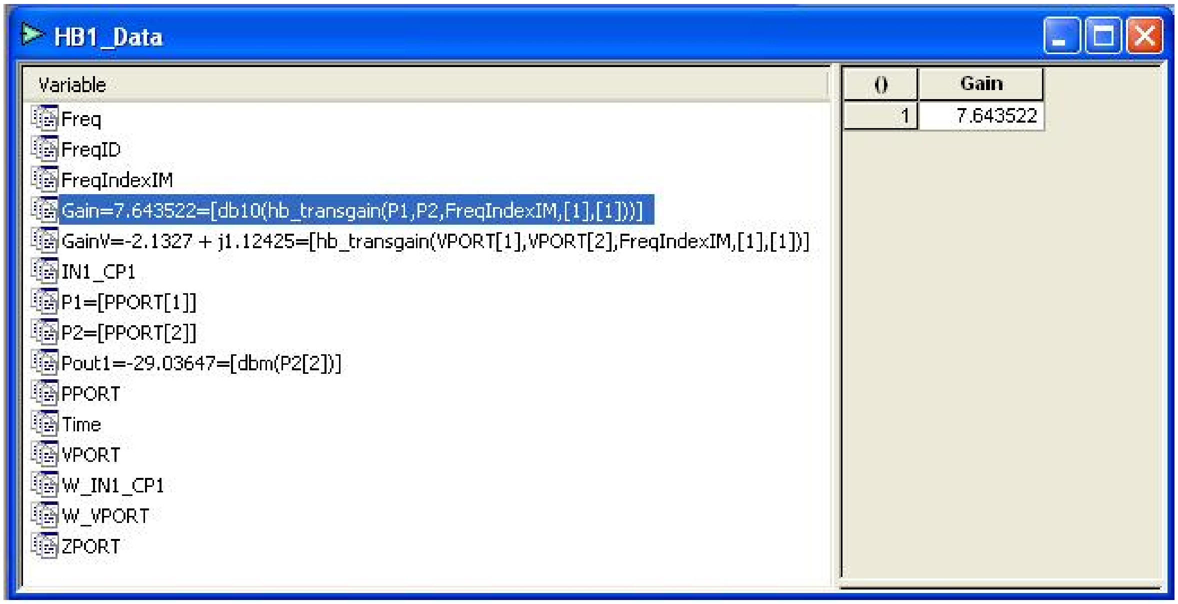
> Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping" of the Parameter Sweep Properties dialog window must be set.

Examples:

The hb_transgain function is demonstrated in the following example workspace: Amplifier Gain Compression.wsx. This file is located in the Example\Amplifiers folder of your

Genesys directory.

The function is used in the HB dataset:



Variable

- Freq
- FreqID
- FreqIndexIM
- Gain=7.643522=[db10(hb_transgain(P1,P2,FreqIndexIM,[1],[1]))]
- GainV=-2.1327 + j1.12425=[hb_transgain(VPORT[1],VPORT[2],FreqIndexIM,[1],[1])]
- IN1_CP1
- P1=[PPORT[1]]
- P2=[PPORT[2]]
- Pout1=-29.03647=[dbm(P2[2])]
- PPORT
- Time
- VPORT
- W_IN1_CP1
- W_VPORT
- ZPORT

()	Gain
1	7.643522

Compatibility
Arrays

See Also
hb_transgaindb (users)

hb_transgaindb

Syntax
hb_transgaindb(SpectrIn, SpectrOut, FreqIndexIM, IndexIn, IndexOut)

Definition
hb_transgaindb calculates and returns the transducer gain from the input spectrum component *SpectrIn* with IM-index *IndexIn* to the output spectrum component *SpectrOut* with IM-index *IndexOut* . The returned values are expressed in decibels as 10 * log(x).

This function is the same as doing:
db10 (hb_transgain(SpectrIn, SpectrOut, FreqIndexIM, IndexIn, IndexOut))

> Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping" of the Parameter Sweep Properties dialog window must be set.

Examples:
See the example for hb_transgain.

Compatibility
Arrays

See Also

- *hb_transgain* (users)
- *db10* (users)

hermitian

Syntax
y = hermitian(x)

Definition
hermitian returns the conjugate transpose of a matrix or swept matrix x.

Examples:

Formula	**Result**	**Or**

x = [1 , 2 , 2 ; 1 , 2 , 1]
y = hermitian(x)
	y = [1 , 1 ; 2 , 2 ; 2 , 1]	

y = hermitian([complex(2 , 4) , complex(1 , -3)])	y = [4.472 ; 3.162]	y[1] = 2 - j4
		y[2] = 1 +j3

y = hermitian([complex(2 , -2) , -5])	y = [2.828 , -5]	y[1] = 2 + j2
		y[2] = -5

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *conj* (users)
- *transpose* (users)

histogram

Syntax
y = histogram(x)
y = histogram(x, NumBins)
y = histogram(x, NumBins, RangeMin, RangeMax)
y = histogram(x, strIndepToCreate)
y = histogram(x, strIndepToCreate, NumBins, RangeMin, RangeMax)

Definition
This function creates bins for the data in x and returns the number of parts in each container. The NumBins, RangeMin, and RangeMax parameters are optional. NumBins, the number of bins to create, defaults to 10. RangeMin and RangeMax, the minimum and maximum values for the bins, default to the minimum and maximum value in x. The return value can be directly plotted on a rectangular graph and will produce a histogram plot.

If the string value strIndepToCreate is not specified, this function creates a new variable called __HistIndep that is the independent value of the returned value which contains the bin values. If strIndepToCreate is specified, the independent value with that name is created.

Compatibility
Vectors, Arrays

identity

Syntax
y = identity(n)

Definition
identity returns an n by n identity matrix. n must be a positive number.

Examples:

Formula	Result
y = identity(2)	y = [1 , 0 ; 0 , 1]
y = identity(3)	y = [1 , 0 , 0 ; 0 , 1 , 0 ; 0 , 0 , 1]

Compatibility

Positive Numeric scalars

iff

Syntax

z = iff(bool , x , y)

Definition

iff returns x if bool is true or y if bool is false. bool is a boolean expression represented as a 1 (true) or 0 (false). Works with numbers and column vectors.

Examples:

Formula	Result
z = iff(1 , 10 , 20)	z = 10
z = iff(0 , 25 , -5)	z = -5
z = iff(0 , [0 ; 1] , [0 ; 2])	z = [0 ; 2]

Compatibility

Numeric scalars, Vectors

ifft

Syntax

ifft(data, len)

Definition

Inverse Discrete Fourier Transform (IDFT) of data. Computed with IFFT algorithm when possible. The parameter len is the IFFT length and is optional.

Examples:

The following example code is taken from the fft example:

```
fft_len = 1024    '  length of the FFT
fs = 8000         '  8000 Hz sampling rate
T = 1/fs          '  sample time
L = 1000          '  length of signal
t = \(0:\(L\-1\)\)*T   '  time vector
' x will be the sum of two sinusoids:
' one at 400 Hz and one at 1500 Hz
x = 0.5*cos\(2*PI*400*t\) + cos\(2*PI*1500*t\)
X = fft\(x, fft_len\)  '  spectrum of x
X = X\[1:\(fft_len/2\)\] ' we only care about single side\-band  \(the rest is redundant\)
f = fs/2 * \(0:\(2/fft_len\):1\)
setindep\("X","f"\)
```

If the following lines of code are now added:

y = ifft(X, fft_len)

then y and x would be identical.

Compatibility

Dataset

See Also
ifft (users)

im

Syntax
y = im(x)

Definition
im returns the imaginary part of a complex number x. Same as imag function

Examples:

Formula	Result
y = im(complex(2 , -5))	y = -5
y = im([complex(10 , 1) , complex(12 , -6)])	y = [1 , -6]

b = [complex(20 , 3) ; complex(1 , 2)]
y = im(b)
	y = [3 ; 2]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
imag

imag

Syntax
y = imag(x)

Definition
imag returns the imaginary part of a complex number x. Same as im function.

Examples:

Formula	Result
y = imag(complex(2 , -5))	y = -5
y = imag([complex(10 , 1) , complex(12 , -6)])	y = [1 , -6]

b = [complex(20 , 3) ; complex(1 , 2)]
y = imag(b)
	y = [3 ; 2]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
im (users)

integrate

Syntax
z = integrate(dep, indep, start, stop)

Definition
Returns the numerical integration of the dependent variable dep with respect to the independent variable indep going from index start to index stop. The start index must be at least 1. The variables dep, and indep should be column vectors.

Examples:

Formula	**Result**
x = [1;2;3;4;5], y = 2 * x, z = integrate(x , y , 1 , 5)	z = 20
x = [2 ; 4 ; 6 ; 8 ; 10], y = x^2, z = integrate(x , y , 2 , 4)	z = 248
x = [0 .1 ; 1 ; 10 ; 100 ; 1000], y = log(x), z = integrate(x , y , 3 , 5)	z = 110

Compatibility
Vectors

See Also
derivative (users)

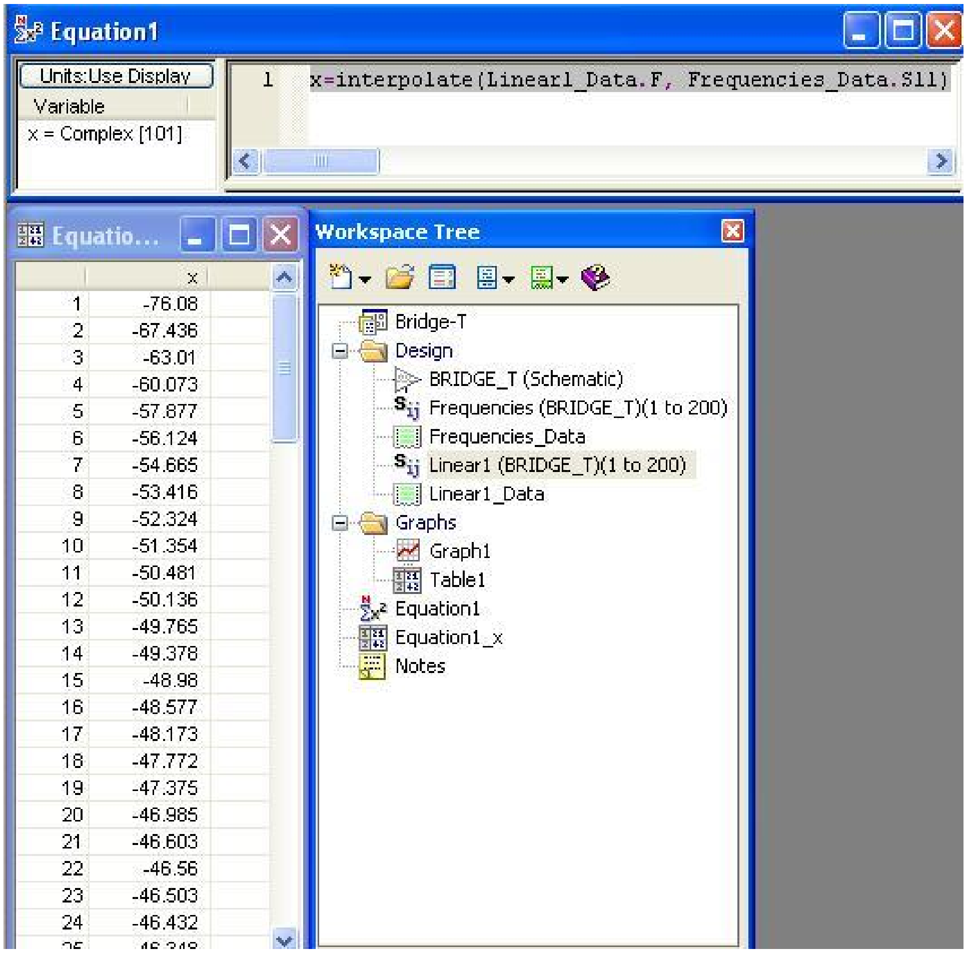
interpolate

Syntax
y = interpolate(new_indep, var)

Definition
interpolate produces an array where the independent value of var is changed to new_indep, interpolating as necessary. The parameter new_indep should be a vector or an array, and the parameter var is a number. Make sure the var you are passing in has an independent value. The S from a linear analysis would be a good example of a variable to user for the var parameter since it has the independent value of F.

Examples:
This example was made using the Bridge-T.wsx example.

Add a new linear analysis and set the frequency range the same as the "Frequencies" Analysis. Change the Number of Points to sweep to 101.Enter x=interpolate(Linear1_Data.F, Frequencies_Data.S11) in an Equations window and view the results as a table.



Variable

x = Complex [101]

```
x=interpolate(Linear1_Data.F, Frequencies_Data.S11)
```

	x
1	-76.08
2	-67.436
3	-63.01
4	-60.073
5	-57.877
6	-56.124
7	-54.665
8	-53.416
9	-52.324
10	-51.354
11	-50.481
12	-50.136
13	-49.765
14	-49.378
15	-48.98
16	-48.577
17	-48.173
18	-47.772
19	-47.375
20	-46.985
21	-46.603
22	-46.56
23	-46.503
24	-46.432

This demonstrates how the interpolate function can be used.

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- Numeric scalars
- *vector* (users)
- *array* (users)

intersect

Syntax
z = intersect(x , y)

Definition
intersect returns a vector z that contain the intersecting values of the two vectors x and y.

Examples:

Formula	Result

x = [1 , 2 , 3 , 4 , 5]
y = [2 , 4 , 6 , 8 , 10]
z = intersect(x , y)
	z = [2 ; 4]

x = [40, 32 , 24 , 16 , 8]
y = [4 ; 8 ; 16 ; 32 ; 64]
z = intersect(x , y)
	z = [32 ; 16 ; 8]

x = [complex(1 , 1) , 25]
y = [8 , complex(1 , 1)]
z = intersect(x , y)
	z = 1.414

Compatibility

Numeric scalars, Vectors, Arrays

See Also

- Numeric scalars
- *vector* (users)
- *array* (users)

inverse

Syntax

y = inverse(x)

Definition

inverse returns the inverse of a matrix. The parameter x must be a non singular square matrix.

Examples:

Formula	Result

x = [2, 4 ; 6 , 8]
y = inverse(x)
	y = [-1 , 0.5 ; 0.75 , -0.25]

x = [4 , 6 , 8 ; 1 , 2 , 3 ; 0 , 3 , 5]
y = inverse(x)
	y = [-0.5 , 3 , -1 ; 2.5 , -10 , 2 ; -1.5 , 6 , -1]

Compatibility

Square Matrix

kurtosis

Syntax

y = kurtosis(x)
y = kurtosis(x, Flag)
y = kurtosis(x, Flag, iDim)

Definition

Returns the sample kurtosis of a vector x. Kurtosis is the fourth central moment of X divided by the fourth power of the standard deviation.

If Flag is 0 (default), kurtosis normalizes by N-1 where N is the sample size. If Flag is 1, kurtosis normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = kurtosis([3 ; 4 ; 8 ; 9])	y = 1.1479
y = kurtosis([1, 2, 3], 1)	y = 1.5

Compatibility

Numeric arrays

See Also

- *std* (users)
- *var* (users)
- *skewness* (users)

largeS

New function, returns the calculated Large Signal S-parameter (LS-parameter) for 1-tone HB-analysis, this function substitutes for the old function hb_larges.
It's much easy for using than hb_larges, because it uses a dataset name instead of list of dataset parameters and external variables.

Syntax

largeS(i, j, dataset)

Definition

largeS returns the calculated Large Signal S-parameter (LS-parameter) for 1-tone HB-analysis.

i - output port index

j - input port index

dataset - HB analysis dataset name

> Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping" of the Parameter Sweep Properties dialog window must be set.

Examples:

The largeS function is demonstrated in the following workspaces: Large Signal S Parameters.wsx and Large Signal S Param LInear Test.wsx. Both workspaces are located in the Examples\Amplifiers folder of your Genesys directory. The functions are used in the HB datasets of the workspaces.

LS11=* *largeS* *(1,1,"HB1_Data")

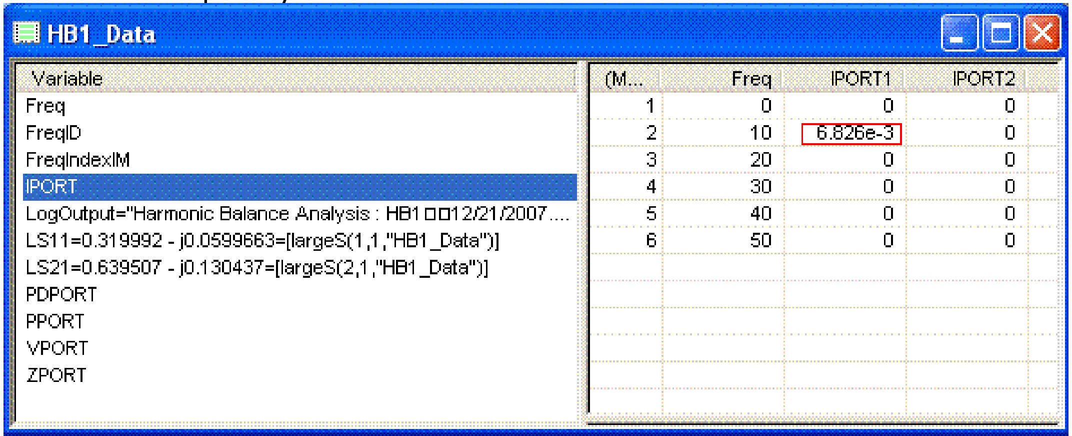
LS21=* *largeS* * (2,1,"HB1_Data")

LS12=* *largeS* * (1,2,"HB2_Data")

LS22=* *largeS* * (2,2,"HB2_Data")

> Note. The function ***largeS*** uses Harbec dataset variable ***IPORT*** , which is an array of spectrums of port excitation currents. It has nonzero components the only for "active" ports, which are ports with signal sources, which frequencies are the signal source frequencies.

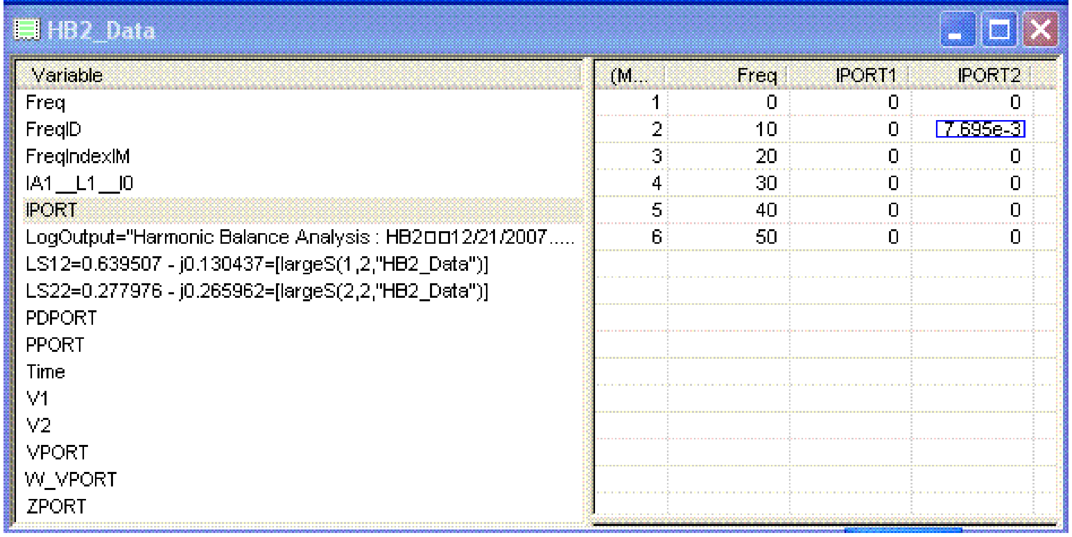
For example, for LS11, LS21 parameters, measured, when the 1st port is excited, the ***IPORT*** variable has only one nonzero component for the excited port Port1, which frequency is the port source frequency (number, marked with a red frame in HB1_Data dataset.) For LS22, LS12 parameters, measured, when the 2nd port is excited (HB2_Data), the ***IPORT*** variable has only one nonzero component, relative to Port2 and its source frequency 10 MHz:



HB1_Data

Variable
- Freq
- FreqID
- FreqIndexIM
- IPORT
- LogOutput="Harmonic Balance Analysis : HB1□□12/21/2007....
- LS11=0.319992 - j0.0599663=[largeS(1,1,"HB1_Data")]
- LS21=0.639507 - j0.130437=[largeS(2,1,"HB1_Data")]
- PDPORT
- PPORT
- VPORT
- ZPORT

(M...	Freq	IPORT1	IPORT2
1	0	0	0
2	10	6.826e-3	0
3	20	0	0
4	30	0	0
5	40	0	0
6	50	0	0



HB2_Data

Variable
- Freq
- FreqID
- FreqIndexIM
- IA1__L1__I0
- IPORT
- LogOutput="Harmonic Balance Analysis : HB2□□12/21/2007.....
- LS12=0.639507 - j0.130437=[largeS(1,2,"HB2_Data")]
- LS22=0.277976 - j0.265962=[largeS(2,2,"HB2_Data")]
- PDPORT
- PPORT
- Time
- V1
- V2
- VPORT
- W_VPORT
- ZPORT

(M...	Freq	IPORT1	IPORT2
1	0	0	0
2	10	0	7.695e-3
3	20	0	0
4	30	0	0
5	40	0	0
6	50	0	0

See Also

largeSmix (users)
largeSconv (users)
hb_larges (users)
hb_largesmix (users)

largeSconv

Syntax

largeSconv(i, j, iFreq, jFreq, dataset)

where

i - output port index

j - input port index

iFreq - index of output port frequency

jFreq - index of input port frequency

dataset - HB analysis dataset name

returns complex Sij-parameter.

Definition

largeSconv returns the Large Signal S-parameter (LS-parameter). This function is used for 1 and multi-tone HB-analysis.
It's used as an internal function for *largeS* (users) and *largeSmix* (users) functions, calculating LS parameters.

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping" of the Parameter Sweep Properties dialog window must be set.

For example, for 1-tone HB analysis the function calls are equivalent:

LS11 = largeSconv(1, 1, 2,2,"HB1_Data")

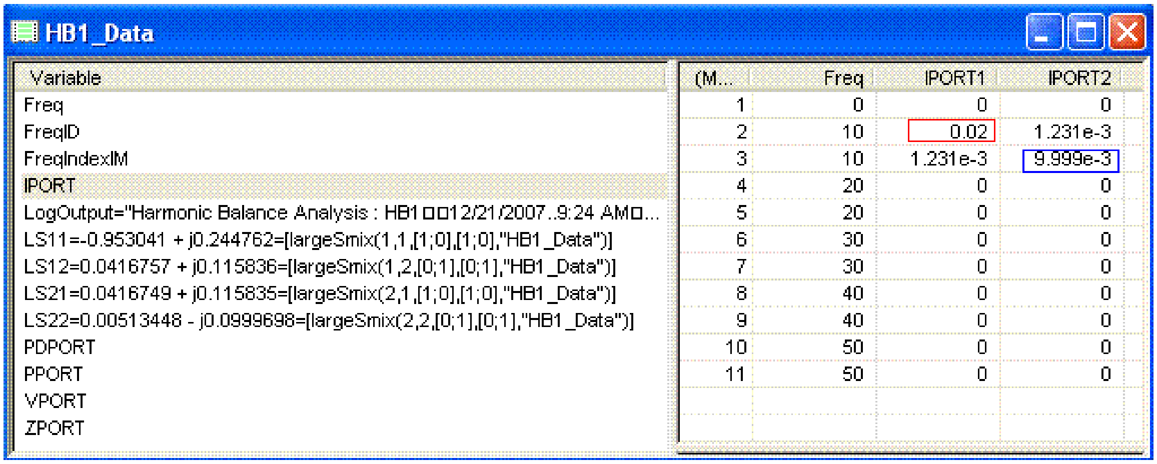
LS11 = largeS(1, 1,"HB1_Data")

LS11 = largeSmix(1, 1, [1], [1], "HB1_Data")

For 2-tone HB analysis, when Port1 is excited by one 10 MHz tone, having index =2 in the HB spectrum, the function calls are equivalent:

LS11 = largeSconv(1, 1, 2, 2,"HB1_Data")

LS11 = largeSmix(1, 1, [1; 0], [1; 0], "HB1_Data")



Variable
Freq
FreqID
FreqIndexIM
IPORT
LogOutput="Harmonic Balance Analysis : HB1 □□12/21/2007..9:24 AM□...
LS11=-0.953041 + j0.244762=[largeSmix(1,1,[1;0],[1;0],"HB1_Data")]
LS12=0.0416757 + j0.115836=[largeSmix(1,2,[0;1],[0;1],"HB1_Data")]
LS21=0.0416749 + j0.115835=[largeSmix(2,1,[1;0],[1;0],"HB1_Data")]
LS22=0.00513448 - j0.0999698=[largeSmix(2,2,[0;1],[0;1],"HB1_Data")]
PDPORT
PPORT
VPORT
ZPORT

(M...	Freq	IPORT1	IPORT2
1	0	0	0
2	10	0.02	1.231e-3
3	10	1.231e-3	9.999e-3
4	20	0	0
5	20	0	0
6	30	0	0
7	30	0	0
8	40	0	0
9	40	0	0
10	50	0	0
11	50	0	0

Note. Because **largeSmix** function parameter list includes indexes of spectrum component, found from FreqIndexIM dataset variable, this function is used as an internal function. It's not recommended using this function in user workspaces. Instead of it use functions ***largeS*** **(users),** ***largeSmix*** **(users).**

See Also

largeS (users), *largeSmix* (users)

largeSmix

New function, calculating Large Signal S-parameter (LS-parameter) for n-tone HB-analysis.
It substitutes the old function *hb_LargeSmix* (users).
It's much easy for using than *hb_LargeSmix* (users), because it uses a dataset name instead of list of dataset parameters and external variables.

Syntax

largeSmix(i, j, iIndexVector, jIndexVector, dataset)

where

i - output port index

j - input port index

iIndexVector[n] - vector of indexes of output component

jIndexVector[n] - vector of indexes of input component

dataset - HB analysis dataset name

returns complex Sij -parameter

Definition

largeSmix returns the Large Signal S-parameter (LS-parameter). This function is used with multi-tone HB-analysis

Note: To use HB analysis functions in sweeps of analyses, they must be defined directly in the analysis dataset, and the checkbox "Propagate All Variables When Sweeping" of the Parameter Sweep Properties dialog window must be set.

Examples:

The hb_LargeSmix function is demonstrated in the following workspace: Large Signal S Param Linear Test(2 tones).wsx. This files is located in the Examples\Amplifiers folder of your Genesys directory.

The function is used inside the HB dataset:

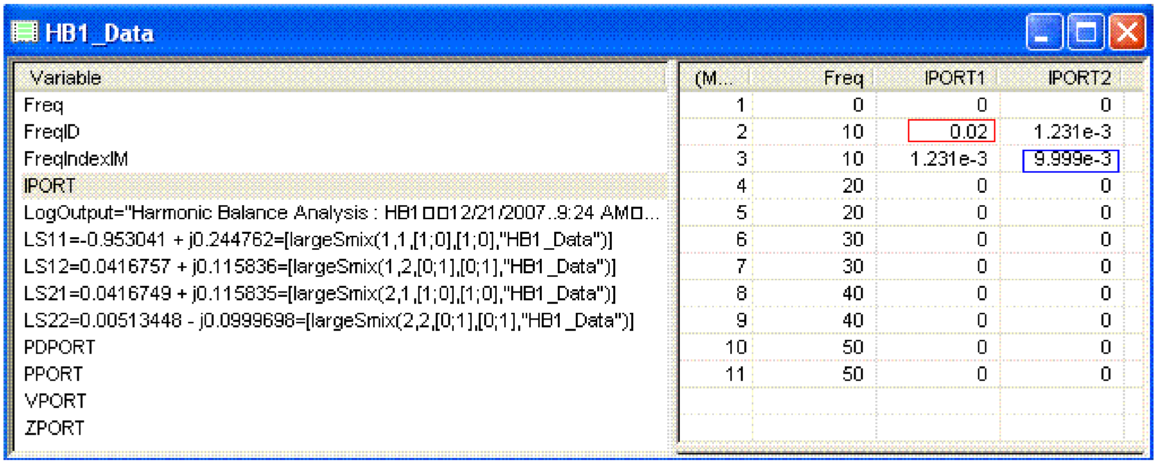
LS11=largeSmix(1,1,[1;0],[1;0],"HB1_Data")

LS12=largeSmix(1,2,[0;1],[0;1],"HB1_Data")

LS21=largeSmix(2,1,[1;0],[1;0],"HB1_Data")

LS22=largeSmix(2,2,[0;1],[0;1],"HB1_Data")

Note. The function ***largeSmix*** uses Harbec dataset variable ***IPORT*** , which is an array of spectrums of the port excitation currents. It has nonzero components the only for "active" ports - ports with signal sources, which frequencies are the signal source frequencies.
For example, for simultaneously measured all LS parameters of a *n* -port network, all ports are active and the ***IPORT*** variable has *n* nonzero components for every ports, which indexes are indexes of port source frequencies in the HB solution spectrum (in this example *n* = 2):



HB1_Data

Variable
Freq
FreqID
FreqIndexIM
IPORT
LogOutput="Harmonic Balance Analysis : HB1□□12/21/2007..9:24 AM□...
LS11=-0.953041 + j0.244762=[largeSmix(1,1,[1;0],[1;0],"HB1_Data")]
LS12=0.0416757 + j0.115836=[largeSmix(1,2,[0;1],[0;1],"HB1_Data")]
LS21=0.0416749 + j0.115835=[largeSmix(2,1,[1;0],[1;0],"HB1_Data")]
LS22=0.00513448 - j0.0999698=[largeSmix(2,2,[0;1],[0;1],"HB1_Data")]
PDPORT
PPORT
VPORT
ZPORT

(M...	Freq	IPORT1	IPORT2
1	0	0	0
2	10	0.02	1.231e-3
3	10	1.231e-3	9.999e-3
4	20	0	0
5	20	0	0
6	30	0	0
7	30	0	0
8	40	0	0
9	40	0	0
10	50	0	0
11	50	0	0

See Also

largeS (users), *largeSconv* (users), *hb_LargeS* (users), *hb_LargeSmix* (users)

lininterp

Syntax
w = lininterp(outputIndeps, inputIndeps, inputDeps)

Definition
lininterp returns the linear interpolation of the X values (outputIndeps) that you want interpolated Y values for. inputIndeps and inputDeps are the X and Y data points. The result of this function are the interpolated Y values for the specified X values from the given data points. inputIndeps and input Deps must have the same dimensions.

Examples:

Formula	**Result**

X = [2 , 5 , 9 , 13 , 17]
Y = [2 , 5 , 9 , 13 , 17]
z = [1 , 7 , 11 , 19]
w = lininterp(z , X , Y)
	w = [2 , 7 , 11 , 17]

X = [1 , 3 ; 7 , 9]
Y = [1 , 2 ; 4 , 5]
z = [2 , 4 , 6 , 8]
w = lininterp(z , X , Y)
	w = [1.5 , 2.5 , 3.5 , 4.5]

X = [8 , 10 , 15 , 17 , 48 , 50 , 63 , 65 , 80 , 82]
Y = sqr(X)
z = [9 , 16 , 49 , 64 , 81]
w = lininterp(z , X , Y)
	w = [2.913 , 3.95 , 6.991 , 7.996 , 9]

Compatibility

- Numeric scalars
- *vector* (users)
- *array* (users)

ln

Syntax
y = ln(x)

Definition

ln returns the natural logarithm of x

Examples:

Formula	Result
ln(2.718)	1
ln([1 , 4.482])	[0 , 1.5]
ln([0.607 , 1.649 ; 0.135 , 7.389])	[-0.499 , 0.5 ; -2.002 , 2]

Compatibility

Numeric Scalars, Vectors, Arrays

See Also

exp (users)

log

Syntax

y = log(x)

Definition

Returns the logarithm (base 10) of x.

Examples:

Formula	Result
log(1)	0
log([10 , 1.5])	[1 , 0.176]
log([2.3 , 0.5 ; 3.7 , 0.8])	[0.362 , -0.301 ; 0.568 , -0.097]

Compatibility

- Numeric scalars
- *vector* (users)
- *array* (users)

mag

Syntax

y = mag(x)

Definition

mag takes the absolute value of a real variable or the magnitude of a complex variable. Same as abs function

Examples:

Formula	Result
mag(-1.5)	1.5
mag(complex(1,1))	1.414
mag([-1;-2;3])	[1;2;3]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

abs (users)

matrix

Syntax
z = matrix(x , y)

Definition
matrix builds an x by y matrix of complex numbers. All parts are initialized to zero.

Examples:

Formula	**Result**
z = matrix(2 , 3)	[0 , 0 , 0 ; 0 , 0 , 0]
z = matrix(1 , 5)	[0 , 0 , 0 , 0 , 0]
z = matrix(5 , 2)	[0 , 0 ; 0 , 0 ; 0 , 0 ; 0 , 0 ; 0 , 0]

Compatibility
Positive integers (users)

max

Syntax
y = max(x)
y = max(x, iDim)

Definition
Returns the maximum part of a vector x. In the case of complex-valued arrays, the magnitude of each part is used.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**

x = [10]
y = max(x)
	y = 10

x = [18 , -20 , 23 , 54 , 4 , 71 , -43]
y = max(x)
	y = 71

x = [27, 86 ; complex(600 , -435) , 34]
y = max(x)
	y = [600 - j435, 86]

x = [27, 86 ; complex(1 , 1) , -34]
y = max(x)
	y = [27, 86]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
min (users)

mean

Syntax

y = mean(x)
y = mean(x, iDim)

Definition

Returns the arithmetic mean of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = mean([3 ; 4 ; 8 ; 9])	y = 6
y = mean([complex(1 , 2) ; complex(1 , 1) ; complex(2 , 1)])	y = 1.333 + j1.333
y = mean([1,2,3;4,5,6;7,8,9])	y = [4, 5, 6]

Compatibility

Numeric arrays

See Also

- *median* (users)
- *mode* (users)

median

Syntax

y = median(x)
y = median(x, iDim)

Definition

Returns the median of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = median([3 ; 4 ; 8 ; 9])	y = 6
y = median([complex(1 , 2) ; complex(1 , 1) ; complex(2 , 1)])	y = 2 + j1
y = median([1,2,3;4,5,6;7,8,9])	y = [4, 5, 6]

Compatibility

Numeric arrays

See Also

- *mean* (users)
- *mode* (users)

min

Syntax

y = min(x)
y = min(x, iDim)

Definition
Returns the minimum part of a vector x. In the case of complex-valued arrays, the magnitude of each part is used.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result

x = [10]
y = min(x)
	y = 10

x = [18 , -20 , 23 , 54 , 4 , 71 , -43]
y = min(x)
	y = -43

x = [27, 86 ; complex(600 , -435) , 34]
y = min(x)
	y = [27, 34]

x = [27, 86 ; complex(1 , 1) , -34]
y = min(x)
	y = [1+j1, -34]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
max

mode

Syntax
y = mode(x)
y = mode(x, iDim)

Definition
Returns the mode of a vector x. If there are several values with equal maximum number of occurrences, the smallest value is returned.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = mode ([8 ; 4 ; 8 ; 9])	y = 8
y = mode ([complex(1 , 2) ; complex(1 , 2) ; complex(2 , 1)])	y = 1 + j2
y = mode ([1,2,3;2,2,3;7,8,9])	y = [1, 2, 3]

Compatibility
Numeric arrays

See Also
mean, median

moment

Syntax
y = moment(x, order)
y = moment(x, order, iDim)

Definition
Returns the central moment of order *order* of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = moment([1 ; 2 ; 3 ; 4])	y = 1.25
y = moment([complex(1 , 2) ; complex(2 , 3)], 2)	y = 0 + j0.5

Compatibility
Numeric arrays

ndim

Syntax
y = ndim(x)

Definition
ndim returns the number of dimensions of an array x

Examples:

Formula	Result
y = ndim([3])	y = 1
y = ndim([10 , 3])	y = 2
y = ndim([14 , 2 ; 10 , 3])	y = 2
y = ndim(array(2 , 3 , 4))	y = 3
y = ndim(array(5 , 2 , 3 , 4))	y = 4

Compatibility

- Numeric Scalars
- *vector* (users)
- *array* (users)

NFtoVNI(NF, Rs, TempC)

Syntax
y = NFtoVNI(NF, Rs, TempC)

Definition
NFtoVNI returns the equivalent input noise voltage given the noise factor NF, the input resistance RS in ohms, and the temperature in Celsius.

When using vector the NF and Rs vectors must be the same length.

Examples:

Formula	Result
y = NFtoVNI(2, 50, 17)	y = 0.895 nV
y = NFtoVNI([3, 7], [50, 100], 17)	y = [1.266, 3.101]

Compatibility
Numeric Scalars, Vectors

NOTE: Temperature must be a scalar

See Also
stage_equivalent_input_noise_voltage (sim)

norm

Syntax
y = norm(x)

Definition
norm returns the square of the magnitude of a complex number x. So y = mag(x)2, where x is a complex number. norm does not work for vectors or arrays

Examples:

Formula	Result
y = norm(complex(3 , 5))	y = 34
y = norm(complex(0 , -7))	y = 49
y = norm(complex(4 , 0))	y = 16
y=norm(6)	y = 36

Compatibility
Numeric Scalars, Complex Numbers

See Also

- *abs* (users)
- *mag* (users)

normcdf

Syntax
y = normcdf(x)
y = normcdf(x, mu)
y = normcdf(x, mu, sigma)

Definition
Returns the cumulative distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x. Mu and sigma are optional and default to 0 and 1, respectively. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	Result
normcdf(1)	0.8413
normcdf(1, [0, 1], 2)	[0.6915, 0.5]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
normpdf, norminv

norminv

Syntax
y = norminv(x)
y = norminv(x, mu)
y = norminv(x, mu, sigma)

Definition
Returns the inverse cumulative distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x. Mu and sigma are optional and default to 0 and 1, respectively. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	**Result**
norminv(0.5)	0.8413
norminv(0.1, [0, 0.1], 2)	[-2.5631, -2.4631]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *normpdf* (users)
- *normcdf* (users)

normpdf

Syntax

y = normpdf(x)
y = normpdf(x, mu)
y = normpdf(x, mu, sigma)

Definition

Returns the probability distribution function of the normal distribution with mean mu and standard deviation sigma evaluated at the values of x. Mu and sigma are optional and default to 0 and 1, respectively. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	**Result**
normpdf(1)	0.242
normpdf(1, [0, 1], 2)	[0.176, 0.1995]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

normcdf (users), *norminv* (users)

numcols

Syntax
y = numcols(x)

Definition
numcols returns the number of columns in a matrix x.

Examples:

Formula	Result

b = [1 , 2 ; 3 , 4 ; 1 , 2 ; 3 , 4]
y = numcols(b)
	y = 2

b = [1 , 2 , 3 , 4 ; 1 , 2 , 3 , 4]
y = numcols(b)
	y = 4

Compatibility
Matrices

See Also
numrows (users)

numrows

Syntax
y = numrows(x)

Definition
numrows returns the number of rows in a matrix x.

Examples:

Formula	Result

b = [1 , 2 ; 3 , 4 ; 1 , 2 ; 3 , 4]
y = numrows(b)
	y = 4

b = [1 , 2 , 3 , 4 ; 1 , 2 , 3 , 4]
y = numrows(b)
	y = 2

Compatibility
Matrices

See Also
numcols (users)

ones

Syntax
y = ones(m , n)

Definition
ones returns a m by n matrix with every part equal to 1. If parameter n is omitted the function returns a vector of 1's of length m.

Examples:

Formula	Result
y = ones(3 , 2)	y = [1 , 1 ; 1 , 1 ; 1 , 1]
y = ones(2 , 2)	y = [1 , 1 ; 1 , 1]
y = ones(5)	y = [1 ; 1 ; 1 ; 1 ; 1]

Compatibility
Numeric Scalars

See Also
zeros (users)

PortVolts

Calculates complex voltage amplitudes at ports of a linearized circuit.

Syntax

PortVolts(S, ZPORT, iSigPort, Vin)

where

S[n,n] - frequency swept matrix of linear S-parameters of the circuit,
ZPORT[n] - frequency swept vector of circuit port impedances,
iSigPort = 1..n - index of signal source port,
Vin - complex amplitude of voltage, applied to the signal port,
n - number of circuit ports.

Definition

PortVolts - calculates complex voltage amplitudes at ports of a linearized circuit, when 1 of its ports is excited by the same complex voltage for each swept frequency.

Examples:

See *Examples**Tutorial**LinearPortVolts.wsx:*

It calculates the port voltages using results of Linear analysis "Filter1_Analysis_Data", when port 1 is excited by complex
voltage amplitude ***V1*exp(j*Phase1)*** :

using("Filter1_Analysis_Data")
Vports=PortVolts(S, ZPORT, 1, V1*complex(cos(Phase1),sin(Phase1)))
setindep("Vports","F")

posterror

Syntax
y = posterror(x)

Definition
Converts the argument to a string (if necessary) and posts the string to the error log as an error. This function causes the equations to error out and not calculate, as well as posting the error. The error stays on the log until the equations are recalculated or the error log is cleared.

Examples:

Formula	Result
posterror("Custom Error")	Error window displays an error of "Custom Error"

Compatibility
string

See Also
postwarning (users)

postwarning

Syntax
y = postwarning(x)

Definition
Converts the argument to a string (if necessary) and posts the string to the error log as a warning. This does not cause the equations to error out. The equations will continue to calculate if the postwarning function is executed. The warning stays on the log until the equations are recalculated or the error log is cleared.

Examples:

Formula	Result
postwarning("Custom Warning")	Error window displays a warning of "Custom Warning"

Compatibility
string

See Also
posterror (users)

pow

Syntax
z = pow(x , y)

Definition
pow returns x raised to the power of y. pow uses the equation exp (y * ln(x)) to find z. x must be greater than zero.

Examples:

Formula	Result
z = pow(2 , 4)	z = 16
z = pow(10 , 5)	z = 100000
z = pow(4 , -3)	z = 0.016
z = pow(complex(1 , 1) , 3)	z = -2 + j2

Compatibility
Numeric Scalars

See Also

- *exp* (users)
- *ln* (users)

prctile

Syntax
y = prctile(x, p)
y = prctile(x, p, iDim)

Definition
Returns the p'th percentiles of a vector x (p can be a scalar or a vector of percent values). Percentiles must be between 0 and 100. For an N-part vector, this function computes percentiles by assigning percentile values to the sorted input data as 100*(0.5/N), 100*(1.5/N), ..., 100*((N-0.5)/N). Linear interpolation is then used to compute percentiles between these values. The minimum or maximum values in the data are returned for percentile values outside that range.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
prctile([1, 2, 3, 4, 5], 50)	3
prctile([10, 20, 50, 100, 200], 60)	75

Compatibility
Vectors, Arrays

See Also

- *quantile* (users)
- *median* (users)

prod

Syntax
y = prod(x)
y = prod(x, iDim)

Definition
Returns the product of parts of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = prod([2 , 3 , 5])	y = 30
y = prod([1 , 3 ; 7 , 5])	y = [7, 15]
y = prod([-3.3 , 0.7 , 5 , 3])	y = -34.65

a = complex(1 , 3)
b = complex(4 , 1)
c = complex(1 , 0)
y = prod([a , b , c])
	y = 1 + 13j

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

sum (users)

quantile

Syntax

y = quantile(x, q)
y = quantile(x, q, iDim)

Definition

Returns the q'th quantiles of a vector x (q can be a scalar or a vector of quantile values). Quantiles must be between 0 and 1. For an N-part vector, this function computes quantiles by assigning quantile values to the sorted input data as (0.5/N), (1.5/N), ..., ((N-0.5)/N). Linear interpolation is then used to compute quantiles between these values. The minimum or maximum values in the data are returned for quantile values outside that range.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
quantile([1, 2, 3, 4, 5], 0.5)	3
quantile([10, 20, 50, 100, 200], 0.6)	75

Compatibility

Vectors, Arrays

See Also

- *prctile* (users)
- *median* (users)

rand

Syntax

y = rand(m , n)

Definition

Rand generates uniformly distributed random numbers on the interval [0 , 1). Both arguments are optional. If both arguments are omitted, the function returns a random scalar. If the second argument is omitted, a vector of m random numbers is generated. When both arguments are present a m by n matrix of random numbers is generated.

> Note: The returned number, vector, or matrix is in complex form where the imaginary part is equal to 0. The magnitude of the complex number is displayed. May need to do the operation real(y) to get the desired display result.

Examples:

Formula	**Result**
y = rand()	random number on the interval [0 , 1)
y = rand(4)	vector of length 4 of random numbers on the interval [0 , 1)
y = rand(3 , 5)	3 by 5 matrix of random numbers on the interval [0 , 1)
y = real(rand(4 , 2))	the real part of a 4 by 2 complex matrix of random numbers on the interval [0 , 1)

Compatibility

Numeric Scalars

See Also

- *real* (users)
- *randn* (users)

randn

Syntax

y = randn(m , n)

Definition

randn generates Gaussian distributed random numbers with a mean of 0 and a variance of 1. Both arguments are optional. If both arguments are omitted, the function returns a random scalar. If the second argument is omitted, a vector of m random numbers is generated. When both arguments are present a m by n matrix of random numbers is generated.

> **Note:** The returned number, vector, or matrix is in complex form where the imaginary part is equal to 0. The magnitude of the complex number is displayed. May need to do the operation real(y) to get the desired display result.

Examples:

Formula	**Result**
y = randn()	random number on the interval with a mean of 0 and a variance of 1.
y = randn(4)	vector of length 4 of random numbers with a mean of 0 and a variance of 1.
y = randn(3 , 5)	3 by 5 matrix of random numbers with a mean of 0 and a variance of 1.
y = real(randn(4 , 2))	the real part of a 4 by 2 complex matrix of random numbers with a mean of 0 and a variance of 1.

Compatibility

Numeric Scalars

See Also

- *rand* (users)
- *real* (users)

re

Syntax

y = re(x)

Definition

re returns the real part of a complex number x. Same as real function

Examples:

Formula	Result
y = re(20)	y = 20
y = re(complex(3,2))	y = 3
y = re(complex(-7 , 1) - complex(-2 , 3)) Same as: re(complex(-5 ,-2))	y = -5

b = [complex(2 , 2) , complex(1 , 1)]
y = re(b)
	y = [2 , 1]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *real* (users)
- *imag* (users)

real

Syntax
y = real(x)

Definition
real returns the real part of a complex number x. Same as re function

Examples:

Formula	Result
y = real(20)	y = 20
y = real(complex(3,2))	y = 3
y = real(complex(-7 , 1) - complex(-2 , 3)) Same as: real(complex(-5 ,-2))	y = -5

b = [complex(2 , 2) , complex(1 , 1)]
y = real(b)
	y = [2 , 1]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *re* (users)
- *imag* (users)

reshape

Syntax
y = reshape(x, newshape)

Definition
reshape sets the dimensions of the variable x to the dimensions described by newshape. If the new dimensions contain more parts than the variable x, then the extra parts are filled with zeros.

Swept-dimensions are NOT counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, reshape(S, [4;1]) would return a variable containing S, but having dimensions 100x4x1)

Examples:

Formula	Result

x = [1 , 2 , 3 ; 4 , 5 , 6 ; 7 , 8 , 9]
y = reshape(x , 5)
	y = [1 ; 2 ; 3 ; 4 ; 5]

x = [1 , 2 , 3 ; 4 , 5 , 6]
y = reshape(x , [1 ; 5])
	y = [1 , 2 , 3 , 4 , 5]

x = [1 , 2 ; 3 , 4 ; 5 , 6 ; 7 , 8]
y = reshape(x , [2 ; 3])
	y = [1 , 2 , 3 ; 4 , 5 , 6]

x = [1 , 2 ; 3 , 4]
y = reshape(x , [3 ; 4])
	y = [1 , 2 , 3 , 4 ; 0 , 0 , 0 , 0 ; 0 , 0 , 0 , 0]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *resize* (users)
- *shape* (users)
- *array* (users)

resize

Syntax
y = resize(x, newshape)

Definition
resize sets the dimensions of the variable x to the dimensions described by newshape. If the new dimensions contain more parts than the variable x, then the extra parts are filled with zeros.

Swept-dimensions are counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, resize(S, [100;4;1]) would return a variable containing S, but having dimensions 100x4x1)

Examples:

Formula	Result

x = [1 , 2 , 3 ; 4 , 5 , 6 ; 7 , 8 , 9]
y = resize(x , 5)
	y = [1 ; 2 ; 3 ; 4 ; 5]

x = [1 , 2 , 3 ; 4 , 5 , 6]
y = resize(x , [1 ; 5])
	y = [1 , 2 , 3 , 4 , 5]

x = [1 , 2 ; 3 , 4 ; 5 , 6 ; 7 , 8]
y = resize(x , [2 ; 3])
	y = [1 , 2 , 3 ; 4 , 5 , 6]

x = [1 , 2 ; 3 , 4]
y = resize(x , [3 ; 4])
	y = [1 , 2 , 3 , 4 ; 0 , 0 , 0 , 0 ; 0 , 0 , 0 , 0]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *reshape* (users)
- *array* (users)

reverse

Syntax
y = reverse(x)

Definition

reverse returns the reverse of a vector x. If x is a matrix then reverse returns a matrix with its columns reversed.

Examples:

Formula	Result

b = [1 ; 2 ; 3 ; 4 ; 5]
y = reverse(b)
	y = [5 ; 4 ; 3 ; 2 ; 1]

b = [1 , 2 , 3 ; 4 , 5 , 6]
y = reverse(b)
	y = [4 , 5 , 6 ; 1 , 2 , 3]

Compatibility

- *vector* (users)
- *matrix* (users)

rltogamma

Syntax

y = rltogamma(rl, ang)

Definition

rltogamma returns the voltage reflection coefficient (gamma) based on the return loss rl in dB and the angle ang in radians. y is returned as a complex number. rl must be a number and ang can be a number or a vector.

Examples:

Formula	Result (complex)	Or (magnitude)
y = rltogamma(60, PI)	y = -0.001 + j1.225 * 10^-19	y = 0.001
y = rltogamma(27, (PI / 3))	y = 0.0223 + j0.0387	y = 0.045
y = rltogamma(45, (4 * PI / 3))	y = -.00281 - j0.487	y = 0.00562

rl = 22
and = [1 , 1.5 , 2]
y = rltogamma(rl, ang)
	y[1] = 0.0241 + j0.0376
	y[2] = 0.00316 + j0.0446
	y[3] = -0.0186 + j0.0406
		y = [0.079, 0.079, 0.079]

Compatibility

Numeric Scalars, Vectors

See Also

rltoz (users)

rltoz

Syntax

y = rltoz(zo, rl, ang)

Definition

rltoz returns the complex impedance based on the normalizing impedance zo, the return loss rl in dB, and the angle ang in radians. zo and ang cannot both be vectors. rl must be a number

Examples:

Formula	Result (complex)	Or (magnitude)
y = rltoz(10 , 22 , 1)	y = 10.8 + j1.452	y = 10.893

y = rltoz([3 , 7] , 26 , 2)	y[1] = 2.866 + j0.262
	y[2] = 6.687 + j0.611
		y = [2.878 , 6.715]

y = rltoz(5 , 27 , [1 , 1.5 , 2])	y[1] = 5.232 + j0.394
	y[2] = 5.012 + j0.448
	y[3] = 4.802 + j0.391
		y = [5.247 , 5.032 , 4.818]

Compatibility
Numeric Scalars, Vectors

See Also
rltogamma (users)

rotate

Syntax
y = rotate(v , r)

Definition
Rotate returns a rotated vector or matrix v by the integer value r. If v is a vector then the function rotate moves all the parts r positions over. If v is a matrix then the function rotate moves all the rows r positions down.

Examples:

Formula	Result (complex)

b = [1 ; 2 ; 3 ; 4]
y = rotate(b , 2)
	y = [3 ; 4 ; 1 ; 2]

b = [1 , 2 ; 3 , 4 ; 5 , 6 ; 7 , 8]
y = rotate(b , 1)
	y = [7 , 8 ; 1 , 2 ; 3 , 4 ; 5 , 6]

b = [1 , 2 , 3 ; 4 , 5 , 6 ; 7 , 8 , 9]
y = rotate(b , 5)
	y = [4 , 5 , 6 ; 7 , 8 , 9 ; 1 , 2 , 3]

Compatibility

- Numeric Scalars
- *vector* (users)
- *matrix* (users)

RsCondToThick

Syntax
Thick = RsCondToThick(Rs, Cond, Tunits)

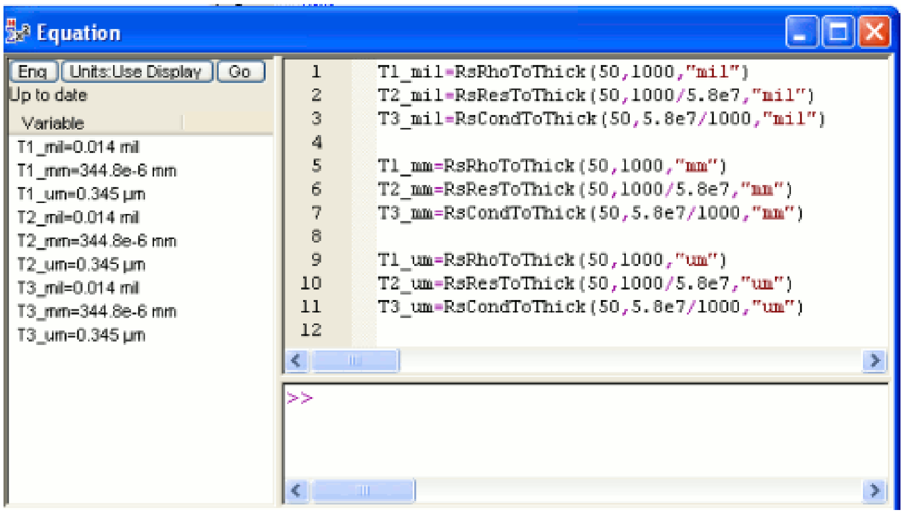
Definition
RsCondToThick Returns metal thickness.
Tunits - output units (string);
Rs - sheet resistance [Ohm/Sq];
Cond - conductivity [Ohm*M]

Examples:

```
 T_mil=RsCondToThick(50,5.8e7,"mil")
 T_mm=RsCondToThick(50,5.8e7,"mm")
```

Note

Use this function for calculating thickness of sheet resistance. When the calculated thickness is used in the Layout Layer properties, not supporting variable parameters, it has to be calculated with Layout units. The calculated value with predefined units may be read from the equation block:



T1_mil=0.014 mil
T1_mm=344.8e-6 mm
T1_um=0.345 µm
T2_mil=0.014 mil
T2_mm=344.8e-6 mm
T2_um=0.345 µm
T3_mil=0.014 mil
T3_mm=344.8e-6 mm
T3_um=0.345 µm

```
T1_mil=RsRhoToThick(50,1000,"mil")
T2_mil=RsResToThick(50,1000/5.8e7,"mil")
T3_mil=RsCondToThick(50,5.8e7/1000,"mil")

T1_mm=RsRhoToThick(50,1000,"mm")
T2_mm=RsResToThick(50,1000/5.8e7,"mm")
T3_mm=RsCondToThick(50,5.8e7/1000,"mm")

T1_um=RsRhoToThick(50,1000,"um")
T2_um=RsResToThick(50,1000/5.8e7,"um")
T3_um=RsCondToThick(50,5.8e7/1000,"um")
```

See Also

RsResToThick (users) *RsRhoToThick* (users)

RsResToThick

Syntax

Thick = RsResToThick(Rs, Res, Tunits)

Definition

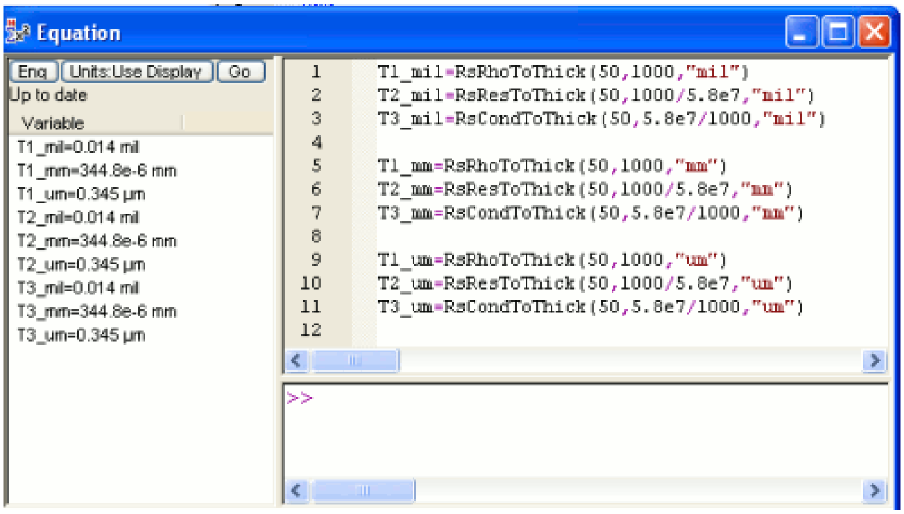
RsCondToThick Returns metal thickness.
Tunits - output units (string);
Rs - sheet resistance [Ohm/Sq];
Res - resistivity [Ohm*M]

Examples:

```
T_mil = RsResToThick( 50, 1.724e-8, "mil")
T_mm = RsResToThick( 50, 1.724e-8, "mm")
```

Note

Use this function for calculating thickness of sheet resistance. When the calculated thickness is used in the Layout Layer properties, not supporting variable parameters, it has to be calculated with Layout units. The calculated value with predefined units may be read from the equation block:



Up to date

Variable

T1_mil=0.014 mil
T1_mm=344.8e-6 mm
T1_um=0.345 µm
T2_mil=0.014 mil
T2_mm=344.8e-6 mm
T2_um=0.345 µm
T3_mil=0.014 mil
T3_mm=344.8e-6 mm
T3_um=0.345 µm

```
T1_mil=RsRhoToThick(50,1000,"mil")
T2_mil=RsResToThick(50,1000/5.8e7,"mil")
T3_mil=RsCondToThick(50,5.8e7/1000,"mil")

T1_mm=RsRhoToThick(50,1000,"mm")
T2_mm=RsResToThick(50,1000/5.8e7,"mm")
T3_mm=RsCondToThick(50,5.8e7/1000,"mm")

T1_um=RsRhoToThick(50,1000,"um")
T2_um=RsResToThick(50,1000/5.8e7,"um")
T3_um=RsCondToThick(50,5.8e7/1000,"um")
```

See Also

RsCondToThick (users) *RsRhoToThick* (users)

RsRhoToThick

Syntax

Thick = RsRhoToThick(Rs, Rho, Tunits)

Definition

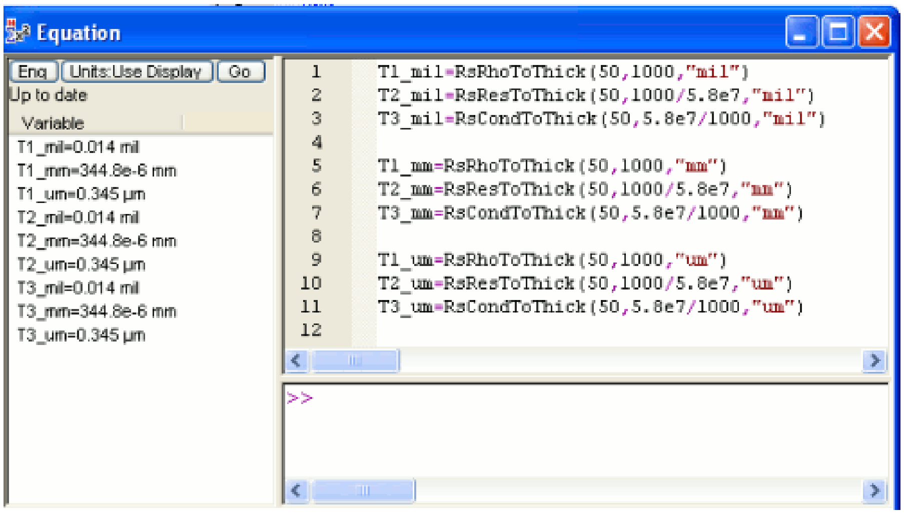
RsRhoToThick Returns metal thickness.
Tunits - output units (string);
Rs - sheet resistance [Ohm/Sq];
Rho - relative to copper resistivity: Rho=Res/ResCop, where copper resistivity ResCop=1.724e-8 [Ohm*M].

Examples:

```
T_mil = RsRhoToThick( 50, 100, "mil")
T_mm = RsRhoToThick( 50, 100, "mm")
```

Note

Use this function for calculating thickness of sheet resistance. When the calculated thickness is used in the Layout Layer properties, not supporting variable parameters, it has to be calculated with Layout units. The calculated value with predefined units may be read from the equation block:



```
T1_mil=0.014 mil
T1_mm=344.8e-6 mm
T1_um=0.345 µm
T2_mil=0.014 mil
T2_mm=344.8e-6 mm
T2_um=0.345 µm
T3_mil=0.014 mil
T3_mm=344.8e-6 mm
T3_um=0.345 µm
```

```
T1_mil=RsRhoToThick(50,1000,"mil")
T2_mil=RsResToThick(50,1000/5.8e7,"mil")
T3_mil=RsCondToThick(50,5.8e7/1000,"mil")

T1_mm=RsRhoToThick(50,1000,"mm")
T2_mm=RsResToThick(50,1000/5.8e7,"mm")
T3_mm=RsCondToThick(50,5.8e7/1000,"mm")

T1_um=RsRhoToThick(50,1000,"um")
T2_um=RsResToThick(50,1000/5.8e7,"um")
T3_um=RsCondToThick(50,5.8e7/1000,"um")

```

See Also

RsCondToThick (users) *RsResToThick* (users)

runanalysis

Syntax

runanalysis("AnalysisName")
runanalysis("AnalysisName", ContinueOnError)

Definition

The runanalysis function is used to force an analysis to run from an equation block. It can be used to control simulations in a sequential manner. The function does not return until the analysis finishes, whether successful or in error.

The second argument, ContinueOnError, is optional and defaults to 0 (false). If ContinueOnError is 0 and an error is encountered when running the analysis, the equation block throws an error and terminates. If ContinueOnError is 1 (true), the equation script continues to run.

Examples:

```
SourceAmpls = [1, 2, 5, 10];  ' We'll step our source's amplitude with these values
SourceAmpls_size = size(SourceAmpls); ' vector representing dimensions of SourceAmpls
for i = 1 to SourceAmpls_size[1]
 CurAmplitude = SourceAmpls[i];  ' This variable is used by our source's Amplitude parameter
 runanalysis("Analysis1");
 ' Post process data from the current analysis run
 ' Post-processing equations would go here
next
```

setindep

Syntax

setindep("dependentvar", "independentvar1", "independentvar2", ...)

Definition

setindep manually sets the independent variable(s) for a swept variable. Both are passed by name. A long name can be used for the independentvar. If independentvar is empty (blank) the dependentvar becomes unswept. All independents should have the same length, equal to the number of rows in the dependent.

Examples:

Formula	**Result**

ind = [0.025;1;2;5]
setindep("x" ,"ind")
	set x to have a 4 part independent vector. x should be of size 4xm or 4xmxn

abest = myS[2,1]
setindep("abest", "myData.F")
	set abest to use MyData.F as an independent vector. F must have the same number of parts as abest has rows.

Compatibility

Vectors and Arrays. The independent var must be numeric.

See Also

getunits (users)

setplottype

Syntax

setplottype("varname", "PlotType")

Definition

setplottype sets the plot-type property of a variable, which indicates both the organization of the data and the drawing style for the plot.

Valid plot types include:
"" (Empty string indicates a standard plot), "Discrete", "Spectrum", "Level", "Spur", "SpurFree", "OutOfRange", "ValidIF", "Contour", "PointPlot", and "Histogram".

Examples:

Formula

x = [1 , 2 , 3 , 4]
setplottype("x" , "Spectrum")

Var1 = [5 , 10 , 15 , 20 ,25]
setplottype("Var1" , "PointPlot")

b = [1.3 , 4.5 , 7.2 , 12.7 , 16.9]
setplottype("b" , "OutOfRange")

Compatibility

Strings

setunits

Syntax

setunits("varname", unit)

Definition

setunits sets a variable named varname to have units specified by the parameter unit. unit may be an integer or a string.

> ℹ setunits is used only to set the units of variables in equations and datasets. It will not change units of a part's parameters.

Examples:

Formula	Result

y = [0.025]
setunits("x" , 6006)
	sets units of y to um
	y = 25000

y = 5
setunit("y" , "mm")
	sets units of y to mm
	y =5000

y = 0.0001
setunits("y" , "uF")
	sets units of y to uF
	y = 100

Compatibility

Numeric Scalars, Strings

See Also

getunits

shape

Syntax

y = shape(x)

Definition

shape returns a vector containing the number of parts in each dimension of x. part one of y corresponds to the number of parts in the first dimension, part two to the second dimension, and so on.

Swept-dimensions are NOT counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, shape(S) returns the vector [2;2]). The function size does count them.

Examples:

Formula	Result

b = [1 , 2 , 3 , 4]
y = shape(b)
	y = [1 ; 4]

b = [1 , 2 , 3 ; 4 , 5 , 6]
y = shape(b)
	y = [2 ; 3]

b = array(5 , 4 , 6)
y = shape(b)
	y = [5 ; 4 ; 6]

b = array(11 , 7 , 3 , 2)
y = shape(b)
	y = [11 ; 7 ; 3 ; 2]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *reshape* (users)
- *size* (users)
- *array* (users)

sin

Syntax
y = sin(x)

Definition
sin returns the sine of the number, in radians (MKS) between -1 <= r < 1

Examples:

Formula	Result
sin(0)	0
sin(PI / 2)	1
sin(-PI / 2)	-1
sin(PI / 4)	0.707
sin(2.094) or sin(2 * PI / 3)	0.866

Compatibility
Numeric scalars, Vectors, Arrays

See Also
asin (users)

sinc

Syntax
y = sinc(x)

Definition
sin returns the sin(PI * x) / (PI * x) or 1 if x is equal to 0.

Examples:

Formula	Result
sinc(0)	1
sinc(PI / 2)	0.198
sinc(PI / 4)	0.253
sinc(2.094) or sinc(2 * PI / 3)	0.044

Compatibility

Numeric scalars, Vectors, Arrays

See Also

sin (users)

sinh

Syntax

y = sinh(x)

Definition

sinh returns the hyperbolic sine of the number, or (ex - e-x) / 2.

Examples:

Formula	Result
sinh(1)	1.175
sinh(5)	74.203
sinh(PI / 3)	1.249
sinh(PI / 6)	0.548
sinh(0)	0

Compatibility

Numeric scalars, Vectors, Arrays

See Also

asinh (users)

size

Syntax

y = size(x)

Definition

size returns a vector containing the number of parts in each dimension of x. part one of y corresponds to the number of parts in the first dimension, part two to the second dimension, and so on.

Swept-dimensions are counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, size(S) returns the vector [100;2;2]). The function shape does not count them.

Examples:

Formula	Result

b = [1 , 2 , 3 , 4]
y = size(b)
	y = [1 ; 4]

b = [1 , 2 , 3 ; 4 , 5 , 6]
y = size(b)
	y = [2 ; 3]

b = array(5 , 4 , 6)
y = size(b)
	y = [5 ; 4 ; 6]

b = array(11 , 7 , 3 , 2)
y = size(b)
	y = [11 ; 7 ; 3 ; 2]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also

- *shape* (users)
- *array* (users)

skewness

Syntax
y = skewness(x)
y = skewness(x, Flag)
y = skewness(x, Flag, iDim)

Definition
Returns the sample skewness of a vector x. Skewness is the third central moment of X divided by the cube of the standard deviation.

If Flag is 0 (default), skewness normalizes by N-1 where N is the sample size. If Flag is 1, skewness normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = skewness([3 ; 4 ; 8 ; 9])	y = 0
y = skewness([1, 2, -5], 1)	y = -0.652

Compatibility
Numeric arrays

See Also

- *std* (users)
- *var* (users)
- *kurtosis* (users)

sort

Syntax
y = sort(x)

Definition
sort returns a vector that has been rearranged in ascending order. If values of x are

complex then the real part of those numbers is used.

Examples:

Formula	**Result**
y = sort([4 , 6 , 2 , 7 , 1 , 3 , 5])	y = [1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7]
y = sort([4 , 6 , 2 ; 1 , 3 , 5])	y = [1 ; 2 ; 3 ; 4 ; 5 ; 6]

y = sort([complex(5 , 1) , complex(7 , 6) , complex(4 , 3) , complex(1 , 6)])	y[1] = 1 + j6
	y[2] = 4 + j3
	y[3] = 5 + j1
	y[4] = 7 +j6
	Note: the magnitudes may appear unsorted.

b = array(2 , 2 , 2) b[2 , 2 , 1] = 1 b[1 , 1 , 1] = 2 b[1 , 2 , 1] = 3 b[2 , 2 , 2] = 4 y = sort(b)	y = [0 ; 0 ; 0 ; 0 ; 1 ; 2 ; 3 ; 4]

Compatibility
Numeric Scalars, Vectors, Arrays

sqr

Syntax
y = sqr(x)

Definition
sqr returns the square root of x. If x is a complex number then the square root of the magnitude is returned as a complex number.

Same as sqrt function

Examples:

Formula	**Result**
y = sqr(16)	y = 4
y = sqr([25 , 9 , 121])	y = [5 , 3 , 11]
y = sqr(complex(4 , 4))	y = 2.197+ j0.9101 or 2.378
y = sqr([3 , 2 ; 5 , 7])	y = [1.732 , 1.14 ; 2.236 , 2.646]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
sqrt (users)

sqrt

Syntax
y = sqrt(x)

Definition
sqrt returns the square root of x. If x is a complex number then the square root of the magnitude is returned as a complex number.

Same as sqr function

Examples:

Formula	**Result**
y = sqrt(16)	y = 4
y = sqrt([25 , 9 , 121])	y = [5 , 3 , 11]
y = sqrt(complex(4 , 4))	y = 2.197+ j0.9101 or 2.378
y = sqrt([3 , 2 ; 5 , 7])	y = [1.732 , 1.14 ; 2.236 , 2.646]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
sqr (users)

std

Syntax
y = std(x)
y = std(x, Flag)
y = std(x, Flag, iDim)

Definition
Returns the standard deviation of a vector x.

If Flag is 0 (default), std normalizes by N-1 where N is the sample size. If Flag is 1, std normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = std([3 ; 4 ; 8 ; 9])	y = 2.9439
y = std([1, 2, 3], 1)	y = 0.8165

Compatibility
Numeric arrays

See Also

- *var* (users)
- *skewness* (users)
- *kurtosis* (users)

stos

Syntax
y=stos(S1, ZPORT1, ZPORT2)

Definition
Converts S1 parameters for port impedance's ZPORT1 to S2 parameters for port impedance's ZPORT2

Examples:

Formula	Result
y=stos(S, 50, 75)	S-Data from 50 Ohm to 75 Ohm reference

Compatibility
Can be used in any dataset and equation block that references to s-parameter data.

stoy

Syntax
Y = stoy(S, ZPORT)

Definition
Converts S-parameters to Y-parameters given the port impedance.

Examples:

Formula	Result
y=stoy(S, 50)	Y-Data converted from S-parameter data using 50 ohms

Compatibility
Can be used in any dataset and equation block that references s-parameter data.

stoz

Syntax
Z = stoz(S, ZPORT)

Definition
Converts S-parameters to Z-parameters given the port impedance.

Examples:

Formula	Result
z=stoz(S, 50)	Z-Data converted from S-parameter data using 50 ohms

Compatibility
Can be used in any dataset and equation block that references s-parameter data.

substrateer

Syntax
y = substrateer(x)

Definition
Returns the dielectric constant (Er) of substrate x.

Examples:

Formula	Result
y=substrateer("Substrate1")	y=4.6

Compatibility
string

See Also
substratetand (users)

substrateh

Syntax
y = substrateh(x)

Definition
Returns the height (Height) of substrate x. The returned value has no units

Examples:

Formula	Result

y=substrateh("Substrate1")
setunits("y","mil")
	y=59

Compatibility
string

See Also
substrateer (users)

substraterho

Syntax
y = substraterho(x)

Definition
Returns the relative resistivity (Rho) of substrate x.

Examples:

Formula	Result
y=substraterho("Substrate1")	y=1

Compatibility
string

See Also
substratetmet (users)

substraterough

Syntax
y = substraterough(x)

Definition
Returns the surface roughness (Sr) of substrate x. The returned value has no units.

Examples:

Formula	Result

y=substraterough("Substrate1")
setunits("y","mil")
	y=0.094

Compatibility
string

See Also
substrateh (users)

substratetand

Syntax
y = substratetand(x)

Definition
Returns the loss tangent (TanD) of substrate x.

Examples:

Formula	**Result**
y=substratetand("Substrate1")	y=0.011

Compatibility
string

See Also
substraterho (users)

substratemet

Syntax
y = substratetmet(x)

Definition
Returns the metal thickness (Thick) of substrate x. The returned value has no units.

Examples:

Formula	**Result**

y=substratetmet("Substrate1")
setunits("y","mil")
	y=1.42

Compatibility
string

See Also
substraterough (users)

sum

Syntax
y = sum(x)
y = sum(x, iDim)

Definition
Returns the sum of parts of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = sum([10 , 3 , 5])	y = 18
y = sum([2 ; 9 ; 11])	y = 22
y = sum([complex(3 , 3) , complex(5 , 2)])	y = 8 + j5
y = sum([3 , 2 , 19 ; 5 , 7 , 1.5])	y = [8, 9, 20.5]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
prod (users)

tan

Syntax
y = tan(x)

Definition
tan returns the tangent of the number, in radians (MKS) between -infinity < r < infinity

Examples:

Formula	Result
tan(PI)	0
tan(PI/4)	1
tan(-PI/4)	-1
tan(5*PI/11)	6.955
tan(-5PI/11)	-6.955

Compatibility
Numeric scalars, Vectors, Arrays

See Also
atan (users)

tanh

Syntax
y = tanh(x)

Definition
tanh retruns the hyperbolic tangent of x or (e2x - 1) / (e2x +1).

Examples:

Formula	Result
tanh(1)	0.762
tanh(5)	1
tanh(PI / 3)	0.781
tanh(PI / 6)	0.48
tanh(0)	0

Compatibility
Numeric scalars, Vectors, Arrays

See Also

atanh (users)

tcdf

Syntax
y = tcdf(x, v)

Definition
Returns the cumulative distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of x. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	**Result**
tcdf(0.5, 5)	0.6809
tcdf(3, [10, 100])	[0.9933, 0.9983]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

- *tpdf* (users)
- *tinv* (users)

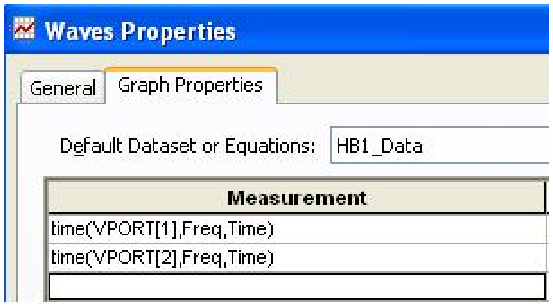
time

Syntax
y = time(spectra, freqs, times)

Definition
time constructs and returns a time-domain waveform from complex spectra at arbitrary frequencies.

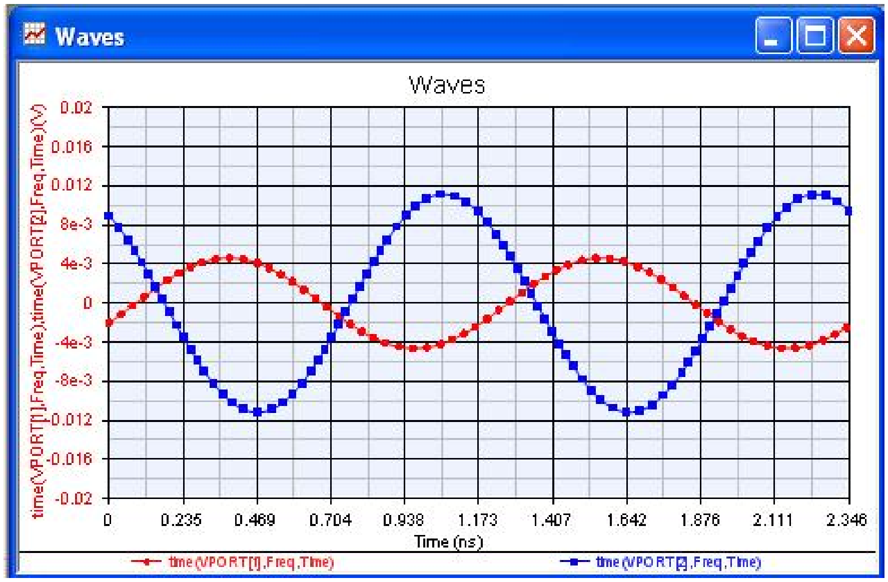
Examples:
The time function is demonstrated in the Amplifier Gain Compression.wsx. This file is located in the Examples\Amplifiers directory of your Genesys directory.

The function is used as a measurement in the Waves graph of the workspace:



Default Dataset or Equations: HB1_Data

Measurement
time(VPORT[1],Freq,Time)
time(VPORT[2],Freq,Time)



Compatibility
array (users)

times

Syntax
y = times(a , b)

Definition
Returns the part-by-part product of the two arguments. If one entry is a scalar and one is an array, the return value is
the scalar multiplied by every part in the array. This is in contrast to the * operator that does matrix multiplication.
Unless one argument is a scalar, then the dimensions of the two parameters must agree.

Examples:

Formula	**Result**

a=2
b=[1,2;3,4]
c=times(a,b)
	c=[2,4;6,8]

a=[1,3,5]
b=[1,3,5]
c=times(a,b)
	c=[1,9,25]

a=[1,2;3,4]
b=[1,2;3,4]
c=times(a,b)
	c=[1,4;9,16]

Compatibility
array (users)

See Also
text

timevector

Syntax
y = timevector(start, stop, step)

Definition

timevector creates and returns a vector of times from start to stop with a specified step size. start must be less than stop and step must be a number greater than 0.

Examples:

Formula	Result
y = timevector(0 , 5 , 0.5)	y = [0 ; 0.5 ; 1 ; 1.5 ; 2 ; 2.5 ; 3 ; 3.5 ; 4 ; 4.5 ; 5]
y = timevector(1 , 100 , 20)	y = [1 ; 21 ; 41 ; 61 ; 81]
y = timevector(1 , 100 , 1001)	y = 1

Compatibility

Numeric scalars

tinv

Syntax

x = tinv(p, v)

Definition

Returns the inverse of the cumulative distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of p. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	Result
tinv(0.5, 5)	0
tinv(0.8, [10, 100])	[0.8791, 0.8452]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

- *tpdf* (users)
- *tcdf* (users)

tpdf

Syntax

y = tpdf(x, v)

Definition

Returns the probability distribution function of the Student's T distribution with degrees of freedom parameter v evaluated at the parts of x. All inputs must match in dimensions or be scalars. Scalars are treated as constant arrays of size compatible with the other arguments.

Examples:

Formula	Result
tpdf(0.5, 5)	0.3279
tpdf(3, [10, 100])	[0.0114, 0.0051]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

- *tcdf* (users)
- *tinv* (users)

transpose

Syntax

y = transpose(x)

Definition

transpose returns the transposition of a matrix or swept matrices x

Examples:

Formula	**Result**
y = transpose([1 , 2 ; 3 , 4 ; 5 , 6])	y = [1 , 3 , 5 ; 2 , 4 , 6]
y = transpose([1 , 2 ; 3 , 4])	y = [1 , 3 ; 2 , 4]
y = transpose([1 , 2 , 3 , 4])	y = [1 ; 2 ; 3 ; 4]
y = transpose([1 ; 2 ; 3 ; 4])	y = [1 , 2 , 3 , 4]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

column (users)

union

Syntax

z = union(x , y)

Definition

union returns a vector representing a set which is the union of two sets (vectors) x and y. The vector z is returned in ascended order. The vectors or matricies can have different dimensions. If x or y have repeated values they will only be found once in the new variable z.

Examples:

Formula	**Result**
z = union([2 ,2 ; 3 , 4] , [5 , 6 ; 3 , 4])	z = [2 ; 3 ; 4 ; 5 ; 6]
z = union([1 , 2] , [3 , 4])	z = [1 ; 2 ; 3 ; 4]
z = union([1 ; 2] , [3 ; 4 ; 5])	z = [1 ; 2 ; 3 ; 4 ; 5]
z = union([1 , 2 , 3] , [4 , 5 ; 6 , 7 ; 8 , 9])	z = [1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8]

x = [1 , 2 , 3 ; 4 , 5 , 6]
y = [7 , 8 , 9 ; 10 , 11 , 12]
z = union(x, y)
	z = [1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 7 ; 8 ; 9 ; 10 ; 11 ; 12]

Compatibility

Numeric scalars, Vectors, Matrices

See Also

column (users)

unwrap

Syntax
z = unwrap(wrappedPhase)

Definition
unwrap returns a vector of unwrapped phase; both input and output of the function are in radians

ⓘ Note that ang() returns a value in radians.

Examples:

Formula
z = unwrap(ang(Linear1.S[2 , 1]))

Compatibility
Numeric scalars, Vectors, Matrices

using

Syntax
using("Dataset")

Definition
using sets the current context in an equation block to the dataset in the parameter. When set, you can use the variables within the dataset as if there were defined in the equation block.

Examples:

Formula	Result

say a dataset called Data1 has a variable called Var1 and it contains the vector
[3 , 6 , 9 , 12]
using("Data1")
z = Var1 / 3
	z = [1 , 2 , 3 , 4]

Compatibility
String

var

Syntax
y = var(x)
y = var(x, W)
y = var(x, W, iDim)

Definition
Returns the variance of a vector x.

If W is 0 (default), var normalizes by N-1 where N is the sample size. If W is 1, var normalizes by N. If W is a vector, it is treated as coefficient weights for computing the variance. In this case, the coefficients of W are scaled so that they sum to unity.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = var([3 ; 4 ; 8 ; 9])	y = 8.6667
y = var([1, 2, 3], 1)	y = 0.6667
y = var([1, 2, 3], [0.7, 0.1, 0.2])	y = 0.65

Compatibility
Numeric arrays

See Also

- *std* (users)
- *skewness* (users)
- *kurtosis* (users)

vector

Syntax
y = vector(x)

Definition
vector returns a vector of complex numbers of length x. parts are initialized to zero..

Examples:

Formula	Result (complex)
y = vector(4)	y = [0 ; 0 ; 0 ; 0]

Compatibility
Numeric scalar

zeros

Syntax
y = zeros(m , n)

Definition
zeros returns a m by n matrix with every part equal to 0. If parameter n is omitted the function returns a vector of 0's of length m.

Examples:

Formula	Result
y = zeros(3 , 2)	y = [0 , 0 ; 0 , 0 ; 0 , 0]
y = zeros(2 , 2)	y = [0 , 0 ; 0 , 0]
y = zeros(5)	y = [0 ; 0 ; 0 ; 0 ; 0]

Compatibility
Numeric Scalars

See Also
ones (users)

Math Language Function Reference

To go directly to entries that start with a specific letter, select one of the following: A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Z.

Function Name	**Description**
abs (users)	absolute value or magnitude
acos (users)	inverse cosine, in radians
acosd (users)	inverse cosine, in degrees
acosh (users)	inverse hyperbolic cosine
acot (users)	inverse cotangent
acotd (users)	inverse cotangent, in degrees
acoth (users)	inverse hyperbolic cotangent
acsc (users)	inverse cosecant
acscd (users)	inverse cosecant, in degrees
acsch (users)	inverse hyperbolic cosecant
all (users)	true if all parts in a vector are nonzero
angle (users)	phase of a complex number, in radians
any (users)	true if any part in a vector is nonzero
asec (users)	inverse secant, in radians
asecd (users)	inverse secant, in degrees
asech (users)	inverse hyperbolic secant
asin (users)	inverse sine, in radians
asind (users)	inverse sine, in degrees
asinh (users)	inverse hyperbolic sine
atan (users)	inverse tangent, in radians
atan2 (users)	4-quadrant inverse tangent, in radians
atand (users)	inverse tangent, in degrees
atanh (users)	inverse hyperbolic tangent
bartlett (users)	Bartlett Window
blackman (users)	Blackman Window
butter (users)	Butterworth filter designer
ceil (users)	smallest integer greater than or equal to argument
cell (users)	create a cell array
cheby1 (users)	Chebyshev type 1 filter designer
cheby2 (users)	Chebyshev type 2 filter designer
class (users)	data-type (class name) of argument
clc (users)	clear the command window
clear (users)	delete a class object
conj (users)	complex conjugate
conv (users)	linear convolution (or polynomial multiplication)
cos (users)	cosine of a radian-valued argument
cosd (users)	cosine of a degree-valued argument
cosh (users)	hyperbolic cosine
cot (users)	cotangent of a radian-valued argument
cotd (users)	cotangent of a degree-valued argument
coth (users)	hyperbolic cotangent
csc (users)	cosecant of a radian-valued argument
cscd (users)	cosecant of a degree-valued argument
csch (users)	hyperbolic cosecant
cumprod (users)	cumulative product of parts of a vector
cumsum (users)	cumulative sum of parts of a vector
dbg_print (users)	output to equation debug window

dbg_showvar (users)	output contents of a variable to equation debug window
deconv (users)	deconvolution (or polynomial division)
diag (users)	create diagonal matrix or extract diagonal of a matrix
diff (users)	difference (or approximate derivative)
eig (users)	eigenvalues and eigenvectors of a matrix
ellip (users)	Elliptic or Cauer filter designer
eps (users)	spacing of floating point numbers
erf (users)	error function
erfc (users)	complementary error function
error (users)	posts to error log or output error to command window
exp (users)	exponential
eye (users)	construct identity matrix
eyediag (users)	build an eye diagram from time data
false (users)	logical false
fclose (users)	close a file or stream
fft (users)	Discrete Fourier Transform (DFT)
fftshift (users)	shift zero-frequency to center of spectrum
fgetl (users)	read a line from a file, discard newline
fgets (users)	read a line from a file, keep newline
filter (users)	one dimensional digital filtering
find (users)	indices of nonzero parts
fir1 (users)	FIR filter design using window method
fix (users)	round toward zero
flipdim (users)	flip matrix along a dimension
fliplr (users)	left/right matrix flip
flipud (users)	up/down matrix flip
floor (users)	largest integer less than or equal to argument
fopen (users)	open file or stream
fread (users)	read binary data from a file or stream
fprintf (users)	write formatted text to a file or stream
fscanf (users)	read formatted text from a file or stream
fwrite (users)	write binary data to a file or stream
gausswin (users)	Gaussian Window
getindep (users)	returns the string property containing the path to the independent value of a variable x. (ie. the reference to the independent variable)
getindepvalue (users)	returns the single independent value of a variable x.
getunits (users)	Returns an integer corresponding to the units of a variable x. This integer may be used by setunits.
getvariable (users)	get the value of a variable from a dataset
hamming (users)	Hamming Window
hann (users)	Hann Window
hankel (users)	construct Hankel matrix
histc (users)	histogram count
ifft (users)	Inverse Discrete Fourier Transform (IDFT)
ifftshift (users)	inverse FFT shift
imag (users)	imaginary part of a complex number
inf (users)	infinity
interp1 (users)	one dimensional interpolation

ipermute (users)	inverse permutation of array dimensions
iscell (users)	true if argument is a cell array
ischar (users)	true if argument is of type character array
isempty (users)	true if argument is empty or array with a dimension of length 0
isequal (users)	true if arrays contain equal values, ignoring NaNs
isequalwithequalnans (users)	true if arrays contain equal values, including NaNs
isfield (users)	true if a field is in a structure or structure array
isfinite (users)	true for finite parts
isfloat (users)	true if argument is a floating point scalar or array
isinf (users)	true for infinite parts
isinteger (users)	true if argument is an integer scalar or array
islogical (users)	true if argument is a logical scalar or array
isnan (users)	true for NaN parts
isnumeric (users)	true if argument is a numeric scalar or array
isreal (users)	true if argument is a real-valued scalar or array
isscalar (users)	true if argument is a scalar
isstr (users)	true if argument is a character array
isstruct (users)	true if argument is a structure array
isvector (users)	true if argument is a vector
kurtosis (users)	sample kurtosis
length (users)	length of a vector
linspace (users)	construct linearly spaced vector
log (users)	natural logarithm
log2 (users)	Base-2 logarithm
log10 (users)	Base-10 logarithm
logspace (users)	construct logarithmically spaced vector
lookup (users)	look up values in a sorted table
lu (users)	LU matrix factorization
max (users)	largest value of a vector
mean (users)	arithmetic mean of a vector
median (users)	median of a vector
min (users)	smallest value of a vector
mkpp (users)	construct a piecewise polynomial
mod (users)	modulus after division
mode (users)	mode (most frequent value) of a vector
moment (users)	nth order central moment of a vector
nan (users)	Not-a-Number
ndims (users)	number of dimensions of the argument
nextpow2 (users)	
num2str (users)	convert number to a character array
numel (users)	total number of parts in an array
ones (users)	construct array with parts set to 1
pchip (users)	construct piecewise cubic Hermite interpolating polynomial
permute (users)	permutation of array dimensions
poly (users)	convert roots to a polynomial
polyval (users)	evaluate a polynomial
polyvalm (users)	evaluate a polynomial with a matrix argument

ppval (users)	evaluate a piecewise polynomial
prctile (users)	p'th percentiles of a vector
prod (users)	product of parts of a vector
quantile (users)	q'th quantiles of a vector
rand (users)	uniformly distributed random numbers between 0 and 1
randn (users)	Normally (Gaussian) distributed random numbers
real (users)	real part of a complex number
rectwin (users)	Rectangular Window
rem (users)	remainder after division
repmat (users)	replicate and tile an array
reshape (users)	change dimensions of an array
roots (users)	roots of a polynomial
rot90 (users)	rotate a matrix 90 degrees
round (users)	round towards nearest integer
runanalysis (users)	Run an analysis in the workspace tree
sec (users)	secant of a radian-valued argument
secd (users)	secant of a degree-valued argument
sech (users)	hyperbolic secant
setindep (users)	set the independent reference for a swept dependent variable to indepvar(s). A minimum of two arguments is required. This function can be used to remove all independent values of a variable by passing in a blank string for the second argument.
setvariable (users)	write a value to a variable in a dataset
setunits (users)	sets a variable named varname to have units specified by unit. unit may be an integer or a string. Example setunits("totaltime","msec") or setunits("frqsweep", "MHz"). The units are used to by graphs to determine the axis labels and values. They can also by used by the Tune window. Use UseMKS if you are settings units of variables manually to avoid confusion.
shiftdim (users)	shift array dimensions
sign (users)	signum
sin (users)	sine of a radian-valued argument
sinc (users)	sinc function (sin(pi*x) / (pi*x))
sind (users)	sine of a degree-valued argument
sinh (users)	hyperbolic sine
size (users)	dimensions of an array
skewness (users)	skewness of a vector
sort (users)	sort a vector in ascending or descending order
spline (users)	cubic spline interpolation
sqrt (users)	square root
std (users)	standard deviation of a vector
str2num (users)	convert a string to a number
strcmp (users)	case-sensitive string comparison
strcmpi (users)	case-insensitive string comparison
strncmp (users)	compare first N characters of a string (case-sensitive)
strncmpi (users)	compare first N characters of a string (case-insensitive)
struct (users)	construct a structure array
sum (users)	sum of the parts of a vector
svd (users)	matrix singular value decomposition
tan (users)	tangent of a radian-valued argument

tand (users)	tangent of a degree-valued argument
tanh (users)	hyperbolic tangent
tcpip (users)	construct tcpip stream object for TCP/IP communications
toeplitz (users)	construct Toeplitz matrix
true (users)	logical true
unmkpp (users)	details of piecewise polynomial
using (users)	sets the current context in an equation block to the dataset called Dataset.
var (users)	variance of a vector
warning (users)	posts a warning to error log or output warning to command window
xcorr (users)	cross correlation
xor (users)	logical exclusive-OR
zeros (users)	construct array with parts set to 0

abs

Syntax
y = abs(x)

Definition
This function takes the absolute value of a real variable or the magnitude of a complex variable. It operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
abs(-1.5)	1.5
abs(complex(1,1))	1.414
abs([-1;-2;3])	[1;2;3]

Compatibility
scalars, vectors, arrays

acos

Syntax
y = acos(x)

Definition
This function returns the inverse cosine of the angular value x, in radians expressed in the MKS range [0, PI]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Examples:

Formula	**Result**	**or**
acos(0)	1.571	PI/2
acos(1)	0	0
acos(-1)	3.141	PI
acos(.707)	0.786	PI/4
acos([-.707 0 1])	[2.356 1.571 0]	[3*PI/4 PI/2 0]

Compatibility
Real valued scalars, vectors, arrays

See Also

acosd (users)
acosh (users)
cos (users)
cosd (users)
cosh (users)

acosd

Syntax

y = acosd(x)

Definition

This function returns the inverse cosine of the angular value x, in radians expressed in the range [0, 180]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Examples:

Formula	Result
acosd(0)	90
acosd(1)	0
acosd(-1)	180
acosd(.707)	45
acosd([-.707 0 1])	[135 90 0]

Compatibility

Real valued scalars, vectors, arrays

See Also

acos (users)
acosh (users)
cos (users)
cosd (users)
cosh (users)

acosh

Syntax

y = acosh(x)

Definition

This function returns the inverse of the hyperbolic cosine of the number x. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

```
cosh(x) = log( x + sqrt(  x^2 - 1))
```

Examples:

Formula	Result
acosh(1)	0
acosh(10)	2.993
acosh(0)	NaN

Compatibility

Real valued scalars, vectors, arrays

See Also

acos (users)
acosd (users)
cos (users)
cosd (users)
cosh (users)

acot

Syntax
y = acot(x)

Definition
This function returns the inverse co-tangent of the angular value x, in radians expressed in the MKS range [0, PI]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Formula	**Result**	**or**
acot(1.732)	0.5236	PI/6
acot(0.577)	1.0472	PI/3

Compatibility
Real valued scalars, vectors, arrays

See Also:

acotd (users)
acoth (users)
cot (users)
cotd (users)
coth (users)

acotd

Syntax
y = acotd(x)

Definition
This function returns the inverse co-tangent of the angular value x, in radians expressed in the range [0, 180]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Formula	**Result**
acotd(1.732)	30
acotd(0.577)	60

Compatibility
Numeric scalars, vectors, arrays

See Also:
acot (users)
acoth (users)
cot (users)
cotd (users)
coth (users)

acoth

Syntax
y = acoth(x)

Definition
This function returns the inverse of the hyperbolic co-tangent of the number x. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Compatibility
Real valued scalars, vectors, arrays

See Also:

acot (users)
acotd (users)
cot (users)
cotd (users)
coth (users)

acsc

Syntax
y = acsc(x)

Definition
This function returns the inverse co-secant of the angular value x, in radians expressed in the MKS range [0, PI]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Compatibility
Real valued scalars, Vectors, Arrays

See Also:
acscd (users)
acsch (users)
csc (users)
cscd (users)
csch (users)

acscd

Syntax
y = acscd(x)

Definition
This function returns the inverse co-secant of the angular value x, in degrees expressed in the range [0, 180]. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Compatibility
Real valued scalars, Vectors, Arrays

See Also:
acsc (users)
acsch (users)

csc (users)
cscd (users)
csch (users)

acsch

Syntax
y = acsch(x)

Definition
This function returns the inverse of the hyperbolic co-secant of the number x. It operates on an part-by-part basis on arrays. It cannot accept a complex valued variable.

Compatibility
Numeric scalars, Vectors, Arrays

See Also:
acsc (users)
acscd (users)
csc (users)
cscd (users)
csch (users)

all

Syntax
all(data)
all(data, dim)

Definition
This function returns true if all values in a vector are non-zero or logical true, otherwise it returns false. If *data* is a matrix, then this function operates on the columns of data.

The *dim* argument is optional and specifies which dimension to operate along. For example, if *dim* is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**
all([1 0 0 1 0])	0
all([1 1 1])	1

For **A** = [1 1 0; 1 0 1; 1 1 1];

Formula	**Result**	**Comments**
all(A)	[1 0 0]	Returns dim=1 column wise results
all(A, 1)	[1 0 0]	Returns column wise results
all(A, 2)	[0; 0; 1]	Returns row wise results

Compatibility
vectors, arrays

See Also
any (users)

angle

Syntax

y = angle(x)

Definition

This function returns the phase of a complex number, in radians.

This function operates on an part-by-part basis on arrays.

Compatibility

Complex valued scalars, vectors, arrays

Real valued variables are treated as vectors with angular value of zero.

any

Syntax

any(data)

any(data, dim)

Definition

This function returns true if any of the values in a vector are non-zero or logical true, otherwise it returns false. If *data* is a matrix, then this function operates on the columns of data.

The *dim* argument is optional and specifies which dimension to operate along. For example, if *dim* is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**
all([1 0 0 1 0])	1
all([0 0 0])	0

For **A** = [0 0 1; 0 1 0; 0 0 0];

Formula	**Result**	**Comments**
all(A)	[0 1 1]	Returns dim=1 column wise results
all(A, 1)	[0 1 1]	Returns column wise results
all(A, 2)	[1; 1; 0]	Returns row wise results

Compatibility

vectors, arrays

See Also

all (users)

asec

Syntax

y = asec(x)

Definition

This function is the inverse secant, in radians in the range [0, PI].

This function operates on an part-by-part basis on arrays.

Compatibility
Real valued scalars, vectors, arrays

See Also
asecd (users)
asech (users)
sec (users)
secd (users)
sech (users)

asecd

Syntax
y = asecd(x)

Definition
This function is the inverse secant, in degrees.
This function operates on an part-by-part basis on arrays.

Compatibility
Real valued scalars, vectors, arrays

See Also
asec (users)
asech (users)
sec (users)
secd (users)
sech (users)

asech

Syntax
y = asech(x)

Definition
This function returns the inverse hyperbolic secant of the argument.
This function operates on an part-by-part basis on arrays.

Compatibility
Real valued scalars, vectors, arrays

See Also
asecd (users)
sec (users)
secd (users)
sech (users)

asin

Syntax
y = asin(x)

Definition
asin returns the inverse sine of the argument, in radians, between -PI / 2 <= r <= PI / 2.
This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result	or
asin (0)	0	0
asin (1)	1.571	PI/2
asin(-1)	-1.571	-PI/2
asin (.707)	0.786	PI/4
asin (-.707)	-0.786	-PI/4

Compatibility

Real valued scalars, vectors, arrays

See Also

asind (users)
asinh (users)
sin (users)
sind (users)
sinh (users)

asind

Syntax

y = asind(x)

Definition

asind returns the inverse sine of the argument, in degrees, in a range of [-180, 180]. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result	in Radians
asin (0)	0	0
asin (1)	180	PI/2
asin(-1)	-180	-PI/2
asin (.707)	45	PI/4
asin (-.707)	-45	-PI/4

Compatibility

Real valued scalars, vectors, arrays

See Also:

asin (users)
asinh (users)
sin (users)
sind (users)
sinh (users)

asinh

Syntax

y = asinh(x)

Definition

This function returns the inverse hyperbolic sine of the argument, equal to log(x + sqrt(x^2 + 1)). This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
asinh(1)	0.881
asinh(10)	2.998
asinh([0 1 10])	[0 0.881 2.998]

Compatibility
Real valued scalars, vectors, arrays

See Also:
asind (users)
asin (users)
sin (users)
sind (users)
sinh (users)

atan

Syntax
y = atan(x)

Definition
This function returns the inverse tangent of the argument, in radians between -PI/2 < r < PI/2. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
atan(0)	0
atan(1)	0.785
atan([-1 .5 -.5])	[-0.785 0.464 -0.464]

Compatibility
Real valued scalars, vectors, arrays

See Also:
tanh (users)
atan2 (users)
atand (users)
atanh (users)
tan (users)
tand (users)

atan2

Syntax
y = atan2(y, x)

Definition
atan2 returns the 4-quadrant inverse tangent of the argument, in radians. The return value is the same size as the input arrays y and x, and is computed on an part-by-part basis. Either argument may be a scalar, in which case that argument is expanded to be the same size as the other argument. For complex inputs, imaginary parts are ignored.

Examples:

Formula	Result	or
atan2(1, 0)	1.571	pi/2
atan2(1, 1)	0.785	pi/4
atan2([1; 0; -1], -1)	[2.356; 3.142; -2.356]	[3*pi/4; pi; -3*pi/4]

Compatibility
Real valued scalars, vectors, arrays

See Also
atan (users)
tan (users)

atand

Syntax
y = atand(x)

Definition
This function returns the inverse tangent of the argument, in degrees.
This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result	in Radians
atan(0)	0	
atan(1)	45	PI/4
atan(-1)	-45	-PI/4

Compatibility
Real valued scalars, vectors, arrays

See Also:
atan (users)
tan (users)
tand (users)
atanh (users)
tand (users)

atanh

Syntax
y = atanh(x)

Definition
This function returns the inverse hyperbolic tangent of the argument, which is equivalent to 0.5 * log((1 + x) / (1 - x)). This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
atanh(1)	undefined
atanh(.5)	0.549
atanh(-.5)	-0.549
atanh(0)	0

Compatibility
Real valued scalars, vectors, arrays

See Also:
atan (users)
tan (users)
atand (users)
tanh (users)
tand (users)

bartlett

Syntax
bartlett(N)

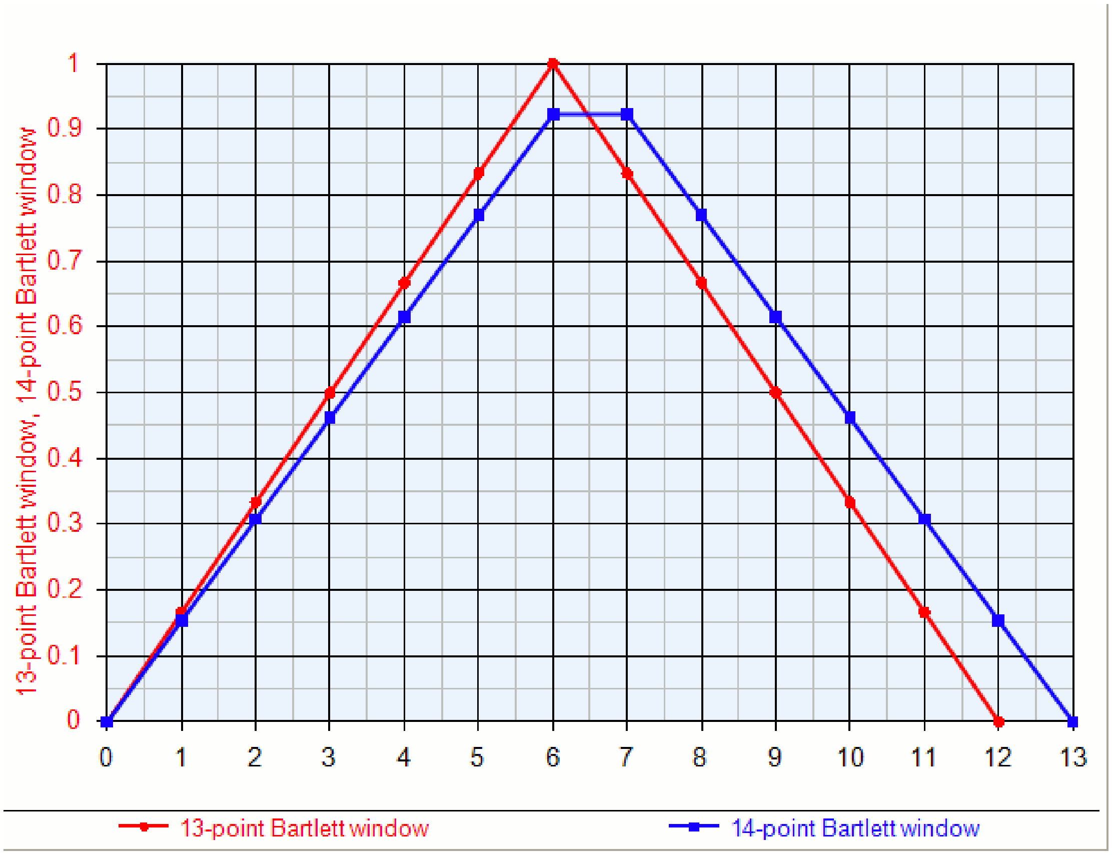
Definition
This function returns a column vector containing a Bartlett window with N points, N being a positive integer greater than 2. The Bartlett window is characteristically triangular in shape with a base value of 0 and an apex value of 1. When N is odd, the apex is explicitly an part of the window function. When N is even, the apex is not explicitly sampled but rather the two sample points which flank the apex are represented in the returned vector.

> ℹ **Note**
> bartlett(2), a redundant usage of this function returns [0 0] whereas bartlett(1) returns [1].

Examples:

Formula	**Result**	**Comment**
bartlett(13)	[0,1,2,3,4,5,6,5,4,3,2,1,0]/6	(13-1)/2=6 is common divisor
bartlett(14)	[0,1,2,3,4,5,6,6,5,4,3,2,1,0]/6.5	(14-1)/2=6.5 is common divisor

The graph shows how bartlett(14) does not sample the peak value of 1 at 6.5 explicitly but bartlett(13) does.



Compatibility
scalar

See Also
blackman (users)
gausswin (users)
hamming (users)
hann (users)
rectwin (users)

blackman

Syntax
blackman(N)

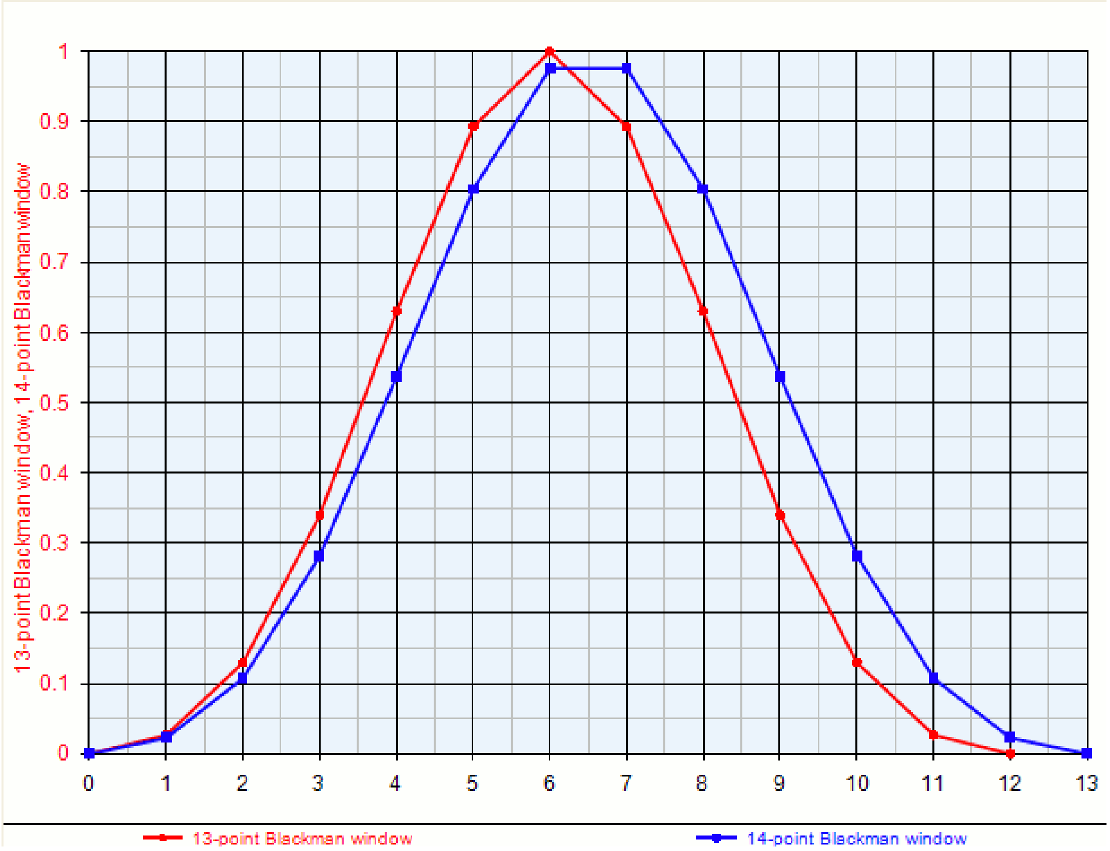
Definition
This function returns a column vector containing a Blackman window with N points, N being a positive integer greater than 2. The Blackman window is composed of raised cosine windows scaled to have a base value of 0 and an apex value of 1 as follows:

```
_blackman_value_at_n_of_N_ = 0.42 - 0.5 * cos( 2*pi*n/N ) + 0.08 * cos( 4*pi*n/N ), 0 <= n <= N
```

When N is odd, the apex is explicitly an part of the window function. When N is even, the apex is not explicitly sampled but rather the two sample points which flank the apex are represented in the returned vector.

> **Note**
> blackman(2), a redundant usage of this function returns [0 0] whereas blackman(1) returns [1].

Examples:



Compatibility
scalar

See Also
bartlett (users)
gausswin (users)
hamming (users)
hann (users)
rectwin (users)

butter

Syntax
[num, denom] = butter(order, normfreq, ftype, domain)
or
[zeros, poles, gain] = butter(order, normfreq, ftype, domain)

Parameters

Name	**Definition**	**Compatibility**	**Usage**	**Default**	**Example**
order	order of Butterworth filter	positive integer >= 3	required		5
normfreq	normalized frequency or range of frequencies defining filter	normalized scalar or 2-part vector	required		0.3
ftype	type of filter	enumerated as 'low','high','pass' or 'stop'	optional	'low'	'pass'
domain	digital (Z-domain) or analog (S-domain) filter	'z' or 's'	optional	'z'	's'

Definition
Depending on the list out output arguments, this function delivers a numerator-

denominator or a pole-zero-gain definition of a maximally-flat Butterworth filter response. Input arguments consist of order, normalized frequency range and the optional enumerated choice of filter type.

Examples:

Note that while zeros and poles are expressed as column vector, numerator and denominator coefficients are expressed as row vectors. Gain is always expressed as a real valued scalar variable.

Formula	**zeros**	**poles**	**gain**	**num**	**denom**
butter(3, 0.5)	[-1+j4.714e-6; -1-j4.714e-6; -1]	[j/√3; -j/√3;0]	1/6	[1/6, 1/2, 1/2, 1/6]	[1, 0, 1/3, 0]
butter(3, 0.5, 'high')	[1+j4.714e-6; 1-j4.714e-6; 0]	[j/√3; -j/√3;0]	1/6	[1/6, -1/2, 1/2, -1/6]	[1, 0, 1/3, 0]
butter(3, [0.25,0.75], 'pass')	[1; 1+j2.597e-6; 1-j2.597e-6; -1; -1+3.772e-6; -1-j3.772e-6]	[-0.537+j0.537; -0.537-j0.537; 0.537+j0.537; 0.537+j0.537; j7.451e-9; -j7.451e-9]	1/6	[1/6, 0, -1/2, 0, 1/2, 0, -1/6]	[1, 0, 0, 0, 1/3, 0, 0]
butter(3, [0.25,0.75], 'stop')	[-3.055e-6+j; -3.055e-6-j; 3.055e6+j; 3.055e-6-j; j; -j]	[-0.537+j0.537; -0.537-j0.537;0.537+j0.537; 0.537-j0.537; 9.125e-9; 9.125e-9]	1/6	[1/6, 0, 1/2, 0, 1/2, 0, 1/6]	[1, 0, 0, 0, 1/3, 0, 0]

See Also

cheby1 (users)
cheby2 (users)
ellip (users)

ceil

Syntax

y = ceil(x)

Definition

ceil returns the smallest integer greater than or equal to the argument. If x is complex, only the real part is used. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
ceil(10)	10
ceil(complex (1.5 , 6))	2
ceil([-0.5, 0.5])	[0 , 1]

Compatibility

Numeric scalars, vectors, arrays

See Also

floor (users)

cell

Syntax

a = cell(y)
a = cell(x, y)
a = cell([x, y])
a = cell(x, y, z,...)
a = cell([x y z ...])

a = cell(size(V))

Definition
a = cell(y) creates a cell array, whose dimension is y-by-y, containing empty matrices. If the parameter y is not of type scalar, then a = cell(y) produces an error message.

a = cell(x,y) and a = cell([x,y]) creates a cell array, whose dimension is x-by-y, containing empty matrices. If the parameters x and/or y are not of type scalar, then an error message is produced when these statements are executed.

a = cell(x,y, z, ...) and a = cell([x,y,z,...]) creates a cell array, whose dimension is x-by-y-by-z-...and so on, containing empty matrices. If any of the parameters x, y, z,..., are not of type scalar, then an error message is produced when these statements are executed.

a = cell(size(V)) creates a cell array, whose dimension matches that of V, containing empty matrices.

Examples:

Formula	**Result**
a = cell(2)	a = {[], []; [],[]}
a = cell([3,2])	a = {[],[];[],[];[],[]}

V = [1;3;5]
a = cell(size(V))
	a = {[];[];[]}

Compatibility
scalar, vector, array

See Also
ones (users)
rand (users)
randn (users)
zeros (users)

cheby1

Syntax
[num, denom] = cheby1(order, normripple, normfreq, ftype, domain)
or
[zeros, poles, gain] = cheby1(order, normripple, normfreq, ftype, domain)

Parameters

Name	**Definition**	**Compatibility**	**Usage**	**Default**	**Example**
order	order of Butterworth filter	positive integer >= 3	required		5
normripple	normalized ripple in passband	positive real	required		0.1
normfreq	normalized frequency or range of frequencies defining filter	normalized scalar or 2-part vector	required		0.3
ftype	type of filter	enumerated as 'low','high','pass' or 'stop'	optional	'low'	'pass'
domain	digital (Z-domain) or analog (S-domain) filter	'z' or 's'	optional	'z'	's'

Definition

Depending on the list out output arguments, this function delivers a numerator-denominator or a pole-zero-gain definition of a Chebyshev filter response of Type 1, which allows ripples in the passband and creates a maximally flat stopband. Input arguments consist of order, normalized in-band ripple, normalized frequency range and the optional enumerated choice of filter type.

Examples:
Note that while zeros and poles are expressed as column vector, numerator and denominator coefficients are expressed as row vectors. Gain is always expressed as a real valued scalar variable.

Formula	**zeros**	**poles**	**gain**	**num**	**denom**
cheby1(3, 0.1, 0.5)	[-1; -1; -1]	[-0.1885+j0.659; 0.0155; -0.1885+j0.659]	0.227	[0.227, 0.682, 0.682, 0.227]	[1, 0.361, 0.464, -0.007]
cheby1(3, 0.1, 0.5, 'high')	[1 1 1]	[0.1885+j0.659; -0.0155; 0.1885-j0.659]	0.227	[0.227, -0.682, 0.682, -0.227]	[1, -0.361, 0.464, 0.007]
cheby1(3, 0.1, [0.25,0.75], 'pass')	[1; 1; 1; -1; -1; -1]	[0.661+j0.499; j0.125; 0.661-j0.499; -0.661=j0.499; -j0.125; -0.661+j0.499]	0.227	[0.227, 0, -0.682, 0, 0.682, 0, -0.227]	[1, 0, -0.361, 0, 0.464, 0 0.007]
cheby1(3, 0.1, [0.25,0.75], 'stop')	[-j; j; -j; j; -j; j]	[0.499-j0.661; 0.125; 0.499+j0.661; -0.499+j0.661; -0.125; -0.499-j0.661]	0.227	[0.227, 0, 0.682, 0, 0.682, 0, 0.227]	[1, 0, 0.361, 0, 0.464, 0, -0.007]

See Also
butter (users)
cheby2 (users)
ellip (users)

cheby2

Syntax
Syntax
[num, denom] = cheby2(order, normripple, normfreq, ftype, domain)
or
[zeros, poles, gain] = cheby2(order, normripple, normfreq, ftype, domain)

Parameters

Name	**Definition**	**Compatibility**	**Usage**	**Default**	**Example**
order	order of Butterworth filter	positive integer >= 3	required		5
normripple	normalized ripple in stopband	positive real	required		0.1
normfreq	normalized frequency or range of frequencies defining filter	normalized scalar or 2-part vector	required		0.3
ftype	type of filter	enumerated as 'low','high','pass' or 'stop'	optional	'low'	'pass'
domain	digital (Z-domain) or analog (S-domain) filter	'z' or 's'	optional	'z'	's'

Definition
Depending on the list out output arguments, this function delivers a numerator-denominator or a pole-zero-gain definition of a Chebyshev filter response of Type 2, which allows ripples in the stopband and creates a maximally flat passband. Input arguments consist of order, normalized out-of-band ripple, normalized frequency range and the optional enumerated choice of filter type.

Examples:
Note that while zeros and poles are expressed as column vector, numerator and

denominator coefficients are expressed as row vectors. Gain is always expressed as a real valued scalar variable.

Formula	**zeros**	**poles**	**gain**	**num**	**denom**
cheby2(3, 0.1, 0.5)	[-0.143+j0.990; -0.143-j0.990; -1]	[-0.138-j0.962; -0.903; -0.137+j0.962]	0.924	[0.924, 1.188, 1.188, 0.924]	[1, 1.178, 1.192, 0.853]
cheby2(3, 0.1, 0.5, 'high')	[0.143-j0.990; 0.143+j0.990; 1]	[0.137+j0.962; 0.903; 0.137-j0.962]	0.924	[0.924, -1.188, 1.188, -0.924]	[1, -1.178, 1.192, -0.853]
cheby2(3, 0.1, [0.25,0.75], 'pass')	[-0.756+j0.655; 0.756+j0.655; 0.756-j0.655; -0.756-0.655; 1; -1]	[0.745+j0.646; 0.951; 0.745-j0.646; -0.745-j0.646; -0.951; -0.745+j0.646]	0.924	[0.924, 0, -1.188, 0, 1.188, 0, -0.924]	[1, 0, -1.178, 0, 1,192, 0, -0.853]
cheby2(3, 0.1, [0.25,0.75], 'stop')	[0.655+j0.756; -0.655+j0.756; -0.655-j0.756; 0.655-j0.756; -j; j]	[0.646-j0.745;-j0.951; 0.646+j0.745; -0.646+j0.745; j0.951; -0.646-j0.745]	0.924	[0.924, 0, 1.188, 0, 1.188, 0, 0.924]	[1, 0, 1.178, 0, 1.192, 0, 0.853]

See Also

cheby1 (users)
butter (users)
ellip (users)

class

Syntax

type = class(object)

Definition

This function returns the type of class of the supplied *object* as a string *type*. The input argument is evaluated as an expression so a combination of existing objects can be applied to this parameter.

Example

- **char**

```
c1 = class( ['This','is','a','char','class','vector'] )
% This is a vector of strings or an array of characters
% c1 = 'char'
```

- **cell**

```
c2 = class( {'This','is','a','char','class','vector'} )
% This is a cell array of characters
% c2 = 'cell'
```

- **double**

```
c3a = class( [1 2 3; 4.5 5 6] )
% This is an array of double precision floating point numbers
% c3a = 'double'
% Note real-valued integers and floating point assigned the 'double'
% whereas,
c3b = class( 4+5i )
% complex-valued numbers are assigned 'complex double'
% c3a = 'complex double'
```

- **logical**

```
c4 = class( [ 3<=4, length('were') < size([2,1])] )
% The 1st expression is evaluating whether or not 3<=4.
% The 2nd expression is evaluating wthere the length of
% the string 'were' is greater than the length of the supplied
% numeric vector [2,1]. Both are logical expressions, making
% the supplied vector of logical class.
% c4 = 'logical'
```

- **struct**

```
c5 = class( struct('Name',{'FirstName','LastName'},'Date Of Birth', [23 04 1999]) )
% The expression defines a structure, thus
% c5 = 'struct'
```

Compatibility
all

See Also
struct (users)

clc

Syntax
clc()

Definition
Clear the command window

clear

Syntax
clear
clear name
clear name1 name2 name3
clear ('name1', 'name2', 'name3')
clear **global** name

Definition
Remove variables from an equation block. This frees up old variables in memory.

clear name just removes the variable *name* from the equation block.

clear name1 name2 name3 or *clear('name1', 'name2', 'name3')* removes variables *name1, name2,* and *name3* from the equation block.

clear global name removes the global variable *name* from the equation block.

conj

Syntax
y = conj(x)

Definition

conj returns the complex conjugate of the argument. The conjugate of a complex number x + jy is x - jy. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
conj(1+2j)	1 - 2j
conj([1 + 2j, 3 - 4j])	[1 - 2j, 3 + 4j]

Compatibility

Numeric Scalars, Arrays, Vectors

conv

Syntax

y = conv(x1,x2)

Definition

This function performs the algebraic convolution between the two vector valued inputs *x1* and *x2*. Given the lengths of the vectors to be *lN* = length(*xN*), N = {1, 2}, the result is of length equal to sum of all lengths minus 1.

Examples:

```
a = [ 2, 3 ]
b = [ 5, 6, 7 ]
c = conv( a, b )
% c = [10 27 32 21] because
% c(1) = a(1)*b(1) = 10
% c(2) = a(1)*b(2) + a(2)*b(1) = 12 + 15 = 27
% c(3) = a(1)*b(3) + a(2)*b(2) = 14 + 18 = 32
% c(4) = a(2)*b(3) = 21
```

Compatibility

Real and complex valued scalars and vectors. Multi-dimensional arrays are not supported.

See Also

fft (users)

ifft (users)

cos

Syntax

y = cos(x)

Definition

cos returns the cosine of a radian-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
cos(0)	1
cos(pi)	-1
cos(pi / 2)	0
cos(pi / 4)	0.707
cos([2*pi/3; pi/2])	[-0.5; 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
cosd (users)
sin (users)
tan (users)

cosd

Syntax
y = cosd(x)

Definition
cosd returns the cosine of a degree-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
cosd(0)	1
cosd(180)	-1
cosd(90)	0
cosd(45)	0.707
cosd([60; 90])	[-0.5; 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
cos (users)

cosh

Syntax
y = cosh(x)

Definition
cosh returns the hyperbolic cosine of the argument, equivalent to (exp(x) + exp(-x)) / 2. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
cosh(1)	1.543
cosh(pi / 3)	1.6
cosh([pi/6; 0])	[1.14; 1]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
acosh (users)

cot

Syntax
y = cot(x)

Definition
cot returns the cotangent of a radian-valued argument, which is equivalent to 1 / tan(x). This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

cotd

Syntax
y = cotd(x)

Definition
cotd returns the cotangent of a degree-valued argument. This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

coth

Syntax
y = coth(x)

Definition
coth returns the hyperbolic cotangent of the argument. This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

csc

Syntax
y = csc(x)

Definition
csc returns the cosecant of a radian-valued argument. This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

cscd

Syntax
y = cscd(x)

Definition

cscd returns the cosecant of a degree-valued argument. This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

csch

Syntax
y = csch(x)

Definition
csch returns the hyperbolic cosecant of the argument. This function operates on an part-by-part basis on arrays.

Compatibility
Numeric scalars, Vectors, Arrays

cumprod

Syntax
b = cumprod(a)
b = cumprod(a,dim)

Definition
Returns the cumulative product of a vector. If a is a vector, then cumprod finds the cumulative product of all the parts and returns it in a vector, b. If a is a matrix, then cumprod finds the cumulative product of each column and returns it in a matrix b, whose dimensions are the same as a.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**
cumprod([1 2 3 4])	[1 2 6 24]
cumprod([1 2 3 4;5 6 7 8])	[1 2 3 4;5 12 21 32]
cumprod([1 2 3 4;5 6 7 8], 2)	[1 2 6 24;5 30 210 1680]

Compatibility
Numeric vectors and arrays

See Also
cumsum (users)

cumsum

Syntax
b = cumsum(a)
b = cumsum(a, dim)

Definition
Returns the cumulative sum of a vector. If a is a vector, then cumsum finds the cumulative sum of all the parts and returns it in a vector, b. If a is a matrix, then cumsum finds the cumulative sum of each column and returns it in a matrix b, whose dimensions

are the same as a.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	Result
cumsum([1 2 3 4])	[1 3 6 10]
cumsum([1 2 3 4;5 6 7 8])	[1 2 3 4;6 8 10 12]
cumsum([1 2 3 4;5 6 7 8], 2)	[1 3 6 10;5 11 18 26]

Compatibility

Numeric vectors and arrays

See Also

cumprod (users)

dbg_print

Syntax

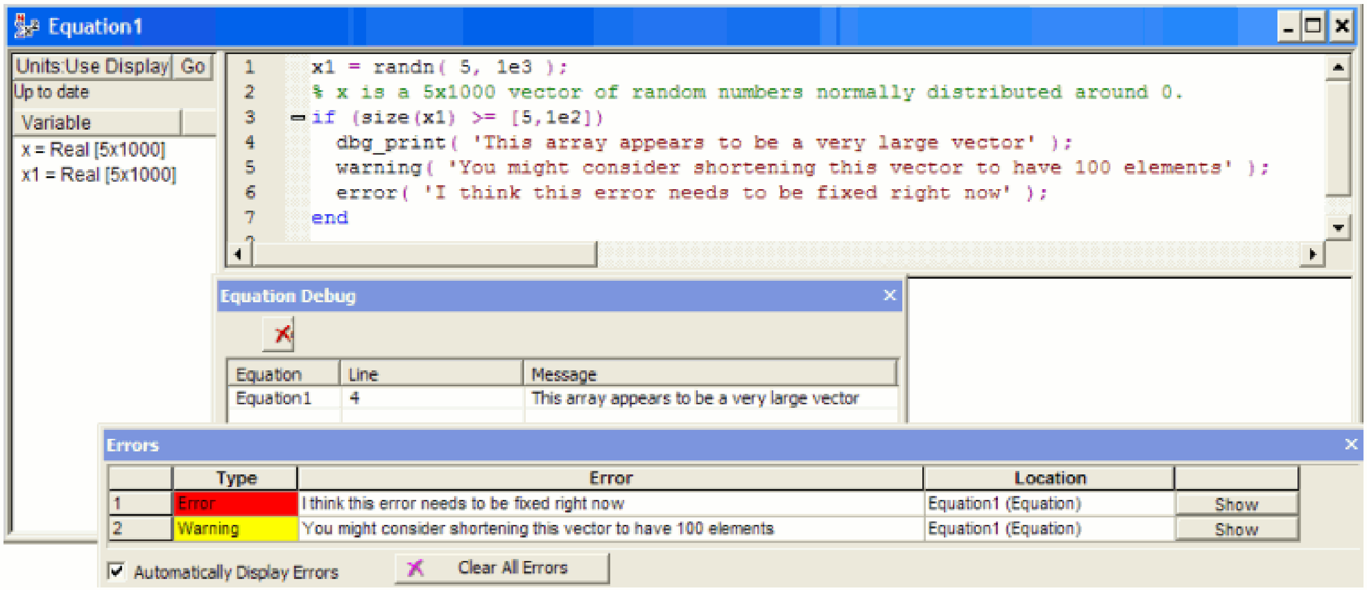
dbg_print('message')

Definition

This function can be invoked from within a set of equations on an equations page in order to report execution status to the **Equation Debug** window. Note that the window for debugging equations is not the same as the **Error Log**. The debug window can be invoked by selecting **View > Advanced Windows > Equation Debug**.

Examples:

Note the difference between the reporting windows and formats for debug and non-debug messages.



x = Real [5x1000]
x1 = Real [5x1000]

```
x1 = randn( 5, 1e3 );
% x is a 5x1000 vector of random numbers normally distributed around 0.
if (size(x1) >= [5,1e2])
  dbg_print( 'This array appears to be a very large vector' );
  warning( 'You might consider shortening this vector to have 100 elements' );
  error( 'I think this error needs to be fixed right now' );
end
```

Equation	Line	Message
Equation1	4	This array appears to be a very large vector

	Type	Error	Location	
1	Error	I think this error needs to be fixed right now	Equation1 (Equation)	Show
2	Warning	You might consider shortening this vector to have 100 elements	Equation1 (Equation)	Show

☑ Automatically Display Errors

Compatibility

string

See Also

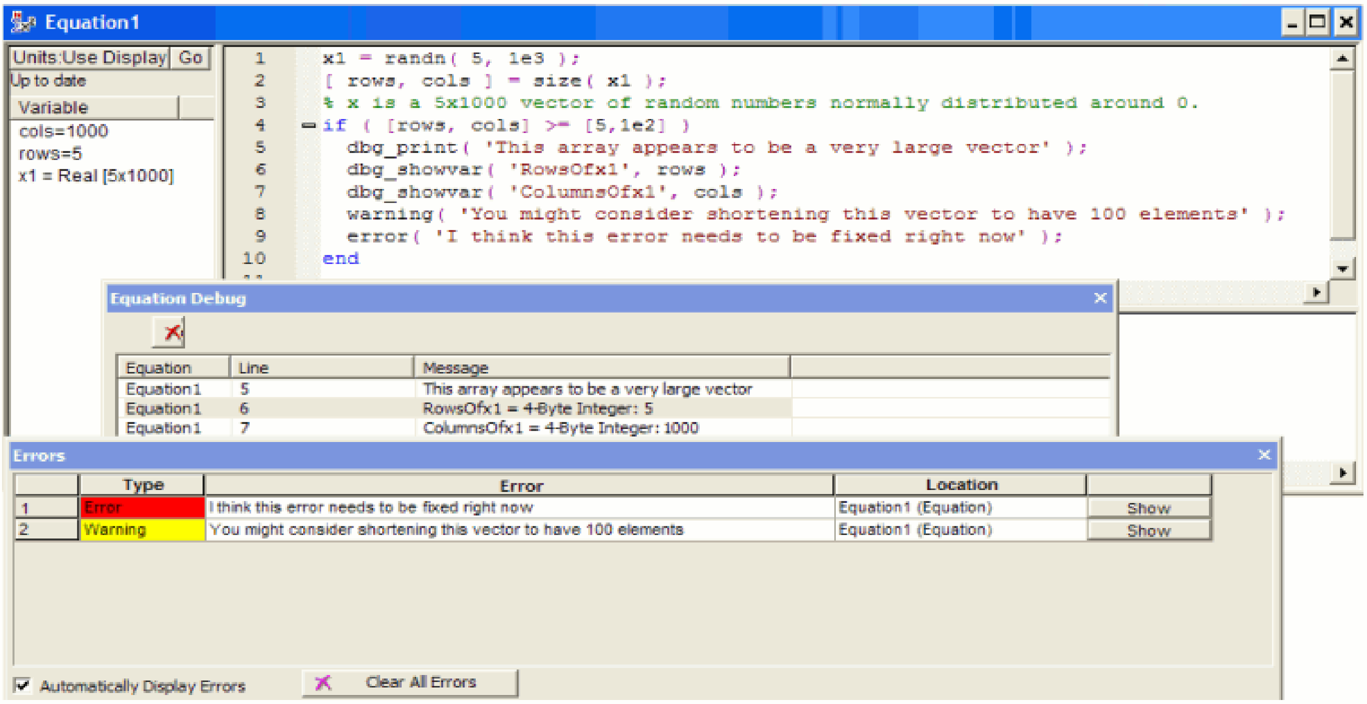
error (users)
dbg_showvar (users)
warning (users)

dbg_showvar

Syntax
dbg_showvar(name, variable)

Definition
This function can be invoked from within a set of equations on an equations page in order to report the current value of a *variable* to the **Equation Debug** window by the supplied *name*. Note that the window for debugging equations is not the same as the **Error Log**. The debug window can be invoked by selecting **View > Advanced Windows > Equation Debug**.

Examples:
In the following example the use model of dbg_showvar() is shown along side that of other relevant Mathematical Language functions.



Variable
cols=1000
rows=5
x1 = Real [5x1000]

```
x1 = randn( 5, 1e3 );
[ rows, cols ] = size( x1 );
% x is a 5x1000 vector of random numbers normally distributed around 0.
if ( [rows, cols] >= [5,1e2] )
  dbg_print( 'This array appears to be a very large vector' );
  dbg_showvar( 'RowsOfx1', rows );
  dbg_showvar( 'ColumnsOfx1', cols );
  warning( 'You might consider shortening this vector to have 100 elements' );
  error( 'I think this error needs to be fixed right now' );
end
```

Equation	Line	Message	
Equation1	5	This array appears to be a very large vector	
Equation1	6	RowsOfx1 = 4-Byte Integer: 5	
Equation1	7	ColumnsOfx1 = 4-Byte Integer: 1000	

	Type	Error	Location	
1	Error	I think this error needs to be fixed right now	Equation1 (Equation)	Show
2	Warning	You might consider shortening this vector to have 100 elements	Equation1 (Equation)	Show

☑ Automatically Display Errors

Formula	Message in Equation Debug
dbg_showvar('Expression', 2+3);	'Expression = 8-byte Real: 5'

string1 = 'hello world';
dbg_showvar('Greeting', string1);
	'Greeting = Array[1x11] of type Char: hello world'

vector1 = [1 2 3];
dbg_showvar('Vector', vector1);
	'Vector = Array[1x3] of type 8-Byte Real: 1 2 3'

array1 = [1 2; 3+2j, 9];
dbg_showvar('Array', array1);
	'Array = Array[2x2] of type 16-Byte Complex: 1 2 3+2i, 9'

cell1 = {'This','is','a','sentence','.'};
dbg_showvar('Cell', cell1);
	'Cell = Array[1x5] of type Variant: [1x4 char] [1x2 char] ['a'] [1x8 char] ['.']'

struct1 = struct('name',{'Jane','Doe'},'AgE', 37);
dbg_showvar('Struct', struct1);
	'Struct = Array[1x2] of type Object:[1x2\ struct] with fields: name AgE'

Compatibility

name - string
variable - any pre-defined variable or expression

See Also
error (users)
dbg_print (users)
warning (users)

deconv

Syntax
[a,b] = deconv(c,d)

Definition
[a,b] = deconv(c,d) deconvolves a vector d out of a vector c and returns it in vector a, and the remainder in b so that c = conv(d,a) + b.

If vectors c and d contain the coefficients of a polynomial, then convolving them is equivalent to mutiplying the polynomials, and deconvolving is equivalent to dividing the polynomials.

Examples:

Formula	**Result**

b = [1 2 3 4]
a = [10 20 30]
[q,r] = deconv(a,b)
	q = [10 20 30]
	r = [0 0 0 0 0 0]

Compatibility
vector

See Also
conv (users)

diag

Syntax
V = diag(x [, a])
v = diag(X)

Definition
If x is a vector, diag(x) gives a matrices V with x on main diagonal. diag(x, a) returns an abs(a)+n (if there are n parts in x) square matrix with the parts of a on the a-th diagonal, main diagonal when a = 0, upper diagonal when a>0, and lower diagonal when a<0.
If X is a matrix, diag(X) returns its main diagonals to a column vector v.

Examples:

Formula	**Result**
diag([2,3])	[2, 0; 0, 3]
diag([1,5], 1)	[0, 1, 0; 0, 0, 5; 0, 0, 0]
diag([1 2 3; 4 5 6; 7 8 9])	[1; 5; 9]

Compatibility
Numeric vectors, Vectors, Matrices

diff

Syntax
A = diff(B)
A = diff(B,r)
A = diff(B,r,dim)

Definition
text here
A = diff(B) returns, in the vector A, the difference between each part in B.
A = diff(B,r) recurses the diff function r times, to find the rth difference.
A = diff(B,r,dim) recurses the diff function r times, to find the rth difference in the scalar dimension dim. If r>= dim, then an empty array is returned.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**

B = [1 5 15 35]
A = diff(B)
	[4 10 20]

N = diff(A)	[6 10]
Z = diff(A,2)	[4]

Compatibility
scalar, vector, array

See Also
prod (users)
sum (users)

eig

Syntax
X = eig(Y)
X = eig(Y,Z)
[U,X] = eig(Y)
[U,X] = eig(Y,Z)
[U,X] = eig(Y,Z,flag)

Definition
X = eig(Y) returns, in vector X, the eigenvalues of the matrix Y.

X = eig(Y,Z) returns, in vector X, the generalized eigenvalues, as long as Y and Z are square matrices.

[U,X] = eig(Y) produces matrices containing the eigenvalues in X, and eigenvectors in U, so: Y*U = U*X

[U,X] = eig(Y,Z) produces a diagonal matrix, X, that contains the generalized eigenvalues, and a full matrix, U, containing the eigenvectors in columns, so: Y*U = Z*U*X

[U,X] = eig(Y,Z,flag) produces the eigenvalues and eigenvectors using a specified algorithm, flag:
'chol' - Computes using Cholesky factorization of Z.
'qz' - Computes using QZ algorithm.

Examples:

Formula	Result
Z =[3 -2 -.9 2*eps; -2 4 1 -eps; -eps/4 eps/2 -1 0; -.5 -.5 .1 1] [VZ,DZ] = eig(Z) [VY,DY] = eig(Z,'nobalance')	Z*VZ - VZ*DZ Z*VY - VY*DY

ellip

Syntax

[num, denom] = ellip(order, passnormripple, stopnormripple, normfreq, ftype, domain)
or
[zeros, poles, gain] = ellip(order, passnormripple, stopnormripple, normfreq, ftype, domain)

Parameters

Name	Definition	Compatibility	Usage	Default	Example
order	order of Butterworth filter	positive integer >= 3	required		5
passnormripple	normalized ripple in passband	positive real	required		0.1
stopnormripple	normalized ripple in stopband	positive real	required		0.1
normfreq	normalized frequency or range of frequencies defining filter	normalized scalar or 2-part vector	required		0.3
ftype	type of filter	enumerated as 'low','high','pass' or 'stop'	optional	'low'	'pass'
domain	digital (Z-domain) or analog (S-domain) filter	'z' or 's'	optional	'z'	's'

Definition

Depending on the list out output arguments, this function delivers a numerator-denominator or a pole-zero-gain definition of an elliptic filter response, which allows controlled amounts of ripples both in the pass and stop bands. Input arguments consist of order, normalized in- and out-of-band ripples, normalized frequency range and the optional enumerated choice of filter type.

Examples:

Note that while zeros and poles are expressed as column vector, numerator and denominator coefficients are expressed as row vectors. Gain is always expressed as a real valued scalar variable.

Formula	zeros	poles	gain	num	denom
ellip(3, 0.1, 0.1, 0.5)	[j; -j; -1]	[j; -j; -0.040]	0.520	[0.520, 0.520, 0.520, 0.520]	[1, 0.040, 1, 0.040]
ellip(3, 0.1, 0.1, 0.5, 'high')	[-j; j; 1]	[-j; j; 0.040]	0.520	[0.520, -0.520, 0.520, -0.520]	[1, -0.040, 1, -0.040]
ellip(3, 0.1, 0.1, [0.25,0.75], 'pass')	[-1/√2+j/√2; 1/√2+j/√2; 1/√2-j/√2; -1/√2-j/√2; 1; -1]	[1/√2+j/√2; 1/√2+j/√2; 0.2; -1/√2+j/√2; -1/√2-j/√2; -0.2]	0.520	[0.520, 0, -0.520, 0, 0.520, 0, -0.520]	[1, 0, -0.040, 0, 1, 0, -0.040]
ellip(3, 0.1, 0.1, [0.25,0.75], 'stop')	[1/√2+j/√2; -1/√2+j/√2; -1/√2-j/√2; 1/√2-j/√2; -j; j]	[1/√2+j/√2; 1/√2-j/√2; j0.2; -1/√2-j/√2; -1/√2+j/√2; -j0.2]	0.520	[0.520, 0, 0.520, 0, 0.520, 0, 0.520]	[1, 0, 0.040, 0, 1, 0, 0.040]

See Also

cheby1 (users)
butter (users)
cheby2 (users)

eps

Syntax

y = eps(m)
y = eps(m, n)
y = eps(m, n, p, ...)
y = eps([m,n,p,...])
y = eps(m, n, p, ..., class)
y = eps([m,n,p,...], class)

Definition

This function is used to create arrays of various sizes containing the default tolerance of a machine in distinguishing between absolute 1.0 and the next higher floating point number. On machines with IEEE floating point arithmetic, the value of eps is 2^(-52) = 2.2204e-16. The function returns a m by n by ... array with every part equal to 2.2204e-16. If only one argument is specified and it is a scalar m, then an m x m matrix is returned. A vector of dimensions may also be passed in. The optional class argument is a string that specifies the data type of the array to return.

Examples:

Formula	Result
y = eps(3 , 2)	y = [2.2204e-16, 2.2204e-16 ; 2.2204e-16, 2.2204e-16 ; 2.2204e-16, 2.2204e-16]
y = eps(2)	y = [2.2204e-16, 2.2204e-16; 2.2204e-16, 2.2204e-16]
y = eps([5 1])	y = [2.2204e-16; 2.2204e-16; 2.2204e-16; 2.2204e-16; 2.2204e-16]

See Also

ones (users)
zeros (users)

erf

Syntax

y = erf(x)

Definition

This function computes the error function of each part of x.
The parts of x must be real.

Examples:

Formula	Result
erf(-1.5)	-0.9661
erf(2)	0.9953
erf([-1; -2; 1.1])	[-0.8427; -0.9953; 0.8802]

Compatibility
Real valued scalars, vectors, arrays

See Also
erfc (users)

erfc

Syntax
y = erfc(x)

Definition
This function computes the complementary error function of each part of x.
The parts of x must be real.

Examples:

Formula	Result
erfc(-1.5)	1.9661
erfc(2)	0.0047
erfc([-1; -2; 1.1])	[1.8427; 1.9953; 0.1198]

Compatibility
Real valued scalars, vectors, arrays

See Also
erf (users)

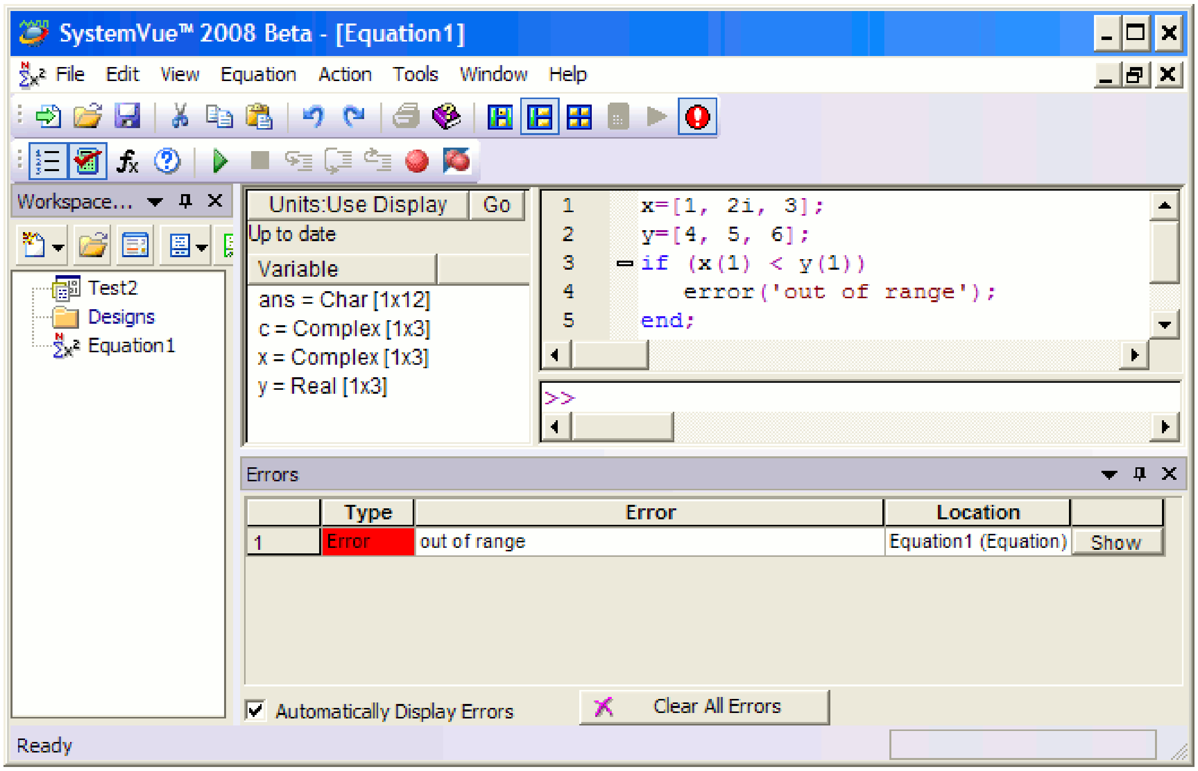
error

Syntax
error('message')

Definition
Posts the error message to the error log and also places the red error symbol on the menu button.

Examples:

Formula	Result
error('out of range')	the message "out of range" is posted to the **Error Log** as an error


SystemVue™ 2008 Beta - [Equation1]
File Edit View Equation Action Tools Window Help
Workspace...
Test2
Designs
Equation1
Units:Use Display
Go
Up to date
Errors
Automatically Display Errors
Clear All Errors
Ready

Variable	
ans = Char [1x12]	
c = Complex [1x3]	
x = Complex [1x3]	
y = Real [1x3]	

```
x=[1, 2i, 3];
y=[4, 5, 6];
if (x(1) < y(1))
    error('out of range');
end;
```

```
>>
```

	Type	Error	Location	
1	Error	out of range	Equation1 (Equation)	Show

Compatibility
Strings

See Also
warning (users)

exist

Syntax
y = exist(Name, Kind, Scope)

Definition
This function checks the existence of a variable or a built-in function. The Name, Kind, and Scope arguments must be strings. Kind and Scope are optional arguments, whereas Name is mandatory. The value of Name must be the name of a variable or built-in function. The exist functions returns 1 if Name is a variable in the Scope, and 5 if it is builtin function, and 0 if the specified Name is not found in the Scope.

If Kind is specified then only that kind is searched for existence. The supported values for Kind are 'var' and 'builtin'.

Default value for an Scope is 'global', a Scope argument can only be specified if Kind = 'var'. The Scope can be either a 'global', a 'local' or the name of a dataset.

Examples:

Formula	Result
iCode = exist('x')	set the variable iCode to 1 if 'x' is a variable name in global scope
iCode = exist('sin')	set the variable iCode to 5 because 'sin' is a built-in function
iCode = exist('sin','var')	set the variable iCode to 0 because 'sin' is a built-in function but it is not of Kind 'var'
iCode = exist('x','builtin')	set the variable iCode to 0 even if the variable named 'x' exist as it is not a built-in function
iCode = exist('S1','var','Design1_Data')	set the variable iCode to 1 if S1 is a variable present in dataset 'Desing1_Data'

Compatibility
Name, Kind, and Scope are strings.

See Also
getvariable (users)
setvariable (users)

exp

Syntax
y = exp(x)

Definition
This function returns the exponential of the argument. The exponential function calculates e to the power of x, where e = 2.7182817... This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
exp(1)	2.718
exp([0 , 1.5])	[1 , 4.482]
exp([-0.5 , 0.5 ; -2 , 2])	[0.607 , 1.649 ; 0.135 , 7.389]

Compatibility
Numeric scalars, Vectors, Arrays. Real and Complex.

eye

Syntax
y = eye(n)
y = eye(m, n)
y = eye(size(A))

Definition
Y = eye(n) returns the n-by-n identity matrix.

Y = eye(m, n) or eye([m n]) returns an m-by-n matrix with 1's on the diagonal and 0's elsewhere.

Y = eye(size(A)) returns an identity matrix the same size as A.

Examples
X = eye(4, 5);

eyediag

Syntax

y = eyediag(x, symbolRate, numCycles, startupDelay)

Definition

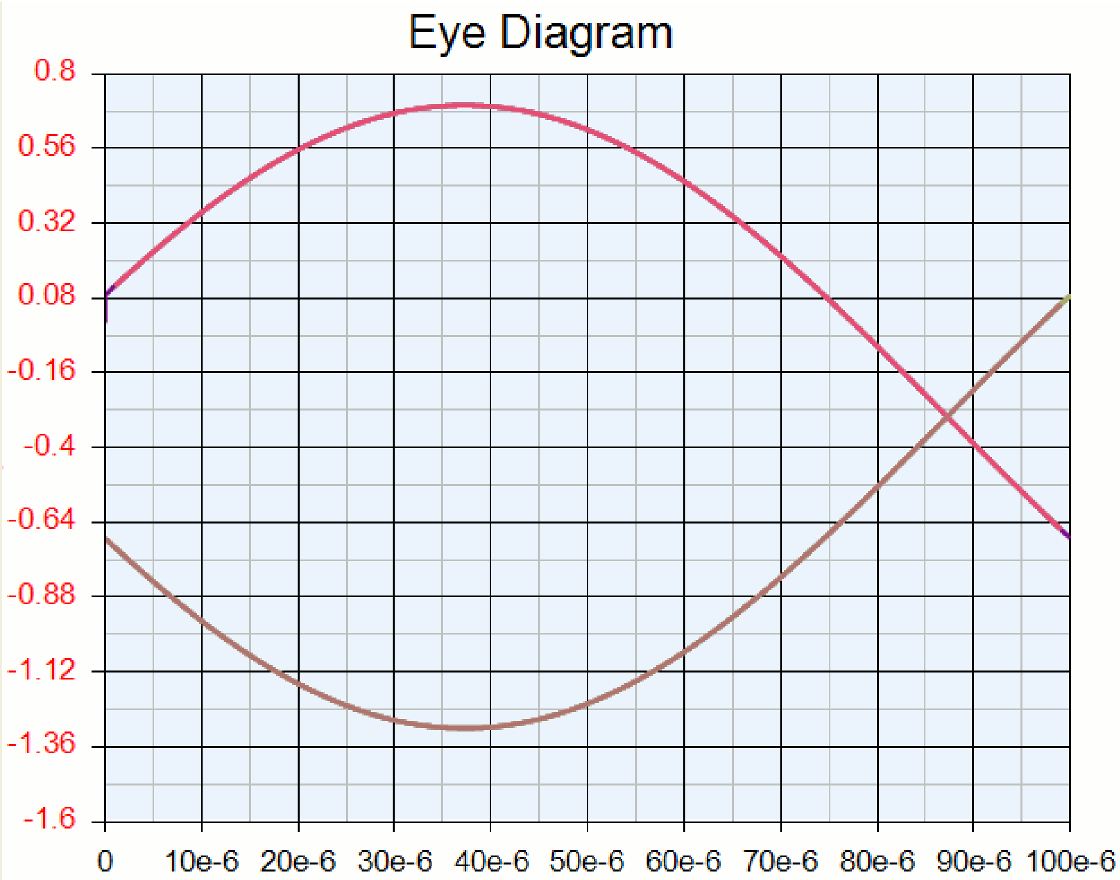
This function builds an eye diagram from a time sequence x.

Parameter	Comment	Unit	Requirement	Compatibility	Default
x	one-dimensional time sequence waveform	V	required	real-valued	
symbolRate	rate of input sequence	Hz	required	real > 0.0	
numCycles	number of unit intervals to be plotted >= 1		optional	integer >=1	1
startupDelay	number of samples that will be removed from beginning of time sequence before plotting >= 0		optional	integer > 0	0

Examples:

y = eyediag(x, 2*5e3, 1, 23)

Note that the following eye diagram was derived from a sinusoid at 5 kHz, so the unit interval was half of the 200 usec period, or just 100 usec. The data-rate or symbol-rate of this simple waveform is therefore 1/unit interval or 10 kHz. The eye-diagram itself was delayed by 23 samples to demonstrate the time-shift propert of this function.



fclose

Syntax

fclose(fileP)

Definition

This function closes the file stream referenced by *fileP* and returns a 0 if the operation is successful.

Examples:

```
fileP = fopen( 'MyFile.txt','r' );
%
% Access first 200 contiguously located floating point numbers
a = fscanf( fileP, '%f', 200 );
%
% Close file
fclose( fileP );
%
```

See Also

fgetl (users)
fgets (users)
fopen (users)
fread (users)
fprintf (users)
fscanf (users)
fwrite (users)
tcpip (users)

fft

Syntax

V = fft(X)
V = fft(X,n)
V = fft(X,[],c)
V = fft(X,n,c)

Definition

Discrete Fourier Transform (DFT) of data. Computed with FFT algorithm when possible.
The parameter len is the FFT length and is optional.
fft(X) gives the discrete Fourier transform of vector X.
fft(X,n) returns the n-point DFT. X adds zeros if n>length of X, X is truncated if n<length of X.
fft(X, [], c) gives the FFT on the dimension c.

Examples:

The following example generates a signal consisting of the sum of two sinusoids: one at 400 Hz, and one at 1500 Hz. The fft function is then used to compute the spectrum of the signal.

```
fft_len = 1024 &nbsp;                         % length of the FFT
fs = 8000                                     % 8000 Hz sampling rate
T = 1/fs                                      % sample time
L = 1000                                      % length of signal
t = (0:(L-1))*T                               % time vector
                                             % x will be the sum of two sinusoids:
                                             % one at 400 Hz and one at 1500 Hz
x = 0.5*cos(2*pi*400*t) + cos(2*pi*1500*t)
X = fft(x, fft_len)                           % spectrum of x
X = X(1:(fft_len/2))                          % we only care about single side\-band  (the rest is
redundant)
f = fs/2 * (0:(2/fft_len):1)
```

The following graph displays the magnitude of X, the spectrum of x.

Compatibility
Vectors, Arrays, Dataset

See Also
ifft (users)

fftshift

Syntax
Y=fftshift(X)

Definition
If X is the output of fft(V):
fftshift(X) moves the zero-frequency to the center,
If X is a vector:
fftshift(X) swaps the left and right
If X is a Matrix:
fftshift(X) swaps the first quadrant with the third and the second quadrant with the fourth.

Examples:

Formula	Result
x=[1 2; 3 4]	y=[4, 3; 2, 1]

Compatibility
Vectors, Arrays

See Also
fft (users)
ifft (users)

fgetl

Syntax
y = fgetl(fileP)

Definition
This function gets the next line from an open file and presents it in a string, after discarding the newline character.

Examples:

```
fileP = fopen( 'MyFile.txt', 'r');
a = fgetl( fileP );
while ( ischar( a ) ) % a will be a number = -1, not a char at end of file
a = fgetl( fileP );
end
fclose( fileP );
```

Compatibility
file pointer

See Also
fclose (users)
fgets (users)
fopen (users)
fread (users)

fprintf (users)
fscanf (users)
fwrite (users)
tcpip (users)

fgets

Syntax
y = fgets(fileP)
y = fgets(fileP, maxChars)

Definition
This function gets the next line from an open file and presents it in a string, including the newline character.

Use the argument *maxChars* to specify the maximum number of character to read. At most_maxChars_ characters will be returned.

Compatibility
fileP - pointer to an open file that is ready for reading
maxChars - maximum number of characters to be read from the next line

See Also
fclose (users)
fgetl (users)
fopen (users)
fread (users)
fprintf (users)
fscanf (users)
fwrite (users)
tcpip (users)

filter

Syntax
y = filter(b,a,X)
[y,zf] = filter(b,a,X)

Definition
y = filter(b,a,X)filters the data in vector/matrix X with the filter described by numerator coefficient vector b and denominator coefficient vector a.
[y,zf] = filter(b,a,X) returns the final conditions, zf, of the filter delays. If X is a row or column vector, output zf is a column vector of max(length(a),length(b))-1.

Examples:

Formula	Result

X = [1:0.4:6]';
windowSize = 4;
filter(ones(1,windowSize)/windowSize,1,X)
	ans =
	? 0.25
	? 0.6
	? 1.05
	? 1.6
	? 2
	? 2.4
	? 2.8
	? 3.2
	? 3.6
	? 4
	? 4.4
	? 4.8
	? 5.2

Compatibility
Vectors, Matrices

See Also
text here

find

Syntax
i = find(A)
I = find(A, n)
I = find(A, n, 'first')
I = find(A, n, 'last')
[r,c] = find(A, ...)
[r,c,x] = find(A, ...)

Definition
i = find(A) returns the indices of all the nonzero parts in array A and places them in vector i.
I = find(A n) returns an n number of indices of all the nonzero parts in an array A and places them in vector i. adding a 'first' argument means that it returns the first n indices of all the nonzero parts, and adding a 'last' argument means that it returns the last n indices.

[r,c] = find(A, ...) finds all the nonzero parts in array A and returns the row location, in r, and column location, in c.
[r,c,x] = find(A, ...) finds all the nonzero parts in array A and returns the row location, in r, and column location, in c, as well as returning the nonzero parts in a vector, x.

Examples:

Formula	Result
find([1, 0, 2, 0, 3, 5])	[1, 3, 5, 6]
find([1, 0, 2; 0, 3, 5], 3)	[1, 4, 5]
find([1, 0, 2; 0, 3, 5], 2, 'last')	[5,6]

Compatibility
Numeric arrays

fir1

Syntax
coefs = fir1(order, bandEdge, filterType, window, normalization)

Definition

This function returns a vector containing n+1 coefficients for a finite impulse response (FIR) filter of *order*=n.

Parameters

Parameter	Description	Requirement	Compatibility	Default	Example
order	order of FIR filter	required	integer >= 1		5
bandEdge	a scalar defining normalized passband edge frequency for lowpass and highpass filters or a 2-part vector defining normalized lower and upper passband edge frequencies for bandpass and bandstop filters	required	0 < real < 1		[0.25 0.75]
filterType	filter response type: lowpass, highpass, bandpass, bandstop	optional	{'low','high','pass','stop'}	'low' if *bandEdge* is a scalar; 'pass' if *bandEdge* is a 2-part vector	'high'
window	a vector containing *order* +1 window coefficients	optional	real	rectangular window with length *order*+1	[1 1 1]
normalization	specifies whether or not the filter passband magnitude is normalized to 0 dB	optional	{'scale','noscale'}	'scale'	'noscale'

If all or any subset of the last three optional parameters are specified, they should be specified in order.

Examples:

Formula	Result
fir1(5,0.1)	[0.0264077,0.140531,0.333061,0.333061,0.140531,0.0264077]
fir1(5,[0.1,0.9],'pass')	[0,-0.134665,0.572442,0.572442,-0.134665,0]

See Also

filter (users)

fix

Syntax

y = fix(x)

Definition

fix rounds the argument toward zero, producing integer. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
fix(2.2)	2
fix(2.2 + 3.3j)	2 + 3j
fix(-2.3 - 3.9j)	-2 - 3j

Compatibility

Numeric scalars, Vectors, Arrays

See Also
floor (users)

flipdim

Syntax
Y = flipdim(X, dim)

Definition
This function returns an array Y which is X flipped along a dimension dim. For example, if dim is 1, X is flipped row-wise down (same as flipud). If dim is 2, X is flipped column-wise left to right (same as fliplr).

Examples:

Formula	**Result**
X = [1 2; 3 4; 5 6] flipdim(X, 1)	Y = [5 6; 3 4; 1 2]
X = [1 2; 3 4; 5 6] flipdim(X, 2)	Y = [2 1; 4 3; 6 5]

Compatibility
Arrays

See Also
flipud (users)
fliplr (users)

fliplr

Syntax
Y = fliplr(X)

Definition
This function returns X, except it flips the columns about the vertical axis, so in the left to right direction.
If X is a column vector, then the function just returns the original X. If X is a row vector, then a vector the same dimensions as X is returned but with the parts flipped left to right.

Examples:

Formula	**Result**

X = [1, 4; 2, 5; 3, 6]
 	
Y = fliplr(X)
	Y = [4, 1; 5, 2; 6, 3]

X = [1, 2, 3, 4, 5]
Y = fliplr(X)
	Y = [5, 4, 3, 2, 1]

X = [1;2;3;4;5]
Y = fliplr(X)
	Y = [1;2;3;4;5]

Compatibility
array

See Also
flipud (users)
flipdim (users)

flipud

Syntax
Y = flipud(X)

Definition
This function returns X, except it flips the rows about the horizontal axis, so in the up-down direction.
If X is a row vector, then the function just returns the original X. If X is a column vector, then a vector the same dimensions as X is returned but with the parts flipped up-down.

Examples:

Formula	**Result**

X = [1, 4; 2, 5; 3, 6;
Y = flipud(X)
	Y = [3, 6; 2, 5; 1, 4]

X = [1;2;3;4;5]
Y = flipud(X)
	Y=[5;4;3;2;1]

X = [1,2,3,4,5]
Y = flipud(X)
	Y = [1,2,3,4,5]

Compatibility
array

See Also
fliplr (users)
flipdim (users)

floor

Syntax
y = floor(x)

Definition
floor returns the largest integer less than or equal to the argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
floor(10)	10
floor(1.5 + 6.2j))	1
floor([-0.5, 0.5])	[-1 , 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
ceil (users)

fopen

Opens a file to read, write or append.

Syntax

fileP = fopen(filename)

fileP = fopen(fileName, operationFormat)

fileP, mess = fopen(filename, operationFormat)

Definition

This function performs file access and returns a handle *fileP* to the beginning of a file whose name is *filename* enclosed in single quotes. The file name can be an absolute path or a relative path. An extension for the file name is optional. The operationFormat is specified in *operationFormat*. Supported operations are:

'r'	Open for reading.
'a'	Open or create a file for writing. Append data the end of the file if content exists.
'w'	Open or create a file for writing. Truncate the file if content exists.
'r+'	Open for reading and writing.
'a+'	Open or create a file for reading and writing. Append data the end of the file if content exists.
'w+'	Open or create a file for reading and writing. Truncate the file if content exists.

If the fopen fails, *fileP* is -1 in contrast to a positive value if the operation was successful.

If two outputs are expected, the first one will be the handle *fileP* and the second one will be an appropriate message indicating whether the file was successfully opened or not.

Note that for binary files, the functions *fread* (users) and *fwrite* (users) should be used for file access.

Example

fileP = fopen('C:\TEMP\test.txt') will open the existing *test.txt* file for reading.

fileP = fopen('C:\TEMP\test.txt', 'w') will create the *test.txt* file and open it for writing.

See Also

fclose (users)
fgetl (users)
fgets (users)
fread (users)
fprintf (users)
fscanf (users)
fwrite (users)
tcpip (users)

fprintf

Syntax
count = fprintf(fid, format, A, ...)

Definition
Formats data from a matrix *A* or set of matrix *...* and writes results to a file *fid*. *count* is the number of elements that were written to the file.

Compatibility
The first argument is a file handle which is returned from a call to fopen, followed by a

format string and then by one or more matrix arguments.

The format string is of the form (only the leading % and *conversionChar* are required):
%{*flags*}{*fieldWidth*}{*.precision*})*conversionChar*

Flags are used to control the alignment of the output. Valid flags are:

Character	Description	Example
Minus sign (-)	Left-justify the output in its field	%-6.4d
Plus sign (+)	Always print a plus or minus sign	%+6.4d
Space character	Inserts a space before the value	% 6.4d
Zero (0)	Pads with zeros rather than spaces	%06.4d

Field Width specifies the minimum number of digits that will be printed for the field

Precision specifies the number of digits to be printed after the decimal point

Conversion Character must be one of the following:

Character	Description
c	Character sequence
d or i	Signed decimal integer
e	Scientific notation (mantise/exponent) using e character
E	Scientific notation (mantise/exponent) using E character
f	Decimal floating point
g	Use the shorter of %e or %f
G	Use the shorter of %E or %f
o	Signed octal
s	String of characters
u	Unsigned decimal integer
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer (capital letters)

See Also
fclose (users)
fgetl (users)
fgets (users)
fopen (users)
fread (users)
fscanf (users)
fwrite (users)
tcpip (users)

fread

Reads binary data from a file.
fread supports both TCP/IP and FILE I/O connections.

TCP/IP

Syntax
cIn = fread(cStream, iValues, cConvert)

Definition

Read some amount of binary data from the stream.

- cStream is a stream class object.
- iValues is the number of values to read.
- cConvert is a string array defining how to read. It can be 'type' to read as this type. It can be '*type' to have both input and output by this type. It can be 'type1=>type2' to have input data interpreted as type1 and output data in type2. By default, input format is byte and output format is double.

Examples:

Formula	**Result**
dOut = fread(t, 12, 'double')	read 12 doubles from the input and save them as doubles (the default). This will consume 96 bytes of input data.
iOut = fread(t, 100, '*int')	read 100 integers from the input and save them as integers. This will consume 400 bytes of input data.
cOut = fread(t, 22, 'uchar=>ushort')	read 22 ascii characters as input and save them as a character array \

FILE I/O

Description

Formula	**Result**
fread(fileP)	reads the contents of the file pointed to by the handle *fileP* (obtained from fopen.) The file is read from beginning to end and *fileP* is finally positioned at the end of the file.
mat=fread(fileP)	does exactly the above and returns a matrix *mat* with the contents of the file.

Compatibility

Scalars, Vectors, Arrays. Real and Complex and Character.

See Also:

fclose (users)
fgetl (users)
fgets (users)
fopen (users)
fprintf (users)
fscanf (users)
fwrite (users)
tcpip (users)
(2 bytes per character for unicode).
This will consume 22 bytes of input data.|

fscanf

Syntax

A = fscanf(fileP, format)
A = fscanf(fileP, format, size)

Definition

This function reads data from a file represented by a file handle *fileP* and converts it to a string using *format*. The result is returned in a matrix *A*.

An optional argument can be passed *size*, to specify the amount of data in the resulting matrix.

Compatibility

fileP - file pointer to an open file ready for reading
format - string description of format in which to access contents of file, e.g. '%f' for floating-point

size - positive integer specifying number of parts to be read in *readFormat*
size can be in the form:

n	read at most n elements from the file
inf	read to the end of the file
[m,n]	read at most m*n elements. Fill at most m rows in *A*

The format string consists of an initial % character and at a minimum a *conversion character*. Optional characters can be entered between the % and the *conversion character*.

digit	Maximum field width
*	Skip over the match value for this format. The value much match but will be ignored and not added to *A*

Valid *conversion characters* are:

c	Character sequence
d or i	Signed decimal integer
e	Scientific notation (mantise/exponent) using e character
E	Scientific notation (mantise/exponent) using E character
f	Decimal floating point
g	Use the shorter of %e or %f
G	Use the shorter of %E or %f
o	Signed octal
s	String of characters
u	Unsigned decimal integer
x	Unsigned hexadecimal integer
X	Unsigned hexadecimal integer (capital letters)

See Also

fclose (users)
fgetl (users)
fgets (users)
fopen (users)
fprintf (users)
fread (users)
fwrite (users)
tcpip (users)

fwrite

Writes binary data to a file.
fwrite supports both TCP/IP and FILE I/O operations.

TCP_IP

Syntax

iWritten = fwrite(cStream, Value)

iWritten = fwrite(cStream, Value, Mode)

iWritten = fwrite(cStream, Value, Precision, Mode)

Definition
Write some amount of binary data to the stream.

- cStream is a stream class object.
- Value is the data to write.
- Mode can be 'sync' or 'async', default is async.
- Precision is a char array defining the output data type. Default is byte.

Examples:

Formula	**Result**
iOut = fwrite(t, 12)	Write the value 12 to the stream as a single byte.
iOut = fwrite(t, [1, 2], 'int', 'async')	Write the vector [1 2] to the stream as two integers.
iOut = fwrite(t, 22, 'sync')	Write the value 22 synchronously to the stream as a single byte.

FILE I/O

Description

Formula	**Result**
fwrite(fileP, mat)	will write the contents of matrix *mat* to the file pointed to by the handle *fileP* (obtained from fopen.) Data is written to the file in column order.
counter=fwrite(fileP, mat)	will do exactly the above. In addition it will return a counter with the number of elements successfully written to the file.

Note: Until the file is closed using the *fclose* (users) function, the contents of that fill cannot be viewed.

Compatibility
Scalars, Vectors, Arrays. Real and Complex and Character.

See Also
fclose (users)
fgetl (users)
fgets (users)
fopen (users)
fprintf (users)
fread (users)
fscanf (users)
tcpip (users)

gausswin

Syntax
c = gausswin(L)
c = gausswin(L,alpha)

Definition
This function returns, in the column vector c, a Gaussian window with L-points and a window width parameter *alpha*. The default value of *alpha* is 2.5. The width of the window is inversely related to the value of *alpha* as shown in the graph below.

```
_gausswin_at_n_of_L_with_alpha_ = exp( -0.5 * (2 * alpha * n / N ) ^ 2 )
where -N/2 <= n <= N/2, L = N+1
```

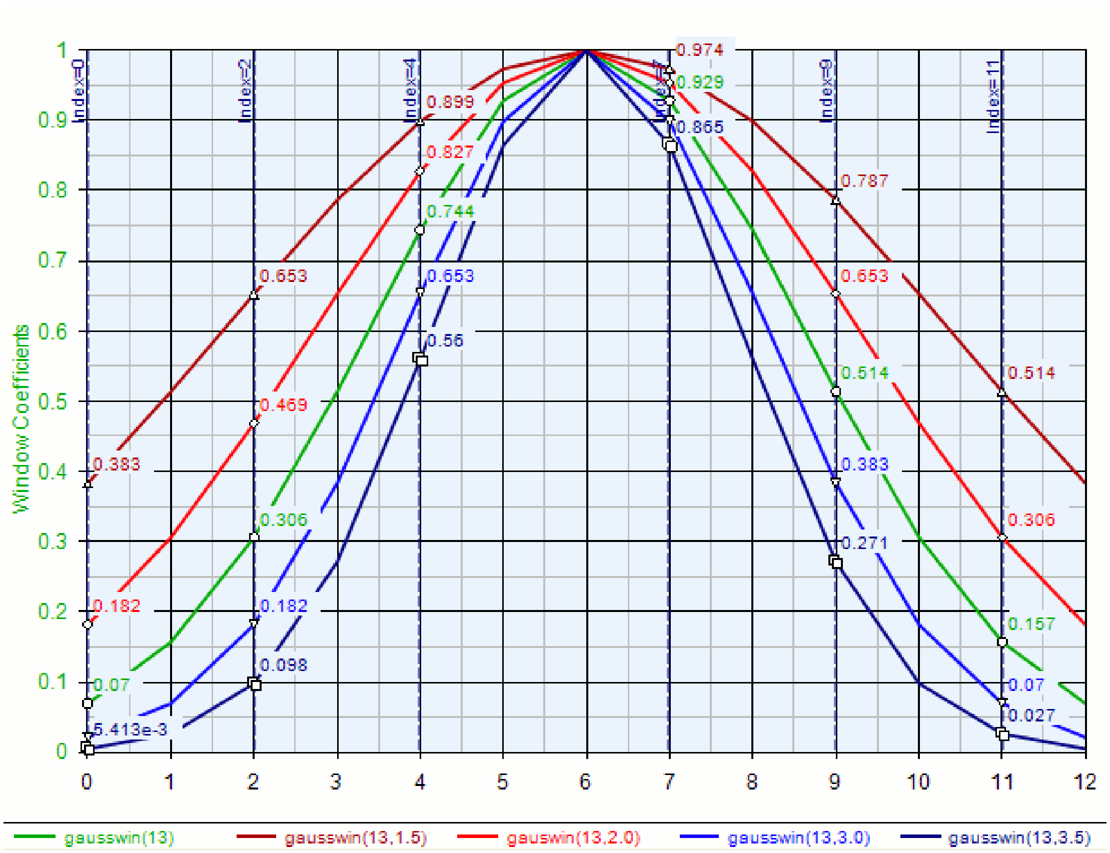
When L is odd valued the apex of 1 is reached by the central sample. When L is even the two samples flanking the unsampled apex have a value of less than 1.

> **Note**
>
> gausswin(2), a redundant usage of this function returns [0.458 0.458], whereas gausswin(1) returns [1].

Examples:

In the following graph, 13-point Gaussian windows are overlaid for *alpha* in the range [1.5,3.5]. Vector values at each sample point are shown. The default behavior of *alpha* =2.5 is shown in green.

Note that the values at the end points of the vector are not forced to zero but rather determined by the value of *alpha*.



Compatibility

scalar

See Also:

bartlett (users)
blackman (users)
hamming (users)
hann (users)
rectwin (users)

getindep

Syntax

y = getindep(x)

Definition

Returns a string with the name(s) of the independent variable(s). x is the variable to check.

Examples:

Formula	Result
n=getindep(S)	if S is a linear analysis result this will usually return "Linear_Data\Eqns\VarBlock\F" (the longname of F)
n=getindep(VPORT)	in a HARBEC analysis this will return "HbData\Eqns\VarBlock\Freq" - the Frequency vector

Compatibility

Swept vectors, arrays

See Also

setindep

getindepvalue

Syntax

y = (x)

Definition

text

Examples:

Formula	Result

Compatibility

text

See Also

text

getunits

Syntax

y = getunits(x)

Definition

Returns an integer corresponding to the units of a variable x. This integer may be used by setunits.

Examples:

Formula	Result

z = 1
setunits("z" , "V")
y = getunits(z)
	y = 9001

z = 1
setunits("z" , "mil")
y = getunits(z)
	y = 6002

z = 1
setunits("z" , "H")
y = getunits(z)
	y = 4003

Compatibility

Numeric scalars, vectors, arrays

See Also

setunits

getvariable

Syntax

y = getvariable(Dataset, Variable)
[y, yindep] = getvariable(Dataset, Variable)

Definition

This function gets a variable value (and, optionally, the value of its independent variable) from a dataset. The Dataset and Variable arguments must be strings. If an independent value is requested but the referenced variable doesn't have one, a warning is issued and yindep is set to a blank value.

Examples:

Formula	Result
OutVar = getvariable('OutData', 'OutVar')	set the variable OutVar from the dataset variable OutData.OutVar
myVar = getvariable('Out', 'Var')	set the variable myVar from the dataset variable Out.Var
[myVar, myIndep] = getvariable('Out', 'Var')	set the variable myVar from the dataset variable out.Var and set myIndep to Out.Var's independent value

Compatibility

Dataset and Variable are strings.

See Also

setvariable (users)

hamming

Syntax

c = hamming(L)

Definition

This function returns a Hamming window with L points into a column vector, c.

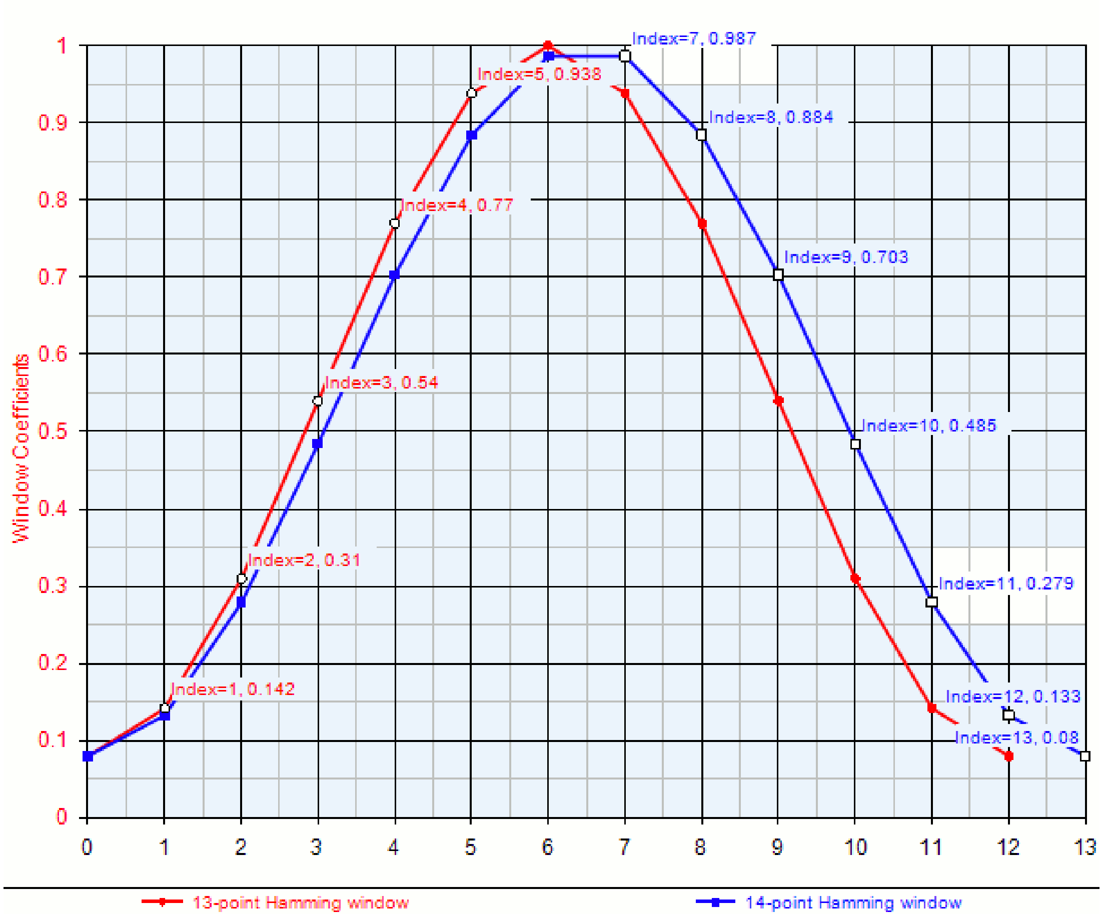
```
_hamming_value_at_n_of_L_symmetric_ = 0.54 - 0.46 * cos( 2*pi*n/N )
where 0 <= n <= N
```

Note that the end points of the vector is not always 0. When N is odd, the apex of 1 is explicitly an part of the window function. When N is even, the apex is not explicitly sampled but rather the two sample points which flank the apex are represented in the returned vector.

Note

hamming(2) a redundant usage of this function returns [0.08 0.08] whereas hamming(1) returns [1].

Examples:



Compatibility
scalar

See Also:
bartlett (users)
blackman (users)
gausswin (users)
hann (users)
rectwin (users)

hankel

Syntax
hm = hankel(col)
hm = hankel(col,row)

Definition
This function returns a Hankel matrix whose first column is defined by the argument *col* and whose parts start at zero below the first anti-diagonal.
If the *row* argument is specified, then the first column is *col* , and last row is *row*. If the last part in *col* suppresses the first part in *row*, if the two values happen to be different.

Examples:

```
col = [1,2,5];
row = [7,8,9,10];
% Note that col(length(col)) == 5 ~= 7 == row(length(row)),
% therefore, the (3,1), (2,2) and (1,3) parts of the Hankel matrix will be 5.
hm = hankel( col, row )
% hm = [  1,  2,  5,  8 ;
%         2,  5,  8,  9 ;
%         5,  8,  9, 10 ]
% Note how the left-upper triangular section of the matrix corresponds to the column vector
% and the right-lower triangular section corresponds to the row vector.
```

Compatibility:
col - Numeric valued vector
row - Numeric valued vector

hann

Syntax
c = hann(L)

Definition
This function returns a hann window with L points into a column vector, c.

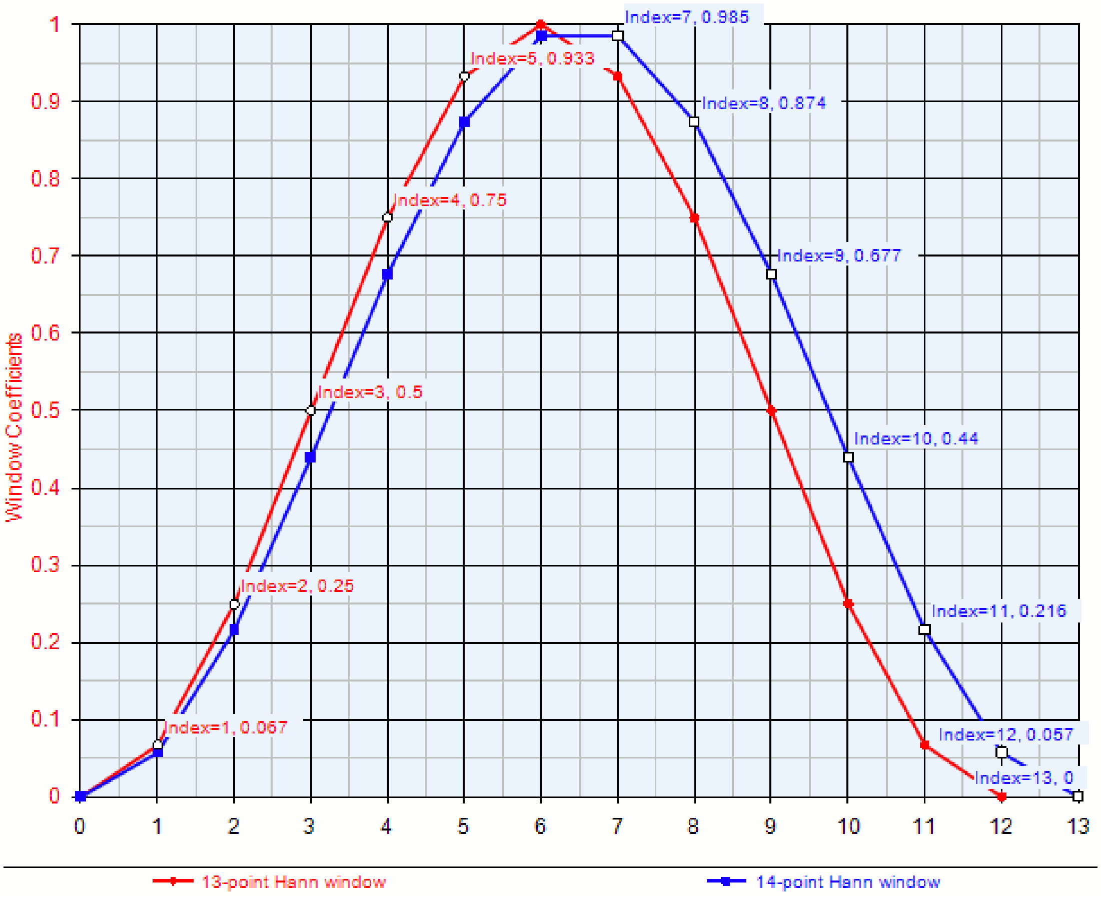
```
_hann_value_at_n_of_L_symmetric_ = 0.5 * ( 1 - cos( 2*pi*n/N) )
where 0 <= n <= N
```

Note that the end points of the vector are always 0. When N is odd, the apex of 1 is explicitly an part of the window function. When N is even, the apex is not explicitly sampled but rather the two sample points which flank the apex are represented in the returned vector.

Note

hann(2) a redundant usage of this function returns [0 0] whereas hann(1) returns [1].

Examples:



Compatibility
scalar

See Also:
bartlett (users)
blackman (users)
gausswin (users)
hamming (users)
rectwin (users)

hilbert

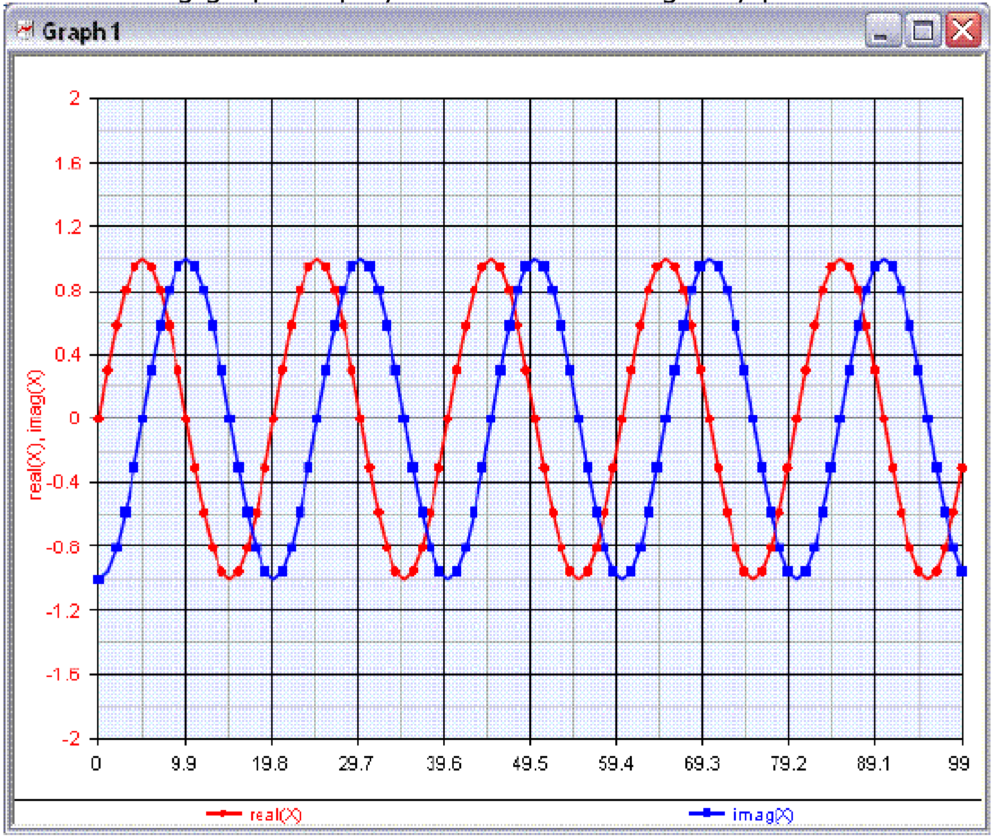
Syntax:
V = hilbert(X)
V = hilbert(X,n)
V = hilbert(X,[],c)
V = hilbert(X,n,c)

Definition:
Computes the *analytic signal* from a real data vector using the Hilbert Transform, where the Discrete Fourier Transform is used to calculate the Hilbert Transform. In the resulting complex vector the original real vector values are stored in the real part, and the imaginary part is the Hilbert Transform of the real vector.
hilbert(X) calculates the *analytic signal* of vector X. If X is a matrix, the *analytic signal* is computed for each column.
hilbert(X,n) returns the n-point *analytic signal*. X is extended by adding zeros if n>length of X, X is truncated if n<length of X.
hilbert(X, [], c) calculates the *analytic signal* on the dimension c.

Examples:
The following example creates a simple sinusoid at 400Hz then generates the analytic signal from that waveform. The resulting complex vector will contain the original sine wave as the real part, and a cosine wave (the Hilbert transform) as the imaginary part.

```
fs = 8000                                   % 8000 Hz sampling rate
T = 1/fs                                    % sample time
L = 100                                     % length of signal
t = (0:(L-1))*T                             % time vector
x = sin(2*pi*400*t)                         % sine wave at 400 Hz
X = hilbert(x)                              % analytic signal calculation
```

The following graph displays the real and imaginary parts of X.



Compatibility:
Vectors, Arrays, Dataset

See Also:

histc

Syntax
y = histc(x,e)
y = histc(x,e,dim)
[y,bin] = histc(x,e)

Definition
This function provides a count of the number of parts of a numeric real-valued vector or array *x* that fall into each histogram bin where the histogram itself is defined by the bin boundaries defined in the vector *e*. By definition *y* is a vector of integers. If *x* is a multi-dimensional array then the dimension of binning may be included as an optional third

argument *dim*. If *dim* is omitted, the innermost non-singleton dimension is chosen as the dimension to operate along.

If the function is called with an optional *bin* output variable, then the actual binning matrix is returned in addition to the bin count vector *y*.

Examples:

```
% Define a large vector of normally-distributed random variables, with mean = 0
x = randn( 1, 1e5 );
% Detect the number that is farthest from 0
elim = max( abs( x ) );
% Create binning vector with bin span being one
e = [-elim-1:elim+1]
% Invoke histogram count
y = histc( x, e );
% y = [0,4,151,2400,14546,34846,33406,12648,1881,117,1,0]
% Note that the normalized distribution of values is captured in the vector y.
%
% Define a large 2-D array of bi-normally distributed random variables with mean = 0,0
x = randn( 3, 1e3 );
% Detect the number that is farthest from 0 along any radius
elim = max( abs( x );
% Create binning vector with bin span being one
e = [-elim-1:elim+1]
% Invoke histogram count row-wise (along dim=2)
y = histc( x, e, 2 );
% y = [0,0,2,1,5,25,65,109,174,182,172,140,78,35,7,4,0,0,1,0;
%       0,0,0,2,11,13,59,113,156,206,193,124,68,33,20,2,0,0,0,0;
%       0,0,0,4,9,27,59,118,162,189,180,127,81,31,8,3,2,0,0,,0];
%
% Note that the normalized distribution of values is captured in each row of y.
```

ifft

Syntax
ifft(data, len)

Definition
Inverse Discrete Fourier Transform (IDFT) of data. Computed with IFFT algorithm when possible. The parameter len is the IFFT length and is optional.

Examples:
The following example code is taken from the fft example:

```
fft_len = 1024 &nbsp;                          % length of the FFT
fs = 8000                                      % 8000 Hz sampling rate
T = 1/fs                                       % sample time
L = 1000                                       % length of signal
t = (0:(L-1))*T                                % time vector
                                             % x will be the sum of two sinusoids:
                                             % one at 400 Hz and one at 1500 Hz
x = 0.5*cos(2*pi*400*t) + cos(2*pi*1500*t)
X = fft(x, fft_len)                            % spectrum of x
X = X(1:(fft_len/2))                           % we only care about single side\-band  (the rest is
redundant)
f = fs/2 * (0:(2/fft_len):1)
```

If the following lines of code are now added:
y = ifft(X, fft_len)

then y and x would be identical.

Compatibility
Dataset

See Also
fft (users)

ifftshift

Syntax
ifftshift(X)

Definition
Inverse FFT shift.
ifftshift(X) swaps the left and right halves of the vector X. For matrices, ifftshift(X) swaps the first quadrant with the third and the second quadrant with the fourth.

Examples:

Formula	Result
x=[1 2]	? ans = ? 2 1
x=[1 2;3 4]	? ans = ? 4 3 ? 2 1

Compatibility
Vectors, Matrices

See Also
fftshift (users)
fft (users)
ifft (users)

imag

Syntax
y = imag(x)

Definition
imag returns the imaginary part of a complex number. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
imag(2 - 5j)	-5
imag([10 + 1j , 12])	[1 , 0]
imag([20 + 3j; 1 + 2j])	[3 ; 2]

Compatibility
Numeric scalars, Vectors, Arrays

inf

Syntax

DA = Inf(n, dist)
DA = Inf(m, n, dist)
DA = Inf(..., classname, dist)

Definition
this function creates an n by n, or m by n, array of class double.
The classname parameter is for specifying the underlying class, which can be either 'double', the default, or 'single'.

Examples:

Formula	Result
x = inf	1.#IOe

Compatibility
Numeric

interp1

Syntax
y2 = interp1(x1,Y1,x2)
y2 = interp1(x1,Y1,x2,method)
y2 = interp1(x1,Y1,x2,method,'extrap')
pp = interp1(x,Y,method,'pp')

Definition
y2 = interp1(x1,Y1,x2) interpolates to find y2, the values of the underlying function Y1 at the points in the vector x1.
method:
'nearest' Nearest neighbor interpolation
'linear' Linear interpolation (default)
'spline' Cubic spline interpolation
'pchip' Piecewise cubic Hermite interpolation
'cubic' (Same as 'pchip')

yi = interp1(x,Y,xi,method,'extrap') uses the specified method to perform extrapolation for out of range values.

Examples:

Formula	Result

x1=[1 2 4 5]
y1=[34 56 67 77]
y2=interp1(x1,y1,3)
	y2
	? ans =
	? 61.5

x1=[1 2 4 5]
y1=[34 56 67 77]
y2=interp1(x1,y1,-1,'linear','extrap')
	y2
	? ans =
	? -10

Compatibility
Numeric scalars, Vectors

See Also
pchip (users)
spline (users)

ipermute

Syntax

b = ipermute(Array, PermutedIndexes)

Definition
Inverse permute dimensions of an array. This inversley permutes Array using the vector of permuted indexes.

Examples:

Formula	**Result**

r=[1,2,3,4; 5,6,7,8]
u=ipermute(r, [2,1])
	>> r
	ans =
	1 2 3 4
	5 6 7 8
	>> u
	ans =
	1 5
	2 6
	3 7
	4 8

r=[1,2,3,4;5,6,7,8]
a=[r , r]
b=reshape(a, [2,4,2])
u=ipermute(b, [2,3,1])
	>> b
	ans(:,:,1) =
	1 2 3 4
	5 6 7 8
	ans(:,:,2) =
	1 2 3 4
	5 6 7 8
	>> u
	ans(:,:,1) =
	1 5
	1 5
	ans(:,:,2) =
	2 6
	2 6
	ans(:,:,3) =
	3 7
	3 7
	ans(:,:,4) =
	4 8
	4 8 >>

Compatibility
Numeric Arrays.

See Also
permute (users)

iscell

Syntax
y = iscell(x)

Definition
This function determines whether the given parameter *x*, is a cell or an array of cells. If so, it returns true, logical 1, and if not it returns false, logical 0.

Examples:

Formula	Result	Comment
iscell(23)	0	scalar is not a cell
iscell(1:10)	0	10-part row-vector is not a cell
iscell('hello')	0	string is not a cell
iscell(['hello','there'])	0	array of strings is not a cell
iscell({'hello'})	1	single part cell
iscell({'hello','there'})	1	array of cells

Compatibility
Numeric and string valued variables.

See Also:
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

ischar

Syntax
y = char(x)

Definition
This function determines whether the given parameter *x*, is a character or array of characters. If so, it returns true, logical 1, and if not it returns false, logical 0.

Examples:

Formula	Result	Comment
ischar(2)	0	scalar is not a character
ischar('2')	1	character
ischar(1:10)	0	numeric vector is not an array of characters
ischar('hello')	1	vector of characters
ischar(['hello','table'])	1	array of characters
ischar({'hello','table'})	0	cell is not an array

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)

isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isempty

Syntax
y = isempty(x)

Definition
This function returns true if *x* is an empty array and false otherwise. An empty array has at least one dimension of size zero, for example, 0-x-0 or 0-x-5. This function does not operate on strings or cells. So supplying an empty string to the function does not get a logical true.

Examples:

Formula	Result
isempty(rand(2,2))	0

b(:,:) = [];
a = isempty(b)
	a = 1;

Compatibility
Numeric scalars, vectors, arrays.

isequal

Syntax
out = isequal(a, b[, ...])

Definition
isequal returns true if the input arrays have the same contents, and false otherwise. Nonempty arrays must be of the same data type and size to be compared.

Examples:

Formula	Result

a = [1,2;3,4]
b = [1,2;3,4]
out = isequal(a,b)
	out=1;

Compatibility
Arrays and scalars.

isequalwithequalnans

Syntax
out = isequalwithequalnans(a, b[, ...])

Definition
isequalwithequalnans returns true if the input arrays have the same contents, and false otherwise. Nonempty arrays must be of the same data type and size to be compared. NaN values are not ignored, and considered to be equal to each other.

Examples:

Formula	**Result**

a = [1,2;NaN,4]
b = [1,2;NaN,4]
out = isequalwithequalnans(a,b)
	out=1;

Compatibility

Arrays and scalars.

isfield

Syntax

y = isfield(x,'fieldname')

y = isfield(x, {'fieldname1','fieldname2',...,'fieldnameN'})

Definition

This function examines the structure, *x*, to confirm whether it contains the field specified by 'fieldname'. It returns a logical 1 if the field exists and a logical 0 otherwise. When multiple field names are specified in a cell array, then an array is returned with the corresponding logical values.

Examples:

```
patient.name = 'John Doe';
patient.billing = 127.00;
y1 = isfield(patient, 'billing')
% y1 = 1
y2 = isfield(patient, {'billing','date of birth','name'})
% y2 = [1, 0, 1]
```

Compatibility

structure, string, cell array

See Also:

iscell (users)
ischar (users)
isempty (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isfinite

Syntax

b = isfinite(Array)

Definition

isfinite returns an array the same size as Array containing true where the parts of Array are finite and false where they are infinite or NaN. For a complex number z, isfinite(z) returns true if both the real and imaginary parts of z are finite, and false if either the real or the imaginary part is infinite or NaN.

Compatibility
Numeric arrays

See Also
isinf (users)

isfloat

Syntax
y = isfloat(x)

Definition
This function determines whether the given parameter *x*, is a floating point number. If so, it returns true, logical 1, and if not, it returns false, logical 0. When *x* is a character or a string, this function returns 0 because the argument is not an explicit numeric value but *isreal* (users) returns 1 because the argument is implicitly real valued because ASCII characters are involved in scalar or vector format.

Examples:

Formula	**Result**	**Comment**
isfloat(23)	1	scalar is a 1-part vector
isfloat(1:0.5:10)	1	row-vector of floating point numbers
isfloat([2+3i;4])	1	column-vector of real and complex numbers
isfloat('h')	0	ASCII character is not a floating point numeric value
isfloat('hello')	0	string is a vector of ASCII characters, not numeric values

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isinf

Syntax
out = isinf(Array)

Definition
isinf returns an array the same size as Array containing true where the parts of Array are +Inf or -Inf and false where they are finite. For a complex number z, isinf(z) returns true if either the real or imaginary part of z is infinite, and false if both the real and imaginary

parts are finite or NaN. For any real a, exactly one of the three quantities isfinite(a), isinf(a), and isnan(a) is true.

isinteger

Syntax
y = isinteger(x)

Definition
This function determines whether the given parameter *x*, is an integer. If so, it returns true, logical 1, and if not it returns false, logical 0. When applied to a multi-part array, all parts must be integers for the function to evaluate to a true.

Examples:

Formula	**Result**	**Comment**
isinteger(-23)	1	is an integer
isinteger(1:10)	1	10-part row-vector of integers
isinteger(1:0.5:2)	0	contains some non-integers

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

islogical

Syntax
y = islogical(x)

Definition
This function determines whether the given expression *x*, is evaluates to a binary logical value. If so, it returns true, logical 1, and if not it returns false, logical 0.

Examples:

Formula	**Result**	**Comment**
islogical(1)	0	numeric scalar not a logical expression even though it is binary valued 1
islogical(2>3)	1	is a logical expression
islogical([3,4] < [5,6])	1	is a logical expression

Compatibility
Numeric and string valued variables.

See Also:

iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isnan

Syntax
out = isnan(Array)

Definition
isnan returns an array the same size as Array containing true where the parts of Array are NaN (not-a-number). For any real a, exactly one of the three quantities isfinite(a), isinf(a), and isnan(a) is true.

isnumeric

Syntax
y = isnumeric(x)

Definition
This function determines whether the given parameter *x*, is of numeric value. If so, it returns true, logical 1, and if not it returns false, logical 0. Strings, cells and structures are not numeric parts.

Examples:

Formula	Result	Comment
isnumeric(23)	1	scalar is a 1-part vector
isnumeric(1:10)	1	10-part row-vector
isnumeric('hello')	0	string is not a numeric array
isnumeric({23})	0	cell is not a numeric part

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isreal

Syntax
y = isreal(x)

Definition
This function determines whether the given parameter *x*, is a real valued number or a vector or array containing only real numbers. If so, it returns true, logical 1, and if not, it returns false, logical 0.

Examples:

Formula	Result	Comment
isreal(23)	1	integer is real valued
isreal(1:0.5:10)	1	10-part real-valued row-vector
isreal([3;4+5i;6])	0	has one complex valued part
isreal('h')	1	character has an ASCII value
isreal('hello')	1	string is an array of ASCII values
isreal({'hello'})	0	cell is not a numeric part
isreal({1.4})	0	cell is not a numeric part

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isscalar (users)
isstr (users)
isstruct (users)
isvector (users)

isscalar

Syntax
y = isscalar(x)

Definition
This function determines whether the given parameter *x*, is a 1x1 part with an ASCII value i.e. a scalar. If so, then it returns true, logical 1, and if not then it returns false, logical 0.

Examples:

Formula	**Result**	**Comment**
isscalar(23)	1	is a scalar
isscalar(1:10)	0	10-part row-vector
isscalar('d')	1	is an ASCII character
isscalar('hello')	1	string is not a scalar
isscalar({1})	1	is a 1-part cell
isscalar({'This is a sentence'})	1	is also a 1-part cell
isscalar({'This','is','a','sentence','.'})	0	is a multi-part cell

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isstr (users)
isstruct (users)
isvector (users)

isstr

Syntax
y = isstr(x)

Definition
This function determines whether the given parameter *x*, is a string. If so, then it returns true, logical 1, and if not then it returns false, logical 0.

Examples:

Formula	**Result**	**Comment**
isstr(23)	0	scalar is not a string
isstr('hello')	1	is a string
isstr({'This','is','a','sentence','.'})	0	array of cells is not a string
isstr({'This'})	0	even single string part in cell is not a string

Compatibility
Numeric and string valued variables.

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)

isscalar (users)
isstruct (users)
isvector (users)

isstruct

Syntax
y = isstruct(x)

Definition
This function determines whether the given parameter *x*, is a structure and if so it returns true, logical 1. Otherwise, it returns false, logical 0.

Examples:

Formula	**Result**	**Comment**
isstruct(2)	0	scalar is not a structure
isstruct([2,3])	0	vector is not a structure
isstruct([2,3;4,5])	0	array is not a structure
isstruct('hello')	0	string is not a structure

```
type.greeting='hi!';
type.date=[01 29 2009];
y = isstruct(type);
```

isstruct(type)	1	type is a structure with fields 'greeting' and 'date'

See Also:
iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isvector (users)

isvector

Syntax
y = isvector(x)

Definition
This function determines whether the given parameter *x*, is a vector. If so, it returns true, logical 1, and if not it returns false, logical 0.

Examples:

Formula	Result	Comment
isvector(23)	1	scalar is a 1-part vector
isvector(1:10)	1	10-part row-vector
isvector([2;3;4])	1	3-part column-vector
isvector([1,2:3,4])	0	2-dimensional array, not a vector
isvector('hello')	1	string is a 5-part vector
isvector({'This','is','a','sentence','.'})	1	cell is a 5-part vector of strings

Compatibility

Numeric and string valued variables.

See Also:

iscell (users)
ischar (users)
isempty (users)
isfield (users)
isfloat (users)
isinteger (users)
islogical (users)
isnumeric (users)
isreal (users)
isscalar (users)
isstr (users)
isstruct (users)

kurtosis

Syntax

y = kurtosis(x)
y = kurtosis(x, Flag)
y = kurtosis(x, Flag, iDim)

Definition

Returns the sample kurtosis of a vector x. Kurtosis is the fourth central moment of X divided by the fourth power of the standard deviation.

If Flag is 0 (default), kurtosis normalizes by N-1 where N is the sample size. If Flag is 1, kurtosis normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

The iDim argument is optional and specifies which dimension to operate along. For example, if iDim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	Result
y = kurtosis([3 ; 4 ; 8 ; 9])	y = 1.1479
y = kurtosis([1, 2, 3], 1)	y = 1.5

Compatibility

Numeric arrays

See Also

std (users)
var (users)
skewness (users)

length

Syntax

y = length(x)

Definition

This function returns the longest dimension of the array *x*. When presented with a single string, it returns the character count. When presented with a list of strings it returns list length even if one of the words has a character count (inner dimension) greater than the word count of the string (outer dimension).

Examples:

Formula	**Result**	**Comment**
length(16)	1	scalar number
length([1 2 3])	3	3-length vector
length([1 2 3; 4 5 6])	3	2x3 matrix, number of colmns > number of rows
length('hello')	5	string length is 5
length({'This','string','is','a','test','string','.'})	7	word count is 7, all words have character count < 7
length({'This','string','is','a','verylongwordedtest','string','.'})	7	word count is 7, even though one word has character count > 7

Compatibility

Numeric and string scalars, vectors, arrays

See Also

size (users)

linspace

Syntax

y = linspace(u,v)
y = linspace(u,v,x)

Definition

This function creates vectors that have values that are linearly spaced, similar to the colon operator. However, unlike the colon operator, this function gives control on specifying the number of points. The points are generated between, and including, *u* and *v*. The number of points generated are determined by the parameter *x*. If not specified, this value defaults to 100.

Examples:

Formula	**Result**
y = linspace(1, 10, 10)	Y = [1,2,3,4,5,6,7,8,9,10]

Compatibility

u - Real valued scalar
v - Real valued scalar

x - Positive integer

See Also:
logspace (users)

log

Syntax
y = log(x)

Definition
This function returns the natural logarithm (base e) of the argument *x*. It operates on an part-by-part basis on arrays. Exceptions of "-1.#INF" (negative infinity) and "-1.#IND"(indefinable) are thrown for zero and negative arguments respectively, as is to be expected. For complex valued arguments, the returned y = a + bi is such that a is log(sqrt(real(x)^2+imag(x)^2)), i.e. the natural log of the magnitude and b is atan(imag(x)/real(x)), i.e. the argument assumed to be in natural log.

Examples:

Formula	**Result**
log(1)	0
log([10 , 1.5])	[2.3 , 0.4]
log([2.3 , 0.5 ; 3.7 , 0.8])	0.832909 -0.693147 1.30833 -0.223144

Compatibility
Real and complex-valued scalars, Vectors, Arrays

See Also:
log10 (users)
log2 (users)

log2

Syntax
y = log2(x)
[f, e] = log2(x)

Definition
When used with one output argument, this function returns the base-2 logarithm of the argument. It operates on an part-by-part basis on arrays. Exceptions of "-1.#INF" (negative infinity) and "-1.#IND"(indefinable) are thrown for zero and negative arguments respectively, as is to be expected. For complex valued arguments, the returned y = a + bi is such that a is log2(sqrt(real(x)^2+imag(x)^2)), i.e. the base-2 log of the magnitude and b is atan(imag(x)/real(x))/log(2), i.e. the argument assumed to be in base-2 log.

When used with two output arguments, the mantissa and exponent of the floating point argument are returned into f and e respectively.

Definition
This function returns the natural logarithm (base e) of the argument *x*.
Examples:

Formula	Result
log2(2)	1
log2([4, 512])	[2, 9]

Compatibility

Real and complex-valued scalars, vectors, arrays

See Also

log (users)
log10 (users)

log10

Syntax

y = log10(x)

Definition

This function returns the 10-base logarithm of the argument *x*. It operates on an part-by-part basis on arrays. Exceptions of "-1.#INF" (negative infinity) and "-1.#IND"(indefinable) are thrown for zero and negative arguments respectively, as is to be expected. For complex valued arguments, the returned y = a + bi such that a is log10(sqrt(real(x)^2+imag(x)^2)), i.e. the log10 of the magnitude of the vector and b is atan(imag /imag)/log(10) i.e. the log10 of the argument, where log(10) is the natural logarithm of 10.

Examples:

Formula	Result
log10(1)	0
log10([10 , 1.5])	[1 , 0.176]
log10([2.3 , 0.5 ; 3.7 , 0.8])	[0.362 , -0.301 ; 0.568 , -0.097]
log10(3+2i)	0.556972 + 0.255366i

Compatibility

Real and complex valued scalars, vectors, arrays

See Also

log (users)
log2 (users)

logspace

Syntax

y = logspace(u,v)
y = logspace(u,v,x)
y = logspace(u,pi)

Definition

This function creates a real-valued vector that is spaced logarithmically. It is the logarithmic equivalent of linspace and the colon operator (:), and is useful for generating frequency vectors.

This function generates values that are spaced from 10^u to 10^v. It creates an x number of points, and if x is not specified, it defaults the value to 50.

If pi is specified instead of v then the values are spaced from 10^u to pi (approx. 3.14). This is useful for digital signal processing where frequencies go around the unit circle.

Examples:

Formula	Result
logspace(1,6,6)	[10, 100, 1000, 1e4, 1e5, 1e6]
logspace(-3,3,7)	[0.001, 0.01, 0.1, 1, 10, 100, 1000]
logspace(0,1,10)	[1, 1.29155, 1.6681, 2.15443, 2.78256, 3.59381, 4.64159, 5.99484, 7.74264, 10]
logspace(0,pi,5)	[1, 1.33134, 1.77245, 2.35973, 3.14159]

Compatibility
u - Real valued scalar
v - Real valued scalar
x - Positive integer

See Also
linspace (users)

lookup

Syntax
index = lookup(x,v)

Definition
This function returns the *index* of the real value of *v* in the vector *x* after sorting the vector in ascending order of real values. If real(*v*) does not explicitly exist in real(*x*), then the returned index is 0 if the sought value is less than the minimum value in the vector. Otherwise, the index of part which would serve as the lower bound of the range containing the value, is returned. Accordingly, if real(*v*) is larger than the maximum value in *x*, then the returned *index* is that of the last part of real(*x*).

Examples:

Formula	Result
lookup([0.1,0.2,0.4,0.3], 0.4)	4
lookup([-4; 3; -2; 1; 0], 2.5	4
lookup([-4; 3; -2; 1; 0], -4.5	0

Compatibility
x - Real or complex valued vector
v - value whose position is being sought.

lu

Syntax
[L,U,P] = lu(A)

Definition
Let A be an m x n matrix and k=min(m,n).

[L,U,P] = lu(A) produces matrices L, U, and P such that L·U = P·A, where
L is a lower triangular (when m≤n) or lower trapezoidal (when m>n) m x k matrix with unit parts in the primary diagonal
U is an upper triangular (when m≥n) or upper trapezoidal (when m<n) k x m matrix
P is a permutation m x m matrix

Examples:

```
>> A=randn(3,3)+j*randn(3,3)
  A =
           0.723014 + 1.18447j      0.934672 + 0.460644j      0.441228 + 0.256457j
          -0.328791 - 0.851946j    -0.861837 + 2.87705j       0.955427 + 1.76944j
           0.179696 - 0.856819j     1.68603 - 0.932334j      -0.0821437 - 1.2154j
>> [L,U,P] = lu(A)
  L =
                  1                        0                  0
          -0.647459 - 0.117631j            1                  0
          -0.459545 - 0.43222j     -0.150244 - 0.569137j      1
  U =
           0.723014 + 1.18447j      0.934672 + 0.460644j       0.441228 + 0.256457j
                  0                -0.310862 + 3.28524j        1.21094 + 1.98739j
                  0                        0                  -0.939387 + 0.080946j
  P =
          1     0     0
          0     1     0
          0     0     1
>> max( max( abs( L*U-P*A ) ) )
  ans =
          1.11022e-016
>> A = rand(6,3)
  A =
          0.60099      0.440156     0.864022
          0.127121     0.130142     0.348069
          0.946835     0.559306     0.632182
          0.766416     0.852558     0.260926
          0.857445     0.0636686    0.47777
          0.447486     0.372438     0.510765
>> [L,U,P] = lu(A)
  L =
          1             0             0
          0.905591      1             0
          0.634736     -0.192272      1
          0.809451     -0.902882     -0.756563
          0.134259     -0.124312      0.565567
          0.472613     -0.244116      0.424851
  U =
          0.946835     0.559306      0.632182
          0           -0.442834     -0.0947284
          0             0             0.444539
  P =
          0     0     1     0     0     0
          0     0     0     0     1     0
          1     0     0     0     0     0
          0     0     0     1     0     0
          0     1     0     0     0     0
          0     0     0     0     0     1
>> max( max( abs( L*U-P*A ) ) )
  ans =
          1.11022e-016
```

max

Syntax

y = max(x)
y = max(x,z)
y = max(x,dim)
[y, i] = max(...)

Definition

Returns the maximum part of a vector x. In the case of arrays, the function returns a row vector with the maximum part in each column. When dealing with multidimensional arrays, it treats the parts along the first non-singleton dimension, or the specified dim, as vectors and returns the maximum of each.

y = max(x,z) returns an array with the same dimensions as x and z containing the maximum parts from vectors x or z. The size of x and z have to be the same.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

[y, i] = max(...) also returns the indices of the maximum parts in a vector i. If more than one maximum of the same value exists, then only the first parts index is returned.

Examples:

Formula	**Result**

x = 10
y = max(x)
	y = 10

x = [18 -20 23 54 4 71 -43]
y = max(x)
	y = 71

x = [27 86; complex(600 , -435), 34]
y = max(x)
	y = [600 - j435, 86]

x = [27 86; complex(1 , 1) , -34]
y = max(x)
	y = [27, 86]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
min (users)

mean

Syntax
y = mean(x)
y = mean(x,dim)

Definition
Returns the arithmetic mean of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by dim, or the first non-singleton dimension if dim, is not specified.

Examples:

Formula	**Result**
y = mean([3 ; 4 ; 8 ; 9])	y = 6
y = mean([complex(1 , 2) ; complex(1 , 1) ; complex(2 , 1)])	y = 1.333 + j1.333
y = mean([1,2,3;4,5,6;7,8,9])	y = [4, 5, 6]

Compatibility
Numeric arrays

See Also

median (users)

median

Syntax
y = median(x)
y = median(x,iDim)

Definition
Returns the median of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	Result
y = median([3 ; 4 ; 8 ; 9])	y = 6
y = median([complex(1 , 2) ; complex(1 , 1) ; complex(2 , 1)])	y = 2 + j1
y = median([1,2,3;4,5,6;7,8,9])	y = [4, 5, 6]

Compatibility
Numeric arrays

See Also
mean (users)
mode (users)

min

Syntax
y = min(x)
y = min(x,z)
y = min(x,dim)
[y,i] = min(...)

Definition
Returns the minimum part of a vector x. In the case of complex-valued arrays, the magnitude of each part is used.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by dim, or the first non-singleton dimension if dim is not specified.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

[y,i] = min(...) also returns the indices of the minimum valued parts in x. If there are more than one minimum parts of the same value, the index of the first one found is returned.

Examples:

Formula	Result

x = [10]
y = min(x)
	y = 10

x = [18 , -20 , 23 , 54 , 4 , 71 , -43]
y = min(x)
	y = -43

x = [27, 86 ; complex(600 , -435) , 34]
y = min(x)
	y = [27, 34]

x = [27, 86 ; complex(1 , 1) , -34]
y = min(x)
	y = [1+j1, -34]

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
max (users)

mkpp

Syntax
y = mkpp(x, coefs [, b])

Definition
builds a piecewise polynomial y (a structure with six fields) from its breaks x, and coefficients cofes. breaks is a vector of length a+1 with strictly increasing parts which represent the start and end of each of a intervals. coefs is an a-by-k matrix with each row coefs(i,:) containing the coefficients of the terms, from highest to lowest exponent, of the order k polynomial on the interval [breaks(i),breaks(i+1)]. the optional parameter l gives the value of each of its coefficients is a vector of length b.

Examples:

Formula	Result

b=[4 5]
c=[3 5 6]
pp=mkpp(b,c)
pp2=mkpp(b,c,3)
	>> pp
	? ans =
	? form: [pp]
	? breaks: [1x2 double]
	? coefs: [1x3 double]
	? pieces: [1]
	? order: [3]
	? dim: [1]
	>> pp2
	? ans =
	? form: [pp]
	? breaks: [1x2 double]
	? coefs: [3x1 double]
	? pieces: [1]
	? order: [1]
	? dim: [3]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
spline (users)
ppval (users)
unmkpp (users)

mod

Syntax

m = mod(a,b)

Definition
This function applies the modulus operation on *a* by *b*.
It returns, m = a - (floor(a./b) .* b). If *b* is a scalar, then all parts of *a* are treated by its value. If *b* is nor

Examples:

Formula	Result
m = mod(13, 5)	m = 3
m = mod([1:5],3)	m = [1,2,0,1,2]

Compatibility
Real valued scalars, vectors, arrays

See Also
rem (users)

mode

Syntax
y = mode(x)
y = mode(x,iDim)
[y, n] = mode(x, ...)
[y,n,ca] = mode(x, ...)

Definition
Returns the mode of a vector x. If there are several values with equal maximum number of occurrences, the smallest value is returned.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

[y, n] = mode(x, ...) also returns an array of same size as y which contains the number of occurrences of each part in y.

[y,n,ca] = mode(x, ...) also returns a cell array with the same size as y and n, and it contains, in each part, a sorted vector of the values that have the same frequency as each part in y.

Examples:

Formula	Result
y = mode ([8 ; 4 ; 8 ; 9])	y = 8
y = mode ([complex(1 , 2) ; complex(1 , 2) ; complex(2 , 1)])	y = 1 + j2
y = mode ([1,2,3;2,2,3;7,8,9])	y = [1, 2, 3]

Compatibility
Numeric arrays

See Also
mean (users)
median (users)

moment

Syntax
y = moment(x,order)
y = moment(x,order,iDim)

Definition
Returns the central moment of order *order* of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = moment([1 ; 2 ; 3 ; 4])	y = 1.25
y = moment([complex(1 , 2) ; complex(2 , 3)], 2)	y = 0 + j0.5

Compatibility
Numeric arrays

NaN

Syntax
array = NaN(n, distdim)
array = NaN(m, n, distdim)
array = NaN(..., classname, distdim)

Definition
This function creates an n-by-n, or m-by-n as specified, distributed array, which is of class double by default. The distributed dimension dim and partition PAR are specified by distdim in the parameters, but if not then it automates to the second dimension and defaultPartition(n) is used.

array = NaN(..., classname, distdim) also allows you to specify the class of the array. These can either be 'double' or 'single.'

Examples:

Formula	**Result**

ndims

Syntax
y = ndims(x)

Definition
This function returns the number of dimensions of *x*. Note that scalars are treated as 1x1 arrays, so the function returns a dimesnion count of 2.

Examples:

Formula	Result
ndims(5)	2
ndims([1, 2])	2

```
r = [1 2 3 4 2;1 2 3 4 2;2 2 3 4 3;1 2 3 4 2]
rdim = ndims(r)
% rdim = 2
m = reshape(r,[2,2,5\])
mdim = ndims(m)
% mdim = 3
```

Compatibility
scalars, vectors, arrays

See Also:
permute (users)
shiftdim (users)

false

Syntax
false

Definition
false as a boolean value.

Examples:

Formula	Result
x=false	false

See Also
true (users)

nextpow2

Syntax
y = nextpow2(x)

Definition
This function returns the power of two that produces the number immediately higher than the absolute value of the supplied scalar number *x*. If *x* is a one-dimensional vector, then *y* is the power of two that would cover length(x). Multi-dimensional arrays are not supported by the function.

Examples:

Formula	Result	Comment
nextpow2(17)	5	2^5 = 32 > 17
nextpow2([3,4,5,6,7])	3	2^3 = 8 > 5 = length([3,4,5,6,7])
nextpow2([17;33;15])	2	2^2 = 4 > 3 = length([17;33;15])

Compatibility
Real-valued numbers and vectors

num2str

Syntax
ystring = num2str(x)

Definition
This function can convert a real-valued scalar, vector or array to a string representation. Only real portions of complex valued numbers will be entered to the string. Arrays are traversed along the innermost (column) dimension. Commas, semicolons, brackets and other non-whitespace delimiters are ignored when drafting the string.

Examples:

Formula	**Result**
num2str(500)	'500'
num2str([500,200;100,400])	'500 100 200 400'
num2str(500+200i)	'500'

Compatibility
Real valued scalars, vectors, arrays

See Also
str2num (users)

numel

Syntax
y = numel(x)

Definition
This function returns the total number of parts in the array *x*.

Examples:

Formula	**Result**
numel(2)	1
numel([1 2 3])	3
numel(diag([1 1]))	4

Compatibility
scalars, vectors, arrays

See Also
ndims (users)

ones

Syntax
y = ones(m)
y = ones(m, n)
y = ones(m, n, p, ...)
y = ones([m,n,p,...])
y = ones(m, n, p, ..., class)
y = ones([m,n,p,...], class)

Definition
This function returns a m by n by ... array with every part equal to 1. If only one argument is specified and it is a scalar m, then an m x m matrix is returned. A vector of dimensions may also be passed in. The optional class argument is a string that specifies the data type of the array to return.

Examples:

Formula	**Result**
y = ones(3 , 2)	y = [1 , 1 ; 1 , 1 ; 1 , 1]
y = ones(2)	y = [1 , 1 ; 1 , 1]
y = ones([5 1])	y = [1 ; 1 ; 1 ; 1 ; 1]

See Also

zeros (users)

pchip

Syntax

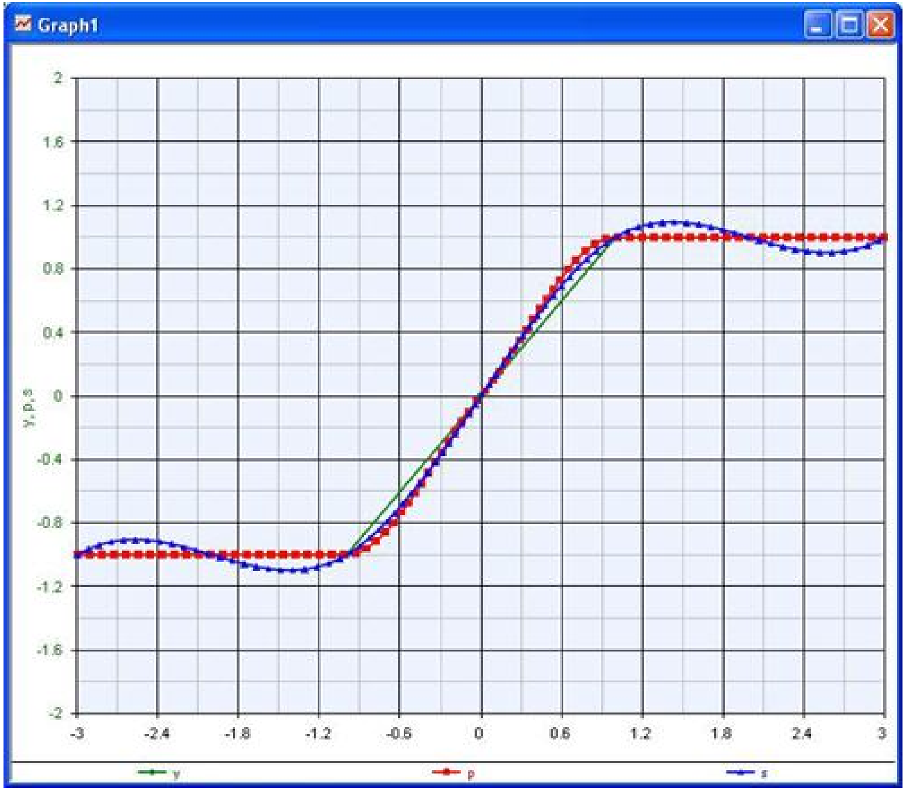
y2 = pchip(x,y,x2)
pp = pchip(x,y)

Definition

By using cubic interpolation with x and corresponding y, y2 = pchip(x,y,x2) returns y2 with is corresponding with x2.
pp = pchip(x,y) returns a polynomial structure pp. both x and y can be row or column vector.

Examples:

```
x = -3:3;
y = [-1 -1 -1 0 1 1 1];
t = -3:.01:3;
p = pchip(x,y,t);
s = spline(x,y,t);
```



Compatibility

Numeric scalars, Vectors, Arrays

See Also

spline (users)
ppval (users)
interp1 (users)

permute

Syntax

y = permute(x,n)

Definition

This function rearranges the dimensions of the array *x*, using an order specified by the vector *n*.

Examples:

```
% x is a 2x3 matrix
x = [1, 2, 3; 4, 5, 6];
y = permute(x,[2,1])
% y is a 3x2 matrix
% y = [1, 4; 2, 5; 3, 6]
%
% z is a 3x2x4 matrix
z=zeros(3,2,4);
z(1,:,:) = [ 1, 2, 3; 4, 5, 6];
z(2,:,:) = 0.1*z(1,:,:);
z(3,:,:) = 10*z(1,:,:);
% Create a permuted version of z
w = permute(z,[2,3,1]);
% w(1,:,:) = [1, 0.1, 10; 2, 0.2, 20; 3, 0.3, 30; 4, 0.4, 40];
% w(2,:,:) = [5, 0.5, 50; 6, 0.6, 60; 7, 0.7, 70; 8, 0.8, 80];
% Note that w is a 2x4x3 matrix since according to the vector [2,...]
% w's 3rd (outermost) dimension is the same as z's 2nd (middle) dimension,
% w's 2nd (middle) dimension is the same as z's 3rd (outermost) dimension
```

See Also

ipermute (users)
shiftdim (users)

poly

Syntax

p = poly(matx)
p = poly(vec)

Definition

p = poly(matx) returns a row vector containing the coefficients of the characteristic polynomial, det(sl-a), ordered in descending powers.

p = poly(vec) returns a row vector containing the coefficients of the polynomial which has roots that are the parts of vec.

Examples:

```
X = [9 2 3; 1 5 6; 7 8 0]
p = poly(X)                  % returns the characteristic equation of matrix X in a row vector p.
r = roots(p)                 % The roots of this polynomial (eigenvalues of matrix X) are returned
```

```
in a column vector r
```

Result

```
>> p
? ans =
?   1  -6  -72  -27
>> r
? ans =
?   12.1229
?   -5.73451
?   -0.388384
```

Compatibility
Vectors, Matrices

See Also
conv (users)
polyval (users)
roots (users)

polyval

Syntax
y = polyval(v,x)

Definition
y = polyval(v,x) returns the value of a polynomial of degree n evaluated at x. The input argument v is a vector of length n+1 whose parts are the coefficients in descending powers of the polynomial to be evaluated.

Examples:

```
v=[2 4 7]                  % same as v = 2x^2+4x+7
y=polyval(v,[0, 1, 2])     % v is evaluated at x=0,1,2
```

result:

```
>> y
? ans =
?   7  13  23
```

Compatibility
Vectors

See Also
polyvalm (users)

polyvalm

Syntax
Y = polyvalm(v,X)

Definition
Y = polyvalm(v,X) evaluates matrix X in the polynomial v.
Polynomial v is a vector whose parts are the coefficients of a polynomial in descending powers, and X must be a square matrix.

Examples:

```
X=[4 5; 1 2]
v=[3 4 1]                % as same as v = 3x^2+4x+1
Y=polyvalm(v,X)          % evaluates v = 3x^2+4x+1 with x = X
```

Result:

```
>> Y
? ans =
?   80  110
?   22  36
```

Compatibility

Matrices

See Also

polyval (users)

true

Syntax

true

Definition

The value true (logical 1).

Examples:

Formula	Result
x = true	x is a true (logical 1)

See Also

false (users)

ppval

Syntax

v = ppval(pp,xx)

Definition

v = ppval(pp,xx) returns the value of the piecewise polynomial f, contained in pp, at the entries of xx. You can construct pp using the functions interp1, pchip, spline, or the spline utility mkpp.

v is obtained by replacing each entry of xx by the value of f there. If f is scalar-valued, v is of the same size as xx.

Examples:

```
% Compare the results of integrating the function cos
a = 0; b = 10;
int1 = quad(@cos,a,b)
int1 =
  -0.5440
% with the results of integrating the piecewise polynomial pp that approximates the cosine
function by interpolating the
```

```
% computed values x and y.
x = a:b;
y = cos(x);
pp = spline(x,y);
int2 = quad(@(x)ppval(pp,x),a,b)
int2 =
  -0.5485
% int1 provides the integral of the cosine function over the interval [a,b], while int2 provides
the integral over the same   % interval of the piecewise polynomial pp.
```

Compatibility

Vectors

See Also

spline (users)
mkpp (users)
unmkpp (users)

prctile

Syntax

y = prctile(x,p)
y = prctile(x,p,iDim)

Definition

Returns the p'th percentiles of a vector x (p can be a scalar or a vector of percent values). Percentiles must be between 0 and 100. For an N-part vector, this function computes percentiles by assigning percentile values to the sorted input data as 100*(0.5/N), 100*(1.5/N), ..., 100*((N-0.5)/N). Linear interpolation is then used to compute percentiles between these values. The minimum or maximum values in the data are returned for percentile values outside that range.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
prctile([1, 2, 3, 4, 5], 50)	3
prctile([10, 20, 50, 100, 200], 60)	75

Compatibility

Vectors, Arrays

See Also

quantile (users)
median (users)

prod

Syntax

y = prod(x)
y = prod(x,dim)

Definition

Returns the product of parts of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by dim, or the first non-singleton dimension if dim is not pecified.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**
y = prod([2 , 3 , 5])	y = 30
y = prod([1 , 3 ; 7 , 5])	y = [7, 15]
y = prod([-3.3 , 0.7 , 5 , 3])	y = -34.65

a = complex(1 , 3)
b = complex(4 , 1)
c = complex(1 , 0)
y = prod([a , b , c])
	y = 1 + 13j

Compatibility
Numeric Scalars, Vectors, Arrays

See Also
sum (users)

quantile

Syntax
y = quantile(x,q)
y = quantile(x,q,iDim)

Definition
Returns the q'th quantiles of a vector x (q can be a scalar or a vector of quantile values). Quantiles must be between 0 and 1. For an N-part vector, this function computes quantiles by assigning quantile values to the sorted input data as (0.5/N), (1.5/N), ..., ((N-0.5)/N). Linear interpolation is then used to compute quantiles between these values. The minimum or maximum values in the data are returned for quantile values outside that range.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
quantile([1, 2, 3, 4, 5], 0.5)	3
quantile([10, 20, 50, 100, 200], 0.6)	75

Compatibility
Vectors, Arrays

See Also
median (users)
prctile (users)

rand

Syntax

y = rand(n1)
OR
y = rand(n1,n2)
OR
y = rand([n1,n2,..nN])

Definition

This function returns an array of random numbers with uniform distribution. The size of the array can be specified either as a list of one or two scalars or a vector for higher-dimensions. If a single scalar *n1* is used as the only parameter, a square matrix of size *n1* x *n1* is returned.

Examples:

```
% Create a 5x5 matrix of uniformly distributed random numbers
y = randn(5)
%
% Create a 5-part row vector of uniformly distributed random numbers
y = randn(1,5)
%
% Create a 3x4x2 matrix of uniformly distributed random numbers
y = randn([3,4,5])
%
```

Compatibility

nN - positive integer valued scalar or vector for all N >= 1.

See Also:

randn (users)

randn

Syntax

y = randn(n1)
OR
y = randn(n1,n2)
OR
y = randn([n1,n2,..nN])

Definition

This function returns an array of random numbers with Normal (Gaussian) distribution. The size of the array can be specified either as a list of one or two scalars or a vector for higher-dimensions. If a single scalar *n1* is used as the only parameter, a square matrix of size *n1* x *n1* is returned.

Examples:

```
% Create a 5x5 matrix of normally distributed random numbers
y = randn(5)
%
% Create a 5-part row vector of normally distributed random numbers
y = randn(1,5)
%
% Create a 3x4x2 matrix of normally distributed random numbers
```

```
y = randn([3,4,5])
%
```

Compatibility
nN - positive integer valued scalar or vector for all N >= 1.

See Also:
rand (users)

real

Syntax
y = real(x)

Definition
This function returns the real part of a complex number. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
real(20)	20
real(3+2j)	3
real([-2+4j 5-3j 2+j])	[-2 5 2]

Compatibility
Numeric scalars, vectors, arrays

See Also
imag (users)

rectwin

Syntax
c = rectwin(L)

Definition

This function returns a column vector, c, a rectangular window with a length of L. Each sample of the vector has the nominal window height of 1.

Examples:

Formula	**Result**
rectwin(5)	[1 1 1 1 1]

See Also:
bartlett (users)
blackman (users)
gausswin (users)
hamming (users)
hann (users)

rem

Syntax
r = rem(a,b)

Definition

This function returns the remainder when dividing *a* by *b*. Both parameters are required to be real arrays or real scalars subject to the restriction that if *b* is a vector or array, it must be the same size as *a* for part-by-part division and remainder computation. when *b* is a scalar, all the parts of *a* are divided by it. When *b* is explicitly zero, the result is *NaN*.

Examples:

Formula	**Result**
rem(2, 1.45)	0.55
rem([2,5,6], 1.45]	[0.55, 0.65, 0.20]
rem([2,5,6], [1.45,1.55,1.65]	[0.55, 0.35, 1.05]

Compatibility

Real valued scalars, vectors, arrays

See Also

mod (users)

repmat

Syntax

repm = repmat(orig,dim1)
OR
repm = repmat(orig,dim1,dim2)
OR
repm = repmat(orig,[dim1,dim2,...,dimN])

Definition

This function repeats the input matrix *orig* in a tiled fashion as many
times along as many dimensions as are specified by the following parameters.

Examples:

```
% Define a 2x3 matrix
orig = [1, 2, 3; 4, 5, 6];
% Create a 2x1x2 tiling of this matrix
repm = repmat( orig, [3, 1, 2] )
% repm(:,:,1) = [1, 2, 3; 4, 5, 6; 1, 2, 3; 4, 5, 6; 1, 2, 3; 4, 5, 6];
% repm(:,:,2) = [1, 2, 3; 4, 5, 6; 1, 2, 3; 4, 5, 6; 1, 2, 3; 4, 5, 6];
% Note that the inner-most dimension is repeated twice such that
% repm(1,1,:) = [1, 1];
% The middle dimension is not-repeated such that
% repm(1,:,:) = [1, 1; 2, 2; 3, 3];
% and the outermost is repeated thrice.
```

Compatibility

orig - Must be a numeric scalar, vector or array
dimN - Positive integer >= 1

See Also:

permute (users)
shiftdim (users)

reshape

Syntax

y = reshape(x , i,j)

y = reshape(x , i,j,k, ...)
y = reshape(x , [i,j,k, ...])
y = reshape(x , ...,[],...)

Definition

y = reshape(x , i,j) returns a i-by-j matrix with elements taken column wise from x. The number of elements in the resulting i-by-j matrix *y* must be same as number of elements in the input matrix *x*.
y = reshape(x , i,j,k, ...) and y = reshape(x , [i,j,k, ...]) will return a i-by-j-by-k-by.... matrix with same elements as in input matrix *x*. The number of elements in the resulting i-by-j-by-k-by.... matrix *y* must be same as number of elements in the input matrix *x*.
y = reshape(x , ...,[],...) replaces [] with an integer scalar number representing the number of elements in the corresponding dimension such that the total number of elements in output matrix *y* is same as the number of elements in input matrix *x*. You can have only one instance of [] in argument.

Swept-dimensions are NOT counted. (eg. if S is the variable produced by a 100 point linear analysis of a 2-port circuit, reshape(S, [4;1]) would return a variable containing S, but having dimensions 100x4x1)

Examples:

Formula	**Result**

x = [1 , 2 , 3 ; 4 , 5 ,6]
y = reshape(x , 1, 6)
	y = [1 , 4, 2, 5, 3 , 6]

x = [1 , 2 , 3 ; 4 , 5 ,6]
y = reshape(x , [6, 1])
Or
y = reshape(x , 6, 1)
	y = [1; 4; 2; 5; 3; 6]

x = [1 , 2 , 3 ; 4 , 5 ,6 ; 7, 8, 9; 10, 11, 12]
y = reshape(x , 6, [])
	y = [1, 8; 4, 11; 7, 3; 10, 6; 2, 9; 5, 12]

Compatibility

Real and complex-valued Scalars, Vectors, Arrays

See Also

permute (users)
shiftdim (users)

roots

Syntax

polyroot = roots(polycoef)

Definition

This function returns a column vector, *polyroot*, whose parts are the roots of the polynomial expressed in the form of the coefficient vector *polycoef*.

Examples:

```
% Find the roots of the polynomial:
% y = 1 - 6*x - 72*x^2 - 27*x^3
polycoef = [1,-6,-72,-27];
polyroot = roots(polycoef);
% polyroot = [12.1229;-5.7345;-0.3884]
```

Compatibility

Real or complex valued vector

See Also

poly (users)

rot90

Syntax

y = rot90(x)
y = rot90(x,n)

Definition

This function takes the matrix *x*, and rotates it 90 degrees in the counterclockwise direction. You can also supply the parameter *n* to specify how many times you want to rotate the object 90 degrees.

Examples:

```
X = [1, 2, 3; 4, 5 ,6; 7, 8, 9];
%
Y = rot90(X)
% Y = [3, 6, 9; 2, 5, 8; 1, 4, 7];
%
Z = rot90(X,-1)
% Z = [7, 4, 1; 8, 5, 2; 9, 6, 3];
%
W = rot90(X,2)
% W = [9, 8, 7; 6, 5, 4; 3, 2, 1];
%
```

See Also

fliplr (users)
flipud (users)
flipdim (users)

round

Syntax

y = round(x)

Definition

round rounds the argument to the nearest integer. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
round(2.2)	2
round(2.2 + 3.7j)	2 + 4j
round(-2.3 - 3.9j)	-2 - 4j

Compatibility

Numeric scalars, Vectors, Arrays

See Also

floor (users)
ceil (users)
fix (users)

runanalysis

Syntax

runanalysis('AnalysisName')
runanalysis('AnalysisName', ContinueOnError)

Definition

The runanalysis function is used to force an analysis to run from an equation block. It can be used to control simulations in a sequential manner. The function does not return until the analysis finishes, whether successful or in error.

The second argument, ContinueOnError, is optional and defaults to false. If ContinueOnError is false and an error is encountered when running the analysis, the equation block throws an error and terminates. If ContinueOnError is true, the equation script continues to run.

Examples:

```
SourceAmpls = [1 2 5 10];  % We'll step our source's amplitude with these values
for i = 1 : length(SourceAmpls)
 CurAmplitude = SourceAmpls(i);  % This variable is used by our source's Amplitude parameter
 runanalysis('Analysis1');
 % Post process data from the current analysis run
 % Post-processing equations would go here
end
```

Compatibility

See Also

sec

Syntax

y = sec(x)

Definition

sec returns the secant of a radian-valued argument. This function operates on an part-by-part basis on arrays.

Compatibility

Numeric scalars, Vectors, Arrays

secd

Syntax

y = secd(x)

Definition

secd returns the secant of a degree-valued argument. This function operates on an part-by-part basis on arrays.

Compatibility

Numeric scalars, Vectors, Arrays

sech

Syntax

y = sech(x)

Definition

sech returns the hyperbolic secant the argument. This function operates on an part-by-part basis on arrays.

Compatibility

Numeric scalars, Vectors, Arrays

setindep

Syntax

setindep("dependentvar", "independentvar1", "independentvar2", ...)

Definition

setindep manually sets the independent variable(s) for a swept variable. Both are passed by name. A long name can be used for the independentvar. If independentvar is empty (blank) the dependentvar becomes unswept. All independents should have the same length, equal to the number of rows in the dependent.

Examples:

Formula	Result

ind = [0.025;1;2;5]
setindep("x" ,"ind")
	set x to have a 4 part independent vector. x should be of size 4xm or 4xmxn

abest = myS[2,1]
setindep("abest", "myData.F")
	set abest to use MyData.F as an independent vector. F must have the same number of parts as abest has rows.

Compatibility

Vectors and Arrays. The independent var must be numeric.

See Also

getunits

setunits

Syntax

setunits('varname', unit)

Definition

setunits sets a variable named varname to have units specified by the parameter unit. unit may be an integer or a string.

> ℹ setunits is used only to set the units of variables in equations and datasets. It will not change units of a part's parameters.

Examples:

Formula	Result

y = [0.025]
setunits('x' , 6006)
	sets units of y to um
	y = 25000

y = 5
setunit('y' , 'mm')
	sets units of y to mm
	y =5000

y = 0.0001
setunits('y', 'uF')
	sets units of y to uF
	y = 100

Compatibility

Numeric Scalars, Strings

See Also

getunits

setvariable

Syntax

setvariable(Dataset, Variable, value)

Definition

setvariable sets a variable value in a dataset

Examples:

Formula	Result
setvariable('OutData', 'OutVar', 3)	set the variable named Outvar in the dataset OutData to the value 3
setvariable('Out', 'Var', [1 2 3])	set the variable named Var in the dataset Out to a vector [1 2 3]

Compatibility

Dataset and Variable are strings. value is any valid value.

See Also

getvariable (users)

shiftdim

Syntax

y = shiftdim(x,n)
[y,n]=shiftdim(x,n)

Definition

This function can shift the dimensions of the matrix *x* by the specified dimension number *n* .

When *n* is positive, the dimensions are shifted to the left and wrapped around to the right. Thus, a 3x2x4 sized matrix will have its parts restructured into a 2x4x3 sized matrix.

When *n* is negative, the new matrix *y* has as many singleton dimensions to the left and the basic structure of *x* is otherwise left intact. Thus, a 3x2x4 sized matrix will be restructured into a 1x1x3x2x4 matrix with a *n* = -2;

Examples:

```
k=1;
a=zeros(3,2,4);
for x=1:3
for y=1:2
for z=1:4
   a(x,y,z) = k;
   k = k+1;
end
end
end
% a is a 3-dimensional matrix defined in the form of the following three 2x4 2-dimensional
matrices.
% a(1,:,:) = [ 1, 2, 3, 4;  5, 6, 7, 8];
% a(2,:,:) = [ 9,10,11,12; 13,14,15,16];
% a(3,:,:) = [17,18,19,20; 21,22,23,24];
%
```

```
% Now shift dimension of a by 1 to the left so that the resulting matrix is 2x4x3
b=shiftdim(a,1);
% b is a 3-dimensional matrix defined in the form of the following 4x3 2-dimensional matrices.
% b(1,:,:) = [1, 9,17;  2,10,18;  3,11,19;  4,12,20];
% b(2,:,:) = [5,13,21;  6,14,22;  7,15,23;  8,16,24];
%
% Now shift dimension of a by -2 to the right so that the resulting matrix is 1x1x3x2x4
c=shiftdim(a,-2);
% c is a 5-dimensional matrix now
% c(1,1,1,:,:) = [ 1, 2, 3, 4;  5, 6, 7, 8];
% c(1,1,2,:,:) = [ 9,10,11,12; 13,14,15,16];
% c(1,1,3,:,:) = [17,18,19,20; 21,22,23,24];
```

See Also

permute (users)

sign

Syntax

y = sign(x)

Definition

sign returns the signum of the argument. The signum function returns -1 if the argument is negative, 1 if the argument is positive, and 0 if the argument is 0. This function operates on an part-by-part basis on arrays.

Compatibility

Numeric scalars, Vectors, Arrays

sin

Syntax

y = sin(x)

Definition

sin returns the sine of the radian-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
sin(0)	0
sin(pi/2)	1
sin(-pi/2)	-1
sin([pi/4 2*pi/3])	[0.707 0.866]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

asin (users)

sind (users)

sinc

Syntax

y = sinc(x)

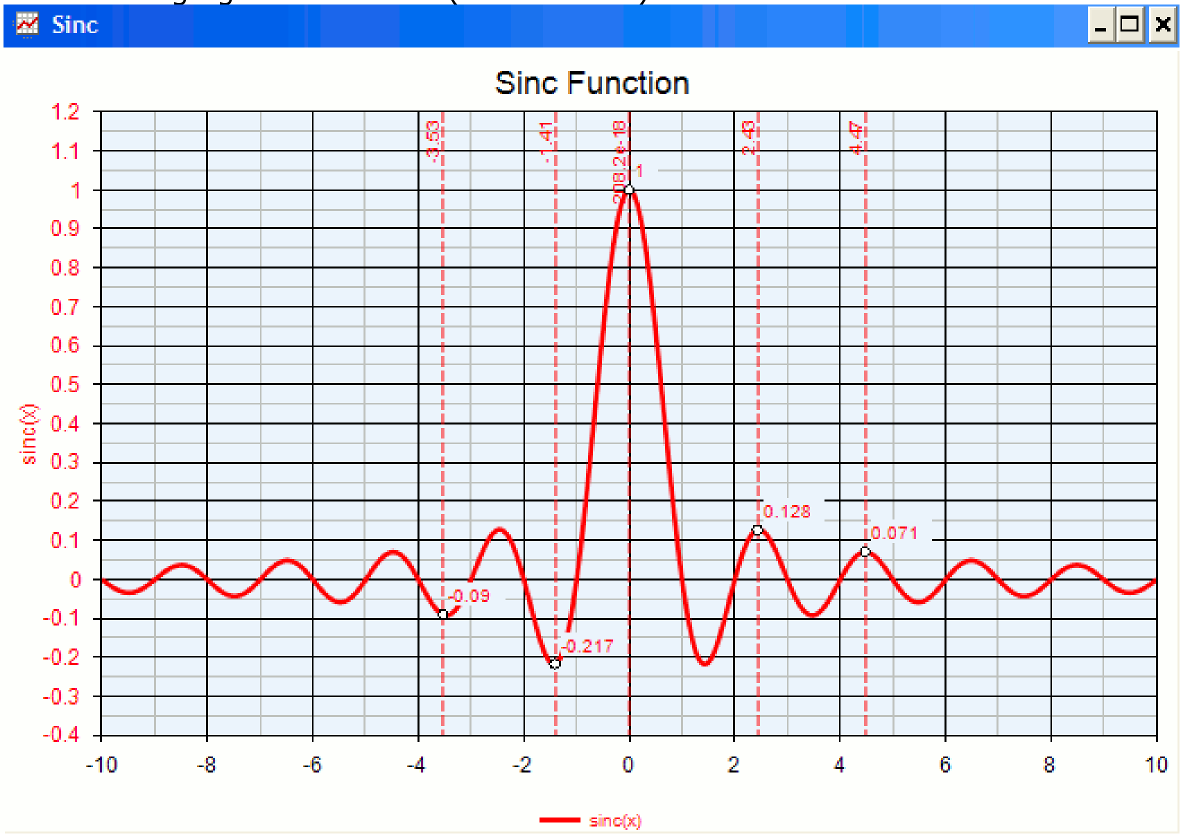
Definition

sinc returns the sinc function of the argument. The sinc function is defined as sin(pi*x)/(pi*x) or 1 if x is equal to 0. This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
sinc(0)	1
sinc(pi/2)	0.198
sinc(pi/4)	0.253
sinc(2*pi/3)	0.044

The following figure shows sinc(-10:0.01:10).



Compatibility
Numeric scalars, Vectors, Arrays

See Also
sin (users)

sind

Syntax
y = sind(x)

Definition
sind returns the sine of the degree-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
sind(0)	0
sind(90)	1
sind(-90)	-1
sind([45 60])	[0.707 0.866]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
asin (users)
sin (users)

sinh

Syntax
y = sinh(x)

Definition
sinh returns the hyperbolic sine of the number, or (exp(x) - exp(-x)) / 2. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
sinh(1)	1.175
sinh(5)	74.203
sinh([pi/3 0])	[1.249 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
asinh (users)

size

Syntax
y = size(x)

Definition
size returns a vector containing the number of parts in each dimension of x. part one of y corresponds to the number of parts in the first dimension, part two to the second dimension, and so on.

Examples:

Formula	Result
size([1 2 3 4])	[1 4]
size([1 2 3; 4 5 6])	[2 3]
size(ones(4,3,2))	[4 3 2]

Compatibility
Numeric Scalars, Vectors, Arrays

skewness

Syntax

y = skewness(x)
y = skewness(x,Flag)
y = skewness(x,Flag,iDim)

Definition
Returns the sample skewness of a vector x. Skewness is the third central moment of X divided by the cube of the standard deviation.

If Flag is 0 (default), skewness normalizes by N-1 where N is the sample size. If Flag is 1, skewness normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = skewness([3 ; 4 ; 8 ; 9])	y = 0
y = skewness([1, 2, -5], 1)	y = -0.652

Compatibility
Numeric arrays

See Also
kurtosis (users)
std (users)
var (users)

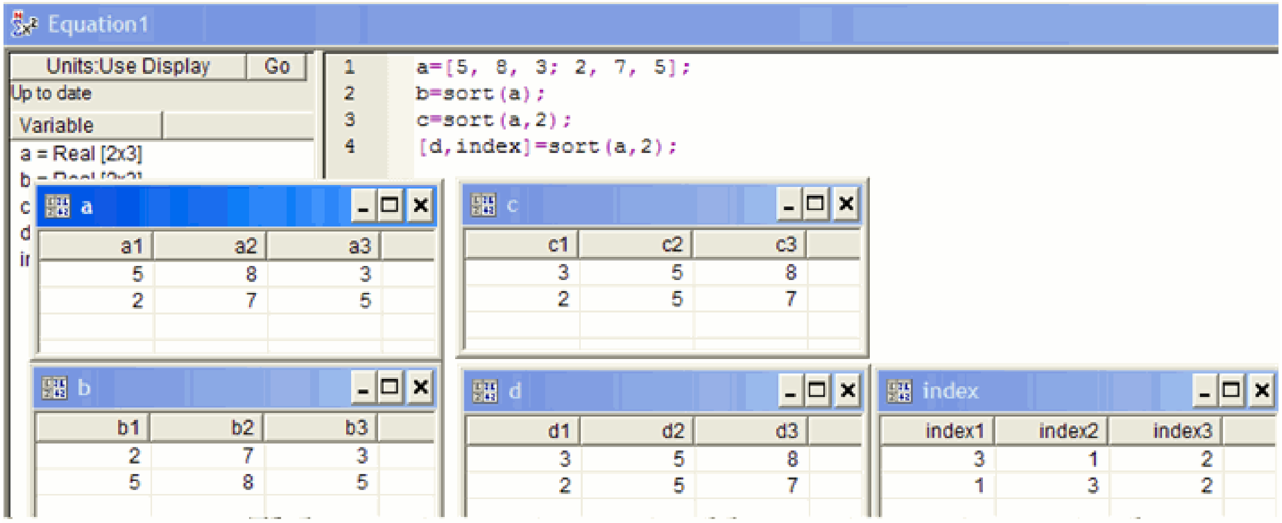
sort

Syntax
y = sort(x)
y = sort(x,dim)
[y,index]=sort(x)
[y,index]=sort(x,dim)

Definition
This function sorts contents of the array x in ascending order along one specific dimension of the array. When unspecified, the innermost non-singleton dimension is chosen. The function can be required to additionally specify the original indices in the sorted order.

Examples:

In the following example note that **b** is the column-wise (default dim is 1 for a 2x3 matrix) sorted whereas **c** and **d** are sorted row-wise. The **index** matrix associated with **d** is interpreted as follows: if the value *k* appears at a specific location along row *i* column *j*, it means that the number now placed (row *i*,column *j*) was originally the number at (row *i*, column *k*).



a = Real [2x3]

```
a=[5, 8, 3; 2, 7, 5];
b=sort(a);
c=sort(a,2);
[d,index]=sort(a,2);
```

a

a1	a2	a3
5	8	3
2	7	5

c

c1	c2	c3
3	5	8
2	5	7

b

b1	b2	b3
2	7	3
5	8	5

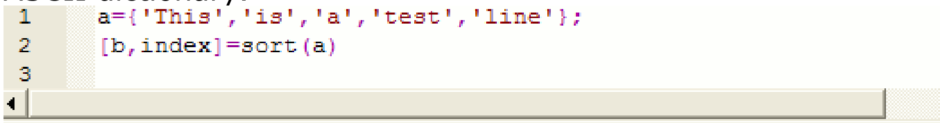
d

d1	d2	d3
3	5	8
2	5	7

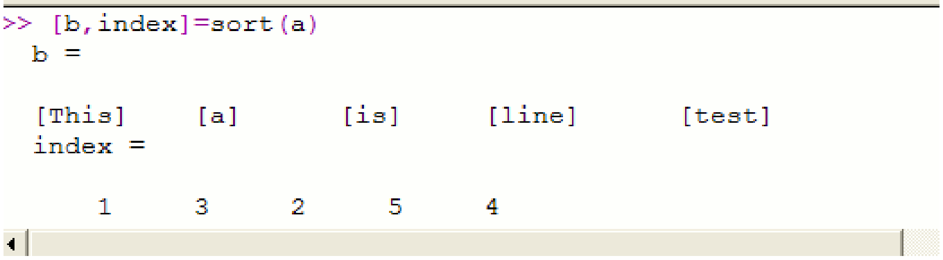
index

index1	index2	index3
3	1	2
1	3	2

Strings can be sorted alphabetically according to ASCII dictionary if the collection is presented as cells as shown in the following example. Note that here the string "This" is retained as the first part because large-cap letters occur before small-cap letters in the ASCII dictionary.



```
a={'This','is','a','test','line'};
[b,index]=sort(a)
```



```
>> [b,index]=sort(a)
  b =

   [This]     [a]        [is]       [line]        [test]
  index =

       1       3       2       5       4
```

Compatibility:
Real-valued numeric vectors and arrays or strings

spline

Syntax

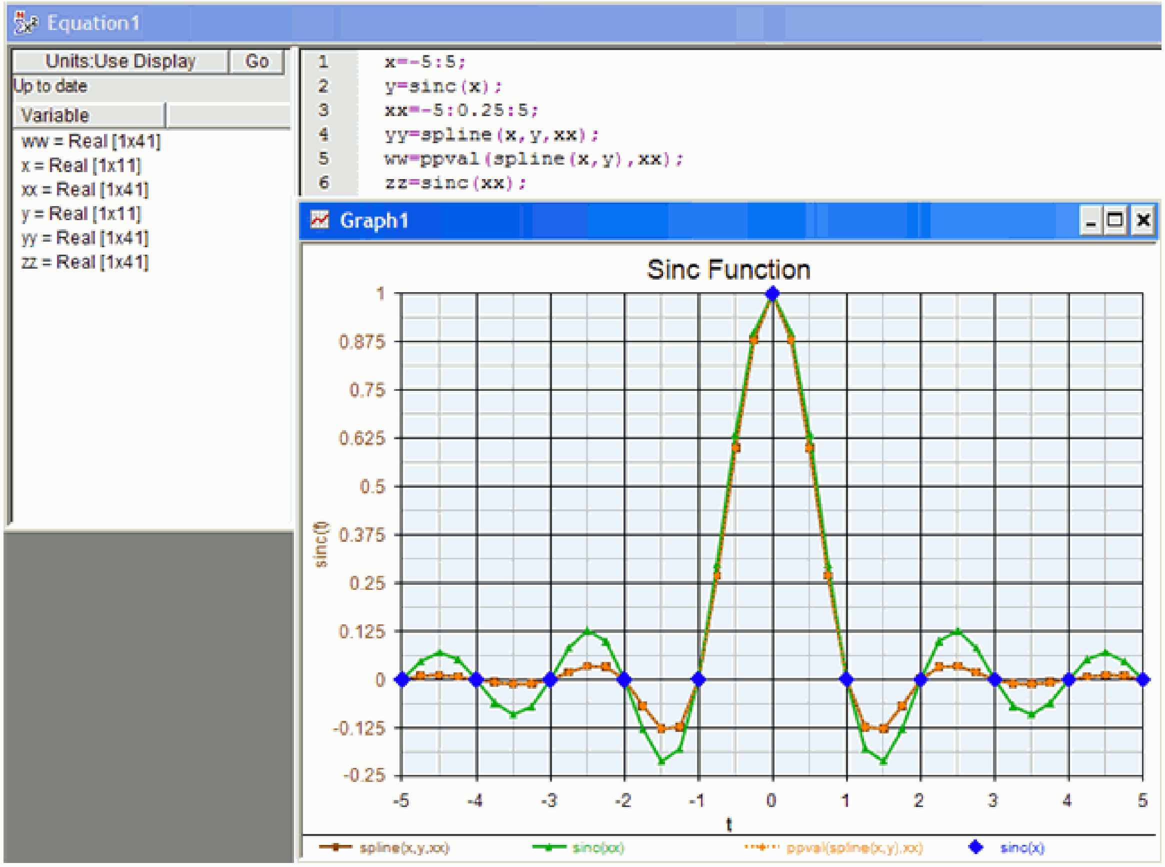
polynomial = spline(originalIndep,originalDep)
OR
fittedDependent = spline(originalIndep,originalDep,fittedIndep)

Definition

This function performs spline polynomial extraction from a one-dimensional function defined as the mapping of an original independent vector onto an original dependent vector. If supplied with a third argument explicitly specifying the independent vector to which fitting is required, the function returns the fitted dependent vector. If the third parameter is not supplied then a structure describing the piece-wise polynomial function is returned, which may then be used in a call to the ppval(polynomial,fittedIndep) function to generate the fittedDependent variable.

Examples:
In the following example, the original mapping of x and sinc(x) are shown in sparsely spaced blue dots, one dot per unit along the independent axis. When four times as much granularity is required, an extended fitting vector xx is introduced. Spline curves produced using this extended independent vector are compared against the true sinc() function of

the extended vector. Note how there is substantial match when some variation is present in the original data, e.g. just one non-zero data point in the original dependent vector. In regions where there is absolutely no off-axis data in the dependent vector i.e. in the side-lobes, the spline() function is still able to partially recover the existence of the side lobes, if not the full amplitude of each.



ww = Real [1x41]
x = Real [1x11]
xx = Real [1x41]
y = Real [1x11]
yy = Real [1x41]
zz = Real [1x41]

```
x=-5:5;
y=sinc(x);
xx=-5:0.25:5;
yy=spline(x,y,xx);
ww=ppval(spline(x,y),xx);
zz=sinc(xx);
```

Compatibility

Real-valued 1-dimensional vector: originalIndep, fittedIndep

Real or complex-valued array: originalDep

sqrt

Syntax

y = sqrt(x)

Definition

This function returns the square-root of the argument.

This function operates on an part-by-part basis on arrays.

Examples:

Formula	**Result**
sqrt(0)	0
sqrt(4)	2
sqrt(2+3i)	1.67415+j0.895977
sqrt(-1)	j
sqrt([9 16 -4])	[3 4 -2j]

Compatibility

Real and complex-valued scalars, vectors, arrays

sscanf

Syntax:
A = sscanf(string, format)
A = sscanf(string, format, size)
[A, count, msg, next] = sscanf(...)

Definition:
Used to read formatted input from a string. Converts the input string using format argument (format) and puts the results into a matrix (A).

size (optional) argument is used to determine how much data is read. Valid values are:

n	read at most n fields from the string
inf	read all of the input string
[m,n]	read at most m*n fields. Fill a matrix with at most m rows.

count (optional) result is the number of matching fields.
msg (optional) is for an error message
next (optional) is one more than the number of characters match in the input string

Format:

- **Whitespace characters** (space, tab or new lines) are used to delimit fields. There are not included in the output.
- **Non-whitespace characters** that are not a part of a format specifier are matched with the next character in string and then discarded. If the character does not match sscanf stops process string.
- **Format specifers**: %[*][width][modifiers]conversionChar, where:

*	(optional) match the data in string but do not put the corresponding match in the output matrix. The format must match but it isn't included in the output.
width	(optional) maximum number of characters to match in string
modifiers	(optional) For compatibility only. valid values (h, l, L)
conversionChar	see table below

Conversion Characters:

Type	**Qualifying Input**
c	Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.
d	Decimal integer: Number optionally preceeded with a + or - sign.
e,E,f,g,G	Floating point: Decimal number containing a decimal point, optionally preceeded by a + or - sign and optionally folowed by the e or E character and a decimal number.
o	Octal integer.
s	String of characters. This will read subsequent characters until a whitespace is found (whitespace characters are considered to be blank, newline and tab).
u	Unsigned decimal integer.
x,X	Hexadecimal integer.

std

Syntax
y = std(x)

y = std(x, Flag)
y = std(x, Flag, iDim)

Definition
Returns the standard deviation of a vector x.

If Flag is 0 (default), std normalizes by N-1 where N is the sample size. If Flag is 1, std normalizes by N.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = std([3 ; 4 ; 8 ; 9])	y = 2.9439
y = std([1, 2, 3], 1)	y = 0.8165

Compatibility
Numeric arrays

See Also
kurtosis (users)
var (users)
skewness (users)

str2num

Syntax
y = str2num('xstring')

Definition
This function can convert a single real-valued number from string format to numeric format.
When supplied with a string containing preceeding non-numeric characters, other than whitespace or tab, the function returns zero.

Examples:

Formula	**Result**
str2num('500')	500
str2num(' 500')	500

Compatibility
String

See Also
num2str (users)

strcmp

Syntax
out = strcmp(str1, str2)
out = strcmp(str, ca)
out = strcmp(ca1, ca2)

Definition
out = strcmp(str1, str2) compares two strings, str1 and str2, and returns true (logical 1) if they are identical. If not, then it returns false (logical 0).

out = strcmp(str, ca) compares str with each string in a cell array. It then returns a logical array, out, that contains the corresponding logical values on whether the two strings are identical.

out = strcmp(ca1, ca2) compares each part in ca1 to the corresponding part in ca2. It then returns a character array that is the same size as ca1 and ca2 with the corresponding logical value on whether the two strings are identical.

This function does not ignore case. To ignore case, use the strcmpi function.

Examples:

Formula	**Result**
out = strcmp('One', 'Two')	out = 0
out = strcmp('Yes', {'No', 'Yes'})	out = [0, 1]

Compatibility
string array, cell array

See Also
strcmpi (users)

strcmpi

Syntax
out = strcmpi(str1, str2)
out = strcmpi(str, ca)
out = strcmpi(ca1, ca2)

Definition
out = strcmpi(str1, str2) compares two strings, str1 and str2, and returns true (logical 1) if they are identical. If not, then it returns false (logical 0).

out = strcmpi(str, ca) compares str with each string in a cell array. It then returns a logical array, out, that contains the corresponding logical values on whether the two strings are identical.

out = strcmpi(ca1, ca2) compares each part in ca1 to the corresponding part in ca2. It then returns a character array that is the same size as ca1 and ca2 with the corresponding logical value on whether the two strings are identical.

This function ignores case. To take the case into account, use the strcmp function.

Examples:

Formula	Result
out = strcmpi('One', 'Two')	out = 0
out = strcmpi('Yes', {'No', 'YES'})	out = [0, 1]

Compatibility
string array, cell array

See Also
strcmp (users)

strncmp

Syntax
out = strncmp(str1, str2, n)
out = strncmp(str, ca, n)
out = strncmp(ca1, ca2, n)

Definition
This function compares the first n characters in str1 and str2 and if they are identical, it returns true (logical 1). Otherwise, it returns false (logical 0).

The function can also compare a string and each part in a cell array, or the parts in two cell arrays.

This function is case sensitive. To ignore case, use the strncmpi function.

Examples:

Formula	Result
out = strncmp('example', 'exam', 4)	out = 1;
out = strncmp('test', {'exam', 'testing'}, 4)	out = [0,1];

Compatibility
string array, cell array

See Also
strncmpi (users)

strncmpi

Syntax
out = strncmpi(str1, str2, n)
out = strncmpi(str, ca, n)
out = strncmpi(ca1, ca2, n)

Definition
This function compares the first n characters in str1 and str2 and if they are identical, it returns true (logical 1). Otherwise, it returns false (logical 0).

The function can also compare a string and each part in a cell array, or the parts in two cell arrays.

This function is not case sensitive. To take case into account, use the strncmp function.

Examples:

Formula	Result
out = strncmpi('example', 'EXAM', 4)	out = 1;
out = strncmpi('test', {'exam', 'TeStING'}, 4)	out = [0,1];

Compatibility
string array, cell array

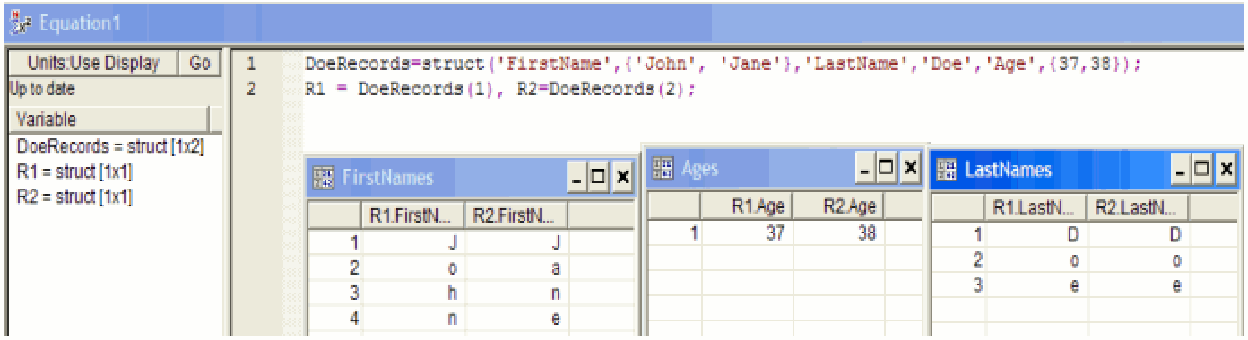
See Also
strncmp (users)

struct

Syntax
y = struct(field1,value1,filed2,value2,....,fieldN,valueN)

Definition
This function creates a structure parts of which can be of various types ranging from strings through complex cell arrays. Each field is assigned the type of the value which succeeds it. If the structure contains more than one cell array, like a matrix, all such cell arrays must be of the same size. Note that fields are always specified as strings.

Examples:
In the figure below, observe how records of two people who share the same last name can be saved to and retrieved from a single structure.



```
DoeRecords=struct('FirstName',{'John', 'Jane'},'LastName','Doe','Age',{37,38});
R1 = DoeRecords(1), R2=DoeRecords(2);
```

DoeRecords = struct [1x2]
R1 = struct [1x1]
R2 = struct [1x1]

FirstNames

	R1.FirstN...	R2.FirstN...
1	J	J
2	o	a
3	h	n
4	n	e

Ages

	R1.Age	R2.Age
1	37	38

LastNames

	R1.LastN...	R2.LastN...
1	D	D
2	o	o
3	e	e

Compatibility
Numeric scalars, Vectors, Arrays

sum

Syntax
y = sum(x)
y = sum(x,dim)

Definition
Returns the sum of parts of a vector x.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by dim, or the first non-singleton dimension if dim is not specified.

The dim argument is optional and specifies which dimension to operate along. For example, if dim is 1, this function operates on each column of the argument. If the argument is omitted, the first non-singleton dimension is chosen as the dimension to operate along.

Examples:

Formula	**Result**
y = sum([10 , 3 , 5])	y = 18
y = sum([2 ; 9 ; 11])	y = 22
y = sum([complex(3 , 3) , complex(5 , 2)])	y = 8 + j5
y = sum([3 , 2 , 19 ; 5 , 7 , 1.5])	y = [8, 9, 20.5]
y = sum([3 , 2 , 19 ; 5 , 7 , 1.5], 2)	y = [24; 13.5]

Compatibility

Numeric scalars, Vectors, Arrays

See Also

prod (users)

svd

Syntax

s = svd(X)
[U,S,V] = svd(X)
[U,S,V] = svd(X, 0)
[U,S,V] = svd(X, 'econ')

Definition

Let X be an m x n matrix and k = min(m,n).

S = svd(X) returns, in the vector S, the singular values (in decreasing order) of the matrix X. S is a column vector of size k.

[U,S,V] = svd(X) produces matrices U, S, and V that form the singular value decomposition of X, that is, X = U·S·V', where
U is a unitary m x m matrix
S is a diagonal m x n matrix whose primary diagonal parts are the singular values (in decreasing order) of X
V is a unitary n x n matrix

[U,S,V] = svd(X, 0) OR [U,S,V] = svd(X, 'econ') produce matrices U, S, and V that form the 'economical' singular value decomposition of X, that is, X = U·S·V', where
U is an m x k matrix containing only the first k columns of the unitary matrix U returned by [U,S,V] = svd(X)
S is a diagonal k x k matrix whose primary diagonal parts are the singular values (in decreasing order) of X
V is an n x k matrix containing only the first k columns of the unitary matrix V returned by [U,S,V] = svd(X)

Examples:

```
>> X = [ 0.60099 0.766416 0.440156; 0.12712 0.857445 0.130142; 0.94683 0.447486 0.559306 ]
 X =
       0.60099    0.766416    0.440156
       0.12712    0.857445    0.130142
       0.94683    0.447486    0.559306
```

```
>> S = svd(X)
 S =
      1.6967
      0.663471
      0.0347664
>> [U,S,V]=svd(X)
 U =
     -0.628061   -0.11714    -0.769297
     -0.41523    -0.78565     0.458627
     -0.658122    0.607481    0.444796
 S =
      1.6967          0           0
           0   0.663471           0
           0          0   0.0347664
 V =
     -0.620837    0.610289    0.492046
     -0.667115   -0.740937    0.0772603
     -0.411726    0.280286   -0.867134
>> X = [0.723014 0.179696 -0.861837 0.441228; -0.328791 0.934672 1.68603 0.955427]
 X =
      0.723014   0.179696  -0.861837   0.441228
     -0.328791   0.934672   1.68603    0.955427
>> [U,S,V]=svd(X)
 U =
     -0.293781  0.955873
      0.955873  0.293781
 S =
      2.25294        0  0  0
            0  1.07425  0  0
 V =
     -0.233779   0.553425   0.721934  -0.343335
      0.373129   0.415505  -0.509149  -0.654903
      0.827729  -0.305779   0.463201  -0.0825176
      0.347831   0.653893  -0.0708765  0.668142
>> [U,S,V]=svd(X,'econ')
 U =
     -0.293781  0.955873
      0.955873  0.293781
 S =
      2.25294        0
            0  1.07425
 V =
     -0.233779   0.553425
      0.373129   0.415505
      0.827729  -0.305779
      0.347831   0.653893
```

tan

Syntax
y = tan(x)

Definition
tan returns the tangent of the radian-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
tan(pi)	0
tan(pi/4)	1
tan(-pi/4)	-1
tan([5*pi/11 -5*pi/11])	[6.955 -6.955]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
atan (users)
tand (users)

tand

Syntax
y = tand(x)

Definition
tand returns the tangent of the degree-valued argument. This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
tand(180)	0
tand(45)	1
tand(-45)	-1
tand([180 45])	[0 1]

Compatibility
Numeric scalars, Vectors, Arrays

See Also
atan (users)
tan (users)

tanh

Syntax
y = tanh(x)

Definition
tanh returns the hyperbolic tangent of the argument, defined as (exp(x) - 1) / (exp(x) + 1). This function operates on an part-by-part basis on arrays.

Examples:

Formula	Result
tanh(1)	0.762
tanh(5)	1
tanh(pi/3)	0.781
tanh([pi/6 0])	[0.48 0]

Compatibility
Numeric scalars, Vectors, Arrays

See Also

atanh (users)

tcpip

Syntax

t = tcpip(ipAddr, nPort)

Definition

tcpip creates a class object to do tcpip i/o over a lan. ipAddr is a string with the IP Address in dotted format, and nPort is a port number for the connection. Once created, use fopen, fwrite, fread, fprintf, fscanf, fclose to manipulate the port.

Examples:

Formula	**Result**
t = tcpip('127.0.0.1', 80)	Create an object to connect to the web server on this computer (port 80 on 'this')

Compatibility

TCP/IP connections via LAN. ipAddr is a char array, and nPort is an integer.

tcpip Properties

Modify the way the tcpip link works by setting properties in the created class object. tcpip supports the following properties

Property	**Description**
LocalHost	Local host descriptor
LocalPort	Local port descriptor
LocalPortMode	Specify automatic local port assignment
ReadAsyncMode	Specfiy whether an asynchronous read operation.
RemoteHost	The remote host ip address (char array)
RemotePort	The remote port # (integer)
Terminator	Terminator string, such as 'CR/LF'. ASCII value 0 - 127, or 'CR', 'LF', 'CR/LF', or 'LF/CR'
TransferDelay	Specifies whether or not to use Nagle's algorithm.
InputBufferSize	Size of the input buffer in bytes.
OutputBufferSize	Size of the output buffer in bytes.
Timeout	Time to wait before timing out on receive (in seconds, floating point).

tic

Measure performance using stopwatch timer

Syntax:

tic

start_time = tic

Definition:

Starts a stopwatch timer. The output will be the time in ms since the operating system started.

Most commonly used with the function toc to measure the performance time of a set of statments.

Examples:

t1 = tic

<statments>

t2 = tic
<statmets>
dt1 = toc(t1) % the time elapsed since the first tic was called (or t1)
dt2 = toc(t2) % the time elapsed since the second tic was called (or t2)

Compatibility:
output is a double

See Also:
toc (users)

toc

Measure performance using stopwatch timer

Syntax:
toc
dt = toc
dt = toc(start_time)

Definition:

1. **toc** if there are no input and output, toc will just stop the timer.
2. **dt = toc** if only one output is asked, then dt will become the time elapsed since last tic was called.
3. **dt = toc(start_time)** this call will have the output dt to be the time elapsed since start_time, where start_time usually is the output of a tic call (eg. start_time = tic)

Examples:

- **Ex 1:**
  tic
  <statments>
  dt = toc % calculates the time elapsed since tic was used, or the time to run the code in the <statments>

- **Ex 2:**
  t1 = tic
  <statments>
  t2 = tic
  <statments>
  dt1 = toc(t1)
  dt2 = toc(t2)

Compatibility:
doubles

See Also:
tic (users)

toeplitz

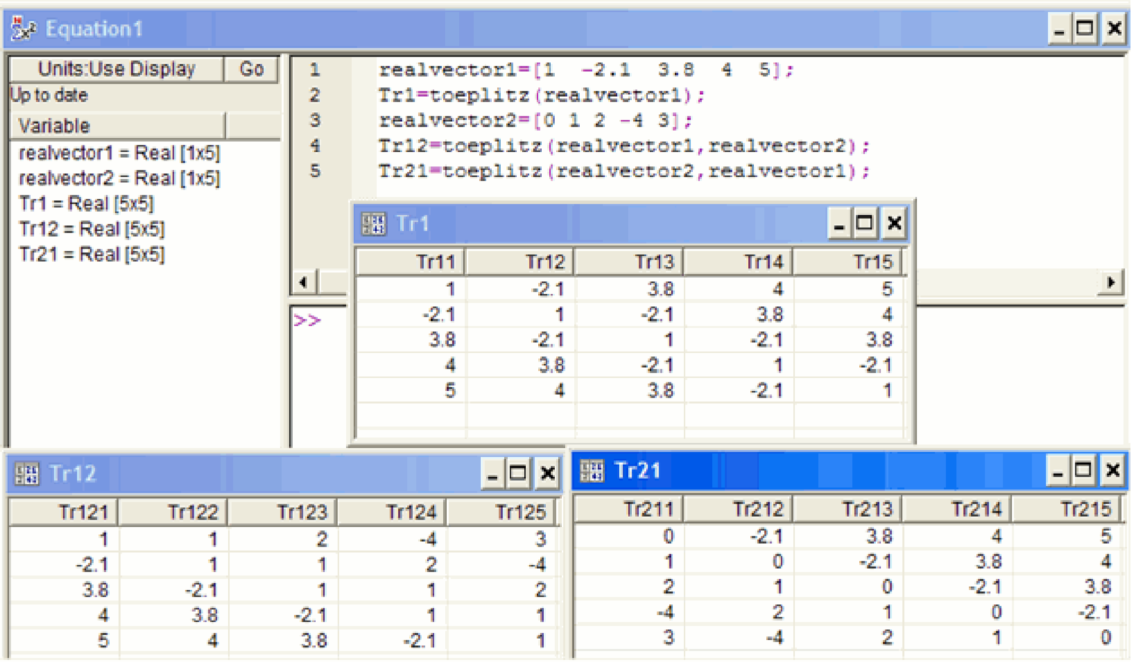
Syntax
tm = toeplitz(x)
OR
tm = toeplitz(x,y)

Definition
This function returns an m x m Toeplitz matrix based on an m-length vector x or a combination of m-length vectors x and y.

When only a single vector is used, the result is a symmetric, Hermitian matrix as shown in the Tr1 table below. Note that the vector parts are distributed symmetrically with respect to the principal dialoginal which is occupied by the first part of the input vector.

When two vectors are present, the first part of the first vector populates the principal diagonal as evidenced in the differences between Tr12 and Tr21. The other parts of the first vector populate the lower-triangle whereas those of the second vector populate the upper-triangle of the resultant matrix.

Examples:



realvector1 = Real [1x5]
realvector2 = Real [1x5]
Tr1 = Real [5x5]
Tr12 = Real [5x5]
Tr21 = Real [5x5]

```
realvector1=[1  -2.1  3.8  4  5];
Tr1=toeplitz(realvector1);
realvector2=[0 1 2 -4 3];
Tr12=toeplitz(realvector1,realvector2);
Tr21=toeplitz(realvector2,realvector1);
```

Tr1

Tr11	Tr12	Tr13	Tr14	Tr15
1	-2.1	3.8	4	5
-2.1	1	-2.1	3.8	4
3.8	-2.1	1	-2.1	3.8
4	3.8	-2.1	1	-2.1
5	4	3.8	-2.1	1

Tr12

Tr121	Tr122	Tr123	Tr124	Tr125
1	1	2	-4	3
-2.1	1	1	2	-4
3.8	-2.1	1	1	2
4	3.8	-2.1	1	1
5	4	3.8	-2.1	1

Tr21

Tr211	Tr212	Tr213	Tr214	Tr215
0	-2.1	3.8	4	5
1	0	-2.1	3.8	4
2	1	0	-2.1	3.8
-4	2	1	0	-2.1
3	-4	2	1	0

unmkpp

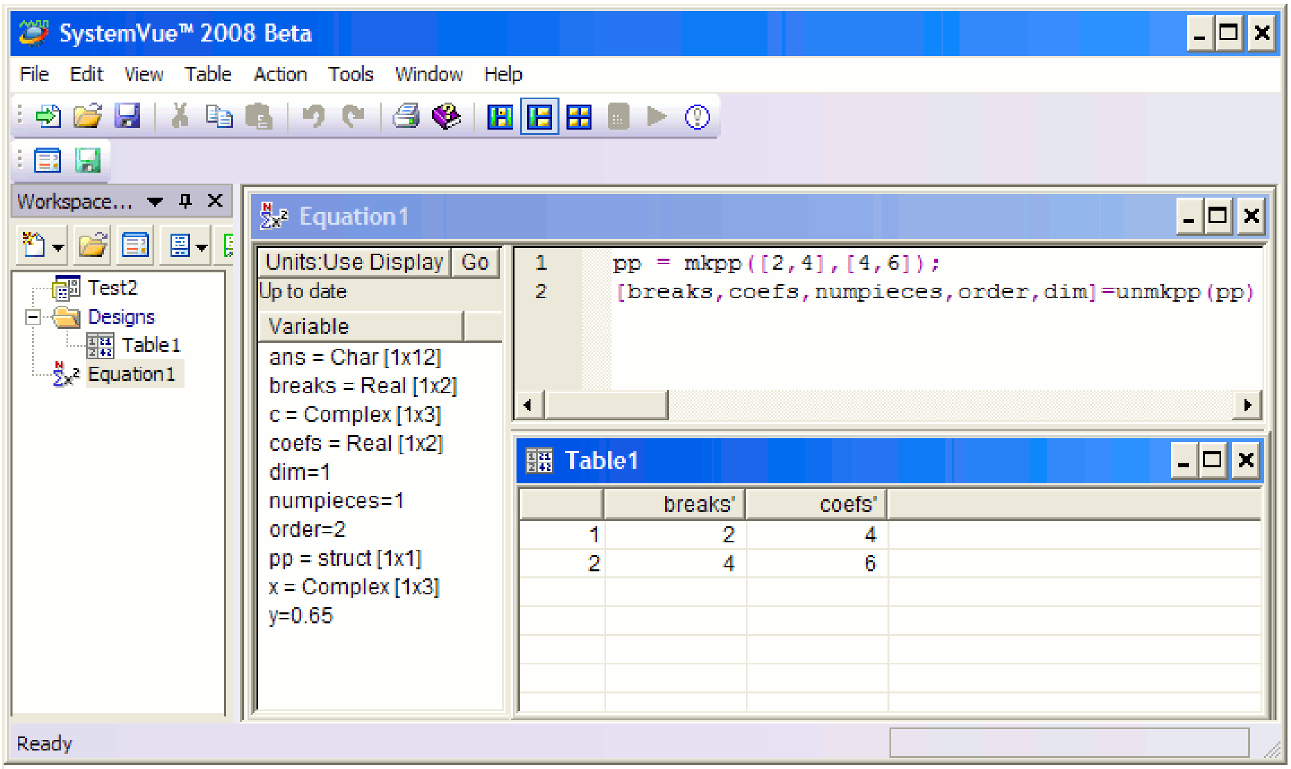
Syntax
[breaks,coefs,pieces,oredr,dimension] = unmkpp(pp)

Definition
This function extracts, from the supplied piecewise polynomial pp, its break points, coefficients, number of pieces, order, and dimension of target. Create pp using spline or the spline utility mkpp.
Breaks and coefficients are presented as row vectors.

Examples:

```
pp = mkpp([2 4],[4 6]);
[bks,coefs,l,k,d] = unmkpp(pp)
```

Result:



```
pp = mkpp([2,4],[4,6]);
[breaks,coefs,numpieces,order,dim]=unmkpp(pp)
```

```
ans = Char [1x12]
breaks = Real [1x2]
c = Complex [1x3]
coefs = Real [1x2]
dim=1
numpieces=1
order=2
pp = struct [1x1]
x = Complex [1x3]
y=0.65
```

	breaks'	coefs'
1	2	4
2	4	6

Compatibility
Structure

See Also
spline (users)
ppval (users)
mkpp (users)

using

Syntax
using('DatasetName');

Definition
This function sets the current context in an equation block to the named dataset. When set, you can use the variables within the dataset as if there were defined in the equation block. This function can be used to context switch between datasets in any post processing Equation page.

Examples:
If there are two datasets, called "Data1" and "Data2" which both contain a variable called "Var1".
Then the way to access these variables without confusion is as follows:

```
using('Data1');
% Assume "Var1" of "Data1" is [3, 6, 9, 12]
z1=Var1/3;
using('Data2');
% Assume "Var1" of "Data2" is [2, 4, 6]
z2=Var1/2;
```

The results are:

```
z1 = [1, 2, 3, 4]
Z2 = [1, 2, 3]
```

Compatibility
String

var

Syntax
y = var(x)
y = var(x, W)
y = var(x, W, iDim)

Definition
Returns the variance of a vector x.

If W is 0 (default), var normalizes by N-1 where N is the sample size. If W is 1, var normalizes by N. If W is a vector, it is treated as coefficient weights for computing the variance. In this case, the coefficients of W are scaled so that they sum to unity.

For matrices, this function operates separately on each column and returns a vector. For multi-dimensional arrays in general, this function operates on the dimension specified by iDim, or the first non-singleton dimension if iDim is not specified.

Examples:

Formula	**Result**
y = var([3 ; 4 ; 8 ; 9])	y = 8.6667
y = var([1, 2, 3], 1)	y = 0.6667
y = var([1, 2, 3], [0.7, 0.1, 0.2])	y = 0.65

Compatibility
Numeric arrays

See Also
kurtosis (users)
std (users)
skewness (users)

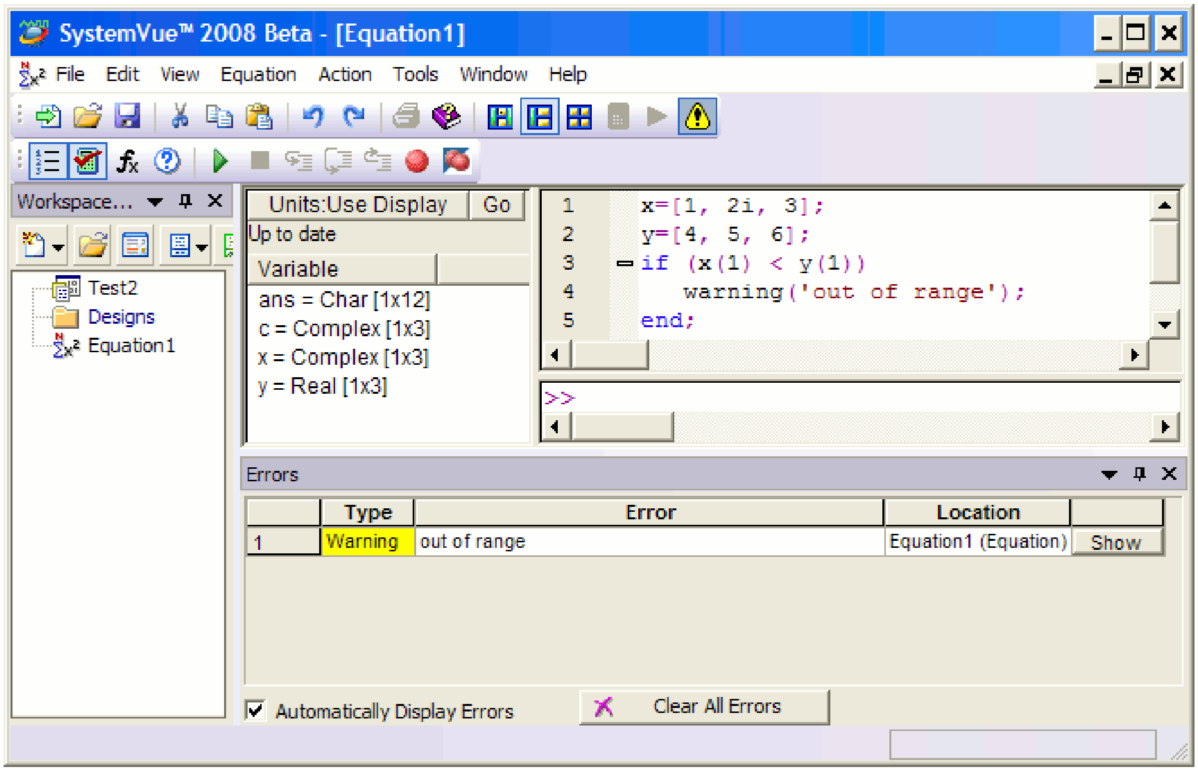
warning

Syntax
error('message')

Definition
Posts the warning message to the error log and also places the yellow warning symbol on the menu button.

Examples:

Formula	**Result**
warning('out of range')	the message "out of range" is posted to the **Error Log** as a warning



Variable	
ans = Char [1x12]	
c = Complex [1x3]	
x = Complex [1x3]	
y = Real [1x3]	

```
x=[1, 2i, 3];
y=[4, 5, 6];
if (x(1) < y(1))
   warning('out of range');
end;
```

```
>>
```

	Type	Error	Location	
1	Warning	out of range	Equation1 (Equation)	Show

Compatibility
Strings

See Also
error (users)

xcorr

Syntax
c = xcorr(x, y, maxlags, 'option')
[c, lags] = xcorr(...)

Definition
xcorr estimates the cross-correlation sequence of a random process. Autocorrelation is a special case of cross-correlation.

y, **maxlags**, and **'option'** are optional parameters.

When only **x** is specified i.e. c = xcorr(x) then **c** is the autocorrelation sequence for the vector **x**.

The various **'options'** are:

- **'biased'** - Biased estimate of the cross-correlation function Rxy_biased(m) = [1 / N] * Rxy(m)
- **'unbiased'** - Unbiased estimate of the cross-correlation function Rxy_unbiased(m) = [1 / (N - | m |)] * Rxy(m)
- **'coeff'** - Normalizes the sequence so the autocorrelations at zero lag are identically 1.0.
- **'none'** - Use the raw unscaled cross-correlations. This is the default.

maxlags - Limits the autocorrelation lag range to [-maxlags:maxlags].

[c, lags] = xcorr(...) returns two variables **c** and **lags**. **lags** is a vector of the lag indices at which c was estimated. The ' ... ' represent the x, y, maxlags, 'option' arguments.

Examples:

Formula	**Result**

x = [1, 2i, 3]
y = [4, 5, 6]
c = xcorr(x, y)
	c = [6 + i333.1e-18, 5 + i12, 22 + i10, 15 + i8, 12 - i333.1e-18]

xor

Syntax

y = xor(A, B)

Definition

This function performs an exclusive OR operation on arrays A and B.
It returns a vector of logical values that are true if only one of the corresponding values in A OR B is nonzero, but not both. Otherwise, the value is false. A and B have to be vectors or arrays of the same size.

Examples:

```
A = [0 0 pi eps], B=[0 -2.4, 0, 1]
C = xor(A, B) = [0, 1, 1, 0]
```

zeros

Syntax

y = zeros(m)
y = zeros(m, n)
y = zeros(m, n, p, ...)
y = zeros([m,n,p,...])
y = zeros(m, n, p, ..., class)
y = zeros([m,n,p,...], class)

Definition

This function returns a m by n by ... array with every part equal to 0. If only one argument is specified and it is a scalar m, then an m x m matrix is returned. A vector of dimensions may also be passed in. The optional class argument is a string that specifies the data type of the array to return.

Examples:

Formula	**Result**
y = zeros(3 , 2)	y = [0 , 0 ; 0 , 0 ; 0 , 0]
y = zeros(2)	y = [0 , 0 ; 0 , 0]
y = zeros([5 1])	y = [0 ; 0 ; 0 ; 0 ; 0]

See Also

ones (users)
eps (users)

Using Engineering Language

Engineering Language is a simple programming language with structured control statements. Variable types are defined in context and matrices and arrays can be easily manipulated with equations.

Example: here's a simple equation block

```
x=3
if x==3 then
 y = 4*x + cos(z)
else
 y = 4*x + sin(z)
endif
```

In this block we set *x* to be 3, then if *x* is 3 we set *y* to be 4* *x* +cos(*z*).

Recommendation: we recommend you create two separate equation blocks if you have input equations (such as tunable variables) and output equations (such as post-processing). This will generally speed up simulations and guarantee validity of the input variables even if the output data is messed up.

Equations (in both languages) are case-sensitive. So X=3 and x=3 define two different variables.

Important: You can use a one-line equation in most part parameters and in many other parameter entry fields.

In a part parameter that expects a string (such as substrate) precede the line with an =. For example,

```
=mysubst
```

will attempt to parse the mysubst formula and then find the substrate named that. If you have mysubst defined in an equation somewhere as mysubst="simple", then this will use the substrate named *simple* . Similarly, to define a string in parameters that may allow equations, precede the line with a single quote or use double-quotes around the line.

```
'simple
"simple"
```

both produce the string *"simple".*

Note: parameters that expect strings by default draw the string in **green**.

Statements

An equation block consists of one or more statements. Statements are separated by line breaks or semicolons. The following two equation blocks are equivalent:

```
X = 2
Y = 3
```

and

```
X = 2; Y = 3;
```

Complicated statements can span multiple lines and use control structures like while loops, for loops, and if statements.

The following statement types are supported by Engineering equations: assignment, comment, label, goto, if, for, while, function, or return. The format of each statement type is described below.

Assignments

An assignment statement assigns a value to a variable. The syntax of an assignment statement is as follows:

```
_Variable_Name_ = Expression
```

For example,

```
X = 3.6
Y = sin(3*PI)
Z = [1;2;3]
s21 = [My Folder].Linear1_Data.S21
```

are all assignments.

A variable name must start with a letter or an underscore character, and can contain alphanumeric characters and underscore characters. An expression can contain numerical operations involving numbers, other variables, and function calls.

Vectors and matrices can be defined inline, as the following example illustrates:

```
x = [1, 2, 3]      'A row vector
y = [1; 2; 3]      'A column vector
z = [1,2,3; 4,5,6] 'A 2x3 matrix
```

Comments

A comment starts with an apostrophe character (') and continues for the rest of that line. The following are examples of comments:

```
X = R * cos( theta )  ' here is one comment
' Here is another comment
```

In the example above, the assignment statement is executed, while both comments are ignored.

label statement

A label statement defines a point in the equation block for goto statement destinations. See the goto statement for more details. The syntax of the label statement is:

```
label _labelname_
```

goto statement

This statement causes the equation parser to jump to a label defined by a label statement and continue execution from that point. The syntax of the goto statement is:

```
goto _labelname_
```

if statement

The if statement is a control structure that allows one set of statements to execute if a condition is met, and optionally, another set of statements to execute if the condition is not met. Valid syntax for the if statement is:

```
 1.
    if _expression_ then
     one_or_more_statements
    else
     one_or_more_statements
    endif
 2.
    if _expression_ then
     one_or_more_statements
    endif
 3.
    if _expression_ then _statement_
```

If *expression* evaluates to a nonzero value, the statement block following *then* is executed, otherwise that statement block is skipped. If an else block is specified and expression evaluates to zero (false), the else block is executed. *Expression* is generally a Boolean expression.

Example:

```
x = 3
y = 2
if x == y then  ' Note that double equals are used for comparison
 x = x + 1
 y = y - 2
else
 x = 0
endif
```

for statement

The for loop statement is a control structure that allows a set of statements to repeatedly execute according to the value of the loop variable. The syntax is as follows:

```
 1.
    for _variable_name_ = _expression_ to expression
     one-or-more-statements
    next
 2.
    for variable_name = expression to expression step expression
     one-or-more-statements
    next
```

A loop variable named *variable-name* is initialized to the first *expression* value. The statement block following the for loop definition is executed, and the loop variable is incremented by 1 in the first syntax, or by the value of the *expression* after *step* in the second syntax. The statement block repeatedly executes until the loop variable exceeds the value of *expression* after *to.* The following example clarifies this:

```
x = 0; y = 0
for i = 1 to 5
 x = x + 10
 y = y + 100
next
```

After execution completes in the above example, i is equal to 5, x is equal to 50, and y is equal to 500.

while statement

The while loop statement is a control structure that executes a set of statements repeatedly based on a condition. The syntax is as follows:

```
while _expression_
 one-or-more-statements
wend
```

As long as *expression* evaluates to a nonzero number, the statements execute. Once the last statement in *one-or-more-statements* is reached, *expression* is once again evaluated. When *expression* evaluates to zero (false), execution continues after *wend.*

function statement

The function statement is used to define functions or procedures. A function takes zero or more parameters as input and returns exactly one value as output. All variables used within a function are local; that is, you cannot use variables defined in a function in another function or in the main equation block. The syntax of the function statement is as follows:

```
function _function-name_ ( _param1_ , _param2_ )
 one-or-more-statements
end
```

If the function takes no parameters, the parentheses must still be present after *function-name* . If the function returns a value, you should use a return statement in the *one-or-more-statements* block.

The following example is a function used to calculate the inductor value necessary to produce a resonance at a given resonant frequency and capacitor value:

```
function ResL(C, F)
 ' L is in nH, C is in pF, F is in MHz
 FHz = 1e6 * F
 CFarads = 1e-12 * C
 Omega = 2 * PI * FHz
 LHenries = 1 / (Omega^2 * CFarads)
 return LHenries * 1e9
end
```

Note: One or more return statements may be present in the function's statement block.

The function defined above may be called as follows:

```
L = ResL(50, 25.8) ' get me the L value in nH &nbsp;for 50 pF and 25.8 MHz
```

return statement

This statement returns a value from a function and exits the function. The format of the return statement is:

```
return expression
```

The function call then evaluates to the value of *expression* .

Operators

Operators and their descriptions are listed in the table below. Examples for each operator are also listed.

Operator	Description	Example
+	Addition	a + b
.+	Addition (ignores whether operands are swept or not)	a .+ b
-	Subtraction	a - b
.-	Subtraction (ignores whether operands are swept or not)	a .- b
*	Multiplication or Matrix Multiplication	a * b
.*	part-by-part Matrix Multiplication (ignores whether operands are swept or not)	a .* b
/	Division	a / b
./	part-by-part Division (ignores whether operands are swept or not)	a ./ b
^	Exponentiation	a ^ 2 (means a-squared)
%	Modulus	a % b
&	Boolean And	a & b
|	Boolean Or	a | b
~	Boolean Not	~a
=	Assignment Operator	a = 2
==	Boolean Comparison	a == b
>	Boolean Greater Than	a > b
>=	Boolean Greater Than or Equal	a >= b
<	Boolean Less Than	a < b
<=	Boolean Less Than or Equal	a <= b
<>	Boolean Not Equal	a <> b
@	Returns indices for which var = operand , F@50 returns indices at which F=50, F@[50,100]	Returns indices at which F is between 50 and 100.

Vectors, Matrices, and Multidimensional Arrays

Engineering Language allows you to create vectors, matrices, and multi-dimensional arrays. Vectors and matrices are arrays of 1 and 2 dimensions, respectively. Arrays are designed to work with numeric data, including complex numbers. Vectors and matrices may be defined inline, or may be defined and sized explicitly with the *vector*, *matrix*, and *array* functions. Here is an example equation block that defines several vectors and matrices, as well as a multi-dimensional array.

```
x = vector(3)              'x is a column vector of 3 parts, all zero.  The datatype is Complex.
y = matrix(4, 3)           'y is a 4x3 matrix with all parts zero.  The datatype is Complex.
```

```
z = array(3, 4, 4, 2)       'z is a 3x4x4x2 array.  The datatype is Complex.
a = [1; 2; 3]               'a is a column vector containing the parts 1, 2, and 3
b = [2.5, 3, 8]             'b is a row vector containing the parts 2.5, 3, and 8
M = [1,2,3; 4,5,6; 7,8,9]   'M is a 3x3 matrix with the first row containing 1, 2, and 3.
M = ["hello"; "goodbye"; "Hi"]  'M is a 3-part string array
```

Note that semicolon denotes the end of a row, while comma separates row parts. The exception to this is string arrays which are always 1-dimensional arrays, regardless of whether semicolons or commas are used to separate the parts when defining the array.

Indexing into Arrays

An part in an array variable is accessed with the following syntax:

```
var[index1, index2, ..., indexN]
```

where *var* is the name of the array, and *var* is an array of dimension N. The first part in a dimension has index 1. **A colon may be used to indicate every part in the dimension.** If the leading dimensions correspond to independent values (eg. the S matrix produced by Linear Analysis, which has the first dimension correspond to the independent value of F, a frequency vector) then these dimensions may be omitted, and a colon is assumed for these dimensions.

The following example illustrates indexing into arrays.

```
M = [1,2,3; 4,5,6; 7,8,9]     'M is a 3x3 matrix with the first row containing 1, 2, and 3.
a = M[2,1]       ' a equals 4
b = M[1,:]       ' b is the row vector [1,2,3]
c = M[:,[1;3]]   ' c is the 3x2 matrix [1,7; 2,8; 3,9]
M[1,1] = 5       ' the part in the first row and first column of M is now 5
```

Note that only scalar assignment is supported when assigning a value into an part in an array.

Vectors may also be used for specifying multiple parts in a dimension. The following example illustrates this:

```
M = [1,2,3; 4,5,6; 7,8,9]
a = M[ [1; 3], [1; 2] ]    ' a is the matrix [1,2; 7,8]
```

Searching Vectors and Indexing into Sweeps

Sometimes it is useful to know at what index or indices in an array a particular value is contained. To find what indices a vector contains a certain value or range of values, the @ operator may be used. This is especially useful for indexing into sweeps to extract desired data. A single number following the @ operator searches a vector for that single number. If a two-part vector follows the @ operator, the two numbers are treated as a range. The following example illustrates the use of the @ operator:

```
F = [100; 200; 300; 400; 500; 600]
i = F@300          ' i equals 3
n = F@[200,350]    ' n is the vector [2;3]
X = F[n]           ' X is the vector [200;300]
```

Suppose S is an S-matrix produced by linear analysis of a 2 port network. S contains data for 10 frequencies: 10 MHz through 100 MHz in steps of 10 MHz. That is, S has an independent value F, the frequency vector, of length 10. Therefore, S is a 10x2x2 array. The following example shows how to extract the S21 parameter at frequencies 30 - 50 MHz:

```
s_21 = S[F@[30,50], 2, 1]
```

Suppose we have a sweep that produced a swept variable PPORT which contains the power at ports. This variable is contained in a dataset called HB1_Data and has two independent variables because both frequency and input power were swept. Let us say that the frequency variable is Freq while the input power variable is called Pin. Suppose we wanted to extract the trace where the frequency is 2000 MHz and the power is swept from -15 to -11 dBm. We can make use of the *intersect* function to find the indices of PPORT corresponding to this trace as follows:

```
using("HB1_Data")
indices = intersect( Freq@2000, Pin@[-15,-11] )
trace = PPORT[indices]   ' trace contains the desired values of output power
```

String arrays may be searched in the same manner, but no ranges are allowed for strings:

```
strings = ["hello", "goodbye", "Hi"]
i = strings@"Hi"    ' i equals 3
j = strings@"hi"    ' i equals 0, since the string "hi" is not found in the array.
```

Range Vectors

A range defines a column vector in either of the following two ways:

```
start:stop
start:stepsize:stop
```

where *start, stepsize,* and *stop* are expressions. If *stepsize* is left out, it is assumed to be 1. A range creates a column vector with the first part value being equal to *start* , each successive part being *stepsize* greater than the previous part, until *stop* is reached. Ranges may also be used to index into arrays and extract desired sub-arrays. The following example illustrates the use of ranges.

```
x = 1:10     'x is the column vector [1;2;3;4;5;6;7;8;9;10]
y = 1:2:10   'y is the column vector [1;3;5;7;9]
M = [1,2,3; 4,5,6; 7,8,9]   ' M is a 3x3 matrix with the first row containing 1, 2, and 3.
a = M[1:2, 2:3]             ' a is the 2x2 matrix: [2,3; 5,6]
b = M[1:2:3, :]             ' b is the 2x3 matrix: [1,2,3; 7,8,9]
```

Mathematical Operations on Arrays

Mathematical operations on arrays are supported. In general, any scalar operation or function may be performed on an array, and the operation will be performed on an part-by-part basis, producing a resulting array that has the same dimensions as the original array. The following example illustrates this:

```
x = [1,2; 3,4]
```

```
y = [1,1; 1,1]
z = x + y      'z is the matrix [2,3; 4,5]
z = z - 1      'z is now the matrix [1,2; 3,4]
w = sin(z)     'w is the matrix [sin(1), sin(2); sin(3),sin(4)]
```

Multiplication is a special-case operator. When using the multiplication operator on a matrix or vector, matrix-multiplication is assumed. To do an part-by-part multiplication, the .* operator is used. Here is an example:

```
x = [1,2; 3,4]
y = [1;1]
z = x * y           'z is the vector [3;7]
w = x .* [1,0;0,1]  'w is the same matrix as x
```

To find out the dimensions of an array, use the *size* function. To find out how many parts are in an array, use the *prod* function to calculate the product of the parts of the vector returned by the *size* function:

```
x = [ 1,2,3; 4,5,6]
x_dims = size(x)            'x_dims is the vector \[2;3\]
num_parts = prod(x_dims) 'num_parts is 6
```

Shortcuts, and Functions

The following are built-in variables and functions that you can use in any equation block.

Variables

- FREQ - Frequency variable (in Hertz) set by the frequency-domain simulators
- TIME - Time variable (in Seconds) set by the time-domain simulator
- TEMPERATURE - Temperature (in Celsius) set by simulators
- PI - Constant equal to 3.1415926535897932

Shortcuts

- SUB_ER substrateer(SUBST) (compatibility with 2004 which used mm for units)
- SUB_TAND substratetand(SUBST) (compatibility with 2004 which used mm for units)
- SUB_RHO substraterho(SUBST) (compatibility with 2004 which used mm for units)
- SUB_TMET (1000*substratetmet(SUBST)) (compatibility with 2004 which used mm for units)
- SUB_ROUGH (1000*substraterough(SUBST)) (compatibility with 2004 which used mm for units)
- SUB_H (1000*substrateh(SUBST)) (compatibility with 2004 which used mm for units)
- SNM where N and M are digits S[N,M] (index into S-parameters)
- YPNM where N and M are digits YP[N,M] (index into Y-parameters. YP will be created and set to stoy(YP, ZPORT))

Measurement Functions

Measurement functions are typically used when looking at the output of linear simulations. Remember that all measurement data is in MKS fundamental units, so these functions use MKS input and produce MKS output. S21, for example, is not in dB.

sm_gamma1(S), sm_gamma2(S)

Returns simultaneous match gamma (reflection coefficient into port 1 or 2) for 2-port S parameters. Shortcut:

GMi sm_gammai(S)

sm_y1(S), sm_y2(S)

Returns simultaneous match input admittance (into port 1 or 2) for 2-port S parameters. Shortcut:

YMi sm_yi(S)

sm_z1(S), sm_z2(S)

Returns simultaneous match input impedance (into port 1 or 2) for 2-port S parameters. Shortcut:

ZMi sm_zzi(S)

stab_fact(S)

Returns stability factor (K) from 2-port S parameters. Shortcut:

K stab_fact(S)

stab_meas(S)

Returns stability measure(B1) from 2-port S parameters. Shortcut:

B1 stab_meas (S)

stoh(s, zport)

Converts 2-port S parameters to H parameters.

stoy(s, zport)

Converts S parameters to Y parameters. Shortcut:

YP stoy(S,ZPORT)

stoz(s, zport)

Converts S parameters to Z parameters. Shortcut:

ZP stoz(S,ZPORT)

voltage_gain(s, zport, i, j)

Returns port-to-port voltage gain from i to j. Shortcut:

Eij voltage_gain(S,ZPORT,i,j)

vswr(s)

Returns the vswr of an S-parameter. Shortcuts:

VSWR vswr(diag(S))   'returns a vector of all port VSWR value

VSWRi vswr(s[i,i])   'VSWR at port _i
_

y toz(Y)

Converts Y parameters to Z parameters.

zin(s, zport), yin(s, zport)

Returns the input impedance or admittance looking into a port. Parameters are S and Zport. Shortcuts:

ZIN zin(diag(S),ZPORT)   'returns a vector of all input impedances

YIN yin(diag(S),ZPORT)   'returns a vector of all input admittances

ZINi zin(S[i,i],ZPORT[i])   'returns input impedance at port *i*

YINi yin(S[i,i],ZPORT[i])   'returns input admittance at port _i
_

Variables in Datasets and other Equations

It is possible to refer to a variable that is located inside a dataset or an equation by using the dot operator (.) to specify the path of the variable in the workspace tree. If the object in the workspace tree that contains the desired variable does not have an alphanumeric name (eg. it contains spaces), then the name of the object must be surrounded by square brackets or braces. The following example illustrates the use of the dot operator:

```
X = Linear1_Data.S          ' The S variable in dataset Linear1_Data is copied to X
Y = [Equation set 1].x      ' Y equals the variable x defined in an equation block named "Equation
set 1"
Z = {Equation set 1}.x      ' same as Y
```

A graph series may refer to variables in the same manner. For example, entering the following measurement as a graph series is valid:

```
Linear1_Data.S21
```

Units of Measure

Data that comes from a dataset (simulation) will always be in MKS. So, for example, PPORT (power at a port) is in Watts. F (linear analysis frequency vector) is in Hz, T (Time) is in seconds. When you use this data in an equation, manipulate the MKS data. When you view the data in a graph or table you can see the table with a display unit of measure.

Example:

PPORT[2,2] - fundamental power at port 2 is stored in Watts but displayed in dBm

F[12] - 12th frequency is stored in Hz but displayed in MHz

To have an equation variable use a unit of measure, use the *setunits* function. So

```
F = [100e6;110e6]
setunits("F","MHz")
```

will create a two part vector at 100 and 110 MHz. Doing the setunits both identifies the type of data (time, here) and the desired (time) unit to display in (MHz). When you want to contrast measured and calculated results, remember to create your data in MKS.

Swept Arrays

A Swept Array is an array variable with an independent variable link. Typically, simulations create swept variables when they run. In Linear Analysis, S is a swept variable and the independent variable (link) is the F variable. When we plot S the X dimension is Frequency and the X coordinates are the values of the F array (the Y coordinates are usually dB(Sxx)).

Swept variables are used in equations differently from non-swept variables. With a swept variable the first dimension (the swept dimension) is often elided (left out) and the variable is treated like many variables with shape equal to the non-swept shape. Also, when swept variables are plotted the swept dimension (variable) is used for the X axis.

Example: in a transient analysis (CAYENNE simulator) of a 2-port device we have a VPORT vector of size Tx2 where T is the size of the time (T) vector. If we assume we have 100 times then T is of size 100x2 and we can do...

VPORT[2]   - the Port_2 voltage array swept over time (size 100x1)

VPORT[1,2] - the first voltage at port 2 (scalar)

[1,2]*VPORT - produce a swept array of size 100x1 where each value is VPORT [f,1] + 2 * VPORT[f,2]

treating VPORT as a bunch of 2 part vectors

Note that in this example both VPORT[2] and VPORT[1,2] are valid constructs because VPORT is swept.

To create a swept variable in equations, use the *setindep* function. This assigns an array as the independent vector for a second array.

Example: to create a swept voltage array

```
V = 1:100             ' create a vector of length 100 consisting of the numbers 1 through 100.
T = 1e-9*(1:100)      ' and again, except scaled by 1e-9
setunits("T","ns")    ' set T to ns(time)
setunits("V","mV")   ' set V to mV(voltage)
setindep("V","T")    ' say the voltage was swept by time
```

Example: if we want to contrast transient voltage with voltage squared (for some reason)

```
Vout = mydata.VPORT[2]      ' get the output port voltage from mydata
Vsq = Vout .* Vout          ' part-by-part mult. (Square each part)
setindep("Vsq","mydata.T") ' use the same indep as VPORT so we can graph it
```

Using Math Language

Math Language, along with most of its built-in functions, was designed to be compatible with m-file script syntax.

Statements

An equation block consists of one or more statements. Multiple statements placed on the same line are separated by line breaks, commas, or semicolons. The following two equation blocks are equivalent:

X = 2
Y = 3

and

X = 2, Y = 3

If you end a statement with a semicolon, it does not generate output in the command window.

Complicated statements can span multiple lines and use control structures like while loops, for loops, and if statements.

The following statement types are supported by Mathematics Language equations: assignment, comment, label, goto, if, for, while, function, or return. The format of each statement type is described below.

Assignments

An assignment statement assigns a value to a variable. The syntax of an assignment statement is as follows:

variableName = Expression

For example,

```
X = 3.6;
Y = sin(3*PI);
Z = [1 2 3];
```

are all assignments.

A variable name must start with a letter, and can contain alphanumeric characters and underscore characters. An expression can contain numerical operations involving numbers, other variables, and function calls.

Vectors and matrices can be defined inline, as the following example illustrates:

```
x = [1 2 3] % a row vector
y = [1;2;3] % a column vector
z = [1 2 3; 4 5 6] % a 2x3 matrix
```

Tune Assignments

A Tune Assignments assigns a variable a specific value while marking it as Tunable. A tunable variable can then be tuned from the Tune Window and can be used by evaluations, such as Sweeps, that operate on tunable variables.

When a Tunable variable is tuned from the Tune Window, the resultant value is then updated in the Equation block where it was originally defined.

The syntax for a Tune Assignment is as follows, where Constant represents a real-valued constant:

variableName = ?Constant

For example:

```
x = ?23   % x is tunable with initial value 23
y = ?-1.5  % y is tunable with initial value -1.5
```

Comments

A comment starts with a percent character (%) and continues for the rest of that line. The following are examples of comments:

```
X = R * cos( theta ) % Here is an in-line comment
% Here is another comment
```

In the example above, only the assignment statement is executed, while both comments are ignored.

if statement

The *if* statement is a control structure that allows one set of statements to execute if a condition is met, and optionally, another set of statements to execute if the condition is not met. Valid syntax for the if statement is:

```
  1.
     if _expression_
     _one_or_more_statements_
     elseif
     _one_or_more_statements_
     else
     _one_or_more_statements_
     end
  2.
     if _expression_
     _one_or_more_statements_
     end
  3.
```

```
    if _expression_, _statement,_ end
```

If *expression* evaluates to a nonzero value, the following statement block is executed, otherwise that statement block is skipped. If an else block is specified and expression evaluates to zero (false), the else block is executed. The *expression* is generally Boolean in construction.

Example:

```
x = 3
y = 2
if x == y % Note that double equals are used for comparison
x = x + 1
y = y - 2
else
x = 0
end
```

for statement

The *for* loop statement is a control structure that allows a set of statements to repeatedly execute according to the value of the *loopVariable*. The syntax is as follows:

```
for _loopVariable_ = _startValue_ : _stepValue_ : _stopValue_
_one_or_more_statements_
end
```

The *loopVariable* is initialized to the *startValue*. When *stepValue* is explicitly mentioned, *loopVariable* increments by it until it reaches or exceeds the *stopValue*. When left unspecified, *stepValue* is assumed to be of unit magnitude. The following example clarifies this:

```
x = 0, y = 0
for i = 1 : 5 % i, the _loopVariable_ takes the values [1 2 3 4 5]
x = x + 10
y = y + 100
end
```

After execution completes in the above example, i is equal to 5, x is equal to 50, and y is equal to 500.

while statement

The *while* loop statement is a control structure that executes a set of statements repeatedly based on a condition. The loop is exited when the condition is no longer satisfied. The syntax is as follows:

```
while _expression_
_one_or_more_statements_
end
```

As long as *expression* evaluates to a nonzero number or a Boolean true, the statements execute repeatedly. When *expression* evaluates to zero (false), execution continues after *end*. The following example clarifies this:

```
x = 1, y = 15;
while ( y )
x = x * y;
y = y - 1;
end
```

After execution completes, when y reaches 0 in the above example, x equals factorial of the original value of y.

function statement

The *function* statement is used to define functions or procedures. A function takes zero or more parameters as input and returns exactly list of values as the result. All variables used within a function are local; that is, you cannot use variables defined in a function in another function or in the main equation block. However, you can use variables defined in the equation block in the function. The syntax of the function statement is as follows:

```
function <resultList> = functionName( <paramList> )
_computation_statements_
_calls_to_other_functions_
end % Note: this end is optional
```

If the function takes no parameters, the parentheses must still be present after *functionName* . If the function returns a value, you should set the values in the <resultList> block.

<paramList> and <resultList> are lists of variable names separated by commas. If the last variable in <paramList> is 'varargin', then the function can take in an unspecified number of arguments, and the remaining arguments are placed into the 'varargin' cell array which can be accessed from within the function. Similarly, if the last variable in <resultlist> is 'varargout', then the function can return an unspecified number of return values which are set in the function by assigning to the 'varargout' cell array.

Inside a function definition, you may use the variables named 'nargin' and 'nargout' which hold the number of arguments passed in to the function and the number of return values requested by the caller, respectively. These may be used for error checking or other purposes.

The following example is a function used to calculate the inductor value necessary to produce a resonance at a given resonant frequency and capacitor value:

```
function resonantInductor = ResL( resonantCapacitor, resonanceFrequency )
% inductance is in nH, capacitance is in pF, frequency is in MHz
FHz = 1e6 * resonanceFrequency;
CFarads = 1e-12 * resonantCapacitor;
Omega = 2 * pi * FHz;
LHenries = 1 / (Omega^2 * CFarads);
resonantInductor = LHenries * 1e9; % the return value
end
```

The function defined above may be called as follows:

L = ResL(50, 25.8) % computes the L value in nH resonanct with 50 pF at 25.8 MHz

You may return multiple values by listing them in the result expression, as in

```
function [Ind, Q] = ResL( C, F, R )
Ind = 1
Q = 2
end
```

this is used as [MyInd, MyQ] = Resl(a,b,c)

The following example illustrates a function that takes in a variable number of arguments and returns a variable number of results.

```
function varargout = f(varargin)
SumOfArgs = 0;
for i = 1 : nargin
SumOfArgs = SumOfArgs + varargin{i}
end
varargout{1} = SumOfArgs
if nargout > 1
varargout{2} = 2 * SumOfArgs
end
end
```

Suppose we call this function as follows:

```
[a, b, c] = f(1, 2, 3, 4)
```

a would be set to 10 (the sum of the input arguments), b would be set to 20, and c would be blank since it was not assigned to in the function.

Operators

Operators and their descriptions are listed in the table below. Examples for each operator are also listed.

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Matrix Multiplication	a * b
.*	part-by-part Matrix Multiplication	a .* b
/	Matrix Right-Division	a / b
./	part-by-part Division	a ./ b
\	Matrix Left-Division	a \ b
.\	part-by-part Left-Division	a .\ b
^	Matrix Exponentiation	a ^ 2 (means a-squared)
.^	part-by-part Exponentiation	a .^ b
'	Matrix conjugate-transpose (hermitian)	a'
.'	Matrix transpose (no conjugation)	a.'
&	part-wise Boolean And	a & b
&&	Boolean And	a && b
|	part-wise Boolean Or	a | b
||	Boolean Or	a || b
~	Boolean Not	~a
=	Assignment Operator	a = 2
==	Boolean Comparison	a == b
>	Boolean Greater Than	a > b
>=	Boolean Greater Than or Equal	a >= b
<	Boolean Less Than	a < b
<=	Boolean Less Than or Equal	a <= b
~=	Boolean Not Equal	a ~= b

Vectors, Matrices, and Multidimensional Arrays

Mathematics Language supports vectors, matrices, and multidimensional arrays. Column vectors are treated as Nx1 matrices, while row vectors are 1xN matrices. Vectors and matrices can be defined inline using bracket notation, as shown below.

```
a = [1;2;3] % a is a column vector containing the parts 1, 2, and 3
b = [2.5 3 8] % b is a row vector containing the parts 2.5, 3, and 8
c = [1, 2, 3] % c is a row vector containing 1, 2, and 3. Commas are optional
M = [1 2 3; 4 5 6; 7 8 9] % M is a 3x3 matrix with the first row containing 1, 2, and 3.
M = ['help1'; 'help2'; 'help3'] % M is a 3 by 5 character array
M = ['help1' 'help2' 'help3'] % M is the string 'help1help2help3'
```

Note that semicolon denotes the end of a row, while comma separates row parts.

Indexing into Numeric Arrays

An part in an array variable is accessed with the following syntax:

```
_matrixVariable_(index1, index2, ..., indexN)
```

where *matrixVariable* is the name of the N-dimensional array.
A colon may be used to indicate every part in the dimension. If only one indexing dimension is specified, then the array is linearly indexed, which means that the array is

treated as a flat list (in column-wise order) and the N'th part of the list is returned.

> Note: If only one indexing dimension is specified and it is a colon, then the array is returned as a single column vector with N parts, where N is equal to the number of parts in the array.

The following example illustrates indexing into arrays.

```
M = [1 2 3; 4 5 6; 7 8 9] % M is a 3x3 matrix with the first row containing 1, 2, and 3.
a = M(2,1) % a equals 4
b = M(1,:) % b is the row vector [1,2,3]
c = M(:,[1;3]) % c is the 3x2 matrix formed by taking the columns of the 1st and 3rd columns of
all the rows [1,3; 4,6; 7,9]
d = M(:) % d is the column vector [1;4;7;2;5;8;3;6;9]
e = M(6) % e equals 8 because it is the 6th part when M is traversed column-by column
M(1,1) = 5 % sets the value of the part in the first row and first column of M to 5
```

Vectors may also be used for specifying multiple parts in a dimension. The following example illustrates this:

```
M = [1,2,3; 4,5,6; 7,8,9]
a = M( [1; 3], [1; 2] ) % a is the matrix [1,2; 7,8]
```

Multi-dimensional arrays are formed by combining arrays of smaller dimensions in nested fashion using [s and semi-colons. For instance, a three dimensional array of size 3x2x2 would have two levels of []s:

```
M3D = [ [1, 2, 3; 4, 5, 6]; [-1, -2, -3; -4, -5, -6] ]
```

Searching Vectors and Indexing into Sweeps

> The example named *Indexing Sweeps with Math Lang.wsx* located in the Examples\Equations folder walks through the material discussed in this section.

Sometimes it is useful to know at what index or indices in an array a particular value is contained. To find what indices a vector contains a certain value or range of values, the *find* function may be used. This is especially useful for indexing into sweeps to extract desired data. The following example illustrates a simple case using the *find* function:

```
F = [100; 200; 300; 400; 500; 600]
i = find(F == 300)    % i equals 3
n = find(F >= 200 & F <= 350)   % n is the vector [2, 3]
X = F(n)          % X is the vector [200;300]
```

Suppose V1 is a waveform of voltages as a function of time. V1 contains data for 101 timepoints: 0 ns through 100 ns in steps of 1 ns. That is, V1 has an independent value T, the time vector, of length 101. Therefore, V1 is a 101x1 array. The following example shows how to extract a subset of the voltage waveform and construct a new time independent variables corresponding to that subset:

```
% Suppose V1 and T already exist, as described above
time_indices = find(T >= 10e-9 & T <= 20e-9);  % indices where T is between 10 and 20 ns
V1_subset = V1(time_indices);  % extract waveform between 10 and 20 ns
time_subset = T(time_indices); % ditto for the time indep
setindep('V1_subset', 'time_subset');  % now if we plot V1_subset, we see a nice x-axis
```

We can use the same approach for indexing into multi-dimensional sweeps. The key is to use the *find* function to extract the correct indices.

Indexed Assignments

Mathematics Language supports assigning a value or values into arrays. If you assign data to parts outside the current dimensions of an array, the array is automatically re-sized to accommodate the new data, while any new parts in the array are initialized to zero.

When assigning from one array to another in the form A = B, the following rules must be obeyed:

- The number of subscripts specified for array B not including trailing 1's may not exceed the number of dimensions of B
- The number of non-scalar subscripts specified for A is equal to the number of non-scalar subscripts specified for B
- The length and order of all non-scalar subscripts specified for A is equal to the length and order of all non-scalar subscripts specified for B

The following code example illustrates various aspects of indexed assignments. Initially the variable x does not exist.:

```
x(2,3) = 5; % x is created to be a 2x3 matrix with the entry at (2,3) equal to 5 and the other
parts equal to zero so x is [0, 0, 0; 0, 0, 5]
x(:,2) = [11; 22] % x is now [0, 11, 0; 0, 22, 5]
x(1, [1 3]) = [100 200] % x is now [100, 11, 200; 0, 22, 5]
x(1:6) = 1:6 % x is now equal to [1 3 5; 2 4 6]
x(1,1,2) = 23 % now x is a 2x3x2 array with x(1,1,2) equal to 23
```

Range Vectors

A range defines a row vector in either of the following two ways:

start:stop

start:stepsize:stop

where *start*, *stepsize,* and *stop* are expressions. If *stepsize* is left out, it is assumed to be 1. A range creates a row vector with the first part value being equal to *start* , each successive part being *stepsize* greater than the previous part, until *stop* is reached. Ranges may also be used to index into arrays and extract desired sub-arrays. The following example illustrates the use of ranges.

```
x = 1:10 % x is the row vector [1 2 3 4 5 6 7 8 9 10]
y = 1:2:10 % y is the row vector [1 3 5 7 9]
M = [1 2 3; 4 5 6; 7 8 9] % M is a 3x3 matrix with the first row containing 1, 2, and 3.
a = M(1:2, 2:3) % a is the 2x2 matrix: [2,3; 5,6]
b = M(1:2:3, :) % b is the 2x3 matrix: [1,2,3; 7,8,9]
```

Mathematical Operations on Arrays

Mathematical operations on arrays are supported. In general, any scalar operation or function may be performed on an array, and the operation will be performed on an part-by-part basis, producing a resulting array that has the same dimensions as the original array. The following example illustrates this:

```
x = [1,2; 3,4]
y = [1,1; 1,1]
z = x + y % z is the matrix [2 3; 4 5]
z = z - 1 % z is now the matrix [1 2; 3 4]
w = sin(z) % w is the matrix [sin(1), sin(2); sin(3),sin(4)]
```

Multiplication is a special-case operator. When using the multiplication operator on a matrix or vector, matrix-multiplication is assumed. To do an part-by-part multiplication, the .* operator is used. Here is an example:

```
x = [1 2; 3 4]
y = [1;1]
z = x * y %z is the vector [3; 7]
w = x .* [1 0;0 1] % w is the same matrix as x
```

To find out the dimensions of an array, use the *size* function. To find out how many parts are in an array, use the *length* function:

```
x = [1 2 3; 4 5 6]
x_dims = size(x) % x_dims is the vector [2 3]
num_parts = length(x) % num_parts is 6
```

Cell Arrays

Cell arrays are arrays that support each part having a differing data type. Each part in a cell array is called a cell. As an example, you may have a 1x3 cell array in which the first cell is a number, the second cell is a character array, and the third cell is a structure. Furthermore, parts of cell arrays may be cell arrays themselves. Cell arrays, just like numeric arrays, may have any number of dimensions. Cell array vectors and matrices may be defined inline as shown here:

```
X = { [1 2; 3 4] 'abc' 3j } % X is a 1x3 cell array containing a 2x2 real matrix, a 1x3 character
array, and a complex scalar
Y = { {1 2}; {1 2; 3 4} } % Y is a 2x1 cell array containing a 1x2 cell array and a 2x2 cell array
```

Indexing into Cell Arrays

There are two ways to index into a cell array, described here:

```
M{indices} % returns the contents of the cell at the index specified by indices
M(indices) % returns the cell or cells at the index or indices specified by indices
```

Numeric arrays contained in cell arrays may be indexed inline as well:

```
M{2,3}(6) % returns 6th part of the array contained in the cell array M at location (2,3)
M{2,3}{2}(6) % returns 6th part of the array contained in the 2nd part of the cell array located
in the cell array M at location (2,3)
```

The following example illustrates indexing into cell arrays.

```
M = { 1 'abcd' [2j 56; 3 j] {6 7} } % M is now a 1x4 cell array
a = M{1} % a equals 1
```

```
b = M{2} % b equals the 1x4 character array 'abcd'
c = M(2) % c equals a 1x1 cell array that contains 1 part: a 1x4 character array 'abcd'
d = M{3}(1,1) % d equals 2j
e = M{4} % e equals the cell array {6 7}
```

Structures

A structure is a data type with named *fields*. Each field has a name and a value. The value may be of any type, including a cell array or another structure. Structure arrays of any number of dimensions are supported. In a structure array, all structures in the array have the same field names.

Structures may be defined inline as shown here:

```
x.field1 = 23; % x is a structure with a field named "field1" with value 23
x.hello = {1 2}; % x now has another field named "hello" whose value is a 1x2 cell array
x(3).hello = 1; % x is now a 1x3 structure array with fields "field1" and "hello". The third
part's hello field has value 1.
```

You may use the *fieldnames* function to determine what field names are in a structure. *fieldnames* returns a cell array of strings.

Structures may also be built using the *struct* function.

Network Communication and Instrument Control

The Math Language includes TCP/IP communication capabilities. This enables control of instruments.

When you create an equation set to do communication you will almost always want it to be not Auto-Calc. It should only calculate when specifically requested, otherwise every time an input variable changes it will rerun. It will also run on load. Turn off auto-calc by clicking the Check-Calculator tool button when viewing the equation set.

TCP/IP communication is done via the tcpip class, which is constructed using the *tcpip* function. A simple example follows (waitfor is a wait-for-character routine, PSAip contains the IP address string of an instrument, while PSASpciPort contains the port number to use for communications):

```
% - set up the tcpip pipe to the instrument
t = tcpip(PSAip, PSASpciPort) % build tcpip object using the PSA ip address and spci port
t.Terminator = 'CR/LF'; % set Terminator field
t.InputBufferSize = 100000; % use a big buffer
% - open the port &nbsp;
fopen(t)
% - set real data format
fprintf(t, 'form:data real,64')
% - swap byte order
fprintf(t, 'form:border swap')
% - read the trace
fprintf(t, 'trace? trace1') % tell it to send the first trace
a3 = waitfor(t, '#') % the # is followed by some count chars
% - get the # of count bytes
aBytecnt = fread( t, 1, 'uchar=>ushort')
tTotal = str2num( aBytecnt)
% - if valid # count bytes, read them
if tTotal > 0 && tTotal < 7 % we will never have more than 6 digits of stuff
```

```
ascCount = fread(t, tTotal, 'uchar=>ushort'); % read n count bytes
nCount = str2num( ascCount); % convert to numeric
nCount = nCount / 8; % convert to doubles at 8 bytes each
else
nCount = 0;
end
% - finally read the actual data
if nCount > 0
dInput = fread(t, nCount, 'double') % get nCount data values
setvariable('OutData', 'aOut', dInput) % save it in our dataset
end
% - close t so we rerun cleanly
fclose(t)
```

Analyzing the previous example

We start by creating a tcpip class object connected to our PSA device. PSAip=='127.0.0.1' or some valid ip address as a char array. PSAspciPort is an integer port number. Once the object is built, we set the terminator (for telnet in this case) and the input buffer size (plenty to avoid overflow).

We do fopen(t) which opens the socket connection.

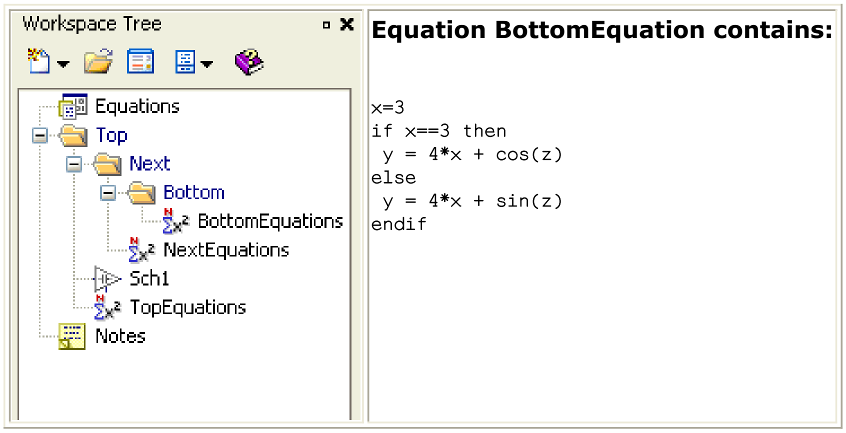
Once connected you can use

fread - read nnn values from the data stream
fwrite - write nnn values to the data stream
fprintf - write a string to the data stream
fscanf - read a string from the data stream

When finished, close the socket by using fclose. If you are totally done with the socket you can use the Math Language clear function to remove the class object entirely.

Hierarchy in Equations

Equations obey hierarchy as defined by their place in the Workspace Tree. Note that this is true for Equation objects as well as equations that are embedded inside a Design object (ie. an Equation tab in a Design).



Equation BottomEquation contains:

```
x=3
if x==3 then
 y = 4*x + cos(z)
else
 y = 4*x + sin(z)
endif
```

In the example above, the value of *z* will be coming from another equation set (NextEquations or TopEquations) to execute without errors. The Equations engines look up the workspace hierarchy until the value of *z* is found, otherwise an error is reported.

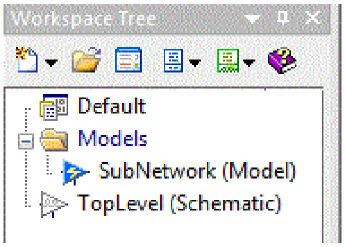
In other words, in this example, if *z* is defined in NextEquations, then *z* will come from there. If NextEquations does not define *z* then the Equations engine looks up another folder level to TopEquations for *z* .\

Equations on the same level of hierarchy should all be visible to each other.

Design-time vs. Run-time hierarchy

The above discussion of hierarchy is called design-time hierarchy, because while you are not simulating, the workspace tree defines the scoping of variables. However, when you run a simulation, the situation can be different.

Suppose, for example, that we have 2 designs as shown in the below picture. Both designs have an Equation tab (and possibly a Parameters tab, which is equivalent, since Parameters get passed into the design's Equations at run-time).

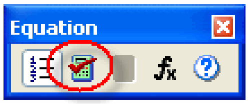


In the situation shown in the above picture, when you are NOT running a simulation (ie. you are in design-time), the design called SubNetwork will be able to see variables that are defined in the Equations tab of the design called TopLevel, simply because SubNetwork is located in a folder beneath the level of hierarchy that TopLevel is in. However, suppose that, as shown, SubNetwork defines a subnetwork model. Also suppose that the schematic in TopLevel defines an instance of SubNetwork (ie. it has a part that references the SubNetwork model). When you run a simulation (ie. during run-time), a Model hierarchy is defined in which SubNetwork is a child of TopLevel, since an instance of a SubNetwork model is instantiated inside of TopLevel. Because of this, SubNetwork can see all of TopLevel's variables. This is what makes the passing of parameters from TopLevel to SubNetwork possible.

It is important to note that when you are editing a design (ie. you are in design-time), the values of parameters you see in your design are those calculated using the design-time hierarchy. For example, if you define a Design that contains Parameters, and you use one of those parameters inside your Design, you will see the value of that parameter correspond to the "Default" value of the Parameter that you defined in the Parameters tab. This is, of course, not necessarily the value that will be seen at run-time when you run a simulation, since that depends on the run-time hierarchy defined by the topology of the network you are simulating.

Automatic Calculation

If an Equation object is set to Auto-Calculate, the equations are always kept up to date whenever a value is requested from them. This is desirable when the equation block defines variables that you use in part parameters on a schematic: when you change these values, you want the part parameters that use them to update. **However, sometimes this is undesirable.** If, for example, you are using an Equation block to import data from a file or to transfer data to and from an instrument, you do not want the Equations to calculate unless you specifically tell them to. In these cases, you should disable Auto-Calculate. The Auto-Calculate toolbar button located on the *Equation Toolbar* (users) toggles automatic calculation on and off.



> There are some cases where you probably want to DISABLE automatic recalculation of an equation block: equations which do file I/O or TCP/IP communications, equations which run simulations via the *runanalysis* function, equations that do time-consuming processing.

If you disable *Automatic Calculation* , the only way to recalculate the equation is manually with the calculate button, or with the F5 or Ctrl+G keyboard shortcuts. Equations that have *Automatic Calculation* turned off will not update during simulations. As mentioned before, you would normally disable *Automatic Calculation* for Math Language equations that control hardware, for example, so the hardware doesn't get re-controlled every time a variable changes.

> If you disable Auto-Calculate in an equation that is a function definition, the function won't exist until you manually calculate the equation.

Debugging Equations

A fully featured and intuitive debugger is built-in to the equation editing user interface.

Using Breakpoints and Single-Stepping

You can use the *Equation Toolbar* (users) or its associated keyboard shortcuts in the equation script editor to set breakpoints and to step through your code one line at a time. Breakpoints can be set both in equations contained in the workspace and in a model's equations (eg. sub-circuits). In all cases, evaluation of equations will be halted when a breakpoint is hit. The user may then execute statements line by line using single-stepping, abort execution, or continue execution until the next breakpoint is hit.

- **Workspace Equations**: to run the equations click the Go button in the *Equation Toolbar* (users). If a workspace equation is set to Auto-Calculate, they will calculate whenever something they are dependent on triggers a calculation. If any breakpoints are set, the evaluation of the equations is halted and the user interface is brought to the front, clearly marking what line of code the equation processor is currently halted at.

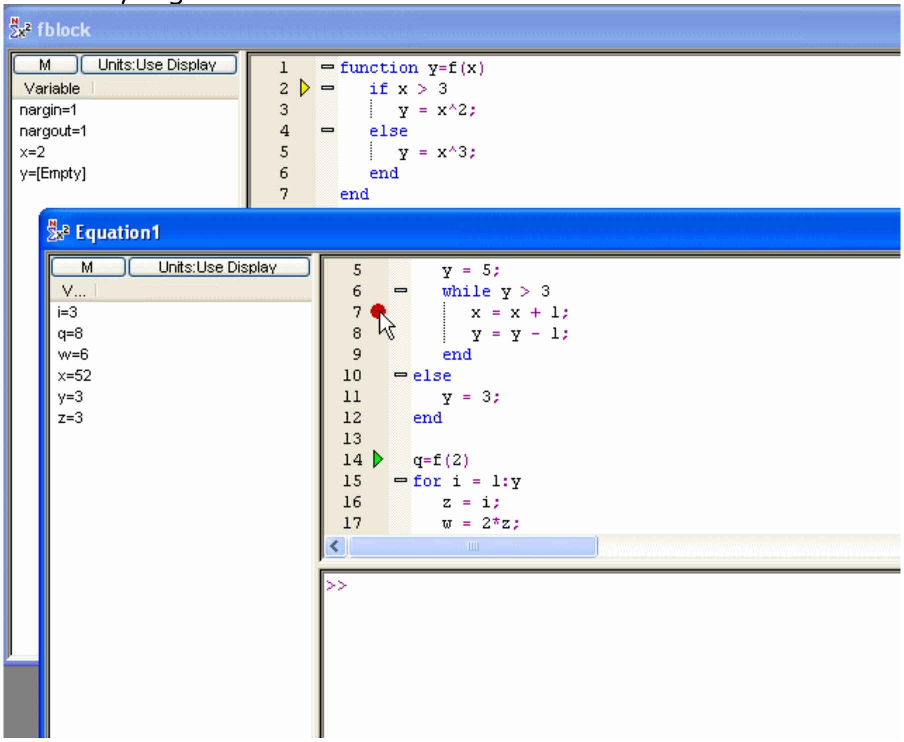
It is important to keep in mind that an equation block may be calculated may times due to various factors, such as a simulator changing a variable. The evaluation of the equations will halt whenever the breakpoint is hit. Typical scenarios include:

- **Equations in sub-circuit models**: the breakpoint will be hit once per run of the simulator *except* when the equation is dependent on the simulator independent variable. For example, in the linear simulator, frequency dependent equations will calculate at each frequency. For CAYENNE (time domain), time dependent equations will calculate at each T.
- **Equations in a Math Language block**: the breakpoint will be hit at each 'tic' of the simulator as data is delivered to the block.

Setting Breakpoints

Click the Breakpoint Margin at the line you wish to set a breakpoint at in the script editor window to toggle a breakpoint on/off. A red dot will appear when the breakpoint is on. The Breakpoint margin is located between the Line Number and Folding margins. You may also set a breakpoint at the current line by using the Ctrl+B keyboard shortcut or clicking the Add/Remove Breakpoint toolbar button.

When the equation processor hits a breakpoint, the current line it is halted at will display a yellow arrow ▷ in the breakpoint margin as can be seen in the picture below. At this point you may single-step, step-into functions, continue, or abort execution. If you step-into a function, a green arrow ▶ marks the line that the function was called from.



nargin=1
nargout=1
x=2
y=[Empty]

```
function y=f(x)
   if x > 3
      y = x^2;
   else
      y = x^3;
   end
end
```

i=3
q=8
w=6
x=52
y=3
z=3

```
    y = 5;
    while y > 3
       x = x + 1;
       y = y - 1;
    end
else
    y = 3;
end

q=f(2)
for i = 1:y
    z = i;
    w = 2*z;
```

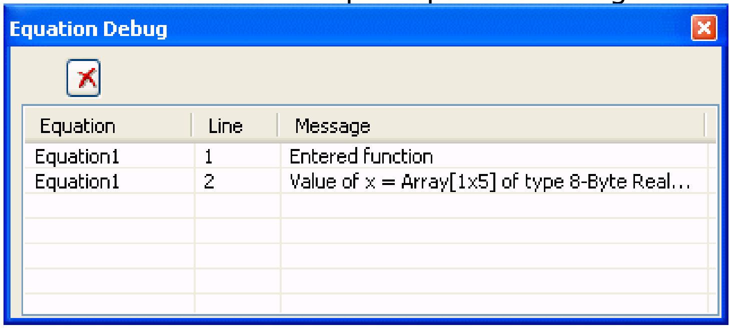
```
>>
```

Using Debug Print functions

The debug print functions shown below produce lines of debug text in the Equation Debug docking window.
Please note that debug lines will only appear in the window after the simulator runs, due to current multi-threading locks.

Equation Debug docking window

The Equation Debug docking window can be shown/hidden using the Edit/View/Docking Windows/Equation Debug menu path or using the show/hide dockers button on the main toolbar. This window has a list of debug lines that your equations generate using functions described below. A sample Equation Debug Window is shown here.



Equation	Line	Message
Equation1	1	Entered function
Equation1	2	Value of x = Array[1x5] of type 8-Byte Real...

Debug Functions

There are two functions available (in both Engineering Language and Mathematics Language) for writing to the Equation Debug docking window. Both functions add lines to the Equation Debug Docking Window so you can trace progress as the program runs. The code samples are written in Mathematics Language.

1.
```
dbg_print( 'Message' )
dbg_print( 'Message', 'Equation')
dbg_print( 'Message', 'Equation', Line)
```
prints the Message in the Equation Debug window. The Equation and Line parameters may be omitted, in which case the equation engine will attempt to auto-detect which equation set and line number called the function.

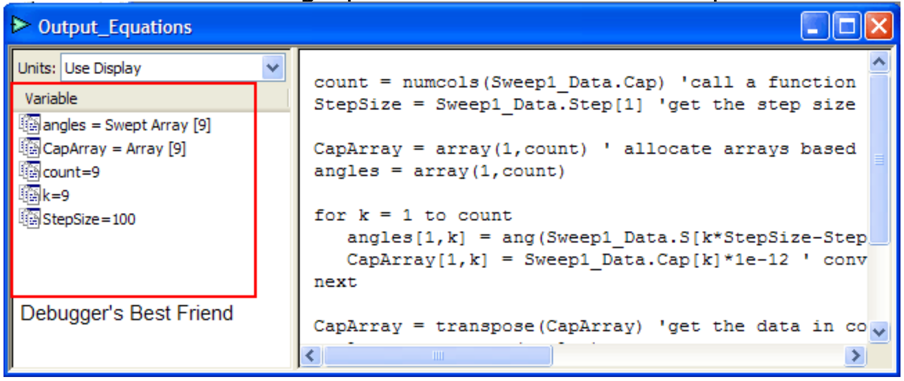
```
dbg_showvar( 'Message', Variable)
```

prints Message=VariableValue in formatted output

Tips for Effective Equation Writing

As a program becomes more complex, it becomes necessary to carefully debug and test the results. Breakpoints and Debug-Print functions can be very helpful, as has already been discussed. In general, however, there are several things one should get accustomed to doing when writing equations. Below are some tips to follow when an equation is causing difficulty:

1. **Make sure the input and output equations are in separate blocks**
   It is a bad idea to have something like:
   c = ?4 ' value of some capacitor in the schematic
   s21 = Linear1_Data.S[2,1] ' s21 from analyzing the schematic
   The "c" is an input to a schematic; it MUST exist before Linear1_Data is ever created, so this equation block will not compile reliably. Any equation statements that call variables from analysis datasets should be in a separate block.
2. **Let each line compile cleanly before typing more text**
   Avoid the temptation to write a long set of statements before verifying that it works; type one line at a time and check that there are no error messages, and that the variables are showing up in the left side of the equation editor.



angles = Swept Array [9]
CapArray = Array [9]
count=9
k=9
StepSize=100

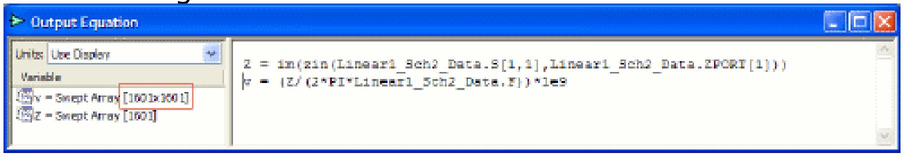
Debugger's Best Friend

```
count = numcols(Sweep1_Data.Cap) 'call a function
StepSize = Sweep1_Data.Step[1] 'get the step size

CapArray = array(1,count) ' allocate arrays based
angles = array(1,count)

for k = 1 to count
   angles[1,k] = ang(Sweep1_Data.S[k*StepSize-Step
   CapArray[1,k] = Sweep1_Data.Cap[k]*1e-12 ' conv
next

CapArray = transpose(CapArray) 'get the data in co
```

3. **Before writing a large loop or in-line vector statement, check the boundary values**
   Instead of writing a large loop then wondering why there are out of bounds errors or wrong calculations, first type something like:
   testA = myVector[firstIndex]
   testB = myVector[lastIndex]

The values will display in the Variable view; this way you first verify that the initial and final values are as expected; then you can let the loop or vector operation run with more confidence.

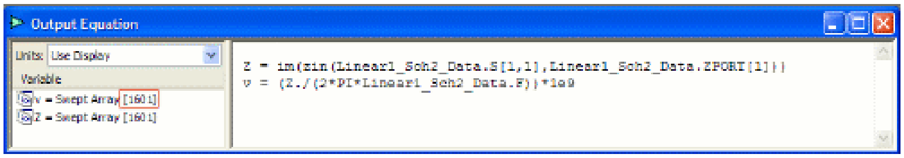
4. **Don't try to pack everything into one line of code**
   It is very difficult to find the problem when there are too many calculations packed into a one line statement. By breaking up a line into several variables and lines you give yourself the chance to debug and find problems, rather than just look at a huge line that doesn't work as intended.
5. **Check dimensions of variables carefully**
   Always pay attention to the size and dimension of variables being used; a common pitfall is to use incorrect multiplication or division of vectors and thus accidentally create wrong-sized matrices or other unwanted results.



v = Swept Array [1601x1601]
Z = Swept Array [1601]

```
Z = im(zin(Linear1_Sch2_Data.S[1,1],Linear1_Sch2_Data.ZPORT[1]))
v = (Z/(2*PI*Linear1_Sch2_Data.F))*1e9
```

Careless use of the "/" operator causes a 1601x1601 matrix to be created; the variables view alerts the user of the problem, so part-wise division can be used instead:



v = Swept Array [1601]
Z = Swept Array [1601]

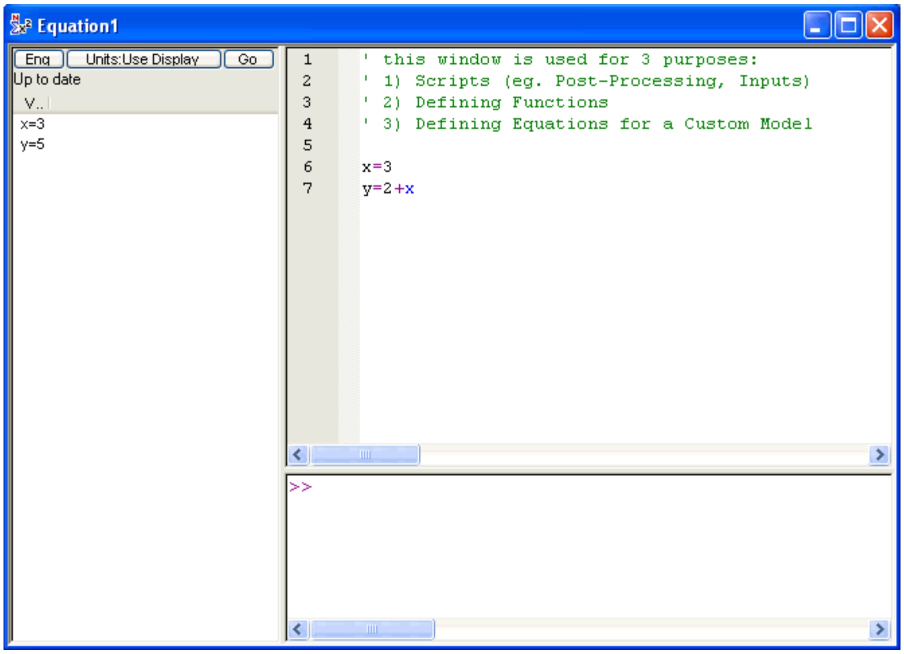
```
Z = im(zin(Linear1_Sch2_Data.S[1,1],Linear1_Sch2_Data.ZPORT[1]))
v = (Z./(2*PI*Linear1_Sch2_Data.F))*1e9
```

Note that the functions **numcols(myMatrix)** and **numrows(myMatrix)** can be used to find the dimensions of a variable. For matrix operations, the number of columns of a left-hand operator should equal the number of rows of a right-hand operator, while for part-wise operations the dimensions should be identical.

6. **Use the Command Window to output or change variable values**
   See Equations User Interface for more information about the Command Window.
7. **Use the online help**
   The online help for equations is extensive. You can select a keyword in the equation editor and press F1 for context help on that keyword. General equation help is in the User's Guide manual Using Equations section.

Equations User Interface

The following image shows a typical Equations window:



```
' this window is used for 3 purposes:
' 1) Scripts (eg. Post-Processing, Inputs)
' 2) Defining Functions
' 3) Defining Equations for a Custom Model

x=3
y=2+x
```

The Equations window has three subwindows:

- The **Variable Viewer**, located on the left.
- The **Script Editor**, located on the upper right.
- The **Command Window**, located on the lower right.

The Equations window also has an associated toolbar, see *Equation Toolbar* (users).

Among the simplest application of equations is to define a variable in the Script Editor area (upper right), such as myvar=123 (then press "Go")
Then myvar can be entered into component properties on the schematic, to drive component values.
Entering myvar=?123 (ie. adding the question mark) makes the value myvar tunable in the tune window.
After pressing "Go", the variable value should appear in the Variable Viewer (left side). If nothing appears in the Variable Viewer after pressing "Go", this usually indicates some problem with equation syntax.
Typically the error messages window will provide some clues.

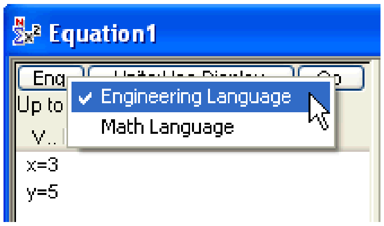
Variable Viewer

The Variable Viewer displays any variables that currently exist in the Equations object. If the variable is a scalar, the value is displayed. If the variable is an array, the type and dimensions of the array are displayed.

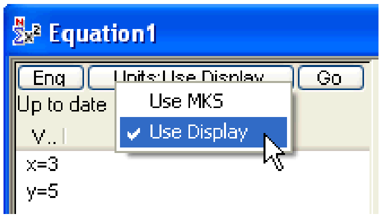
If you right-click on any variable displayed in the Variable Viewer, you will be presented with a menu containing options to plot the variable on a graph or display it in a table. If you wish to see the variable's value without creating a table, you can do so in the Command Window, as discussed below.

The following buttons are located at the top of the Variable Viewer window: Equation Language, Units, and Go.

The Equation Language button allows you to set the language of the Equations to be defined, as shown here:



The Units button allows you to define how the values of the variables shown in the variable list are to be interpreted when used elsewhere, such as part-parameters. If the Units are set to "Use MKS", then the value of the variables in this Equations block will be treated as MKS values. If, on the other hand, the Units are set to "Use Display", then the units will be defined where the value is actually used.



"Use Display" means to interpret the unit of measure of a parameter as a scale factor. So, if an equation variable X=20 is used in an Inductor set to nH in Use MKS the inductor value is 20H (MKS) in Use Display the inductor value is 20nH. "Use MKS" is very important for Model portability and units portability.

You will almost always want to use "Use Display", since you will usually want a unit to be attached wherever the variable is used. "Use MKS" may be used for model equations to ensure portability of models regardless of an end user's unit preferences.

The Go button provides an easy way to force execution of the equations. Its function is equivalent to the Go button on the *Equation Toolbar* (users).

In Engineering Language, the variable block is always cleared before the equations are run when you hit the Go button. This is not true in Math Language. You must add the *clear* statement as the first line in your Math Language equations to achieve this same behavior.

Script Editor

The Script Editor is used to type in a sets of equation statements to be executed. More specifically, the Script Editor window is used to:

- Post-Process data, or define variables as inputs to be used elsewhere.
- Create user-defined functions.
- Define equations inside a Model.

The Script Editor includes Find and Replace support, accessible from the Edit menu or with the Ctrl+F or Ctrl+H keys, respectively.

New If you want context sensitive help on a function, select the keyword and press F1 in the Script Editor.

New Use Ctrl_MouseWheel to zoom in and out on the equations Script Editor.

Command Window

The Command Window is used to execute statements line-by-line. It interacts with the same variables that are visible to the Script Editor. It is a useful debugging tool since the contents of a variable can be displayed here.

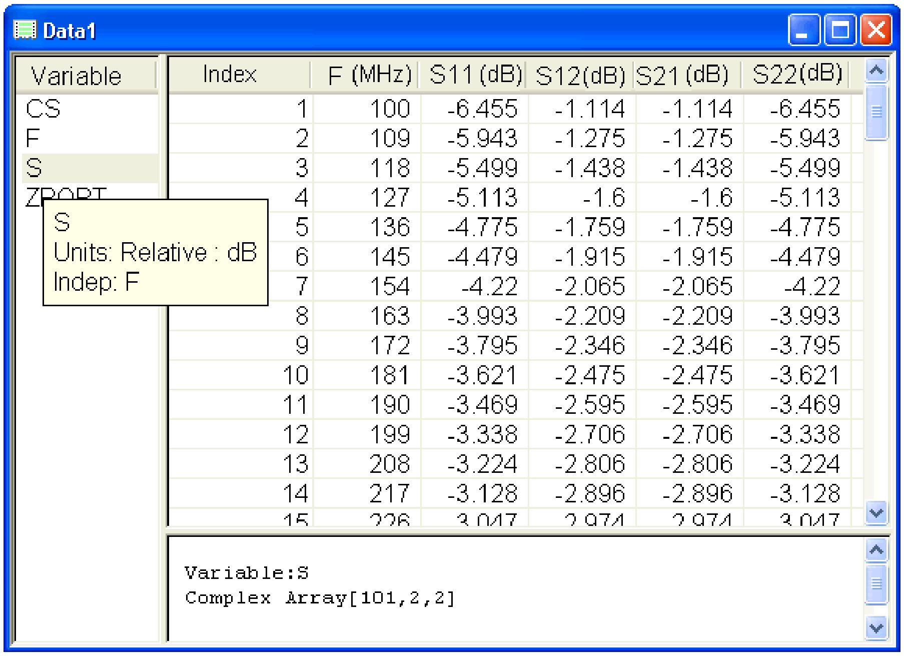
If an assignment statement does not end in a semicolon, the results of that assignment are outputted in the Command Window, as can be seen in the above figure. If an assignment statement does end in a semicolon, then the dump of the contents of the result is suppressed.

Any errors or warnings caused by executing a line in the Command Window are outputted to the Command Window.

Examining Datasets

Datasets are containers which hold data, such as the results of a simulation or a table of input. The results are stored in Variables which can be viewed in tabular form within the dataset, plotted on a graph, displayed in an output Table, etc. Examine a dataset by opening it with a double-click. You can also add new variables to a dataset (for sweeping or just for analizing the data in greater detail).

Open the Linear Simulation Template (via the Start Page). On Linear1_Data, click the variable "S" on the left-side of the window, to see its values. Hovering the mouse over a variable pops up some info, which varies according to the measurement.



S
Units: Relative : dB
Indep: F

Index	F (MHz)	S11 (dB)	S12(dB)	S21 (dB)	S22(dB)
1	100	-6.455	-1.114	-1.114	-6.455
2	109	-5.943	-1.275	-1.275	-5.943
3	118	-5.499	-1.438	-1.438	-5.499
4	127	-5.113	-1.6	-1.6	-5.113
5	136	-4.775	-1.759	-1.759	-4.775
6	145	-4.479	-1.915	-1.915	-4.479
7	154	-4.22	-2.065	-2.065	-4.22
8	163	-3.993	-2.209	-2.209	-3.993
9	172	-3.795	-2.346	-2.346	-3.795
10	181	-3.621	-2.475	-2.475	-3.621
11	190	-3.469	-2.595	-2.595	-3.469
12	199	-3.338	-2.706	-2.706	-3.338
13	208	-3.224	-2.806	-2.806	-3.224
14	217	-3.128	-2.896	-2.896	-3.128
15	226	3.047	2.974	2.974	3.047

```
Variable:S
Complex Array[101,2,2]
```

In the display above the left-hand pane shows all of the result variables (including **F**, the frequency or independent variable). The right-hand pane shows whatever piece of data you have selected in the left pane. The upper left-corner box in the grid is the units of measure (MHz down and dB for the values). The lower right pane (which is usually collapsed – drag the divider bar upwards to see it) displays a summary of the variable information.

Each type of analysis creates a different dataset with differing variables which are determined by the Analysis. Often, the variable is directly associated with a particular measurement. Linear analysis (as seen above) creates data with F (frequency), S, ZPORT (port impedance) and CS (noise correlation matrix).

Each dataset contains variables, which can be matrices, vectors, or scalars. These variables are either automatically created by simulation runs or manually by the user. Note that when a dataset is created by a simulation, the data within that dataset is always in MKS. You may *display* the data in a unit of your choice, but the actual data values are MKS values.

Click **S** on the left to show the tabular display of values in the grid on the top-right. It shows that the frequencies analyzed were 100, 109, 118, ..., 1000 MHz. The S-Parameters are shown in columns. The single grid-cell (top left corner of the grid) which says MHz:dB shows that the units for Frequency are MHz and the data values are shown in dB. The display on the bottom-right (which is usually collapsed) shows the type and size of the clicked data.

In addition to seeing the simulation results, Datasets can have short equations to help you analyze and diagnose issues with your circuits. For details, see *Creating Variables* (users).

Contents

Creating Datasets

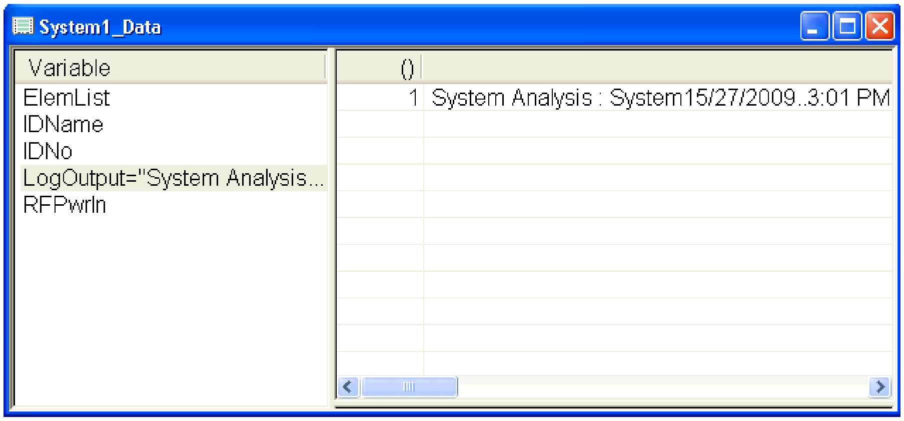
Datasets are usually created automatically when Analyses run. Some analyses (particularly SPECTRASYS) can create more than one dataset. Within the dataset are the fundamental results – measurements created by the simulation.

In addition, a blank dataset can be created manually from the workspace tree (in the docking window) via the "new item" button (although that is rarely neccesary).

The actual data within a dataset is determined the Analyses settings. CAYENNE and HARBEC and SPECTRASYS all let you limit which data is created during the simulation run. This can reduce the size of datasets significantly and also reduce their complexity.

To examine a dataset, open it by double-clicking it in the workspace tree.

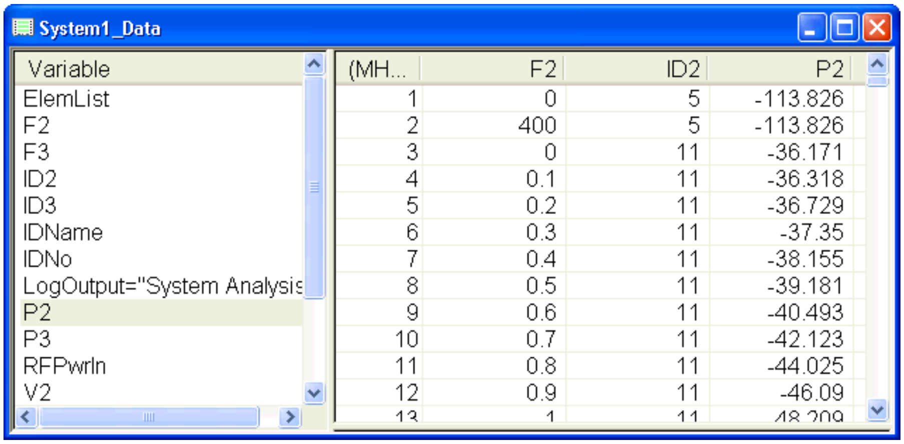
Here's a minimal SPECTRASYS dataset:



Variable
ElemList
IDName
IDNo
LogOutput="System Analysis...
RFPwrIn

	()
1	System Analysis : System15/27/2009..3:01 PM

If we rerun SPECTRASYS with all of the output options enabled, we get this:



Variable

ElemList
F2
F3
ID2
ID3
IDName
IDNo
LogOutput="System Analysis
P2
P3
RFPwrIn
V2

(MH...	F2	ID2	P2
1	0	5	-113.826
2	400	5	-113.826
3	0	11	-36.171
4	0.1	11	-36.318
5	0.2	11	-36.729
6	0.3	11	-37.35
7	0.4	11	-38.155
8	0.5	11	-39.181
9	0.6	11	-40.493
10	0.7	11	-42.123
11	0.8	11	-44.025
12	0.9	11	-46.09
13	1	11	-48.209

Now we can't even fit the entire dataset contents in the window.

Although more complex and intimidating there are many cases where more data is better than less. However, file storage requirements go way up with this sort of data.

Creating Variables

Variable Properties Dialog Box

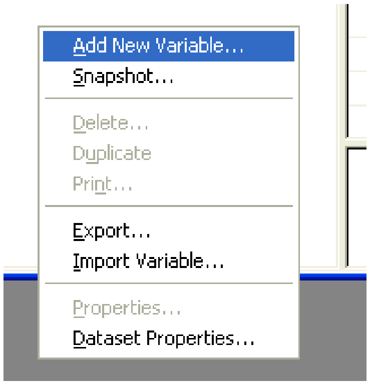
For complete description of **Variable Properties** dialog box, see *Variable Properties* (users)

Why add variables to a dataset?

1. Add a variable to examine more closely a piece of data (such as ang(S[2,1]) to examine S21's angle). Don't forget that all measurement data is fundamentally in MKS units.
2. Add a variable to propagate it during a sweep (enable the propagate option in the sweep and it will sweep the variable along with the rest of the measurement data).
3. Add a variable to use in an optimization.

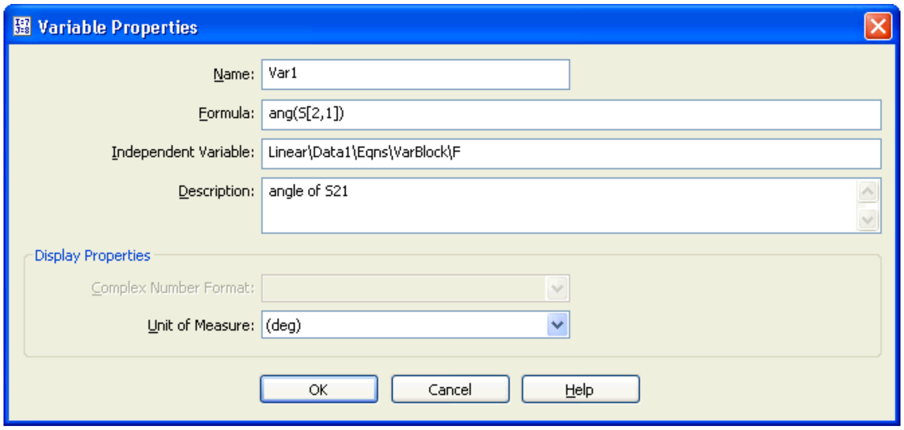
How to add a variable to a dataset

1. Open Linear Simulation Template / Linear1_Data, as described above.
2. Right-click the white area on the left and select Add New Variable...



3. Add a variable named Var1.
4. Type ang(S[2,1]) for the formula.
5. Leave the Independent Variable field blank; it will be automatically filled in based on the indep associated with the "S" variable.
Optionally, you can choose a display option for the dataset view of the variable. If the

6.
variable type is integer or floating point, select a display unit; if it's complex, select a complex number formatting option.
7. Click OK.



Name: Var1

Formula: ang(S[2,1])

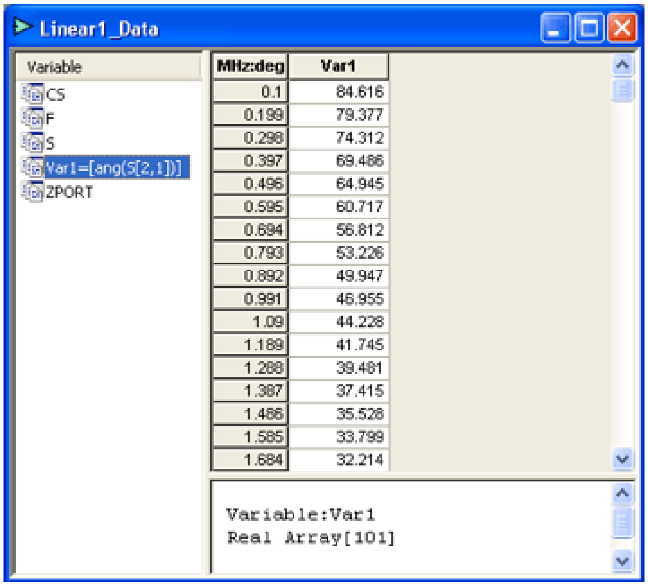
Independent Variable: Linear\Data1\Eqns\VarBlock\F

Description: angle of S21

Complex Number Format:

Unit of Measure: (deg)

8. To get...



MHz:deg	Var1
0.1	84.616
0.199	79.377
0.298	74.312
0.397	69.486
0.496	64.945
0.595	60.717
0.694	56.812
0.793	53.226
0.892	49.947
0.991	46.955
1.09	44.228
1.189	41.745
1.288	39.481
1.387	37.415
1.486	35.528
1.585	33.799
1.684	32.214

```
Variable:Var1
Real Array[101]
```

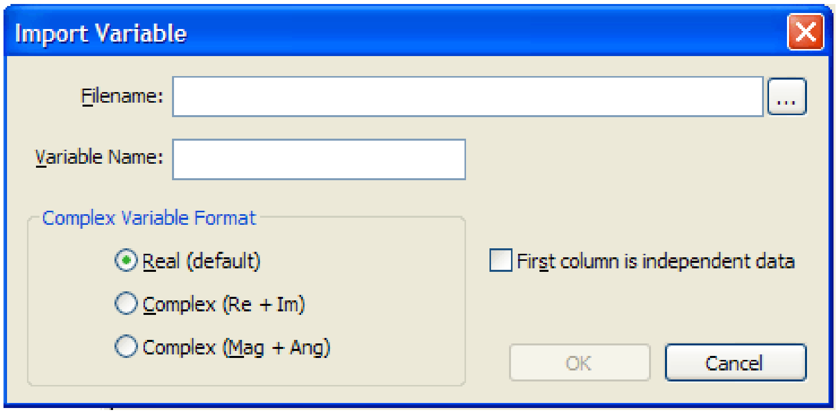
9. For most formulas, the Unit of Measure and Independent Variable will fill themselves in once the formula is parsed.

How to delete a variable from a dataset

- Right-click the variable and select Delete

Importing Variables

Variables can be imported to the dataset from any text file. Access this feature by right-clicking in the variable block of the data set and choosing "Import Variable".



Import Variable

Filename:

Variable Name:

Complex Variable Format

● Real (default)

○ Complex (Re + Im)

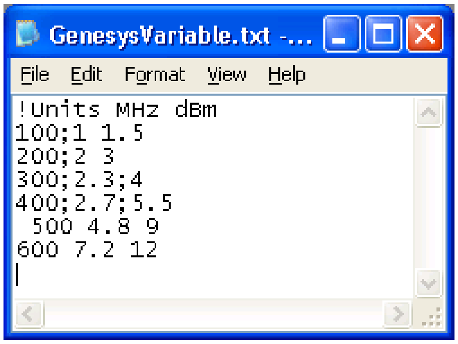
○ Complex (Mag + Ang)

□ First column is independent data

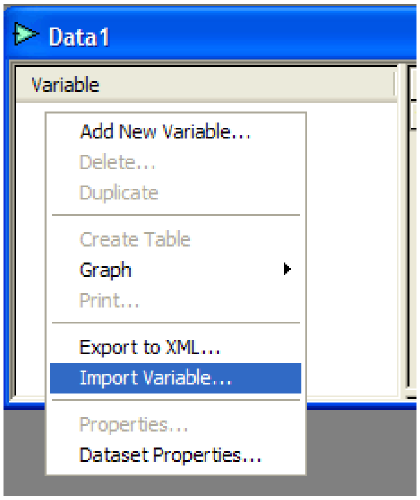
Browse and select a file. Enable "First Column is Independent Data" if the first column of the data is independent data (swept). Name the variable in the Variable Name field.

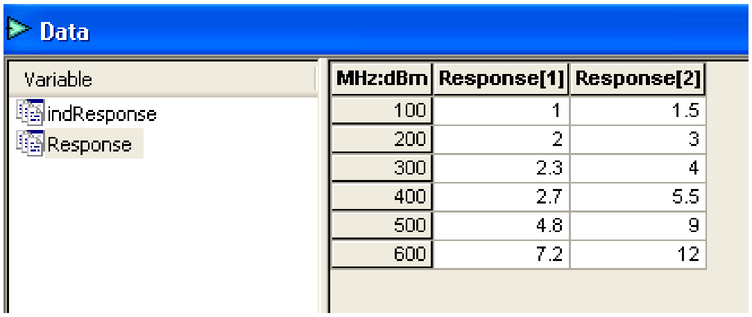
The data should be formatted as a list or matrix of numbers. Semicolons (" ; ") and spaces (" ") are used to indicate breaks between values Other characters are treated as zeroes. Begin the data with !Units *unitindep unitdep* to define a unit of measure for the data. Other rows that begin with a ! are ignored as comments.

Example (Choose Real, check First column as independent)



```
!Units MHz dBm
100;1 1.5
200;2 3
300;2.3;4
400;2.7;5.5
 500 4.8 9
600 7.2 12
```

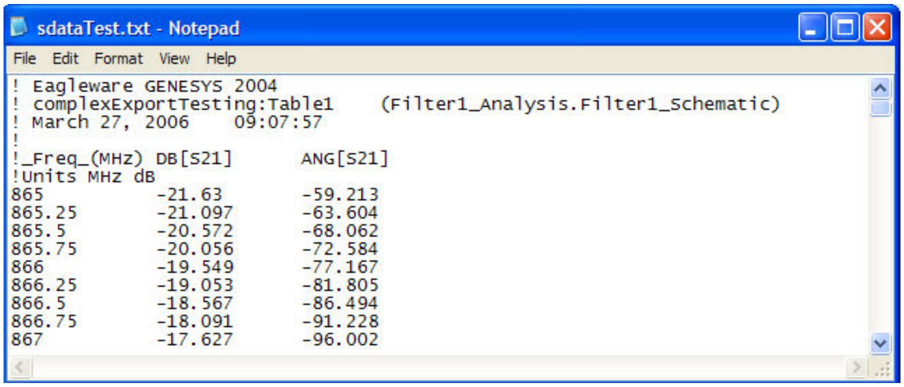




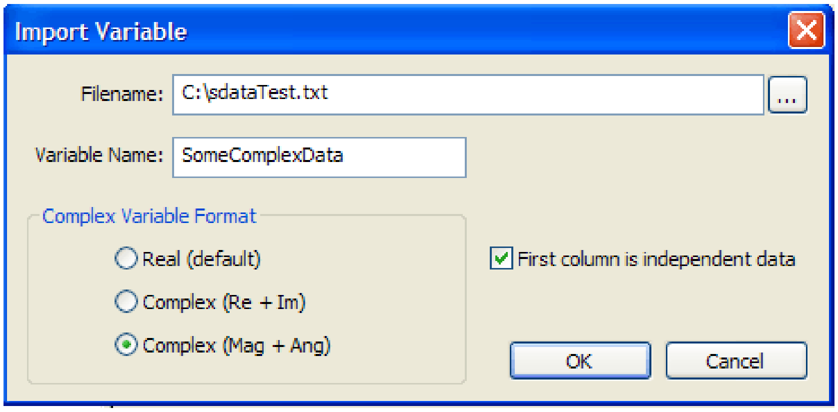
MHz:dBm	Response[1]	Response[2]
100	1	1.5
200	2	3
300	2.3	4
400	2.7	5.5
500	4.8	9
600	7.2	12

Importing Complex Variables

Complex data can be imported in several formats. A typical usage is shown below, where the independent vector is frequency (MHz) and the dependent is S21 in DB and ANG format. The same conventions apply here as for reals; spaces, tabs, and semicolons define breaks between entries.



```
! Eagleware GENESYS 2004
! complexExportTesting:Table1    (Filter1_Analysis.Filter1_Schematic)
! March 27, 2006     09:07:57
!
!_Freq_(MHz) DB[S21]       ANG[S21]
!Units MHz dB
865          -21.63        -59.213
865.25       -21.097       -63.604
865.5        -20.572       -68.062
865.75       -20.056       -72.584
866          -19.549       -77.167
866.25       -19.053       -81.805
866.5        -18.567       -86.494
866.75       -18.091       -91.228
867          -17.627       -96.002
```



Filename: C:\sdataTest.txt

Variable Name: SomeComplexData

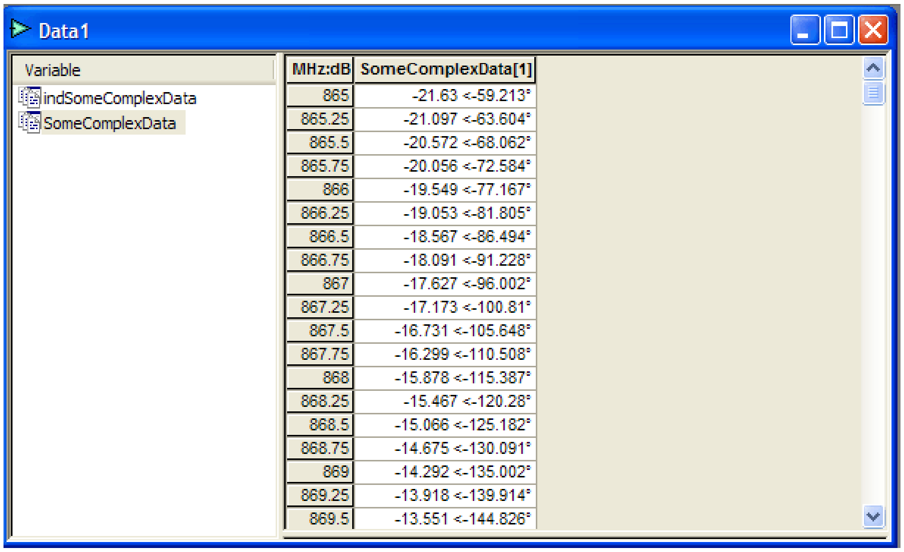
Complex Variable Format

○ Real (default)

○ Complex (Re + Im)

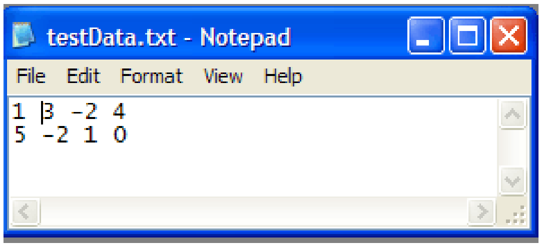
● Complex (Mag + Ang)

☑ First column is independent data

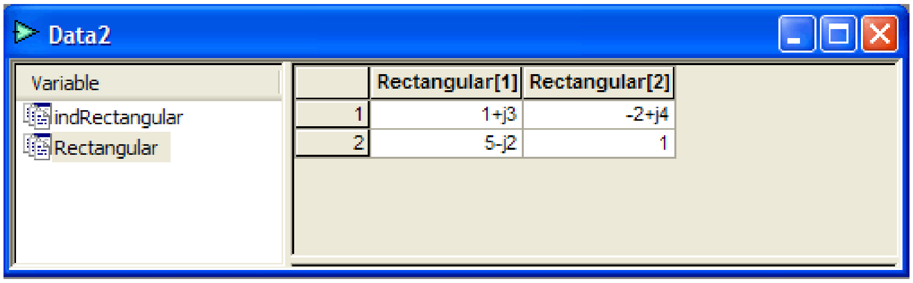


MHz:dB	SomeComplexData[1]
865	-21.63 <-59.213°
865.25	-21.097 <-63.604°
865.5	-20.572 <-68.062°
865.75	-20.056 <-72.584°
866	-19.549 <-77.167°
866.25	-19.053 <-81.805°
866.5	-18.567 <-86.494°
866.75	-18.091 <-91.228°
867	-17.627 <-96.002°
867.25	-17.173 <-100.81°
867.5	-16.731 <-105.648°
867.75	-16.299 <-110.508°
868	-15.878 <-115.387°
868.25	-15.467 <-120.28°
868.5	-15.066 <-125.182°
868.75	-14.675 <-130.091°
869	-14.292 <-135.002°
869.25	-13.918 <-139.914°
869.5	-13.551 <-144.826°

Example using rectangular coordinates (Re + Im)



```
1 3 -2 4
5 -2 1 0
```



	Rectangular[1]	Rectangular[2]
1	1+j3	-2+j4
2	5-j2	1

Notes

1. Complex data should come in pairs of columns; two parts are needed to specify a point in the 1D complex space. A warning is given if there is an odd number of columns (excluding the independent vector).
2. To use the dB scale for complex numbers, the unit should be specified as dB; otherwise the absolute scale is used based on whatever unit is defined. For example, input impedance should have a unit of "Ohm" which can also potentially have a phase; thus it cannot be in Ohms and dB simultaneously.
3. Typical units: dB, dBm, dB10, dB20, Abs, Ohms, V, A, mil, pF, nH
4. The independent variable must be real (this will typically correspond to time or frequency, both of which are real quantities).

Using Dataset Variables

You can create variables and analyses will create variables when they run.

To graph a Dataset variable

- Right-click the variable and see *creating a graph from a dataset* (users).

To duplicate a Dataset variable

- Right-click the variable and select Duplicate

To edit a Dataset variable

- You can not edit Measurement variables (variables created during a simulation run). You can edit variables you create. Double-click the variable or right-click it and select Properties from the menu.

To delete a Dataset variable

- You should not delete Measurement variables (variables created during a simulation run). You may delete variables you create. Right-click it and select Delete from the menu.

To view a Dataset variable

- If the variable is an array, click it and the right pane will fill with the array values. If the variable is a scalar the value should be shown in the list on the left.

To export a Dataset variable

- Right-click the variable and select Export. This will export it into an XML data form.

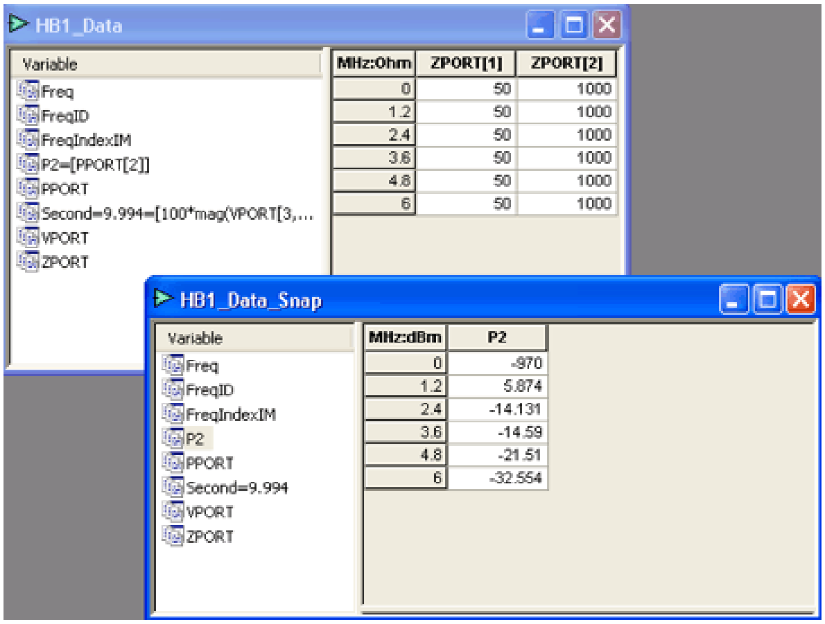
Using Datasets

Datasets are extremely useful for comparing different circuit configurations. You can run a simulation, save the data, then change some parameters, rerun the simulation and compare the two sets of data easily.

Normally, an analysis has the dataset name stored within it. You might set that name to a formula based on the parameters, but it's simpler to just Snapshot or Checkpoint the dataset.

To Checkpoint a Dataset

Right-click the dataset and select **Snapshot**. Another dataset named mydata_Snap is created. This Snap dataset contains the numerical data from the first dataset (all formulas are parsed and converted to data and the formula text is stored in the variable description).



HB1_Data

Variable
Freq
FreqID
FreqIndexIM
P2=[PPORT[2]]
PPORT
Second=9.994=[100*mag(VPORT[3,...
VPORT
ZPORT

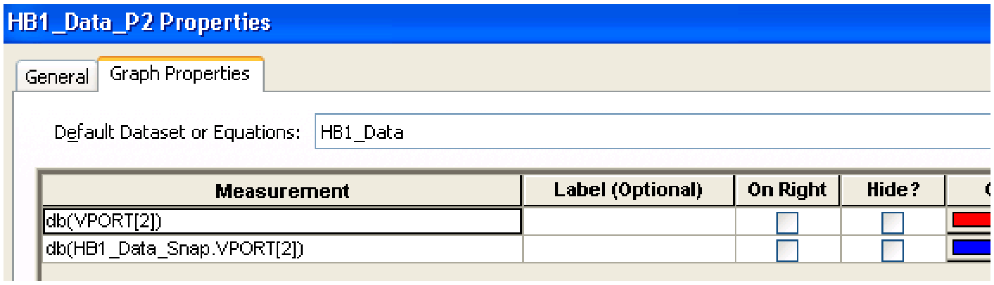
MHz:Ohm	ZPORT[1]	ZPORT[2]
0	50	1000
1.2	50	1000
2.4	50	1000
3.6	50	1000
4.8	50	1000
6	50	1000

HB1_Data_Snap

Variable
Freq
FreqID
FreqIndexIM
P2
PPORT
Second=9.994
VPORT
ZPORT

MHz:dBm	P2
0	-970
1.2	5.874
2.4	-14.131
3.6	-14.59
4.8	-21.51
6	-32.554

To compare PPORT[2] for the two datasets just enter two measurements in a graph or table like this:



HB1_Data_P2 Properties

Default Dataset or Equations: HB1_Data

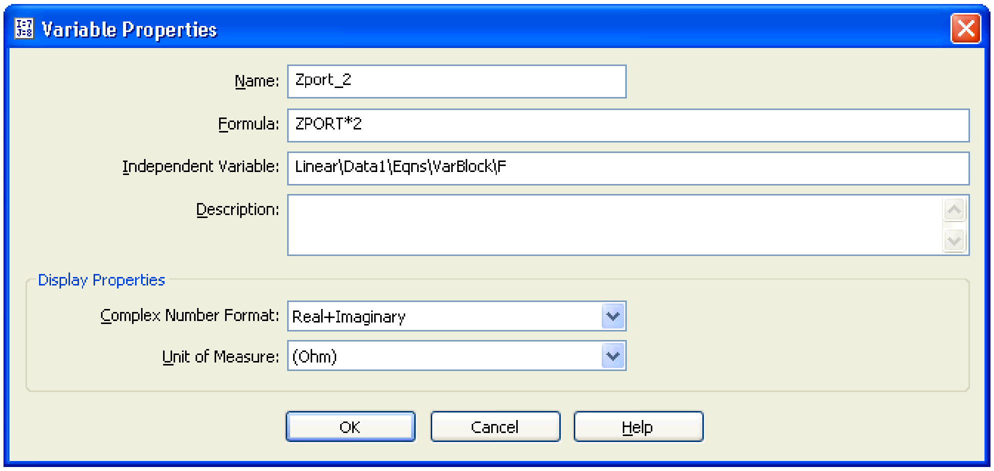
Measurement	Label (Optional)	On Right	Hide?	
db(VPORT[2])		□	□	🟥
db(HB1_Data_Snap.VPORT[2])		□	□	🟦

The HB1_Data_Snap.VPORT entry says to use the VPORT variable from HB1_Data_Snap.

Note we use db() here because the data in the dataset is in MKS and we want dBV for display.

Variable Properties

This window defines a variable and its display properties:



Name: Zport_2
Formula: ZPORT*2
Independent Variable: Linear\Data1\Eqns\VarBlock\F
Description:
Complex Number Format: Real+Imaginary
Unit of Measure: (Ohm)

- **Name** – The variable name.
  - The name must start with a letter and contain *only* letters, numbers, and/or the underscore "_" character.
  - Names are case-sensitive. (**V1** is a different variable than **v1**.)
- **Formula** – The equation which defines the variable's value.
  - The Engineering Language or Math Language equation may refer to other variables, functions, define vectors, matrices, etc. Please see the appropriate section in the *User's Guide* for details on using Equations.
- **Independant Variable** – One or more associated variables, which define related data, such as the X-Axis variable (which is the first indep).
  - If there is more than one independant variable, each indep should be separated by a vertical-bar character "|".
- **Description** – The discription of the variable, usage notes, etc.
- **Complext Format** – If the value is one or more complex numbers, the values can be displayed in several formats:
  - **Default** – Allows Genesys to automatically determine the most appropriate format to use.
  - **Real + Imaginary** – Displays the values using real and imaginary values.
  - **Magnitude + Angle** – Displays the values using magnitude and angle. The magnitude is displayed using the units specified in the "Display Magnitude In" dropdown. The angle is displayed using the global Angle units, as specified in Tools / Options.
  - **Magnitude Only** – Only the magnitude is displayed.
- **Units Of Measure** – The units used for displaying the variable in the DataSet view window.
- **Display Magnitude In** – If the value is a complex number and the Complex Format includes magnitude, this specifies the units to use (for magnitude).

Graphs

Graphs display data from *datasets* (users) or *equations* (users), which are usually measurement data derived from the analysis of a design. For more information on menu items, refer *Graph Menu* (users) or *Graph Toolbar* (users) in the Appendix sections.

Contents

Types of Graphs

Genesys has several types of graphs, including:

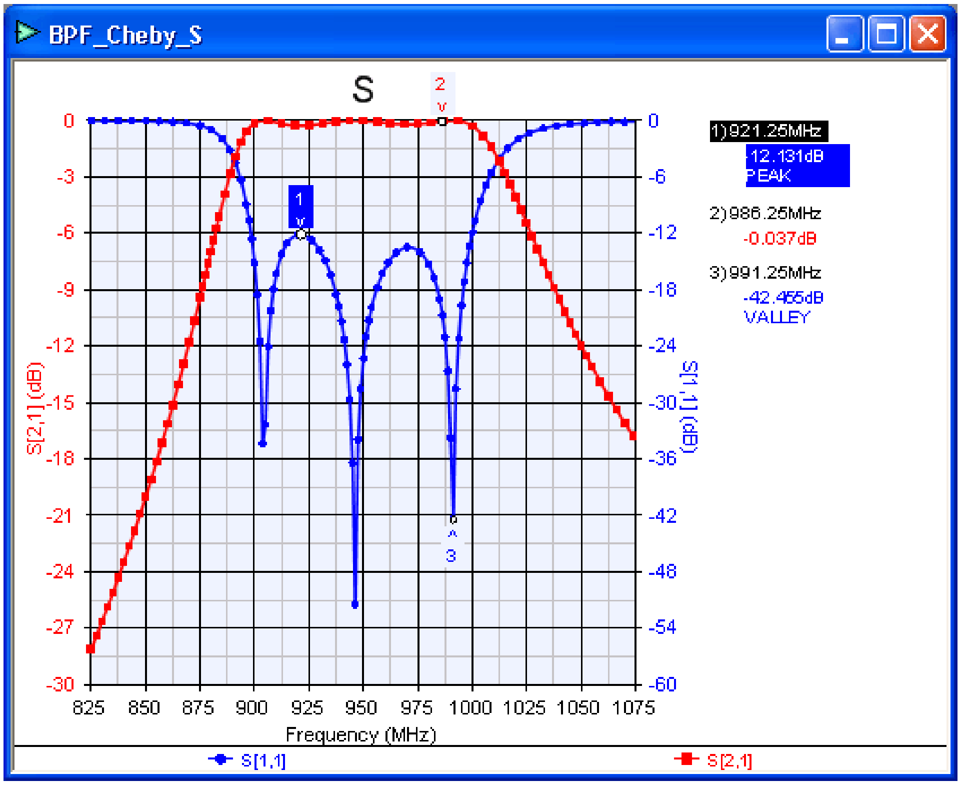
- **Rectangular Graphs** - a Cartesian coordinate plot.
- **3D Graphs** - can display a measurement versus frequency versus a parameter sweep.
- **Antenna Plots** - displays Far Field measurements.
- **Polar Charts** - displays complex data, such as S-Parameters or impedances.
- **Smith Charts** - similar to a Polar Chart, displays impedance and/or admittance.

In addition, data can be displayed in a spreadsheet-style ***Table*** (users) view. These differing output options allow you to display data in a variety of formats.

Rectangular Graphs

A rectangular graph is a Cartesian coordinate plot. You can use a rectangular graph to display two-dimensional data versus frequency (for example: magnitude or phase of a complex measurement, but not both).

In the figure below, the S-parameter insertion loss and return loss of a bandpass filter are plotted. There are 3 types of markers shown: a peak marker, a regular (fixed frequency) marker, and a valley marker. You can add to any rectangular graph. Regular markers can also be placed on circular charts.



1)921.25MHz
-12.131dB
PEAK

2)986.25MHz
-0.037dB

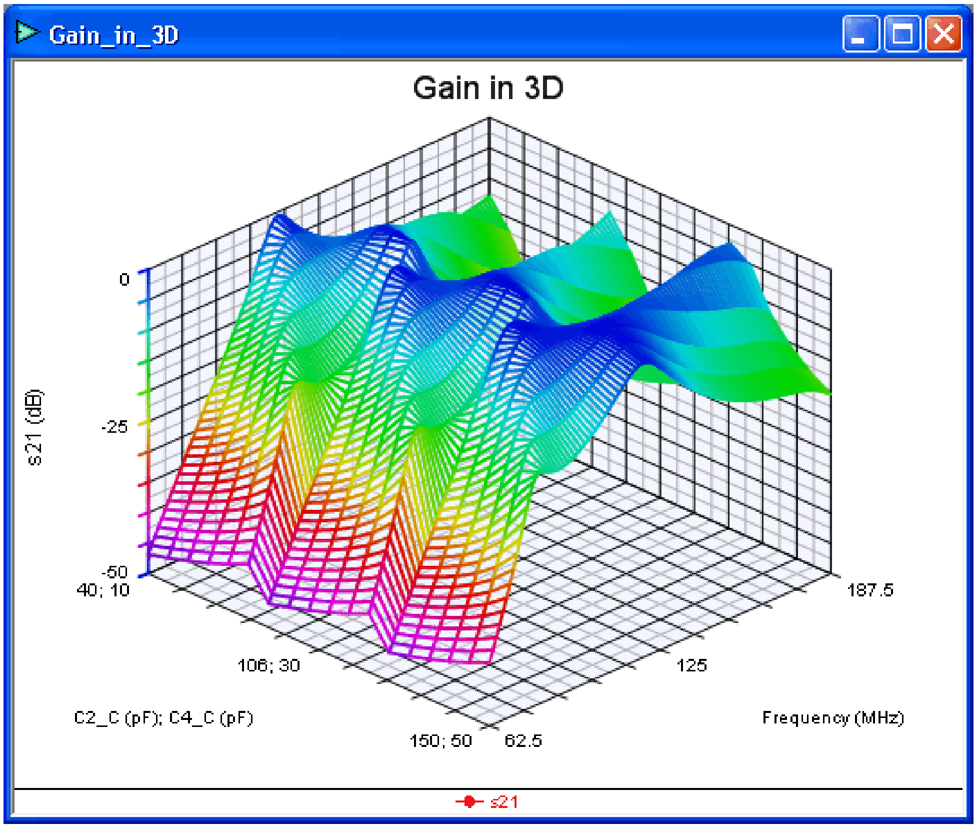
3)991.25MHz
-42.455dB
VALLEY

If you do not like the small circles, squares, or triangles that show each data point, hide them using the **Show Symbols On Trace** option on the Graph menu.

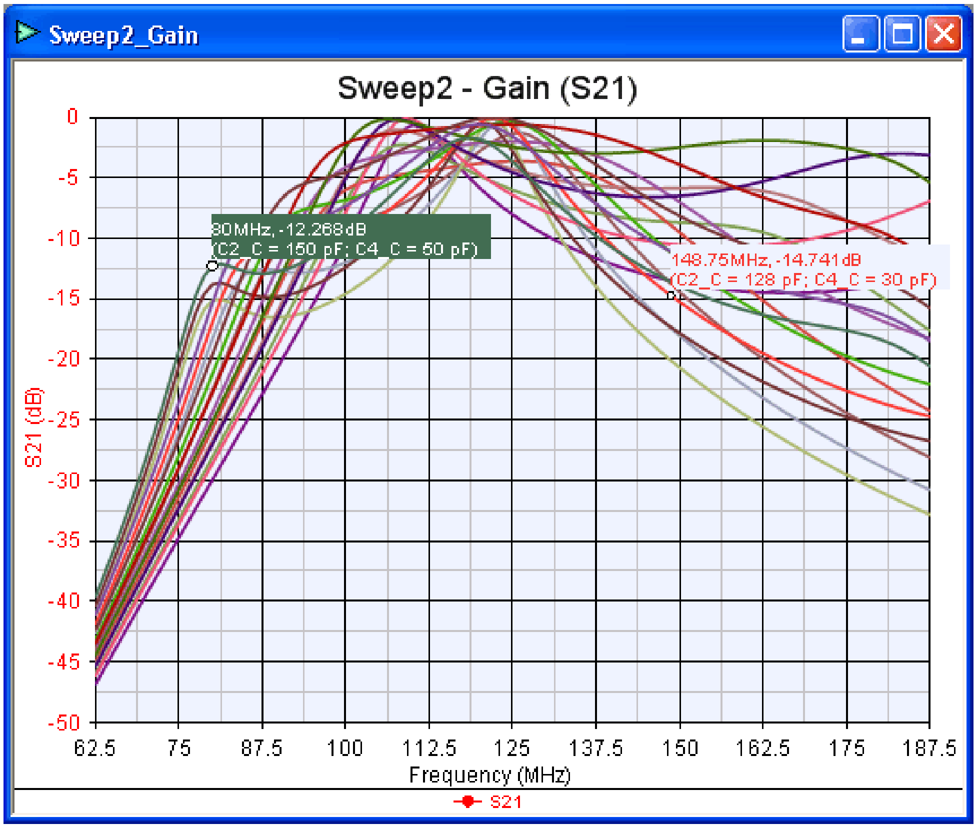
3D Graphs

A 3D graph is used to display a measurement versus frequency versus a parameter sweep. You can sweep any tunable parameter or variable. In the figure below, the gain of a 5th-order Butterworth bandpass filter is displayed from 62.5 to 87.5 MHz while 2 capacitors are varied (swept).

A 3D graph *requires* a parameter sweep to generate three-dimensional data.

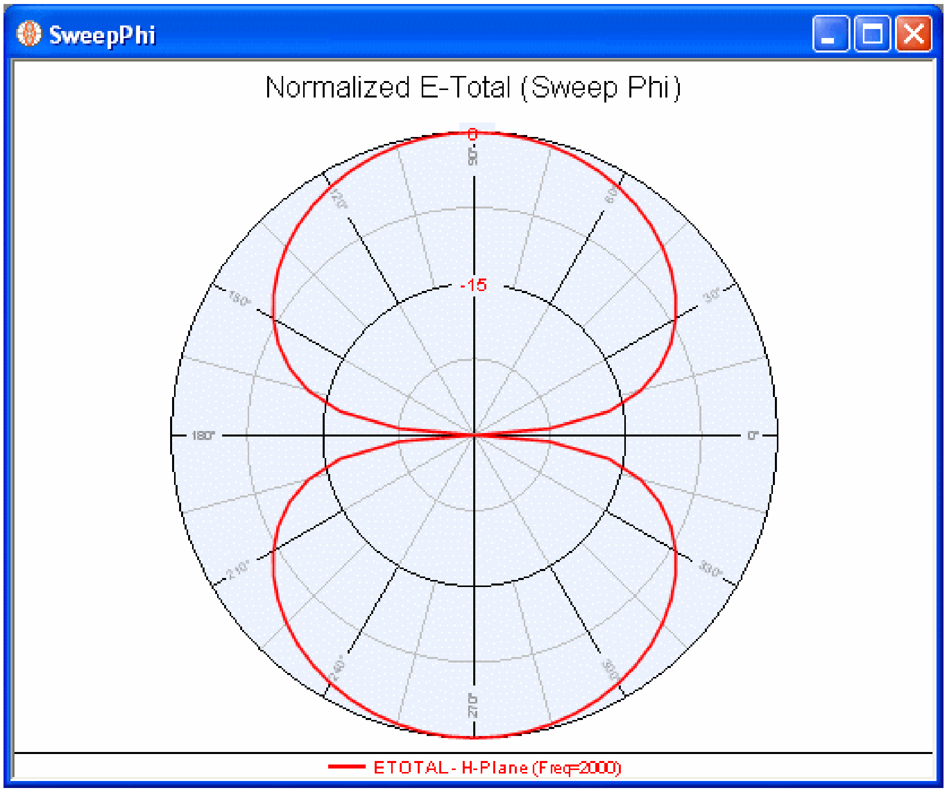


3D graphs cannot display markers. However, you can plot a sweep on a rectangular graph if you need markers. Here's the same sweep:



Antenna Plots

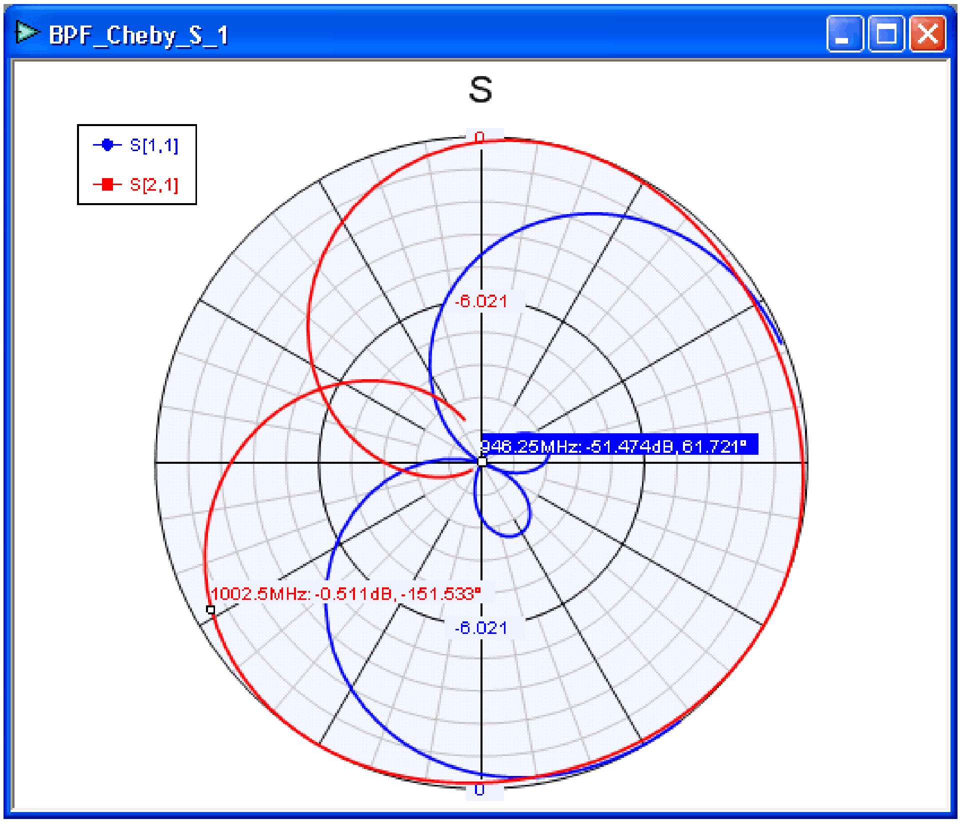
Antenna plots display two dimensional far field radiation patterns. The electromagnetic analysis must be setup to gather this radiation data. See the Empower Viewer and Antenna Patterns section of the Empower Simulation manual for information on setting up the Empower analysis and the Radiation Patterns and Antenna Characteristics section of the Momentum GX manual. The measurement wizard can be used to slice three dimensional data for plotting on two dimensional graphs.



Polar Charts

A polar chart is used to display complex data, such as S-Parameters or impedances. In the figure below, S11 (input reflection coefficient) and S22 (output reflection coefficient) are plotted. The horizontal axis on a polar chart represents purely real numbers, while the vertical axis represents purely imaginary numbers. Numbers that lie between the two axes have both imaginary and real components.

Smith charts and polar charts generate the same plots for S-parameters (only the background and scales are changed). Additionally, certain measurements (such as Y Parameters) may be plotted on polar charts and not on Smith charts (where those measurements don't really make sense).

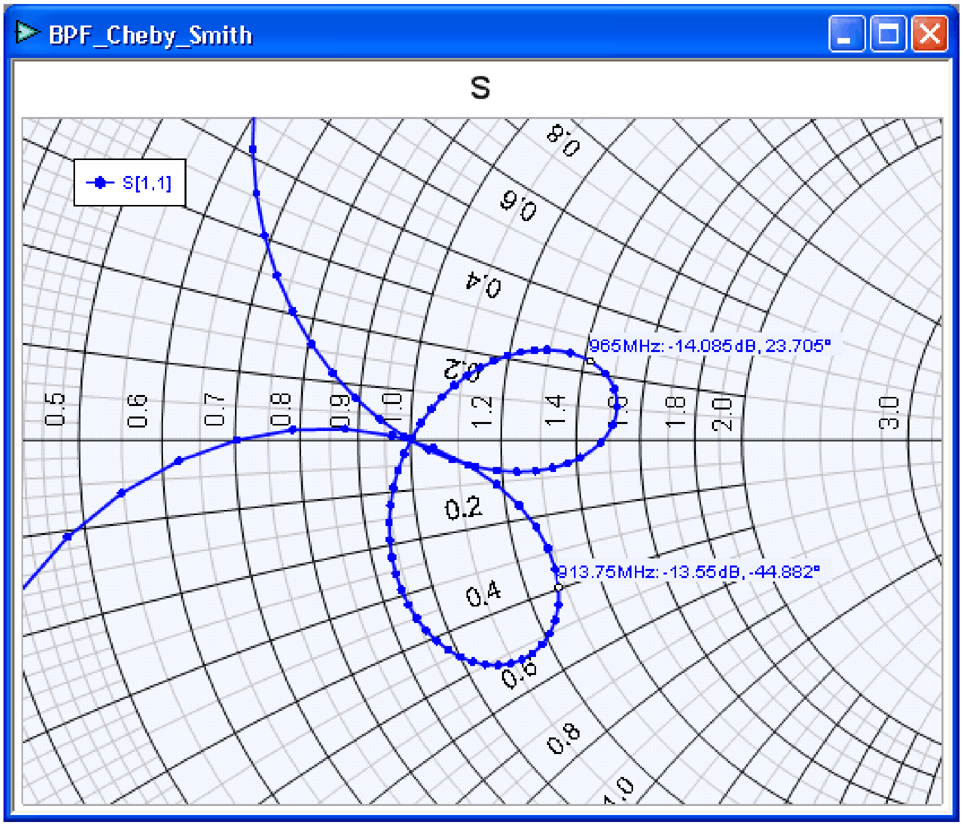


Smith Charts

A Smith chart is used to display complex data, such as S-Parameters or impedance. In the figure below, S11 (input reflection coefficient) is plotted on a Smith chart. The horizontal axis on a Smith chart represents real numbers from zero (0) to infinity, and numbers off the horizontal axis represent numbers having a nonzero imaginary part.Smith charts in Genesys have two grid options:

- Impedance
- Admittance

You can enable or disable both grids using the Smith Chart Properties window.



Smith charts can plot special circular measurements, like gain, noise, and stability circles. When displaying one of these measurements, place one or more makers to set a locus for the measurement circles.

Creating Graphs

Graphs can be created manually, however the easier way to provide a context first. See sections on creating a graph from a dataset or creating a graph from a schematic.

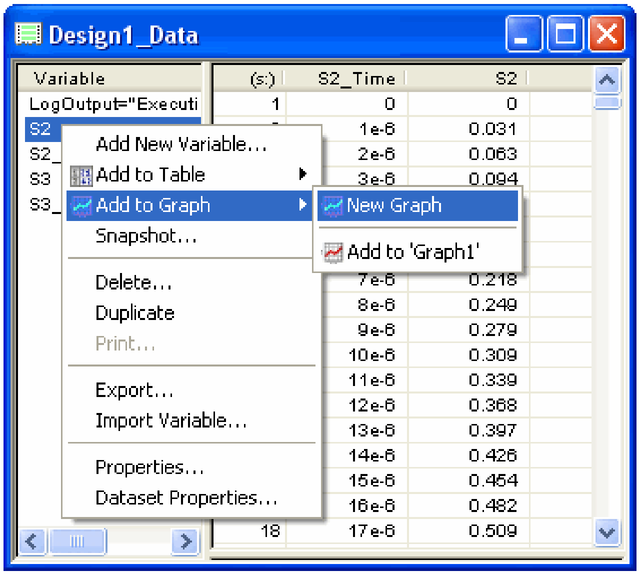
The easiest way to add an arbitrary measurement to an existing graph is via the *Measurement Wizard* (users).

Manually create a graph

1. Click the New Item button ( ) on the Workspace Tree toolbar.
2. Select Graphs.
3. Click on the graph type, e.g click on menu item *Add Rectangular Graph...*
4. A property page for that graph type will pop-up.
5. Click on the Graph Properties tab.
6. Add the measurement you want to plot.
7. Click **OK**.

Create a graph from a dataset

Right click a variable in a dataset. Select **Add Graph...** and click on **New Graph**.

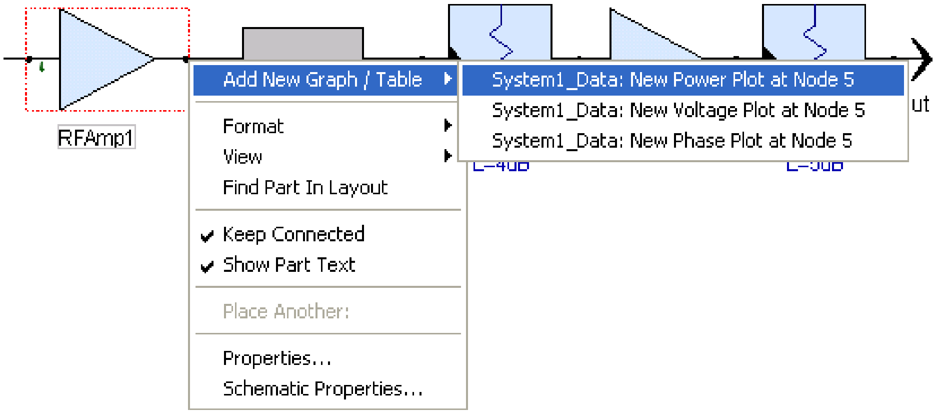


(s:)	S2_Time	S2
1	0	0
	1e-6	0.031
	2e-6	0.063
	3e-6	0.094
	7e-6	0.218
	8e-6	0.249
	9e-6	0.279
	10e-6	0.309
	11e-6	0.339
	12e-6	0.368
	13e-6	0.397
	14e-6	0.426
	15e-6	0.454
	16e-6	0.482
18	17e-6	0.509

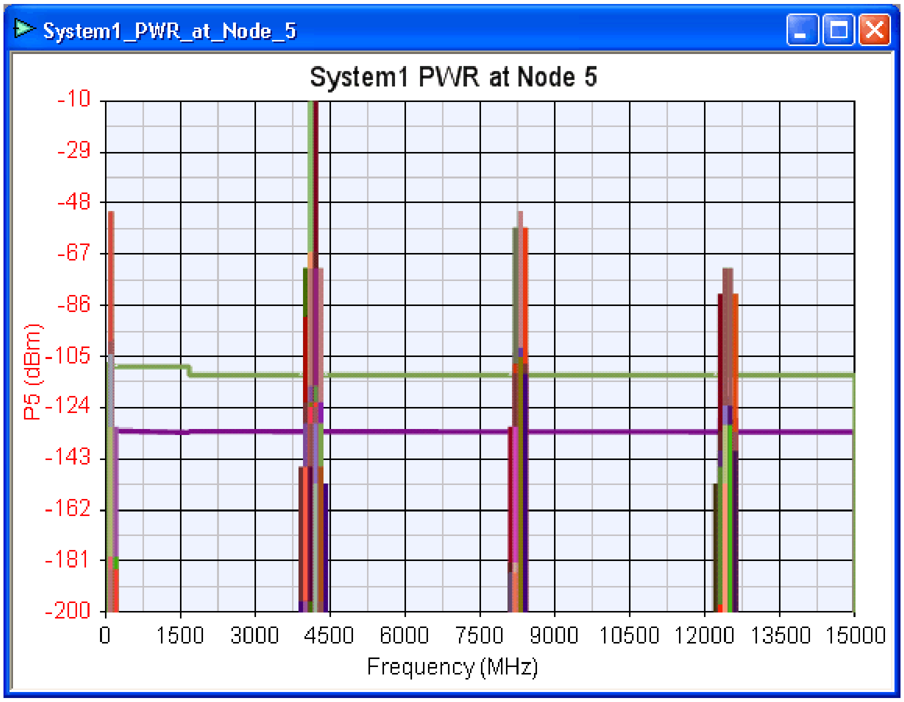
Create a graph from a schematic

1. Right-click a port or node on a schematic and select **Add New Graph/Table** then the measurement you want to graph from the menu.

> The actual items available on the menu are context-sensitive, based on the part or node you clicked and the simulations available. For example, the Relevant S-Parameters option generates measurements for all S-parameter measurements that are pertinent to the indicated port. Also, the workspace must contain at least one analysis referring to this schematic design to make this feature available. (Otherwise there is nothing to plot.)



2. To create another graph, right click the port again and select a different option. Your screen should now have a spectrum similar to this:



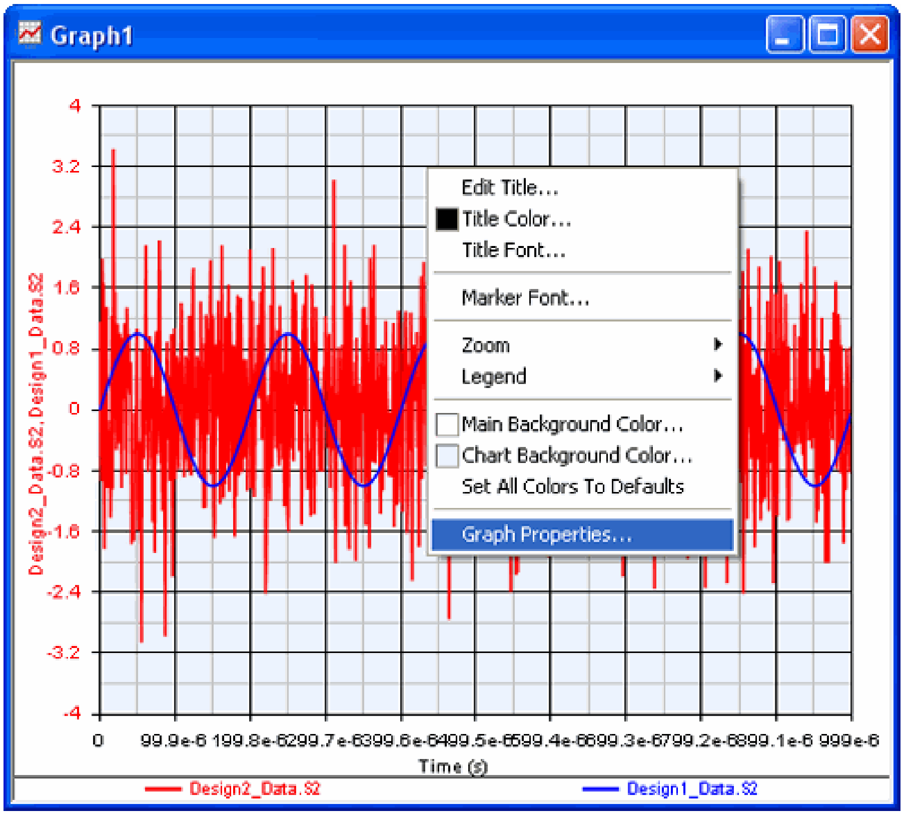
3. Double-click a graph to change the graph's properties. Right-click a trace or legend to make specific changes to the appearance of the trace or legend. Hover over a symbol (a dot on the trace) to get a pop-up showing the value at that point. Check out the Graphs tutorial video for tips and techniques.

Graph Properties

Graph properties define a graph object. The **Graph Properties** window permits changes to properties such as the title or a series, i.e. a plot of a measurement variable.

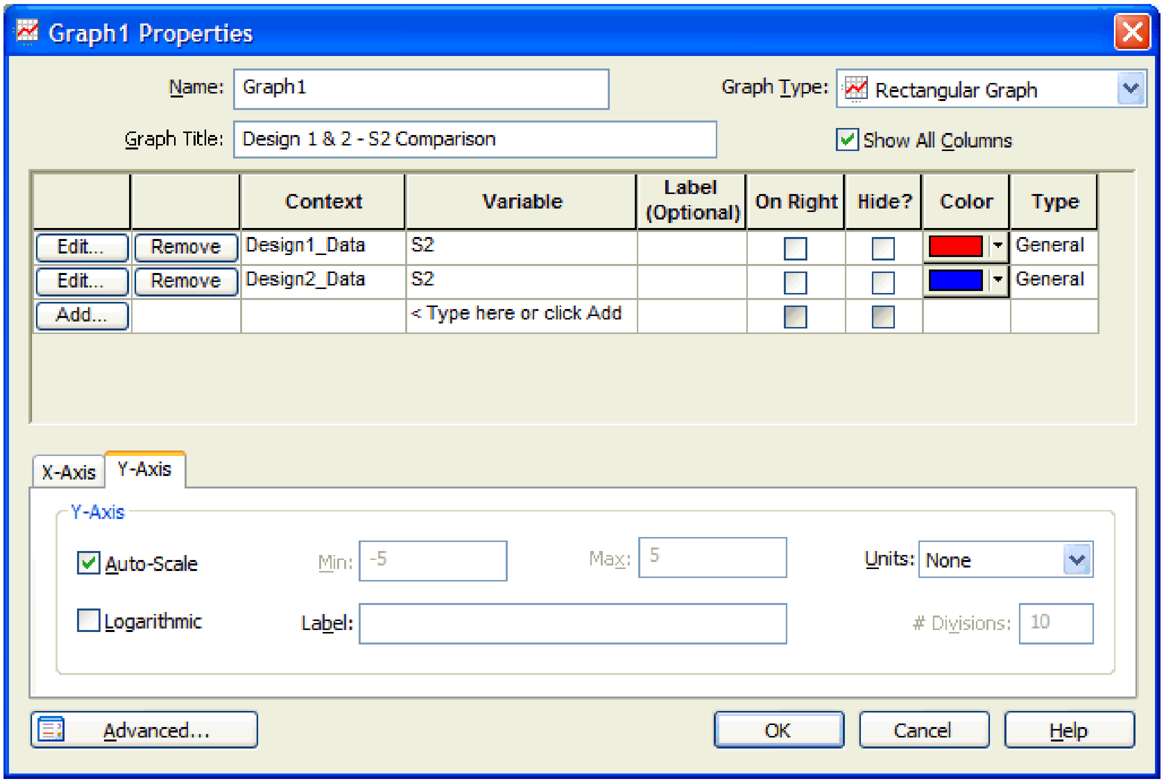
Changing Graph Properties

The Graph Properties window initially appears when a graph is created, so that you can add a series and/or customize the graph. You can make additional changes after the graph is created by right clicking the graph window or double clicking an "empty" area of the graph window and then selecting **Graph Properties...** as illustrated below.



Graph Properties Dialog

The following **Graph Properties** window was created for a *General* plot type with a *Rectangular* graph format and plots two variables from two different datasets.


Graph1 Properties

Name: Graph1

Graph Type: Rectangular Graph

Graph Title: Design 1 & 2 - S2 Comparison

☑ Show All Columns

		Context	Variable	Label (Optional)	On Right	Hide?	Color	Type
Edit...	Remove	Design1_Data	S2		□	□	🟥	General
Edit...	Remove	Design2_Data	S2		□	□	🟦	General
Add...			< Type here or click Add		□	□		

X-Axis | Y-Axis

Y-Axis

☑ Auto-Scale Min: -5 Max: 5 Units: None

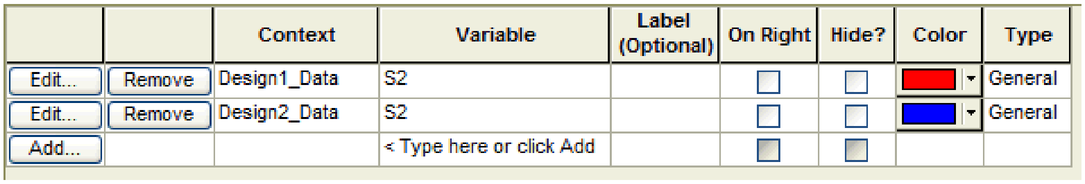
□ Logarithmic Label: # Divisions: 10

- **Name** – The name of the graph object, which is shown on the workspace tree
- **Graph Title** – The plot title, which is drawn at the top of the graph (like a heading)

- **Show All Columns** – When this box is checked, infrequently-used columns in the **Series** window (such as **On Right** and **Hide?**) are shown.
- **Advanced... Button** – Clicking this button displays the **Advanced Graph Properties** dialog, as described below.

Series Settings

The following series window has defined two series for plotting.



		Context	Variable	Label (Optional)	On Right	Hide?	Color	Type
Edit...	Remove	Design1_Data	S2		□	□	🟥 ▾	General
Edit...	Remove	Design2_Data	S2		□	□	🟦 ▾	General
Add...			< Type here or click Add		□	□		

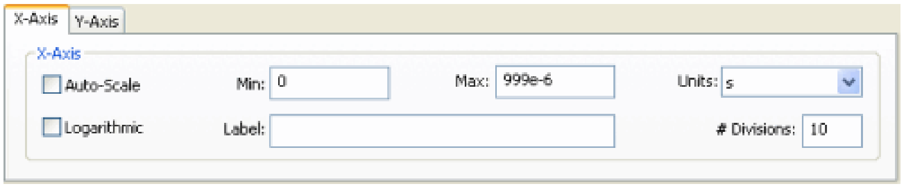
- **Edit/Add Button** – Clicking on the **Edit** or **Add** buttons will pop-up the *Graph Series Wizard* (users) for a series definition.
- **Remove Button** – Clicking on this button removes the associated series.
- **Context** – The text provides context for the **Variable** text box. If left blank, the **Variable** text box must have a fully qualified variable name. Typically, the context is the **dataset name** where the variable is defined. To graph an equation variable, set this text box to **[Equations]**. The equation hierarchy is searched for the equation variable. If the equation variable is not found, an error is logged.
- **Variable** – The text contains the name of the variable that is to be graphed.
- **Label (Optional)** – The text contains the axis label for the series. If left blank, the **Variable** text is used.
- **On Right / On Bottom** – If the box is checked, the an alternate vertical axis for the series is placed on the right side of a rectangular graph. **Polar** charts use On Bottom to indicate the use of the "lower" radial axis.
- **Hide?** – If the box is checked, the series is not plotted.
- **Color Button** – Click on this button to change the color that has been assigned to the series.
- **Type** – This informational (read only) text box states the series plot type.

> **Note**: Checkpoint traces are NOT shown in the series grid. You can remove all the checkpoint traces on a graph by clicking the Checkpoint button on the *Graph Toolbar* (users). You can change the trace color by right-clicking a trace.

Axis Settings Tabs

The lower portion of the window contains various axis and settings tabs, which depend on the graph type.

The following is an example of a **rectangular graph** with a single vertical axis.



□ Auto-Scale Min: 0 Max: 999e-6 Units: s

□ Logarithmic Label: # Divisions: 10

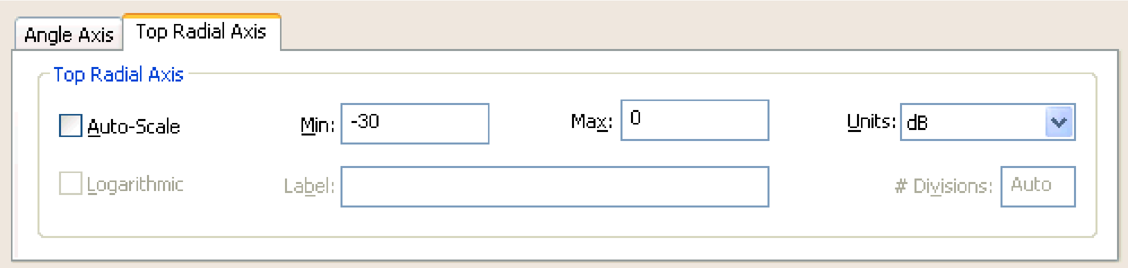
If both vertical axes are used, the **Y-Axis** tab name is changed to **Left Y-Axis**, and an additional tab labeled **Right Y-Axis** is added. Most of the settings are similar.

- **Auto-Scale** – When checked, the axis automatically sets its limits to match the

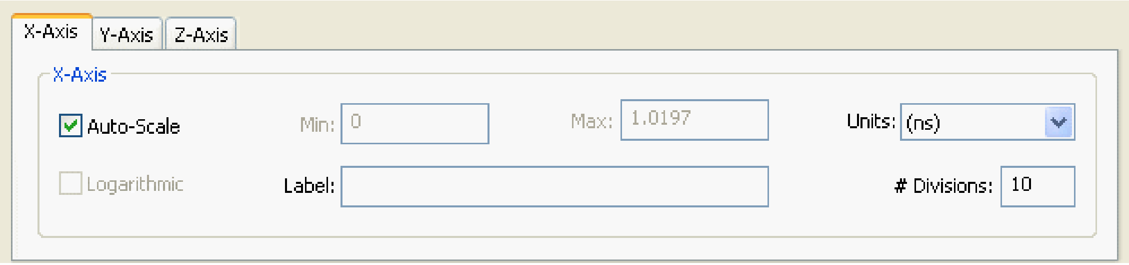
range of the data which is being plotted
- **Label** – Use this label to customize the axis name
- **Logarithmic** – When checked, the axis is drawn with a logarithmic scale
- **Min** – Sets the lower numerical range of an axis
- **Max** – Sets the upper numerical range of an axis
- **Units** – Sets the units-of-measure used by the axis (and Min and Max)
- **# Divisions** – Sets the number of divisions to use on the axis; contains **Auto** if the divisions will be determined automatically

Tip: To control the axis tick marks, you can set the **Min** and **Max** fields to appropriate numbers, e.g. in the above examples you might want to specify the **Max** to 1000e-6 s.

Similarly, an Antenna plot has **Angle Axis** and **Top Radial Axis**, and a 3D chart has **X, Y,** and **Z-Axis**, but the individual settings are otherwise identical.

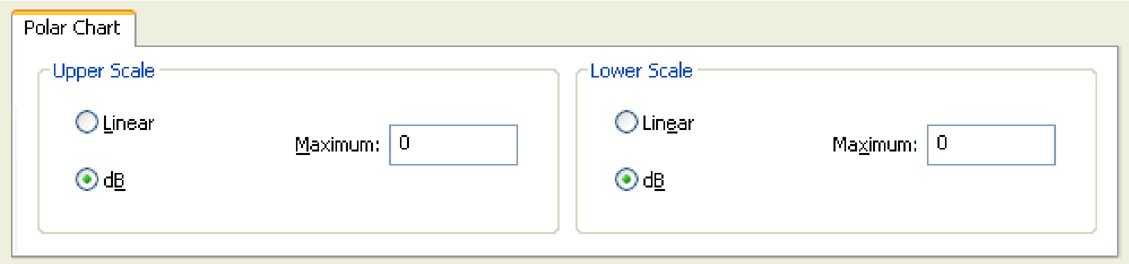


□ Auto-Scale
Min: -30
Max: 0
Units: dB
□ Logarithmic
Label:
Divisions: Auto



☑ Auto-Scale
Min: 0
Max: 1.0197
Units: (ns)
□ Logarithmic
Label:
Divisions: 10

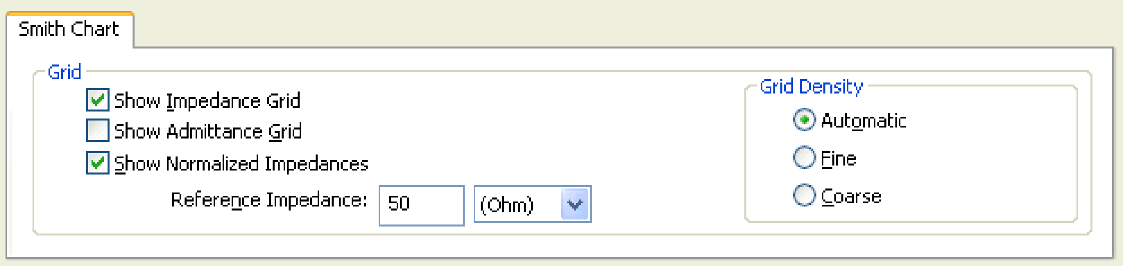
For a polar plot, there is simply a tab labeled **Polar**.



Upper Scale: ○ Linear ● dB Maximum: 0
Lower Scale: ○ Linear ● dB Maximum: 0

- **Upper and Lower Scale** – Polar charts have both an upper and lower scale, so that different numerical ranges may be compared on the same plot.
- **Linear or dB** – Indicates which scaling method to use
- **Maximum** – Typically 0.0 for dB and 1.0 for Linear

Likewise, a Smith chart has a **Smith** tab, with appropriate settings.



Grid

☑ Show Impedance Grid

□ Show Admittance Grid

☑ Show Normalized Impedances

Reference Impedance: 50 (Ohm)

Grid Density

● Automatic

○ Fine

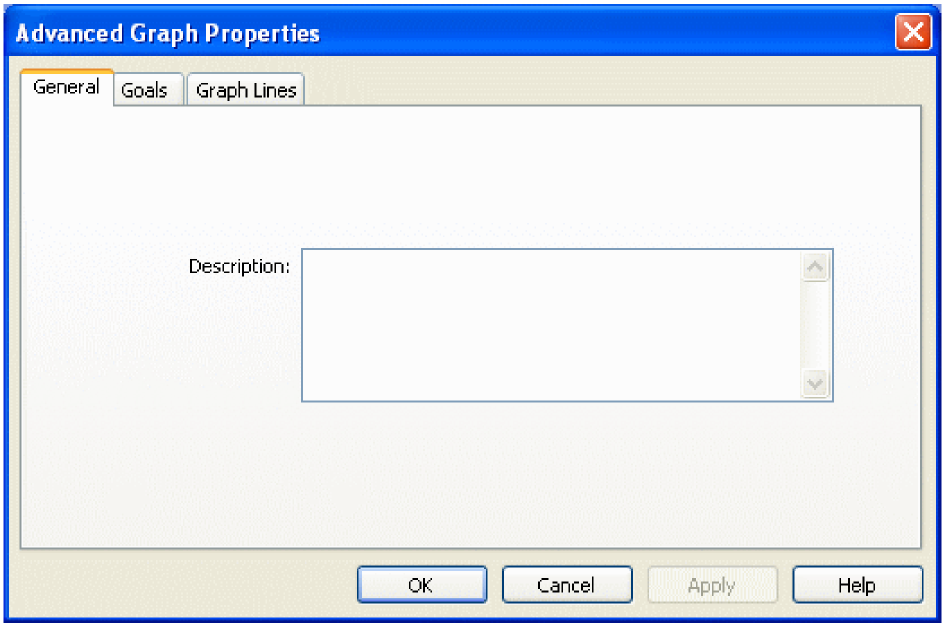
○ Coarse

- **Show Impedance Grid** – When checked, this option enables the impedance background "graph paper" (default ON, drawn in black)
- **Show Admittance Grid** – Enables the admittance background "graph paper" (default OFF, drawn in dark red)
- **Show Normalized Impedances** – Normally checked: The graph will use normalized impedances (range will be converted from 0.0 - Reference Impedance to 0.0 - 1.0); when unchecked, the graph will be drawn using the full numerical range of the data
- **Reference Impedance** – Allows user to change the labels shown on the Smith chart if desired. Note that the Reference Impedance entry is only active if the "show normalized impedances" box is NOT checked. The default is 50 Ohms, although 75 often used for certain applications (like cable TV).
- **Grid Density** – Sets grid spacing
  - **Automatic** – Determines appropriate grid spacing (ultra fine to very coarse) and optionally draws text labels
  - **Fine** – Has text labels and uses fine grid spacing
  - **Coarse** – No text labels with widely spaced grid

Note: The **Reference Impedance** entry does not affect the position of traces on the Smith chart – it only affects the labels (for user convenience in reading the chart). For instance, it is possible to have a schematic with 2 different ports, port1 having Zo=50 and port2 having Zo=100. Then S11 and S22 could simultaneously be displayed on a Smith chart. The center value of the Smith chart is dependent on the port impedance from which a given measurement is derived.

General Tab

The General tab contains generalized graph settings, such as a description field.

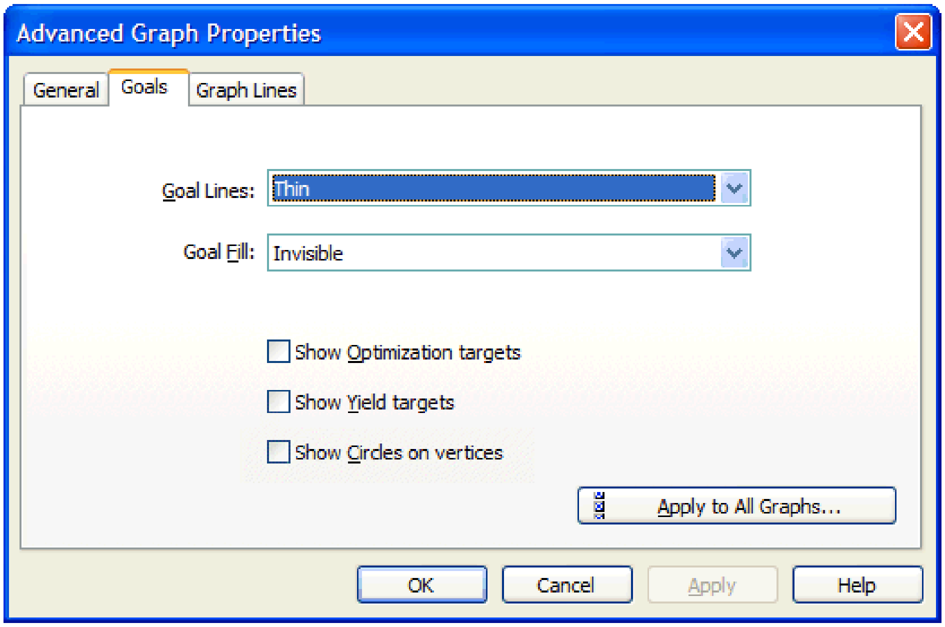


Description:

- **Description** – An optional description which is saved with your graph

Goals Tab

If the appropriate check boxes are filled, matching optimization or yield targets for all measurements can be displayed as a goal line or as a goal area.



Goal Lines: Thin

Goal Fill: Invisible

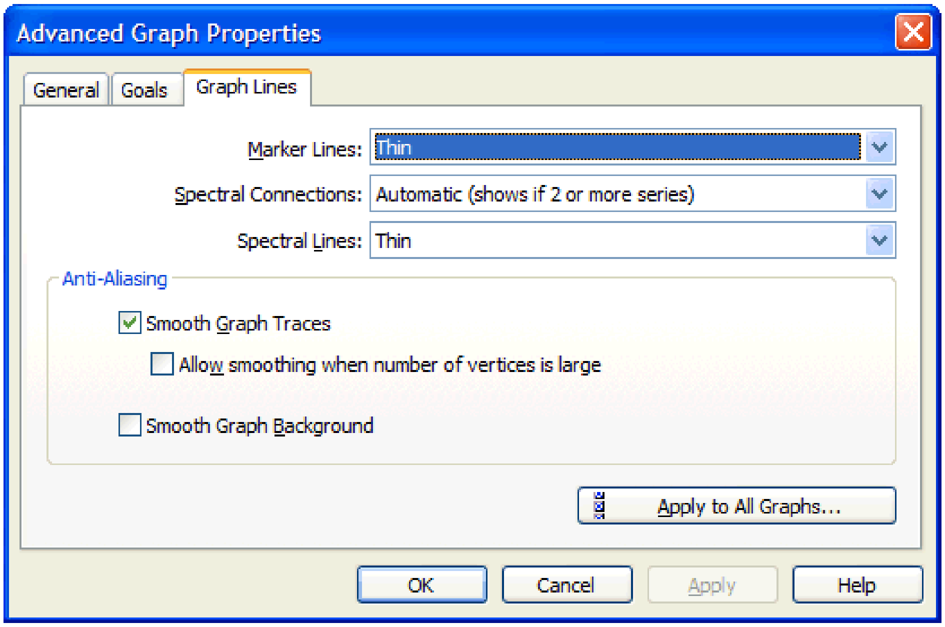
□ Show Optimization targets

□ Show Yield targets

□ Show Circles on vertices

- **Goal Lines** – Specifies the thickness of the goal line(s): Thin, Medium, Thick, Heavy, or None
- **Goal Fill** – Sets the transparency of the shaded goal area: Invisible, Faint, Semi-Transparent (Normal), or Heavy
- **Show Optimization targets** – Enables Optimization target drawing. (Shows targets set up in separate Optimization evaluation.)
- **Show Yield targets** – Enables Yield target drawing. (Shows targets set up in separate Yield evaluation.)
- **Show Circles on vertices** – This setting is only available for circular graphs (like Smith or Polar). When checked, this option draws circle(s) at every vertex of a circular measurement, such as SB1, SB2, etc.
- **Apply to All Graphs button** – Applies the current **Goals** tab settings to all the graphs in the current workspace.

Graph Lines Tab



Marker Lines: Thin

Spectral Connections: Automatic (shows if 2 or more series)

Spectral Lines: Thin

☑ Smooth Graph Traces

☐ Allow smoothing when number of vertices is large

☐ Smooth Graph Background

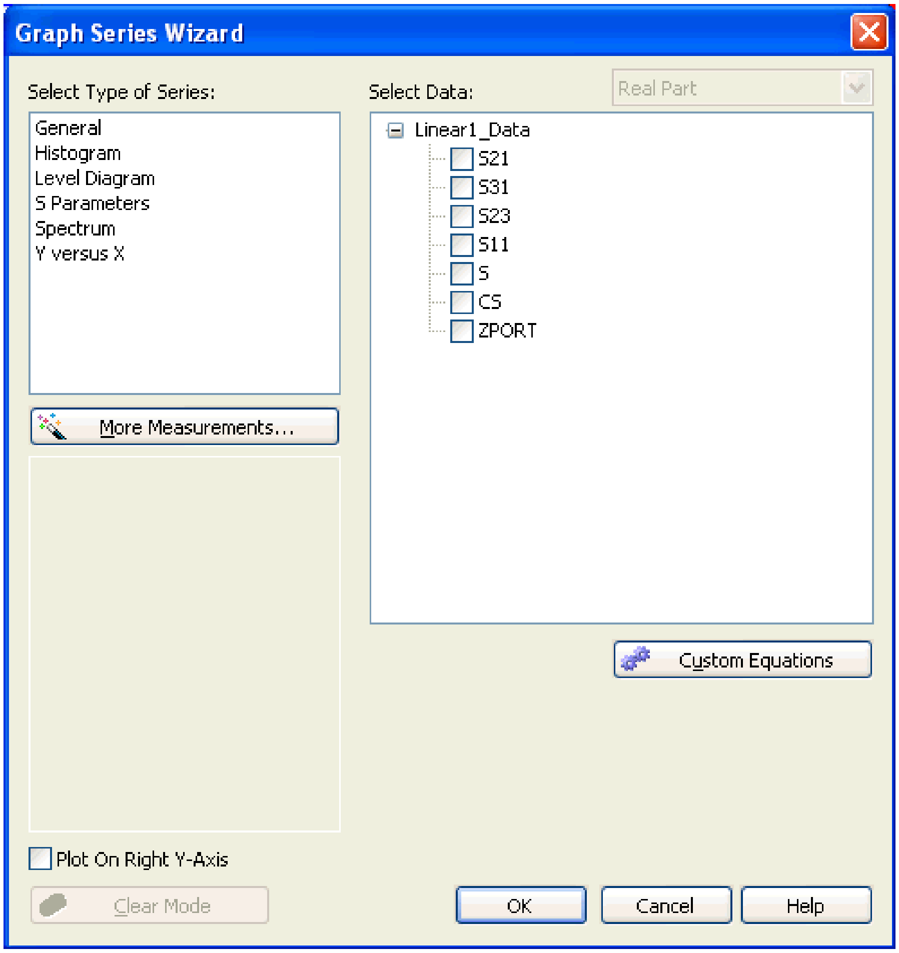
- **Marker Lines** – Sets the thickness of the graph traces: Thin, Medium, Thick, Heavy, or None
- **Spectral Connections** – When displaying more than one set of spectral data on a graph, it can be difficult to determine the heights of overlapping peaks. This feature connects the peaks with thin semi-transparent lines, so that spectral plots can be read more easily: Automatic, Always Hidden, or Always Visible
- **Spectral Lines** – Determines the thickness of spectral peak lines: Thin, Medium (Normal), Thick, or Extra Thick
- **Smooth Graph Traces** – By default, anti-aliasing techniques are used to remove jagged pixel edges, however by default, traces with a large number of vertices are not smoothed
- **Allow smoothing when number of vertices is large** – Also smooth traces that contain a lot of points
- **Smooth Graph Background** – Smooths the "graph paper" background; only available for circular charts
- **Apply to All Graphs button** – Applies the current **Graph Lines** tab settings to all the graphs in the current workspace.

OK, Cancel, Apply and Help Buttons

Clicking the **OK** button accepts the property changes and exits the dialog. Clicking the **Cancel** button dismisses any changes and exits the dialog. Clicking the **Apply** button temporarily accepts property changes for previewing. The **Help** button links to documentation.

Graph Series Wizard

This wizard initializes properties for a graph object. For a new graph, the wizard is invoked by adding a graph to the work space tree or by selecting a variable from a dataset and adding a new graph. For a created graph, clicking on the **Add** or **Edit** buttons in the ***Graph Properties*** (users) dialog will pop-up the **Graph Series Wizard**. The following shows the wizard when a graph object is added to the work space tree.



Note that a complete list of series (plot) types is available and that all dataset variables are ready for selection. This wizard state can be reached by clicking the **Clear Mode** button.

A specific series can be directly chosen from the list in the **Type of Series** window, and consequently the list of available dataset variables is refined. Conversely, a dataset variable can be chosen from the **Data** window, and the list of available plot types is refined.

Wizard Components

The components are described in top-down order.

Type of Series Window

Choosing a series type limits which dataset variables can be selected for the series. It also determines how many **Data** windows are displayed (1 or 2). Note the different series types will often share some of the same variables (the measurement sets may overlap).

The list of available series types follows.

- **General**. Variables are plotted against their domains. Every variable is compatible with this type.
- **Level Diagram**. A variable generated by **SpectraSys** (RF System Analysis) which is a measurement at components along a selected path is graphed as a function of its path position.

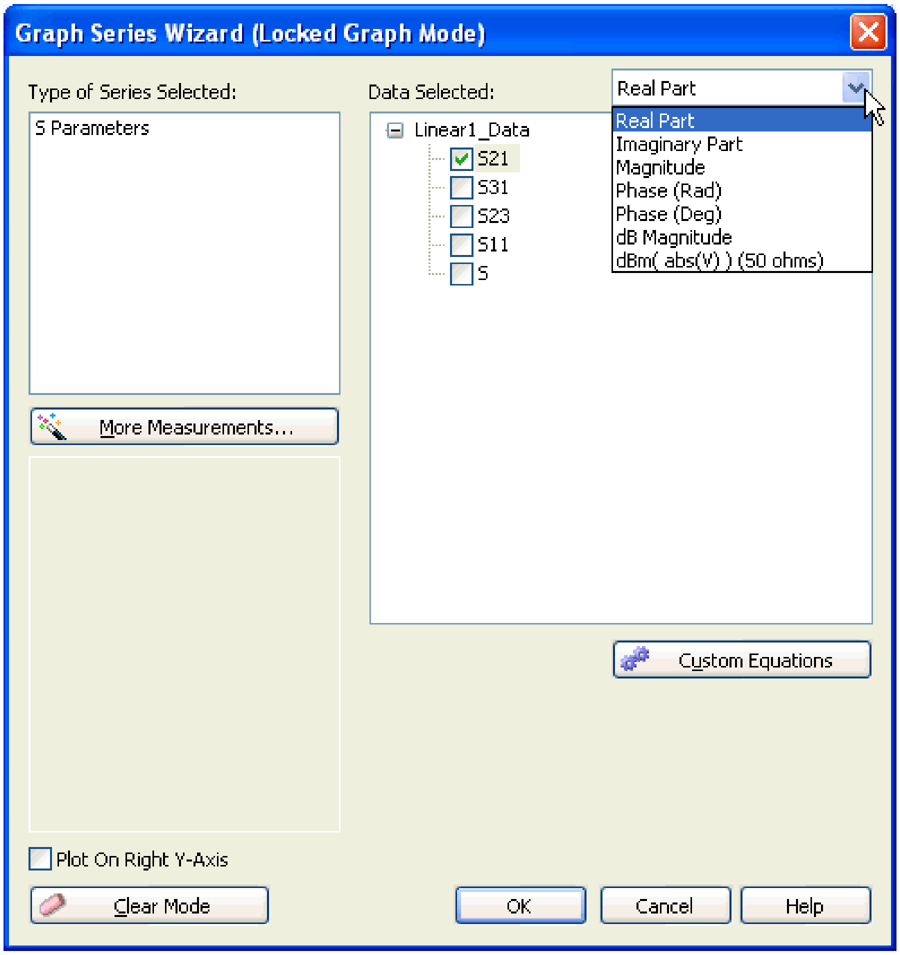
- **Spectrum**. Plot a variable whose independent axis is Frequency. This plot type is also compatible with variables whose independent is Time and will produce a post-processed set of equations which involve taking an FFT. Various options for the FFT are available - see the Post Processed equation block.

- **S Parameters**. A scattering parameter measurement generated by **SpectraSys** (RF System Analysis) is graphed as a function of its frequency domain.
- **Group Delay**. An input-output delay measurement of a component (usually a filter) is generated by the **Linear Analysis** or by **SpectraSys** (RF System Analysis) and is graphed as a function of its frequency domain.
- **Histogram**. The range of a real variable is binned. The number of samples in a bin is plotted against the bin value.

Data Window

Once a series type is selected, the possible dataset variables required for the plot are displayed in one or more **Data** windows. Alternatively, if a variable is chosen then a refined list of plot types is shown in the **Type of Series** window.

Note that an un-named pull-down menu that is located at the upper right of a **Data** window can be used to modify a selected variable. This pull-down menu is activated when there is a potential need to further specify the selected variable. In the following example, only the real part of S21 is desired.



Graph Series Wizard (Locked Graph Mode)

Type of Series Selected: S Parameters

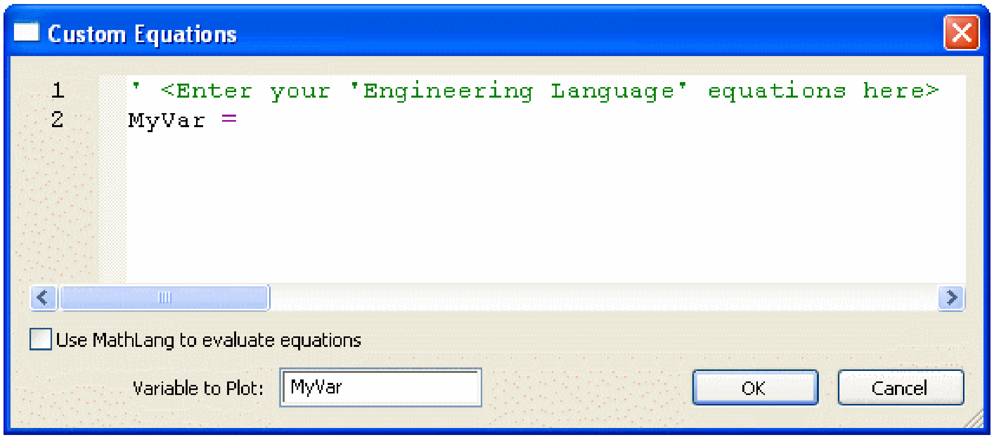
Data Selected: Real Part

Linear1_Data
- ☑ S21
- □ S31
- □ S23
- □ S11
- □ S

□ Plot On Right Y-Axis

Custom Equations button

While equations are automatically added, one can customize the equations. To edit the equations, click on the **Custom Equations** button.



```
' <Enter your 'Engineering Language' equations here>
MyVar =
```

☐ Use MathLang to evaluate equations

Variable to Plot: MyVar

For the description of the equation language, see ***Using Mathematics Language*** (users). The language functions are described in ***Math Language Function Reference*** (users).

Clear Mode button

The new graph object wizard state can be reached by clicking this button.

Plot On Right Check Box

If this box is checked, the vertical axis on the right is used for this series. This check box is also available on a per series basis in the ***Graph Properties*** (users) dialog.

OK, Cancel and Help Buttons

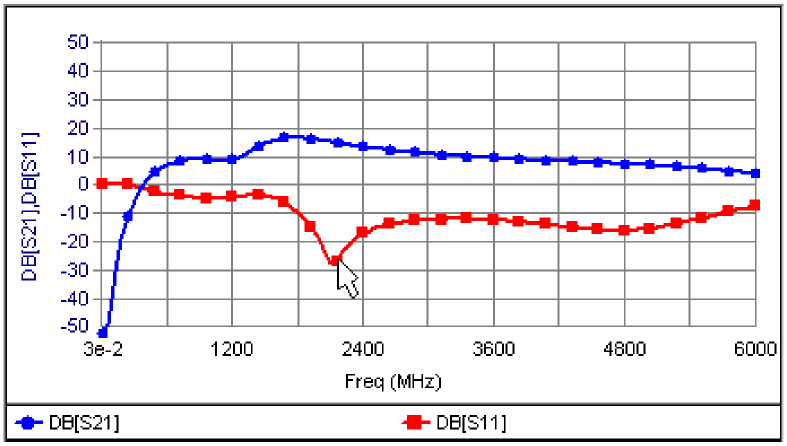
Clicking the **OK** button proceeds to the ***Graph Properties*** (users) dialog. Clicking the **Cancel** button dismisses the wizard. The **Help** button links to documentation.

Using Markers on Graphs

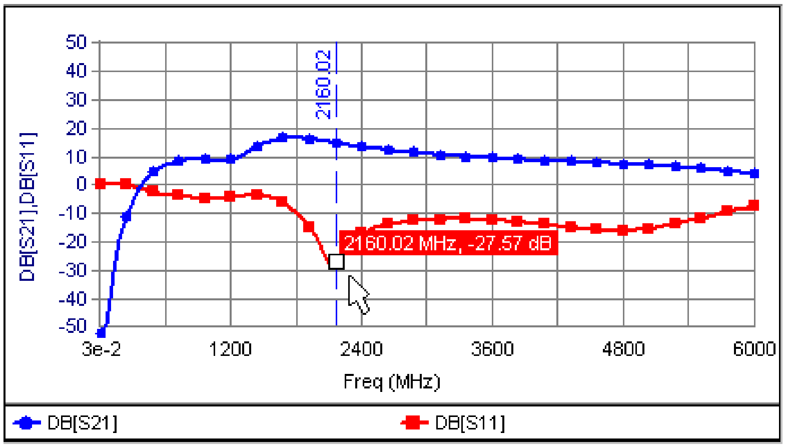
Markers are a useful way to examine and document data values on a graph.

Adding Markers to Graphs

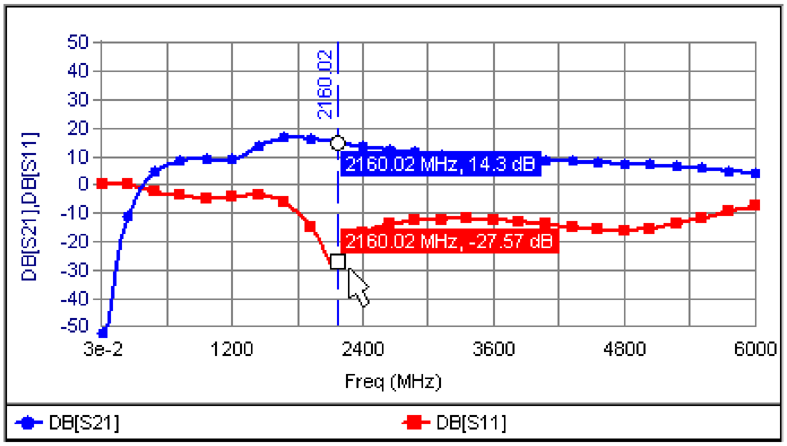
You can add markers to any graph except 3D graphs. The following figure shows a rectangular graph before adding a marker:



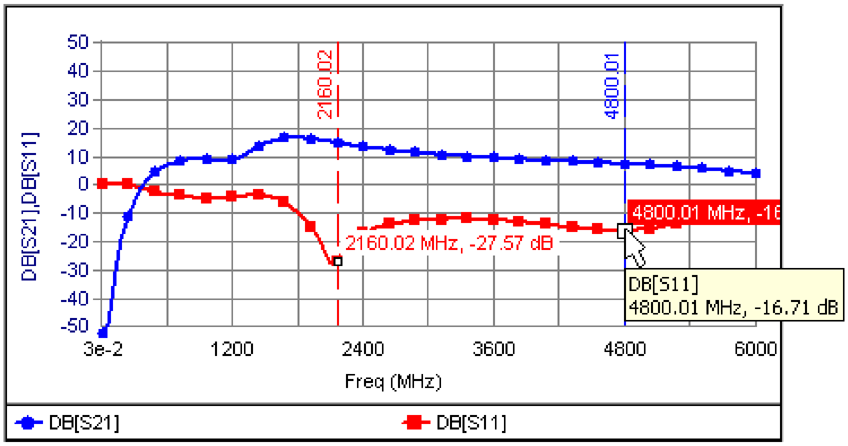
Here is the graph after placing a standard marker on the red trace:



The Mark All Traces mode displays additional marker flags on all relevant traces of chart as shown in the following figure:



Whenever a marker is selected as the currently active marker, the marker text colors are inverted (white on a colored rectangle). The figure below shows two markers. The marker on the right is selected.



To add a marker

- Click a data point on a trace. Clicking on a graph data point will create a new Marker.

To add a marker to all of the traces on a graph

- Click the **Mark All Traces** button on the graph toolbar.

To select a marker

- Click the marker you want to select.

To change the properties of a marker

- Double-click the marker to display the Marker Properties window.

To delete a marker

- Select the marker and then press the Delete key
- Alternate: click the Delete All Markers button

Marker Styles (Peak, Valley, etc.)

Genesys has several marker types. These markers are available only for rectangular graphs. A marker's type can be changed in the Marker Properties dialog box.

- Standard - A non-moving, fixed frequency marker.
- Peak - A marker that automatically tracks the peaks of a graph, even while tuning.
- Valley - A marker that automatically tracks the valleys of a graph, even while tuning.
- Bandwidth - A composite marker for ease-of-use. Bandwidth markers are **peak** markers which drop two relative markers to measure the bandwidth of the peak. A bandwidth marker can also be a **valley** marker, simply by setting the Relative offset to be a positive number.
- Relative - A marker that automatically tracks the position of another marker and are adjusted to the relative offset (dB down). Relative markers are rarely used, except when automatically placed by Genesys to indicate the limits of a bandwidth marker.
- Delta - Any marker style can be used as a delta marker. A delta marker displays the x / y distance to another marker.

Placing a Marker on a Trace

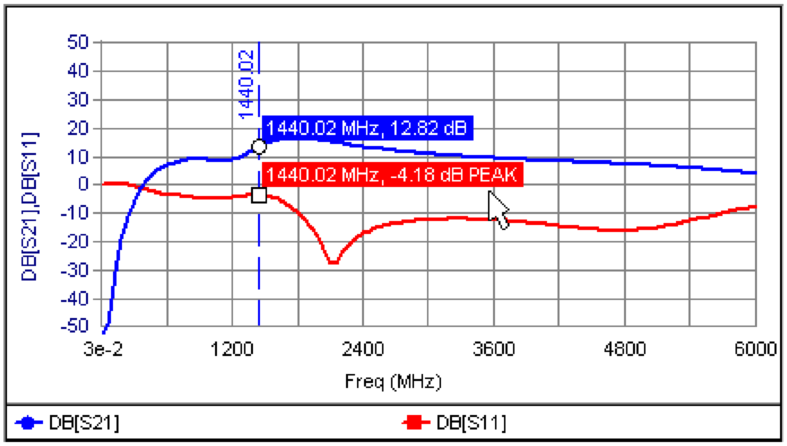
Standard marker

1. To place a Standard marker on a graph, just position the mouse over the spot where the marker is needed and click the graph trace (on or near a data point) with the left mouse button.
2. The marker can then be changed to one of the other marker types like Peak, as desired (using the Marker Toolbar or the Marker Properties window).

Peak marker

1. Place a Standard marker on a graph, as described above.
2. Click the marker and set its style to Peak using the Marker Toolbar.

The following figure is an example of what happens when a standard marker (red) is changed to a peak marker:



Notice how the marker travels to the nearest peak.

Peak/Valley detection works as follows: The "aperture" window is a box that is used for peak / valley detection. A peak or valley must be at least as large as the box, otherwise it is ignored. In general, a user should never need to adjust these, as the defaults are pretty good. A local maximum can be rejected by increasing the aperture window. Small peaks can be detected by decreasing the window size. The same criteria are used for valley detection (with a sign flip). The parameters are percentages (which are scaled by the bounds of the graph) and then used to evaluate candidate peaks / valleys and reject those that are too small to be of interest.

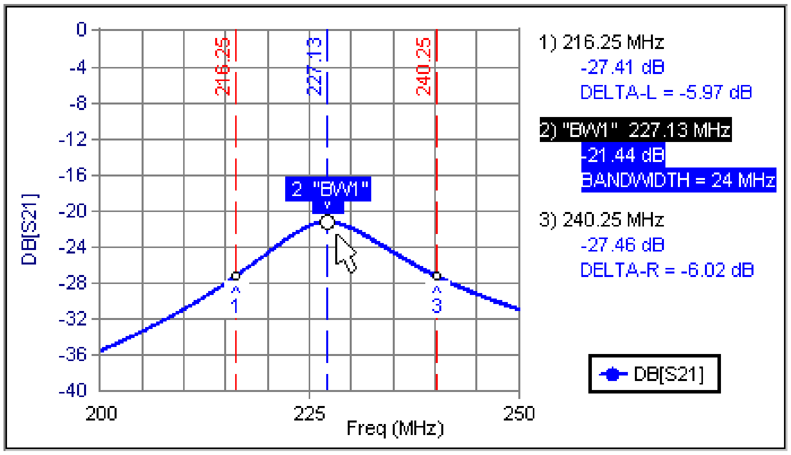
Valley marker

1. Place a Standard marker on a graph, as described above.
2. Click the marker and set its style to Valley using the *Marker Toolbar* (users).

Bandwidth marker

1. Place a Standard marker on a graph, as described above.
2. Click the marker and then click Bandwidth on the *Marker Toolbar* (users). The marker's style and name will be updated and 2 associated relative markers will be placed automatically.

Here is an example of a Bandwidth marker, along with its associated relative markers:



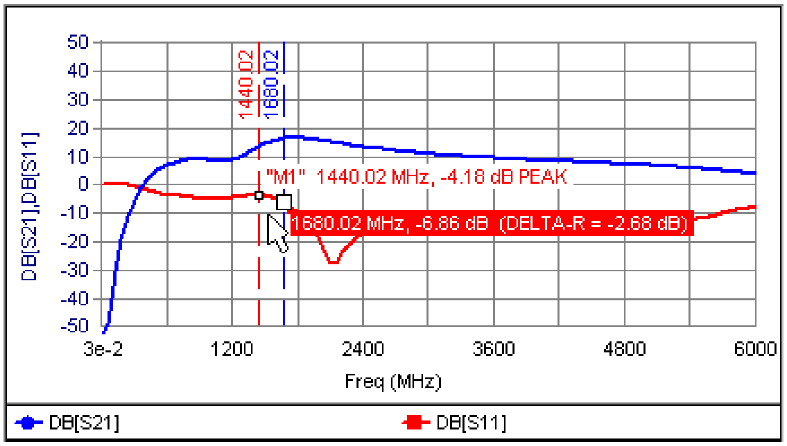
Notice that the actual measured bandwidth of 24 MHz is displayed. It is calculated directly from the positions of the two relative markers, which are both set to -6.0 dB down. You can increase the number of data points in the simulation as needed so that the relative markers are positioned with sufficient precision. You can adjust the dB down settings of both relative markers associated with the Bandwidth marker at the same time by setting the Bandwidth marker's properties. Set an individual relative marker's properties to independently set the dB down to different values.

> If you need the bandwidth based on a fixed center frequency, place a bandwidth marker, change the marker type to Standard, and then type the frequency in the marker's properties window. The relative markers automatically follow the marker to its new location.

Relative marker

1. Place a Standard marker on a graph, as described above.
2. Click the marker and set its style to Relative Left or Relative Right using the Marker Toolbar. The associated relative markers will be automatically placed.

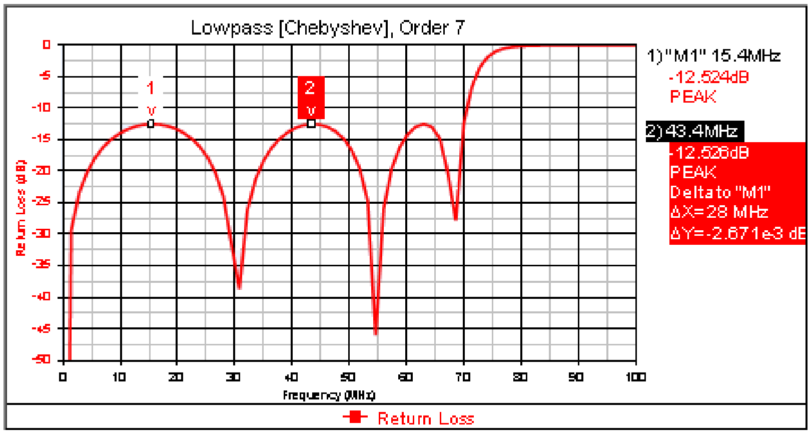
The following figure is an example of a Relative maker (on right) that is relative to the first marker ("M1"):



Notice the delta value (-2.68 dB) is displayed. That is the actual value derived from the simulation data, even though the marker's default dB down of -3.0103 dB was requested. The relative marker is always placed on an actual simulation data point.You can increase*the number of data points in the simulation as needed to get the relative marker value closer to -3 dB down.*Because this relative marker is attached to a peak marker, both markers track tuning changes in tandem.Also, notice that the original marker is automatically named M1 so the relative marker can reference it.

Delta marker

1. Double-click the existing marker that you want to measure the delta to and ensure that it has a name.
2. Place a Standard marker on a graph, as described above. This will become the "delta marker".
3. Double-click the new marker.
4. Check Show delta X (and/or delta Y).
5. Select the original marker name in the Relative To combo box.
6. Click OK.



Naming Markers

Name a marker for reference or documentation purposes. You **must** name a marker if a Relative or Delta marker references it.

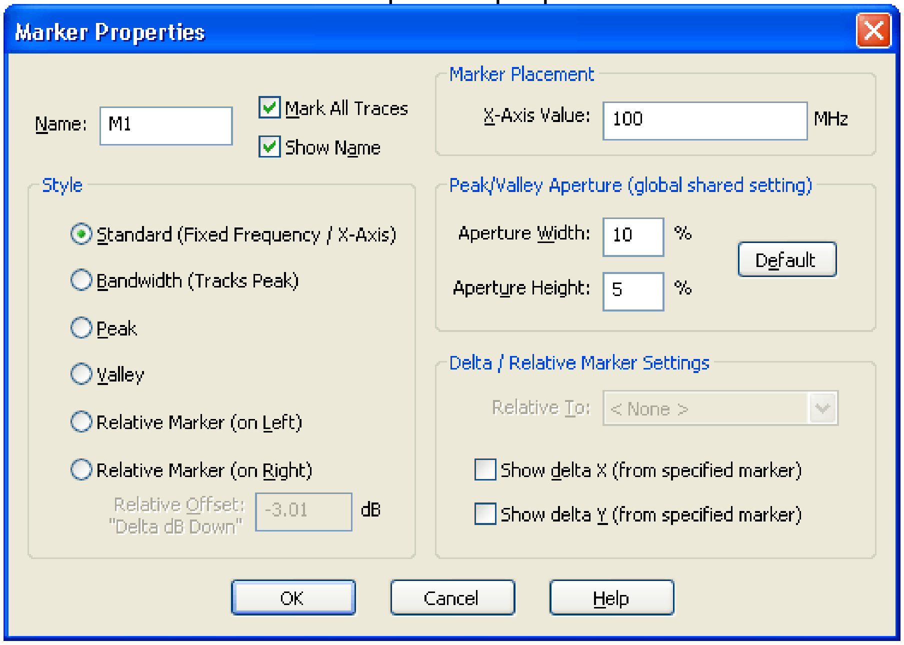
Bandwidth markers are automatically named in the format BW1, BW2, and so on. Other markers are automatically named M1, M2, and so on.You can hide marker names using the Marker Properties window; however, the name always displays on a tool tip.

Graph Marker Properties

The marker properties window lets you change the attributes of a graph marker.

To change the properties of a graph marker:

1. Double-click a marker to open its properties window.



Marker Properties

Name: M1

☑ Mark All Traces

☑ Show Name

Marker Placement

X-Axis Value: 100 MHz

Style

● Standard (Fixed Frequency / X-Axis)

○ Bandwidth (Tracks Peak)

○ Peak

○ Valley

○ Relative Marker (on Left)

○ Relative Marker (on Right)

Relative Offset: "Delta dB Down" -3.01 dB

Peak/Valley Aperture (global shared setting)

Aperture Width: 10 %

Aperture Height: 5 %

Default

Delta / Relative Marker Settings

Relative To: < None >

□ Show delta X (from specified marker)

□ Show delta Y (from specified marker)

2. Make the changes you want to the following settings:
   - **Name** - The name of the marker, which is optional, unless the marker needs to be referenced by a relative or delta marker.
   - **Mark All Traces** - When checke,d the marker will mark all traces (otherwise it will only mark a single trace).
   - **Show Name** - When checked, the name of the marker will be displayed on the graph.

- **Standard** - A normal, fixed-frequency marker.
- **Bandwidth** - A marker which uses 2 relative markers to display the bandwidth of a peak (or valley if Relative Offset is a positive number).
- **Peak** - A peak marker, which tracks a peak on the graph (even while tuning).
- **Valley** - A valley marker, which tracks a valley on the graph.
- **Relative Marker (on Left or Right)** - A tracking marker, used to measure bandwidth, etc.
- **X-Axis Value** - The marker's location on the X-axis.
- **Aperture Width / Height** - These values are shared between all graphs and are used to track peaks and valleys. The values are a percentage of the width / height of the graph window.
- **Default** - Sets the Aperture Width and Height back to the Factory Default settings of 10% and 5%.
- **Relative To** - The name of the marker to reference for relative and delta markers.
- **Show Delta X / Y** - When checked, the distance from the "delta" marker to the reference marker will be displayed.

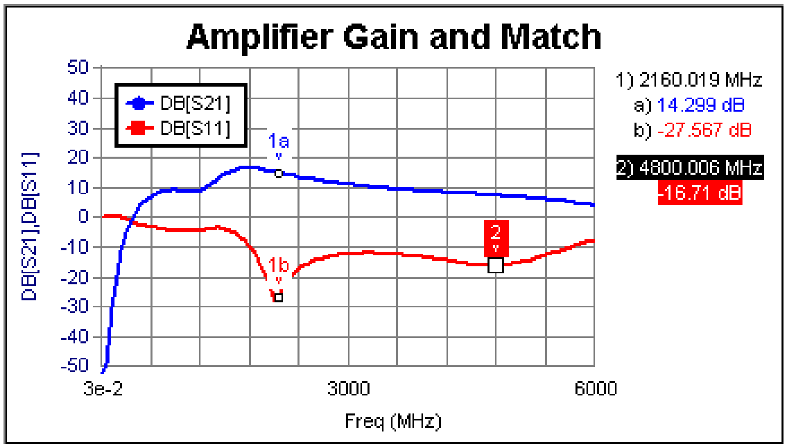
3. Click **OK**.

Customizing Graphs and Markers

Customize your graphs and markers to create a neater, more usable graph. There are many graph and marker options from which to choose.They include the following:

- Hiding vertical lines and trace symbols using the Graph menu.
- Placing marker text on the right using the Graph menu.
- Changing graph and marker settings using the Graph menu or Graph toolbar.
- Adding titles and annotations by right-clicking the graph and using the menu.
- Moving graph legends by dragging them into place.
- Removing symbols on traces using the Graph menu.

This following figure shows a less cluttered graph:



Annotating Graphs

The Annotation button ( ) on the Graph toolbar gives you access to the *Annotation Toolbar* (users). The Annotation toolbar provides lines, circles, and text that you can use to point out details of interest on a graph.

For example, the **Text Balloon** annotation has a "tail" which can be anchored to a data point on a graph, to the page, or not anchored by right-clicking on the balloon and selecting **Anchor Pointer** on the menu.

> ℹ To create a balloon that's initially anchored to a data point, first ensure that no marker is selected. If the trace vertices are not visible, right-click the trace and select **Show Vertex Symbols**. Right-click a trace vertex (or **WhatIF** bar) and select **Create Info Balloon**. The balloon will be anchored to the point and filled from the info box that is displayed when the mouse hovers over a data point.

> ℹ **Tip for advanced users:** To copy the text from the balloon to the **Windows** clipboard, click on the balloon, right-click on the balloon and select **Enter Text**, select the text and copy it to the clipboard using **Ctrl_V**.

Zooming Graphs

You can zoom on graphs using buttons on the *Graph Toolbar* (users). Depending on which graph type you are using, some of these buttons might be grayed out.

> ℹ Normally, **only** the **X-Axis** on a rectangular graph is zoomed. Hold down the **Ctrl** key to also zoom the **Y-Axis**.

> ℹ As you zoom out, the graph background may selectively skip drawing excessive details. This is intentional. Similar to using a street atlas, a state map or a world map, only the appropriate details are shown at a particular zoom setting.

Some different ways to zoom a graph follows:

1. Click one of the following buttons on the Graph toolbar to use a tool:

Click this button	To select this tool	Keyboard Shortcut
✥	Pan the graph.	P
🔍	Zoom the graph.	Z

After selecting the tool, click and drag in the graph to use the tool. When you let up on the mouse the tool disappears.
Zoom in on a rectangular area of the graph by click-dragging with the zoom tool. Click the left button to zoom in; click the right button to zoom out.

2. Click one of the following buttons on the Graph toolbar to automatically zoom to a region

Click this button	To do this
🔍	Zoom the graph to page.
🔍	Maximize the graph to show all the data.

3. Move the mousewheel in/out to zoom the schematic in/out
4. Use the keyboard + and - keys to zoom in and out.

Graph Axis Favorites

As you work with graphs, you will find that you have sections of the graph you want to study consistently. You can define an **Axis Favorite** and easily return to it with one of the following buttons on the *Graph Toolbar* (users).

Click this button	To do	Keyboard Shortcut
⭐	Save the current axis settings as a favorite.	F
⭐	Use an axis favorite setting (cycle through the saved settings).	B

When you click the Save Axis Favorite button (or use a hot key found in the Graph menu), the current axis settings will be saved in a short list of favorites. If the list is full, the new settings will overwrite the oldest **Axis Favorite**.

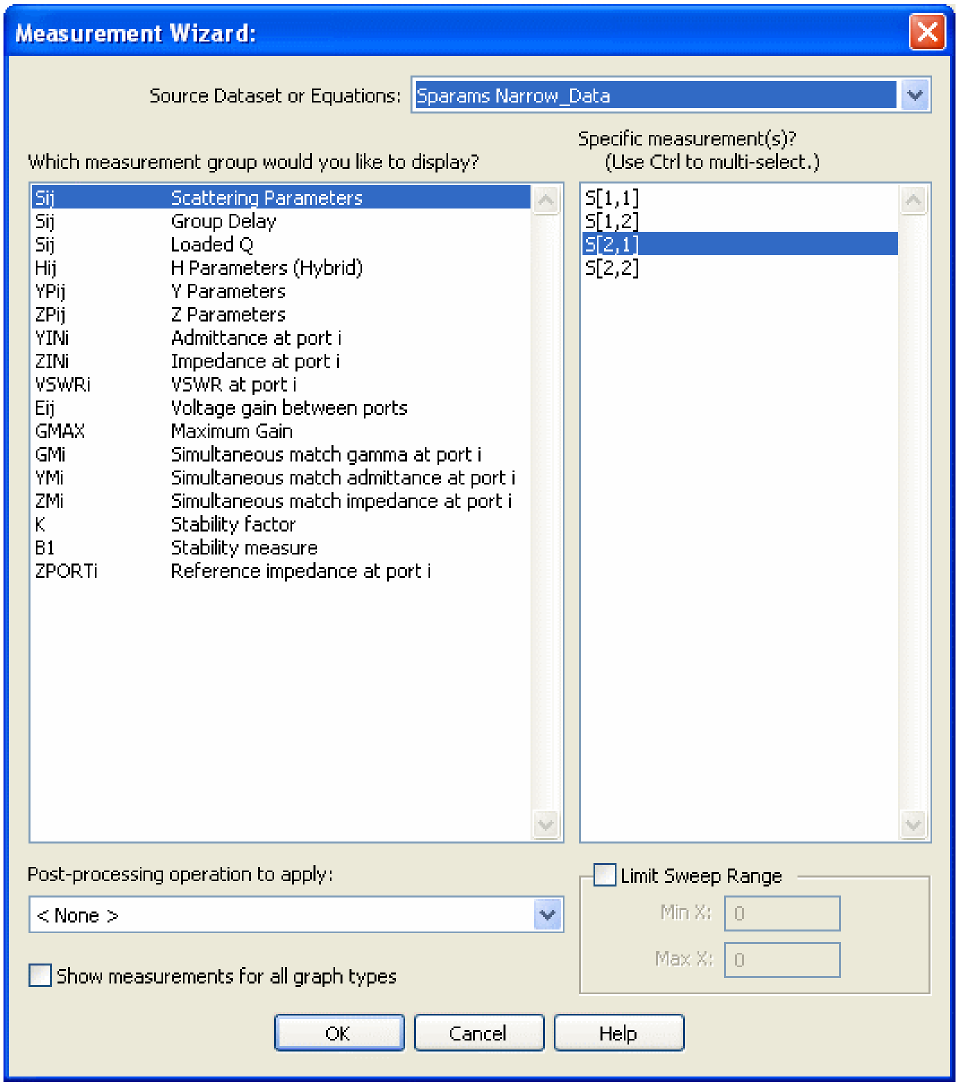
Measurement Wizard

The Measurement Wizard is used to help users find measurements and extract the right syntax and format.

The process has 4 steps:

1. Selecting the **source of data**
2. Selecting the **measurement group**
3. Selecting **specific measurements**
4. Applying any **post processing formatting** or **range limiting**

> The context of the measurement wizard is dependent on the type of graph selected on the graph properties before the measurement wizard is invoked. Measurements that are inappropriate for the given graph type will not be shown in the measurement group. For example, if a rectangular chart was specified before the measurement wizard was invoked then any type of circular measurements would be excluded from the measurement group list. *This can be overridden by selecting the Show measurements for all graph types*.



Measurement Wizard:

Source Dataset or Equations: Sparams Narrow_Data

Which measurement group would you like to display?

Sij	Scattering Parameters
Sij	Group Delay
Sij	Loaded Q
Hij	H Parameters (Hybrid)
YPij	Y Parameters
ZPij	Z Parameters
YINi	Admittance at port i
ZINi	Impedance at port i
VSWRi	VSWR at port i
Eij	Voltage gain between ports
GMAX	Maximum Gain
GMi	Simultaneous match gamma at port i
YMi	Simultaneous match admittance at port i
ZMi	Simultaneous match impedance at port i
K	Stability factor
B1	Stability measure
ZPORTi	Reference impedance at port i

Specific measurement(s)?
(Use Ctrl to multi-select.)

S[1,1]
S[1,2]
S[2,1]
S[2,2]

Post-processing operation to apply: < None >

☐ Limit Sweep Range
Min X: 0
Max X: 0

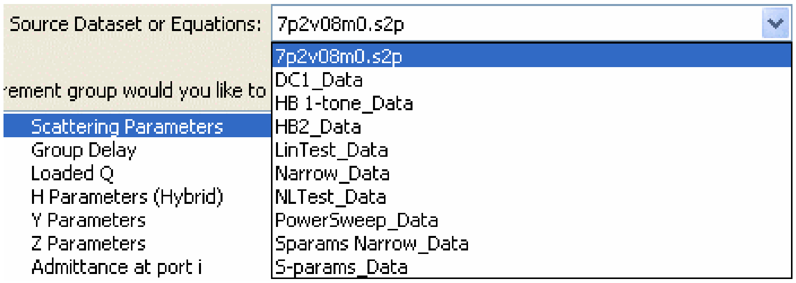
☐ Show measurements for all graph types

Using the Measurement Wizard

1. Click the Measurement Wizard button (More Measurements...) on the graph series

wizard dialog box.

2. Select the dataset at the top of the dialog box. All available measurements for that dataset will be refreshed.



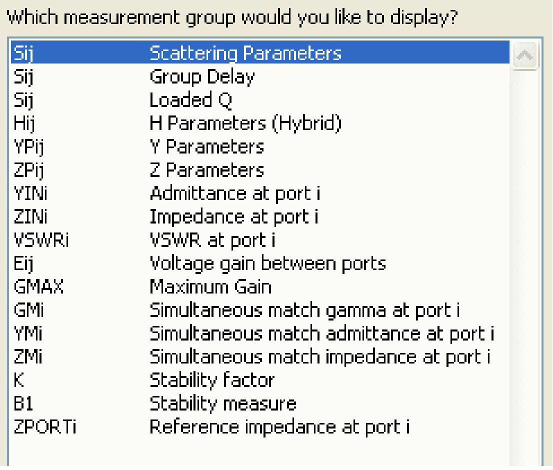
Source Dataset or Equations: 7p2v08m0.s2p

ement group would you like to

- Scattering Parameters
- Group Delay
- Loaded Q
- H Parameters (Hybrid)
- Y Parameters
- Z Parameters
- Admittance at port i

- 7p2v08m0.s2p
- DC1_Data
- HB 1-tone_Data
- HB2_Data
- LinTest_Data
- Narrow_Data
- NLTest_Data
- PowerSweep_Data
- Sparams Narrow_Data
- S-params_Data

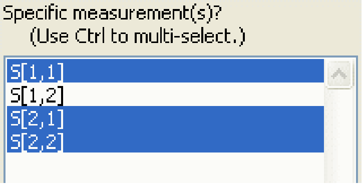
3. Select the measurement group in the list box to the left. Specific measurements for that group will be refreshed and appear in the list box to the right.



Which measurement group would you like to display?

Sij	Scattering Parameters
Sij	Group Delay
Sij	Loaded Q
Hij	H Parameters (Hybrid)
YPij	Y Parameters
ZPij	Z Parameters
YINi	Admittance at port i
ZINi	Impedance at port i
VSWRi	VSWR at port i
Eij	Voltage gain between ports
GMAX	Maximum Gain
GMi	Simultaneous match gamma at port i
YMi	Simultaneous match admittance at port i
ZMi	Simultaneous match impedance at port i
K	Stability factor
B1	Stability measure
ZPORTi	Reference impedance at port i

4. Select specific measurement(s) in the right list box.

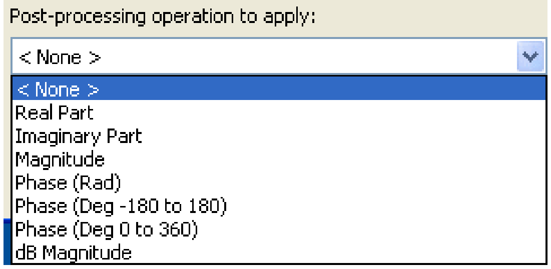


Specific measurement(s)?
(Use Ctrl to multi-select.)

- S[1,1]
- S[1,2]
- S[2,1]
- S[2,2]

Standard Windows multiselection is available when selecting specific measurements. To select a continuous block of measurements single click the top item in the list then hold the **Shift** key down and single click the bottom item. To select individual measurements hold the **Ctrl** key down while single clicking the desired measurements.

5. Select any post processing format to apply.



Post-processing operation to apply:

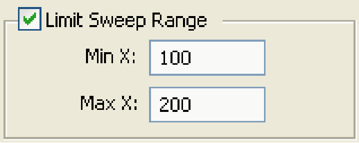
< None >

- < None >
- Real Part
- Imaginary Part
- Magnitude
- Phase (Rad)
- Phase (Deg -180 to 180)
- Phase (Deg 0 to 360)
- dB Magnitude

6. Click the **Ok** button to finish.

Limit Sweep Range

At times it is convenient for the user to limit the range of data being display. For sweeps, like linear analysis this data range can be limited by checking the **Limit Sweep Range** checkbox and setting the X axis range. For example a linear simulation may generate data from 10 to 500 MHz, but only the range from 100 to 200 MHz is of interest.



☑ Limit Sweep Range

Min X: 100

Max X: 200

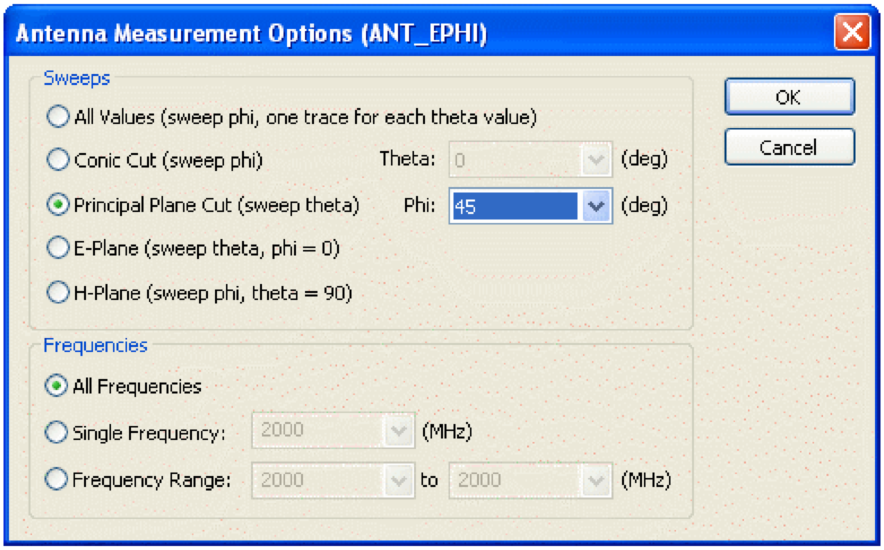
Show Measurements for all graph types

By default only measurement groups for the **graph type** set in the graph properties dialog box is used. This prevents users from trying to plot circular measurements on rectangular types of graphs and vise-versa. To override this behavior the *'Show measurements for all graph types'* can be checked.

☐ Show measurements for all graph types

Antenna Measurement Options

When an **Antenna Plot** graph type has been selected and a radiation pattern measurement is selected in the measurement wizard the **Antenna Measurement Options** dialog box will be invoked. The parameters on this dialog box can be used to slice 3D data into the desired two dimensional pattern to be plotted.



Antenna Measurement Options (ANT_EPHI)

Sweeps

○ All Values (sweep phi, one trace for each theta value)

○ Conic Cut (sweep phi) Theta: 0 (deg)

● Principal Plane Cut (sweep theta) Phi: 45 (deg)

○ E-Plane (sweep theta, phi = 0)

○ H-Plane (sweep phi, theta = 90)

Frequencies

● All Frequencies

○ Single Frequency: 2000 (MHz)

○ Frequency Range: 2000 to 2000 (MHz)

Importing and Exporting

Contents

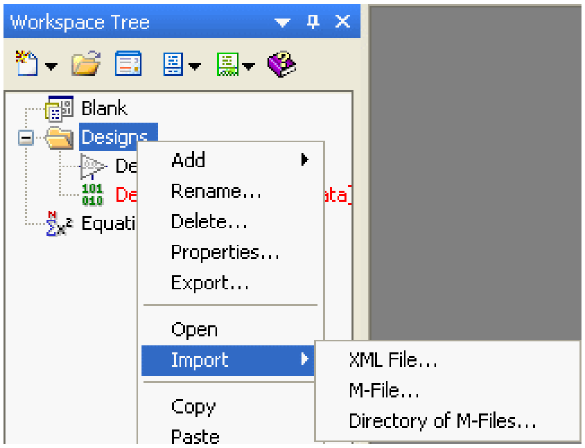
Importing Data Files Using Genesys

Genesys can import the following file types:

- DXF/DWG File
- GDSII File
- Genesys Netlist
- Gerber File
- Load Pull Data File
- M-File
- Directory of M-Files
- S-Data File
- SPICE File
- XML File
- 6.0 Model Library
- Old Genesys S-Data File
- CITI File

To import a file

1. Click File on the Genesys menu and select a file type from the Import menu.
2. Follow the instructions in the windows that appear.
   Or
3. Right click on a folder in the Workspace Tree and select the Import menu.
4. Follow the instructions in the windows that appear.

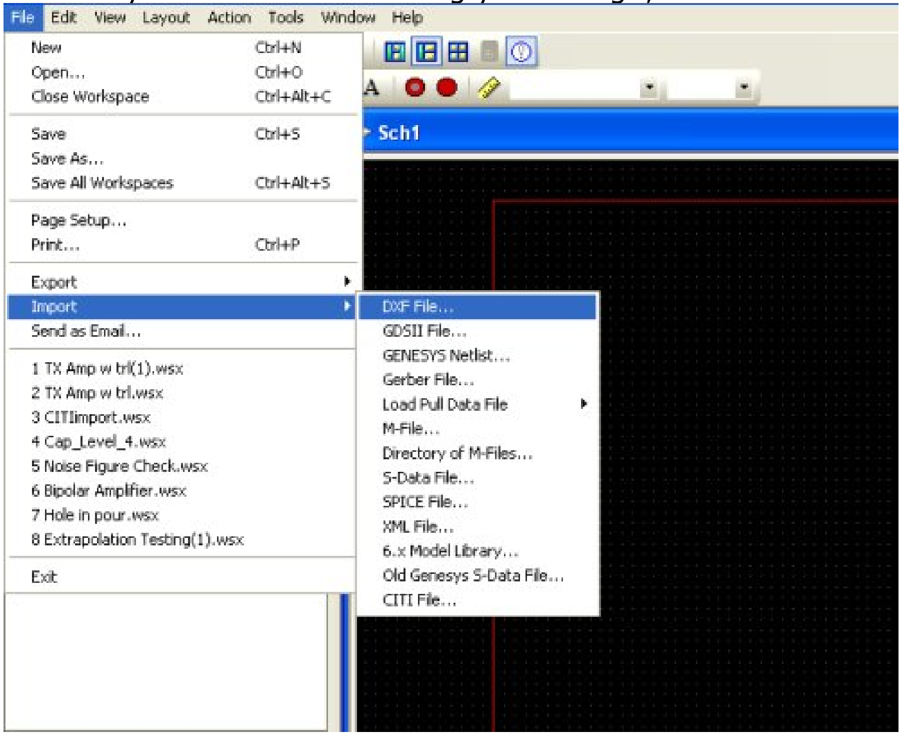


DXF/DWG Files Import

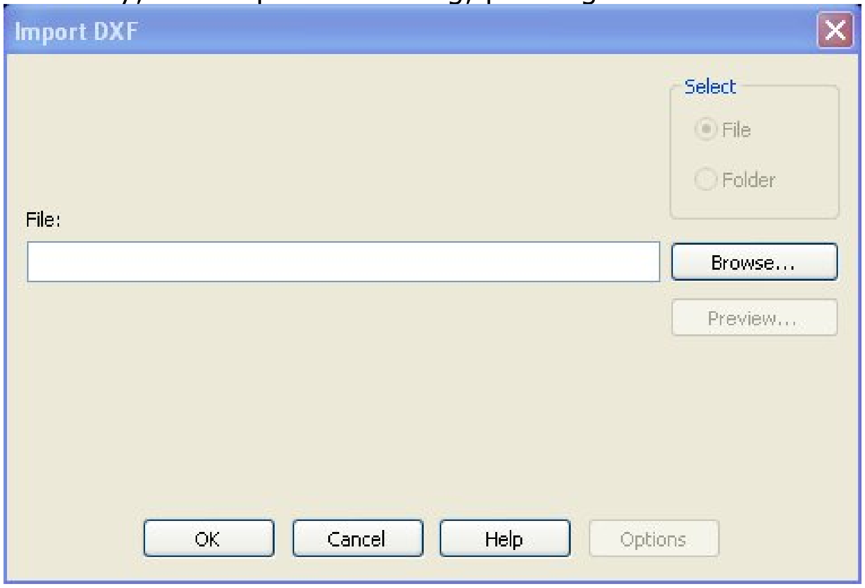
The *Import DXF/DWG Options* dialog enables you to specify units, and layers to control the import of DXF/DWG format files. A DXF/DWG file must be selected in the main export dialog for the *Import DXF/DWG Options* dialog to open.

To import AutoCAD DXF/DWG files:

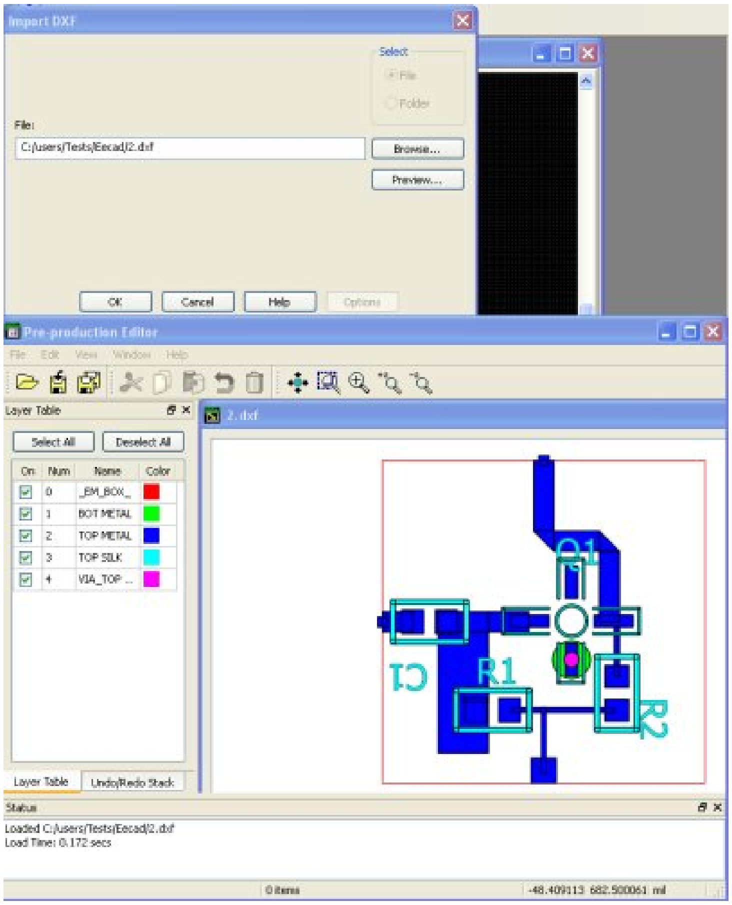
1. In the layout window containing your design, choose **File > Import > DXF File...**



2. Enter path to imported DXF(DWG) file in the opened "Import DXF" dialog window manually, from Open File dialog, pushing "Browse" button:



3. If the path point to existing file, it enables "Preview" button, pushing on which opens the importing file in Preproduction Editor



Import DXF

Select

● File

○ Folder

File:

C:/users/Tests/Eecad/2.dxf

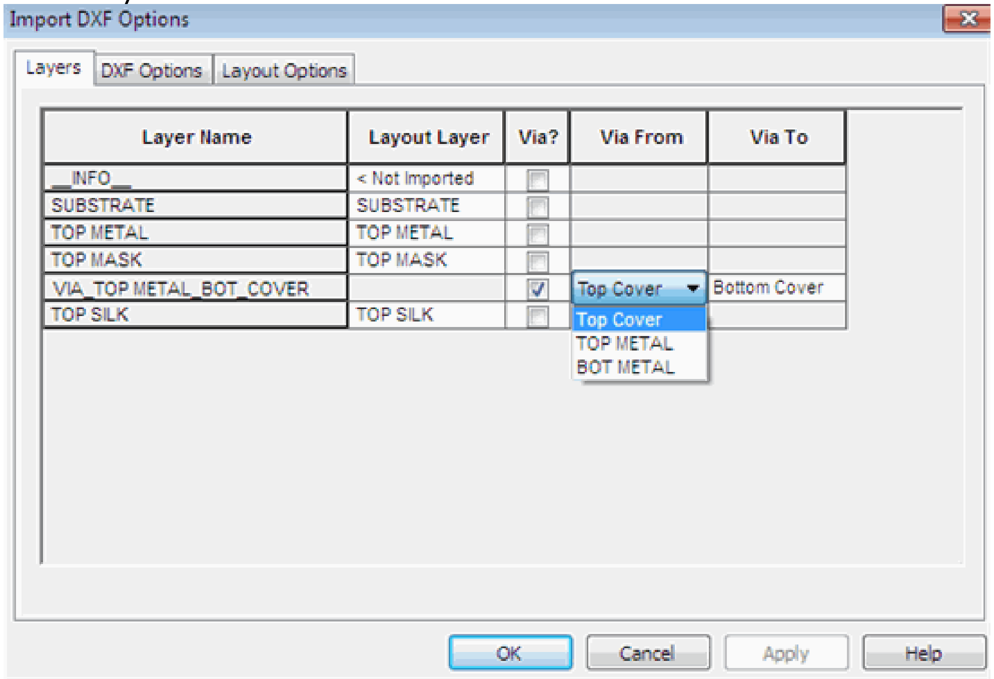
Pre-production Editor

On	Num	Name	Color
☑	0	_EM_BOX_	🟥
☑	1	BOT METAL	🟩
☑	2	TOP METAL	🟦
☑	3	TOP SILK	🟦
☑	4	VIA_TOP ...	🟪

Loaded C:/users/Tests/Eecad/2.dxf
Load Time: 0.172 secs

0 items   -48.409113 682.500061 mil

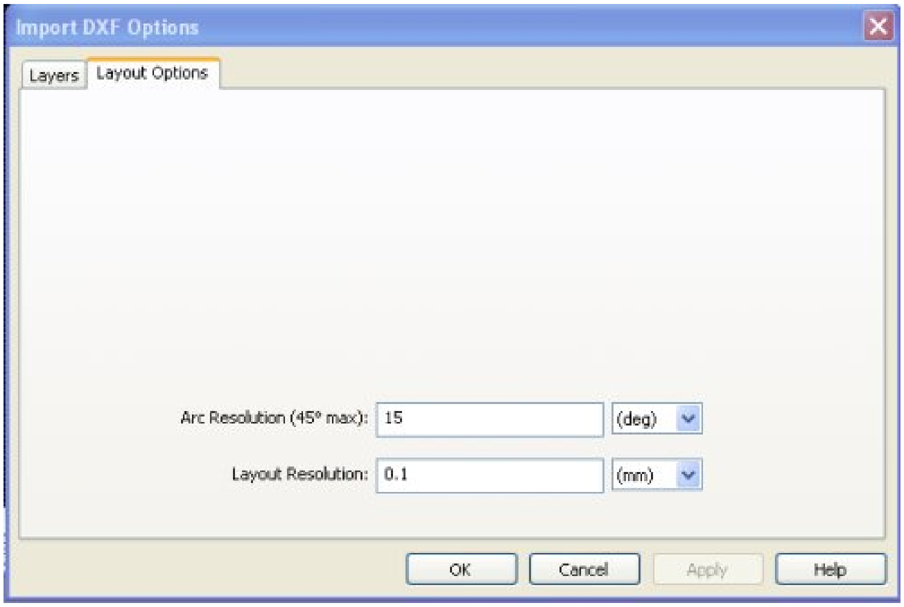
4. Push button **OK** to continue DXF/DWG import. It Opens **Import DXF Options** dialog, **Layers** tab.The *Layers* tab displays a list all the layers in DXF/DWG file. Layers can be selectively imported to Genesys using the import column. By default all layers are imported. Map importing file layers to Genesys layers, choosing them from **Layout Layer** combo boxes, or to viahole layers, checking **Via?** checboxes, and choosing metal layers from **Via From** and **Via To** combos:



Import DXF Options

Layer Name	Layout Layer	Via?	Via From	Via To
__INFO__	< Not Imported	□		
SUBSTRATE	SUBSTRATE	□		
TOP METAL	TOP METAL	□		
TOP MASK	TOP MASK	□		
VIA_TOP METAL_BOT_COVER		☑	Top Cover	Bottom Cover
TOP SILK	TOP SILK	□		

5. Set layout arc and linear items resolutions, or use default:



Arc Resolution (45° max): 15 (deg)

Layout Resolution: 0.1 (mm)

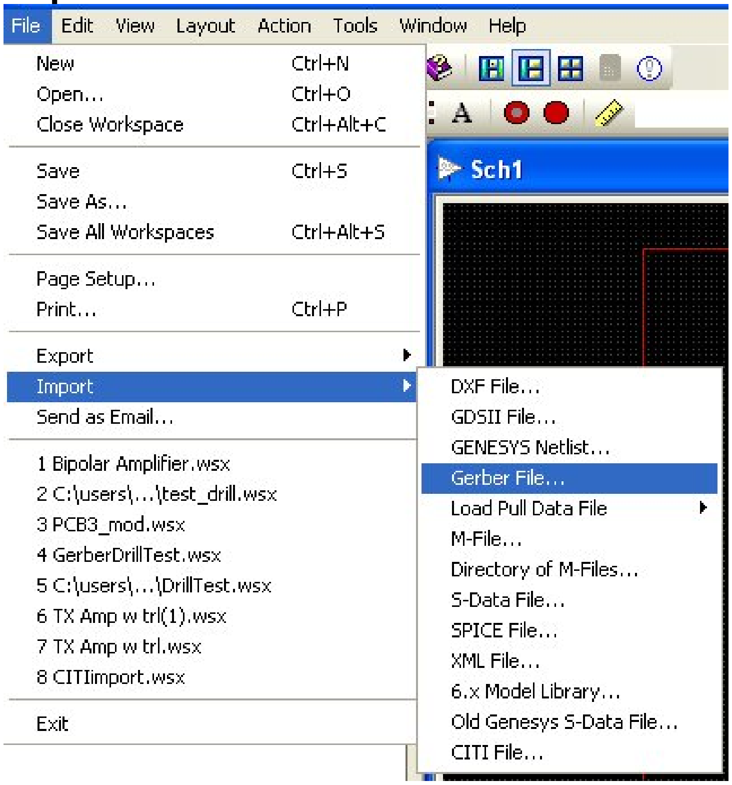
6. Push **OK** button to import the AutoCAD file.

GDSII File Import

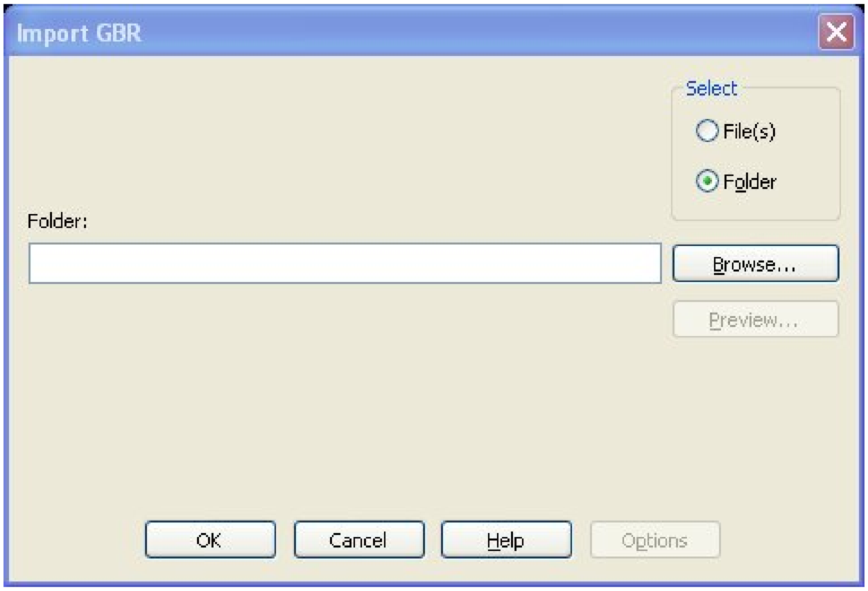
Genesys Netlist Import

Gerber File Import

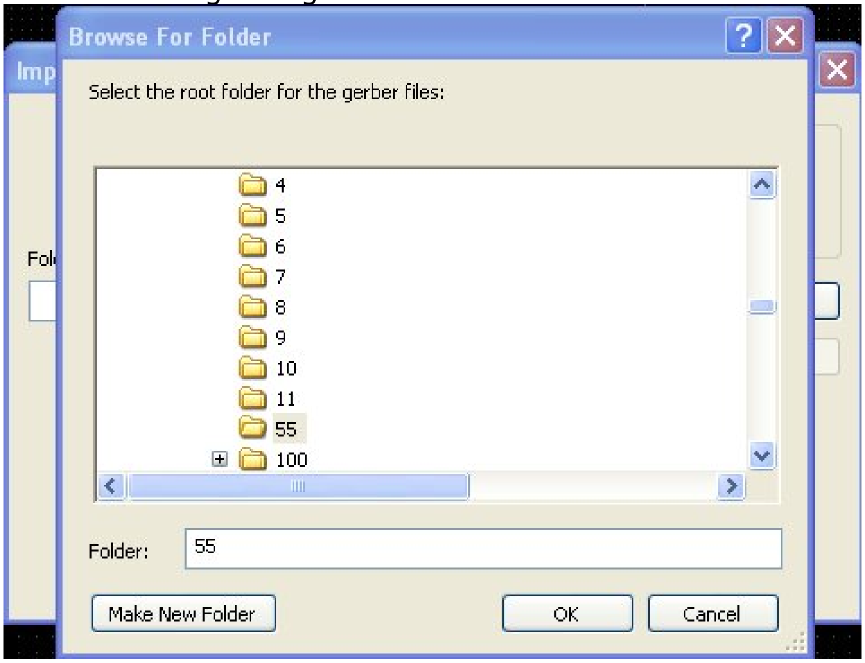
1. In the layout window where you woud like to import Gerber design, choose **File > Import > Gerber File**...



2. In the opened "Import GBR" dialog window select import File or Folder. Folder is default for Gerber Import:



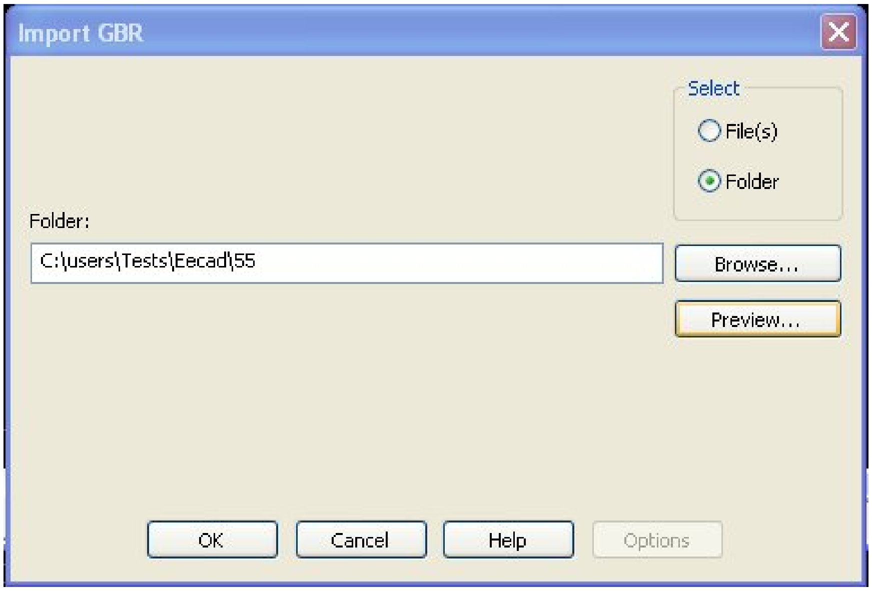
3. Enter the imported File or Folder path, or push button "Browse" to open File or Folder entering dialog:



Select the root folder for the gerber files:

Folder: 55

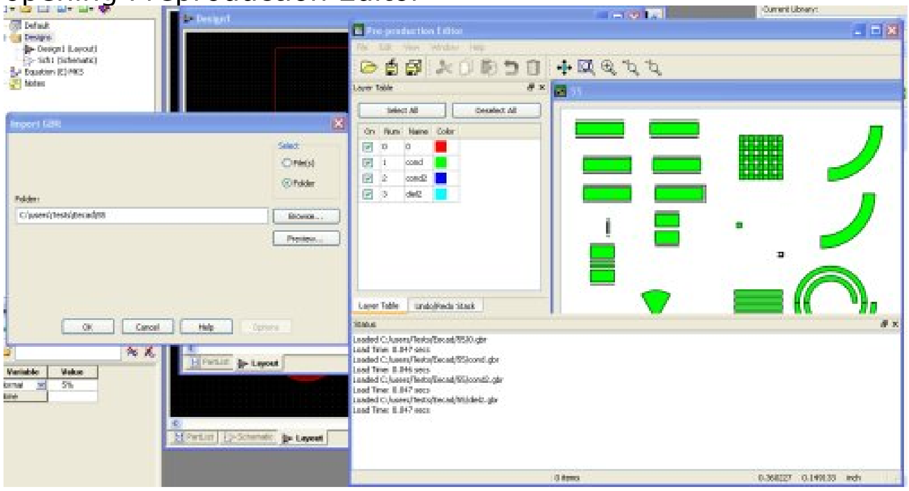
4. After entering correct File (or Folder) path, it may be previewed by pushing button "Preview",



Select
○ File(s)
● Folder

Folder: C:\users\Tests\Eecad\55

opening Preproduction Editor

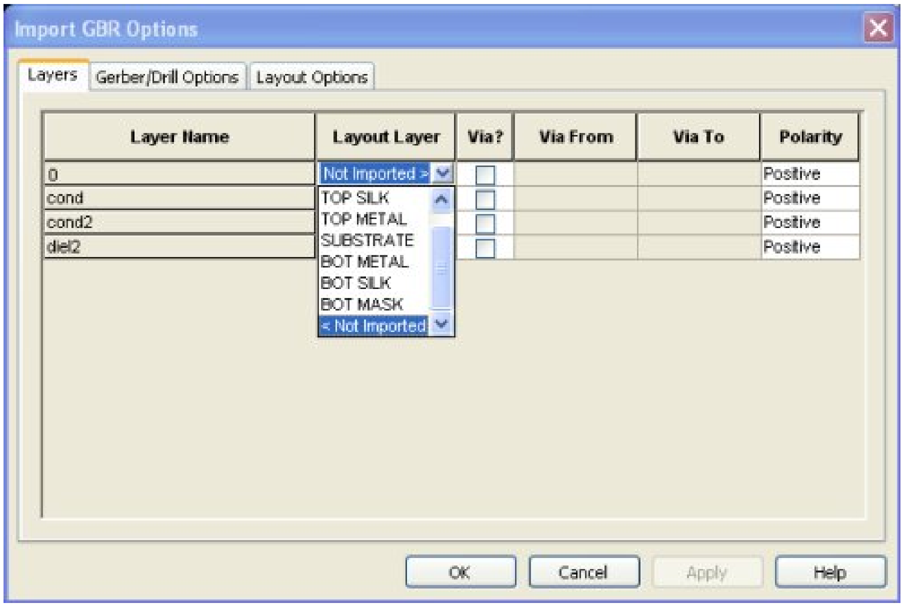


On	Num	Name	Color
☑	0	0	🟥
☑	1	cond	🟩
☑	2	cond2	🟦
☑	3	diel2	🟦

```
Loaded C:/users/Tests/Eecad/55\0.gbr
Load Time: 0.047 secs
Loaded C:/users/Tests/Eecad/55\cond.gbr
Load Time: 0.046 secs
Loaded C:/users/Tests/Eecad/55\cond2.gbr
Load Time: 0.047 secs
Loaded C:/users/Tests/Eecad/55\diel2.gbr
Load Time: 0.047 secs
```

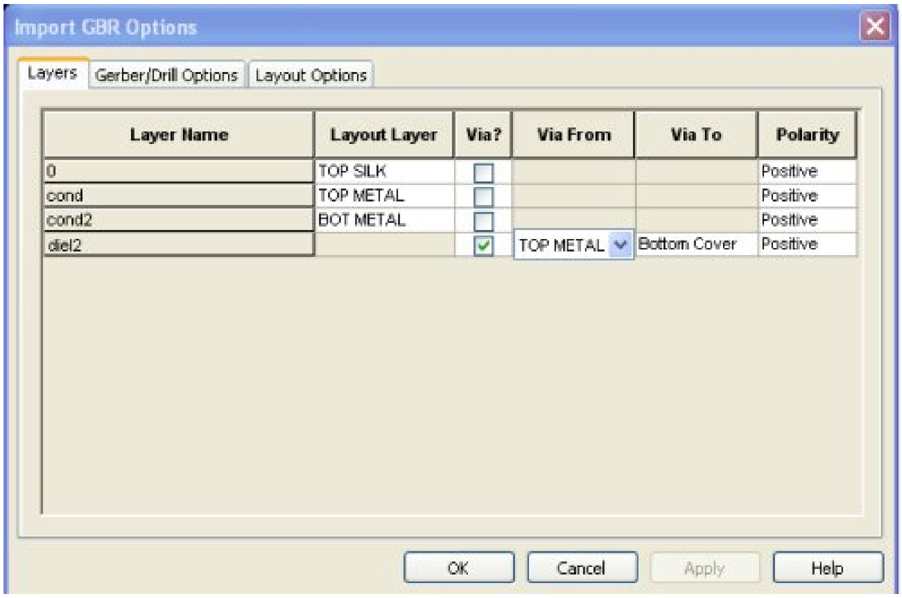
0 items 0.268227 0.149133 inch

5. Push button **OK** of the Import GBR dialog. It opens "Import GBR Options" dialog. Map imported Gerber files layers to Genesys Layout window layers, selecting it from pull down combo boxes:



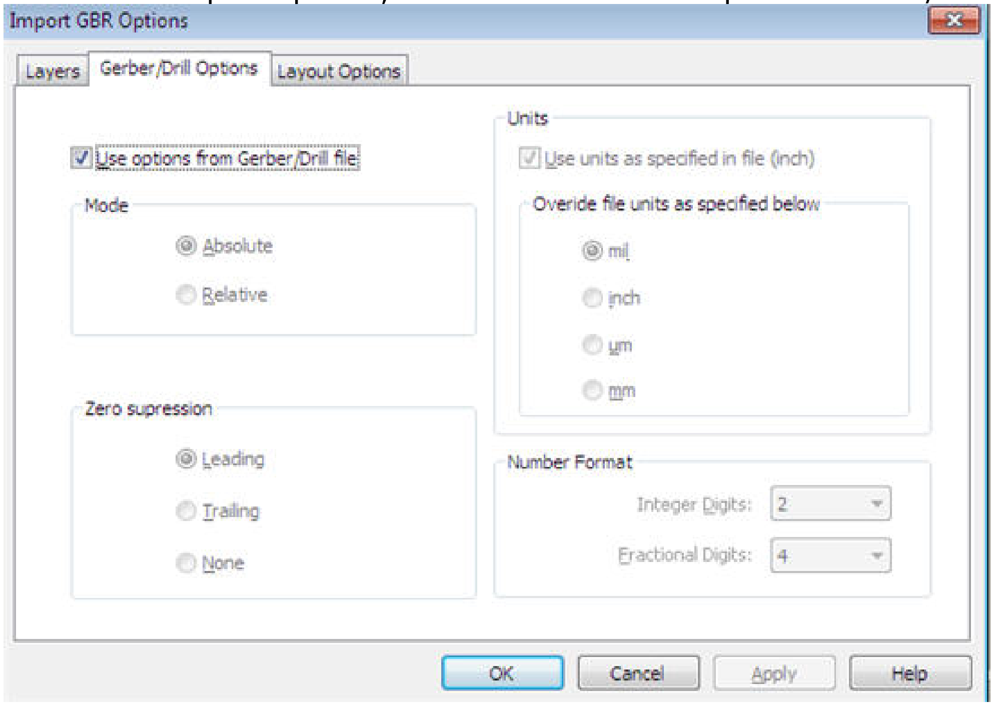
Layer Name	Layout Layer	Via?	Via From	Via To	Polarity
0	Not Imported >	☐			Positive
cond		☐			Positive
cond2		☐			Positive
diel2		☐			Positive

and to Genesys Layout viaholes layers, selecting checkbox "Via?", and selecting the viahole metal layers form pull down Genesys metal layers (including metal covers) combo boxes:



Layer Name	Layout Layer	Via?	Via From	Via To	Polarity
0	TOP SILK	□			Positive
cond	TOP METAL	□			Positive
cond2	BOT METAL	□			Positive
diel2		☑	TOP METAL	Bottom Cover	Positive

6. Set Gerber import options, or use default from imported Gerber/Drill files:



☑ Use options from Gerber/Drill file

Mode

● Absolute
○ Relative

Zero supression

● Leading
○ Trailing
○ None

Units

☑ Use units as specified in file (inch)

Overide file units as specified below

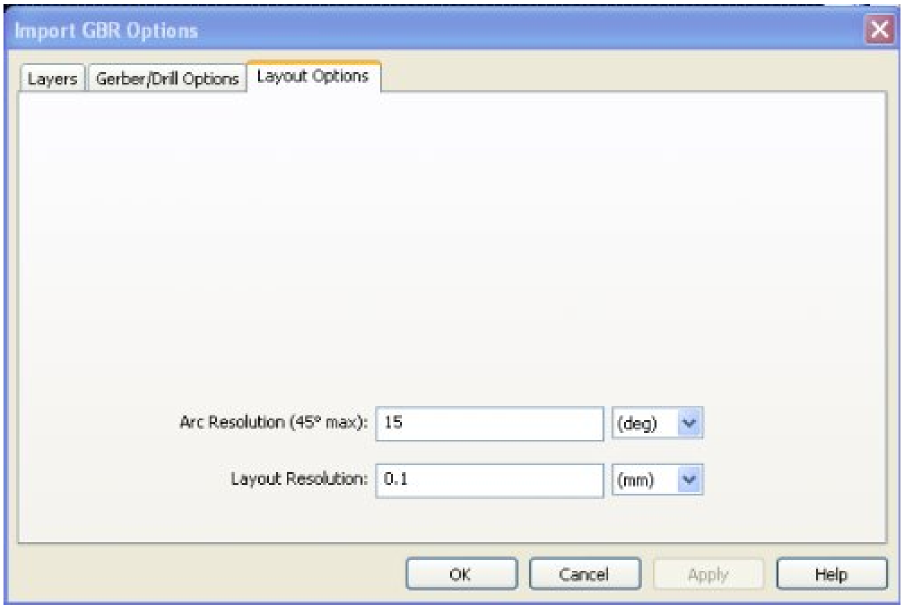
● mil
○ inch
○ um
○ mm

Number Format

Integer Digits: 2

Fractional Digits: 4

7. Set layout resolution options at "Layout Options" tab for importing arcs and linear layout object items:



Arc Resolution (45° max): 15 (deg)

Layout Resolution: 0.1 (mm)

8. Push **OK** button for importing, or **Cancel** to cancel the operation.

Load Pull Data Import

M-File Import

Imported M-Files are placed in an equation block on the workspace tree.

1. Browse to the M-file of interest
2. Click **OK**

Directory of M-Files Import

All of the M-Files located in a selected directory are imported and placed in an equation blocks on the workspace tree.

1. Browse to the M-file directory of interest
2. Click **OK**

S-Parameter Files Import

When S-Parameters are imported a dataset is create and placed in the workspace tree. This dataset is saved and loaded with the workspace and will be cached in memory to increase the simulation speed. The dataset can be deleted from the workspace. Memory cache will be used until there is a need to re-read the dataset from the workspace tree or if the dataset is not found the original file will be re-imported and cached once again.

S-Parameter can be imported in one of several ways detailed below:

Importing from a library

1. Open up the **Parts Selector** (Ctrl_Shift_A) or click the part selector button (🗏)
2. Change the **Current Library** to the S-Parameter library of interest
3. Click the library part of interest (the mouse cursor will change to a + sign)
4. Place the part in the schematic by clicking the schematic

> ⓘ NOTE: On first use of the selected library it will be unzipped and the S-Parameter file associated with the part will be imported into the workspace tree

Or

Importing from the Main Menu

1. Select **File**, **Import**, then **S-Data File** from the main Genesys menu
2. Browse to the S-Parameter file
   Or

Importing from a Part

1. Place a S-Parameter part in the schematic (*1-port* (part), *2-port* (part), *n-port* (part)). This can be done from the Linear Toolbar or the Part Selector
2. Double click the part to bring up the part properties
3. Click the **Browse** button to browse to the S-Parameter file

Manually Imported S-Parameters

Manually imported S-Parameters don't use a filename name. The data must exist on the workspace tree. If the dataset is deleted then there is insufficient information to correctly build the model. The model has no need of the filename and simply needs to know the name of the dataset.

1. Place a dataset part in the schematic (*NPOD* (part)). This can be done from the Linear Toolbar or the Part Selector
2. Double click the part to bring up the part properties
3. Set the **dataset** name to the name of the imported S-Parameters
4. Add an analysis and point it to the desired schematic
5. Run the analysis

Reference

NPOD

SPICE File Import

One of the easiest ways to get nonlinear device models into Genesys is to import them from a manufacturer supplied SPICE file. SPICE files have the following advantages over other methods of using nonlinear device data:

- They are often supplied by manufacturers.
- Entering device data manually is tedious and error-prone.
- SPICE files often contain very complete macro-model device characterizations.
- They are in plain text format and can be corrected in any text editor.

To import a SPICE file:

1. Click **File** on the Genesys menu and select **SPICE File** from the **Import** menu. Select the SPICE file and click **Open**.
2. The models are imported into your workspace, but if you also want to add them to a design library, check **Automatically import these models into library** and select a library from the drop-down list.
3. If you want to store the original SPICE code with the models (recommended), check **Embed the original spice file into a notes section**.

4. Some devices, such as transistors, normally have their nodes reversed in the SPICE file. Genesys does not have enough information to know what device is described by any particular SPICE subcircuit, so you have to tell it whether the first two nodes of any model with 3 to 5 total nodes should be reversed. **Netlist options for devices with 3 to 5 nodes** allow you to set node reversal for the whole file or receive a separate prompt for each device found.
5. Click **OK**.

Notes:

1. The import operation may take up to several minutes, depending on the number of models in your SPICE file. A whole library of models can reside in one file.
2. Genesys will not allow two models with the same name in one workspace. If a name conflict happens, you can either replace the existing model, or rename the new model in the SPICE file and try importing it again.
3. For a model with more than 5 nodes Genesyss will issue a warning (an integrated circuit can have a number of transposed nodes).
4. You can edit the design's netlist in Genesys (after the model is imported) to fix the issues beyond the node 1 & 2 reversal.
5. Occasionally SPICE files have errors in them, mostly simple typos. Those need to be corrected in a text editor prior to importing the file.

Once imported, the model can be used just like any design created in Genesys.

Quite often SPICE models are organized into a hierarchy, in which case you need to have all of the model's dependents available in order to use the model. Sometimes dependents will be in a different SPICE file which you will need to import into the same workspace or design library.

SPICE File Compatibility

SPICE file import in Genesys is mainly based on the PSpice Version 10.0 specification. Where possible, Genesys has also been made compatible with other SPICE flavors. The following devices are recognized:

C - Ideal Capacitor
D - Nonlinear Diode
E - Voltage-Controlled Voltage Source
F - Current-Controlled Current Source
G - Voltage-Controlled Voltage Source
H - Current-Controlled Current Source
I - Independent Current Source
J - Nonlinear JFET
K - Mutual Inductance (coupling of two inductors only)
L - Ideal Inductor
M - Nonlinear MOSFET
Q - Nonlinear BJT
R - Ideal Resistor
T - Ideal Transmission Line
V - Independent Voltage Source
X - Subcircuit
Z - Nonlinear MESFET (model types NMF and PMF)

The intent of the SPICE file import facility in Genesys is to capture parameters of real manufacturer parts rather than to represent more abstract models, and thus only SPICE commands directly related to concrete device modeling are supported: .SUBCKT, .ENDS, .MODEL and .PARAM.

Expressions are supported only as far as they are compatible with the Genesys equation parser, and .FUNC command is not supported. Also unsupported in this release are options VALUE, TABLE, LAPLACE, FREQ and CHEBYSHEV.

Subcircuits can have parameters, but not optional nodes, and option AKO is not supported in models.

Based on the extensive review of available SPICE files these limitations don't seem too restrictive, however, SPICE file compatibility will be improving in future releases.

New Spice model import supports the LEVEL keyword for JFET, MOSFET and MESFET models.

The value of the parameter LEVEL may be numerical or string. The string format is:

<Model Name>@<Full Path to model>

Use the string format to define an external link to a Genesys VerilogA model. The parameter list may be unknown to Genesys, but for the import it is compiled from the VerilogA file.

For example:

BSIM4_NMOS@bsim4.va

ANGELOV_NFET@c:\Program Files{product-name}\Model\VerilogA\angelov.va

> **Note** Spice import converts all SPICE file lines to upper case, while Genesys model names are case sensitive. To support the VerilogA model SPICE import, the module name in the VerilogA file must be the upper case string, otherwise after the import is performed, the model will not be found.

The numerical values of the LEVEL parameter are defined from the tables:

Table 1: MESFET models LEVEL parameter value (TYPE = NFET, PFET)

LEVEL	**Model**	**Description**
0, 4	MES_<TYPE>	Spice MESFET
1	CURTICE2_<TYPE>	Curtice-2 (quadratic)
2	STATZ_<TYPE>	Statz
3	TOM_<TYPE>	TOM-1
5	TOM2_<TYPE>	TOM-2
6	CURTICE3_<TYPE>	Curtice-3 (cubic)

Table 2: MOSFET models LEVEL parameter value (TYPE = NMOS, PMOS)

LEVEL	Model	Description
0, 1	MOS1_<TYPE>	Spice MOS1
2	MOS2_<TYPE>	Spice MOS2
3	MOS3_<TYPE>	Spice MOS3
4	BSIM1_<TYPE>	Spice BSIM-1
5	BSIM2_<TYPE>	Spice BSIM-2
7, 8	BSIM3_<TYPE>	Spice BSIM-3
14	BSIM4_<TYPE>	Spice BSIM-4

Table 3: JFET models LEVEL parameter value (TYPE = NJF, PJF)

LEVEL	Model	Description
1	JFET_<TYPE>	Spice JFET
2	JFET2_<TYPE>	Spice JFET-2 (Parker-Skellern)

XML File Import

Each Genesys object in the workspace tree has an XML format associated with it. Workspace tree objects that have been exported to an XML format can be re-imported into any workspace. To import an XML file:

1. Select the Import menu option from one of the given methods
2. Click XML file
3. Browse to the XML file of interest

6.x Model Library

Old Genesys S-Parameters

CITI file Import

Overview

CITIfile is a standardized data format that is used for exchanging data between different computers and instruments. CITIfile stands for *Common Instrumentation Transfer and Interchange* file format.

This standard is a group effort between instrument and computer-aided design program designers. As much as possible, CITIfile meets current needs for data transfer, and it is designed to be expandable so it can meet future needs.

CITIfile defines how the data inside an ASCII package is formatted. Since it is not tied to any particular disk or transfer format, it can be used with any operating system, such as DOS or UNIX, with any disk format, such as DOS or HFS, or with any transfer mechanism, such as by disk, LAN, or GPIB.

By careful implementation of the standard, instruments and software packages using CITIfile are able to load and work with data created on another instrument or computer. It is possible, for example, for a network analyzer to directly load and display data measured on a scalar analyzer, or for a software package running on a computer to read data measured on the network analyzer.

Data Formats

There are two main types of data formats: binary and ASCII. CITIfile uses the ASCII text format. Although this format requires more space than binary format, ASCII data is a transportable, standard type of format which is supported by all operating systems. In addition, the ASCII format is accepted by most text editors. This allows files to be created, examined, and edited easily, making CITIfile easier to test and debug.

File and Operating System Formats

CITIfile is a data storage convention designed to be independent of the operating system, and therefore may be implemented by any file system. However, transfer between file systems may sometimes be necessary. You can use any software that has the ability to transfer ASCII files between systems to transfer CITIfile data.

The descriptions and examples shown here demonstrate how CITIfile may be used to store and transfer both measurement information and data. The use of a single, common format allows data to be easily moved between instruments and computers.

CITIfile Definitions

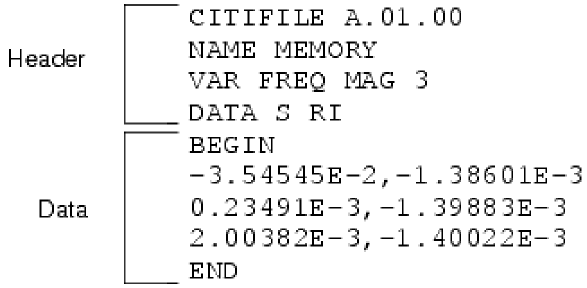
This section defines: *package* , *header* , *data array* , and *keyword* .

Package

A typical CITIfile package is divided into two parts:

- The *header* is made up of keywords and setup information.
- The *data* usually consists of one or more arrays of data.

The following example shows the basic structure of a CITIfile package:


Header
Data

```
CITIFILE A.01.00
NAME MEMORY
VAR FREQ MAG 3
DATA S RI
BEGIN
-3.54545E-2,-1.38601E-3
0.23491E-3,-1.39883E-3
2.00382E-3,-1.40022E-3
END
```

When stored in a file there may be more than one CITIfile package. With the Agilent 8510 network analyzer, for example, storing a *memory all* will save all eight of the memories held in the instrument. This results in a single file that contains eight CITIfile *packages* .

Header

The header section contains information about the data that will follow. It may also include information about the setup of the instrument that measured the data. The CITIfile header shown in the first example has the minimum of information necessary; no instrument setup information was included.

Data Array

An array is numeric data that is arranged with one data part per line. A CITIfile package may contain more than one array of data. Arrays of data start after the `BEGIN` keyword, and the `END` keyword follows the last data part in an array.

A CITIfile package does not necessarily need to include data arrays. For instance, CITIfile could be used to store the current state of an instrument. In that case the keywords VAR , BEGIN , and END would not be required.

When accessing arrays via the DAC (DataAccessComponent), the simulator requires array parts to be listed completely and in order.

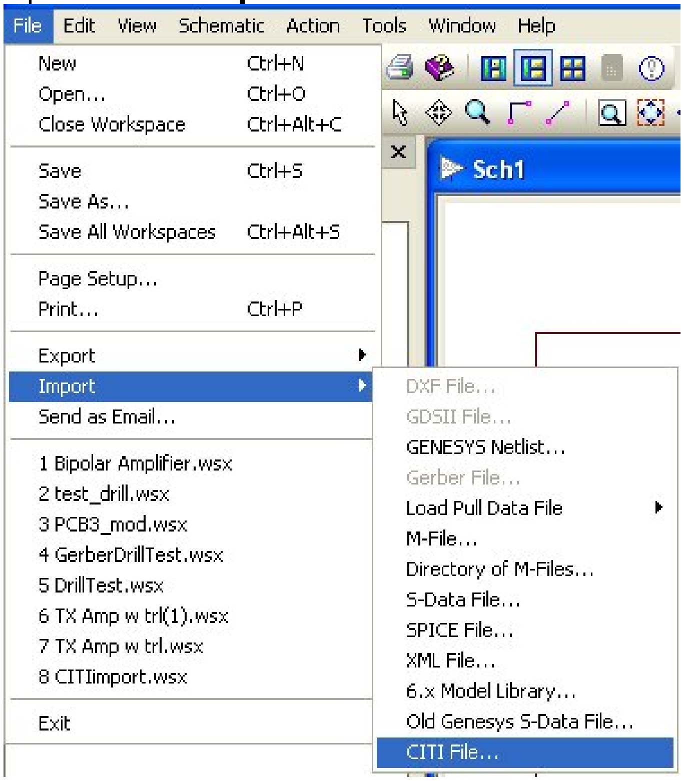
Example: S[1,1], S[1,2], S[2,1], S[2,2]

Keywords

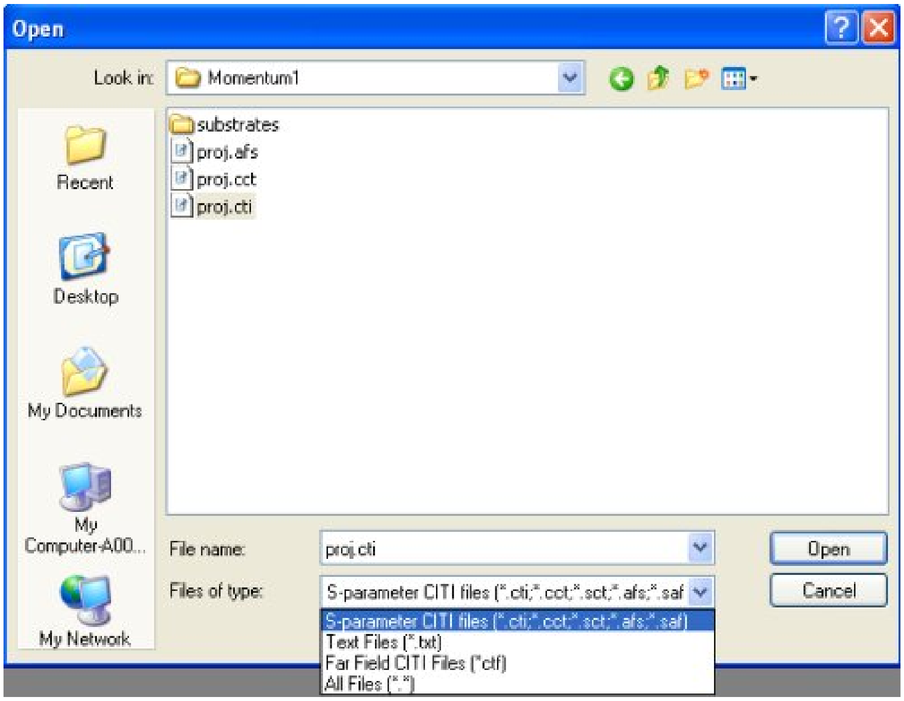
Keywords are always the first word on a new line. They are always one continuous word without embedded spaces. A listing of all the keywords used in version A.01.00 of CITIfile is shown in CITIfile Keyword Reference.

To import CITI File in Genesys:

1. Open **File > Import > CITI File..**



2. Select CITI File for Import in the **Open** file dialog:



Look in: Momentum1

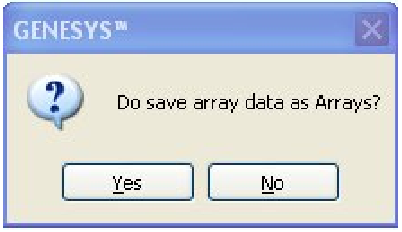
substrates
proj.afs
proj.cct
proj.cti

File name: proj.cti

Files of type: S-parameter CITI files (*.cti;*.cct;*.sct;*.afs;*.saf

S-parameter CITI files (*.cti;*.cct;*.sct;*.afs;*.saf)
Text Files (*.txt)
Far Field CITI Files (*ctf)
All Files (*.*)

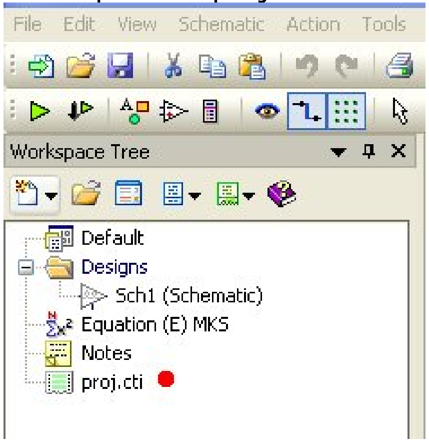
3. If the CITI file has arrays (variable, which names have indexes in square braces[]), it opens the dialog, defining output format of the array data.



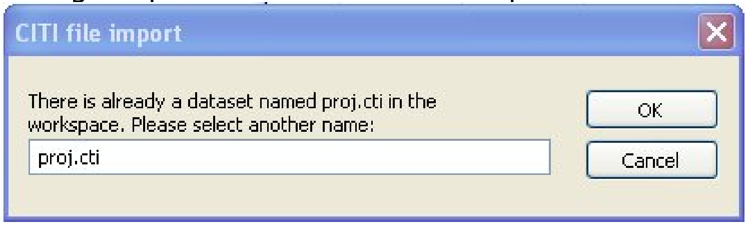
Do save array data as Arrays?

Select **Yes** to convert array data in arrays, or **No**- otherwise:

4. The imported file creates dataset with name = the file name in the Genesys workspace tree.
We imported "proj.cti" CITI file, which created same named dataset:



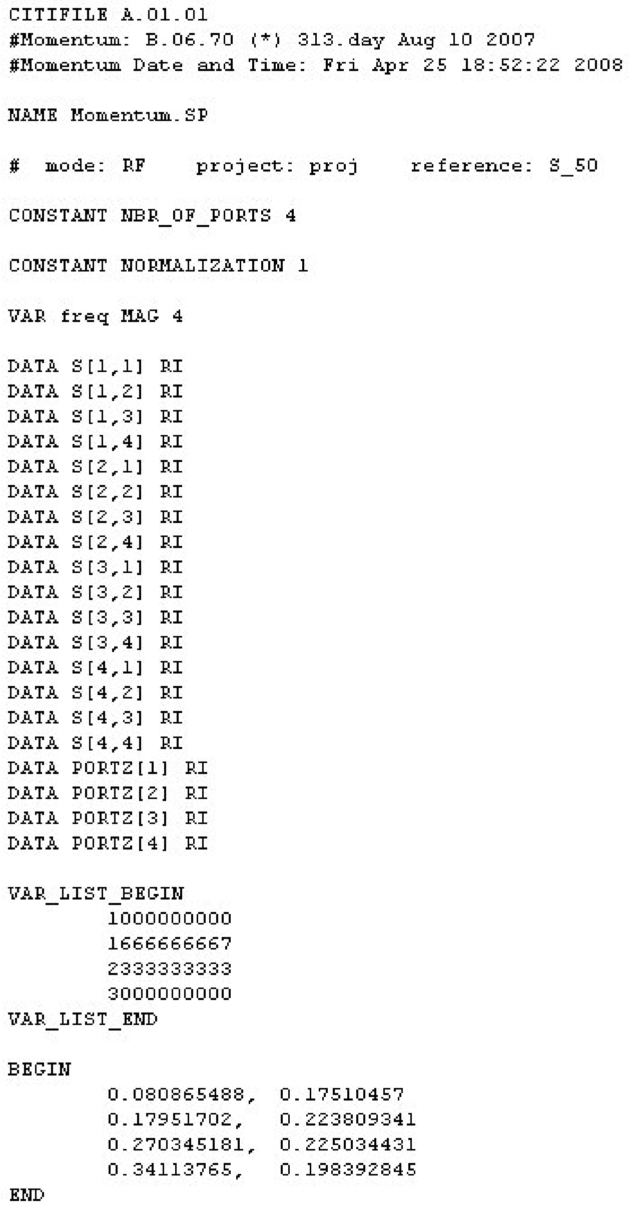
5. If the workspace tree has dataset with name of the imported CITI file, then this dialog is opened to rename the output dataset:



There is already a dataset named proj.cti in the workspace. Please select another name:

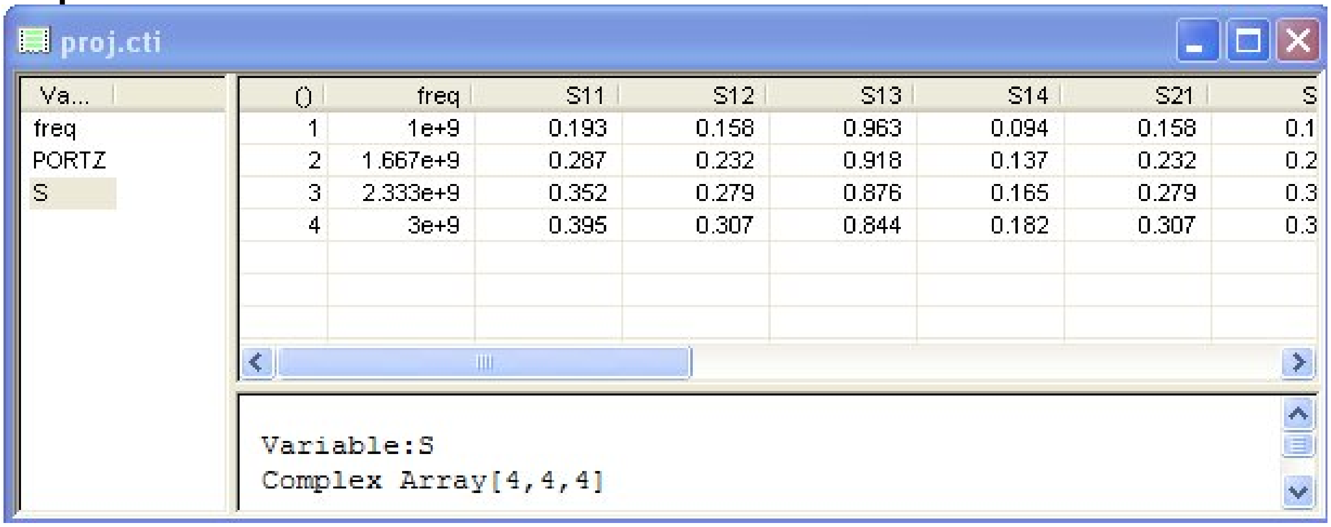
proj.cti

For example, if CITI file has array data Si,j



```
CITIFILE A.01.01
#Momentum: B.06.70 (*) 313.day Aug 10 2007
#Momentum Date and Time: Fri Apr 25 18:52:22 2008

NAME Momentum.SP

#  mode: RF    project: proj    reference: S_50

CONSTANT NBR_OF_PORTS 4

CONSTANT NORMALIZATION 1

VAR freq MAG 4

DATA S[1,1] RI
DATA S[1,2] RI
DATA S[1,3] RI
DATA S[1,4] RI
DATA S[2,1] RI
DATA S[2,2] RI
DATA S[2,3] RI
DATA S[2,4] RI
DATA S[3,1] RI
DATA S[3,2] RI
DATA S[3,3] RI
DATA S[3,4] RI
DATA S[4,1] RI
DATA S[4,2] RI
DATA S[4,3] RI
DATA S[4,4] RI
DATA PORTZ[1] RI
DATA PORTZ[2] RI
DATA PORTZ[3] RI
DATA PORTZ[4] RI

VAR_LIST_BEGIN
        1000000000
        1666666667
        2333333333
        3000000000
VAR_LIST_END

BEGIN
        0.080865488,  0.17510457
        0.17951702,   0.223809341
        0.270345181,  0.225034431
        0.34113765,   0.198392845
END
```

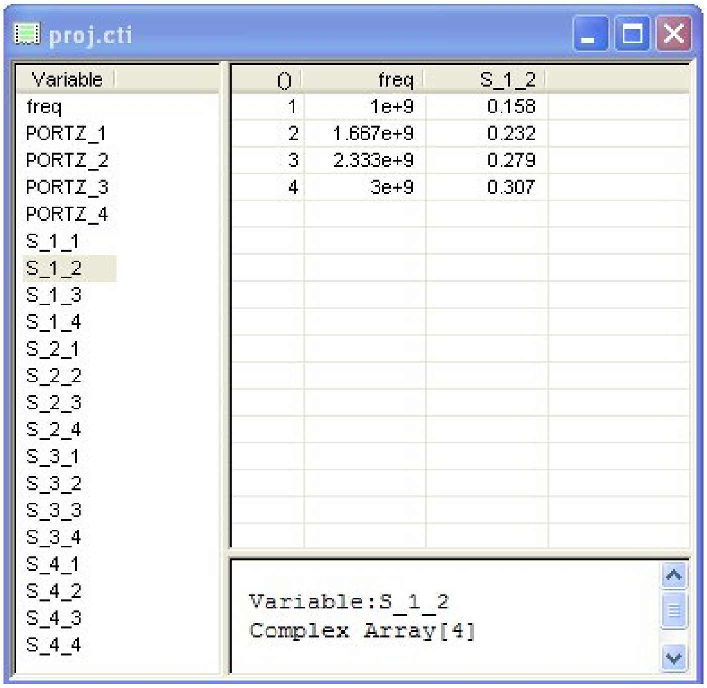
Saving its array data as Arrays will creates swept relative to independent variable **freq** vector **PORTZ** and matrix **S**:



()	freq	S11	S12	S13	S14	S21	S
1	1e+9	0.193	0.158	0.963	0.094	0.158	0.1
2	1.667e+9	0.287	0.232	0.918	0.137	0.232	0.2
3	2.333e+9	0.352	0.279	0.876	0.165	0.279	0.3
4	3e+9	0.395	0.307	0.844	0.182	0.307	0.3

```
Variable:S
Complex Array[4,4,4]
```

otherwise the array data will be saved as scalar varaiables **PORTZ_i** and **S_i_j**:



Variable
freq
PORTZ_1
PORTZ_2
PORTZ_3
PORTZ_4
S_1_1
S_1_2
S_1_3
S_1_4
S_2_1
S_2_2
S_2_3
S_2_4
S_3_1
S_3_2
S_3_3
S_3_4
S_4_1
S_4_2
S_4_3
S_4_4

()	freq	S_1_2
1	1e+9	0.158
2	1.667e+9	0.232
3	2.333e+9	0.279
4	3e+9	0.307

```
Variable:S_1_2
Complex Array[4]
```

CITIfile Examples

The following are examples of CITIfile packages.

Display Memory File

This example shows an Agilent 8510 display memory file. The file contains no frequency information. Some instruments do not keep frequency information for display memory data, so this information is not included in the CITIfile package.

Note that instrument-specific information (#NA = network analyzer information) is also stored in this file.

```
CITIFILE A.01.00
#NA VERSION HP8510B.05.00
NAME MEMORY
#NA REGISTER 1
VAR FREQ MAG 5
DATA S RI
BEGIN
-1.31189E-3,-1.47980E-3
-3.67867E-3,-0.67782E-3
-3.43990E-3,0.58746E-3
-2.70664E-4,-9.76175E-4
0.65892E-4,-9.61571E-4
END
```

Agilent 8510 Data File

This example shows an 8510 data file, a package created from the data register of an Agilent 8510 network analyzer. In this case, 10 points of real and imaginary data was stored, and frequency information was recorded in a segment list table.

```
CITIFILE A.01.00
#NA VERSION 8510B.05.00
NAME DATA
```

```
#NA REGISTER 1
VAR FREQ MAG 10
DATA S[1,1] RI
SEG_LIST_BEGIN
SEG 1000000000 4000000000 10
SEG_LIST_END
BEGIN
0.86303E-1,-8.98651E-1
8.97491E-1,3.06915E-1
-4.96887E-1,7.87323E-1
-5.65338E-1,-7.05291E-1
8.94287E-1,-4.25537E-1
1.77551E-1,8.96606E-1
-9.35028E-1,-1.10504E-1
3.69079E-1,-9.13787E-1
7.80120E-1,5.37841E-1
-7.78350E-1,5.72082E-1
END
```

Agilent 8510 3-Term Frequency List Cal Set File

This example shows an 8510 3-term frequency list cal set file. It shows how CITIfile may be used to store instrument setup information. In the case of an 8510 cal set, a limited instrument state is needed to return the instrument to the same state that it was in when the calibration was done.

Three arrays of error correction data are defined by using three `DATA` statements. Some instruments require these arrays be in the proper order, from `E[1]` to `E[3]` . In general, CITIfile implementations should strive to handle data arrays that are arranged in any order.

```
CITIFILE A.01.00
#NA VERSION 8510B.05.00
NAME CAL_SET
#NA REGISTER 1
VAR FREQ MAG 4
DATA E[1] RI
DATA E[2] RI
DATA E[3] RI
#NA SWEEP_TIME 9.999987E-2
#NA POWER1 1.0E1
#NA POWER2 1.0E1
#NA PARAMS 2
#NA CAL_TYPE 3
#NA POWER_SLOPE 0.0E0
#NA SLOPE_MODE 0
#NA TRIM_SWEEP 0
#NA SWEEP_MODE 4
#NA LOWPASS_FLAG -1
#NA FREQ_INFO 1
#NA SPAN 1000000000 3000000000 4
#NA DUPLICATES 0
#NA ARB_SEG 1000000000 1000000000 1
#NA ARB_SEG 2000000000 3000000000 3
VAR_LIST_BEGIN
1000000000
2000000000
2500000000
3000000000
VAR_LIST_END
BEGIN
1.12134E-3,1.73103E-3
4.23145E-3,-5.36775E-3
```

```
-0.56815E-3,5.32650E-3
-1.85942E-3,-4.07981E-3
END
BEGIN
2.03895E-2,-0.82674E-2
-4.21371E-2,-0.24871E-2
0.21038E-2,-3.06778E-2
1.20315E-2,5.99861E-2
END
BEGIN
4.45404E-1,4.31518E-1
8.34777E-1,-1.33056E-1
-7.09137E-1,5.58410E-1
4.84252E-1,-8.07098E-1
END
```

When an instrument's frequency list mode is used, as it was in this example, a list of frequencies is stored in the file after the `VAR_LIST_BEGIN` statement. The unsorted frequency list segments used by this instrument to create the `VAR_LIST_BEGIN` data are defined in the `#NA ARB_SEG` statements.

2-Port S-Parameter Data File

This example shows how a CITIfile can store 2-port S-parameter data. The independent variable name FREQ has two values located in the VAR_LIST_BEGIN section. The four DATA name definitions indicate there are four data arrays in the CITIfile package located in the BEGIN...END sections. The data must be in the correct order to ensure values are assigned to the intended ports. The order in this example results in data assigned to the ports as shown in the table that follows:

```
CITIFILE A.01.00
NAME BAF1
VAR FREQ MAG 2
DATA S[1,1] MAGANGLE
DATA S[1,2] MAGANGLE
DATA S[2,1] MAGANGLE
DATA S[2,2] MAGANGLE
VAR_LIST_BEGIN
1E9
2E9
VAR_LIST_END
BEGIN
0.1, 2
0.2, 3
END
BEGIN
0.3, 4
0.4, 5
END
BEGIN
0.5, 6
0.6, 7
END
BEGIN
0.7, 8
0.8, 9
END
```

DATA	FREQ = 1E9	FREQ = 2E9
s[1,1]	s[0.1,2]	s[0.2,3]
s[1,2]	s[0.3,4]	s[0.4,5]
s[2,1]	s[0.5,6]	s[0.6,7]
s[2,2]	s[0.7,8]	s[0.8,9]

CITIfile Keyword Reference

The following table lists keywords, definitions, and examples.

h7. CITIfile Keywords and Definitions

Keyword	Example and Explanation
CITIFILE	Example: CITIFILE A.01.00 Identifies the file as a CITIfile and indicates the revision level of the file. The CITIFILE keyword and revision code must precede any other keywords. The CITIFILE keyword at the beginning of the package assures the device reading the file that the data that follows is in the CITIfile format.The revision number allows for future extensions of the CITIfile standard. The revision code shown here following the CITIFILE keyword indicates that the machine writing this file is using the A.01.00 version of CITIfile as defined here. Any future extensions of CITIfile will increment the revision code.
NAME	Example: NAME CAL_SET Sets the current CITIfile package name. The package name should be a single word with no embedded spaces. Some standard package names: RAW_DATA : Uncorrected data. DATA: Data that has been error corrected. When only a single data array exists, it should be named DATA . CAL_SET: Coefficients used for error correction. CAL_KIT: Description of the standards used. DELAY_TABLE: Delay coefficients for calibration.
VAR	Example: VAR FREQ MAG 201 Defines the name of the independent variable (FREQ); the format of values in a VAR_LIST_BEGIN table (MAG) if used; and the number of data points (201).
CONSTANT	Example: CONSTANT *name value* Lets you record values that do not change when the independent variable changes.
#	Example: #NA POWER1 1.0E1 Lets you define variables specific to a particular type of device. The pound sign (#) tells the device reading the file that the following variable is for a particular device. The device identifier shown here (NA) indicates that the information is for a network analyzer. This convention lets you define new devices without fear of conflict with keywords for previously defined devices. The device identifier can be any number of characters.
SEG_LIST_BEGIN	Indicates that a list of segments for the independent variable follows. Segment Format: *segment type start stop number of points* The current implementation supports only a signal segment. If you use more than one segment, use the VAR_LIST_BEGIN construct. CITIfile revision A.01.00 supports only the SEG (linear segment) segment type.
SEG_LIST_END	Sets the end of a list of independent variable segments.
VAR_LIST_BEGIN	Indicates that a list of the values for the independent variable (declared in the VAR statement) follows. Only the MAG format is supported in revision A.01.00.
VAR_LIST_END	Sets the end of a list of values for the independent variable.
DATA	Example: DATA S[1,1] RI Defines the name of an array of data that will be read later in the current CITIfile *package* , and the format that the data will be in. Multiple arrays of data are supported by using standard array indexing as shown above. CITIfile revision A.01.00 supports only the RI (real and imaginary) format, and a maximum of two array indexes. Commonly used array names include: S - S parameter E - Error Term Voltage - Voltage VOLTAGE_RATIO - a ratio of two voltages (A/R)

CITIfile Guidelines

The following general guidelines aid in making CITIfiles universally transportable:

Line Length. The length of a line within a CITIfile package should not exceed 80 characters. This allows instruments which may have limited RAM to define a reasonable input buffer length.

Keywords. Keywords are always at the beginning of a new line. The end of a line is as defined by the file system or transfer mechanism being used.

Unrecognized Keywords. When reading a CITIfile, unrecognized keywords should be ignored. There are two reasons for this:

- Ignoring unknown keywords allows new keywords to be added, without affecting an older program or instrument that might not use the new keywords. The older instrument or program can still use the rest of the data in the CITIfile as it did before. Ignoring unknown keywords allows "backwards compatibility" to be maintained.
- Keywords intended for other instruments or devices can be added to the same file without affecting the reading of the data.

Adding New Devices. Individual users are allowed to create their own device keywords through the # (user-defined device) mechanism. (Refer to the table immediately above for more information.) Individual users should *not* add keywords to CITIfiles without using the # notation, as this could make their files incompatible with current or future CITIfile implementations.

File Names. Some instruments or programs identify a particular type of file by characters that are added before or after the file name. Creating a file with a particular prefix or ending is not a problem. However in general an instrument or program should not require any such characters when *reading* a file. This allows any file, no matter what the filename, to be read into the instrument or computer. Requiring special filename prefixes and endings makes the exchange of data between different instruments and computers much more difficult.

A CITIfile package is as described in the main CITIfile documentation: the CITIFILE keyword, followed by a header section, usually followed by one or more arrays of data.

> **Note**
> There are some specific problems with the current version in reading and/or writing this data format. On the Agilent EEsof web site, refer to the Release Notes in Product Documentation, and to Technical Support for more information and workarounds (http://www.agilent.com/find/eesof).

Generic MDIF Format

The generic MDIF provides a generalized MDIF format for unifying the various specific MDIF formats, and overcoming some limitations of other formats. The generic format enables diverse applications to use a common data I/O interface, so long as the intent is to access/save multidimensional (multiple independent vs dependent variables) data.

The general format is as follows:

```
VAR var1Name(var1Type) = var1
ValueVAR var2Name(var2Type) = var2Value
..
VAR varNName(varNType) = varNValue
BEGIN blockName
% bVar1Name(bVar1Type) bVar2Name(bVar2Type) ....
% bVarLName(bVarLType) ...
% ...
% bVarQName(bVarQType) ... bVarPName(bVarPType)
```

```
bVar1Value bVar2Value ...
bVarLValue ..
..
bVarQValue ... bVarPValue
bVar1Value bVar2Value ...
bVarLValue ..
..
bVarQValue ... bVarPValue
...
END
```

where `var*Type` can be the token:

> 0 or int
> 1 or real
> 2 or string

Type bVar*Type can be one of the above as well as:

> 3 or complex
> 4 or boolean
> 5 or binary
> 6 or octal
> 7 or hexadecimal
> 8 or byte16

The variable names above constitute a name-space uniquely identified by the string blockName which is either:

- alphanumeric: all bVar*Name block variables are dependent, except bVar1Name, which is usually the most rapidly changing (innermost) independent variable.
  or
- DSCR(blockName): all bVar*Name block variables are dependent, and there is an indexing implicit independent variable.

Guidelines

- A string type variable's value must be surrounded by "".

- If there are multiple blocks, the outermost independent variables (e.g., `VAR var1Name(var1Type) = var1`) apply only to the block immediately following the variable definitions, and not to any other blocks.

- The block data (bVar*Value) lines must follow the pattern (order, number of values per line, and number of lines) of the format (%) lines. If the number of values in any data line does not match the number of dependent variables specified in the corresponding format (%) line, incorrect results will occur. A variable's value cannot be split across lines. Although there is no line length limit specified, MDIF file readers may choose to truncate at some finite length. This may result in a file read error, or, if the file was carefully crafted, truncated names and/or string-type values.

- Scale factors, which can be applied only to real numbers, may be case-insensitive suffixes as follows:

  > f = 1e-15, p = 1e-12, n = 1e-9, u = 1e-6, mil = 2.54e-5, m = 1e-3,
  >
  > k = 1e3, g = 1e9, t = 1e12

E.g.: 15mA = 15e-3, 30KHz = 30e3

There should be no space between the number and the suffix, and extra characters are ignored. Unrecognized suffixes result in 1.0. The above is not totally consistent with the rest of ADS.

- The format of complex data is real/imag, with a column for real and a column for imaginary.
- Multidimensional data is organized by outer to inner independent variables. VAR statements go from outermost to innermost.
- Vary innermost independent variables first, proceeding toward outermost variables changing last.
- Independent variables should change monotonically.

Example

```
!=============================================================
! Example 1
REM This has 3 indepVars: v1, v2, v3(innermost) and
REM 4 depVars: dv1(integer), dv2(real), dv3(string) and
REM dv4(hexadecimal), but is read in as a string.
REM The outermost indepVars: v1, v2 apply only to the block
REM immediately following them, and not to any other block.
! There are 2 data nodes
VAR v1(0) = 1
VAR v2(1) = 2.2
BEGIN blk1
% v3(1) dv1(1) dv2(1) dv3(2) dv4(hexadecimal)
7.7 8 9.9999 "line 1" 0xabc
8.8 9 1.11 "line 2 " 0x123
END
VAR v1(0) = 2
VAR v2(1) = 3.2
BEGIN blk1
% v3(1) dv1(1) dv2(1) dv3(2) dv4(hexadecimal)
8.7 9uF 10.9999mA "line 1" 0xff
9.8 10uF 11.11mA "line 2 " 0xdef
END
!==============================================================
! Example 2
! Created Tue Mar 9 13:39:19 1999
! Data Acquired Tue Mar 9 13:38:34 1999
BEGIN NDATA_noise
% freq(real) Sopt(complex) NFmin(real) Rn(real) PortZ[1](real)
    1e+09     0.098481     0.017365       1          5    50
    2e+09      0.18794     0.068404       2         10    50
    3e+09      0.25981         0.15       3         15    50
    4e+09      0.30642      0.25712       4         20    50
    5e+09      0.32139      0.38302       5         25    50
    6e+09          0.3      0.51962       6         30    50
    7e+09      0.23941      0.65778       7         35    50
    8e+09      0.13892      0.78785       8         40    50
9.543e+09    -0.014122        0.911  9.5445     46.166    50
END
```

X-parameter GMDIF Format

This section describes:

- Choosing an X-parameter file for use with an X-Parameter part
- An overview of the X-parameter file
- Examples of various details in X-parameter files

Overview

These files contain X-parameter data for nonlinear n-port devices, or subcircuits. They are ASCII files in GMDIF format. They use extension: .xnp.
The X-parameter files completely comply by Generic MDIF Format. The specific block and variable names used in the X-parameter GMDIF files are described in this section.
This section describes Version 2.0 X-parameter GMDIF files.
An X-parameter GMDIF file can be used with an X-Parameter part to model the behavior of a nonlinear device or subcircuit using X-parameters. The file contains the X-parameters, the part is placed within the schematic.

Linking an X-parameters GMDIF File to an X-parameters Part

To link a file to the part:

1. Add X-parameters part to your schematic. It can be found in the **RF Design** library.
2. Set up the X-parameters parameters. For instructions on how to set the parameters, click Model Help in the part's dialog box.

Comments

GMDIF files support comments in two ways:

- by using "!" or
- by using "REM" statement.

The "!" can be used in the beginning of a line, or at the end of the line where as, "REM" can be used only in the beginning of a line.
Version 2.0 X-parameter GMDIF files contain a pre-defined comment section at the beginning of the files, which provides useful information about the range of operating conditions covered by the data as shown in the example below:

Example

```
! Created Fri Jul 10 15:29:17 2009
! Version = 2.0
! HB_MaxOrder = 9
! XParamMaxOrder = 3
! NumExtractedPorts = 3
! fund_1=[1e+09->1.4e+09]   NumPts=5
! VDC_3=[10->11]   NumPts=2
! ZM_2_1=50   NumPts=1
! ZP_2_1=0   NumPts=1
! AN_1_1=[3.16228e-03(-20.000000dBm)->70.7107e-03(6.989700dBm)]   NumPts=36
```

The version of the file is stated just for convenience. The statement determining the version is elsewhere. The comment "HB_MaxOrder = 9" tells you that the Harmonic

Balance with MaxOrder=9 was used by X-Parameter Generator. The comment "XParamMaxOrder = 3" tells you that the X-parameter data in this file contains mixing indices up to the 3rd order.
The comment "NumExtractedPorts = 3" indicates the total number of ports used for X-parameter generation. In case of non-consecutive port numbering this value may be smaller than the highest port number.
The lower part of this comment section indicates various independent variables together with the covered sweeps for each of them. See *X-parameter Independent Variables* (users) for explanation of the variable names.

X-parameter GMDIF File Blocks

Version 2.0 of X-parameter GMDIF files contains three types of blocks:

- XParamAttributes
- XParamPortData
- XParamData

The first two blocks appear only once in the file. The third block appears as many times as the number of distinct different sweep points present in the data for all but the innermost independent variable. The following sections provide details for these blocks.

XParamAttributes Block

The **XParamAttributes** block provides the vehicle for the official statements of (1) the file version, (2) the number of ports, and (3) the number of fundamental frequencies (tones).

Example

```
BEGIN XParamAttributes
% Index(int)      Version(real)      NumPorts(int)      NumFundFreqs(int)
 0               2.0                3                  1
END
```

The sole purpose of the Index column is compliance with the Generic MDIF format.
The **NumPorts** entry indicates the highest port index in the data.

XParamPortData Block

The XParamPortData block provides reference impedances for the incident and reflected waves at each port covered by the data. The reference impedances can be complex and the power definition of the waves is used, as follows:

$$a_p = \frac{V_p + Z_p \cdot I_p}{\sqrt{8\,\mathrm{Re}(Z_p)}} \qquad\qquad b_p = \frac{V_p - Z_p^* \cdot I_p}{\sqrt{8\,\mathrm{Re}(Z_p)}}$$

In the above equations, Vp and Ip represent amplitude phasors.

Example

```
BEGIN XParamPortData
% PortNumber(int)   RefZ0(complex)                PortName(string)
 1                  50              0             "Input"
 2                  50              0             "Output"
 3                  50             0              "VDC"
END
```

The **XParamPortData** block also includes the port names. This information is particularly useful in proper hookup of the **X-parameters part** in cases where more than two ports are present and a mixture of port types is used.

XParamData Block

The **XParamData** block provides the actual X-parameters. This block may appear many times in the file, each containing X-parameters at one sweep point (of all but the innermost independent variable) at a time.
Each **XParamData** block is preceded by m-1 VAR statements for m-1 independent variables, where m is the total number of independent variables. These VAR statements provide the types and the values of the independent variables. These values apply to the **XParamData** block immediately following the VAR statements, and only to that block.

Example

```
VAR fund_1(real) = 1e+09
VAR VDC_3(real) = 10
VAR ZM_2_1(real) = 50
VAR ZP_2_1(real) = 0
BEGIN XParamData
% AN_1_1(real)  FI_3(real)  FB_1_1(complex)  ...
...
...
...
END
```

The last, mth, independent variable is the innermost variable and is placed as the first variable inside the block. In the above example that variable is "AN_1_1".
The naming convention for the independent variables in X-parameter files is described in *X-parameter Independent Variables* (users).
All the dependent variables (the X-parameters) are provided inside the block. Following the mth independent variable, the names and the types of the dependent variables are specified in the header lines (lines starting with a "%" character). The header lines are specified once per block at the beginning of the block. They are then followed by as many data groups as the number of sweep points of the innermost independent variable. Each group consists of data values formatted into lines exactly in the same way as the block header lines with each entry representing a value of the correspondingly placed variable in the header lines. Complex data is specified in the rectangular format (real, imaginary) by two numbers.

Example

```
VAR fund_1(real) = 1e+09
VAR VDC_3(real) = 10
VAR ZM_2_1(real) = 50
VAR ZP_2_1(real) = 0
BEGIN XParamData
% AN_1_1(real)  FI_3(real)  FB_2_1(complex)  S_1_2_2_2(complex)
0.0657          -0.32       0.113    1.01    0.222    -0.0031
0.0667          -0.33       0.111    1.02    0.222    -0.0034
0.0677          -0.34       0.110    1.05    0.222    -0.0039
END
```

In the above example the complex number (0.111 + j1.02) is the value of the dependent

variable FB_2_1 at the multidimensional point established by all the values of the independent variables, including the value of 0.0667 of AN_1_1.
The naming convention for the dependent variables in X-parameter files is described in *X-parameter Dependent Variables* (users).

X-parameter Variables

Notation

All independent and dependent variables are defined with respect to port and harmonic (or mixing) indices. For each variable these indices, separated by the underscore character "*", form a string appending the reserved name of the variable. Negative indices, if allowed, are represented by a string in which the "m" character is used in place of the minus ("-") sign, with no space between the sign and the number. For example "_m2*" represents the index "-2". For clarity of presentation the following table shows the notation used in indexing the X-parameters.

k	fundamental frequency index; 1 in the case of single tone X-parameters;all consecutive numbers must be present
p	port index - a positive integer; may not be consecutive pIn - denotes the "input" port index pOut - denotes the "output" port index
n	harmonic index; positive integer nIn - denotes the harmonic on the "input" port nOut - denotes the harmonic on the "output" port in case of multi-tone X-parameters there is a mixing index that is concatenated from harmonic indices w.r.t. to subsequent fundamentals, for example "_1_m2_2" in the three-tone case refers to the mixing product f1-2f2+2f3 - the index w.r.t. the first fundamental is expected to be non-negative and all-zero entries are not allowed.

Independent Variables

The following table lists all the supported independent variables in Version 2.0 X-parameter files. In general, all X-parameters are functions of some or all of these independent variables. Their dependence is tabulated in the X-parameter files for all sweep points of the independent variable values.
All independent variables are real numbers.

fund_k	kth fundamental frequency; assumed non-commensurate if more than one is present; fund_1 is required
VDC_p	DC voltage applied to port p; not required; mutually exclusive with *IDC_p* at the same port p
IDC_p	DC current applied to port p; not required; mutually exclusive with *VDC_p* at the same port p
AN_p_n	magnitude of a large-signal incident wave applied to port p at harmonic n; only one per each fundamental is both allowed and required; phase of this incident wave is not tabulated in the X-parameter files as this incident wave serves as a Reference Signal (Refer to **ADS document** for detailed description); power definition of incident waves is used
AM_p_n	magnitude of any other than Reference Signal large-signal incident wave applied to port p at harmonic n; required only if *AP_p_n* is used at the same port p and harmonic n; power definition of incident waves is used
AP_p_n	phase in degrees of any other than Reference Signal large-signal incident wave applied to port p at harmonic n; required only if *AM_p_n* is used at the same port p and harmonic n
GM_p_n	magnitude of the reflection coefficient of the load at port p and harmonic n; required only if *GP_p_n* is used at the same port p and harmonic n; power definition of the reflection coefficient and the reference impedance specified for port p are used; mutually exclusive with other formats of specifying load at the same port p and harmonic n
GP_p_n	phase in degrees of the reflection coefficient of the load at port p and harmonic n; required only if *GM_p_n* is used at the same port p and harmonic n; mutually exclusive with other formats of specifying load at the same port p and harmonic n
GX_p_n *GY_p_n*	alternative to *GM_p_n* and *GP_p_n*; real and imaginary parts of the reflection coefficient; mutually exclusive with other formats of specifying load at the same port p and harmonic n
ZM_p_n *ZP_p_n*	alternative to *GM_p_n* and *GP_p_n*; magnitude and phase of the load impedance; mutually exclusive with other formats of specifying load at the same port p and harmonic n
ZX_p_n *ZY_p_n*	alternative to *GM_p_n* and *GP_p_n*; real and imaginary parts of the load impedance; mutually exclusive with other formats of specifying load at the same port p and harmonic n

Dependent Variables

The following table provides the notation for the dependent variables (X-parameters) used in Version 2.0 X-parameter files. The X-parameters can be either real or complex numbers. In the latter case the rectangular format (real and imaginary parts) is used. It is not essential for any specific dependent variable to be present in an X-parameter file. In general, the default value is zero for any absent parameter that could otherwise be included in the file (some parameters are mutually exclusive with some other parameters).

FB_pOut_nOut	complex	B-type X-parameter - measured reflected wave at output port pOut and harmonic nOut as the response to all large-signal excitations (i.e., under the large-signal operating conditions); power definition of the reflected waves is used
FI_pOut	real	I-type X-parameter - DC current measured at output port pOut under the large-signal operating conditions
FV_pOut	real	V-type X-parameter - DC voltage measured at output port pOut under the large-signal operating conditions
S_pOut_nOut_pIn_nIn	complex	S-type X-parameter providing the small-signal added-contribution to the reflected wave at output port pOut and harmonic nOut due to a small-signal incident wave at input port pIn and harmonic nIn measured under the large-signal operating conditions; power definition of the incident and reflected waves is used
T_pOut_nOut_pIn_nIn	complex	T-type X-parameter providing the small-signal added-contribution to the reflected wave at output port pOut and harmonic nOut due to a phase-reversed small-signal incident wave at input port pIn and harmonic nIn measured under the large-signal operating conditions; power definition of the incident and reflected waves is used
XY_pOut_pIn_nIn	complex	Y-type X-parameter providing the small-signal contribution to the DC current at output port pOut due to a small-signal incident wave at input port pIn and harmonic nIn measured under the large-signal operating conditions; power definition of the incident waves is used; the real-valued contribution to the DC current is the real part of complex product of this X-parameter and the corresponding incident wave
Yre_pOut_pIn_nIn *Yim_pOut_pIn_nIn*	real real	alternative to XY_p_n, obsolete in Version 2.0 X-parameter files; two real numbers: the real part and negative of the imaginary part are provided instead of one complex number, as XY = Yre - j*Yim
XZ_pOut_pIn_nIn	complex	Z-type X-parameter providing the small-signal contribution to the DC voltage at output port pOut due to a small-signal incident wave at input port pIn and harmonic nIn measured under the large-signal operating conditions; power definition of the incident waves is used; the real-valued contribution to the DC voltage is the real part of complex product of this X-parameter and the corresponding incident wave
Zre_pOut_pIn_nIn *Zim_pOut_pIn_nIn*	real real	alternative to XZ_p_n, obsolete in Version 2.0 X-parameter files; two real numbers: the real part and negative of the imaginary part are provided instead of one complex number, as XZ = Zre - j*Zim

Restrictions

If the independent variable *VDC_pOut* is specified for the port *pOut* then neither the V-type (*FV_pOut*) nor the Z-type (*XZ_pOut_pIn_nIn*, *Zre_pOut_pIn_nIn*, *Zim_pOut_pIn_nIn*) X-parameters can be specified for the port *pOut*.
Similarly, if the independent variable *IDC_pOut* is specified for the port *pOut* then neither the I-type (*FI_pOut*) nor the Y-type (*XY_pOut_pIn_nIn*, *Yre_pOut_pIn_nIn*, *Yim_pOut_pIn_nIn*) X-parameters can be specified for the port *pOut*.

Exporting Files Using Genesys

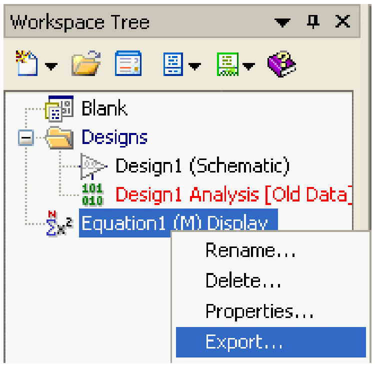
Genesys can export the following file types:

- ASCII Drill List
- Export Schematics to ADS
- Bill of Materials
- Bitmap (Active Window)
- Bitmap (Entire Screen)
- DXF/DWG File
- GDSII File
- Gerber File
- HPGL File

- IFF Schematic File
- Part Placement List
- S-Parameters
- SPICE File
- Touchstone File
- XML File

Export a file

- To export a file please follow the following steps detailed below:
  - Select the object in the Workspace Tree to be exported.
  - Click **File** on the Genesys menu and select a file type from the **Export** menu.
  - Follow the instructions in the windows that appear.
    Or
  - Right click on the object in the Workspace Tree to be exported.
  - Select the **Export** menu.
  - Follow the instructions in the windows that appear.



ASCII Drill List Export

Schematics to ADS Export

Bill of Materials Export

Bitmap (Active Window) Export

A bitmap of the active window can be exported. To export the active window:

1. Open the window and make sure it is the active window
2. Select the File > Export > Bitmap (Active Window) menu
3. When prompted specify the directory and filename

Bitmap (Entire Screen) Export

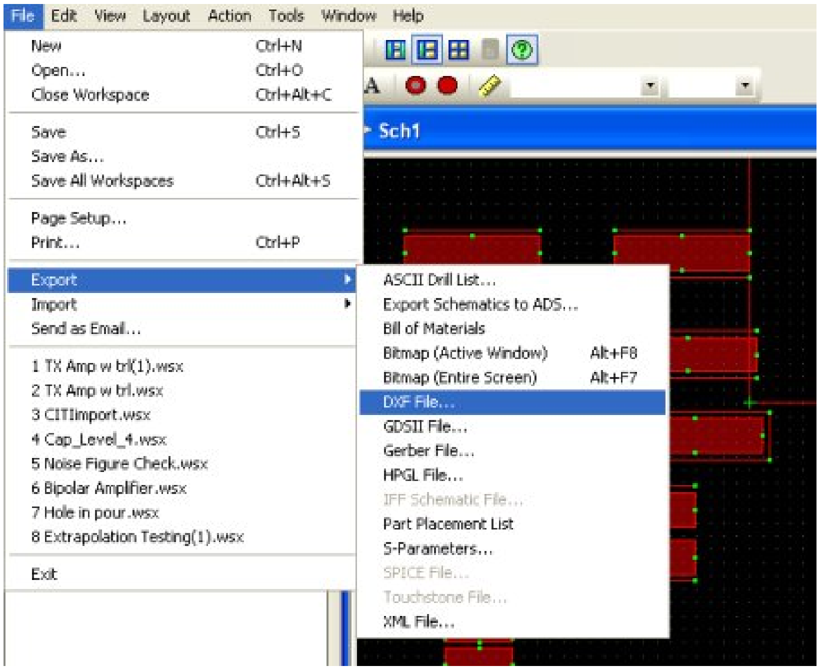
A bitmap of the entire screen can be exported. To export the entire screen:

1. Select the File > Export > Bitmap (Entire Screen) menu
2. When prompted specify the directory and filename

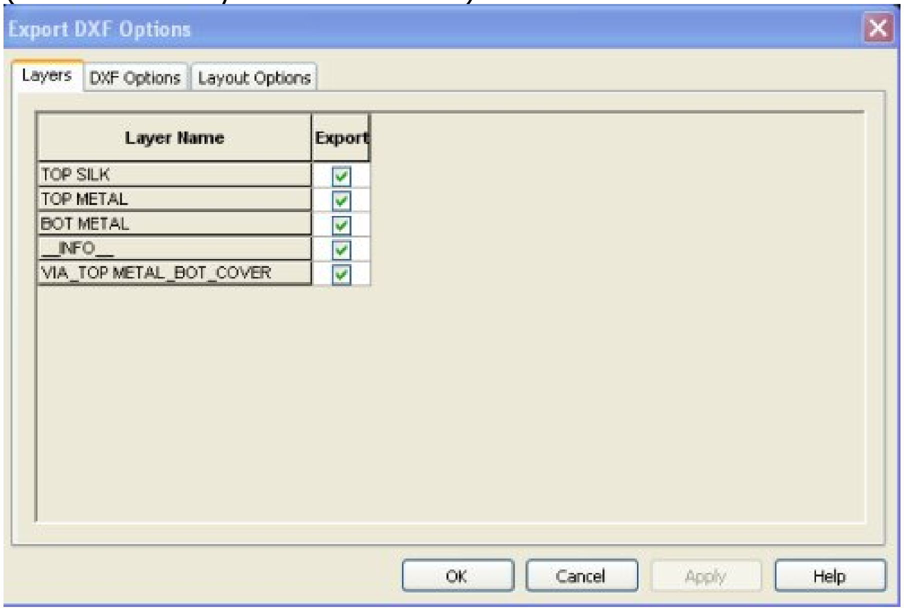
DXF/DWG File Export

The *Export DXF/DWG Options* dialog box enables you to set file format, Autocad version, path geometry, and hole format.

- To export AutoCAD DXF/DWG files:
  - In the layout window containing your design, choose **File > Export > DXF File...**

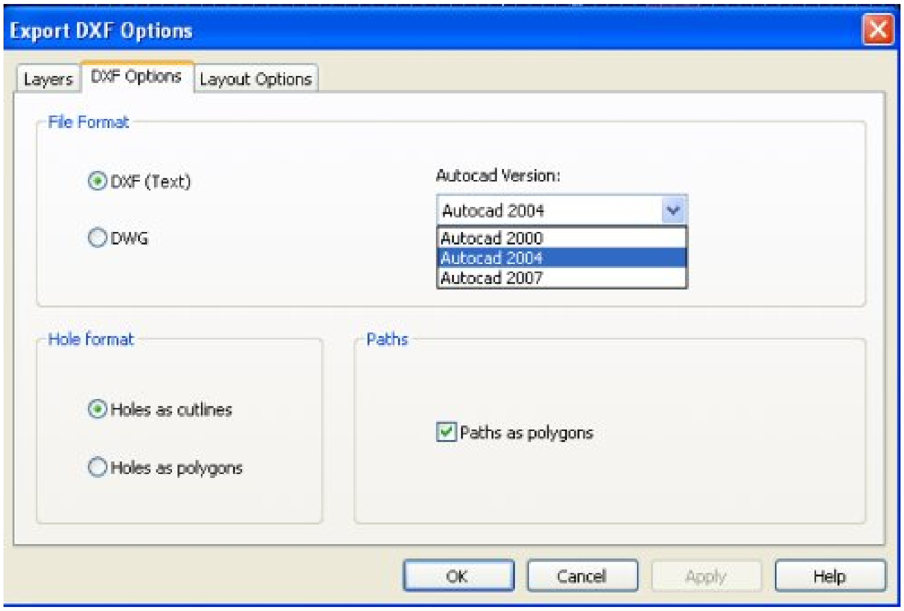


  - It opens **Export DXF Options** dialog. Select exported layers in the **Layers** tab (default - all layers are selected)



Layer Name	Export
TOP SILK	☑
TOP METAL	☑
BOT METAL	☑
__INFO__	☑
VIA_TOP METAL_BOT_COVER	☑

  - Select exporting **DXF Options**: output *File Format* DXF(Text) or DWG(Binary), AutoCAD version, format of *Holes*, and format pf *Paths*.



Export DXF Options

Layers | DXF Options | Layout Options

File Format

● DXF (Text)

○ DWG

Autocad Version: Autocad 2004

Autocad 2000
Autocad 2004
Autocad 2007

Hole format

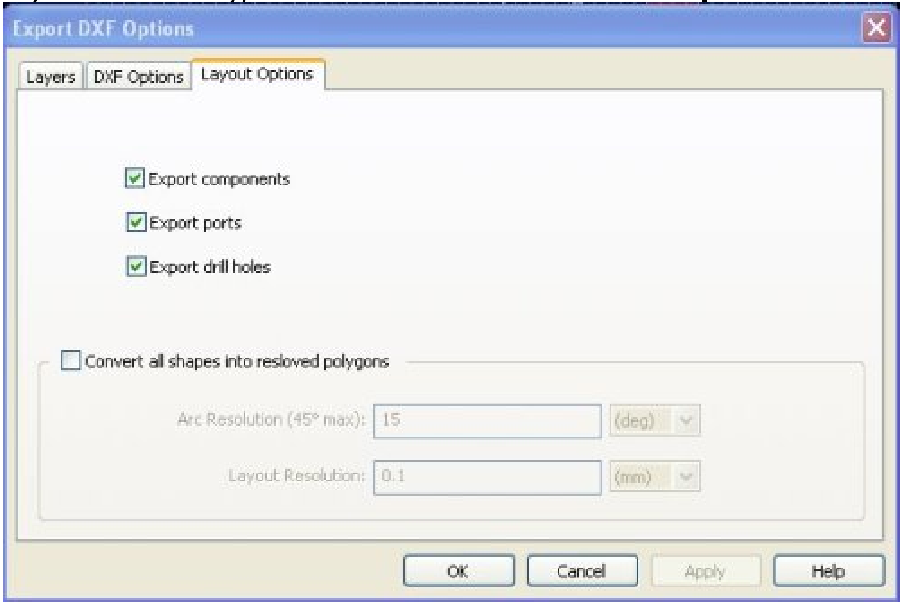
● Holes as outlines

○ Holes as polygons

Paths

☑ Paths as polygons

- At the **Layout Options** tab define exporting Layout objects (components, EM-ports, drill holes). To convert all layout objects to resolved polygons (excepting layers with text), set checkbox **Convert all shapes into resolved polygons.**



Export DXF Options

Layers | DXF Options | Layout Options

☑ Export components

☑ Export ports

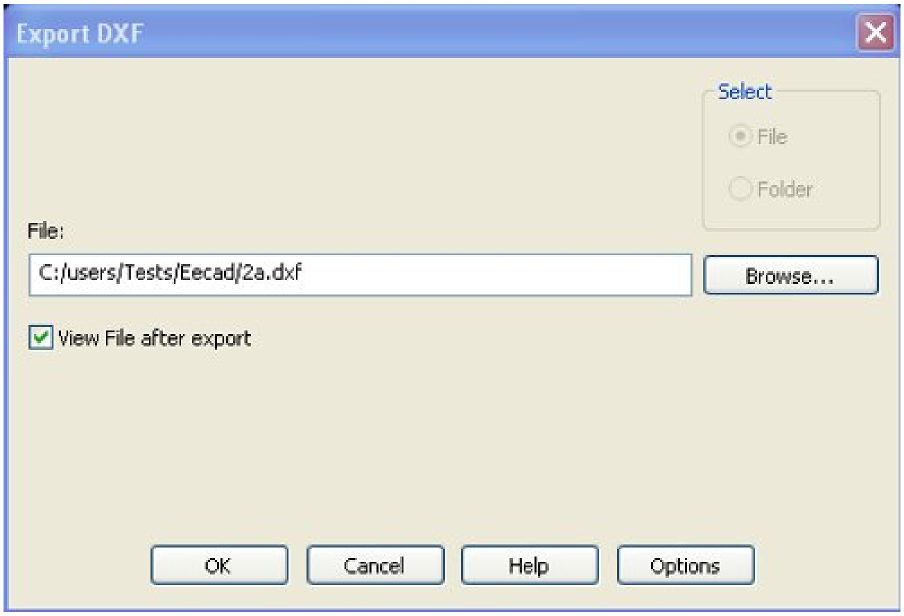
☑ Export drill holes

□ Convert all shapes into reslolved polygons

Arc Resolution (45° max): 15 (deg)

Layout Resolution: 0.1 (mm)

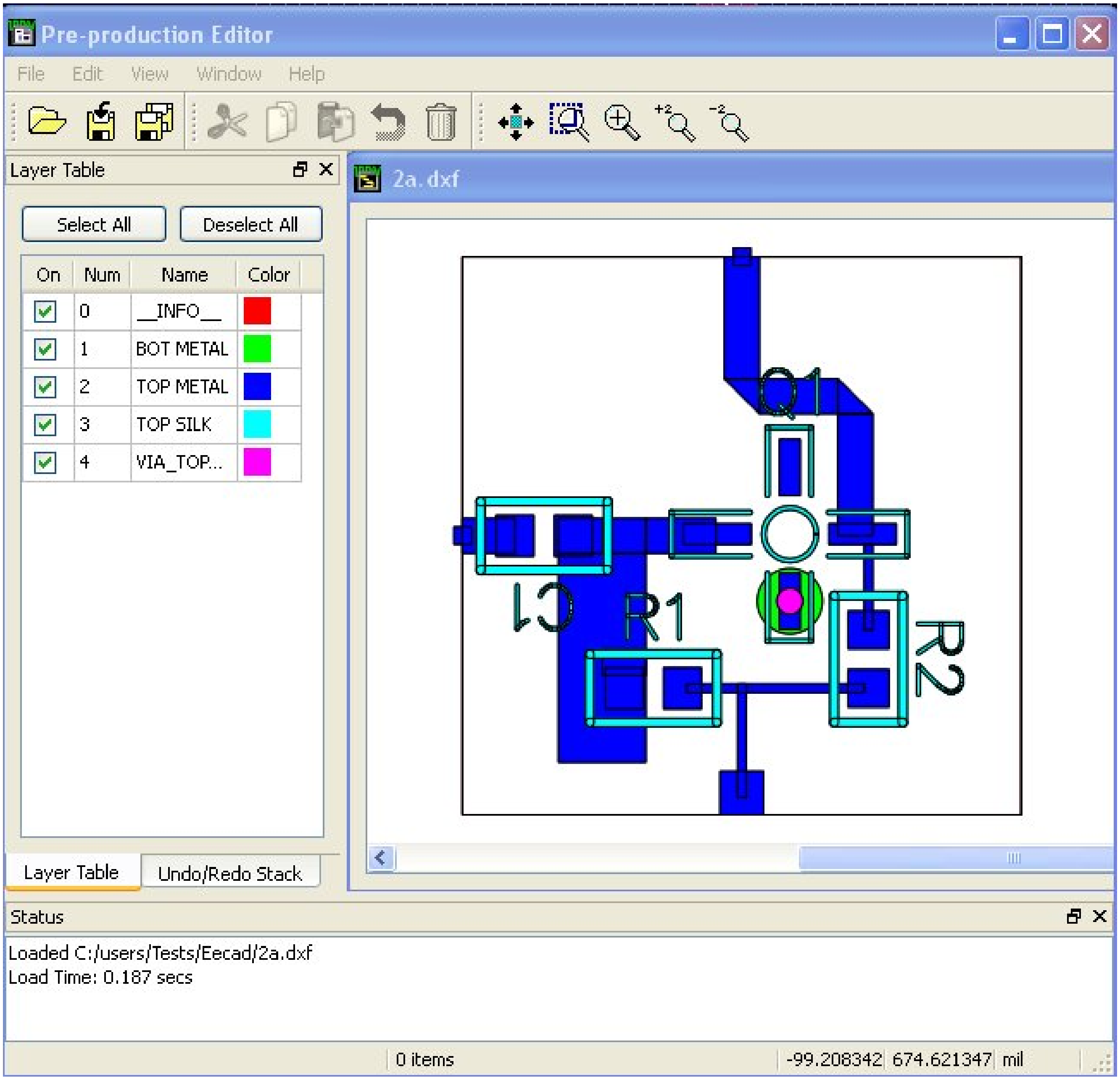
- Close the dialog by **OK** button opens **Export DXF** (or **Export DWG**) file opening dialog. Enter full path to output file manually, or using **Browse** button. Select box **View File after export** to view output file after export.



File: C:/users/Tests/Eecad/2a.dxf

☑ View File after export

- Push **OK** button. After export it opens Preproduction editor, displaying results of the export:

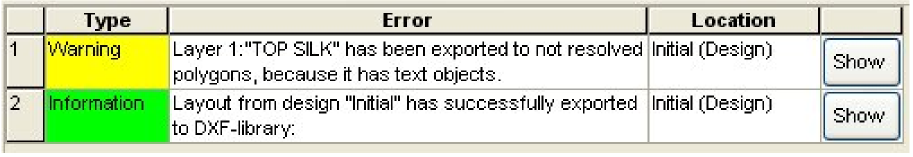


On	Num	Name	Color
☑	0	__INFO__	🟥
☑	1	BOT METAL	🟩
☑	2	TOP METAL	🟦
☑	3	TOP SILK	■
☑	4	VIA_TOP...	🟪

```
Loaded C:/users/Tests/Eecad/2a.dxf
Load Time: 0.187 secs
```

```
0 items        -99.208342  674.621347  mil
```

Genesys Error/Messages window displays results of the export:



	Type	Error	Location	
1	Warning	Layer 1:"TOP SILK" has been exported to not resolved polygons, because it has text objects.	Initial (Design)	Show
2	Information	Layout from design "Initial" has successfully exported to DXF-library:	Initial (Design)	Show

GDSII File Export

Gerber File Export

HPGL File Export

IFF Schematic File Export

Export any schematic created in Genesys to another program using Interchange File Format (IFF). This lets you use your schematic in programs other than Genesys, such as ADS.

Exporting IFF Files That are ADS Compatible

Only schematics are supported for export to IFF files.

Note: Schematics that you manually create should import into ADS. If you have any compatibility issues, please contact Agilent directly.

To export a schematic to an IFF file:

1. Open a schematic.
2. Click **File** on the Genesys menu and select*IFF Schematic File* from the Export menu.
3. **Save** the file in .iff format.
4. Now **open** the IFF file in ADS.

Translating Direct Parts to IFF

When a part in Genesys is the same as a part in ADS, Genesys uses that ADS part. Parts are considered the same if:

- The part symbols are exactly the same size.
- The parameters are compatible. The parameter names do not have to match exactly, but the parameters from Genesys must be convertible with a simple mapping table or formulas.

For example, a resistor maps to IFF because the standard resistor size is one inch and the parameter R (resistance) has a standard meaning.

Translating Custom Parts to IFF

If a part does not directly translate to IFF, Genesys creates a custom part and symbol in ADS. This creates an additional network in ADS, and possibly a netlist file, to support the new part. These custom networks use a standard naming convention so that your ADS designs are easily shared and edited.

Translating DisCos to IFF

While DisCos are not directly supported in ADS, Genesys recreates these parts with the additional terminals necessary for ADS compatibility. All DisCos work in ADS without manual modifications.

Exporting Other Items to IFF

In addition to a schematic, other items are automatically exported to IFF files. These include:

- All equations
- Substrates used by the schematic
- Linear simulations

Using Custom Mapping Files for IFF Export

Genesys IFF file export is controlled by a sophisticated internal mapping file. You can create a user mapping file if you want to customize the IFF export. For more information on this mapping file and customization services, please contact Agilent or your local distributor.

Part Placement List Export

S-Parameters Export

Noise parameters are not exported with the S-Parameters.

S-Parameters can be exported in one of two ways.

Exporting from the Workspace Tree

1. Right click the linear analysis dataset in the workspace tree and select **Export** from the menu
2. Set the name and directory of the S-Parameter file (*.snp)
   Or

Exporting from the Dataset Window

1. Open the linear analysis dataset (it must be the active window)
2. Select **File**, **Export**, and then **S-Parameters** from the Genesys menu
3. Set the name and directory of the S-Parameter file (*.snp)

SPICE File Export

SPICE files are exported from a schematic. This is because many SPICE device models are significantly different from linear simulator models. Terminations are often handled differently in SPICE and linear simulators. In Genesys, oscillators are analyzed open loop while the loop is closed for SPICE analysis. These differences are resolved in the schematic.

Translating Parts to SPICE

There are three categories of part translation to SPICE:

- Direct
- Compound
- Incompatible

For translations of specific parts, see the *Part Catalog* (part).

Direct - Direct parts are translated on a one-to-one basis. Examples include capacitors, inductors, resistors, signal ground (DC voltage source), and electrical transmission lines.

Compound - Compound parts are translated as SPICE subcircuits. They include mutual inductors , op-amps , VCC , and Crystal. This provides comparable simulations in Genesys and SPICE. For example, an Genesys VCC is modeled by two resistors and a voltage controlled current source. To use just the SPICE VCC device without the resistors, you can override the default translation by double-clicking the VCC device to open the properties window. Select G from the SPICE Device list and click OK to save the changes.

Please note the following:

- The SPICE opamp E model (Genesys translates OPA as an E model) is ideal in that the unity crossover frequency is infinite. You can substitute a SPICE library model or subcircuit for the opamp. Most opamp manufacturers have SPICE models for their products.

- The Genesys TRF device (ideal transformer) is not supported in SPICE. You should specify mutually coupled inductors (MUI). You need to specify appropriate winding inductance and coupling.
- The Genesys FET and BIP devices do not include any biasing information and, therefore, are not translated. You can specify how to translate these parts by defining the translation device in the properties window.

Incompatible - Incompatible parts are those parts that have no simple SPICE equivalent, including physical models, S-parameter or Y-parameter devices, and internal transistor models (FET and BIP). Incompatible parts are identified in the translated SPICE file with an exclamation point (!) at the front of the part line. You can assign a SPICE model (.MODEL) or subcircuit to that part in the properties window.

You must also place a SPICE model definition (.MODEL block) in the exported SPICE file. This is done either through SPICE command text or manually after exporting. If the SPICE simulator supports libraries (both PSPICE and IsSPICE support libraries), the library reference is included in SPICE command text entries.

Touchstone File Export

Touchstone circuit files are similar to Genseys netlists and are generally a one-to-one translation.

XML file Export

Each Genesys object in the workspace tree has an XML format associated with it. Workspace tree objects that can be exported to an XML format. To export an XML file:

1. Click the object in the workspace tree to be exported
2. Select the Export menu option from one of the given methods
3. Specify the name of the directory and filename of the exported object

Layout Designs

There are several tools available to you to help with the export/conversion of Layout Designs.

AutoCAD DXF/DWG Translator

The DXF/DWG translator enables you to convert Genesys designs into AutoCAD's DXF/DWG file format, as well as convert DXF/DWG files into Genesys designs. DXF is a very simple file format that can be read by most CAD programs that support DXF. The DXF/DWG translator is bidirectional and provides support for hierarchical and all layer separation.

DWG is a binary format, therefore the files take less time to load and save. Genesys supports importing and exporting to/from AutoCAD. AutoCAD versions 12 to 2007 are supported for import. AutoCad versions 2000, 2004 and 2007 are supported for export. Carefully consider how you want to use DXF/DWG output - including layer numbering, use of holes, and polygon shapes - before beginning your layout Layout. Setting up the proper layout rules in Genesys can save a lot of time in generating acceptable DXF/DWG output. For specific considerations or limitations, consult your AutoCAD documentation.

Note: Password protected DWG files are not supported.

Genesys includes an editor that can display the DXF/DWG Layout. Using the editor, you can compare the DXF/DWG files to the original Genesys layout Layout to verify that the Layout was exported correctly. Similarly, DXF/DWG files can be previewed prior to import into Genesys:

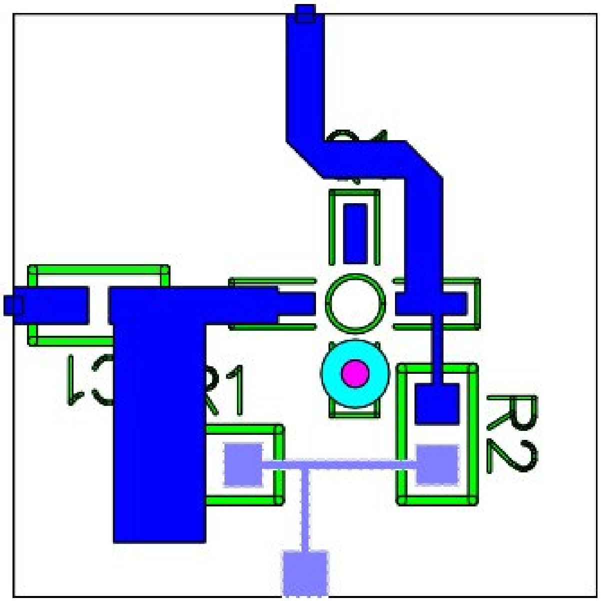
- to edit a large Layout down to a smaller Layout prior to importing it into Genesys. This edited Layout can be saved and then imported in Genesys.
- or after import, to verify that the DXF/DWG Layout data and the Gensys-imported Layout appear similar.

For details, see Using the Pre-Production Editor.

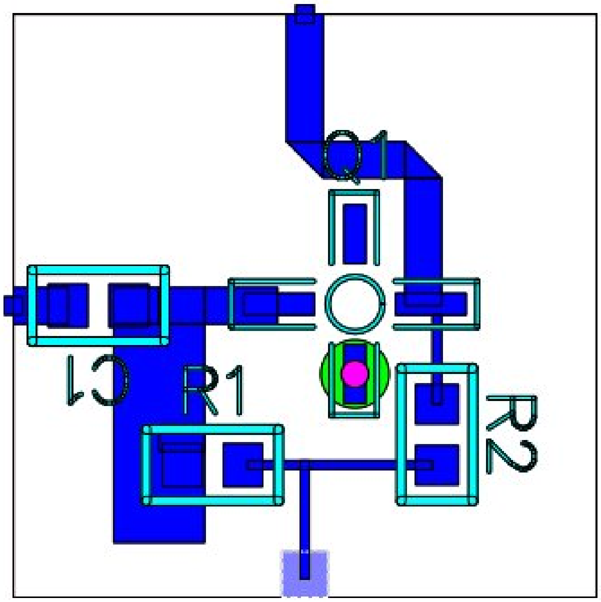
Export Layout Options

We can show up Layout options settings, defining exporting Layout objects (components, EM-ports, drill holes).

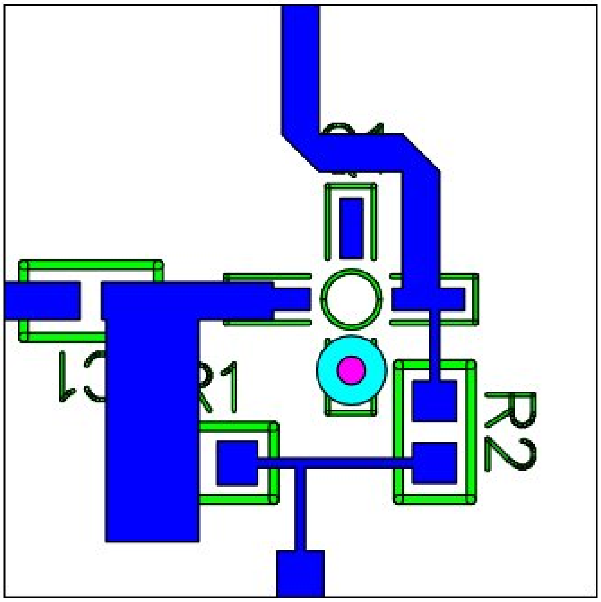
1. Converting to resolved polygons (Export ports = Yes):
2. All layout objects are converted to polygons, which are resolved to polygon unions for all layers, not having text objects.



3. Exporting without converting to "resolved polygons" (Export ports = Yes):
   All layout objects are mapped to output "as is", without converting to polygons.



4. Converting to resolved polygons (Exports ports = No):
   All layout objects are converted to polygons, which are resolved to polygon unions for all layers, not having text objects. No EM ports are exported.



Options

File format	
DXF (Text)	Select this option to export the Genesys Layout to DXF format as an ASCII text file. This selection is the default setting.
DWG	Select this option to export your Genesys Layout to a binary DWG file format. Exporting to DWG file format is faster than the DXF (Text) format. Also file size of the DWG file will be smaller as compared to corresponding DXF (Text) file.
Autocad version	Use this drop-down list to select the AutoCad version. Genesys layout can be exported in three AutoCad versions: 2000, 2004, and 2007.
Paths as polygons	Select **Paths As Polygons** to export the Layout paths or traces as polygons. Paths or traces have mitered or curved corners that need to be preserved in the translation. The Genesys layout has paths with endpoint types other than flush that need to be preserved in the program database. This option is selected as the default.
Hole format	The Hole format section enables you to define how the translator deals with holes in a Layout.
Holes as polygons	Select **Holes As polygons** to convert holes into polygons. One polygon will be created for each hole in the same layer.

Note
Some systems may not be able to tolerate complex polygons with cutlines. For these systems, select **Holes as polygons**. This option is deselected as the default.

Holes as cutlines	Select **Holes as cutlines** to convert holes into cutlines. This option is selected as the default.

Gerber Translator

The *Gerber Artwork Translator* can translate artwork directly from circuit layouts created with Genesys Layout editor into Gerber format. It exports Genesys layouts into ASCII files that control Gerber photoplotting equipment.

Genesys includes an editor that can display the Gerber/Drill design. Using the editor, you can compare the Gerber/Drill files to the original Genesys layout design to verify that the design was exported correctly.

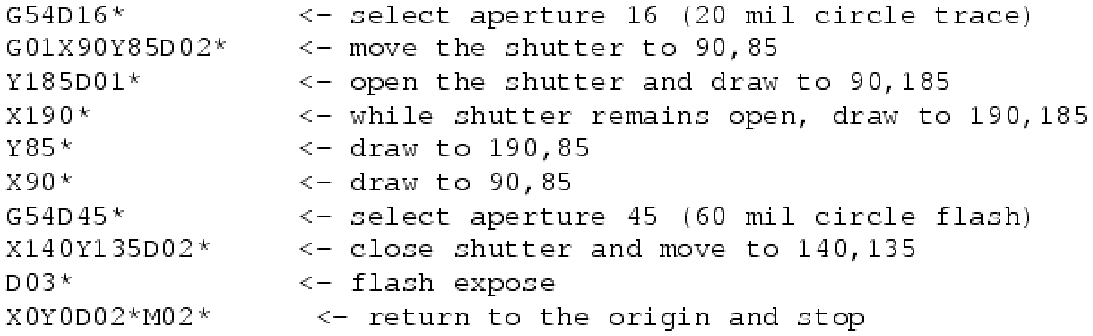
For details, see Using the Pre-Production Editor.

Gerber Command Format

The Gerber format is a numerical control language developed to generate photo artwork. The output of this translator is an ASCII file that contains the following Gerber commands:

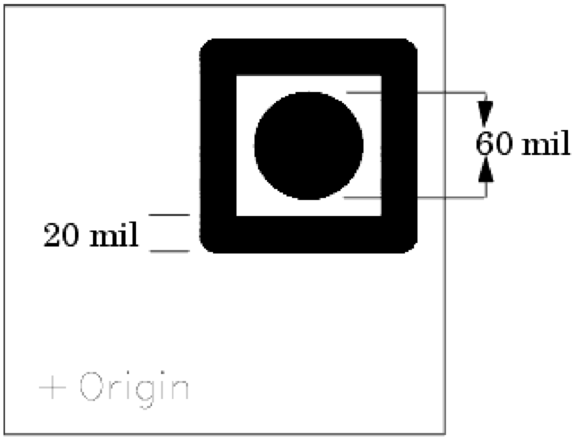
G01 = linear interpolation
G54 = aperture select
D01 = shutter open
D02 = shutter close
D03 = flash
M02 = end of program
X and Y = coordinates
* = end of block

Coordinates are absolute, with implied decimal point and optional leading zero suppression. However, the literal string values may be modified in the message file. For example, *G01* may be changed to *AB*. A sample listing of the Gerber file commands, with interpretation to the right, might look like this:



```
G54D16*           <- select aperture 16 (20 mil circle trace)
G01X90Y85D02*     <- move the shutter to 90,85
Y185D01*          <- open the shutter and draw to 90,185
X190*             <- while shutter remains open, draw to 190,185
Y85*              <- draw to 190,85
X90*              <- draw to 90,85
G54D45*           <- select aperture 45 (60 mil circle flash)
X140Y135D02*      <- close shutter and move to 140,135
D03*              <- flash expose
X0Y0D02*M02*       <- return to the origin and stop
```

This example would produce the following figure.



The output file (Gerber command file) is the Gerber Drawing File (*.gbr*).

Exporting Gerber Files

Genesys Gerber export tool can generate Gerber files in RS274X and MDA format for raster photo plotters.

The Gerber export tool creates one output file for each different layer used in the layout design. For example, if the design contains two layers, TOP METAL and TOP SILK, two Gerber files will be generated:

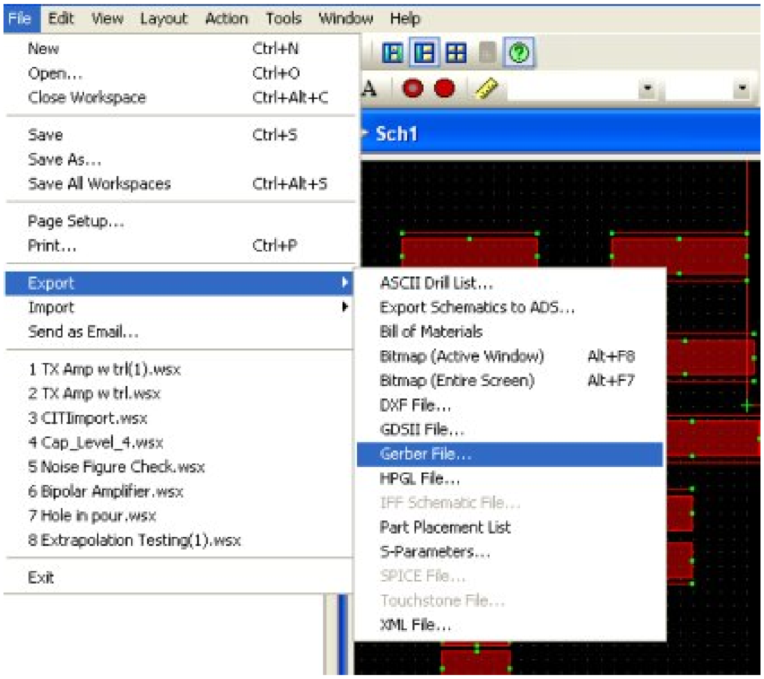
- TOP_METAL.gbr
- TOP_SILK.gbr

All exported viaholes with identical metal layers are exported in 2 files: Gerber layer .gbr, and drill file .drl. For example, if the design has viaholes between TOP METAL layer and BOTTOM COVER, and between TOP COVER and TOP METAL layer, export of the viaholes creates 4 files:

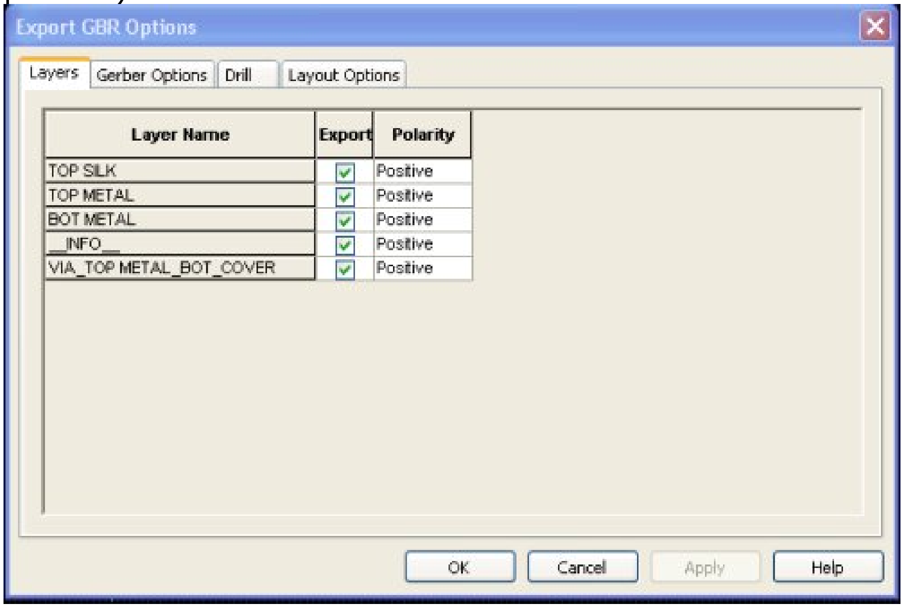
- TOP_METAL_BOT_COVER.gbr
- TOP_METAL_BOT_COVER.drl
- TOP_COVER_TOP_METAL.gbr
- TOP_COVER_TOP_METAL.drl

To export Gerber files:

1. In the layout window containing your design, choose **File > Export > Gerber File...**

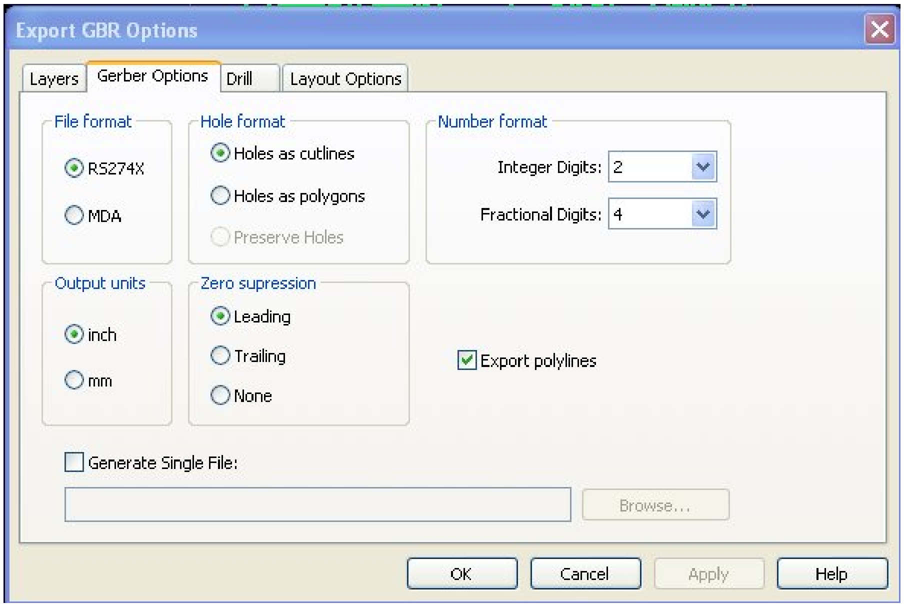


2. The Export dialog box appears. Select export layers, and their polarity (default is positive)



Layer Name	Export	Polarity
TOP SILK	☑	Positive
TOP METAL	☑	Positive
BOT METAL	☑	Positive
__INFO__	☑	Positive
VIA_TOP METAL_BOT_COVER	☑	Positive

3. Set Gerber export options at the tab "Gerber Options" that apply.
For available options, see Gerber File Options.



File format
● RS274X
○ MDA

Hole format
● Holes as cutlines
○ Holes as polygons
○ Preserve Holes

Number format
Integer Digits: 2
Fractional Digits: 4

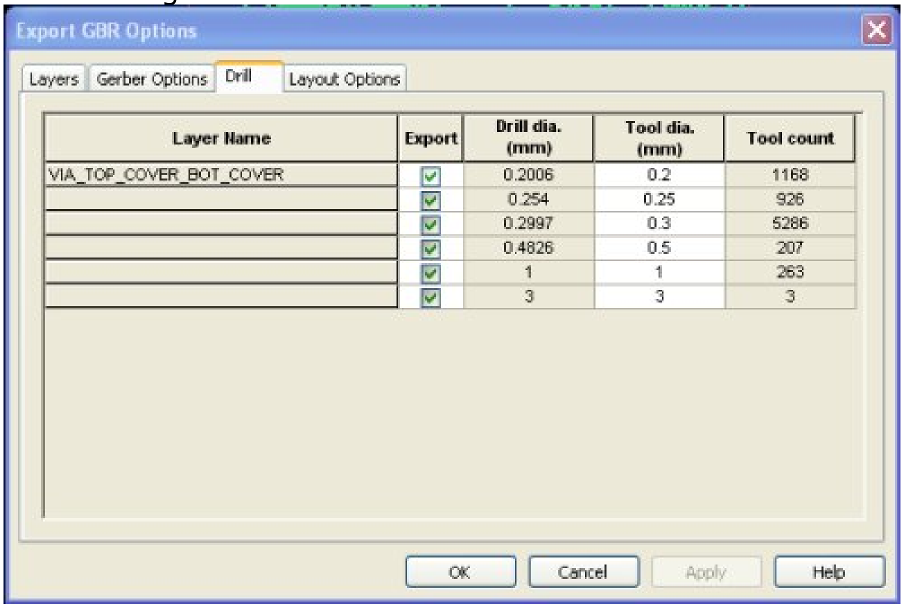
Output units
● inch
○ mm

Zero supression
● Leading
○ Trailing
○ None

☑ Export polylines

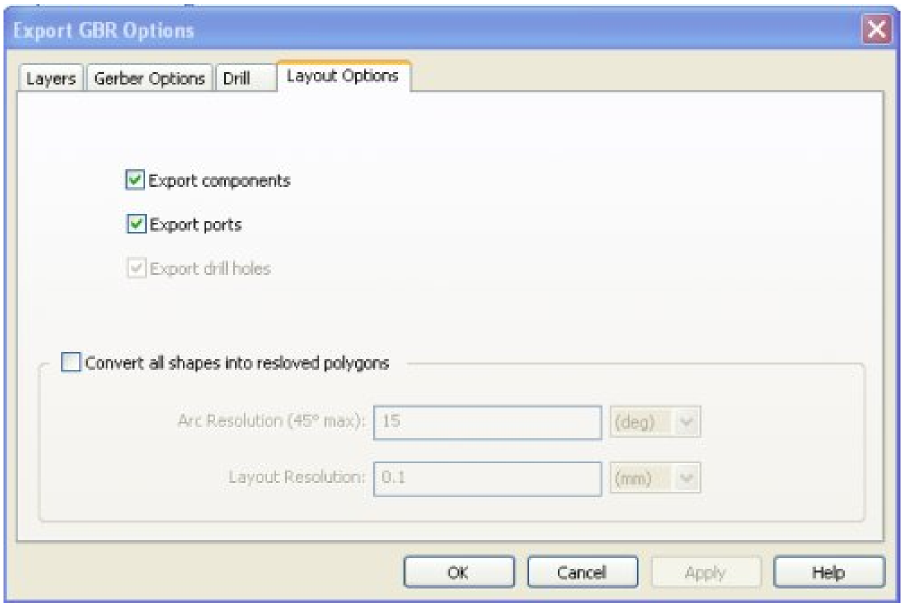
□ Generate Single File:

4. Drill file export is accomplished simultaneously as part of the Gerber export. The procedure for generating the drill file is similar to generating the Gerber file.To create Gerber drill file, select exporting drill layers. You may edit actual drill tool diameter for each drill hole.
   Note. All drill holes from the same the viahole layer are selected together.
   Drill files are configured from the *Drill* tab in the *Export Gerber Options* dialog.Select the check box to designate the layer for which you want to generate the drill file. When a layer is selected, all the tools on that layer are automatically selected. By default, no drill file will be generated during Gerber export.
   The drill file options *Number format*, *Output unit*, and *Zero suppression* are same as options set for Gerber export in *Options* tab of *Export Gerber Options* dialog. However, note that the zero suppression in Drill is actually a zero inclusion.
   Tool Dia. (Diameter) is an editable field. By default the Tool Diameter value equals the Drawing Diameter value.



Layer Name	Export	Drill dia. (mm)	Tool dia. (mm)	Tool count
VIA_TOP_COVER_BOT_COVER	☑	0.2006	0.2	1168
	☑	0.254	0.25	926
	☑	0.2997	0.3	5286
	☑	0.4826	0.5	207
	☑	1	1	263
	☑	3	3	3

5. The exporting layout objects and output objects format may be specified at the "Layout options" tab. It may include component footprints, EM-ports, and drill holes. All Genesys layout objects may be converted to "resolved" polygons by selecting "Convert all shapes into resolved polygons". In this mode you may specify layout geometry resolution parameters for arcs and linear layout object items.



Export GBR Options

☑ Export components

☑ Export ports

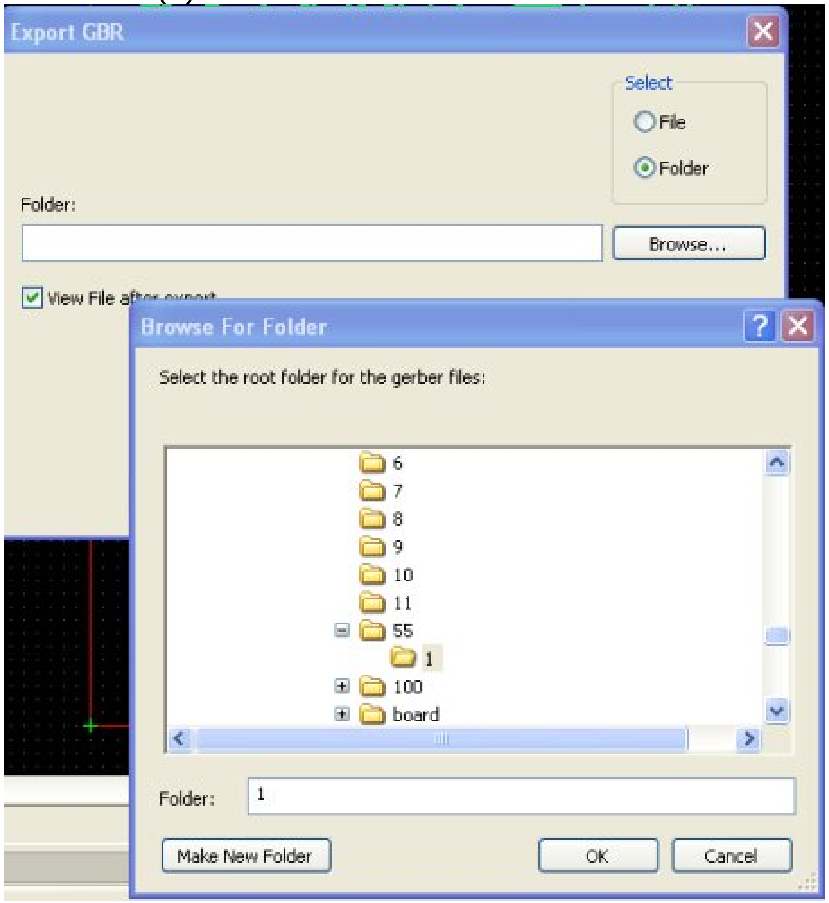
☑ Export drill holes

□ Convert all shapes into resloved polygons

Arc Resolution (45° max): 15 (deg)

Layout Resolution: 0.1 (mm)

6. Click **OK** to save your settings or **Cancel** to retain the default settings.
   It closes "Export GBR Options" dialog and opens "Export GBR" dialog, setting output Gerber file(s) location.



Export GBR

Select
○ File
● Folder

Folder:

☑ View File after export

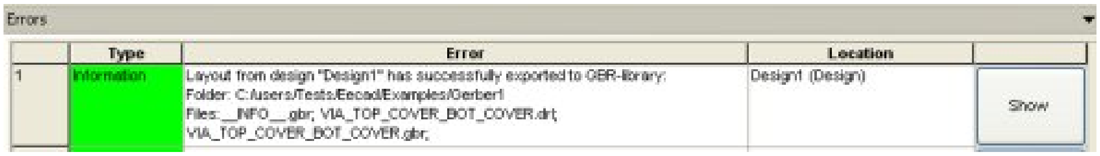
Browse For Folder

Select the root folder for the gerber files:

Folder: 1

As a default all output Gerber files will be exported in a single folder - one file per layer(.gbr) and per drill layer(.drl).
If it's exported to a single file - all Layout layers and drill holes will be saved in one output Gerber file.

7. Click **OK** to export the Genesys layout design to a Gerber file, or click **Options** to update export options.
   A Status window appears detailing the export information.

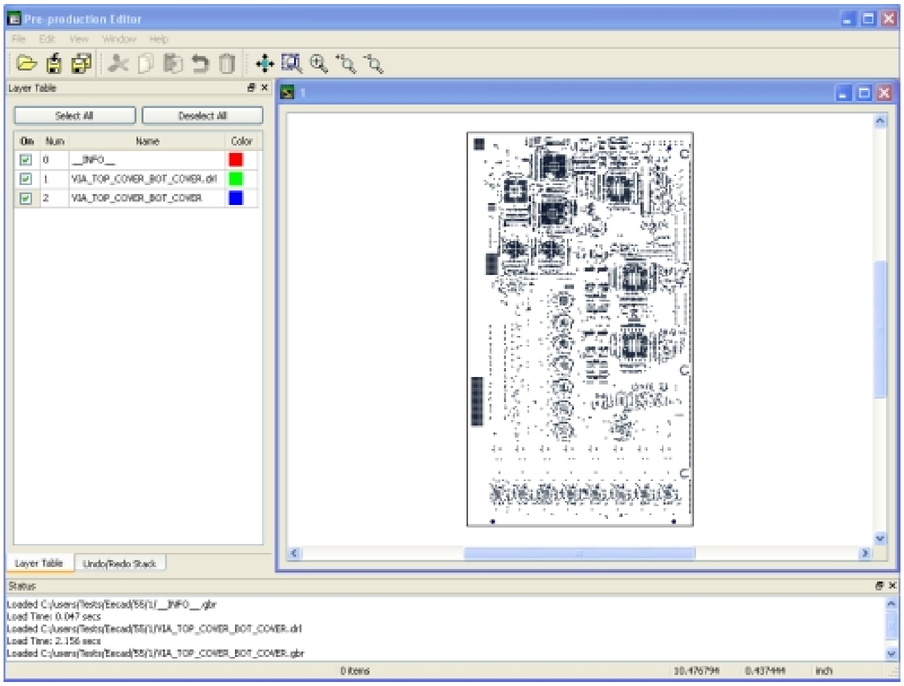


Errors

	Type	Error	Location	

1	Information	Layout from design "Design1" has successfully exported to GBR-library:
		Folder: C:/users/Tests/Eecad/Examples/Gerber1
		Files: __INFO__.gbr; VIA_TOP_COVER_BOT_COVER.drl;
		VIA_TOP_COVER_BOT_COVER.gbr;
			Design1 (Design)	Show

8. If checkbox "View File after Export" is set, after the export it opens "Preproduction editor":



On	Num	Name	Color
☑	0	__INFO__	🟥
☑	1	VIA_TOP_COVER_BOT_COVER.drl	🟩
☑	2	VIA_TOP_COVER_BOT_COVER	🟦

```
Loaded C:/users/Tests/Eecad/55/1/__INFO__.gbr
Load Time: 0.047 secs
Loaded C:/users/Tests/Eecad/55/1/VIA_TOP_COVER_BOT_COVER.drl
Load Time: 2.156 secs
Loaded C:/users/Tests/Eecad/55/1/VIA_TOP_COVER_BOT_COVER.gbr
```

0 items 10.476794 0.437444 inch

Gerber File Options

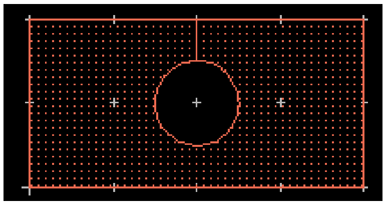
File format

- Select **RS274X** for an enhanced version of Gerber RS274D format that supports embedded apertures and G36/G37 polygons.
- Select **MDA** if your photoplot shop uses a FIRE 9000 plotter. It embeds apertures and supports POEX/POIN polygons for fill on the fly.

Hole format

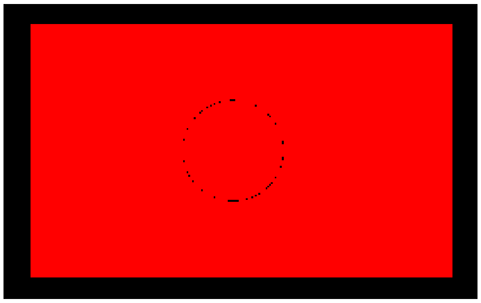
In Genesys, when a hole is created in a geometrical figure (like polygon), a cutline is introduced. This is a false edge connecting the outer boundary of the polygon with the inner boundary. This polygon is actually a single re-entrant polygon. When you select **Holes as cutlines** this re-entrant polygon is translated to Gerber as-is. The default is **Holes as cutlines**.

Holes as Cutlines



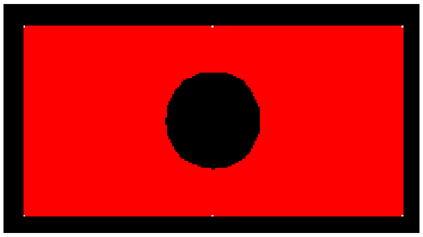
Select **Holes as polygons** or **Preserve holes** to remove the false edge from the polygon.

Holes as polygons



When **Holes as polygons** is selected, holes are exported as filled parts. Therefore the polygon will appear to have no holes.

Preserve holes



When **Preserve holes** is selected, the resultant polygon in Gerber contains a dark area and empty area. This option is not available for exporting Gerber files in RS274X format.

Unit

Available units are inch or mm. Inch is the default.

Number format

The number of integers placed before and after the decimal point. If chosen incorrectly, Gerber data resolution can be poor. The default is 2.4. If the unit is set to mm then you can set the number format to 3.3.

A warning message is displayed in the Status window to notify the user that to preserve the precision in the Gerber data, number format has to be selected carefully. If the Gerber data is not generated correctly for the selected number format then a suitable error message is displayed in the Status window.

Zero suppression

Available settings are Leading and None. Select **Leading** (the default) to remove all leading zeros in the coordinate data, making the Gerber file smaller.

Line width for polylines

Polylines in Genesys have zero width, but you can use this option to provide a width to be used for drawing this polyline in the Gerber file.

If the polylines and arcs (both zero width entities) are not required to be exported, then set the width to zero.

Generate single file

When this check box is selected, a single file is generated for all the selected layers in the *Layer* tab. Specify the file name with extension .gbr. This file will be created in the destination directory selected in the *Export* dialog box.

Gerber Layer Options

All the layers used in the design are displayed in a table in the *Layers* tab of the *Export Gerber Options* dialog. By default, all the layers will be displayed with positive polarity and all layers will be exported. You can avoid exporting a specific layer by de-selecting the check box in the Export column for that layer.

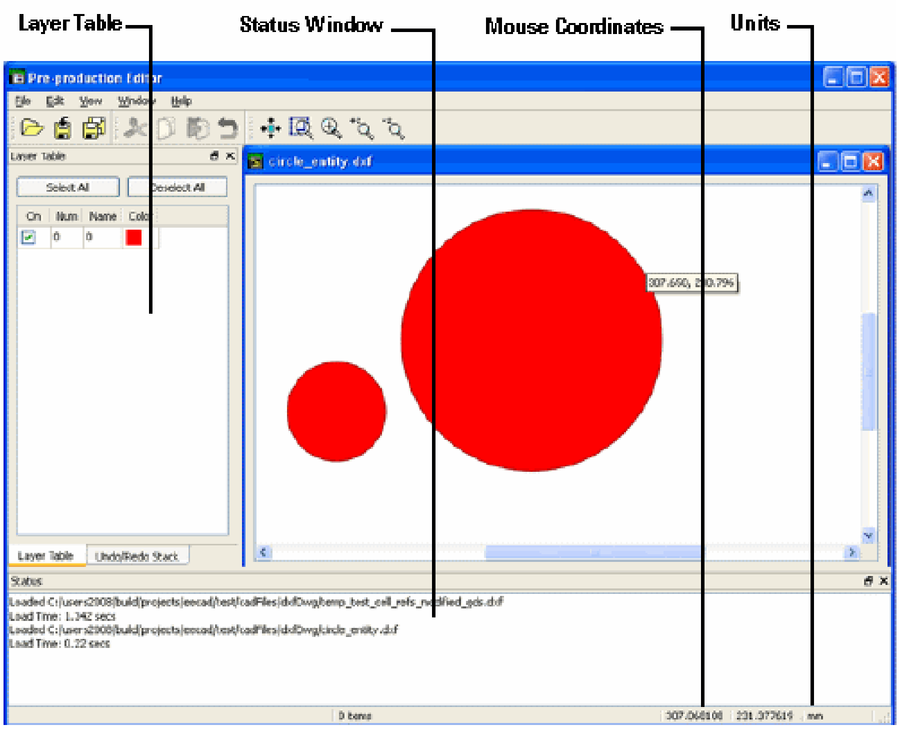
Limitations and Considerations

- No intermediate mask file is generated.
- The new Gerber export tool is targeted for the raster photoplotters to generate RS274X and MDA file format.
- No aperture filling is supported for polygons.
- No separate aperture file is required for RS274X and MDA files.
- You must set the appropriate number format in the *Export Gerber Options* dialog so that the precision in Gerber data is more than the precision in layout design.
- The Gerber translator can generate a single Gerber file for all layers. However, the Gerber Viewer cannot process a Gerber file in which multiple layers are defined in a single file.
  It imports only the first layer defined in the file.
- Exporting MDA files using "Generate single file" option will export all the figures in the same layer. It is advisable not to use "Generate single file" option for MDA format.
- If a layer does not have any via (circles) then it will not be displayed in the *Drill* table and no drill file will be generated for that layer.
- One drill file will be generated for each layer.
- Drill files will be created in the Destination directory selected in the *Export* dialog.
- Drill file name is the same as layer name with extension .drl. For example, drill file for the layer *cond* will be *cond.drl*.
- If Tool Diameter is set to 0.0, the Tool Diameter will be equal to the Drawing Diameter in the output drill file.

Using the Pre-Production Editor

Genesys includes an editor that can display DXF/DWG and Gerber/Drill designs. Using the editor, you can compare the DXF/DWG and Gerber/Drill files to the original Genesys layout design to verify that the design was exported correctly. Similarly, DXF/DWG and Gerber/Drill files can be previewed prior to import into Genesys:

- to edit a large design down to a smaller design prior to importing it into Genesys. This edited design can be saved and then imported in Genesys.
- or after import, to verify that the DXF/DWG and Gerber/Drill design data and the Genesys-imported design appear similar.



```
Loaded C:/users2008/build/projects/eecad/test/cadFiles/dxfDwg/temp_test_cell_refs_modified_gds.dxf
Load Time: 1.342 secs
Loaded C:/users2008/build/projects/eecad/test/cadFiles/dxfDwg/circle_entity.dxf
Load Time: 0.22 secs
```

File Menu

The File menu contains common Open, Save, Save As, and Close functions and lists the four most recent files viewed in the editor.

Note
When files are saved as Gerber/Drill files using the Pre-Production editor, the layer name is used as the file name.

Edit Menu

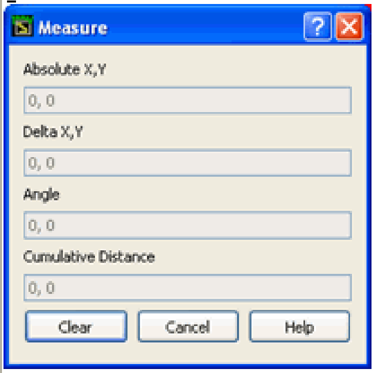
The Edit menu contains common and specialized editing commands.

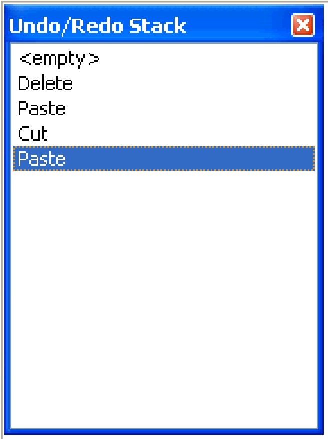
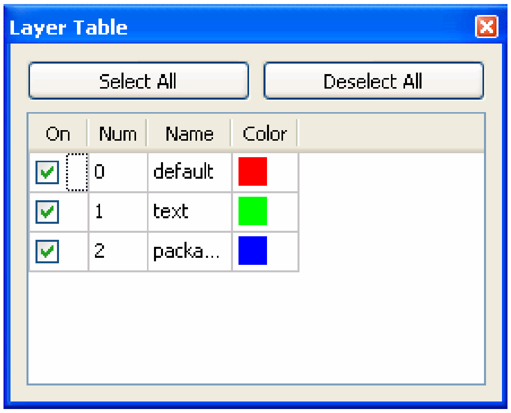
Enable Editing	Enables editing commands to be performed on the current design. De-select this item to view the design in read-only mode. In this mode, editing commands are unavailable.
Undo	Undoes the last editing command. A stack of edit commands is created enabling you to choose Undo repeatedly to return to an earlier state of your design. A stack is maintained for each window, thus the Undo command works independently from window to window.
Redo	Returns the file to the pre-undo state.
Cut	Enables you to delete one or more items from one window, and paste in another window.
Copy	Enables you to copy items in a given design window and then paste those items within the same design window or another design window.
Paste	Enables you to paste items that you previously cut or copied. You are prompted to enter the X and Y coordinates of the position where you want to paste the copied items. Select **Apply** to paste or **Cancel** to dismiss pasting.
Delete	To delete selected items, click the **Delete** button on the toolbar, or press the **Delete** key on the keyboard, or choose **Delete** from the *Edit* menu. Deleted items can be restored using the Undo command.
Select All	Enables you to select all figure(s).
Deselect All	Deselects all the selected figure.
Modify	This choice lets you modify the following items: • Select figures and click **Union** to perform union of the figures. • Select figures and click **Intersection** to perform union of the figures. • Select figures and click **Union Minus Intersection** to perform union of the figures • Select **Crop** to crop an area from a figure(s). • Select **Chop** to chop an area from a figure.

Note
Union, Intersection, Union Minus Intersection, Crop, and Chop do not work for open ended figures with zero width (for example arc and polylines).

View Menu

The view menu contains commands that allow you to alter the view of the current design.

View All	Your design is scaled and repositioned to fit within the viewing area.
The Zoom Commands	The Zoom commands enable you to enlarge or shrink the area being viewed. You can zoom in and zoom out using mouse wheel. The enlarged or condensed figure is moved towards the center of the window. • **Zoom To Area** > click a point and then with the mouse button pressed move the mouse to the bottom right of the area and release the mouse button to define a new view. • **Zoom to Point** to zoom in on a specified point in the window. Click to specify a point and the current view is magnified by a factor of two, moving the point you specify to the center of the window. • **Zoom In By 2** to zoom in by a factor of 2. • **Zoom Out By 2** to zoom out by a factor of 2.
Measure	 Absolute X,Y: 0, 0 Delta X,Y: 0, 0 Angle: 0, 0 Cumulative Distance: 0, 0 The Measure dialog allows you to measure lengths.

	• *Absolute X, Y* displays the absolute coordinates of the point of mouse click. • *Delta X, Y* gives the relative X and Y coordinates from previous mouse click. • *Angle* gives the angle with respect to a line horizontal to the X axis. • *Cumulative distance* gives the total length from starting point to current point. • Click **Clear** to reset the values for Absolute X,Y, Delta X, Y, Angle and Cumulative distance. • Click **Cancel** to dismiss the measure dialog box.
Undo/Redo Stack	 Undo/Redo Stack <empty> Delete Paste Cut Paste Shows last performed actions. You have an option to undo or redo the last operations performed.
Layer Table	 Layer Table Select All · Deselect All On · Num · Name · Color ☑ · 0 · default · 🟥 ☑ · 1 · text · 🟩 ☑ · 2 · packa... · 🟦 The layer Table displays the layer information. It enables you to switch on/off layers from display. It also enables you to view layer number, layer name and layer color. Click **Select All** to view all the layers and **Deselect All** to switch off all the layers. All the layers are displayed by default.
Status	The status window displays messages about the status of the current design, as well as warning and error messages.

Window Menu

This menu contains the following common Window commands:

Close	to close the active window.
Close All	to close all open designs.
Tile	to arrange all child windows in a tile pattern.
Cascade	to arrange all the child windows in a cascade pattern.
Arrange Icons	to arrange all iconified windows at the bottom of the workspace.
Next	to navigate to next open window.
Previous	to navigate to previous open window.

Viewing Files in the Pre-Production Editor During Export

To view a DXF/DWG or Gerber/Drill file during export from Genesys:

1. Select **DXF/DWG** or **Gerber/Drill** from File type drop-down list in *Export* dialog.
2. Select the **View file after export** checkbox and click **OK**.
   This will display the exported DXF/DWG or Gerber/Drill file. A status window shows if the export is complete or failed and the design is displayed in the Pre-Production Editor.

Viewing Files in the Pre-Production Editor During Import

To view a DXF/DWG or Gerber/Drill file during import to Genesys:

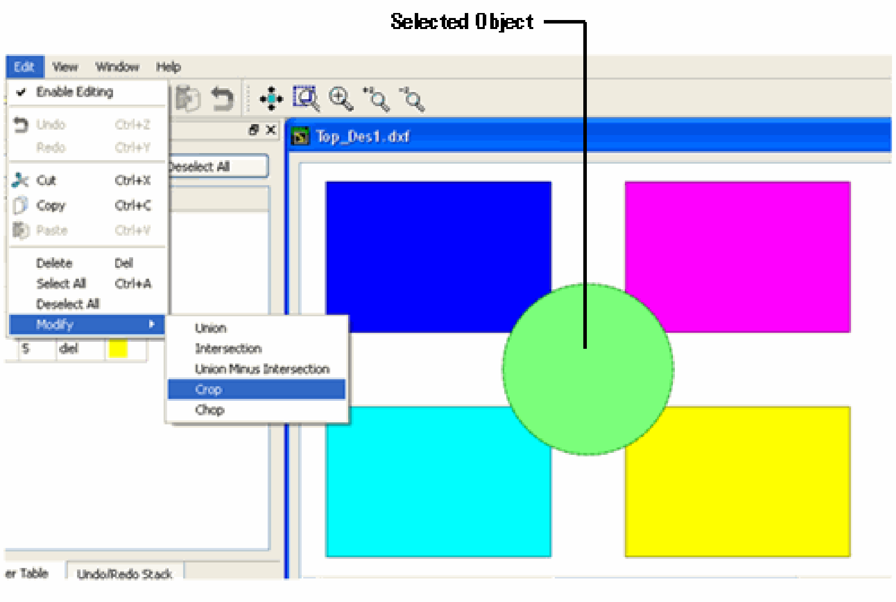
1. Select **DXF/DWG** or **Gerber/Drill** from File type drop-down list in *Import* dialog.
2. Click **Preview** to view the file in the Pre-Production Editor.

Editing Files in the Pre-Production Editor

Using the Pre-Production Editor you can cut, copy, paste and modify designs. From the *Modify* menu, you can perform editing functions like union, intersection, union minus intersection, crop, and chop.

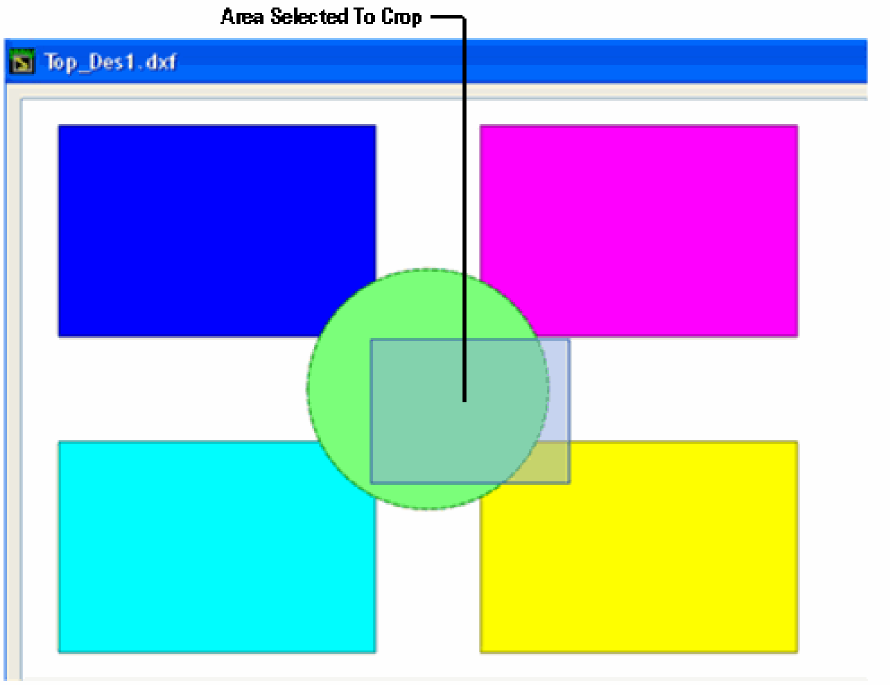
To edit using the Pre-Production Editor:

1. Open a design in the Pre-Production Editor as described above.
2. Click **Edit** > **Edit Enable**.
   - To cut a figure, select the figure and then click **Edit** > **Cut**.
   - To copy a figure, select the figure and then click **Edit** > **Copy**.
3. To crop or chop a figure:
   - Select the figure.

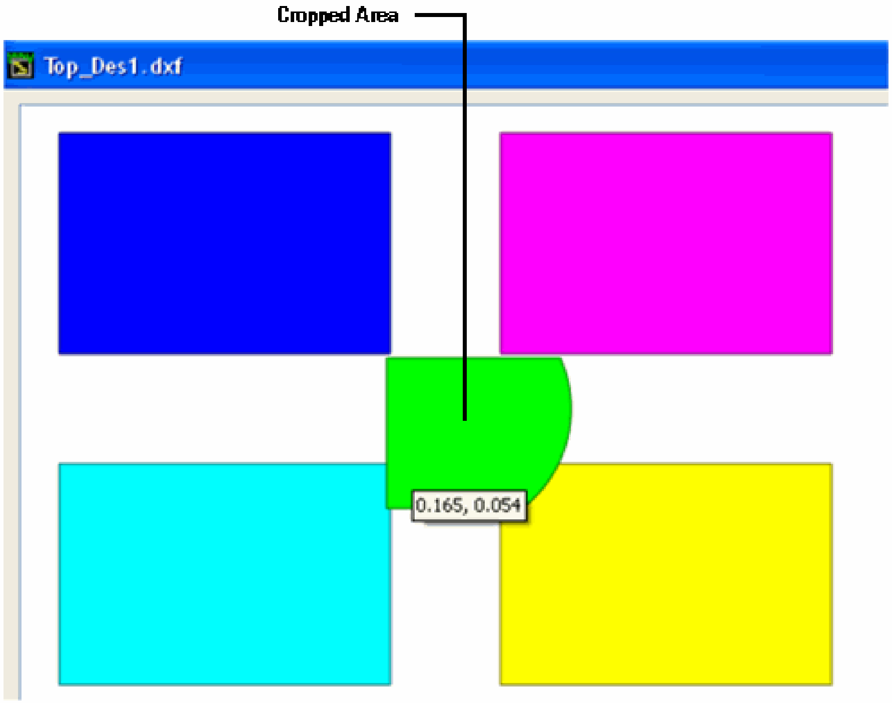


Selected Object

   - Click **Crop** or **Chop**.
   - Mark the area to crop/chop using mouse.



- Release the mouse to crop/chop the selected area.



4. Click **Delete** to delete selected item.
5. Click **Undo** to revoke last Edit command.

Using Files From Earlier Genesys Versions

Genesys lets you load files created using all versions of Genesys prior to the current version. Once you save the files using the current version of Genesys, the files will not load in versions prior to GENESYS2005 (forward compatibility, and not backward compatibility).

GENESYS2005 and beyond are backward compatible as long as you disable Compact Data Format (Tools / Options).

Old Support:

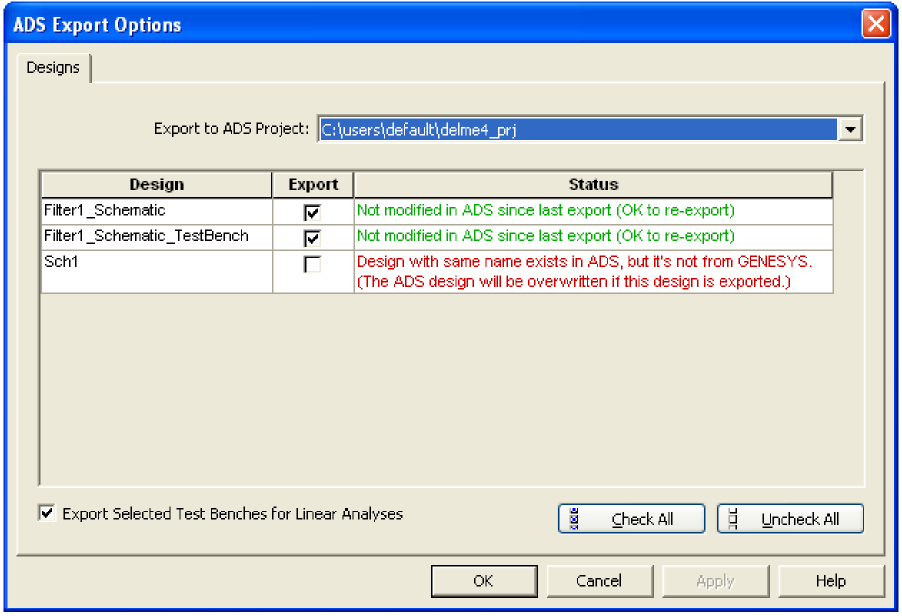
Genesys imports files created using SUPERSTAR version 4 or later and any version of Genesys. You cannot import the following features from versions of SUPERSTAR created prior to version 4:

- **Post Processing** - Genesys now has better post processing capability that is not compatible with the old technique. For information on the new post processing procedures, see the *Simulation* manual and the *Using Equations* chapter of the *User's Guide* . Also, see the Genesys example file Model Extract.wsx.
- **3D Graphs** - Parameter sweeps are now different, so you must recreate 3D graphs after importing them.
- **EMPOWER Simulation Data** - EMPOWER calculated data is now stored inside the workspace file, instead of separate files. You must recalculate the data after importing. Also, you might want to set up any decomposition examples again, because Genesys version 7 and later provides better ways to set up these files. For more information, see the *EMPOWER* manual.
- **Multiple Impedances** - Files with multiple WINDOW blocks reusing the same network to simulate different terminations use only the first set of terminations. Impedances are now stored in the ports and are tunable. If you still need multiple impedances:
  1. Load your schematic.
  2. Create other schematics using NET blocks so your schematic can use the terminations you want.

- ***Microstrip Radial Stubs (MRS)* (part)** - There is a new radial stub model with only one port that was introduced in Genesys version 7. Replace any radial stubs with a microstrip tee and the new stub. For more information, see the *part Catalog* .

Using the ADS Link

Genesys provides a link to Advanced Design System (ADS). You can export schematics from Genesys directly to an open ADS session by selecting File > Export > Export Schematics to ADS. Genesys displays a dialog box showing all instances of ADS with open projects.



ADS Export Options

Designs

Export to ADS Project: C:\users\default\delme4_prj

Design	Export	Status
Filter1_Schematic	☑	Not modified in ADS since last export (OK to re-export)
Filter1_Schematic_TestBench	☑	Not modified in ADS since last export (OK to re-export)
Sch1	□	Design with same name exists in ADS, but it's not from GENESYS. (The ADS design will be overwritten if this design is exported.)

☑ Export Selected Test Benches for Linear Analyses

You first select the instance that you want (if there is more than one running), then you select which design/s to export to ADS. When you click OK, the selected schematics are transferred to ADS. See ADS RF Architect and Synthesis documentation (in the Transfer/IO section of the ADS documentation) for more information on using the link in ADS.

The link transfers schematics only from Genesys to ADS. It does not transfer schematics from ADS to Genesys. Only schematics and linear simulations will be transferred. Layouts, plots and datasets are not transferred. Not all schematics will transfer exactly, see ADS documentation for substrate/model mapping information.

Installation Requirements

- An installed version of Genesys 2006.10 or newer
- An installed Windows version of ADS 2006A or newer

Licensing

- If Genesys was purchased as an ADS add-on, choose ADS style licensing in Genesys.
- On multi-CPU systems which use network licenses, it is important to always:
    1. Stop the ADS simulator before launching Genesys.
    2. Exit Genesys before starting a simulation in ADS.

Accessing ADS Documentation

Documentation is available on the web at:
http://edocs.soco.agilent.com/display/doc/Home ,
or can be accessed by clicking a help button in ADS.

Layouts

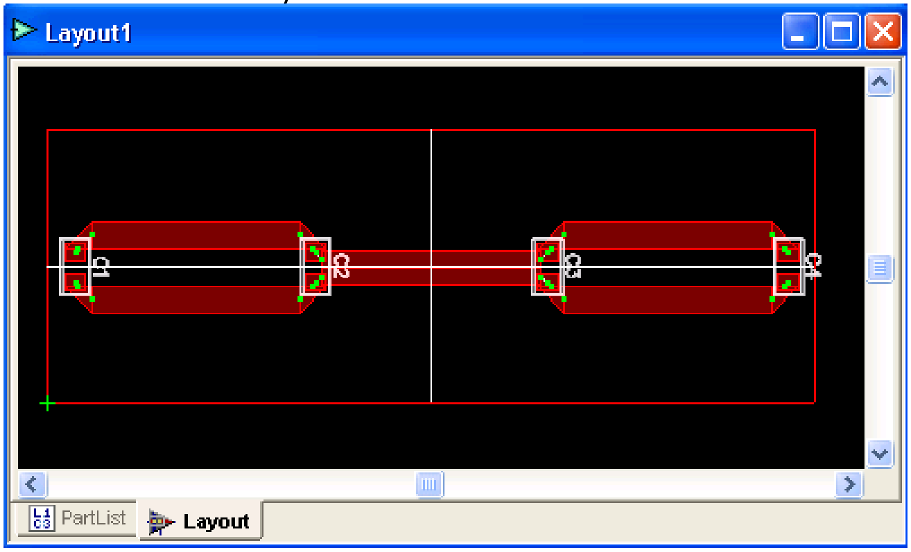
Layout creates a board description for sending to a board plotter or for simulating the EM effects using EMPOWER simulation. You can export the layout for milling or etching in standard industry formats such as GERBER, DXF, and GDSII. You can also create a layout from scratch.

Contents

Creating Layouts

A layout is the physical arrangement of parts on a circuit board. It creates a board description that you can send to a circuit board manufacturer or use to simulate electromagnetic effects through MomentumGXF. You can export a layout for milling or etching in standard industry formats such as Gerber, DXF, and GDSII. You can create a layout based on a schematic, from scratch, or from artwork imported from a file.

You can create a layout from scratch or based on a schematic.



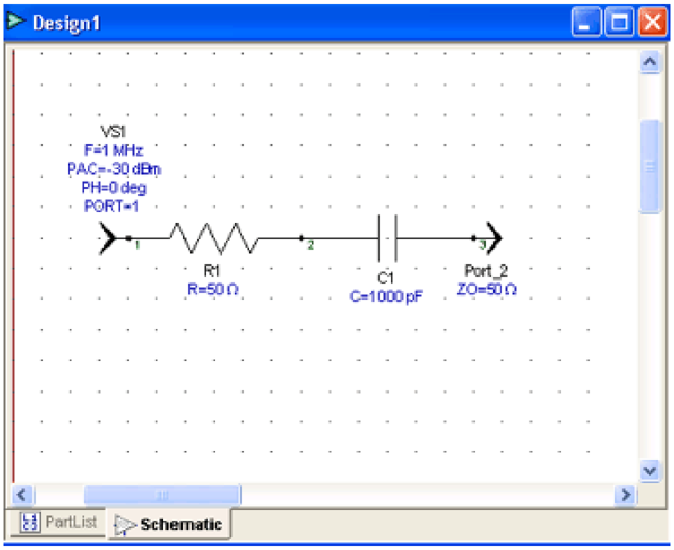
To create a design with a layout:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Layout** from the Designs menu.
2. Define the layout properties and click **OK**.

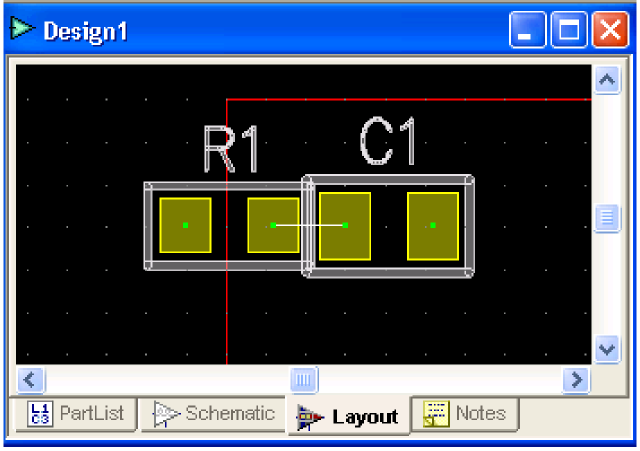
3. Use the layout toolbar to add metal to the layout.
4. To add parts to a layout, place them in the associated schematic.

To create a design with a layout based on a schematic:

1. Create a design with a schematic.
2. Use the Part Selector to add parts to your schematic.
3. Right-click the **Schematic** tab and select **Add Layout**.
4. Select a component and click the Change Footprint button ( ) on the Layout toolbar to change the footprint used for specific components. For example, change the footprint used by a specific capacitor to a larger or smaller footprint.
5. Move and rotate footprints to the positions you want.
6. Place lines and arcs as required to connect footprints and resolve the rubber band lines.
7. Place any other required objects, such as text, connectors, non-schematic footprints, and ground planes.
8. Send the finished layout to a printer, plotter, or file.We can add a layout to our current design..



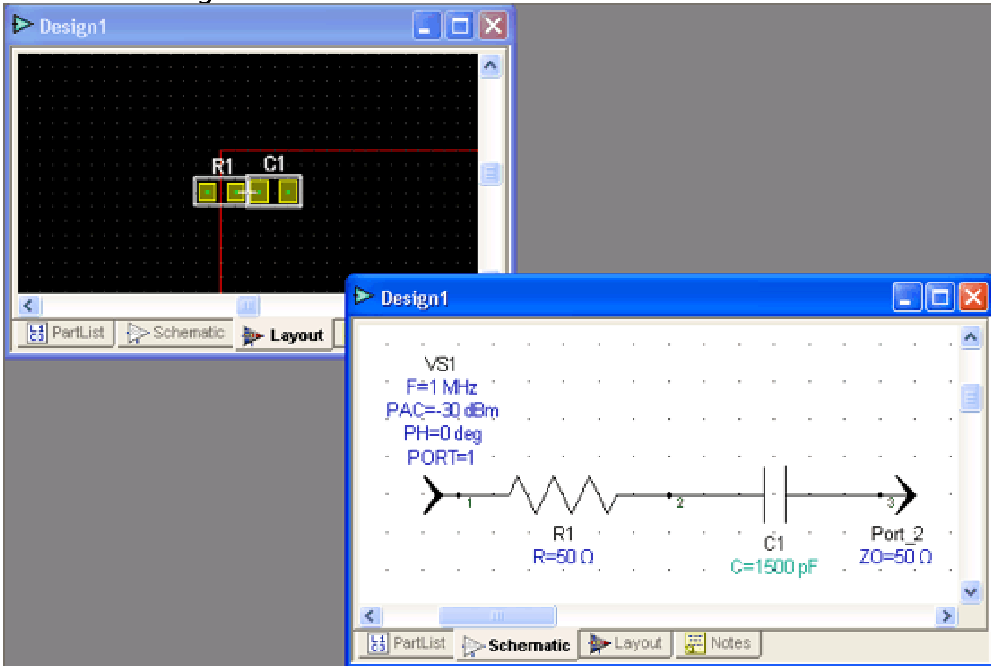
The layout is automatically generated and added to the design.



Note: As you edit the schematic or partlist and add or delete parts these changes are reflected in the layout.

Note: Layouts can be associated with more than one design. Check the layout properties and ensure you are using the correct design(s).

Advanced technique - multiple windows Select Window / New Window from the menu, then resize and rescale (*hint: use Ctrl+Home to maximize the parts*) the schematic to get this..



Note: Some edits made in one window will be reflected in the other window only when it is clicked (to reduce processing overhead).

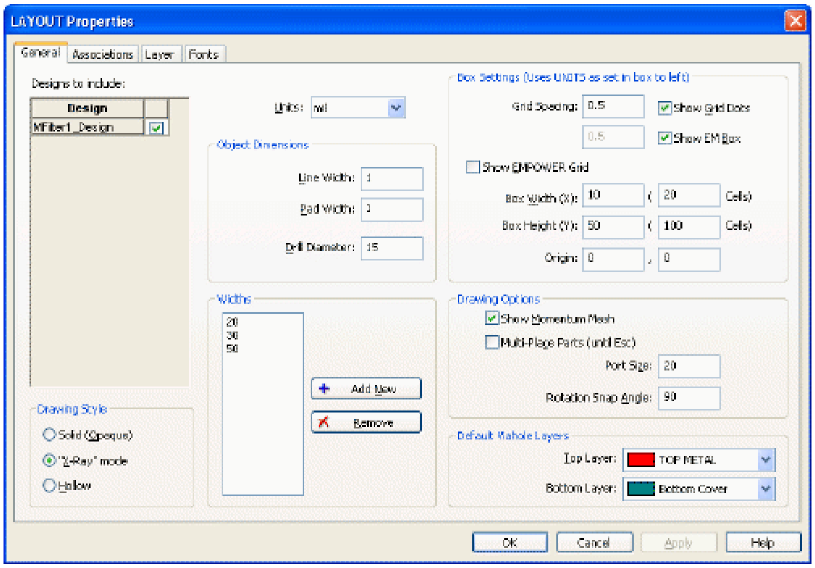
Changing Layout Properties

Changing Layout General Properties

Use the Layout Properties General page to change the general properties of a layout.

To change the general properties of a layout:

- Click **Layout** on the Genesys menu and select **Layout Properties**.
  - Click the **General** tab.
  - Make the changes you want.
  - Click **OK**.



Designs to include:

Design	
MFiber1_Design	☑

Units: mil

Object Dimensions

Line Width: 1

Pad Width: 1

Drill Diameter: 15

Box Settings (Uses UNITS as set in box to left)

Grid Spacing: 0.5 ☑ Show Grid Dots

0.5 ☑ Show EM Box

☐ Show EMPOWER Grid

Box Width (X): 10 (20 Cells)

Box Height (Y): 50 (100 Cells)

Origin: 0 , 0

Widths

20
30
50

Drawing Options

☑ Show Momentum Mesh

☐ Multi-Place Parts (until Esc)

Port Size: 20

Rotation Snap Angle: 90

Drawing Style

○ Solid (Opaque)

● "X-Ray" mode

○ Hollow

Default Viahole Layers

Top Layer: 🟥 TOP METAL

Bottom Layer: 🟦 Bottom Cover

- **Designs to Include** - This grid shows all available designs to place on the layout. If a box is checked, the layout will contain footprints and rubber band lines corresponding to the parts in that design. These footprints and rubber-bands will automatically update as needed.
- **Drawing Style** - This is the fill mode for all metal filled objects (lines, rectangles, polygons, etc).
    - **Solid Drawing Mode** - Classic opaque fill.
    - **"X-Ray" Drawing Mode** - Semi-Transparent fill that allows users to see overlapping layers.
    - **Hollow Drawing Mode** - Emphasizes edges, with a faint interior fill.
- **Units** - The available units for dimensioning objects on the layout. If you enter a number for a custom unit, simply use a constant multiplier for converting the unit to millimeters. Some common numbers are:

mm	1
mils	.0254
meters	1000
inches	25.4

- **Object Dimensions** - Default sizes for most commonly used objects. These numbers define the default dimensions line and pad widths, and the drill diameter for viaholes.
- **Box Settings** - Determines how the page is displayed on the layout screen. You can use the page as a board edge indicator, for use in placing the footprints. This box also corresponds to the EMPOWER box. The following options are available:
    - **Widths** - The available widths for lines and arcs. The widths shown here are available in the Line Width combo box on the main LAYOUT screen.
    - **Remove** - Removes the selected width from the Widths box (see above).
    - **Add New** - Adds a new width to the available list in the Widths box (see above).
    - **Grid Spacing** - The on-screen vertical and horizontal grid spacing, using the selected units. Parts are placed on this grid by default, so this number determines the resolution for part placements.
    - **Grid Spacing X, Grid Spacing Y** - These control the cell size for the EMPOWER run as well as the grid snap feature in LAYOUT. When using the EMPOWER Grid Style, there will be LAYOUT snap points between each grid line which allow lines to be centered between two grid points if necessary. They are often referred to as dx and dy and should be small with respect to a wavelength at the maximum frequency to be analyzed, preferably less than l/20 and always less than l/10. These parameters correspond directly to the DELTA statement in the TPL file.
    - **Show EMPOWER Grid** - Turning on this check box forces LAYOUT to display the rectangular EMPOWER grid. It also allows different grid spacings in the X

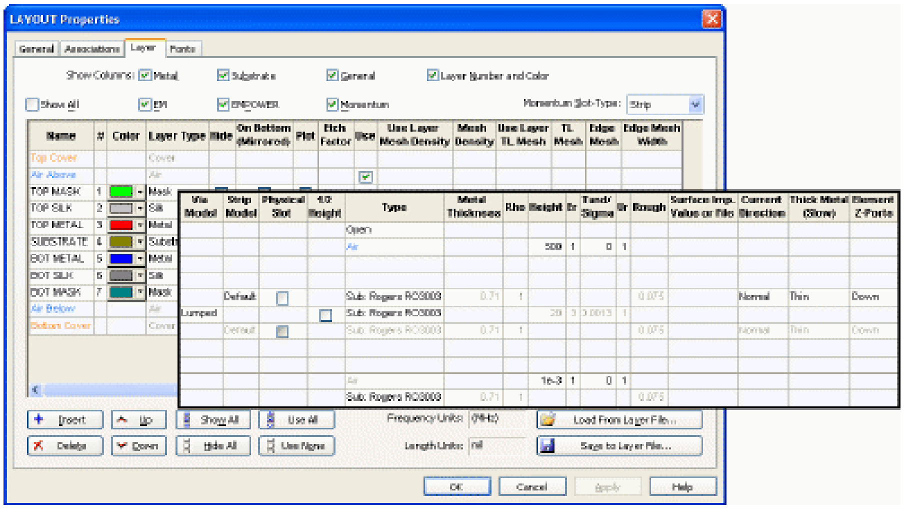
and Y dimensions. It is strongly recommended to turn this check box on whenever you are creating a layout for EMPOWER.

-
    - **Box Width (X,Y)** - The desired width and height of both the page and the surrounding EM analysis box, using the units selected in the Units box.
    - **Origin** - The origin for mouse cursor measurements. The on-screen coordinates display information relative to this origin. The absolute origin is at the lower left of the page. To specify a new origin, enter coordinates relative to the absolute origin, using the selected units. For example, if your page is 100 x 200 mils and you want the origin at the upper left, you would enter 0, 100.
    - **Show Box** (Check box) - Shows or hides the page boundary.
    - **Show Grid Dots** (Check box) - Shows or hides the part placement grid.
- **Drawing Options** - The following options are available:
    - **Port Size** - The size (using the current units) for drawing ports.
    - **Rotation Snap Angle** - The incremental angle (in degrees) used for rotating objects. This can be any number, but should be positive and < 360.
    - **Multi Place Parts** (Check box) - Turns on or off multiple placement. In the main LAYOUT window, you click an object button (such as the Line button), to place an object on the layout. Normally, the object button must be selected every time the object is to be placed. The Multi Place Parts option allows you to place as many parts as you like by selecting the object button only once. Press Escape when done placing parts.
    - **Default Viahole Layers** - The Start Layer and End Layer combo boxes control the default layers for the viaholes. These layers can be overridden individually for each viahole if necessary. Currently, viaholes in EMPOWER can only go from the metal layer through one substrate layer to either the top or bottom cover.

Changing Layout Layer Properties

Use the Layout Properties Layer page to make changes to any of the layers in a layout.
To change the general layer properties of a layout:

1. Click **Layout** on the Genesysmenu and select **Layout Properties**.
2. Click the **General Layer** tab.
3. Use the **Show Columns** check-boxes at the top to control which cells are displayed in the information grid.
4. Make the changes you want.
5. Click **OK**.



Show Columns: ☑ Metal ☑ Substrate ☑ General ☑ Layer Number and Color

☐ Show All ☑ EM ☑ EMPOWER ☑ Momentum

Momentum Slot-Type: Strip

Name	#	Color	Layer Type	Hide	On Bottom (Mirrored)	Plot	Etch Factor	Use	Use Layer Mesh Density	Mesh Density	Use Layer TL Mesh	TL Mesh	Edge Mesh	Edge Mesh Width
Top Cover			Cover											
Air Above			Air					☑						
TOP MASK	1	🟩	Mask											
TOP SILK	2	⬜	Silk											
TOP METAL	3	🟥	Metal											
SUBSTRATE	4	🟫	Subst											
BOT METAL	5	🟦	Metal											
BOT SILK	6	⬛	Silk											
BOT MASK	7	🟦	Mask											
Air Below			Air											
Bottom Cover			Cover											

Via Model	Strip Model	Physical Slot	1/2 Height	Type	Metal Thickness	Rho	Height	Er	Tand/ Sigma	Ur	Rough	Surface Imp. Value or File	Current Direction	Thick Metal (Slow)	Element Z-Ports
				Open											
				Air			500	1	0	1					
	Default	☐		Sub: Rogers RO3003	0.71	1					0.075		Normal	Thin	Down
Lumped			☐	Sub: Rogers RO3003			20	3	0.0013	1					
	Default	☐		Sub: Rogers RO3003	0.71	1					0.075		Normal	Thin	Down
				Air			1e-3	1	0	1					
				Sub: Rogers RO3003	0.71	1					0.075				

Frequency Units: (MHz)

Length Units: mil

- **Name** - The name assigned to each layer. This name is used throughout the program to identify the layer. Although you can type anything for the layer name, you should limit the length to about 12 characters, since combo boxes within the program are not wide enough to display lengthy names.

- **#** - Layer number
- **Color** - Shows the selected color for each layer. Click the button for any layer to select another color.
- **Layer Type** - Identifies the layer type. Available options are:
    - **None** - This layer is considered blank (it is not used).
    - **Metal** - All conductive traces and pads go on a metal layer.
    - **Substrate** - Separates metal layers, and is used to indicate board dimensions. Any cuts or holes in the board (screw holes, etc.) go on a substrate layer.
    - **Silk** - Silk screen is used for labeling on the final board. It is often white or yellow for easy identification.
    - **Mask** (Solder Mask) - This is a negative layer - objects on this layer indicate an absence of solder mask. This layer is automatically generated from pads and viaholes.

      > **Note:** Momentum will project mask layer objects onto the closest metal layer (as copper).

    - **Assembly** - Indicates exact positions for component placement. It is used as a diagram for placing components at the production stage, and does not actually get used during board creation.
    - **Paste** - Indicates where solder paste should be placed.
- **Hide** - When selected, the corresponding layer is not shown in the layout.
- **On Bottom (Mirrored)** - When selected, the corresponding layer is mirrored (shown reversed) in the layout. This is useful for bottom layers, which would be reversed when viewing the top layer.
- **Plot** - Selects whether to plot the corresponding layer, when generating output (printing, exporting as Gerber, etc.)
- **Etch Factor** - Adds an etch factor to the corresponding layer, using the Layout units (mils, etc.) selected on the General tab.
- **Use** - Check this box to include the corresponding layer in the EM simulations (Momentum and/or Empower); uncheck it to omit the layer from EM simulations.
- **Momentum-only Parameters:**
    - **Use Layer Mesh Density** - When checked, Momentum will override its default layer mesh density with the per-layer value specified in the next column on the right.
    - **Mesh Density** - The mesh density, specified as an integer number of cells / wavelength; the default is 30.
    - **Use Layer TL Mesh** - When checked, Momentum will override its default layer transmission line mesh setting with the per-layer value specified in the next column.
    - **TL Mesh** - Transmission Line Mesh - An integer number of cells / width. 0 is automatic.
    - **Edge Mesh** - When checked, Momentum will override its default edge mesh width setting with the per-layer value specified in the next column.
    - **Edge Mesh Width** - The width of the edge mesh, using the Layout units (mils, etc.) selected on the General tab.
    - **Via Model**
        - **Default** - Use the via setting specified in the Momentum GX analysis.
        - **Lumped** - simulate the via using lumped parts.
        - **1D, 2D, 3D** - Use 1 (wire), 2 (planar - no horizontal currents), or 3-dimentional (spacial - includes horizontal currents) simulation for the via.
    - **Strip Model** - not available when the layer is a "physical slot"
        - **Default** - Use the strip model setting specified in the Momentum GX analysis.
        - **2D, 3D** - Use 2 or 3-dimensional simulation.

  - **Precedence** - metal layer precedence for Momentum mesher (from 0 to 9, default = 0).
- **Type** - Other entries in this table are different for each type of layer. To find help on each column, see the section below for the type of layer: Top/Bottom Cover, Air Above/Below, Metal, or Substrate layers.
- **Top Cover & Bottom Cover** - Describes the top and bottom covers (ground planes) of the circuit. Types include:
  - **Lossless:** The cover is ideal metal.
  - **Physical:** The cover is lossy. These losses are described by Rho (resistivity relative to copper). Momentum assumes a cover thickness of 0, while Empower uses the Thickness and Surface Roughness parameters as well.
  - **Electrical:** The cover is lossy and is described by an impedance or file. See the description below under metal for more information.
  - **Open:** In Momentum, no cover will be simulated, also "Open" is not supported in box simulation mode. In Empower (which always uses a box), the design is simulated as if the box walls and uppermost substrate/air layer extend up or down forever (an infinite tube). In earlier versions of Genesys, this option was known as "Semi-Infinite Waveguide".
  - **Magnetic Wall:** The cover is an ideal magnetic wall. This setting is only used in advanced Empower (no Momentum) applications.
  - **Substrates:** Choosing a substrate causes the cover to get the rho, thickness, and roughness parameters from that substrate definition. We recommend using this setting whenever possible so that parameters do not need to be duplicated between substrates and layouts. Momentum ignores Thickness and Surface Roughness. For cover it always uses 0 thickness.
- **Air Above & Air Below** - The presence of air at the top of the box (as in microstrip) or the bottom of the box (as in suspended microstrip) is so common that special entries have been provided for these cases. Checking the box to turn these layers on is the equivalent of adding a substrate layer with Er=1, Ur=1, and Height (in units specified in the Dimensions tab) as specified. For "Open" cover the Height of the cover air is ignored, when the other Air dielectric parameters (Er, Ur, Tand) define the free space dielectric parameters.

  > Caution: When setting up a new circuit, be sure to check the height of the air above, as it is often the only parameter on this tab which must be changed, and is therefore easily forgotten.

- **Metal Layers** - Metal layers are used for metal and other conductive material such as resistive film. The following types are available:
  - **Lossless:** The layer is ideal metal.
  - **Physical:** The layer is lossy. These losses are described by Rho (resistivity relative to copper), Thickness, and Surface Roughness.
  - **Electrical:** The layer is lossy and is described by an impedance or file. This type is commonly used for resistive films and superconductors. If the entry in this box is a number, it specifies the impedance of the material in ohms per square. If the entry in this box is a filename, it specifies the name of a one-port data file which contains impedance data versus frequency. This data file will be interpolated/extrapolated as necessary. See the Device Data section for a description of one-port data files.
  - **Substrates:** Choosing a substrate causes the layer to get the rho, thickness, and roughness parameters from that substrate definition. We recommend using this setting whenever possible so that parameters do not need to be duplicated between substrates and layouts.

    > **Caution:** Unless thick metal is selected, thickness is only used for calculation of losses. It is not otherwise used, and all strips are calculated as if they are infinitely thin.

Metal layers have additional settings available:

- **Metal Thickness** - Thickness of the metal layer on a substrate.
- **Rho** - "Resistivity (relative to copper) is usually 1.0.
- **Current Direction** - Specifies which direction the current flows in this layer. The default is along X and Y. "X Only" and "Y Only" can be used to save times on long stretches of uniform lines. "Z Up", "Z Down", "XYZ Up", and "XYZ Down" allow the creation of thick metal going up/down to the next level or cover.
- **Thick Metal** - Checking this box instructs the EM analysis to model the metal including thickness. Empower does this by putting two metal layers close together, duplicating the traces on each, and connecting them with z-directed currents. If thick metal is used, then Current Direction is ignored.
- **Part Z-Ports** - Specifies the default direction for automatically created part ports, either to the level above or to the level below. Generally, you should choose the electrically shortest path for this direction.
- **Physical Slot** - Slot Type - In Momentum, if the **Momentum Slot-Type** combo-box is set to "Slot", check this box to swap the metal and non-metal areas of the metal layer. (In Strip mode, this setting is ignored). In Empower, check this box to simulate the non-lossless-metal areas (as opposed to the metal areas). Use this for ground-planes and other layers which are primarily metal. Do not use this for lossy layers. See your Empower manual for details.
- **Rough** - Check this box to specify metal roughness in Empower. (This field is ignored by Momentum.)
- **Substrate Layers** - These layers are used for substrate and other continuous materials such as absorbers inside the top cover. An unlimited number of substrate/media layers can be used. The following types are available:
    - **Physical w/Tand:** The layer is lossy. The layer is described by Height (in units specified in the Dimensions tab), Er (relative dielectric constant), Ur (relative permittivity constant, normally 1), and Tand (Loss Tangent).
    - **Physical w/Sigma:** The layer is lossy. The layer is described by Height (in units specified in the Dimensions tab), Er (relative dielectric constant), Ur (relative permittivity constant, normally 1), and Sigma (Bulk Conductivity).
    - **Substrates:** Choosing a predefined Substrate causes the cover to get the height, Er, Ur, and Tand parameters from that substrate definition. We recommend using this setting whenever possible so that parameters do not need to be duplicated between schematics and layouts.

    > ⚠ **Caution:** For true stripline (triplate), be sure to check the **Use 1/2 Height** check box if you are using a Substrate. This forces Empower to use 1/2 of the Substrate height for each substrate (above and below) so that the total height for both media layers is correct.

Substrate layers have additional settings available:

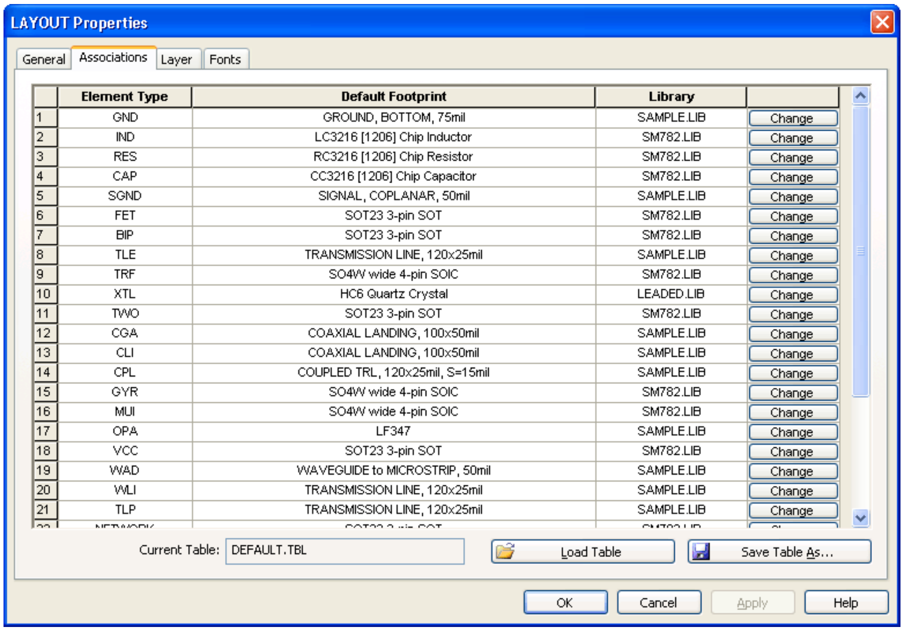
- **1/2 Height** - If the layout is a stripline circuit and is using a named substrate, checking this box forces Empower to use 1/2 of the Substrate height for each substrate (above and below) so that the total height for both media layers is correct.
- **Height** - Height (thickness) of the substrate.
- **Er** - The "relative dielectric constant quantifies a material's ability to store charge when used as a capacitor dielectric, which affects the properties of transmission lines. The higher the constant the higher the energy stored within the capacitor. Er is defined as the ratio of a material's capacitance to the capacitance of air. (Air = 1.0)
- **Tand / Sigma** - Tand and Sigma share the same column, since the parameters are mutually exclusive. Tand - the "dielectric loss tangent - is defined as the real part of relative permittivity / the Imaginary part of the relative permittivity. Sigma is the dielectric "Bulk Conductivity" value.
- **Ur** - The Relative Permeability of the substrate is also know as the magnetic constant is usually 1.0.

- **Surface Impedance Value or File** - The impedance of the surface of the substrate OR the name of a file Empower (only) to use instead. This field is only available for metal and covers, when **Type** is set to Electrical.
- **Momentum Slot-Type** - This setting affects any layer with a check mark in the **Physical Slot** column. The combo-box has two choices, which indicate how Momentum should analyze the layout.
    - **Strip** - Momentum will interpret metal layers with "Physical slot" checked just like Empower, i.e. without swapping the layer's metal and non-metal areas. This allows for the creation of layouts which are compatible with both Empower and Momentum: Empower never swaps metal with non-metal areas in the "physical slot" metal layers - it simulates "Slot" layers internally.
    - **Slot** - Each layer's "Physical slot" check box indicates that Momentum should swap its metal and non-metal areas. (Momentum changes the slot layer topology of the layout).
- **Insert Layer** - Inserts a new layer at the current cursor position.
- **Delete Layer** - Deletes the layer at the current cursor position.
- **Up / Down** - Moves a layer up (or down) in the layer stack.
- **Show / Hide All** - Shows or hides all the layers (by checking all the check-boxes in the Hide column).
- **Use All / Use None** - Checks all the "Use" column check-boxes, which indicates which layers will be simulated by Momentum or Empower.
- **Load From Layer File** - Loads a new layer configuration from a file.
- **Save To Layer File** - Saves the current layer configuration to a new file. Currently, only general and color info is retained (no Momentum, EM, or Empower layer data is saved).

Changing Layout Associations Properties

Use the Layout Properties Associations page to determine which footprint to initially use for each type of design part using the Association table. The Association table is generally modified only when a new layout is created. It is automatically saved using the current name when you close the window, or you can save the table to a new name. You can also load a previously saved table.

Note: You must save the .TBL file into the {product-name}\LIB subdirectory. You cannot use the Association table if it is saved into other directories.



	Element Type	Default Footprint	Library	
1	GND	GROUND, BOTTOM, 75mil	SAMPLE.LIB	Change
2	IND	LC3216 [1206] Chip Inductor	SM782.LIB	Change
3	RES	RC3216 [1206] Chip Resistor	SM782.LIB	Change
4	CAP	CC3216 [1206] Chip Capacitor	SM782.LIB	Change
5	SGND	SIGNAL, COPLANAR, 50mil	SAMPLE.LIB	Change
6	FET	SOT23 3-pin SOT	SM782.LIB	Change
7	BIP	SOT23 3-pin SOT	SM782.LIB	Change
8	TLE	TRANSMISSION LINE, 120x25mil	SAMPLE.LIB	Change
9	TRF	SO4W wide 4-pin SOIC	SM782.LIB	Change
10	XTL	HC6 Quartz Crystal	LEADED.LIB	Change
11	TWO	SOT23 3-pin SOT	SM782.LIB	Change
12	CGA	COAXIAL LANDING, 100x50mil	SAMPLE.LIB	Change
13	CLI	COAXIAL LANDING, 100x50mil	SAMPLE.LIB	Change
14	CPL	COUPLED TRL, 120x25mil, S=15mil	SAMPLE.LIB	Change
15	GYR	SO4W wide 4-pin SOIC	SM782.LIB	Change
16	MUI	SO4W wide 4-pin SOIC	SM782.LIB	Change
17	OPA	LF347	SAMPLE.LIB	Change
18	VCC	SOT23 3-pin SOT	SM782.LIB	Change
19	WAD	WAVEGUIDE to MICROSTRIP, 50mil	SAMPLE.LIB	Change
20	WLI	TRANSMISSION LINE, 120x25mil	SAMPLE.LIB	Change
21	TLP	TRANSMISSION LINE, 120x25mil	SAMPLE.LIB	Change
22	NETWORK	SOT23 3-pin SOT	SM782.LIB	Change

Current Table: DEFAULT.TBL

- **Part Type** - The category of parts for which footprints are loaded.
- **Default Footprint** - The footprint which is selected for the corresponding category (given in the Name column).
- **Library** - The name of the library containing the footprint for the corresponding category (given in the Name column).
- **Change** Button - Allows you to select a new footprint for the corresponding category (given in the Name column).
- **Current Table** - The file name of the current footprint association table.
- **Save Table As Button** - Saves the current table into a new file.
- **Load Table** Button - Loads a different footprint association table from a file.

To modify an entry in the Association table:

1. Click **Layout** on the Genesysmenu and select **Layout Properties**.
2. Click the **Associations** tab.
3. Scroll the table to find the entry you want.

1. Click the **Change** button to open the Choose Footprint Library window.
2. Click a footprint library in the Select Library box.
3. Click a footprint in the Available Footprints box.
4. Click **OK**.

To save the Association table to a new file:

1. Click **Layout** on the Genesysmenu and select **Layout Properties**.
2. Click the **Associations** tab.
3. Click the **Save Table As** button.
4. Type a name in the File Name box.
5. Click **Save**.

To open a previously saved Association table:

1. Click **Layout** on the Genesysmenu and select **Layout Properties**.
2. Click the **Associations** tab.
3. Click the **Load Table** button.

> **Note:** If the current table is already modified, a message asks whether to save the current table. Click **Yes** to continue.

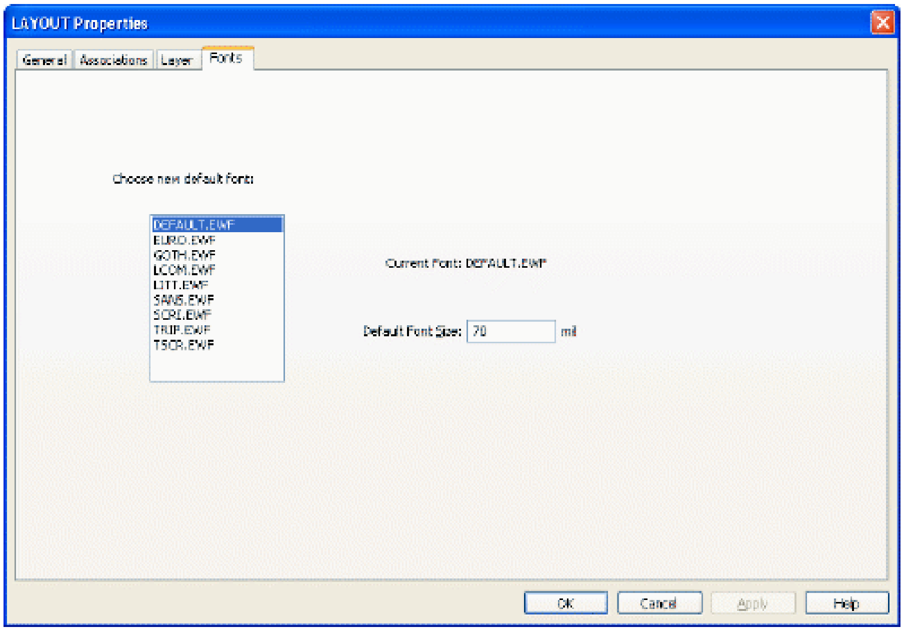
1. Type a name in the File Name box.
2. Click **Open**.

Changing Layout Font Properties

Use the Layout Properties Fonts page to change the default font properties for a layout. You can change only the font type and size. If you want to change the individual text in a layout, use the Text Properties window.

To change the default font properties of a layout:

1. Click **Layout** on the Genesysmenu and select **Layout Properties**.
2. Click the **Fonts** tab.
3. Click a font in the Choose New Default Font box.
4. Type a font size in the Default Size box.
5. Click **OK**.



Choose new default font:

DEFAULT.EWF
EURO.EWF
GOTH.EWF
LCOM.EWF
LITT.EWF
SANS.EWF
SCRI.EWF
TRIP.EWF
TSCR.EWF

Current Font: DEFAULT.EWF

Default Font Size: 70 mil

- **Choose New Default Font** - Lists the available fonts. The default font is automatically selected when the dialog is opened. To choose a new default font, simply select another font in this box. All text already placed on the layout will be updated to incorporate the new font. For a description of each font, please refer to *Reviewing Layout Fonts* (users).
- **Old Default** - The old default font. If you have not selected a new font since opening the dialog, this font stays selected in the Choose New Font box. This font will remain the default style if you select Cancel on the dialog, even if you have selected another.
- **Default Size** - This is the default size for text placed on the layout. Changing this number will update any text already on the layout that has the Use Default Size box checked in its properties box.

Manipulating Layouts

Objects in a layout are manipulated much like objects in any modern drafting or drawing program. Features like selecting objects, dragging, using object handles, grouping, and panning are included.

Changing Object Properties

Object properties windows are used to adjust any properties of an object. These window are especially useful when you must enter exact coordinates for objects. They appear automatically during construction of ports, text, viaholes, and pads.

To change object properties:

1. Double-click any of the following objects in a Layout window:
   - ARC
   - Component
   - EM Port
   - Group
   - Line
   - Pad
   - Polygon
   - Port
   - Pour
   - Rectangle
   - Text

- Viahole

2. Make the changes you want.
3. Click **OK**.

Selecting Objects

Generally, you must select objects before you can manipulate them. Objects change color when selected.
To select an object:

1. Click the Select button ( ) on the Layout toolbar. This button indicates that no object is currently being constructed.
2. Click the object you want to select.

To select multiple objects:

1. Click the Select button ( ) on the Layout toolbar.
2. Drag the mouse until a box is drawn completely around the items you want selected.

> **Note:** The window pans automatically if you drag the mouse off the layout.

To individually select multiple objects:

1. Click the Select button ( ) on the Layout toolbar.
2. Hold down the **Shift** key while clicking each object. Other selected objects remain selected.

To select all objects:

1. Click **Edit** on the Genesys menu and select **All** from the Select menu. All the objects in the layout are selected.

To select an object hidden behind other objects:

1. Click the Select button ( ) on the Layout toolbar.
2. Click where the object should be. If several objects overlap, continue clicking in the same spot until the object is selected. The layout cycles through the overlapped objects with each click. Do not click too fast (double-click) because the object's properties window might appear.

> **Note:** Another method of finding hidden objects is to use the X-ray mode. This makes the parts semi-transparent. You can find this option by using the General tab in the Layout Properties window.

Grouping and Ungrouping Objects

With the exception of port objects, you can combine objects into groups. This keeps objects together as you move or rotate them. You can also nest groups so that one group contains other groups. When objects are grouped, there is no change to the final output generated by the layout.
Whenever any part of a group is selected, the entire group is selected. If you want to ungroup parts after they are grouped, you must break apart the entire group.
To group two or more objects:

1. Select the objects you want to group.
2. Click the Group Objects button ( ) on the Layout toolbar.

To ungroup previously grouped objects:

1. Select the group you want to ungroup.
2. Click the Ungroup button ( ) on the Layout toolbar.

Converting a Component to a Group

When a component is placed in a layout, you can make only simple changes to it:

- Move, rotate, or move the component to a new layer.
- Change or remove text.
- Move text using object handles.
- Hide the silk screen using the Component Properties window.

If you need to make other changes to the object, use the Footprint editor. However, if the change is unique, such as removing unused pins from an edge connector, make the change in the Layout window by first converting the component to a group.
To convert a component to a group:

1. Click the Component Footprint button ( ) on the Layout toolbar to place the component if it is not already in the layout.
2. Select the component you want to group.
3. Click the Group Objects button ( ) on the toolbar.

> **Note:** If a message appears stating you cannot change the object to a group because it has associated schematic parts, follow the instructions in the Deleting Objects section to remove the link.

Using Object Handles

You can modify objects in various ways using object handles. Object handles are small squares that appear when an object is selected. You can manipulate object handles only with a mouse.

> **Note:** There is a limited ability to manipulate objects if there is an associated schematic part.

On rare occasions, you might want to manipulate object handles without snapping to the grid, nodes, or a specific angle. Be aware that when doing so, it is easy to break electrical connections or to rotate parts to nonstandard angles, so only do so when necessary. **To use an object handle:**

1. Select the objects you want to move.
2. Drag an object handle to manipulate the object. An outline image updates as changes are made.

> **Note:** The window pans automatically if you drag the mouse off the layout.

To use an object handle without snapping to grid, nodes, or angles:

1. Select the objects you want to move.
2. Hold down the **Shift** key and drag an object handle to manipulate the object. An outline image updates as changes are made.

   > **Note:** The window pans automatically if you drag the mouse off the layout.

Moving Objects

Often, you must move objects to complete a layout. In addition, on rare occasions, you might want to move objects to a point not on the grid.

> ⚠ **Caution:** It is easy to break electrical connections, so please move objects to a point not on the grid only when absolutely necessary.

To move an object:

1. Select the object you want to move.
2. Drag the mouse until the outlined image is in the location you want. Do not drag using an object handle. The image snaps to the grid.

   > **Note:** The window pans automatically if you drag the mouse off the layout.

To move an object without snapping to grid or nodes:

1. Select the object you want to move.
2. Hold down the **Ctrl** key and drag the mouse until the outlined image is in the location you want. Do not drag using an object handle.

Cutting, Copying, and Pasting Objects

You can cut, copy, or paste layout objects. When you cut an object, you remove it from the layout. Copying an object leaves the object in the layout. Once cut or copied to the buffer, you can paste duplicates of objects back to the layout.

> **Note:** You cannot cut objects associated with schematic parts. If a message appears stating that you cannot cut an object because it has associated schematic parts, use copy instead.

Cutting and pasting objects allows for the easy duplication of objects. The layout remembers the last cut or copied object until you exit Genesys. Only one cut or paste buffer is used for both the standard Layout editor and the Footprint editor, allowing you to cut and paste objects between them. You can paste as many duplicates as you want. Duplicates are always pasted to the same place as the original objects. You can move them to a different location.

> ⚠ **CAUTION:** The contents of the cut and paste buffer are lost when you exit Genesys.

To cut an object:

1. Select the object you want to cut.
2. Click **Edit** on the Genesys menu and select **Cut**.

To copy an object:

1. Select the object you want to copy.

2. Click **Edit** on the Genesys menu and select **Copy**.

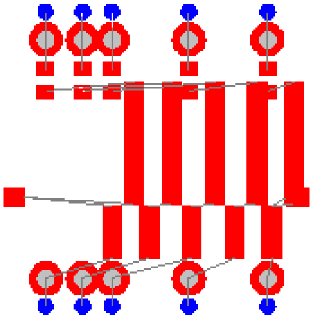
To paste duplicates of the last object cut or copied:

1. Click **Edit** on the Genesys menu and select **Paste**.
2. Move the object to the location you want.

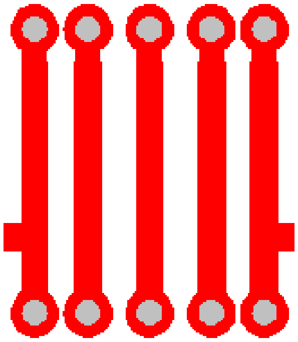
> **Note:** Objects you paste are not associated with schematic parts even if the original objects have associated schematic parts.

Connecting Layout Parts Automatically

A layout can automatically snap parts together. This is very useful for microstrip and stripline circuits that have parts automatically created from a schematic (such as files from M/FILTER). For example, in two easy steps you can turn this layout:



into this layout:



To automatically connect objects:

1. Select the objects you want to connect.
2. Click **Layout** on the Genesys menu and select **Connect Selected Parts**.

Adding Lines, Rectangles, and Arcs

Lines and arcs are typically used to make electrical connections. In layouts strictly for EMPOWER, you should turn off the round ends for these lines and arcs. For most other purposes, you should use round ends, because they make better connections. Also, in layouts for EMPOWER, you might find it easier to use rectangular objects for most purposes.
Using the Layout toolbar, you can set round or square ends, width, and layer information for lines and arcs. You can also make lines orthogonal (90 degrees) using the Layout toolbar or the Line properties window.
To draw a straight line:

1. Click the Line button (▬) on the Layout toolbar.

2. Click in the layout at the starting point of the line.
3. Drag your mouse to the ending point of the line.

To draw two connected orthogonal (90 °) lines:

1. Click the Line button ( ) on the Layout toolbar and draw a straight line.
2. Draw a second line perpendicular to the first line and connect one end of the second line to one end of the second line (green dots).
3. Click one of the lines.
4. Press the **O** key to convert the line to two orthogonal segments.
5. Press the **F** key to flip the orthogonal direction, if necessary.

To draw a rectangle:

1. Click the Rectangle button ( ) on the Layout toolbar.
2. Click in the layout where you want the upper-left corner of the rectangle.
3. Drag the mouse to the lower-right corner of the rectangle.

To draw an arc:

1. Click the Arc button ( ) on the Layout toolbar.
2. Click in the layout at the starting point of the arc.
3. Drag your mouse to the ending point of the arc. You see a thin line between the starting point and the mouse position as you drag the mouse.
4. Move the mouse to define the curvature of the arc.
5. Click to finish drawing the arc.

Deleting Objects

When you delete objects from a layout, the objects are not placed into a buffer the same as when you cut or copy objects. Deleting removes objects from the program, and you cannot paste deleted objects back into the layout.

> **Note:** You cannot directly delete layout objects with associated schematic parts. Instead, you must either delete the part from the schematic, remove the schematic from the layout, or set the schematic object to not include a layout object.

To delete layout objects:

1. Select the object you want to delete.
2. Click **Edit** on the Genesys menu and select **Delete**.

To remove a schematic from a layout:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **General** tab.
3. Click the check box for your layout in the Designs to Include box.
4. Click **OK**. All parts and rubber bands from the schematic are removed.

To remove a layout object with an associated schematic part:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **Schematic** tab.
3. Click the schematic part that corresponds to the layout part you want to delete.

4. Double-click the part to open its properties window.
5. Click the **Layout** button.
6. Click either the **Replace Part With Open** or **Replace Part With Short** button.
7. Click **OK** to close the Layout Options window.
8. Click **OK** to close the properties window.

Changing an Association Table

The Association table is used whenever a new layout is created or any time a layout is updated with new parts in the design. It is only used once for any given part. Association tables are used by a layout to determine which footprint to initially use for each type of design part. You can later change this footprint from within the layout to override the Association table.

Whenever a file containing a layout that depends on a schematic is loaded, the date of the schematic file is checked against the date of its Association table. If the Association table is newer than the schematic file, a message appears. This message is useful to help ensure that outdated footprints are not accidentally used in a layout.

The following information helps clarify the use of an Association table:

- A new schematic is created containing a capacitor, C1.
- A new layout is created. In the Association table, the entry for CAP shows the library name SM782.LIB, and the footprint shows the name CC1005 [0402] Chip Capacitor. This footprint is automatically placed in the layout for the capacitor.
- If the Association table is later changed, it has no effect on the capacitor that was already placed.
- If the footprint for the existing capacitor is changed (for example, to CC2012 [0805] Chip Capacitor), it is not automatically changed back to the Association table entry.
- Even if the schematic is modified, the capacitor does not automatically move or change back to the Association table entry.
- If the schematic part is deleted, the corresponding part in the layout is deleted.
- If the Association table contains a multi-part footprint (such as a quad op-amp footprint), all devices in each component is used before beginning a new one. For example, if a schematic contains seven op-amps and the OPA entry in the Association table contains a quad op-amp footprint, two components are placed. The first component uses all four devices, and the second component uses three of its four devices.

Changing the Component Footprints

Once a component is automatically generated for a schematic part, you can change its footprint. The component continues to use the new footprint even if the schematic is modified.

To change a component's footprint:

1. Select the component you want to change.
2. Click the Change Footprint button ( ) on the Layout toolbar to open the Choose Footprint Library window.
3. Click a footprint library in the Select Library box.
4. Click a footprint in the Available Footprints box.
5. Click OK.

Adding Text and Changing Fonts

You can add text to your layout to label objects or to convey information on silk layers. Included in Genesysis a variety of different fonts that you can use for text objects in a layout. You can also use your own TrueType font.

Note: The text changes you make to a specific layout affect only the text in that layout. If you want to change the default text font and size for all layouts, you must use the Layout Properties Font window.

Adding Text to Layouts

Once you add text to a layout, you can edit it, change the font, or change the size.
To add text to a layout:

1. Click the Text button (A) on the Layout toolbar.
2. Click in the layout where you want to place text. The Text Properties window appears.
3. Type the text in the Text box.
4. Make any other changes you want.
5. Click OK.

To change the text in an existing text object:

1. Select the text you want to edit.
2. Click Edit on the Genesysmenu and select Parameters to open the Text Properties window.
3. Edit the text in the Text box.
4. Click OK.

To change the font used by an existing text object:

1. Select the text whose font you want to change.
2. Click Edit on the Genesysmenu and select Parameters to open the Text Properties window.
3. Select a font in the Font list.
4. Click OK.

To change the size of an existing text object:

1. Select the text whose size you want to change.
2. Click Edit on the Genesysmenu and select Parameters to open the Text Properties window.
3. Clear the Use Default Size check box if it is selected.
4. Type the new text size in the Text Size box. The size is given in the units specified in the General tab of the Layout Properties window.
5. Click OK.

Changing the Default Text

Most text placed in a layout is associated with a footprint, and you cannot control the font and size. However, you can change the default font and size, which changes the text for all other objects simultaneously. This process adjusts the font and size of all text objects that are marked (in their properties windows) to use the default font or the default size. This includes all text in footprints from libraries supplied with layouts.
To change the default font and size:

1. Click Layout on the Genesysmenu and select Layout Properties.
2. Click the Fonts tab.
3. Click a font in the Choose New Default font list.
4. Type a new size in the Default size box.

5. Click OK.

Reviewing Layout Fonts

The provided fonts are all in Eagleware-Elanix font (.EWF) format, which is a proprietary font format. They are located in the Font subdirectory of the main Eagleware-Elanix directory (C:{product-name}\Font). If additional fonts are needed, you can copy TrueType fonts (.TTF) into this directory, with the following constraints:

- There are many different TrueType font formats available and not all are guaranteed to work in a layout.
- When text using a TrueType font is converted to Gerber format, each letter of the text is converted into a filled polygon. Extremely large Gerber files result if TrueType fonts are used extensively. TrueType fonts are best used for highlights and user instructions, such as a company's name and logo or jumper settings. In contrast, the size of text in a Gerber file using an .EWF font is roughly proportional to the complexity of the .EWF font.

Layout includes the following fonts (complexity directly relates to Gerber file size):

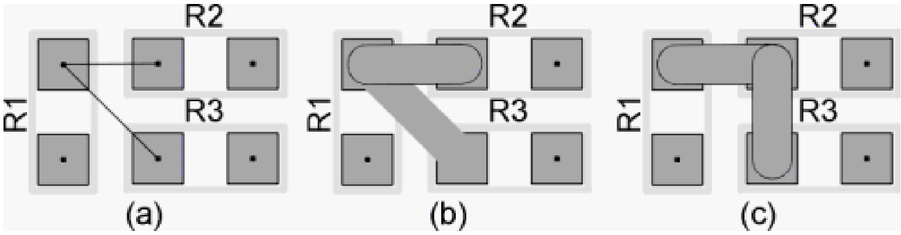
Font	**Description**	**Complexity (1-10)**
DEFAULT.EWF	A thin font. This font is appropriate for use on most boards.	4
EURO.EWF	A thin font that is somewhat boxier than DEFAULT.EWF.	4
GOTH.EWF	A very artistic gothic font.	10
LCOM.EWF	A font resembling Times Roman. This font is especially useful for placing a large company name on a layout.	6
LITT.EWF	A very simple thin font. This is the simplest provided font.	2
SANS.EWF	A sans serif font of medium stroke thickness.	7
SCRI.EWF	A thin ornamented font designed to resemble handwriting.	5
TRIP.EWF	A bold font resembling Times Roman Bold.	9
TSCR.EWF	A bold italic font resembling Times Roman Bold Italic.	9

Reviewing Nodes and Rubber Band Lines

Nodes (shown as green dots) are shown in a layout where you can make electrical connections. Any objects placed or handles moved snap automatically to a nearby node to ensure a true connection. These nodes allow consistent electrical connections even when parts do not line up on a grid. (If you do not want to snap to the grid or to a node, hold down the Shift key while moving or constructing an object.)

Rubber band lines (shown as thin white lines) are shown in a layout where you can make electrical connections. These connections are determined from the designs. The rubber band lines update whenever the designs change. Rubber band lines disappear automatically as connections are made.

Rubber band lines show only one possible set of connections. For example, figure (a) below shows three parts that can connect. The most obvious connection method is shown in figure (b). However, you do not need to follow the rubber band lines exactly. The connections in figure (c) also resolve the rubber bands because they electrically connect the three nodes.



You cannot use objects other than lines, arcs, or viaholes to resolve rubber bands. For example, if a polygon is placed between two nodes, those two nodes are not considered connected, even if the polygon appears to connect the two nodes.

> **Note:** One cause of rubber band lines not disappearing is the failure to actually connect nodes together. If rubber bands do not disappear, zoom in and examine the nodes to ensure the connections are properly made.

The automatic resolution, or removal, of rubber band lines is intelligent. You can use any combination of lines, arcs, and viaholes to make connections. The layout resolves the rubber bands properly no matter how complex the interconnections.

> **Tip:** A common mistake is to use a fine grid, which allows components to look connected when they actually have a small gap between them. This problem is often hard to find in large layouts, but you can ensure that actual connections take place by keeping the grid spacing coarse.

Reviewing Rubber Band Resolutions

You can use the Statistics window (sometimes referred to as a scorecard) to review your progress resolving rubber bands. This window tells how many rubber bands there were initially and how many rubber bands are successfully resolved.

To review your progress resolving rubber bands:

- Click **Layout** on the Genesys menu and select **Statistics**.

Using Layers

A layout recognizes six distinct layer types, which are considered a Layer table:

- **Metal** – Used for all conductive traces. Only traces on metal layers are used to automatically resolve rubber bands.
- **Silk** – Used for labels on the final board. It is generally white or yellow on the production board. Silk screen objects should not overlap with solder mask objects.
- **Substrate** – Used to designate cuts in the circuit board. If automatic cutting is employed, you should use only straight or orthogonal rounded-end lines. The center of these lines represents the saw path for cutting.
- **Assembly** – Used to indicate exact placement of components. This is an intermediate layer. It is only used as an aid in the production process and is not seen on the final board.
- **Mask (Solder Mask)** – Objects indicate an absence of solder mask. This is a negative layer. It is automatically generated from pads and viaholes and generally does not require user intervention.
- **Paste (Solder Paste)** – Objects indicate places to use solder paste. If a solder paste layer is required, you must manually generate it.

> **Note:** You can enter layer information using the **Layer** tab of the **Layout Properties** window.

Reviewing the Types of Layers

A layout can deal with almost any board configuration, from a simple one-layer board to a complex sixteen or more layer board. Mixed media, such as combining microstrip with stripline, are easily accommodated. There are up to 128 different layers to use, and you can use each type as often as you want. Some possible layer setups are:
For a simple, single-layer board or prototype with a solid ground plane (no traces or cutouts on the ground plane) (SIMPLE.LYR):

- Silk
- Metal

For a typical single-layer production board (SINGLE.LYR):

- Top Assembly
- Top Silk
- Top Mask
- Top Metal
- Substrate
- Bottom Metal (Mirrored)
- Bottom Mask (Mirrored)
- Bottom Silk (Mirrored)
- Bottom Assembly (Mirrored)

For a four-layer production board (FOUR.LYR):

- Top Assembly
- Top Silk
- Top Mask
- Top Metal
- Substrate 1
- Metal 2
- Substrate 2
- Metal 3
- Substrate 3
- Metal 4
- Substrate 4
- Bottom Metal (Mirrored)

- Bottom Mask (Mirrored)
- Bottom Silk (Mirrored)
- Bottom Assembly (Mirrored)

Selecting a Drawing Style

Several drawing styles are available to facilitate the construction and manipulation of objects on various layers. The drawing styles are:

- Solid (Opaque)
- X-ray Mode
- Hollow

To select a drawing style:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **General** tab.
3. Click a button to indicate the drawing mode.

Note: Use the hollow or x-ray modes to make all objects visible, regardless of the layer.

4. Click **OK**.

Hiding Layers

Often it is necessary to turn off the display of certain layers. For example, if an assembly layer is not being modified, you can hide it:
To hide a layer:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **Layer** tab.
3. Scroll down to the layer you want.
4. Click the check box in the Hide column.
5. Click **OK**.

Mirroring Layers

Be sure to mirror layers on the back side of a board. For example, when viewing from the top of the board, the lettering and component footprints are reversed. When a layer is marked as mirrored, all future text and components placed on that layer are mirrored.
To mark a layer as mirrored:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **Layer** tab.
3. Scroll down to the layer you want.
4. Click the check box in the Mirror column.
5. Click **OK**.

Using Layer Files

You can save layer setups into files for later use in other boards. Unlike Footprint library files, layer files (LYR) do not remain associated with a particular layout. If layer settings are saved to an LYR file and the settings are later changed in the current layout, the LYR file does not update automatically. If the LYR file is later changed, no layouts are changed unless the LYR file is explicitly loaded.
To save layer settings into an LYR file:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **Layer** tab.
3. Click the **Save to Layer File** button.
4. Type a name in the File Name box.
5. Click **Save**.

To load layer settings from an LYR file:

1. Click **Layout** on the Genesys menu and select **Layout Properties**.
2. Click the **Layer** tab.
3. Click the **Load from Layer File** button.
4. Type the name of the Layer file to load in the File Name box.
5. Click **Open**.

Adding Footprints to Layouts

A footprint is a pattern of metal, silk, and other layers that generally corresponds to a physical part, such as SOT23 or 0603 packages. You can create or edit footprints using the Footprint editor and then save the footprints into a footprint library. If a footprint library is changed, all components in all layouts using that footprint are changed. You can associate footprints with parts using the Association table or by editing a part in the Part Selector to add a footprint property to the part.

Creating and Saving Footprints

You can create and save footprints to use in other layouts. Store the footprints in a library you create.

To create a new footprint:

1. Click **Tools** on the Genesys menu and select **New Footprint** from the Footprint Editor menu.
2. Place objects in the Footprint editor.
3. Place pads wherever you want to make solder connections.
4. Place silk screen objects (for example, designators).
5. Place ports. Always place ports before saving a footprint.
6. Click **Tools** on the Genesys menu and select **Save Footprint** from the Footprint Editor menu to save the footprint in a library file.

To save an existing footprint:

1. Click **Tools** on the Genesys menu and select **Save Footprint** from the Footprint Editor menu.
2. Click the library that contains the old footprint in the Select Library box.
3. Click the footprint in the Available Footprints box.
4. Click **OK**.

⚠ **Warning:** The old footprint is lost if this method is used to save because any layouts using the old footprint are modified. Use a new name if the old footprint is needed later.

To add a new footprint to an existing library:

1. Click **Tools** on the Genesys menu and select **Save Footprint** from the Footprint Editor menu.
2. Click the library to add the footprint to in the Select Library box.
3. Click **<New Object>** in the Available Footprints box.
4. Click **OK**.
5. Type a new name for the footprint in the box.
6. Click **OK**.

To create a new library and save the current footprint into it:

1. Click **Tools** on the Genesys menu and select **Save Footprint** from the Footprint Editor menu.
2. Click **<New File>** in the Select Library box.
3. Ensure that **<New Object>** is selected in the Available Footprints box.
4. Click **OK**.
5. Type a name for the new library file in the File Name box, and then click **Save**.
6. Type a new name for the footprint in the box.
7. Click **OK**.

Creating Multi-Part Footprints

Multi-part footprints (for example, bus resistors) are created by assigning port numbers to different parts within the footprint. The first part (schematic part) within the footprint is usually labeled part A. For example, the first resistor in a bus package is part A.
The first pin of part A is labeled pin 1, so the port that designates the first pin of part A is labeled A1 on the footprint. This is shown in the *Footprint Example 2* (users) section.

Using the Footprint Libraries

Four footprint libraries are included with a layout. You can use these libraries to add footprints to your layouts. Certain footprints, such as 14-pin DIP packages, have hundreds of possible uses. They can be a quad op-amp or a multiple transistor package. Port assignments in these devices are sequential, such as pin 1 through 14. When you use such a device, you must create a specific footprint with different port numbering.
You can create new footprints based on a footprint in one of these footprint libraries. Just load an existing footprint, modify it as necessary, save it with a new name, and store it in a new footprint library.

- **SM782.LIB** - A library based on the IPC SM 782 surface mount standard (SM782.LIB). There footprints are from the Institute for Interconnecting and Packaging Electronic Circuits SM 782 standard, Revision A, August 1993. You can contact IPC at 7380 N. Lincoln Ave., Lincolnwood, IL, 60646.
  This standard is very specific on dimensions for the landing patterns. In general, maximum dimensions are used in creating the library footprints.
  The silk screens are generated by Agilent based on general industry convention determined by reviewing several PWBs.
- **LEADED.LIB** - A leaded component library (LEADED.LIB). These are leaded part footprints. They are generated from data provided by the following manufacturers: Coilcraft, ITT, J.W. Millar, Kyocera AVX, Motorola, Murata, Panasonic, R-Ohm, and Toko.
  The through holes in this library are typically 20 mils larger than the lead diameter, and the pads are 35 mils larger than the through hole diameter. In certain active devices, such as DIP ICs and TO-92 transistors, sufficient spacing is not available and smaller margins are used.
  The silk screens are generated by Agilent based on general industry convention determined by reviewing several PWBs.
- **HPLIB.LIB** - A library of sample footprints (SAMPLE.LIB). These transistor footprints are taken from the HP Communications Components GaAs & Si Products data book.
- SAMPLE.LIB - A small library of active RF devices (HPLIB.LIB). These miscellaneous leaded and surface mount footprints are for objects such as mounting screws, coplanar grounds, grounds with via holes, grounds with wagon wheel pads and a sample quad operational amplifier.

Generating Transmission Lines Automatically

In typical digitally oriented PCB layout or CAD programs, you must manually generate transmission lines and their junctions (discontinuities). This is often tedious, time consuming, and error prone.
In contrast, a layout automatically generates footprints for microstrip and stripline parts entered in a schematic. The dimensions of these footprints are automatically determined from the schematic parts and substrate parameters, ensuring that the board is laid out exactly as simulated. The table that follows lists [part types|superstar_parts] that are handled automatically by a layout (and, correspondingly, are not listed in the Association table).

Part Type	Description
Microstrip and stripline bends (MBN, SBN)	A square or chamfered (triangular) polygon for the corner section.
Microstrip and stripline single lines and coupled lines (MCN, MCP, MLI, MTAPER, SLI, SCN, SCP)	Generates lines and spacings.
Microstrip cross and tee, and stripline tee (MCR, MTE, STE)	A square or rectangular polygon for the area where three or four lines come together.
Microstrip curved line (MCURVE)	A curved line is generated.
Microstrip inductors & capacitors (MIDCAP, MRIND, MSPIND)	Complex footprints representing these parts are generated.
Microstrip and stripline open end effects (MEN, SEN)	Have no size and are removed from the layout.
Microstrip and stripline gaps (MGA, SGA)	Two small metal guide pieces separated by the gap width.
Microstrip radial stub (MRS)	Generates a pie-shaped piece.
Microstrip and stripline steps (MST, SSP)	Removed for symmetrical microstrip and stripline steps. For asymmetrical microstrip, two guide pieces plus nodes are generated.
Microstrip viaholes (MVH)	Generates a viahole with its pads.

Note: Be sure that the units given in the substrate are correct, because these are the units used for generating the microwave footprints. The units specified on the General tab of the Layout Properties window are not used for this purpose.

When a layout containing these parts is created, footprints for these parts are generated automatically and are connected by rubber bands initially just like any other parts. The main difference is that instead of connecting these parts with lines, they are normally mirrored, moved, and rotated until their nodes join, eliminating the rubber band lines. You can manually or automatically join these parts.

To automatically join multiple microwave footprints:

1. Select the objects you want connected.
2. Click **Layout** on the Genesys menu and select **Connect Selected Parts**.

To manually join together two microwave footprints:

1. Select the first object.
2. Click the Mirror button ( ) on the Layout toolbar if the first object must be mirrored.
3. Use the object handles to rotate and move the object as necessary. For unusual positioning or rotation angles, use the Component Properties window.
4. Select the second object.
5. Click the Mirror button on the Layout toolbar if the second object must be mirrored.
6. Use the object handles to rotate the object as necessary. For an unusual rotation angle, use the Component Properties window.A rubber band now connects a node on the first object (node A) with a node on the second object (node B).
7. Drag node B to node A. The two objects snap together, eliminating the rubber band.

Placing Component Objects

Component objects in a layout represent lumped parts, such as resistors, transistors, and integrated circuits. You can also use component objects if a complex pattern needs repeating in many layouts, such as connectors.

All components are based on a footprint in a footprint library. If a footprint library is changed, all components in all layouts using that footprint are changed.

If a layout is based on one or more designs, components and rubber bands are automatically placed corresponding to the design topologies. The **General** tab of the **Layout Properties window** controls which designs to include in the layout. You can use the Association table to determine which footprint to use for each type of part. If any components are added, modified, or deleted in the designs, the layout components and rubber bands are automatically added, modified, or deleted.

To place a component object:

1. Click the Component Footprint button ( ) on the Layout toolbar.
2. Click the library that contains the footprint in the Select Library box.
3. Click the multi-part footprint in the Available Footprints box.
4. Click **OK**.
5. Click in the layout where you want to place the component.

Switching Parts in Multi-Part Footprints

Multi-part footprints such as resistor packs or quad op-amp integrated circuits (ICs) are supported in a layout. You can move two or more individual parts with associated schematic parts into a multi-part footprint.

Sometimes, you might want to switch existing component associations. For example, if there are two quad op-amp packages in the layout, each with four associated schematic parts, you can change which part is associated with which part.

To switch parts with associated schematic parts into a multi-part footprint:

1. Select the part you want to switch.
2. Click the Change Footprint button ( ) on the Layout toolbar.
3. Select the desired multi-device footprint from the library window.
4. Choose Switch/Move Parts from the Layout Menu. For the next several steps, help will appear on screen in the status bar.
5. Select the next component to place into the multi-part package.
6. Select the component with the multi-device footprint to place the second component into.
7. A message will appear stating that the original component no longer is associated with any schematic parts and asking if the original part should be deleted. Normally, you will press the "Yes" button.
8. Repeat steps five through seven for any remaining individual parts to place into the multi-part footprint.

Sometimes, you may want to switch existing component associations. For example, if there are two quad op-amp packages on the layout, each with four associated schematic parts, you may want to change which device is associated with which part.

To switch schematic part associations between two component devices:

1. Choose Switch/Move Parts from the Layout Menu. For the next several steps, help will appear on screen in the status bar.
2. Select the first component's device to switch. This component must have an associated schematic part.
3. Select the multi-device footprint to switch with or move into.
4. If a message appears stating that the original footprint no longer is associated with any schematic parts and asking if the original part should be deleted, you will normally press he "Yes" button.

> **Note:** For information on modifying the Association table to use multi-part footprints, see the *Changing the Associations Layout Properties* (users) section. For information on changing the footprint of an existing part, see the *Changing Component Footprints* (users) section.

Loading and Merging Footprints

To load an existing footprint:

1. Click **Tools** on the Genesys menu and select **Load Footprint** from the Footprint Editor menu.
2. Click the library that contains the footprint in the Select Library box.
3. Click the footprint in the Available Footprints box.
4. Click **OK** to load the footprint.

To merge an existing footprint with the current footprint:

1. Click **Tools** on the Genesys menu and select **Merge Footprint** from the Footprint Editor menu.
2. Click the library that contains the footprint in the Select Library box.
3. Click the footprint in the Available Footprints box.
4. Click **OK** to merge the footprint.

Renaming and Deleting Footprints

You can change the name of a footprint or delete a footprint if it is no longer needed.
To rename a footprint:

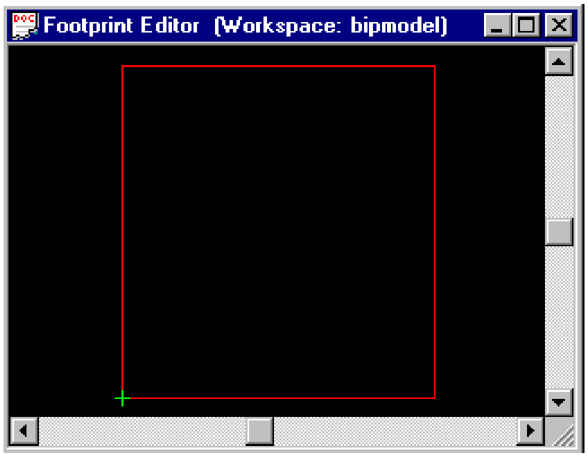
1. Click **Tools** on the Genesys menu and select **Modify Footprint Library** from the Footprint Editor menu.
2. Click the library that contains the footprint in the Select Library box.
3. Click the footprint in the Available Footprints box.
4. Click the **Rename** button.
5. Type a new name for the footprint in the New Name box.
6. Click **OK**.

To delete a footprint:

1. Click **Tools** on the Genesys menu and select **Modify Footprint Library** from the Footprint Editor menu.
2. Click the library that contains the footprint in the Select Library box.
3. Click the footprint in the Available Footprints box.
4. Click the **Delete** button.

Using the Footprint Editor

Layouts in Genesys also include an editor for creating and editing footprints and libraries. The figure below shows a Footprint Editor window.



To open the Footprint editor:

- Click **Tools** on the Genesys menu and select **New Footprint** from the Footprint Editor menu.

Reviewing Layers in the Footprint Editor

The following layers are available in the Footprint editor:

- Top Assembly
- Top Silk
- Top Paste
- Top Mask
- Top Metal
- Sub Above
- Metal
- Sub Below
- Bottom Metal
- Bottom Mask
- Bottom Paste
- Bottom Silk
- Bottom Assembly

For almost all components, place pads on the metal layer and not the top metal layer. Place all silk screen objects on the top silk or bottom silk layers.

> **Note:** If you might ever use your footprint on the back side of a board, you must place your pads on the **bottom** metal layer and not the top metal layer so that the layout automatically mirrors your pads and moves the component to the back of the board.

Placing Objects in the Footprint Editor

Objects in the Footprint editor are placed exactly the same way that they are placed in the Layout window. Whenever you select an object, the Layout toolbar shows the available options for that object. You can change any of these options before placing the object. The available objects are listed on the toolbar at the top of the Footprint Editor window. They are as follows:

- Lines (▬)
- Rectangles (🟥)
- Arcs (⌒)

- Polygons (⬟)
- EM Ports (☒)
- Text (A)
- Viaholes (◉)
- Pads (●)

To place an object in the Footprint editor:

1. Click the appropriate button on the Layout toolbar.
2. Click in the Footprint editor to place the object.

> **Note:** If the object's properties window appears, specify the options you want and then click **OK** .

Placing Pads in the Footprint Editor

You should place pads wherever solder connections are made. Pads always have a node at the center, regardless of the pad shape.

To place a pad in the Footprint editor:

1. Click the Pad button (●) on the Layout toolbar.
2. Click in the Footprint editor to place the pad.
3. Select the options you want in the Pad Properties window.
4. Click **OK**.

Placing Ports in the Footprint Editor

You must place ports before you can save a footprint to a library. Ports designate where layout parts connect (for example, lines, arcs, or other footprints). For multiple component footprints such as a quad op-amp IC, ports identify the different components. For example, port A1 is used for pin 1 of op-amp #1. Port D-3 is pin 3 of op-amp #4.

To place a port on the footprint:

1. Click the EM Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to place the port. The Port Properties window appears.
3. Select the device that the port belongs to from the Device list. For example, if this is the second component in a package, select B.
4. Type the port number in the Port Number box. This is the pin within this device that the port belongs to. For example, if this is pin # 3 of an op-amp (the output), type 3.
5. Make any other changes you want.
6. Click **OK**.

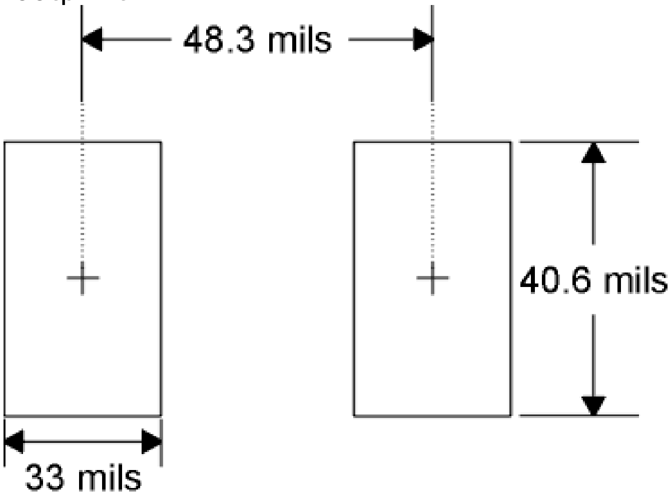
Using Power and Ground Connections

Some multi-part footprints contain extra ports that are not used by any device (marked with * or None, in the Footprint editor). These ports are most commonly used for power and ground connections and are connected manually; no rubber bands appear to ensure their connection.

Caution: Make any necessary power and ground connections to footprints. Again, there are no rubber bands on these pins to help you remember.

Footprint Example 1: 0805 Capacitor

This example demonstrates how to create a layout footprint for an 0805 capacitor in the Footprint editor. The figure below shows the dimensions of a standard 0805 capacitor footprint:



You can create the above example as follows:

1. Create a footprint
2. Place the first pad
3. Place the second pad
4. Draw the silk screen
5. Place the designator text
6. Place footprint ports

To create a footprint:

1. Click **Tools** on the Genesys menu and select **New Footprint** from the Footprint Editor menu.
2. Double-click in the Footprint editor to open the Layout Footprint Editor Properties window.
3. Click the **General** tab.
4. Select **mil** from the Units list.
5. Click **OK**.

To place the first pad:

1. Click the Pad button (●) on the Layout toolbar, and then click in the Footprint editor.
2. Click the **Square/Rect** button in the Pad Properties window.
3. Type **33** in the Pad Width box.
4. Type **40.6** in the Pad Height box.
5. Select **Metal** from the Layer list to place the pad on metal.
6. Type **0**, **0** in the Location boxes to place the pad center at the origin.
7. Click **OK**.

To place the second pad:

1. Repeat steps 1-5 above.
2. Type **48.3**, **0** in the Location boxes as the second pad location. This sets the center-to-center pad spacing to 48.3 mils as shown in the figure.
3. Click **OK**.

To draw the silk screen:

1. Click the Line button (▬) on the Layout toolbar.
2. Draw a line in the Footprint editor, and then double-click the line.
3. Type **10** (mils) in the Line Width box.

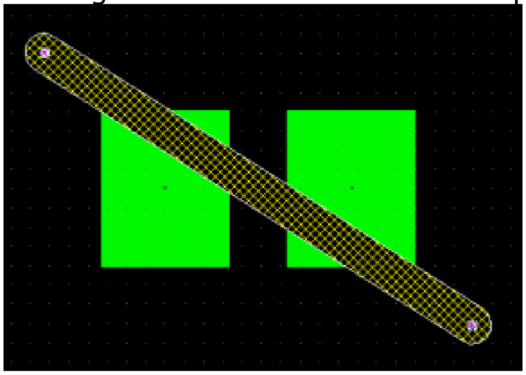
> **Note:** To prevent silk screen interference with metal layer objects, all silk is kept at least 10 mils from the nearest metal.

4. Click the **Rounded Ends** check box to change the line shape to round.
5. Select **Top Silk** from the Layer list to place the line on the top silk layer.
6. Type **-31.5**, **35.5** in the Start boxes.
7. Type **79.8**, **-35.3** in the End boxes.

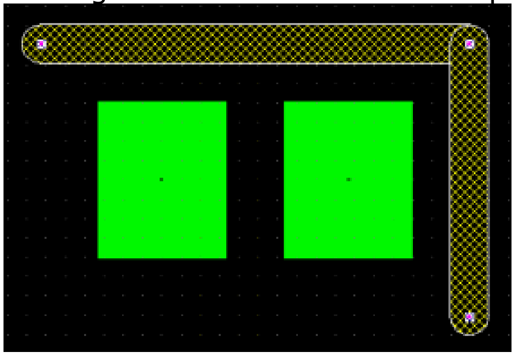
> **Note:** The Start and End figures include 10 mils beyond the pad width plus 5 mils for half the silk line width.

8. Click **OK**.
9. Press the **O** key to create a 90-degree line.

This figure shows a silk line before pressing the O key:



This figure shows a silk line after pressing the O key:



1. Draw another rounded line on silk from **-31.5**, **35.3** to **79.8**, **-35.3**.
2. Press the **O** key to create another angle, and then press the **F** key to flip the angle. This creates a box around the pads with a 10 mil clearance.

To place the designator text:

1. Click the Text button (A) on the Layout toolbar.
2. Click in the Footprint editor to open the Text Properties window.

3. Type **@DES** in the Text box. This allows the schematic part using this footprint to fill in the part designator on the layout.
4. Select **Top Silk** from the Layer list.

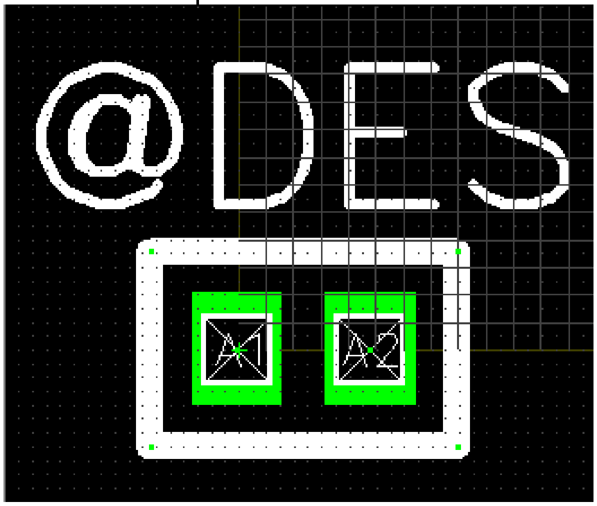
1. Click the **Use Default Size** check box to allow text sizing later when the footprint is used in a layout.
2. Type **24.15**, **50.3** in the Location boxes. This centers the text horizontally and allows a 10 mil vertical clearance for the silk screen box.
3. Click the **Center X** button for X-justification. This forces the text to always center horizontally and keeps the 10 mil clearance from the part box.
4. Click the **Bottom** button for Y-justification. Any other Y-justification allows the text to expand downward with increasing size, breaking the 10 mil separation rule.
5. Click **OK**.

To place footprint ports:

1. Click **Layout** on the Genesys menu and select **Place Footprint Port**.
2. Click the center of the first pad to place a footprint port on that pad. The Port Properties window appears.
3. Type **25** in the Draw Size box.
4. Click **OK**.
5. Repeat steps 1-4 to place a footprint port on the second pad.

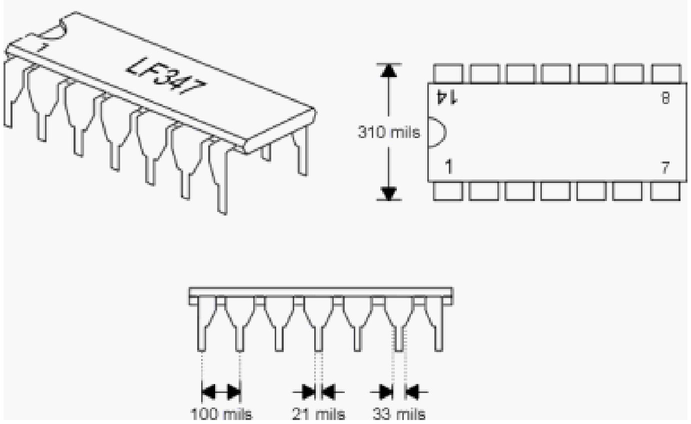
> **Note:** Be sure to place footprint ports on the same metal layer as the pad.

The final footprint is shown below:

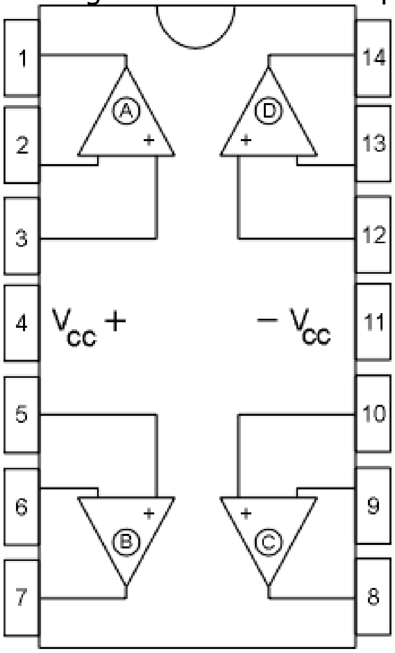


Footprint Example 2: LF347 Quad Op-Amp

This example demonstrates how to construct a footprint for a Texas Instruments LF347 quad op-amp in the footprint editor. The figure below shows the dimensions of the LF347 package:



For this example, you set the current units to mils and place the pads first. Because this is a leaded part, use through holes at each pad location. The pins are 100 mils apart, so set the pad center spacing at 100 mils. The hole diameter must be at least 21 mils to accommodate the pins, but not exceed 33 mils, which is the width of the seating flange. For this example, the pad width is 75 mils, and the through-hole diameter is 30 mils.
The figure below shows a pin-out diagram for the LF347 package:



You can create the above example as follows:

1. Place a viahole and pad
2. Place ports for the first op-amp
3. Place the positive power supply port
4. Place ports for the second op-amp
5. Place ports for the third op-amp
6. Place the negative power supply port
7. Place ports for the fourth op-amp
8. Draw a silk screen box inside the pad perimeter
9. Place the designator text
10. Label the type of package

Please note the following:

- Three ports are used for each of the op-amps shown in the pin-out diagram. Each op-amp is considered a part, so the footprint has four parts with three ports each.
- The port numbers are assigned according to the order that the associated schematic

part uses. The order is found in the schematic part's diagram in the *Reference* manual.

- In the OPA model, the non-inverting input is pin 1, the inverting input is pin 2, and the output is pin 3.

> ⚠ **Warning:** Please adhere to this port numbering convention when creating a footprint. Otherwise, the rubber-band lines created by the layout indicate erroneous connections to make.

To place a viahole and pad:

1. Create a design with a layout.
2. Click the Viahole button (●) on the Layout toolbar.
3. Click in the Layout window to open the Viahole Properties window.
4. Type **30** in the Drill Diameter box.
5. Type **0**,**0** in the Location boxes because this is the first pad.
6. Click the **Square/Rect** button for the pad shape. This identifies pin 1 during assembly.
7. Type **75** in both the Pad Width and Pad Height boxes.
8. Click **OK**.
9. Repeat steps 2-7 to place a viahole with a round pad at location **100**, **0**, which properly spaces the hole centers. Specify **75** mils for the drill diameter and **30** mils for the pad width and height.
10. Place the remaining five pads for the first side at 100 mil increments as in step 8, with the final pad for the first side (labeled pin 7 above) set at **600**, **0**.
11. Place the remaining viaholes at the same X-offset as the first side, with a Y-offset of **310** mils (the distance given in the first figure above for the pin-to-pin cross dimension). The drill diameter and pad dimensions are the same as before.

To place the ports for the first op-amp:

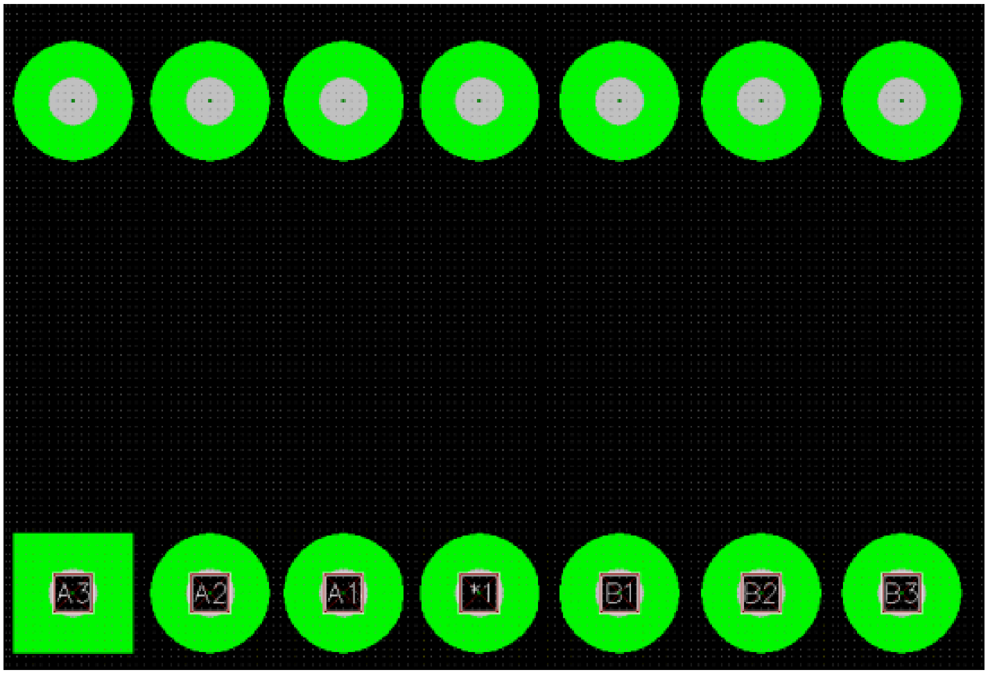
1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
3. Type **25** in the Draw Size box. This specifies a drawing size of 25 mils for the ports, which is large enough to see against a 75-mil pad.
4. Assign this port to the first op-amp by choosing "A" in the "Device:" combo.
5. Type **3** in the Port Number box. This assigns the op-amp output to pin 1 on the physical package.
6. Select **Top Metal** from the Layer list. This places the layout objects on the metal layer before connecting to the pad.
7. Type **0**, **0** in the Location boxes. This centers the port on the first pad.
8. Click **OK**.
9. Repeat steps 1-8 to place ports 2 and 1 for device A on top of pads 2 and 3 at locations **100** , **0** and **200**, **0**.

To place the positive power supply port:

1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
3. Type **300**, **0** in the Location boxes.
4. Since this port is not assigned to a schematic object, choose "None" in the Device: combo
5. Type **1** in the Port Number box.

To place ports for the second op-amp:

1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
3. Type **400**, **0** in the Location boxes.
4. The device should be set to "B", since this is the second op-amp in the package.
5. Type **1** in the Port Number box. This corresponds to the non-inverting input.
6. Repeat steps 1-5 to place two more ports for device B at locations **500**, **0** and **600**, **0** . Number the ports **2** and **3**, corresponding to the inverting input and the output.
   The footprint should look like this:



To place ports for the third op-amp:

1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
3. Type **600**, **310** in the Location boxes.
4. The device should be set to "C", and the pin number set to "3".
5. Type **3** in the Port Number box.
6. Repeat steps 1-5 to place two more ports for device C at locations **500**, **310** and **400**, **310**. Number the ports **2** and **1**, corresponding to the inverting input and the output.

To place the negative power supply port:

1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
3. Type **300**, **310** in the Location boxes.
4. The device should be set to "None".
5. Type **2** in the Port Number box.
6. Click **OK**.

To place ports for the fourth op-amp:

1. Click the Port button (☒) on the Layout toolbar.
2. Click in the Footprint editor to open the EM Port Properties window.
   Type **200**, **310** in the Location boxes.

3.
4. The device should be set to "D"
5. Type **1** in the Port Number box.
6. Repeat steps 1-5 to place two more ports for device D at locations **100**, **310** and **0**, **310**. Number the ports **2** and **3**. corresponding to the inverting input and the output.

To draw silk screen box inside the pad perimeter:

1. Click the Line button (—) on the Layout toolbar.
2. Click in the Footprint editor to open the Line Properties window.
3. Select **Top Silk** from the Layer list to place the line on the top silk layer.
4. Type **10** (mils) in the Line Width box.

> ℹ **Note:** To prevent silk screen interference with metal layer objects, all silk is kept at least 10 mils from the nearest metal.

5. Type **-32.5**, **257.5** in the Start boxes.
6. Type **632.5**, **52.5** in the End boxes.
7. Click **OK**.
8. Press the **O** key to create a 90-degree line.

> ℹ **Note:** All of the above steps clear the pads by 10 mils, and extends to the edge of the pads on the open ends of the footprint.

9. Draw another **10** mil line on Top Silk from **-32.5**, **257.5** to **632.5**, **52.5**. Press the **O** key to create a 90-degree line and then press the **F** key to flip the angle.

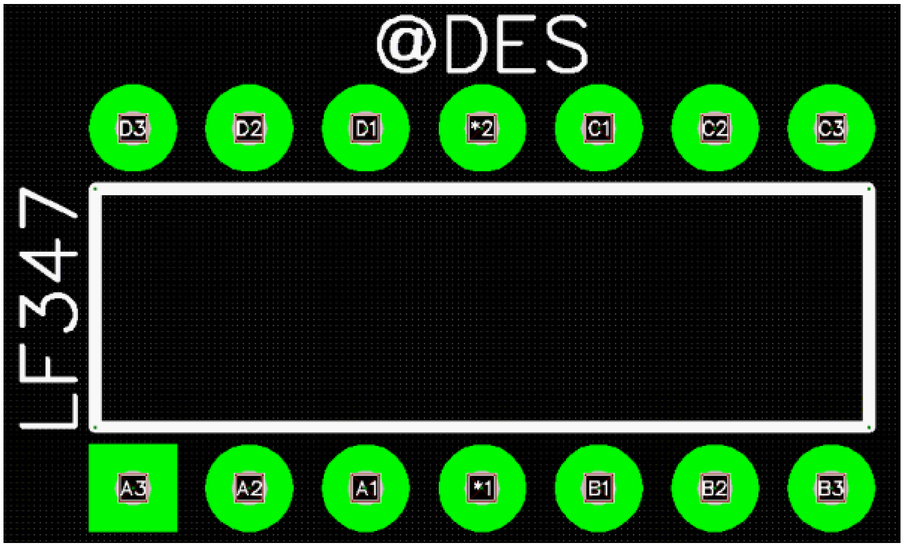
To place the designator text:

1. Click the Text Button (A) on the Layout toolbar.
2. Click in the Footprint editor to open the Text Properties window.
3. Select **Top Silk** from the Layer list.
4. Type **@DES** in the Text box.
5. Click the **Use Default Size** check box.
6. Type **300**, **357**.**5** in the Location boxes.
7. Click the **Center X** button for X-justification.
8. Click the **Bottom** button for Y-justification.
9. Click **OK**.

To label the type of package:

1. Click the Text Button (A) on the Layout toolbar.
2. Click in the Footprint editor to open the Text Properties window.
3. Type **LF347** in the Text box.
4. Type **65** in the Size box. This text does not change with the default settings, and it is always the width of the silk screen box.
5. Type **90** in the Angle box. This rotates the text counter-clockwise 90 degrees.
6. Select **Top Silk** from the Layer list.
7. Type **300**, **357**.**5** in the Location boxes.
8. Set the location to "-47.5,155".
9. Click the **Center X** button for X-justification.
10. Click the **Bottom** button for Y-justification. This places the text at the left edge of the silk screen box with a clearance of 10 mils.
11. Click **OK**.

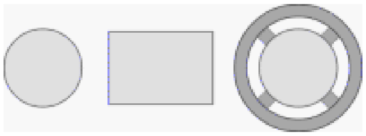
The footprint is now complete. The figure below shows the final footprint:



Note: Be sure to place footprint ports on the same metal layer as the pad.

Using Pads in Layouts

A pad is a flat surface used to make electrical contact. There are three types of pads available: round, square, and wagon wheel. These three pad types are shown in the figure below:



Note: The wagon wheel is often used for thermal relief when connecting to a ground plane, which makes soldering easier.

Placing Pads in Layouts

You should place a pad in a layout wherever solder connections are made.

To place a pad in a layout:

1. Click the Pad button (●) on the Layout toolbar.
2. Click in the layout where you want to place the pad. The Pad Properties window appears.
3. Make the changes you want.
4. Click **OK**.

Marking Pads as Grounded Objects

When pouring a ground plane polygon, the keep away applies to pads as well as to other objects. If a pad is marked as User Ground, the ground plane touches the pad (and any lines and arcs connected to it) instead of avoiding it.

To mark a pad as a grounded object:

1. Double-click the pad to open its properties window.
2. Click the **User Ground** check box.
3. Click **OK**.

Disabling Mask Generation for a Pad

The solder mask layer for a pad is automatically generated. If the pad's layer is marked as mirrored in the Layer table, the mask goes on the next mask layer below the pad's layer. Otherwise, the mask goes on the next mask layer above the pad's layer. If a mask is not needed for a particular pad, you can turn it off for that pad.

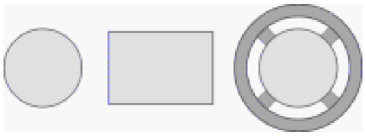
To disable mask generation for a pad:

1. Double-click the pad to open its properties window.
2. Click the **Don't Create Mask** check box.
3. Click **OK**.

Using Viaholes in Layouts

Viaholes are drilled, plated, through holes used to connect traces on different layers. Some viaholes are blind or buried, meaning they are not drilled through all layers. Viaholes automatically place a pad (round, square, or wagon wheel) on each metal layer between the start (top) and end (bottom) of the viahole, including pads on the start and end layers. The drill hole is always in the center of the pad. Electrically, viaholes connect every layer between the start and end layers.

Viaholes automatically generate solder mask for the two top and bottom pads. The shape of the mask that is placed is identical to shapes placed for pads and is shown in the figure below.



If a mask is not needed for a particular viahole, you can turn it off.

Placing Viaholes in Layouts

Most viaholes are drilled through all layers. Blind or buried viaholes are not drilled through the entire board. These viaholes are useful in multilayer boards to connect some layers while leaving other layers unaffected. However, the production of boards with blind and buried viaholes is often more expensive.

To place a viahole in a layout:

1. Select the Viahole button (●) on the Layout toolbar.
2. Click in the layout where you want to place the viahole. The Viahole Properties window appears.
3. Make the changes you want.
4. Click **OK**.

To place a blind or buried viahole in a layout:

1. Select the Viahole button (●) on the Layout toolbar.
2. Click in the layout to place the viahole. The Viahole Properties window appears.
3. Clear the **Use Default Layers** check box if it is selected. This lets you change the start and end layers.
4. Select the start and end layers from the Start and End Layer lists.
5. Make any other changes you want.
6. Click **OK**.

Marking Viaholes as Grounded Objects

When pouring a ground plane polygon, the keep away applies to viahole pads as well as to other objects. If a viahole is marked as User Ground, then the ground plane touches the viahole pads (and any lines and arcs connected to them) instead of avoiding them.
To mark a viahole as a grounded object:

1. Double-click the viahole to open its properties window.
2. Click the **User Ground** check box.
3. Click **OK**.

Disabling Mask Generation for a Viahole

The solder mask layer for a viahole is automatically generated. If the viahole's layer is marked as mirrored in the Layer table, the mask goes on the next mask layer below the viahole's layer. Otherwise, the mask goes on the next mask layer above the viahole's layer. If a mask is not needed for a particular viahole, you can turn it off for that viahole.
To disable mask generation for a particular viahole:

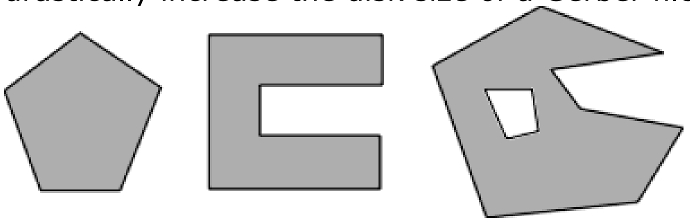
1. Double-click the viahole to open its properties window.
2. Click the **Don't Create Mask** check box.
3. Click **OK**.

Adding Polygons, Pours, and Ground Planes

Polygons are filled areas on the layout. You can use polygons on any layer, create one of any shape, and even have ones with cutouts (holes). A pour is a polygon that is poured around other objects on the same layer. Pouring a polygon causes it to keep away from other objects by a specified distance. Pours are especially useful for constructing ground planes, which can touch objects marked as User Ground while keeping away from other objects.

Constructing Polygons

Polygons are often used for landing pads with unusual shapes. Use polygons only when necessary if you plan to create a Gerber file from the layout, because polygons can drastically increase the disk size of a Gerber file. Some typical polygons are shown below:



To construct a polygon:

1. Click the Polygon button ( ) on the Layout toolbar.
2. Click the first vertex (corner) of the polygon.
3. Click the next vertex.

   Note: If the last point placed is incorrect, press the Backspace key to remove it.

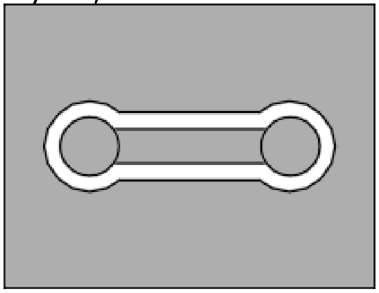
4. Continue clicking all vertices.
5. Double-click the last vertex (or click the first vertex again) to complete construction of the polygon.

To add a cutout (hole) to a polygon:

1. Select the polygon you want to add a cutout.
2. Click the Cutout button (✂) on the Layout toolbar.
3. Click the first vertex (corner) of the cutout. Make sure that all vertices and lines of the cutout are within the original polygon and not overlapping any other cutouts in that polygon.
4. Click the next vertex.
5. Continue clicking all vertices.
6. Double-click the last vertex (or click the first vertex again) to complete the cutout.

Pouring Polygons

The figure below shows a polygon that has been poured around two pads and a line. The keep away distance is the distance that the pour stays from the line and pads and is the width of the white space in the figure. Pours are often used for coplanar ground planes. You should only use pours when necessary if you want to create a Gerber file from the layout, because the use of pours can drastically increase the disk size of a Gerber file.



To pour a polygon:

1. Select the polygon you want to pour.
2. Click the Pour Polygon ( ) button on the Layout toolbar. The Pour Properties window appears.
3. Type the keep away distance in the Keep Away box. The units for keep away and resolution are the current units as specified on the General tab of the Layout Properties window.
4. Type the resolution in the Tolerance box.
5. Type the number of segments in the # Segments box.
6. Click **OK** to begin pouring the polygon. A window appears showing the status of the pour.

Using Ground Plane Pours

Ground plane pours can touch objects that are marked as User Ground and keep away from other objects. Lines and arcs that touch grounded viaholes and pads are also considered grounded. This is similar to the intelligence used to resolve rubber bands.

To pour an existing polygon as a ground plane:

1. Select the polygon you want to pour.
2. Select the Pour Polygon ( ) button on the Layout toolbar. The Pour Properties window appears.
3. Type the keep away distance in the Keep Away box. The units for keep away and resolution are the current units as specified on the General tab of the Layout Properties window.
4. Type the resolution in the Tolerance box.
5. Type the number of segments in the # Segments box.
6. Click the **Ground Plane** check box.

7. Click **OK** to begin pouring the polygon. A box appears showing the status of the pour.

Importing and Exporting Layout Files

Genesys lets you import or export layouts using many file types. Importing a layout lets you use layouts created in other programs. Exporting a layout using one of the supported file types lets you use a Genesys layout in other programs and ensures it is understood by a boarding manufacturer.

Importing Layout Files

You can import a layout using any of the following file types:

- 6.x Model Library
- DXF File
- Excellon (Gerber) Drill List
- GDSII File
- Gerber File
- XML File

To import a layout file:

1. Click **File** on the Genesys menu and select a file type from the Import menu.
2. Follow the instructions in the windows that appear.

Exporting Layout Files

You can export a layout using any of the following file types:

- ASCII Drill List
- Bill of Materials
- DXF File
- Excellon (Gerber) Drill List
- GDSII File
- Gerber File
- HPGL File
- Part Placement List
- XML File

To export a layout file:

1. Open (or select) the layout window to make the **Layout** the active window.
2. Click **File** on the Genesys menu and select a file type from the Export menu.
3. Follow the instructions in the windows that appear.

Using Gerber Files

A Gerber file lets you export layouts using a commonly used format. This lets you use layouts in other programs or export them to board manufactures in a standard format they understand. A circuit board can have many layers; therefore, a Gerber file is created for each layer.

Importing Gerber Files

You can import Gerber files for use in generating or modifying layouts.
To import a Gerber file:

1. Click File on the Genesysmenu and select Gerber File from the Import menu.
2. Click the file you want to import.
3. Click Open.
4. Specify options for importing the file.
5. Click OK.

Exporting Gerber Files

You can write a Gerber file for the current layout.
To export a Gerber file:

1. Click File on the Genesysmenu and select Gerber File from the Export menu.
2. Specify options for exporting the file.
3. Click OK.
4. Click the name of the file.
5. Click Save.

Using Custom Apertures

Custom apertures are apertures created by a layout specifically for each Gerber file. The Aperture list contains a list of user apertures defined by users for specific needs. Generally, you should use user apertures only if your company has a standard set of apertures to which Gerber exports must conform. Otherwise, you should use custom apertures.
A layout has a built-in optimization routine that selects the best list of apertures for efficient polygon fills and flashes. These custom apertures result in the smallest possible Gerber files, whereas a user list can give very inefficient, incorrect, and large files.

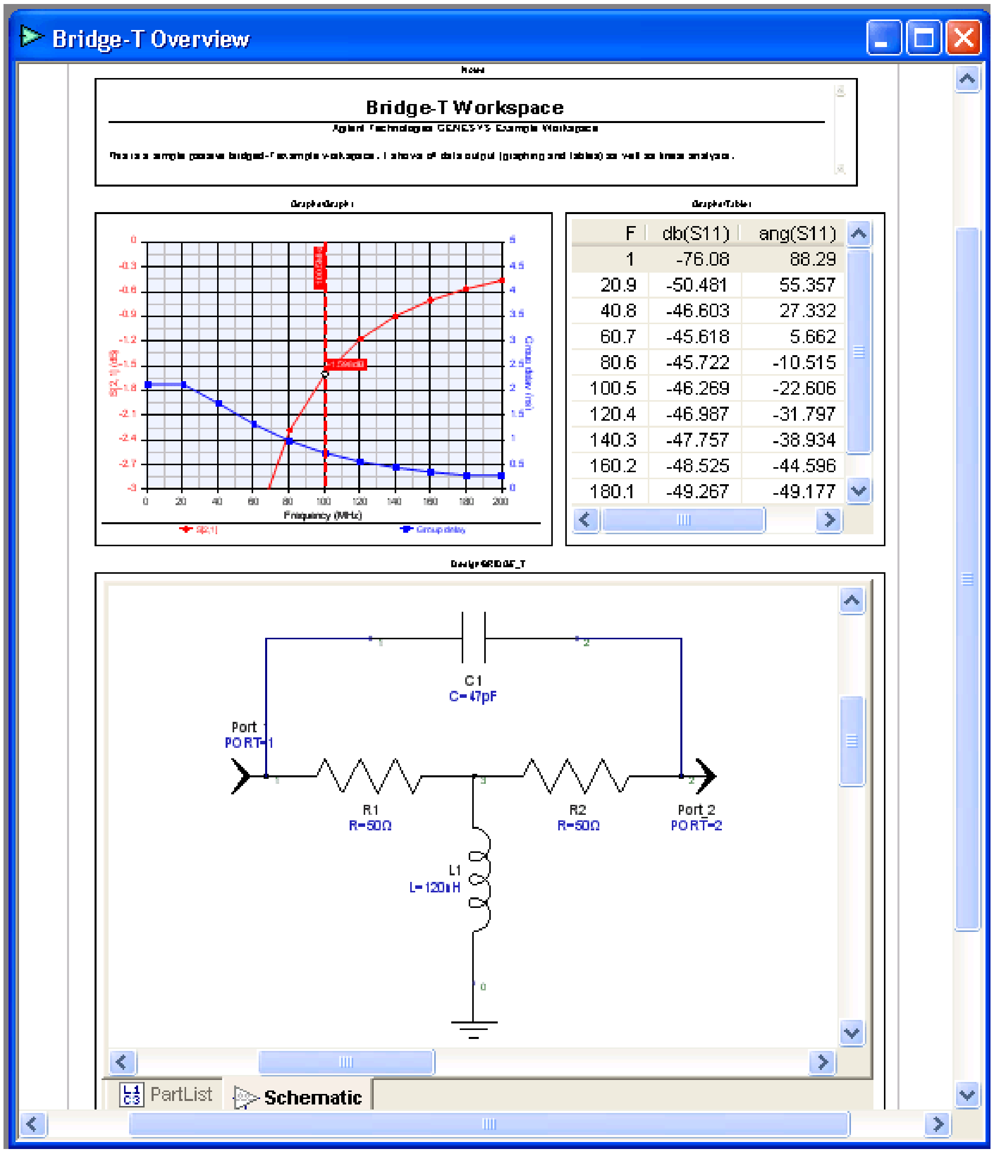
Editing the Aperture List

Use the Editing Aperture List window to customize apertures in a Gerber file.
To edit an aperture list:

1. Click File on the Genesysmenu and select Gerber File from the Import menu.
2. Click the Options tab.
3. Clear the Generate Custom Apertures check box.
4. Click the Edit DEFAULT.APL button.
5. Make the changes you want.
6. Click OK.

LiveReports

A LiveReport is a *living* page that can contain live views of various kinds of Genesys objects. You can mix Graphs, Designs, Equations, Notes, Tables, and Datasets all in a single printable and viewable page. Below is an example of a LiveReport page from the simple Bridge-T Example.



Bridge-T Overview

Notes

Bridge-T Workspace

Agilent Technologies GENESYS Example Workspace

This is a simple passive bridged-T example workspace. It shows off data output (graphing and tables) as well as linear analysis.

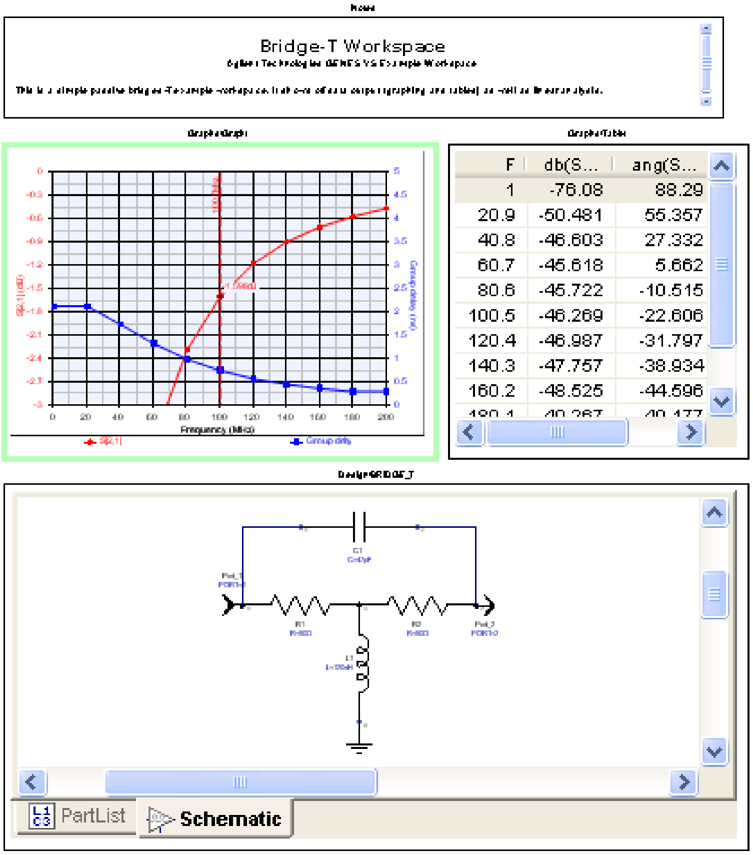
Graph=Graph1

Graph=Table1

F	db(S11)	ang(S11)
1	-76.08	88.29
20.9	-50.481	55.357
40.8	-46.603	27.332
60.7	-45.618	5.662
80.6	-45.722	-10.515
100.5	-46.269	-22.606
120.4	-46.987	-31.797
140.3	-47.757	-38.934
160.2	-48.525	-44.596
180.1	-49.267	-49.177

Design=BRIDGE_T

It's a *Live* Report because you can click in any of the windows on the page and work exactly as you would work in single windows in Genesys. The border turns green when a window is active, as seen below.



Notes

Bridge-T Workspace

Agilent Technologies GENESYS Example Workspace

This is a simple passive bridged-T example workspace that shows off output graphing and tables as well as linear analysis.

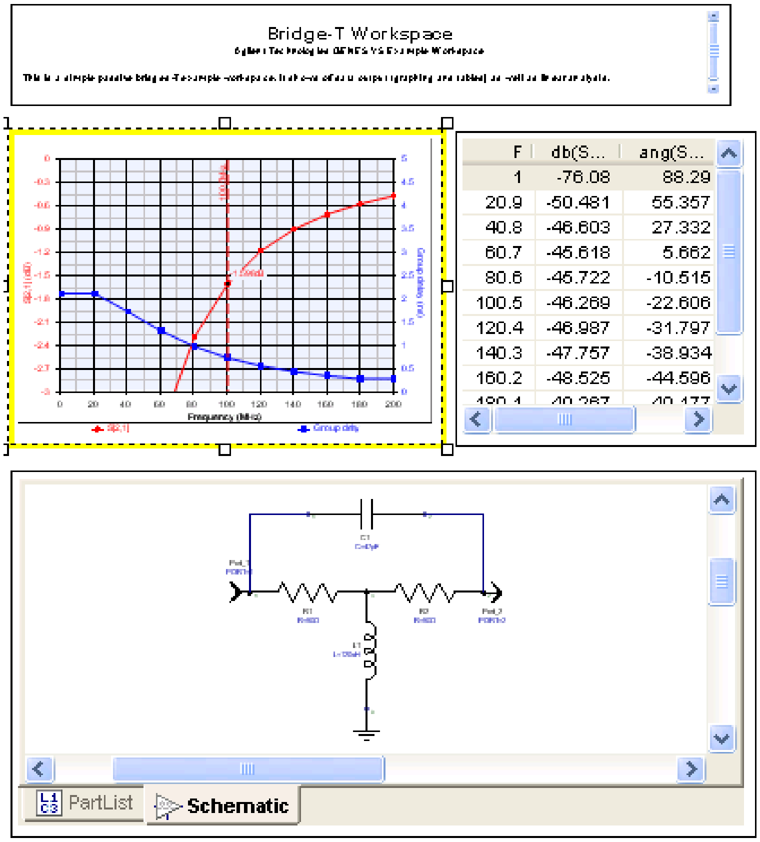
Graphs\Graph

Graphs\Table

F	db(S...	ang(S...
1	-76.08	88.29
20.9	-50.481	55.357
40.8	-46.603	27.332
60.7	-45.618	5.662
80.6	-45.722	-10.515
100.5	-46.269	-22.606
120.4	-46.987	-31.797
140.3	-47.757	-38.934
160.2	-48.525	-44.596
180.1	-49.267	-49.177

Design\BRIDGE_T

When you want to move or resize a window within a LiveReport click the black border (box) outside the window and it will turn yellow and gain handles you can drag/move.



Bridge-T Workspace

Agilent Technologies GENESYS Example Workspace

This is a simple passive bridged-T example workspace that shows off output graphing and tables as well as linear analysis.

F	db(S...	ang(S...
1	-76.08	88.29
20.9	-50.481	55.357
40.8	-46.603	27.332
60.7	-45.618	5.662
80.6	-45.722	-10.515
100.5	-46.269	-22.606
120.4	-46.987	-31.797
140.3	-47.757	-38.934
160.2	-48.525	-44.596
180.1	-49.267	-49.177

To use the LiveReport rather than one of the windows, click outside all of the windows and you will see the LiveReport toolbar and the LiveReport menu. No windows will be green or yellow.

Contents

Creating a LiveReport

To manually create a LiveReport:

1. Click the New Item button () on the Workspace Tree toolbar, select *add* **LiveReport...**.
2. If desired, change the LiveReport name, layout, or other properties.
3. Click **OK** to create the LiveReport or click **Cancel** to not create the new LiveReport.

Supported LiveReport Object Types

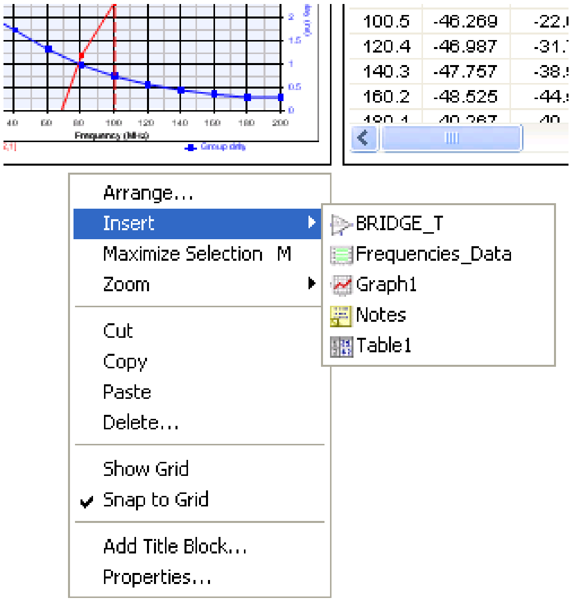
LiveReports supports most of the standard Genesys object types.

Object Type	**Supported?**	**Limitations**
Graph	yes	A Graph has a single aspect ratio. If you have an open graph and a graph in report only the only the graph that is currently selected will own the aspect ratio.
Design	yes (partial)	Not supported - Parameters, PartList, and SubstrateSet
Notes	yes	
Datasets	yes	
Equations	yes	
Scripts	yes	
Tables	yes	
Analyses	no	Linear, Transient, ... have no view
Evaluations (sweeps)	no	Evaluations have no view
Syntheses	no	Syntheses have no view
Substrates	no	Substrates have no view

Adding a View Window to a LiveReport

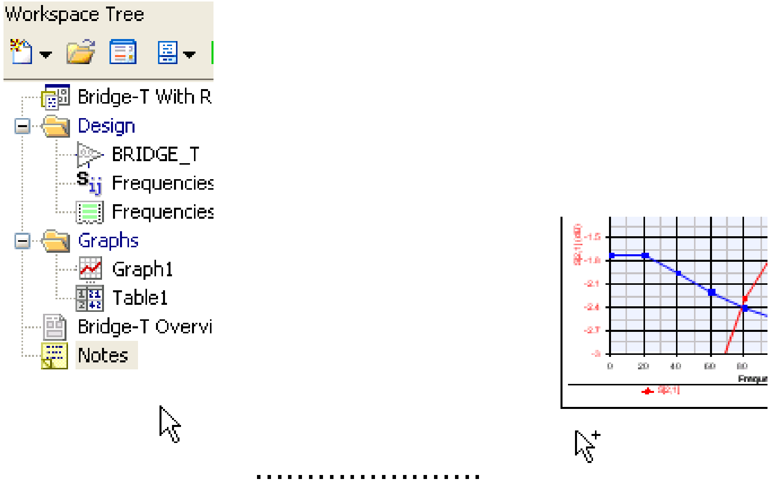
There are two ways to put objects (windows) on a LiveReport.

- Right-click the page and select Insert then select one of the objects listed to insert a window with that object.



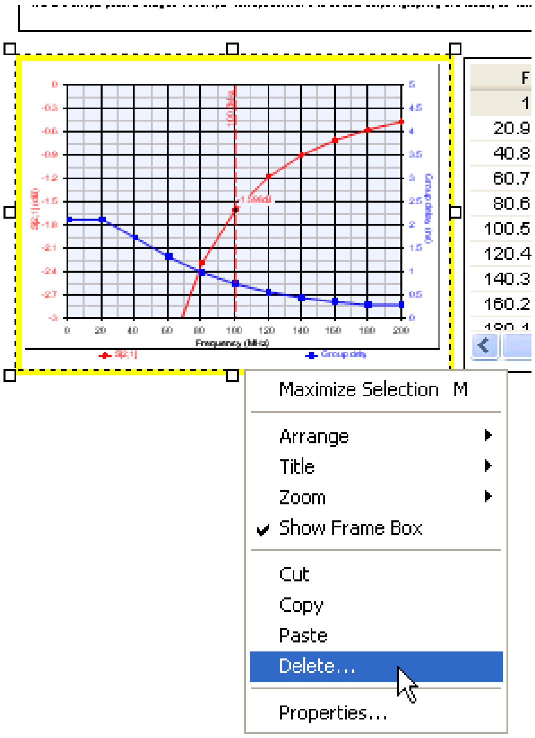
100.5	-46.269	-22.
120.4	-46.987	-31.
140.3	-47.757	-38.
160.2	-48.525	-44.

- Drag-drop an object from the workspace tree into the page using the mouse left button.



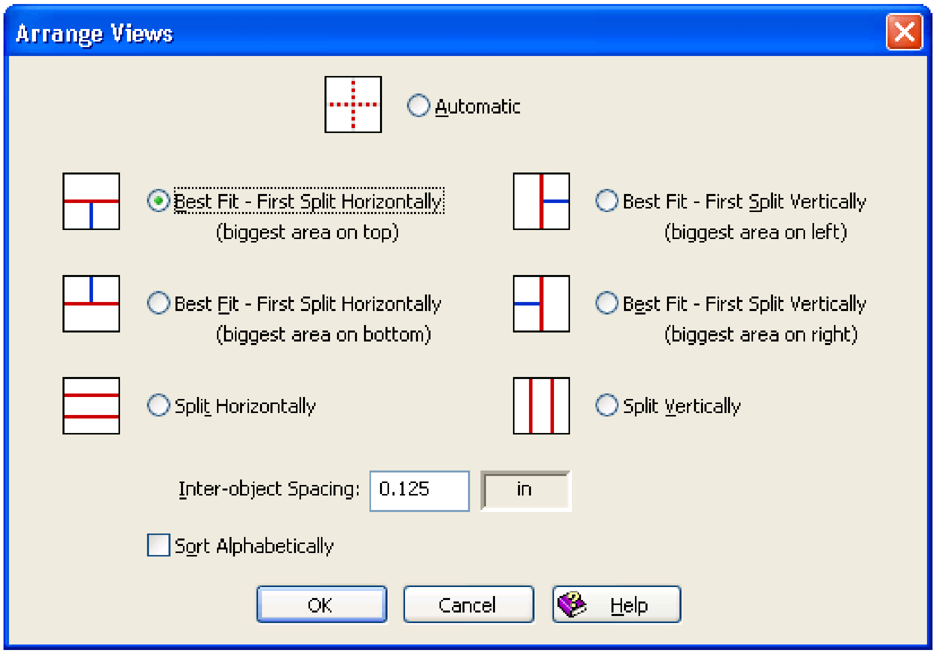
Removing a Window from a LiveReport

Select the surrounding rectangle (click it or select multiple with the select tool and draw a box) and then either click the Del key or right-click the mouse and select Delete... from the menu.



Arranging Views

Use the LiveReport Arrange Views dialog box to arrange the sub-objects of a LiveReport.


Arrange Views

○ Automatic

● Best Fit - First Split Horizontally (biggest area on top)

○ Best Fit - First Split Vertically (biggest area on left)

○ Best Fit - First Split Horizontally (biggest area on bottom)

○ Best Fit - First Split Vertically (biggest area on right)

○ Split Horizontally

○ Split Vertically

Inter-object Spacing: 0.125 in

□ Sort Alphabetically

Automatic - In general, automatic arrangement is quick, easy, and does a reasonable job of laying out the view windows.

If a window is not placed in the desired location when OK is clicked, simply drag the window into place and swap it with another, then re-do the Arrange Views.

If the LiveReport page is not divided in the desired fashion, use one of the Best Fit or Split options.

Best Fit - The best-fit options arrange sub-objects in a tiled arrangement, similar to the way Windows arranges Tiled views. The images show the order of the major and minor page splits.

Split Horizontally / Vertically - Arrange sub-objects so they are ordered in a linear fashion and are all the same size.

Inter-object Spacing - The distance (gap) between the arranged views. The units can be set in Page Properties.

Sort Alphabetically - This rarely-used option sorts the views by name, instead of the usual geometric positioning based on current pane positions. This options is mostly used to display libraries of symbols or parts.

> ℹ The images to the left of the radio buttons may be double-clicked to quickly select an arrangement and close the dialog box.

LiveReport Properties

There are several tab pages that you can use to change the properties of a LiveReport:

- Page Properties
- Margin Properties
- Header and Footer Properties

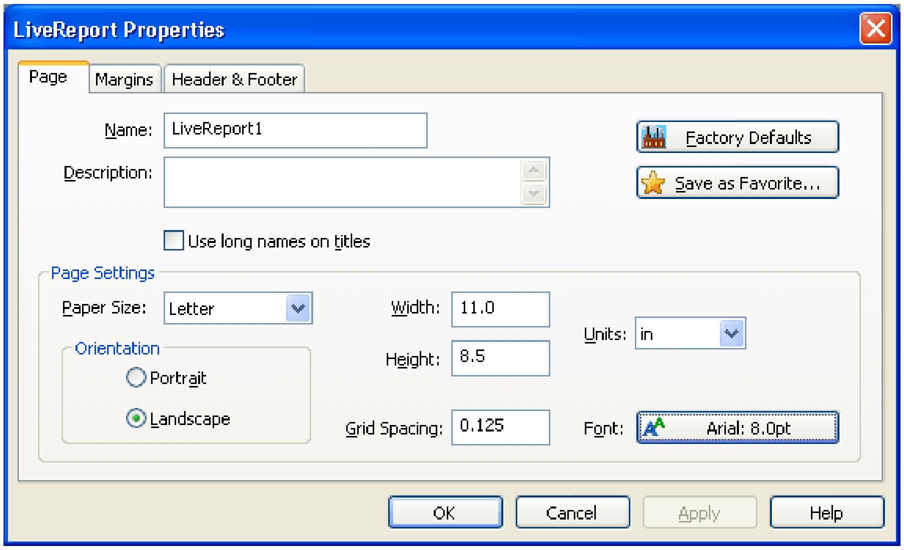
To change the properties of a LiveReport:

1. Double-click the report or click **LiveReport** on the Genesys menu and select **Properties**.
2. Click the desired tab.
3. Make the changes you want.
4. Click **OK**.

> ℹ When double-clicking the LiveReport, Genesys uses the mouse cursor location to pick an appropriate tab. Double-click the upper or lower page area to initially display the Header & Footer page; double-click the side margins to initially display the Margins page; anywhere else displays the Page settings tab page.

Page Properties

Use the LiveReport Page Properties tab page to change the general properties of a LiveReport.


LiveReport Properties

Name: LiveReport1

Description:

□ Use long names on titles

Page Settings

Paper Size: Letter

Width: 11.0

Height: 8.5

Units: in

Orientation

○ Portrait

● Landscape

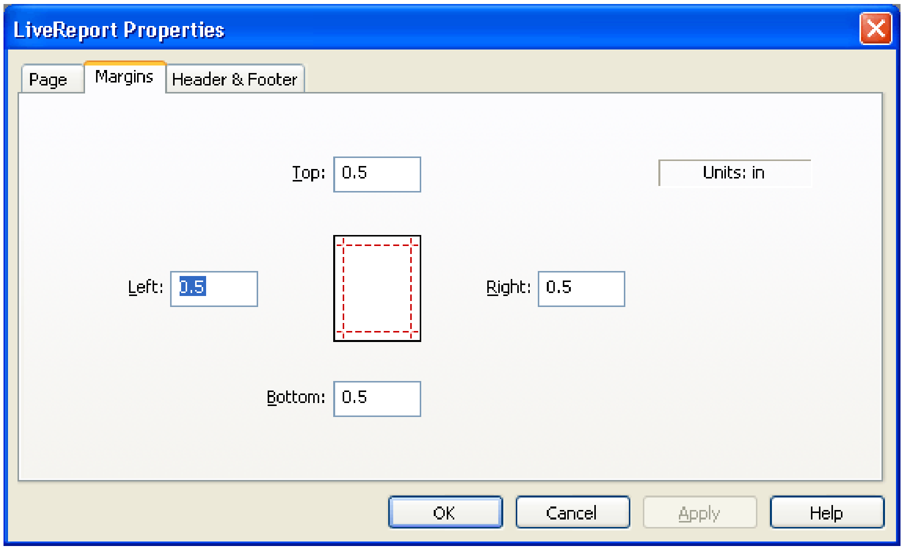
Grid Spacing: 0.125

Font: Arial: 8.0pt

1. **Name** - The name of the LiveReport.
2. **Description** - The LiveReport description (optional).
3. **Use long names on titles** - When checked, the sub-object view windows will show the full workspace pathname in their title.
4. **Paper Size** - Use this combo-box to set your page size.
5. **Orientation** - Sets the page to portrait (tall) or landscape (wide) mode.
6. **Width & Height** - The size of the paper (in current units).
7. **Grid Spacing** - The distance between grid dots.
8. **Units** - The units used by LiveReport for all of its settings, including margins and arrangement spacing.
9. **Font** - The page font, which is used for sub-object titles.

Margin Properties

Use the LiveReport Margins Properties page to change the margins of a LiveReport.


LiveReport Properties

Top: 0.5

Units: in

Left: 0.5

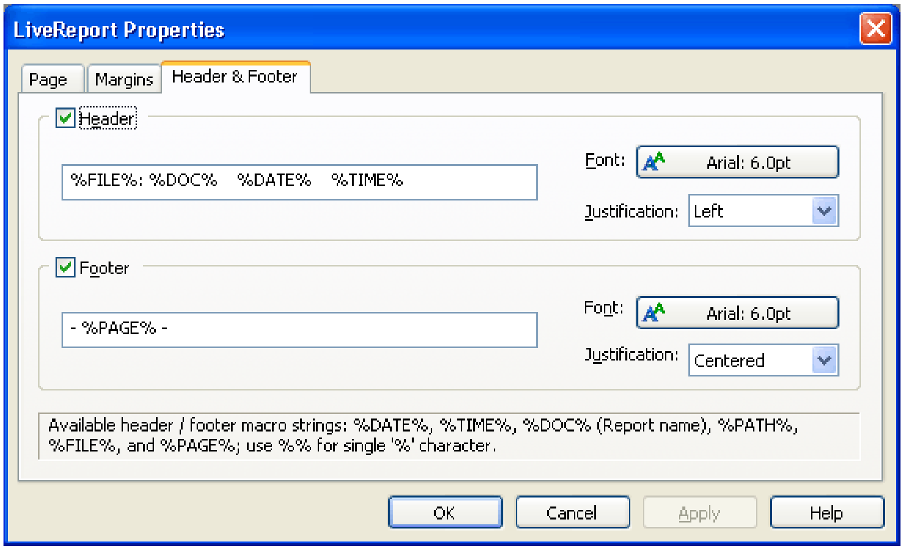
Right: 0.5

Bottom: 0.5

1. **Top, Left, Right, Bottom** - The margin widths to use for the page. The margins are shown by a light-gray, non-printing box; the box can be hidden using the eye toolbar button menu.
2. **Units** - The units used by LiveReport for all of its settings can be set on the Page tab.

Header and Footer Properties

Use the LiveReport Header & Footer Properties page to change the margins of a LiveReport.



☑ Header

%FILE%: %DOC%   %DATE%   %TIME%

Font: Arial: 6.0pt

Justification: Left

☑ Footer

- %PAGE% -

Font: Arial: 6.0pt

Justification: Centered

Available header / footer macro strings: %DATE%, %TIME%, %DOC% (Report name), %PATH%, %FILE%, and %PAGE%; use %% for single '%' character.

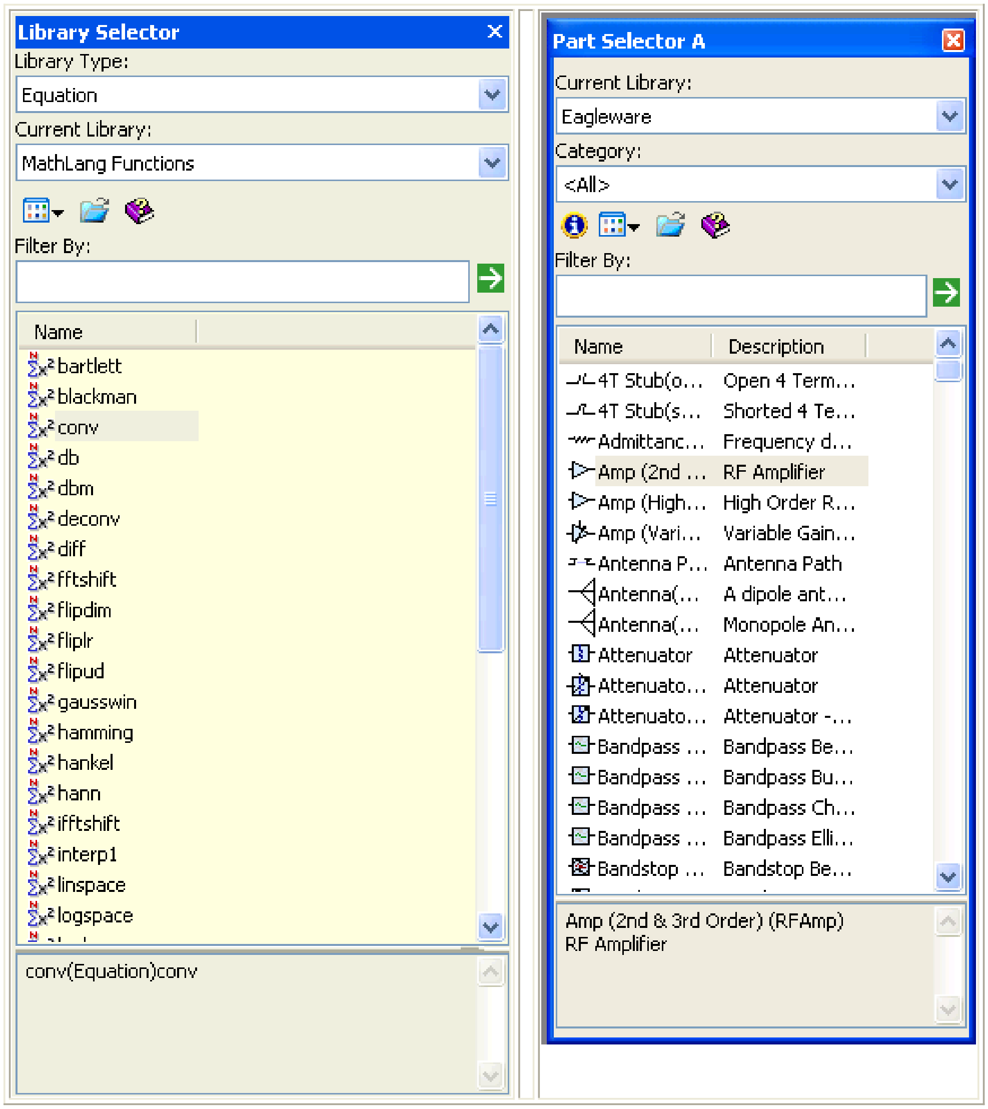
1. **Header** - When checked, the header is enabled. It will print at the top of the LiveReport page. The text is completely customizable; strings such as "Company Confidential" may be used. Also, macro strings like "%DATE%" and "%TIME%" will be converted into the actual date, time, filename, etc.
2. **Font** - Sets the header font.
3. **Justification** - Determines the header horizontal justification (left / centered / right).
4. **Footer** - The footer works just like the header, but is printed at the bottom of the page.

Managing Libraries

Libraries serve as a container for parts, designs, equations, and lots of other Genesys objects. They let you keep all of your objects in one place, which makes it easier for you to organize the contents. Genesys provides a number of libraries for your convenience and allows you to add custom libraries. Libraries that are added will be auto-loaded when Genesys starts.

There are two dialog boxes in Genesys that allow you to interact with Libraries: The Library Selector and the Part Selector. There is also a Library Manager dialog box that allows you to do things such as import and remove libraries.

The two dialog boxes that enable the interaction with objects are the Library Selector and the Part Selector.



Library Selector

Library Type: Equation

Current Library: MathLang Functions

Filter By:

Name
bartlett
blackman
conv
db
dbm
deconv
diff
fftshift
flipdim
fliplr
flipud
gausswin
hamming
hankel
hann
ifftshift
interp1
linspace
logspace

conv(Equation)conv

Part Selector A

Current Library: Eagleware

Category: <All>

Filter By:

Name	Description
4T Stub(o...	Open 4 Term...
4T Stub(s...	Shorted 4 Te...
Admittanc...	Frequency d...
Amp (2nd ...	RF Amplifier
Amp (High...	High Order R...
Amp (Vari...	Variable Gain...
Antenna P...	Antenna Path
Antenna(...	A dipole ant...
Antenna(...	Monopole An...
Attenuator	Attenuator
Attenuato...	Attenuator
Attenuato...	Attenuator -...
Bandpass ...	Bandpass Be...
Bandpass ...	Bandpass Bu...
Bandpass ...	Bandpass Ch...
Bandpass ...	Bandpass Elli...
Bandstop ...	Bandstop Be...

Amp (2nd & 3rd Order) (RFAmp)
RF Amplifier

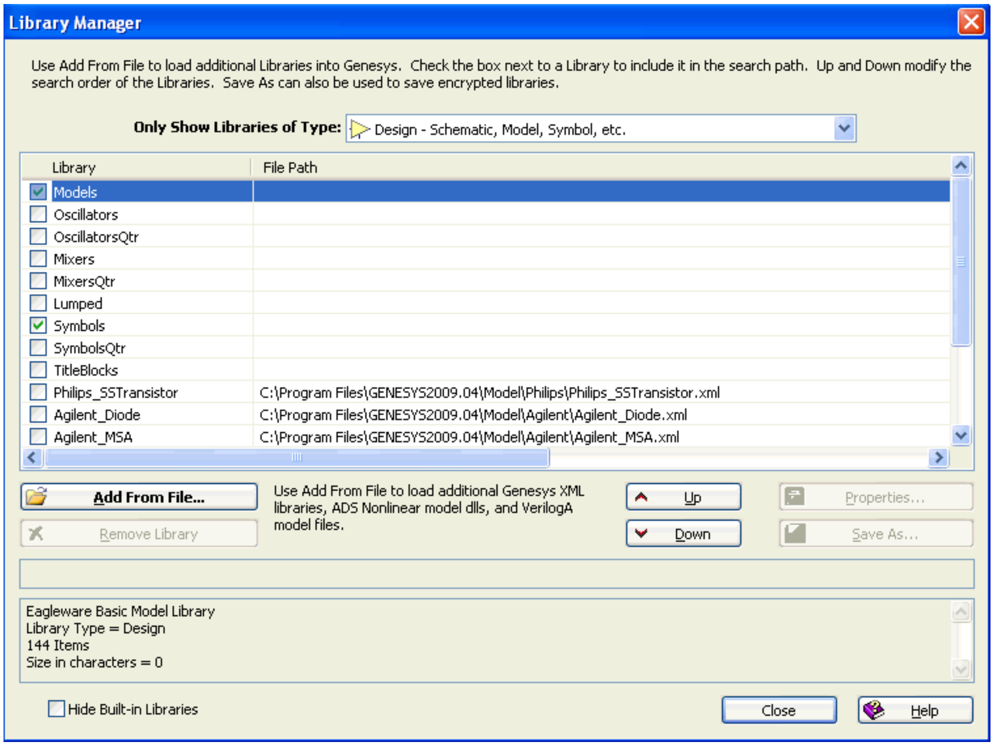
The Library Selector allows interaction with all object types except parts, since the Part Selector is used to interact with Parts. You can think of the Part Selector as a specialized Library Selector where the Library object type is Parts. Both the Library Selector and Part Selector allow you to bring objects into your workspace or view objects that you have put into them from the workspace. A search text box is provided in both selectors to help you find objects in your libraries.

Contents

Using the Library Manager

To open the Library Manager window:

- Click the Library Manager ( ) button on the Part Selector or Library Selector toolbar.
  or
- Select the **Tools** menu and **Library Manager**.



Library Manager

Use Add From File to load additional Libraries into Genesys. Check the box next to a Library to include it in the search path. Up and Down modify the search order of the Libraries. Save As can also be used to save encrypted libraries.

Only Show Libraries of Type: Design - Schematic, Model, Symbol, etc.

Library	File Path
☑ Models	
□ Oscillators	
□ OscillatorsQtr	
□ Mixers	
□ MixersQtr	
□ Lumped	
☑ Symbols	
□ SymbolsQtr	
□ TitleBlocks	
□ Philips_SSTransistor	C:\Program Files\GENESYS2009.04\Model\Philips\Philips_SSTransistor.xml
□ Agilent_Diode	C:\Program Files\GENESYS2009.04\Model\Agilent\Agilent_Diode.xml
□ Agilent_MSA	C:\Program Files\GENESYS2009.04\Model\Agilent\Agilent_MSA.xml

Use Add From File to load additional Genesys XML libraries, ADS Nonlinear model dlls, and VerilogA model files.

Eagleware Basic Model Library
Library Type = Design
144 Items
Size in characters = 0

- **Only Show Libraries of Type** - Selects which type of libraries to view in the main window.
- **Add From File...** - Add a Genesys XML library, ADS Nonlinear model dll, or VerilogA model.
- **Remove Library** - Remove the selected library from the Library Manager.
- **Up** - Move the selected library up one position on the Library list.
- **Down** - Move the selected library down one position on the Library list.
- **Properties...** - View the properties of the selected library.
- **Save As...** - Save the selected library as some other name, or as a encrypted library.
- **Hide Built-in Libraries** - Hides all built in libraries from the library list. Only vendor and custom libraries are displayed.

You can use the Library Manager to:

- Add a Library from a File
- View Libraries of Different Types
- Add Libraries to the Search Path
- Remove a Library

- Edit the Properties of a Library
- Export an Encrypted Library

Add a Library from a File

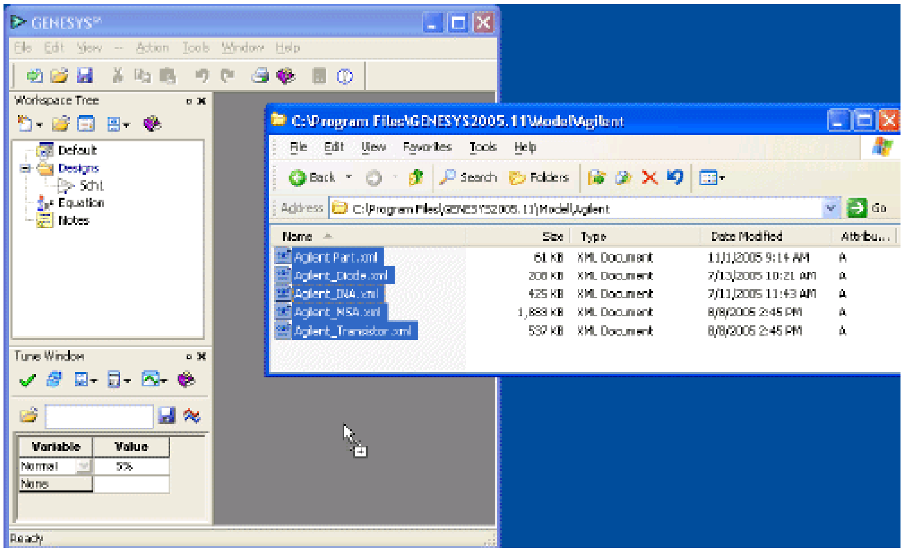
Select the **Add From File...** button to add a Genesys XML library, a ADS Nonlinear Model dll, or a VerilogA model to the Library Manager. Genesys ships with a huge range of prebuilt vendor libraries (parts and models) located in the Model folder of the Genesys directory. The Model folder also contains all of the VerilogA and ADS Nonlinear models that ship with Genesys.

Adding XML Libraries

1. Click the **Add From File...** button.
2. Browse to the folder with the XML library you want to use.
3. Select one or more libraries (use Shift+Click, Ctrl+Click, or Ctrl+A to select more than one).
4. Click **Open** to add the library or Libraries to the available Libraries.

Adding XML Libraries via Drag-Drop

- Find the XML libraries you want to add using Windows Explorer. Select all of the libraries and drag then drop them into the Genesys work area.


GENESYS
File Edit View Action Tools Window Help
Workspace Tree
Default
Designs
Schl
Equation
Notes
Tune Window
Variable
Value
Normal
5%
None
Ready
C:\Program Files\GENESYS2005.11\Model\Agilent
File Edit View Favorites Tools Help
Back
Search
Folders
Address
Go

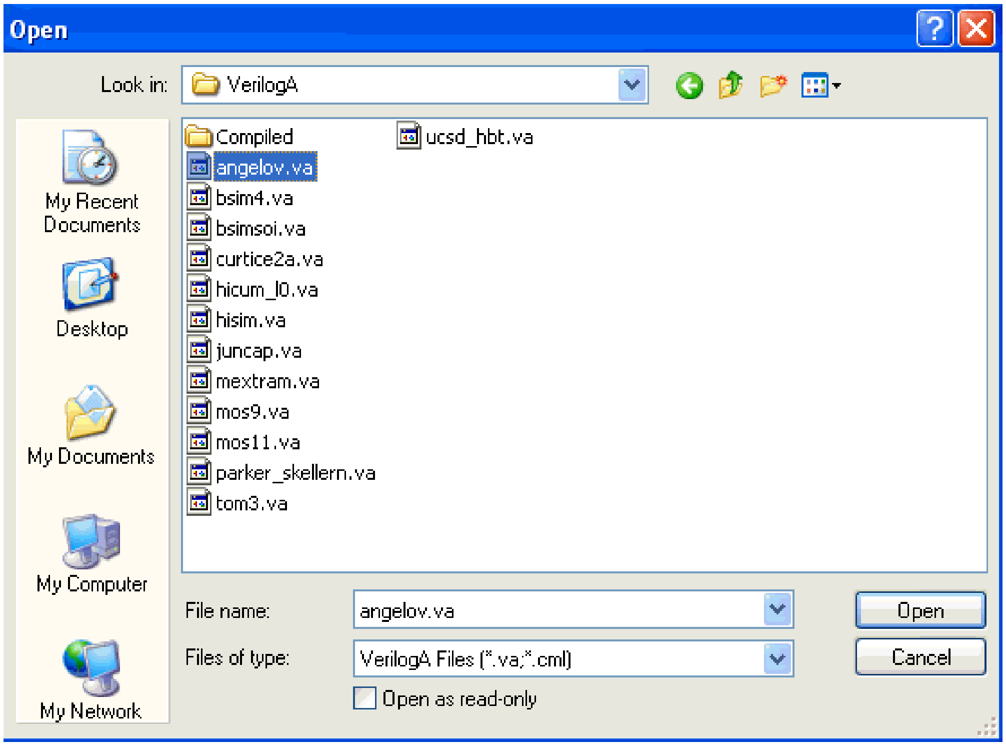
Address: C:\Program Files\GENESYS2005.11\Model\Agilent

Name	Size	Type	Date Modified	Attribu...
Agilent Part.xml	61 KB	XML Document	11/1/2005 9:14 AM	A
Agilent_Diode.xml	208 KB	XML Document	7/13/2005 10:21 AM	A
Agilent_INA.xml	425 KB	XML Document	7/11/2005 11:43 AM	A
Agilent_NSA.xml	1,883 KB	XML Document	8/8/2005 2:45 PM	A
Agilent_Transistor.xml	537 KB	XML Document	8/8/2005 2:45 PM	A

In the image above we add 5 new libraries (one part library and four model libraries) to Genesys.

Adding VerilogA Models

1. In the Library Manager, select the **Add From File...** button.
2. Set the Files of Type field to **VerilogA Model Files (*.va, *.cml)**.
3. Browse for and select the VerilogA model you wish to add.
4. Click **Open**.



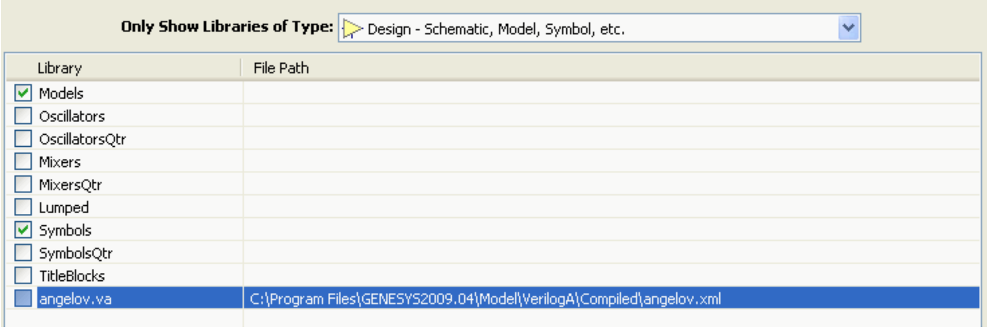
Look in: VerilogA

File name: angelov.va

Files of type: VerilogA Files (*.va;*.cml)

□ Open as read-only

The VerilogA file is added to the Library Manager as a Library as shown below.

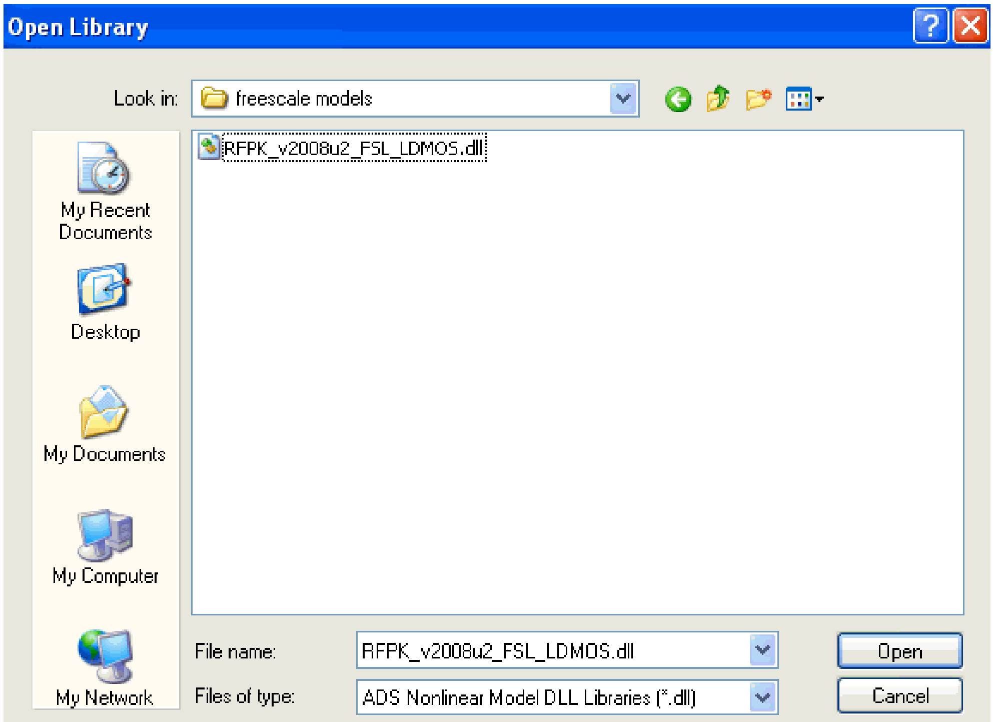


Only Show Libraries of Type: Design - Schematic, Model, Symbol, etc.

Library	File Path
☑ Models	
□ Oscillators	
□ OscillatorsQtr	
□ Mixers	
□ MixersQtr	
□ Lumped	
☑ Symbols	
□ SymbolsQtr	
□ TitleBlocks	
□ angelov.va	C:\Program Files\GENESYS2009.04\Model\VerilogA\Compiled\angelov.xml

Note: Each added VerilogA .va/.cml file will automatically create a matching xml library for use within Genesys.

Adding ADS Nonlinear Models



Open Library

Look in: freescale models

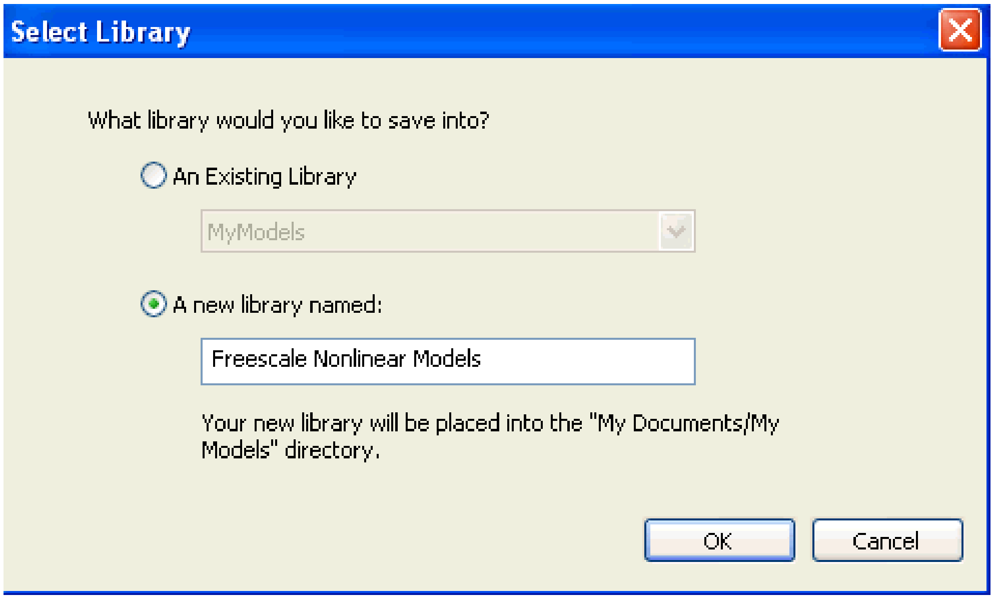
RFPK_v2008u2_FSL_LDMOS.dll

File name: RFPK_v2008u2_FSL_LDMOS.dll

Files of type: ADS Nonlinear Model DLL Libraries (*.dll)

1. In the Library Manager, select the **Add From File...** button.
2. Set the Files of Type field to **ADS Nonlinear Model DLL Libraries (*.dll)**.
3. Browse for the and select the ADS Nonlinear model you wish to add.
4. Click **Open**.
5. Choose to save the model into and existing library or to create a new library.
6. Click **OK**

The DLL file must be in the MINT format. Only MINT model DLLs can be added this way.



Select Library

What library would you like to save into?

○ An Existing Library

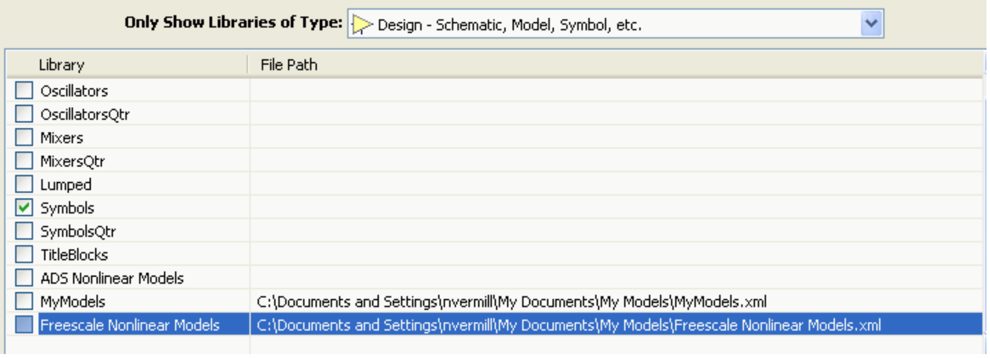
MyModels

● A new library named:

Freescale Nonlinear Models

Your new library will be placed into the "My Documents/My Models" directory.

The model is added to the library of your choice. Multiple models can be added to the same library.

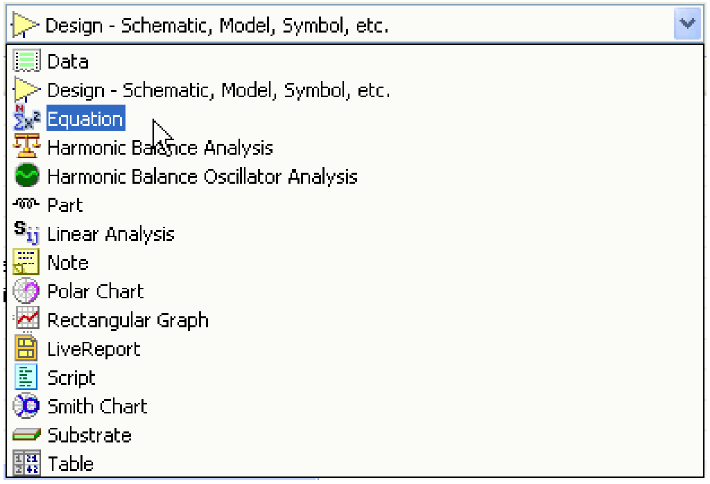


Only Show Libraries of Type: Design - Schematic, Model, Symbol, etc.

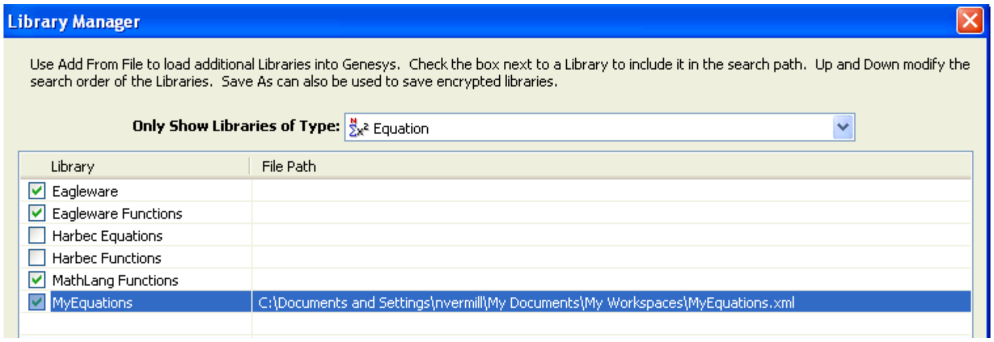
Library	File Path
□ Oscillators	
□ OscillatorsQtr	
□ Mixers	
□ MixersQtr	
□ Lumped	
☑ Symbols	
□ SymbolsQtr	
□ TitleBlocks	
□ ADS Nonlinear Models	
□ MyModels	C:\Documents and Settings\nvermill\My Documents\My Models\MyModels.xml
□ Freescale Nonlinear Models	C:\Documents and Settings\nvermill\My Documents\My Models\Freescale Nonlinear Models.xml

View Libraries of Different Types

The Library Manager lists all of the libraries that are currently loaded into Genesys. Use the dropbox **Only Show Libraries of Type** to select which type of libraries to view in the Library Manager.


Design - Schematic, Model, Symbol, etc.
Data
Design - Schematic, Model, Symbol, etc.
Equation
Harmonic Balance Analysis
Harmonic Balance Oscillator Analysis
Part
Linear Analysis
Note
Polar Chart
Rectangular Graph
LiveReport
Script
Smith Chart
Substrate
Table

The selection Design - Schematic, Model, Symbol, etc. shows all libraries that contain schematics, models, or symbols. Notice that if you change the Type to **Equation** only libraries of equations are shown in the Library Manager.



Use Add From File to load additional Libraries into Genesys. Check the box next to a Library to include it in the search path. Up and Down modify the search order of the Libraries. Save As can also be used to save encrypted libraries.

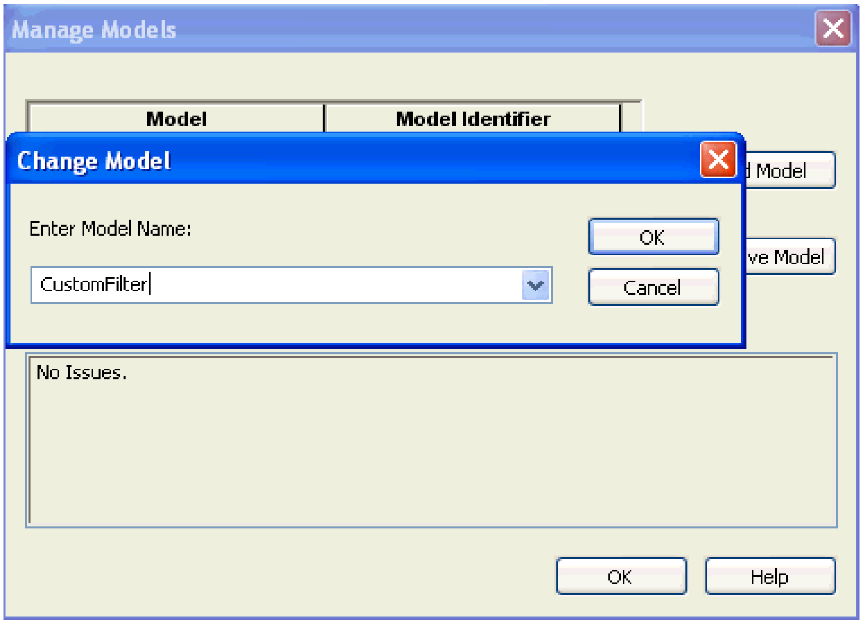
Only Show Libraries of Type: Equation

Library	File Path
☑ Eagleware	
☑ Eagleware Functions	
□ Harbec Equations	
□ Harbec Functions	
☑ MathLang Functions	
☑ MyEquations	C:\Documents and Settings\nvermill\My Documents\My Workspaces\MyEquations.xml

Adding Libraries to the Search Path

The checkbox next to each library determines whether or not the library is included in the search path. Once a library has been added to the search path, it will always be in the search path unless you manually remove it. Libraries at the top of the list have the highest priority in the search path. Use the **Up** and **Down** buttons to move libraries around in the list.

This feature is useful for libraries of custom models or symbols that you may have. Models and symbols from libraries that are included in the search path can be type in the Change Model or Change Symbol dialogs. For example, a library called MyModels that contains a model called CustomFilter has been added to the Library Manager and added to the search path. The model CustomFilter or any other model in MyModels can now be entered in the Change Model dialog directly.

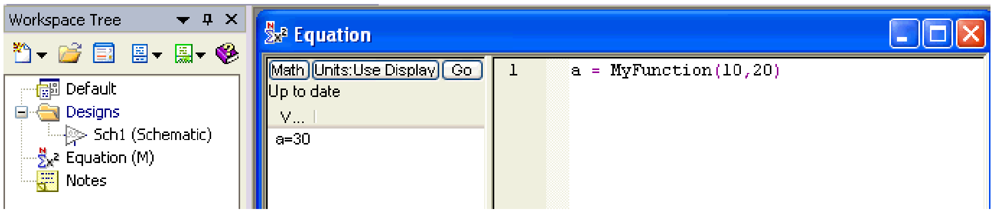


Enter Model Name: CustomFilter

No Issues.

Adding the MyModels library to the search path removes the need to type CustomFilter@MyModels or to add the model from the library to the workspace tree in order to use the model in a part.

Another helpful tip is to add custom Equation libraries to the search path. Functions in an Equation library that has been added to the search path can be called directly from any equation block. For example, a library called "MyEquations" that contains a function called MyFunction is added to the Library Manager and included into the search path. The function MyFunction or any other function in MyEquations can now be called from any equation block.



Up to date

a=30

```
1    a = MyFunction(10,20)
```

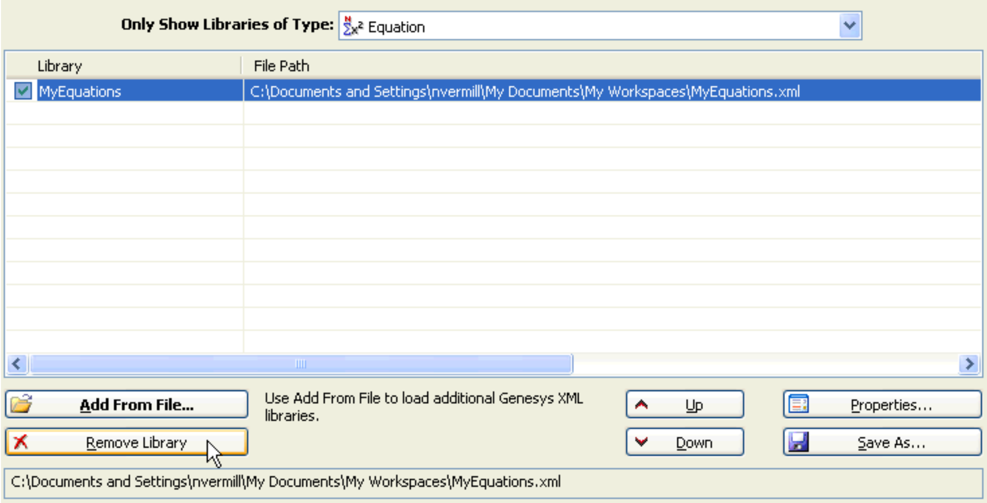
Functions in the MyEquations library do not need to be added to the workspace if MyEquations has been added to the search path.

Removing a Library

When you remove a library, you only remove it from the Library box in the Library Manager. The external library file is not deleted.

You cannot remove the Internal libraries. These libraries are read-only.

To remove a library from the Libraries list:



Only Show Libraries of Type: Equation

Library	File Path
☑ MyEquations	C:\Documents and Settings\nvermill\My Documents\My Workspaces\MyEquations.xml

Use Add From File to load additional Genesys XML libraries.

C:\Documents and Settings\nvermill\My Documents\My Workspaces\MyEquations.xml

1. Click the name of the library in the Library Manager dialog.
2. Click **Remove Library**.

This does not delete the library file. It just assures that the library is not auto-loaded the next time you run Genesys.

Editing Library Properties

You can use the Library Manager to edit the properties of your libraries such as the name or description of the library.

To edit the properties of a library:

1. Click the name of a library in the Library list.
2. Click the **Properties...** button.
3. Make the changes you want, and then click **OK**.

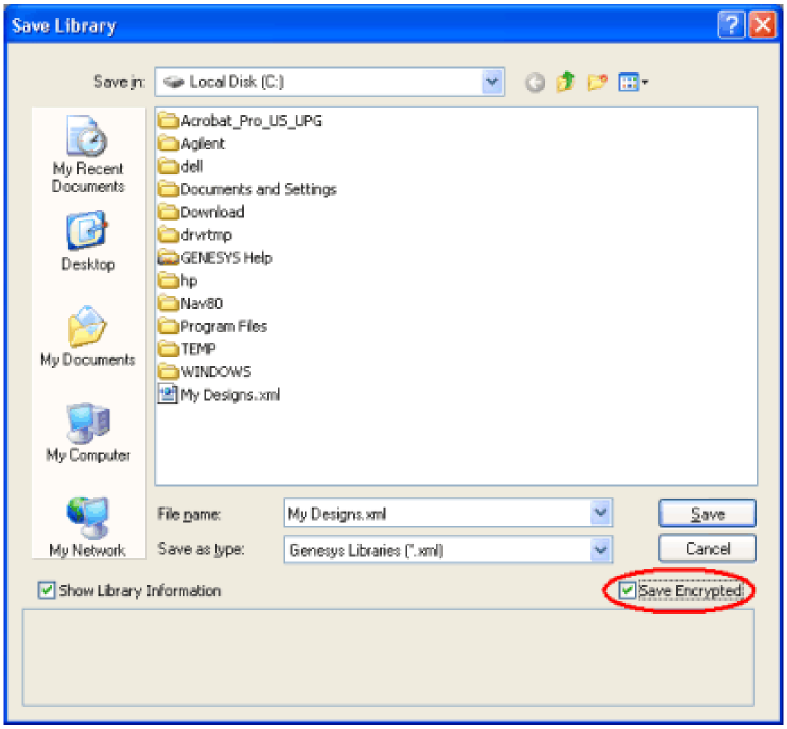
You cannot edit Internal libraries nor can you edit Encrypted Libraries. These libraries are read-only.

Export an Encrypted Library

Genesys supports encrypting and using encrypted libraries. The option to encrypt a library is accessible through the Library Manager, which itself is accessible from the Library Selector (View/Docking Windows/Library Selector) or the the Tools Menu. Once you have created a custom library, you may save it as encrypted.

In the Library Manager, select your custom library and click the **Save As...** button on the right.

In the Save As dialog box, there is a checkbox labeled **Save Encrypted** on the lower right. Check this box as shown here:



Save in: Local Disk (C:)

File name: My Designs.xml

Save as type: Genesys Libraries (*.xml)

☑ Show Library Information

☑ Save Encrypted

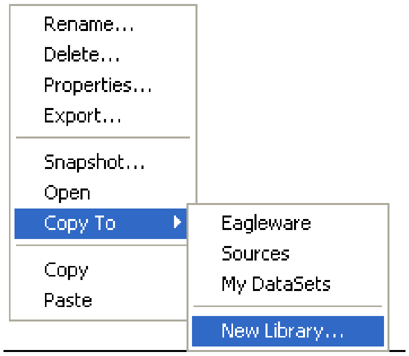
The library will then be saved as encrypted. If the library you are encrypting is a Design Library, you will not be able to view the contents of the designs' part lists or equations.

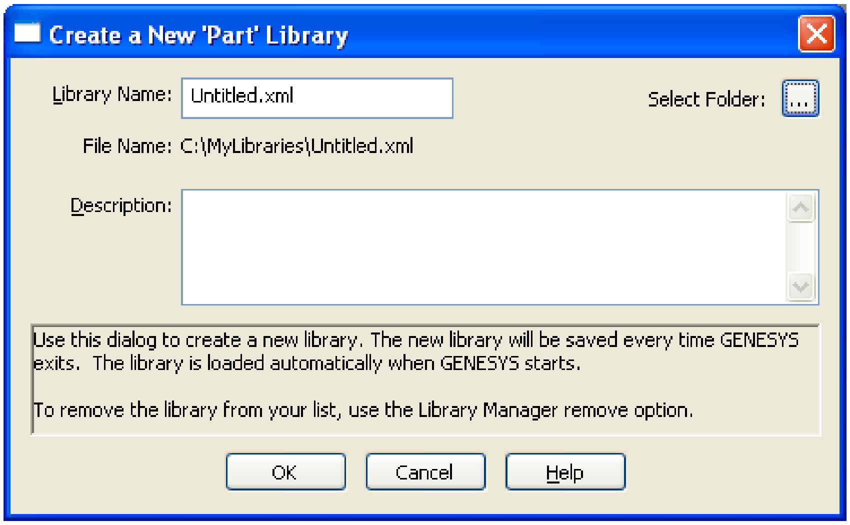
Creating Custom Libraries

A custom library can contain custom parts or designs (models and symbols and circuits), or anything else. Each Library Manager section can only hold custom libraries of its specific type (so you can not use a design library in the part selector, you can not use a Dataset library in the design selector, and so on).

To create a custom library

1. Right click an object in the workspace tree or for a part right click a part in the Part Selector
2. Select New Library... from the Copy To menu





Create a New 'Part' Library

Library Name: Untitled.xml

Select Folder: ...

File Name: C:\MyLibraries\Untitled.xml

Description:

Use this dialog to create a new library. The new library will be saved every time GENESYS exits. The library is loaded automatically when GENESYS starts.

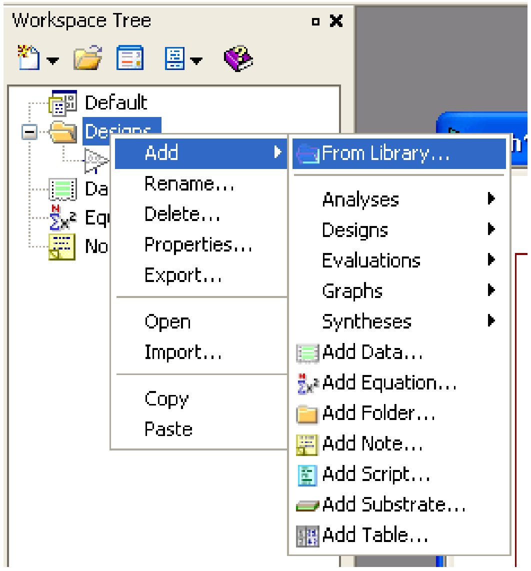
To remove the library from your list, use the Library Manager remove option.

3. Set the library name.
4. If you want, browse for a different path for the new library. We recommend the My Workspaces folder for libraries, though.
5. Click **OK**.

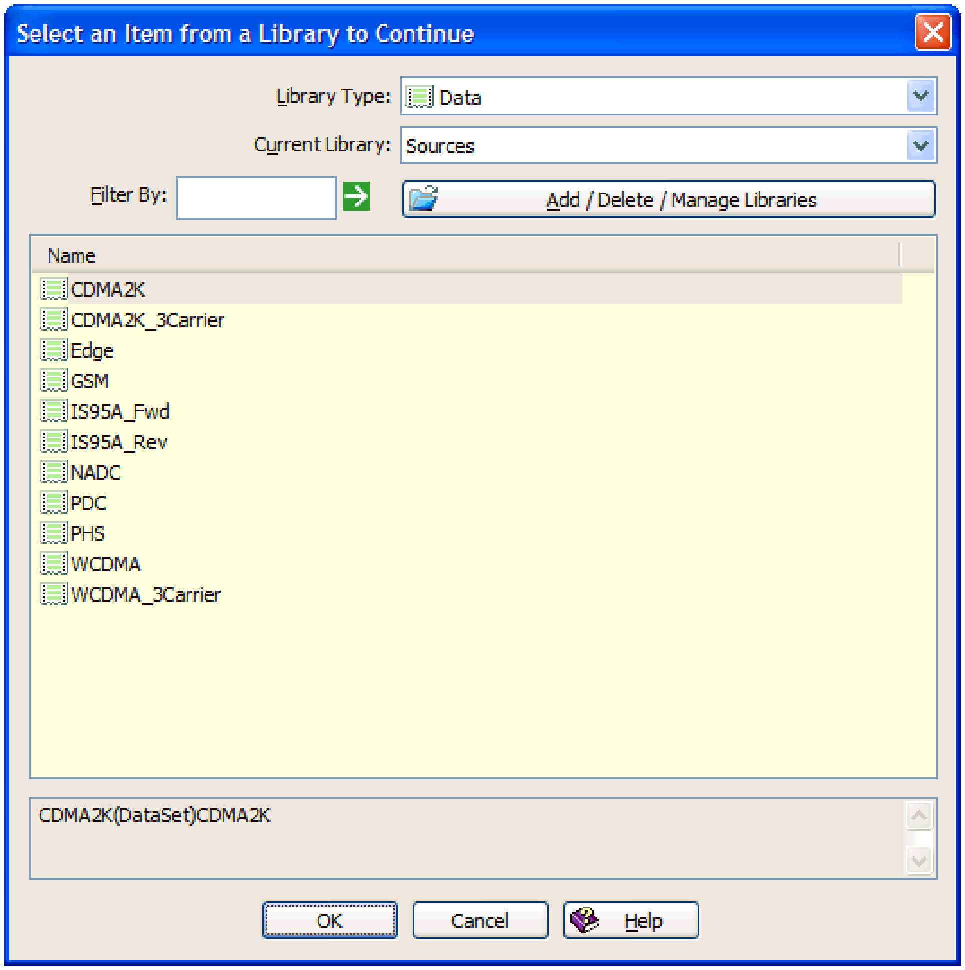
Adding Library Items to Your Workspace

Any object in a library can be added to your workspace. In the case of Symbols or Models you can just double-click (or edit) the object in the Library Selector. You can use the library selector to add any object type into your workspace.

If you don't want the docking library selector to be visible (taking up screen real estate) use the Add From Library Option as seen below.



Parts are special. They are not added (separately) to your workspace, but instead, are placed using a mouse onto a schematic design.



To Insert an Object from a Library into your workspace

1. Select **From Library...**
2. Set the **Library Type** to the type of object you want to insert in your workspace
3. Set the **Current Library** to the Library you want to add from
4. Double-click the specific Object you want to add.

Optimization

Practical design problems such as standard values, component tolerances, and parasitics may preclude a purely theoretical solution. When only a few variables in a schematic require adjustment, manual tuning is effective. As the number of variables increases, visualization of the multidimensional variable space is difficult, and tuning becomes less effective. Optimization is used to solve these types of problems. Optimization is often a compromise of conflicting requirements with no exact solution. Effective use of optimization consists of tuning variables according to some algorithm, running a simulation, comparing the results to specified goals, and repeating the process until an acceptable error level is achieved.

Contents

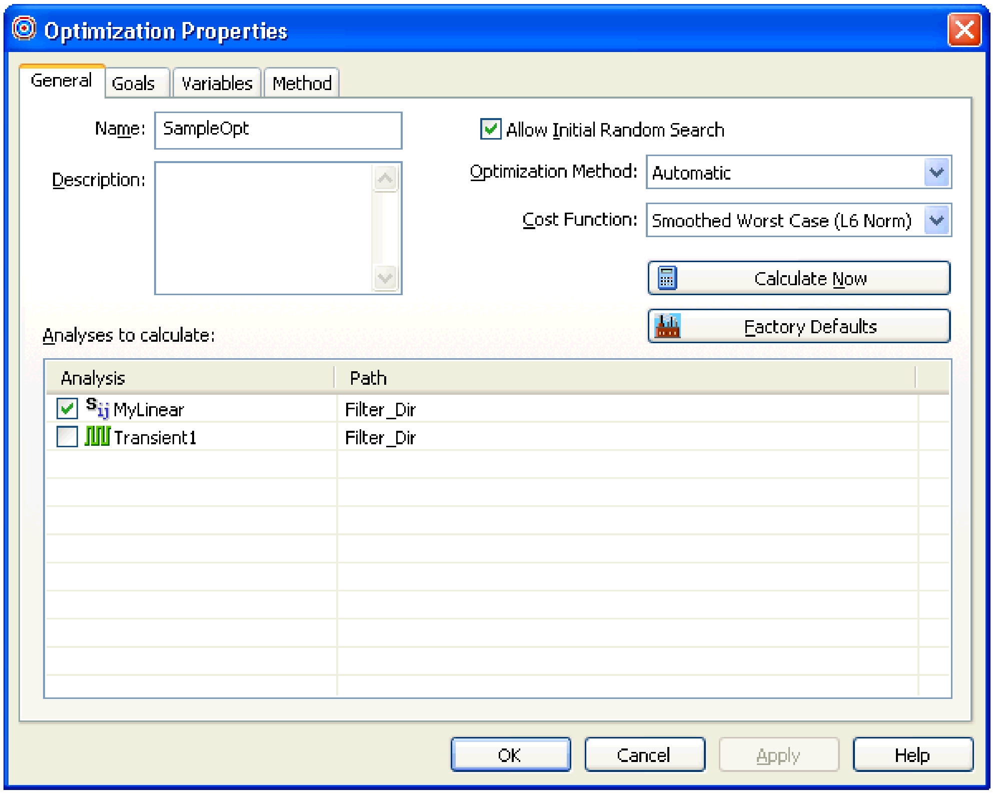
Creating an Optimization

To create an Optimization:

- Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Optimization** from the Evaluations submenu.

Select Analyses to Run

- The General tab page sets which analyses will run when you optimize, what optimization method to use, the name of the optimization, etc.



Name: SampleOpt

Description:

☑ Allow Initial Random Search

Optimization Method: Automatic

Cost Function: Smoothed Worst Case (L6 Norm)

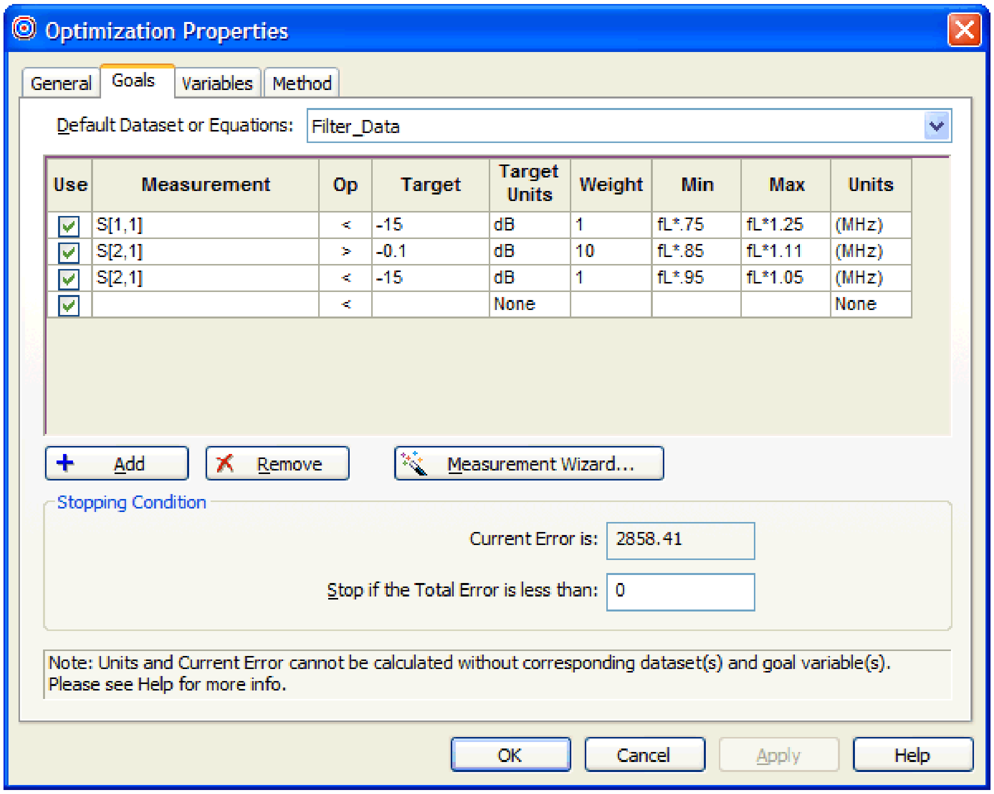
Analyses to calculate:

Analysis	Path
☑ MyLinear	Filter_Dir
□ Transient1	Filter_Dir

- **Name** – The name of the optimization, which will be shown in the workspace tree.
- **Description** – A description of this optimizaion, to help you keep track of multiple optimization and to hold notes specific to this item.
- **Optimization Method** – Which optimization technique to use
    - **Automatic** – Cycles between gradient and pattern searches until no further improvement is possible. This is the recommended setting.
    - **Gradient Search** – Locate a **local** minimum of the error terms.
    - **Pattern Search** – Locates a **global** minimum.
- **Cost Function** – Selects which cost function to use
    - **Least Squares (L2 Norm)** – A method used to find an approximate solution which minimizes the sum of the squares of the error terms. This is a second-order function.
    - **Smoothed Worst Case (L6 Norm)** – A sixth-order approximation, which maximizes the minimum value of the current position. This is the same cost function as the old, mis-named MiniMax option in earlier versions of Genesys.
    - **Worst Case (L∞ Norm)** – An infinite order approximation, which maximizes the minimum value of the current position.
- **Calculate Now** – Closes the property window and runs the optimization with the current settings.
- **Factory Defaults** – Resets the optimization settings to the original default settings (when you first installed Genesys).
- **Analysis List** – Determines which analyses will run when you optimize; check them on or off to enable them.

Select Goals

- Determine the goals of the optimization in the second tab.



Default Dataset or Equations: Filter_Data

Use	Measurement	Op	Target	Target Units	Weight	Min	Max	Units
☑	S[1,1]	<	-15	dB	1	fL*.75	fL*1.25	(MHz)
☑	S[2,1]	>	-0.1	dB	10	fL*.85	fL*1.11	(MHz)
☑	S[2,1]	<	-15	dB	1	fL*.95	fL*1.05	(MHz)
☑		<		None				None

Stopping Condition

Current Error is: 2858.41

Stop if the Total Error is less than: 0

Note: Units and Current Error cannot be calculated without corresponding dataset(s) and goal variable(s). Please see Help for more info.

- **Default Dataset or Equations** – Indicates the default source of the measurements, unless the measurement itself specifies another dataset.
- **Goal** rows – Each grid row describes a goal for the optimizer, which is discussed in

detail below.

- **Add** – Adds another row to the Goal table.
- **Remove** – Click a row to select it, then click Remove to get rid of it.
- **Measurement Wizard** – Brings up a window, to let you easily select a measurement to add to the Goal table.
- **Current Error** – Displays the error term, using the curent values of the measurements.
- **Stop if the Total Error is less than** – Specifies a stopping point for the optimization.

> **Note**
> **Current Error** is based on the current values of the measurements listed in the Goals table. If the measurement (or its dataset) does not yet exist, "N/A" will be displayed, since the error term cannot be evaluated. Also, if any of the measurements already meet the specified goal, its contribution to Current Error is zero (0), so changing its Weight will have NO EFFECT on the Current Error.

This optimization block is for a filter. It instructs Genesys to try to make the forward gain (S21) at least -.1 dB and the input reflection (S11) better than (less than) -15 and -25 dB. The frequency range (because Sij is frequency dependent) for all targets is an equation dependent on fL (defined in workspace Equations), with a loose range for S21 and two ranges for S11 (tighter for the -25dB constraint). Finally, the Weight of the S21 target is 10, meaning that the optimizer will put more importance on that target than on the two S11 targets.

Each row specifies an optimizer **goal**, using one or more of the **Measurements** described in the Simulation manual, as well as any other valid equation.

If **Use** is checked, this goal will be used by the optimizer; if unchecked it will be ignored

Op – The available operators are =, >, < and %.

- The = operator attempts to optimize to the specified target value. (note that the error function will rarely reach zero when using the = operator)
- < or > attempt to optimize a parameter to be less than or greater than the specified target value.
- The % operator attempts to flatten the specified parameter, without regard to specific value of the parameter. When using the % parameter, the target value is used to indicate acceptable peak-to-peak value of the given parameter. For instance; Target of db(S21) with operator % and target of 0.2 describes the desire to flatten S21 to within 0.2 dB. When the P-P value is less than the target, the term adds nothing to the error function. To specify a flattening command that flattens the desired measurement completely, enter a target of 0.

This four line set of targets for a bandpass filter:

Measurement	Op	Target	Weight	Min	Max
db(S21)	<	-40	1	10	40
db(S21)	>	-1	1	55	85
gd(S21)	%		1	100	130
db(S21)	<	-40	1	100	130

attempts to achieve at least 40 dB of rejection in lower and upper stopbands and less than 1 dB insertion loss with flat delay in the passband.

This four line optimization block for an amplifier

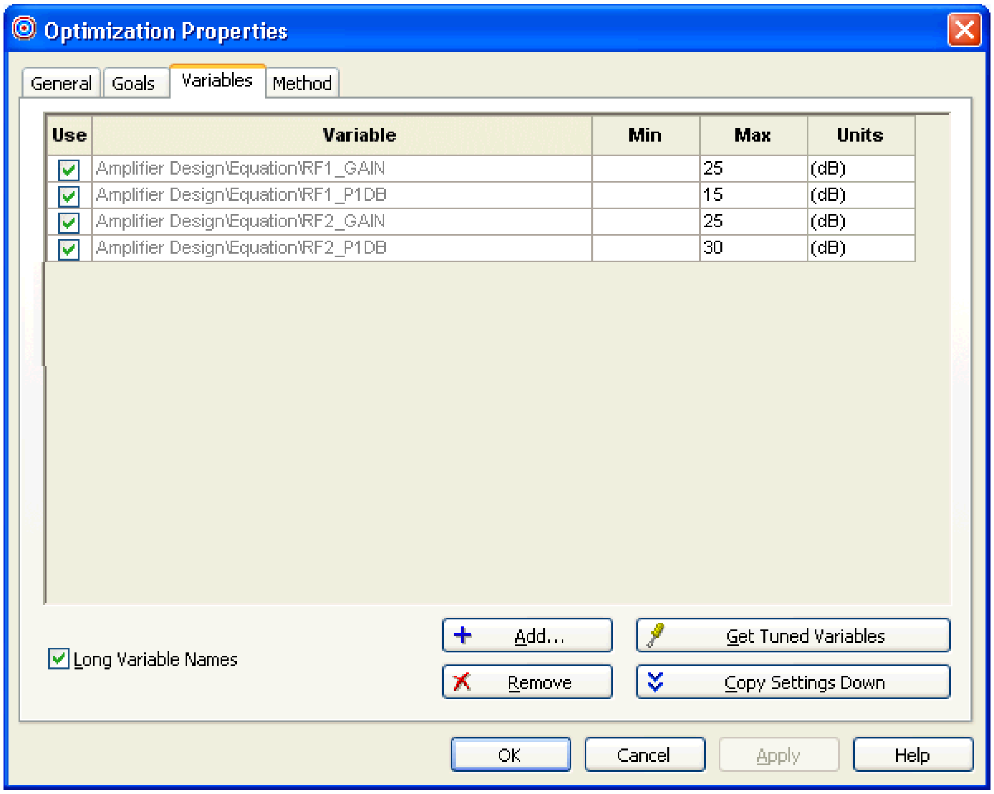
Measurement	Op	Target	Weight	Min	Max
db(S21)	>	11.5	1	2000	4000
db(S21)	<	12.5	1	2000	4000
db(S11)	<	-10	1	2000	4000
db(S22)	<	-10	1	2000	4000

attempts to achieve better than 10 dB of return loss in an amplifier with 11.5 to 12.5 dB of gain. A similar optimization block would be

Measurement	Op	Target	Weight	Min	Max
db(S21)	=	12	1	2000	4000
db(S11)	<	-10	1	2000	4000
db(S22)	<	-10	1	2000	4000

Tip: To reduce optimization time, specify a minimum number of frequency points.

Select Variables to Optimize



Use	Variable	Min	Max	Units
☑	Amplifier Design\Equation\RF1_GAIN		25	(dB)
☑	Amplifier Design\Equation\RF1_P1DB		15	(dB)
☑	Amplifier Design\Equation\RF2_GAIN		25	(dB)
☑	Amplifier Design\Equation\RF2_P1DB		30	(dB)

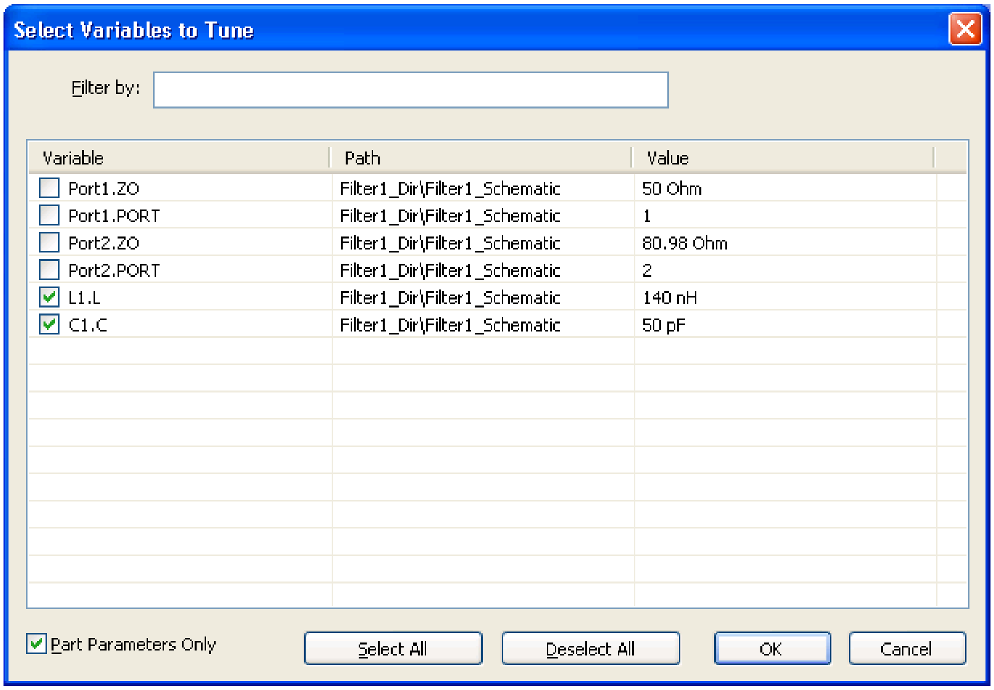
☑ Long Variable Names

When the optimization runs, it will automatically tune the variables listed in the Variables window.

- Enter a value in Min and/or Max to set limits on the variables. A blank entry implies no limit.
- Check **Long Variable Names** to see the entire name of the variable (as shown above).
- Click **Add** to add a new variable to the list. This brings up the **Select Variables to Tune** dialog, as described below.
- Click a Min or Max column entry then click **Remove** to remove a variable from the list
- Click **Get Tuned Variables** to get all currently tuned variables (the contents of the Tune docking window)

- Click **Copy Settings Down** to copy Min and Max from the top row to the remaining rows.

If you do not specify any variables to use, the Optimizer will use the current set of Tunable variables as seen in the Tune Window.



Select Variables to Tune

Filter by:

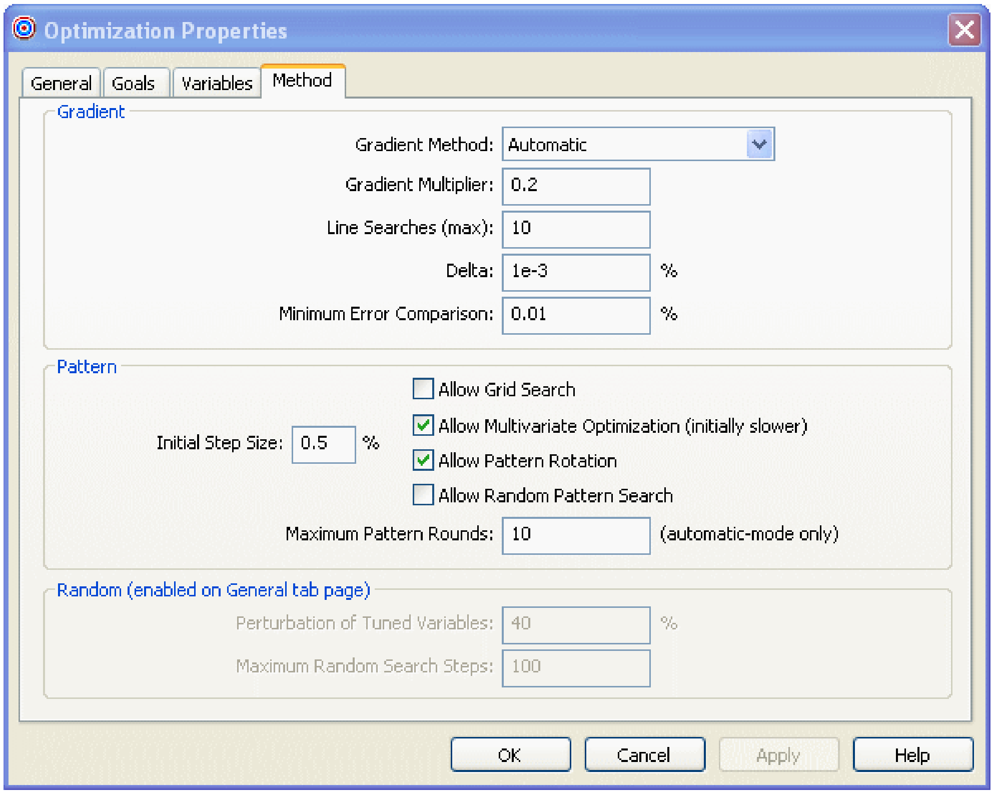
Variable	Path	Value
□ Port1.ZO	Filter1_Dir\Filter1_Schematic	50 Ohm
□ Port1.PORT	Filter1_Dir\Filter1_Schematic	1
□ Port2.ZO	Filter1_Dir\Filter1_Schematic	80.98 Ohm
□ Port2.PORT	Filter1_Dir\Filter1_Schematic	2
☑ L1.L	Filter1_Dir\Filter1_Schematic	140 nH
☑ C1.C	Filter1_Dir\Filter1_Schematic	50 pF

☑ Part Parameters Only

Check the variables you want to add to the optimization set.

Caution: It is very important that the frequencies at which the circuit is optimized are also frequencies included in the simulations. Optimization will not create any new simulation frequencies.

Select Optimization Method Options (advanced)



Gradient

Gradient Method: Automatic

Gradient Multiplier: 0.2

Line Searches (max): 10

Delta: 1e-3 %

Minimum Error Comparison: 0.01 %

Pattern

Initial Step Size: 0.5 %

☐ Allow Grid Search

☑ Allow Multivariate Optimization (initially slower)

☑ Allow Pattern Rotation

☐ Allow Random Pattern Search

Maximum Pattern Rounds: 10 (automatic-mode only)

Random (enabled on General tab page)

Perturbation of Tuned Variables: 40 %

Maximum Random Search Steps: 100

Options for the optimizer:

Gradient

- **Gradient Method** – selects the algorithm which the gradient optimizer will use.
  - **Automatic** – Automatically cycles through the gradient algorithms (listed below) until no further improvement is possible. This is the recommended setting.
  - **Steepest Descent** – This technique is a first-order optimization algorithm. It uses steps proportional to the negative of the gradient at the current location to find the local minimum.
  - **Fletcher-Reeves** – One of the three common conjugate gradient method functions; it may converge faster than the steepest-descent algorithm.
  - **Polak-Ribière** – This technique is an alternative to the Fletcher-Reeves formula which attempts to avoid the problem of progressively "lost conjugacy of the generated directions", due to inaccuracies of the linear searches and the non-quadratic terms of the Fletcher-Reeves method.
  - **Hestenes-Stiefel** – A third conjugate gradient method function, which is similar to the Lanczos method to symmetric matrices, but which uses coupled two-term recurrences, instead of the Lanczos three-term formulation.
- **Use Squared Error** – determines how quickly the algorithm accelerates.
- **Gradient multiplier** – the distance the gradient search travels at each iteration.
- **Line Searches (max)** by default the gradient search up to 10 times (and at least 5) along a single gradient line to find the best fit. This could cause a slowdown with no improvement in convergence.
- **Delta** – in percent, the amount the variables are perturbed to calculate a delta for the gradient. 1e-3 is average. If the optimizer gets stuck and can't improve you may want to decrease this. If the convergence is too slow you may want to increase this. Values above .1 or below 1e-8 are not recommended.
- **Maximum pattern rounds** – during automatic optimization the pattern search won't do more than this # of rounds before returning to gradient search.

Pattern

- **Initial step size** – Primarily for automatic optimization; this is the initial pattern step size. **Note:** This is now a percentage, but it used to be a fractional amount: 50% is the same as the previous 0.5 setting value.
- **Allow Grid Search** – The grid search algorithm is designed for only 2 or 3 parameters and then it generally isn't as good as a well tuned pattern search, but we allow it as an option for discontinuous searches. Enable this carefully as it may seem to hang the system with a lot of parameters.
- **Allow Multivariate Optimization** – If the problem does not converge you may want to enable this. This will converge better with paired parts (e.g. LC circuits) but will be slower on standard problems.
- **Allow Pattern Rotation** – This lets the pattern search be order independent.
- **Allow Random Pattern Search** – Enables random search during a Pattern search iteration.

Random – These settings are enabled on the General tab page, via the "Allow Initial Random Search" checkbox.

- **Perturbation of Tuned Variables** – The percentage amount to vary the selected tunable variables (on the Variables tab).
- **Maximum Random Search Steps** – Specifies how many search steps may be used.

Cost Function

A cost function is created from goal values, goal weights, and actual values.

For **Least Squares** and **Smoothed Worst Case** the equation is:

$$E = \sqrt{\sum_{n} \left((G_n - V_n)W_n\right)^p}$$

For **Worst Case Error** the equation is:

$$E = \max_{n}(|G_n - V_n| \cdot W_n)$$

where

- *n* = Number of goals
- *p* = Power of error function (2 for Least Squares, 6 for Smoothed Worst Case)
- *Gn* = Set of goal values
- *Vn* = Set of actual values
- *Wn* = Set of goal weights (default weight = 1)

Optimization routines attempts to minimize the cost error by changing user specified variables during an optimization simulation.

Weights

Default weight values are set to 1. Weight values greater than one (such as 10) make the goal more important. Weight values between zero and one (such as 0.1) make the goal less important.

For many optimizations, weight values are unnecessary. This is often the case when the specified goals are similar in value. Use of < and > operators instead of = also reduces the need for weights. When the error resulting from a parameter is zero because the condition has been satisfied, the weight multiplier has no effect.

When the optimized parameters represent a wide range of goals, weights are used to obtain a more desirable solution. Remember, optimization is a search for a compromise. Weights are not used to obtain a "correct" solution, but rather to "sculpt" a solution more desirable to the user. The user observes the optimization results and then modifies the parameter goals and weights to obtain a "better" solution.

Optimization Methods

This section documents additional technical details for the optimization methods.

Gradient

The following gradient methods can be used:

1. Steepest Descent
2. Fletcher-Reeves
3. Polak-Ribiere
4. Hestenes-Stiefel

Steepest Descent

Algorithm for Steepest Descent

1. Start at a point **X** $_{\mathbf{0}}$
2. Calculate **F(X** $_{\mathbf{0}}$**)**
3. Calculate the next **X** $_{\mathbf{n}}$ according to: $\mathbf{x}_{n+1}=\mathbf{x}_n-\gamma_n \nabla F(\mathbf{x}_n),\ n \ge 0.$
4. Calculate the next **F(X** $_{\mathbf{n}}$**)**
5. Repeat steps 3 and 4 until the minimum value is achieved

Nonlinear Conjugate Gradient Methods

The following methods fall into the nonlinear conjugate gradient methods:

1. Fletcher-Reeves
2. Polak-Ribiere
3. Hestenes-Stiefel

For a quadratic function:

$$f(x)=|Ax-b|^2$$

The minimum is obtained when the gradient is 0.

$$\nabla_x f=2A^\top(Ax-b)=0$$

For linear conjugate gradients methods a solution is sought for the linear equation:

$$A^\top Ax=A^\top b$$

However, the gradient alone is used to find the local minimum for the nonlinear conjugate gradient methods.

Algorithm for Initial Point

1. Start in the Steepest Descent direction: $\Delta x_0=-\nabla_x f(x_0)$
2. Perform a line search in the steepest direction until the minimum value is reached:

$\alpha_0 := \arg\min_{\alpha} f(x_0 + \alpha \Delta x_0)$ and $x_1 = x_0 + \alpha_0 \Delta x_0$

Algorithm for Remaining Points

1. Calculate the steepest direction: $\Delta x_n = -\nabla_x f(x_n)$
2. Compute β_n according to one of following nonlinear conjugate gradient methods
3. Update the conjugate direction: $\Lambda x_n = \Delta x_n + \beta_n \Lambda x_{n-1}$
4. Perform a linear search and optimize: $\arg\min_{\alpha_n} f(x_n + \alpha_n \Lambda x_n)$
5. Update the new position: $x_{n+1} = x_n + \alpha_n \Lambda x_n$

If the function is a pure quadratic the minimum will be reached in approximately N iterations. A non-quadratic will make slower progress.

Fletcher-Reeves

$$\beta_n^{FR} = \frac{\Delta x_n^\top \Delta x_n}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

Polak-Ribiere

$$\beta_n^{PR} = \frac{\Delta x_n^\top (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^\top \Delta x_{n-1}}$$

Hestenes-Stiefel

$$\beta_n^{HS} = \frac{\Delta x_n^\top (\Delta x_n - \Delta x_{n-1})}{\Lambda x_{n-1}^\top (\Delta x_n - \Delta x_{n-1})}$$

Pattern

Algorithm for Pattern Search

1. Find the initial error (**E** **$_{\text{initial}}$**) from all initial variable values
2. Find the variable that produces the lowest error (loop through each variable)
    1. Find error (**E** **$_{\text{hi}}$**) from current variable multiplied by adaptive step
    2. Find error (**E** **$_{\text{lo}}$**) from current variable divided by adaptive step
    3. Step size is reduced and repeats steps a and b for all variables to find the best error (**E** **$_{\text{best}}$**)
3. Now the best error has been found for the variable repeat step 2 to see if we can find the next variable that has the best error
4. Repeat step 3 to keep minimizing the error for each variable that is the largest contributor to the error

Random

Algorithm for Random Search

1. Find a random value [1] for every variable

2. Calculate the cost error function
3. If the stop error value has been achieved then stop
4. If we have not met the stop error value then repeat steps 1 through 3 until the maximum number of steps is reached

[1] A uniform distribution is used. If a value has hard limits those are used. However, if there are no limits then the specified percentages are used for both lower and upper limits.

Using Optimization

A variety of search methods can be used to minimize the error of a selected cost function. Cost functions relate goals and actual values to create an error number. The value of this cost error is mostly meaningless. However, it's magnitude indicates whether the solution is getting closer or farther away from the specified goals. Variables controlled by the optimization can be selected and limits can be placed on these variables by the user. The optimizer will change these variables according to the selected optimization method then will run a simulation and compare a current error with lowest achieved error to determine if the variable changes are going in the right direction. These variable changes will continue to occur until the cost error reaches zero or doesn't improve after various attempts.

The Optimization Methods are:

1. **Automatic** (default)
2. **Gradient**
3. **Pattern**

The cost functions are:

1. **Least Squares (L2 Norm)** (default)
2. **Smoothed Worst Case (L6 Norm)**
3. **Worst Case Error (L inf Norm)**

Optimization Methods

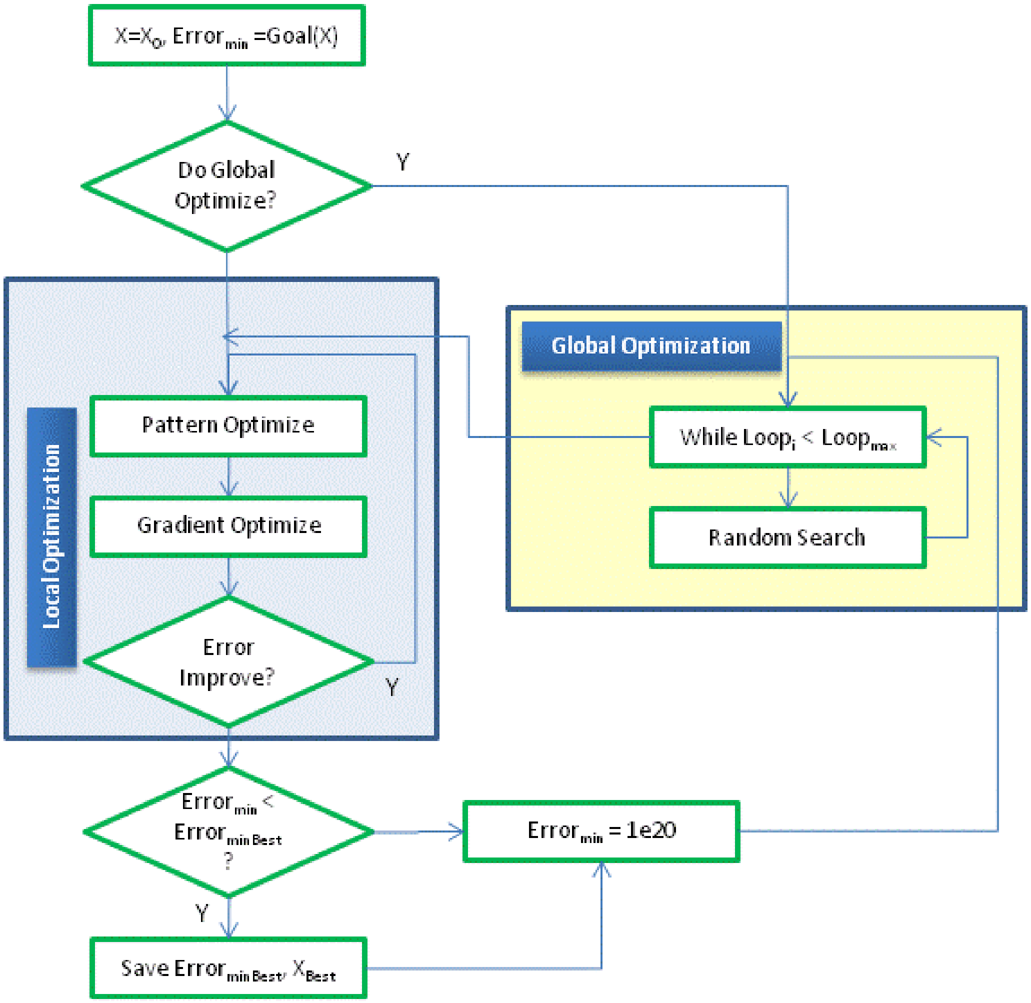
If enabled, a Random search is used initially to randomly find the best starting point.

> **Note**
> For more details on these methods see *Optimization Methods* (users).

Automatic

If the Random search has been enabled then this method will be invoked first to try and find the best overall global minimum. The Simplex method is used in conjunction with the random search to find the best global local minimum. Once the maximum number of global search attempts has been exceeded a pattern search is selected to find a local minimum. The pattern search is used in conjunction with the gradient method to find a local minimum. If the cost error is hasn't met the stop error criteria and yet there is no improvement in the error over a number of iterations the entire process will be repeated to ascertain whether a better minimum occurs elsewhere. This process will continue until the cost error reaches zero or the user specified stop error is reached.

Visually, that looks like:



Gradient

There are several gradient methods:

- **Automatic** (default)
- **Steepest Descent**
- **Fletcher-Reeves**
- **Polak-Ribiere**
- **Hestenes-Stiefel**

See *Optimization Methods* (users) for details about these methods.

When **Automatic** is selected the optimizer will try each gradient method to determine the best error. It will continue to use that method until the error doesn't improve. If the error doesn't improve another method will be selected.

Gradient methods are very effective in the early phase of an optimization. It is reasonably tolerant of poor initial component values and a large number of components. It often makes significant progress after only a few rounds. However, gradient search algorithm progress tends to halt before achieving optimum final values since this method tends to only find local minimums.

Pattern

In this algorithm, each element is stepped up and down and the errors are recorded. Only the element with the best error reduction is permanently changed. This process repeats by again searching for the element with the best error reduction. This algorithm therefore attempts to align with search in an optimum direction.

The Pattern search is based on an optimizer described in the paper, "The Effectiveness of Four Direct Search Optimization Algorithms", Randall W. Rhea, IEEE 1987 MTT-S International Microwave Symposium Digest, June 9, 1987. The implementation uses an adaptive and independent variable step size to increase the resistance to "hang."

Random

The method is used when a good starting point is not known. All variables are perturbed within a user specified percentage for a given number of steps. When a better error is achieved this becomes the starting point for the next randomization of variables. The minimum error found for the specified number of steps will be used as a starting point for the rest of the optimization.

This method if a fairly costly method since it doesn't rely on information from one step to the next to predict where the minimum is.

Note
The random search mode is only available when enabled on the 'General' tab.

Cost Functions

See *Cost Function* (users) for additional details.

Least Squares (L2 Norm)

This cost function is generally used when all goals are equally as important.

Smoothed Worst Case (L6 Norm)

This cost function is generally used when emphasizing the goal with the worst error. However, other goals are still considered with much less emphasis. This function doesn't have discontinuities.

Note
The **Smoothed Worst Case** is the same algorithm called **Minimax** in versions prior to Genesys 2010.05.

Worst Case Error (L inf Norm)

This cost function is used when only the worst case goal matters. All other goals are ignored unless they become the worst case goal as the worst case goal errors are being reduced.

Optimization Output

During a simulation the optimizer status is written to the status window, showing the following information:

Field	Description
Round	Integer value that indicates the total number of times a new direction if found to a lower cost or error value. One round involves 1 or more steps.
Step	Integer value that indicates the total number of calculation steps performed.
Search Type	Shows the optimization method used. For a **Gradient Search** the **Gradient Method** will also be shown along with the number of successful rounds yielding a lower cost or error value.
Best Error	Best error achieved over the total number of steps.
Current Error	Error for the current step.

Parts, Models and Symbols

Contents

Parts

A **part** is a the fundamental building block in any schematic. Each part contains both a **Model** and a **Symbol**, which, for maximum flexibility, may be changed independently.

Only **parts** can be placed on schematics. The part's **symbol** is the *image* on a schematic and the part **model** is *what is being simulated*. Users connect parts together on a schematic by placing wires between the part's symbol terminals. These *connection points* are called **nets**. The part itself maps symbol terminal pins to the model nets which are what actually gets simulated.

Double-click a part to access **Part Properties**, which provides a quick way to identify or change:

- The visible symbol - via the Advanced Settings button
- The model being simulated
- The model's parameter settings and
- Other part characteristics
- Furthermore, every part has the ability to ignore the parent model – which can **short** all simulation nets together or make all part nets have an **open** connection.

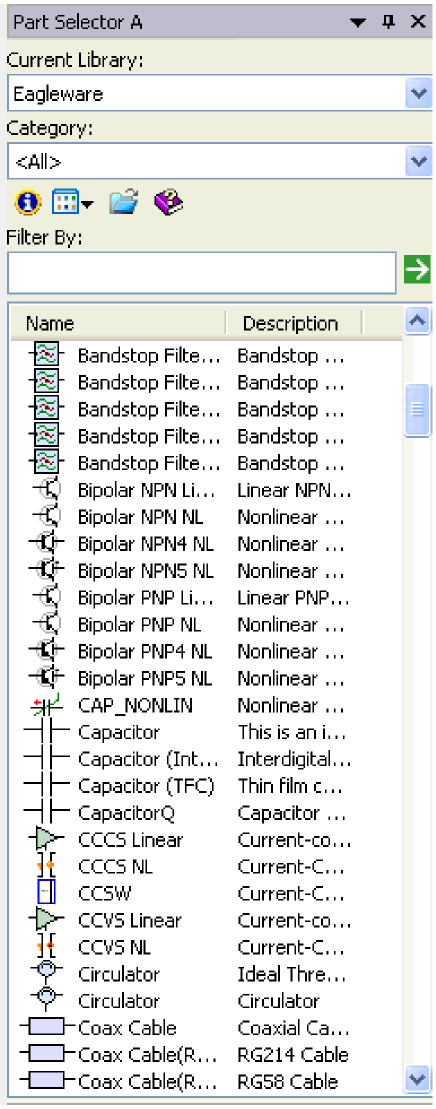
Each part supports a list of multiple models, with a means to manage these models in Part Properties.
Parts, their models, and symbols can be saved in libraries for reuse.

Placing Parts on a Schematic

From the Part Selector

To place a part from the part selector:

- **Click** on the part in the part selector.
- **Move** the mouse over the schematic. The mouse cursor will change to a plus sign when placed over the schematic.
- **Click** the schematic where the part is to be placed.



Current Library: Eagleware

Category: <All>

Filter By:

Name	Description
Bandstop Filte…	Bandstop …
Bandstop Filte…	Bandstop …
Bandstop Filte…	Bandstop …
Bandstop Filte…	Bandstop …
Bandstop Filte…	Bandstop …
Bipolar NPN Li…	Linear NPN…
Bipolar NPN NL	Nonlinear …
Bipolar NPN4 NL	Nonlinear …
Bipolar NPN5 NL	Nonlinear …
Bipolar PNP Li…	Linear PNP…
Bipolar PNP NL	Nonlinear …
Bipolar PNP4 NL	Nonlinear …
Bipolar PNP5 NL	Nonlinear …
CAP_NONLIN	Nonlinear …
Capacitor	This is an i…
Capacitor (Int…	Interdigital…
Capacitor (TFC)	Thin film c…
CapacitorQ	Capacitor …
CCCS Linear	Current-co…
CCCS NL	Current-C…
CCSW	Current-C…
CCVS Linear	Current-co…
CCVS NL	Current-C…
Circulator	Ideal Thre…
Circulator	Circulator
Coax Cable	Coaxial Ca…
Coax Cable(R…	RG214 Cable
Coax Cable(R…	RG58 Cable

From the Keyboard

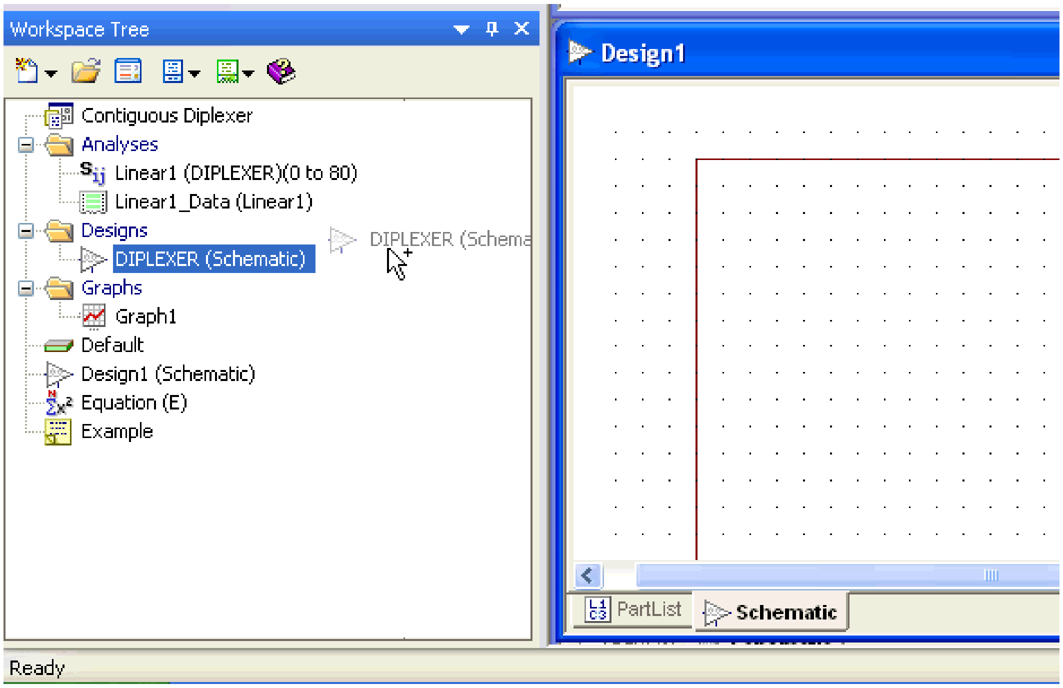
Certain frequently-used parts can be placed via the keyboard. Inside Genesys, use the Help / Keystroke Commands menu to display Appendix A, which lists the available parts.

From the Workspace Tree

Schematics can be dropped into other schematics to create a sub-network model. Models, and S-Parameter files can be dragged and dropped onto the schematic. When a schematic or model is dragged and dropped on a schematic a sub-network model is created along with a generic symbol. When an S-Parameter file is dragged and dropped onto a schematic the dataset part will placed on the schematic.

To place a part from the workspace tree:

- **Click** on the schematic, model, or S-parameter dataset.
- **Move** the mouse over the schematic. The mouse cursor will change to a plus sign when placed over the schematic.
- **Click** the schematic where the part is to be placed.

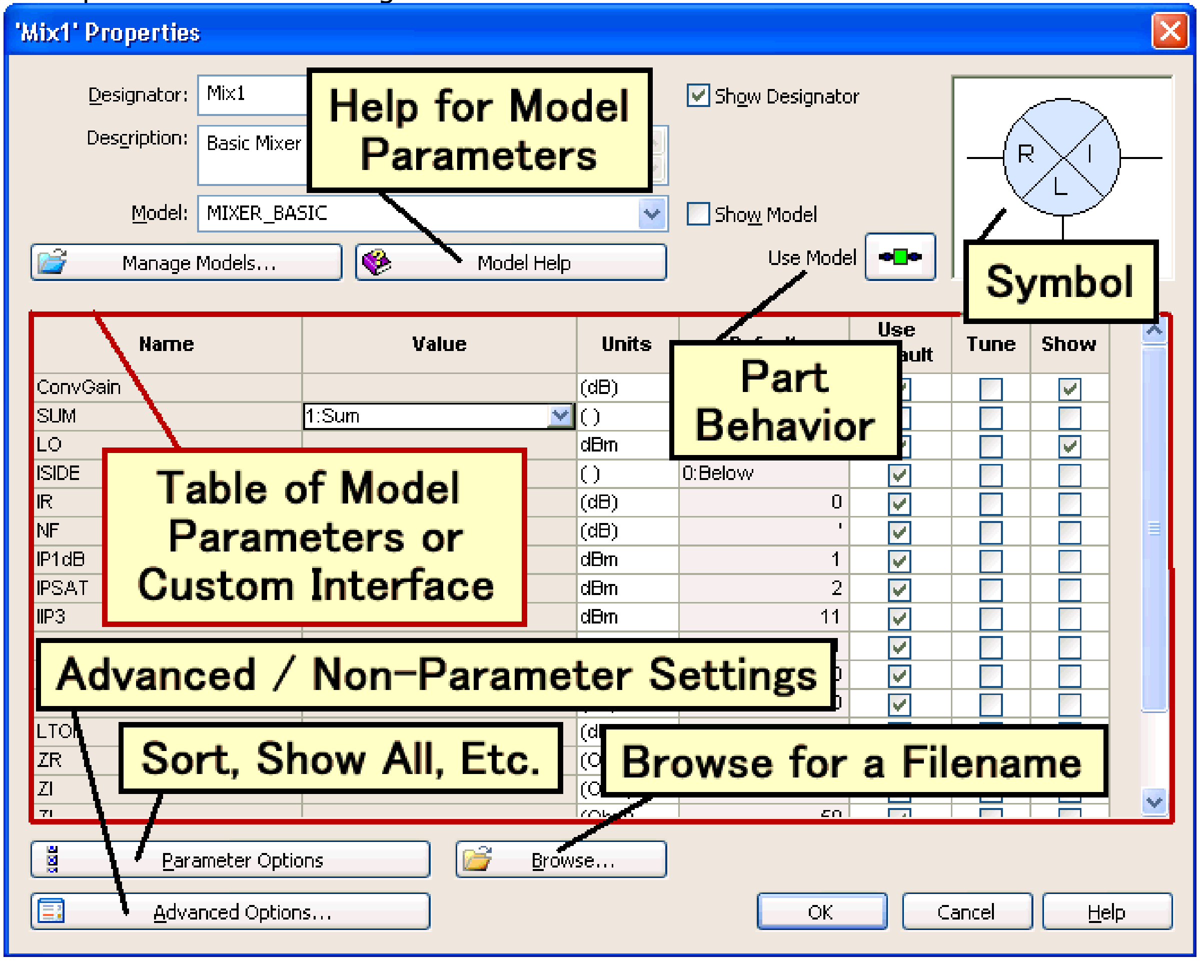


- Contiguous Diplexer
- Analyses
  - Linear1 (DIPLEXER)(0 to 80)
  - Linear1_Data (Linear1)
- Designs
  - DIPLEXER (Schematic)
- Graphs
  - Graph1
- Default
- Design1 (Schematic)
- Equation (E)
- Example

Ready

Part Properties

Each part has the following characteristics:



Designator: Mix1

☑ Show Designator

Description: Basic Mixer

Model: MIXER_BASIC

☐ Show Model

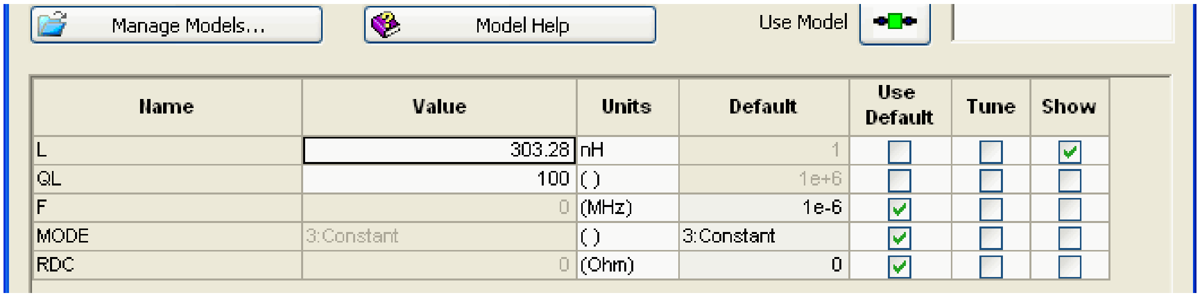
Name	Value	Units		Use ault	Tune	Show
ConvGain		(dB)			☐	☑
SUM	1:Sum	( )			☐	☐
LO		dBm			☐	☑
ISIDE		( )	0:Below	☑	☐	☐
IR		(dB)	0	☑	☐	☐
NF		(dB)	'	☑	☐	☐
IP1dB		dBm	1	☑	☐	☐
IPSAT		dBm	2	☑	☐	☐
IIP3		dBm	11	☑	☐	☐
				☑	☐	☐
				☑	☐	☐
				☑	☐	☐
LTO		(d				
ZR		(O				
ZI		(O				
ZI		(Oh	50			

- **Designator** - Descriptive text that appears on the schematic that references the part.

- **Show Designator** - When checked the designator will appear on the schematic.
- **Description** - Documentation info for the part. This info can be displayed in the part selector.
- **Model** - Name of the model to be simulated. The format is **ModelName@ LibraryName**. From this combo box the user can select the **active model**. The **model parameters table** will automatically be updated with this selection.
- **Show Model** - When checked the model name will appear alongside the designator on the schematic.
- **Manage Models** - When clicked will open a dialog box giving the user the ability to manage the models that are available for selection.
- **Model Help** - When clicked will open the help page providing descriptive information for all parameters in the model parameter table.
- **Part Behavior** - This button controls the behavior of the part. The four options are:
    - **Use Model** - Use the currently specified model ( ). This is the default state.
    - **Disable, Open** - All model net connections are opened ( ). No data will flow through this model.
    - **Disable, Short** - All model net connections are shorted ( ). The model is bypassed.
    - **Control by Equation...** - Use an equation expression to control the Part Behavior. The expression must evaluate to 0 = Use Model, 1 = Disable to Open, 2 = Disable to Short.
- **Symbol** - Shows a picture of the symbol associated with the part. This symbol can be changed on the **Advanced Options** dialog box.
- **Models Parameters Table** - This table contains the list of parameters specified by the model. In some cases there are models that have a custom interface that appears in the same area of the dialog box. See the model help for specifics on these models.
- **Browse** - This button will be enabled when the user clicks on a model parameter that needs a filename. The user can then browse to the desired file.
- **Advanced Options** - Gives the user the ability to change / create a symbol. Change its positioning and manage the mapping of the **symbol terminals** to the **model net**. (See details on each Advanced Options tab page below.)
- **Sort Alphabetically** - When checked will sort all parameters in the model parameter table alphabetically.

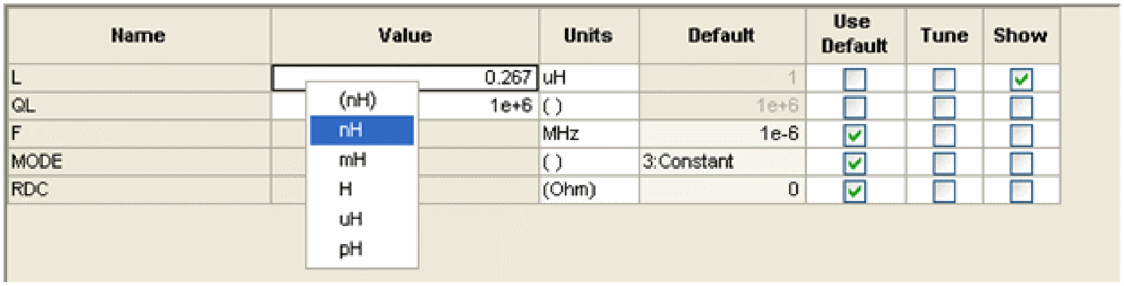
Model Parameters Table

This table lists all model parameters, their values, units, and characteristics.



Name	Value	Units	Default	Use Default	Tune	Show
L	303.28	nH	1	☐	☐	☑
QL	100	( )	1e+6	☐	☐	☐
F	0	(MHz)	1e-6	☑	☐	☐
MODE	3:Constant	( )	3:Constant	☑	☐	☐
RDC	0	(Ohm)	0	☑	☐	☐

Column Headings	**Description**
Name	Parameter name specified in the model. (Read-only)
Value	Parameter value. The values can be in following forms: numeric, enumeration, variable, or formula. Allowed enumeration values are specified by the model. A formula or an equation can be used in an enumeration field.
Units	Determines the units the parameter value is interpreted in.
Default	This is the default parameter value specified in the model. (Read-only)
Use Default	When checked the default model parameter value will be used.
Tune	When checked will make this model parameter value tunable.
Show	When checked with show the parameter and its value on the schematic.



Name	Value	Units	Default	Use Default	Tune	Show
L	0.267	uH	1	□	□	☑
QL	1e+6	( )	1e+6	□	□	□
F		MHz	1e-6	☑	□	□
MODE		( )	3:Constant	☑	□	□
RDC		(Ohm)	0	☑	□	□

Changing the units in the units drop down will NOT do unit conversion. If you want the value converted to a new unit you need to right click on the parameter's Value or Units field and select the new unit. An example right click menu is shown above.

For example, in the picture above when changing the unit for L from uH to nH:

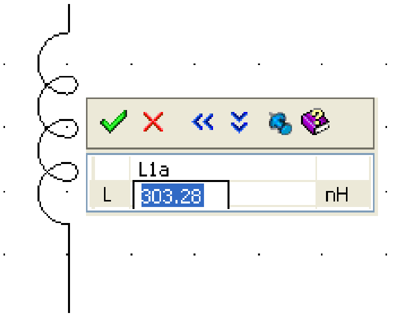
- if you use the right click menu the value will change from 0.267 to 267.
- if you use the normal drop down unit menu the value will remain 0.267.

Editing Part Parameters On a Schematic

Part parameters that appear on the schematic can be directly edited without opening up the part properties dialog box.
To edit the part parameters:

- Move the mouse pointer over the part text on the schematic. Note that the mouse pointer changes to resemble an I-beam (the text edit cursor). Click in the text.

The following editor will appear:



L1a

L 303.28 nH

Things you can do when editing a single part:

- Type a new value
    - Click outside the box or click Accept to close/accept the changes
    - Click a different part to Accept and switch parts (if pinned)

- Click in the S column to set a parameter show/hide
- Click in the T column to set a parameter tunable/fixed
- Click up/down arrows to edit other parameters
- Use a button to do more

The buttons on top:

Accept	Do an OK
Cancel	Cancel all changes
<< and >>	Expand and contract the box to show/hide the Tune and Show columns.
Up and Down	Expand and contract the box to show/hide non-shown parameters.
Pin / Unpin	When pinned, clicking another part will move the box. When unpinned, the box just closes (Accept).
Help	Brings up this help

Keys Supported:

Up Cursor	Move to prior value
Down Cursor	Move to next value
Tab	same as Down Cursor
Shift+Tab	same as Up Cursor
Enter	Accept
Esc	Cancel

Advanced Options

Part symbols and part connectivity can be changed on the advanced options dialog box.

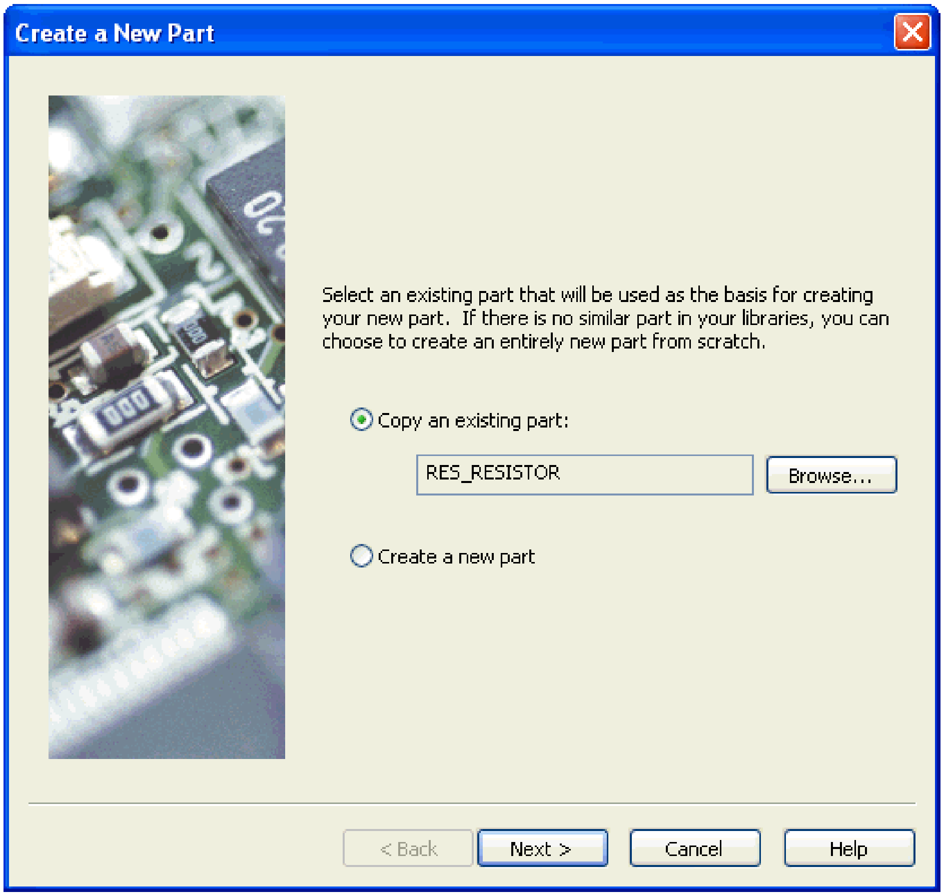
Click the Advanced Options button (Advanced Options ...) to bring up the Advanced Options dialog box.

See individual tab page topics below for additional information.

Creating a Part

When the user has a **model** and **schematic symbol** they want **combined into a part** they can use the Create Part Wizard to automate this process. The finished part must be placed in a library for future reuse.
To create a part using the Create Part Wizard:

1. Click **Action** on the menu and select **Create Part Wizard**.
2. Browse for an existing part to use as a starting point or begin with a blank part.



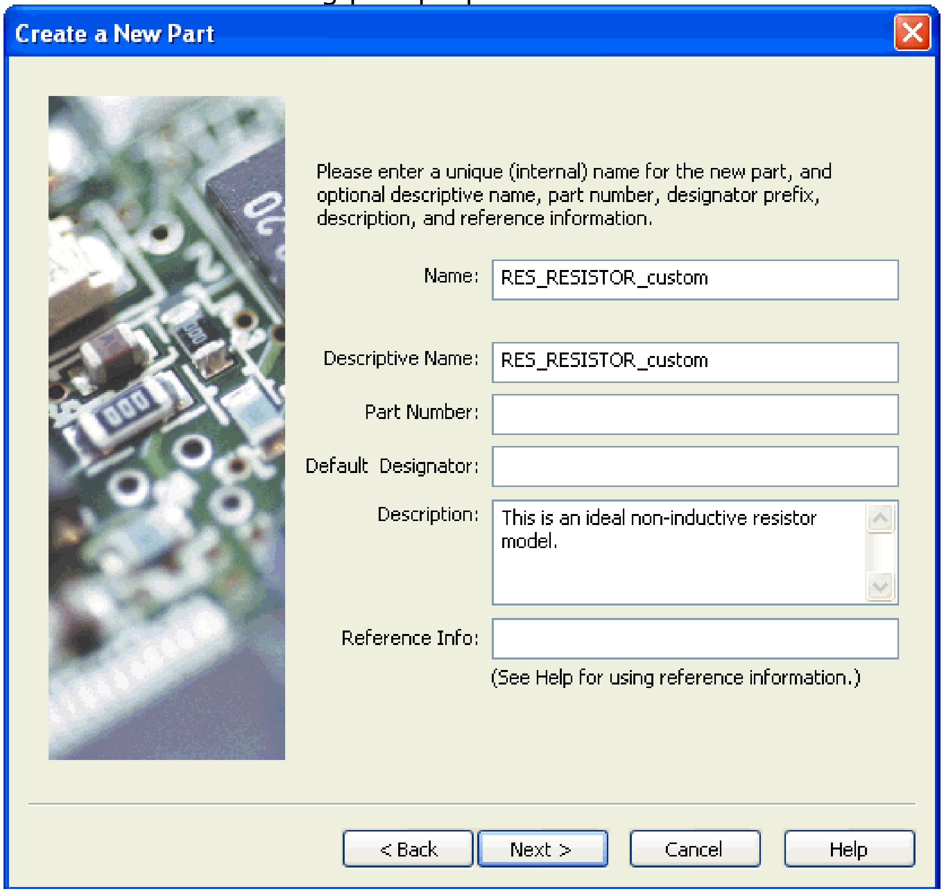
Select an existing part that will be used as the basis for creating your new part. If there is no similar part in your libraries, you can choose to create an entirely new part from scratch.

● Copy an existing part:

RES_RESISTOR

○ Create a new part

3. Click **Next**.
4. Fill in the descriptive fields as you want. If you re-used an existing part, the fields will be fill from the existing part properties.



Please enter a unique (internal) name for the new part, and optional descriptive name, part number, designator prefix, description, and reference information.

Name: RES_RESISTOR_custom

Descriptive Name: RES_RESISTOR_custom

Part Number:

Default Designator:

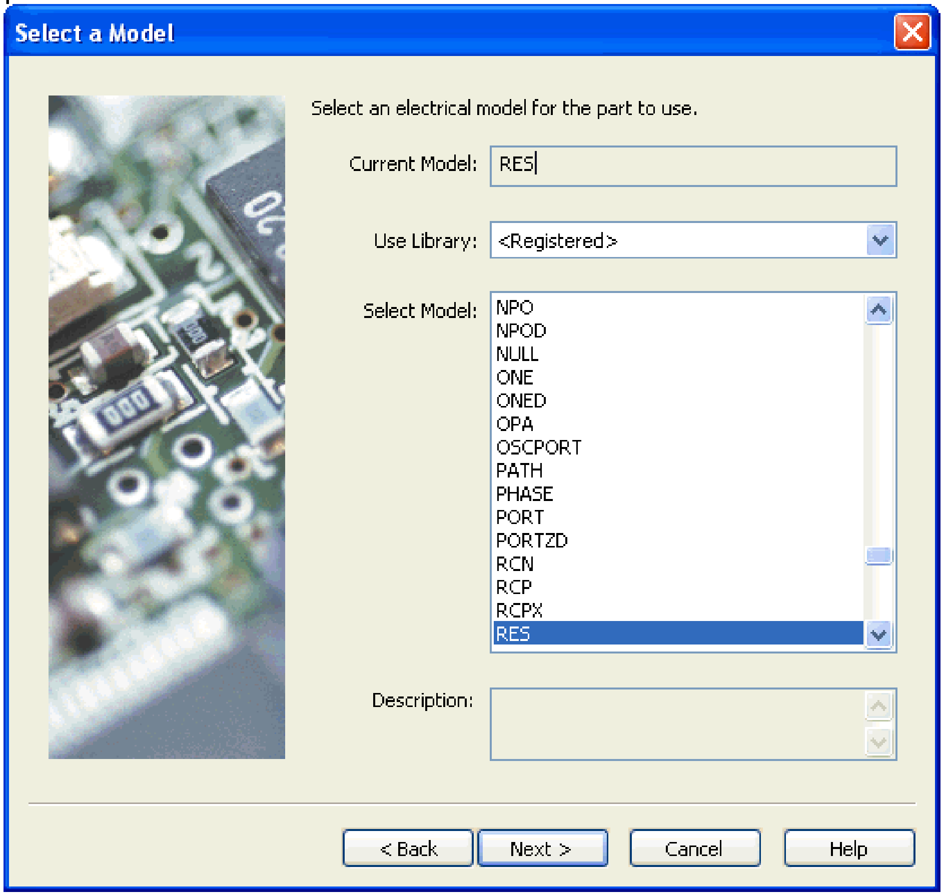
Description: This is an ideal non-inductive resistor model.

Reference Info:

(See Help for using reference information.)

Note: The RefInfo string is composed of |-delimited "name|reference-link" pairs of substrings. Each pair consists of a menu item and a command, usually a URL, directory path, or file (.doc, .txt, .htm, etc.)

5. Click **Next**.
6. Select the model to use. Note that the <registered> models are internal to the product and cannot be found in the Model libraries.



Select a Model

Select an electrical model for the part to use.

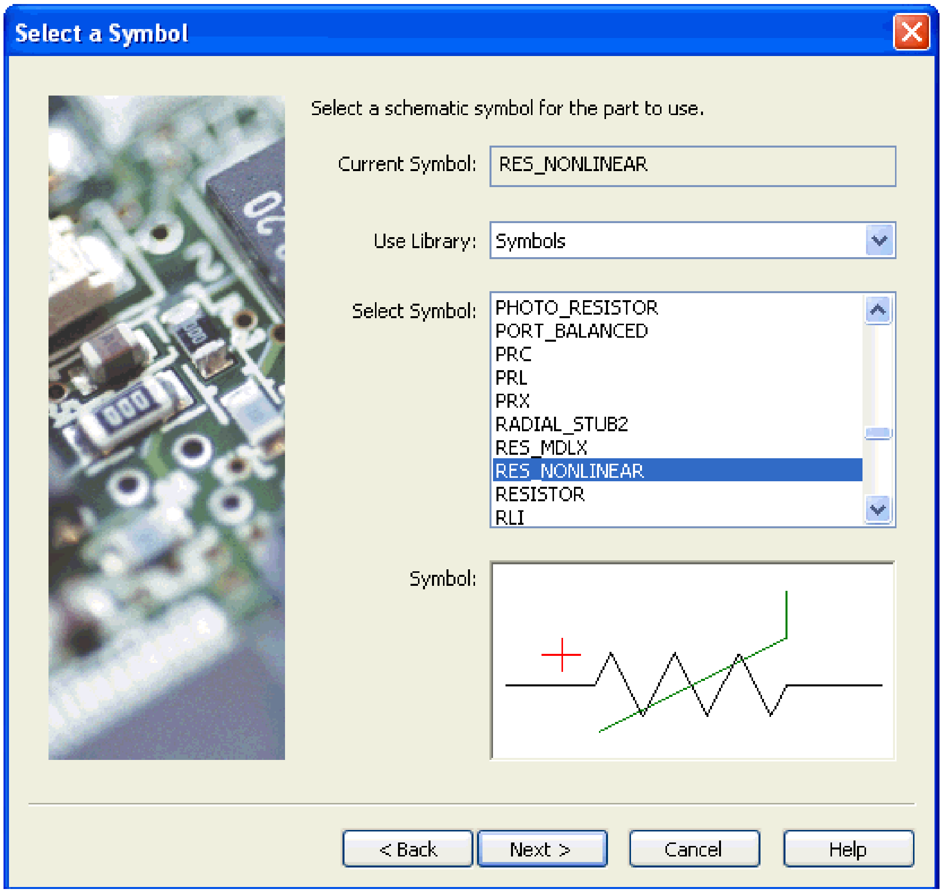
Current Model: RES

Use Library: <Registered>

Select Model:
NPO
NPOD
NULL
ONE
ONED
OPA
OSCPORT
PATH
PHASE
PORT
PORTZD
RCN
RCP
RCPX
RES

Description:

7. Click **Next**.
8. Select a symbol.



Select a Symbol

Select a schematic symbol for the part to use.

Current Symbol: RES_NONLINEAR

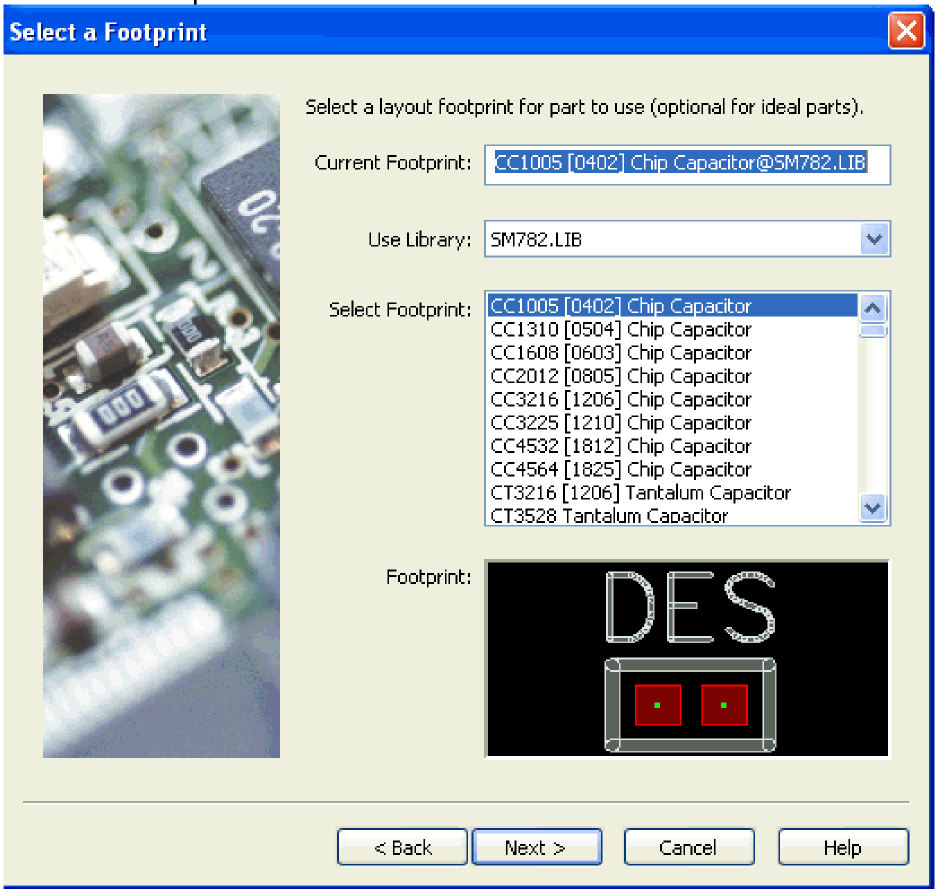
Use Library: Symbols

Select Symbol:

PHOTO_RESISTOR
PORT_BALANCED
PRC
PRL
PRX
RADIAL_STUB2
RES_MDLX
RES_NONLINEAR
RESISTOR
RLI

Symbol:

9. Click **Next**.
10. Select a Footprint



Select a Footprint

Select a layout footprint for part to use (optional for ideal parts).

Current Footprint: CC1005 [0402] Chip Capacitor@SM782.LIB

Use Library: SM782.LIB

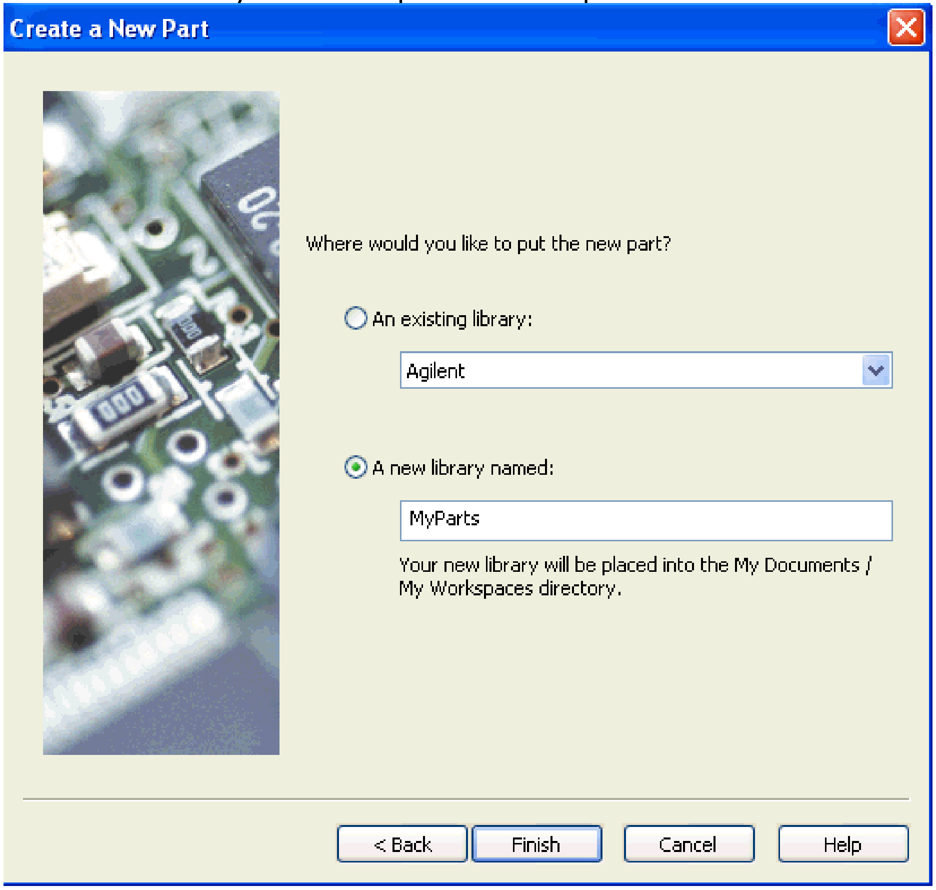
Select Footprint:

CC1005 [0402] Chip Capacitor
CC1310 [0504] Chip Capacitor
CC1608 [0603] Chip Capacitor
CC2012 [0805] Chip Capacitor
CC3216 [1206] Chip Capacitor
CC3225 [1210] Chip Capacitor
CC4532 [1812] Chip Capacitor
CC4564 [1825] Chip Capacitor
CT3216 [1206] Tantalum Capacitor
CT3528 Tantalum Capacitor

Footprint:

11. Click **Next**.

12. Select the library were the part is to be placed.



Where would you like to put the new part?

○ An existing library:

Agilent

● A new library named:

MyParts

Your new library will be placed into the My Documents / My Workspaces directory.

13. Click **Finish**.
14. Follow dialog prompts to add the new part to a library.

Models

A single part can support multiple models. Models can even be different types. Supported models types are:

- ***Math*** **(users)**
- **Code**
- ***Sub-Network Models*** **(users)**

When a model is changed any common properties from one model to the next are copied over to the new model. Furthermore, the old model is cached so if the user decides to return to the old model they won't need to re-enter the parameters.

Models can be saved in libraries or in the workspace tree. The model naming convention is: **ModelName@LibraryName**. Local model versions can be copied from an original model in a library. By default these local model copies have the same model name as the parent model in the library.

For more information on models see **User Defined Models** under the **Using Genesys** section in the users guide.

Tip: Use the toolbar Show/Hide (eyeball) button to show or hide all the model names on a schematic.

Hint: During a simulation a model appearing in the workspace tree will always be used before a model in a library even if the library name has been specified for the model.

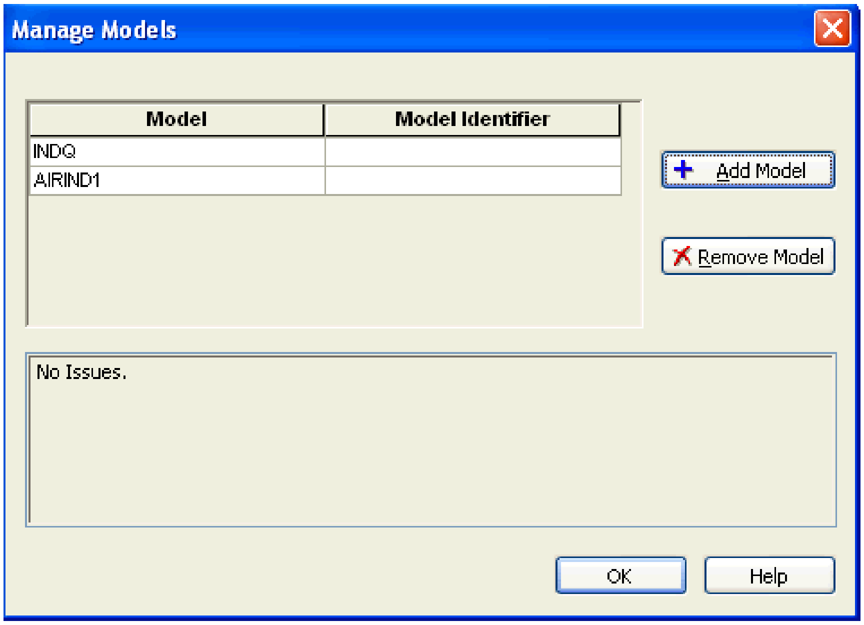
Note: A part can have several models and each model can have its own set of parameters. Those parameters with the same name and type are shared, i.e. a change in value for a shared parameter is a change for all models that have this parameter.

Changing a Model

Each part can contain several models. Pick one using the Model combobox in Part Properties.

Note: Certain specialized symbols, like those using %MACROS%, are designed for use with certain specific models. When you change a model, you may occasionally also need change the part's symbol, since the symbol might no longer match.

The list of available models can be changed using the model manager. Click on the Manage Models button (Manage Models...) to bring up the model manager dialog box.



Model	Model Identifier
INDQ	
AIRIND1	

No Issues.

Model Identifier - This parameter is used to distinguish between models of the same name.

Status Window - This window will alert users to potential errors or warnings.

To add a model:

1. Click on the Add Model button (Add Model)
2. Select the desired option:
    1. (A **list of models** in the workspace are listed)
    2. **From Library** (load a model from a library)
    3. **Enter Model Name** (select a model name)

To remove a model:

1. Click on the model to be removed
2. Click on the Remove Model button (Remove Model)

Creating a Model

To create a model select the type of model to be created. Follow those instructions:

- ***Math*** **(users)**
- **Code**
- ***Sub-Network Models*** **(users)**

Symbols

The part **symbol** is the graphical picture the user see's on the schematic that represents the part. Symbols can easily be created, modified, or changed for a given part.

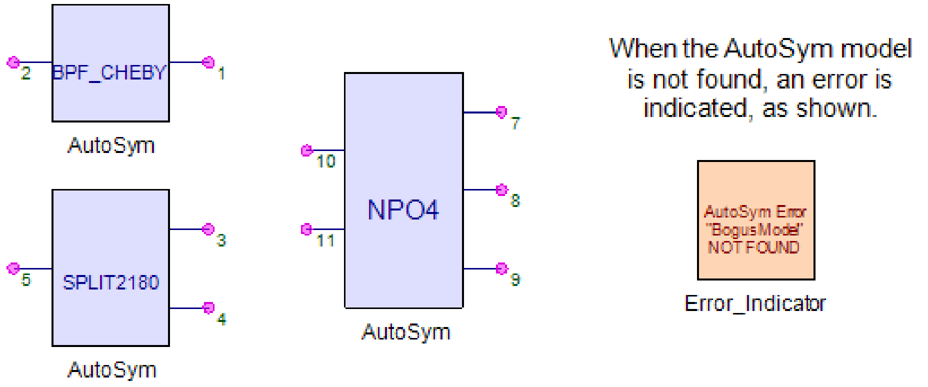
Algorithmic and Automatic (Dynamic) symbols

"Algorithmic" symbols are those that are created automatically by Genesys, as opposed to being hand-drawn and stored in an XML Symbol Library. These symbols are defined using a root name / modifier format. Usually the modifier is a simple number 'N' representing the number of ports, switch-throws, etc. Here's a list of commonly-used algorithmic symbols:

- **SwitchN** – switch with 'N' throws
- **SplitN** – an N-way splitter
- **Box-M-N** – a filled rectangle w/ M pins on left and N pins on right
- **N-Lines** – group of 'N' T-lines, labeled with numbers
- **N-Port** – long bar with terminals along the bottom
- **N-Chip** – horizontal chip (DIP)
- **N-VChip** – vertical chip (DIP)
- **N-QChip** – quad pack chip
- **N-SFile** – S-Parameter File
- **N-SData** – S-Parameter Dataset
- **N-SFile-Gnd** – S-Parameter File with ground
- **N-SData-Gnd** – S-Parameter Dataset with ground
- **N-XFile** – X-Parameter File
- **N-XData** – X-Parameter Dataset
- **N-XFile-Gnd** – X-Parameter File with ground
- **N-XData-Gnd** – X-Parameter Dataset with ground

> Note: These parts are normally built for the Genesys-standard schematic grid spacing of 1/6th inches. To generate symbols for an ADS-standard 1/8th grid, append the @SymbolsQtr suffix to the symbol name.

"Automatic" symbols are a subset of algorithmic symbols, which are based on a **model**. The symbol is a filled box with terminal pins on the left (input) and right (output); if in/out is not specified, the pins will be split evenly between the 2 sides. In addition, model port info is used to label the symbol pins.



When the AutoSym model is not found, an error is indicated, as shown.

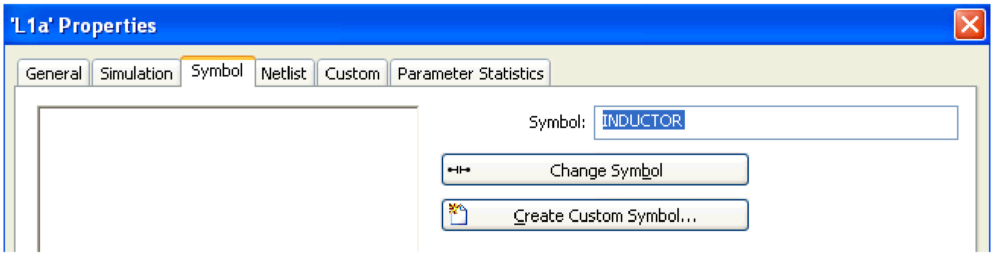
Automatic Symbols are specified using the special **AutoSym** symbol name

- Note that if the symbol cannot find the specified model, an error is shown as indicated.
- An *optional* model suffix may be specified; for example, use **AutoSym-MyModel@MyModelLibrary** to fully specify the model. The @Lib is optional, but can only be omitted if the model can be found without it.

Changing a Symbol

To change a symbol:

1. Click the Advanced Options button (Advanced Options ...) on the part properties dialog box.
2. Click the Change Symbol button (Change Symbol)
3. Select a new symbol or option
   1. (A **list of symbol names** in the workspace are listed)
   2. **From Library** (load a symbol from the library)
   3. **Edit Symbol Name** (change the name of a symbol)



Symbol: INDUCTOR

Create a Symbol

To create a symbol based on an existing part:

1. Right click the part and select Open / Symbol.
2. Modify the new symbol
3. Optionally, double-click the original part and change the symbol to the new custom symbol.

To create a new symbol "from scratch":

1. Click the New Item button ( ) on the Workspace Tree toolbar, then click "Designs", then select "Add Schematic Symbol"
2. Enter the symbol's name
3. Draw the symbol in the schematic area. Use the Annotation toolbar to place text, lines, arcs, and other drawing objects.
4. Place input (**i** key) and output (**o** key) ports where symbol terminals are to be located.

   > Note: Ports do not appear on the schematic when the symbol is used in a part.

5. Connect the symbol to the ports. These connection points are the connection points seen on a schematic when a part is placed.
6. Change the **port designator** to give the **symbol terminals** a name. This name is used to map symbol terminals to model nets.

Displaying Parameter Values on a Symbol:

When any (not just an algorithic) symbol is drawn on a schematic, symbol text is processed prior to display, using a technique called "Macro Substitution". The text within the '%' characters will be replaced with the appropriate value. For example, Name=%Model% would be displayed as "Name=Resistor" on a symbol using a resistor model.

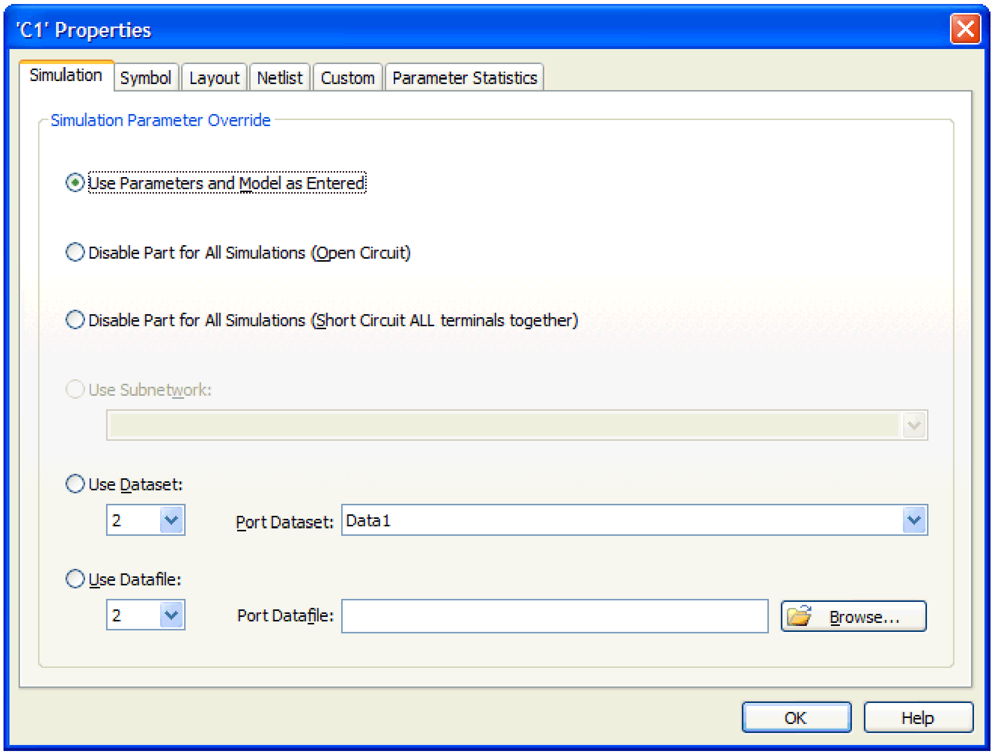
For example, when "Impedance = %L%" is drawn on a schematic, the value of parameter 'L' is retrieved from its model and the result is "Impedance = 1.5". Another common use is to place the model name on the symbol.

To use this advanced feature, place special "macro" strings in any symbol text:

1. Place a text annotation anywhere on the schematic symbol
2. Double-click it and change the text, so that it includes one or more macros from the table below.
3. Click OK

Macro	**Result**
%Model%	Name of the model attached to the schematic part
%MODEL%	Name of the model in UPPERCASE
%Des%	The part designator: R1, L3, etc.
%ParameterName%, where the name is any model parameter name, such as R, C, L, etc.	The actual value of the parameter.
%%	Displays a single % character.

Simulation Options



● Use Parameters and Model as Entered

○ Disable Part for All Simulations (Open Circuit)

○ Disable Part for All Simulations (Short Circuit ALL terminals together)

○ Use Subnetwork:

○ Use Dataset: 2 Port Dataset: Data1

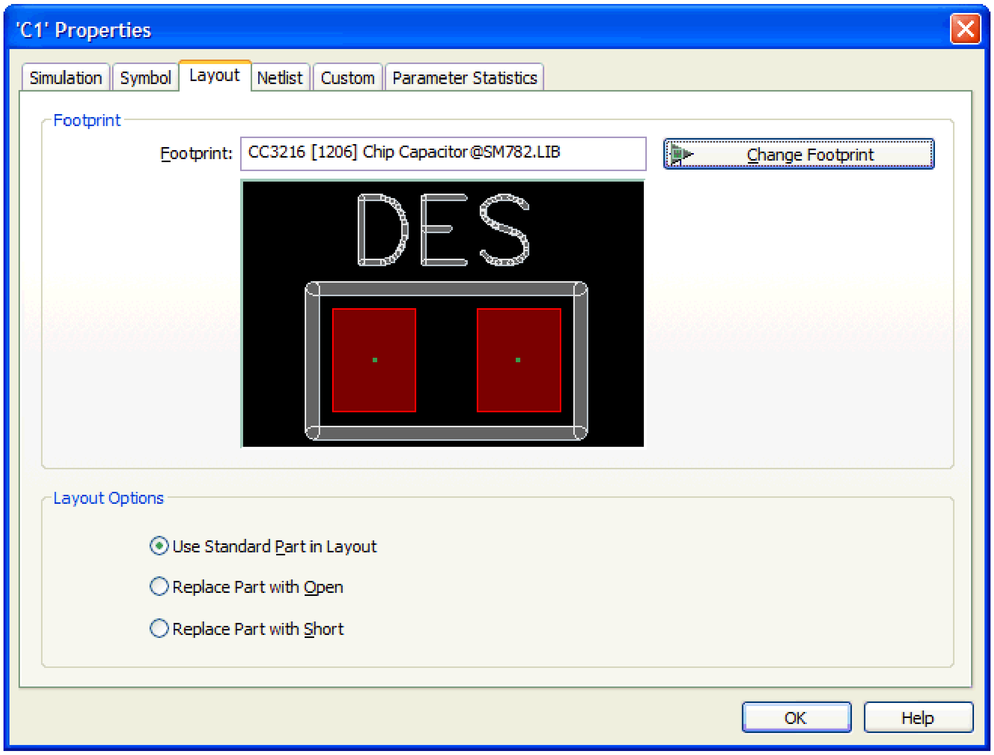
○ Use Datafile: 2 Port Datafile:

The Simulation tab provides another way to override the model for a schematic element.

- **Use Parameters and Model as Entered** – Use the Model specified (default behavior).

- **Disable Part for All Simulations (Open Circuit)** – Omits the part from simulations (leaving the adjacent elements disconnected).
- **Disable Part for All Simulations (Short Circuit ALL terminals together)** – Omits the part from simulations (simulates part as connecting wires).
- **Use Subnetwork** – For simulations, replaces the part with a subnetwork in the workspace.
- **Use Dataset** – For simulations, substitutes an S-Parameter dataset for the element's model. The combo-box selects the number of ports to use.
- **Use Datafile** – For simulations, substitutes an S-Parameter file for the element's model. The combo-box selects the number of ports.

Layout Options



Footprint: CC3216 [1206] Chip Capacitor@SM782.LIB

● Use Standard Part in Layout

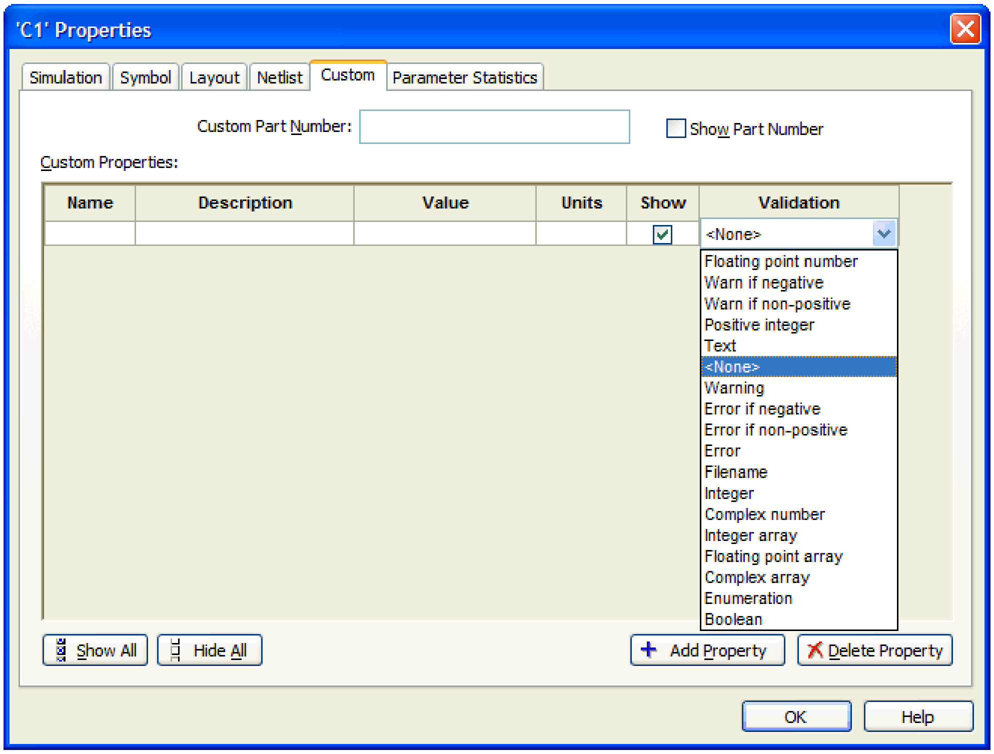
○ Replace Part with Open

○ Replace Part with Short

The Layout tab specifies the part's layout footprint and provides another way to override the model for a layout element (for simulation with Empower and/or Momentum).

- **Footprint** – Specifies the Layout Footprint to use for this part.
- **Change Footprint** – Allows you to select another footprint to use (from a library).
- **Use Standard Part in Layout** – Use the Model/Footprint as specified (default behavior).
- **Replace Part with Open** – Omits the part from Layout simulations (leaving the adjacent elements disconnected).
- **Replace Part with Short** – Omits the part from Layout simulations (simulates part as connecting wires).

Custom Properties



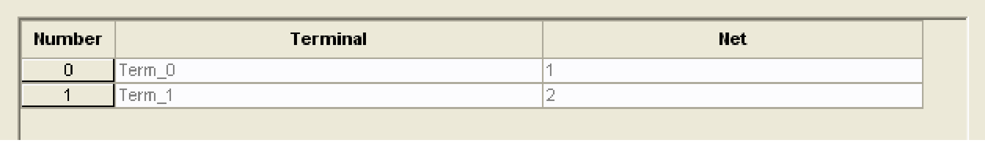
Name	Description	Value	Units	Show	Validation
				☑	<None>

- **Custom Part Number** – Specifies a part number to be associated with this part.
- **Show Part Number** – If checked, the part number will be shown on the schematic, next to the element.
- **Custom Properties** – Additional, user defined, parameters for this part (use the Add Property button to insert new items).
    - **Name** – The name of the property.
    - **Description** – A long description of the property and (perhaps) how to use it.
    - **Value** – The setting value.
    - **Units** – The units of the value.
    - **Show** – Should the property be displayed on the schematic, next to the part?.
    - **Validation** – What are the legal values? If the value is invalid, an error or warning will be posted to the Errors Window.
        - **Floating point number** – The value should be a floating point number, lisk 3.141592 (a double, not text, complex, etc).
        - **Warn if negative** – Post a warning, if the value is less than zero (< 0)
        - **Warn if non-positive** – Post a warning, if the value is less than or equal to zero (<= 0)
        - **Positive integer** – The number must not have a fractional part and it must be greater or equal to one (>= 1)
        - **Text** – a string parameter
        - **<None>** – No validation is performed
        - **Warning - always warns** – This parameter always posts a warning
        - **Error if negative** – Post an error, if the value is less than zero (< 0)
        - **Error if non-positive** – Post an error, if the value is less than or equal to zero (<= 0)
        - **Error - always fails** – This parameter always posts an error
        - **Filename** – The parameter is the name of a file (text) and the Browse button will be enabled.
        - **Integer** – The parameter cannot have a fractional part (after a decimal '.').
        - **Complex number** – The value has a real and imaginary part.
        - **Integer array** – The value is a list of integers.
        - **Floating point array** – The value is a list of floating point numbers.

  - **Complex array** – The value is a list of complex numbers.
  - **Enumeration** – The value is picked from a list of named values.
  - **Boolean** – only 1 (true) or 0 (false) is allowed.
- **Show All** – Checks all the "Show" checkboxes.
- **Hide All** – Unchecks all the "Show" checkboxes.
- **Add Property** – Adds a new, blank property.
- **Delete Property** – Removes the currently selected property.

Netlist Options

The **Netlist** tab page shows the current part connectivity. (That is, which terminal is connected to which schematic network node.)



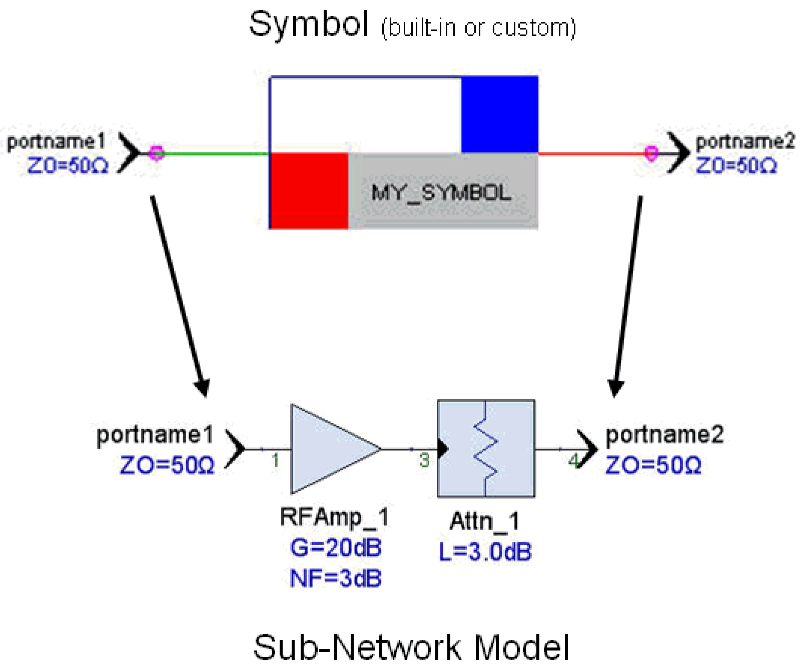
Number	Terminal	Net
0	Term_0	1
1	Term_1	2

Terminal - Names of the symbol terminals.
Net - Name of the schematic net the symbol is connected to.

Note: These fields are read-only unless there is no schematic. Connectivity is then determined by the names in the Net field.

Mapping Symbols to Models in Parts

The mapping between schematic symbols and models nets is through **port names and numbers** connected to those symbol terminals or model nets. The mapping **precedence** if first by **port name** and then **port number**. If port names match then port numbers are ignored.



Note: Symbol port naming is important otherwise the model may appear in the simulation backwards because model nets were inadvertently connected to the wrong symbols terminals.

Finding Symbols and Models during Simulation

ℹ If a version of the symbol or model is contained in the workspace tree that symbol or model will be used for display and simulation purposes regardless of whether the symbol or model is located in a library or not. If they are not found in the workspace then the symbol or model will be retrieved from the library.

Part Properties for Scripts

A part in Genesys consists of a footprint and any number of sections with each containing a model and a symbol. For example, a dual op-amp part might have a footprint (perhaps SOIC) and two identical sections. Each section contains an op-amp model and an op-amp symbol. For ease in drawing, Schematic supports a short symbol (accessed with the Shift key when you place a part).

To change the values entered in a part, double-click the part in the schematic and select parameter options.

Each section of a part can contain a list of models and symbols that are standard alternates for the part. Property names are case-sensitive.

Parts might also contain reference information in the form of Web pages or files, default settings to set default parameters, a category list to select from a part library, and a button ID for more information.

Standard Library Properties

Property	**Description**	**Required**	**Default**	**Example**
Inherit	A list of custom part properties that are inherited by a part when it is placed in a schematic.	No	None	Cost|On Hand|Alternate

Standard Part Properties

Property	**Description**	**Required**	**Default**	**Example**
XName	A unique name for the part.	Yes	None	10_CAP_MUR_0805_001_S_CAP_MDLX
UserName	The friendly name shown in the list.	Yes	XName value	10pF Murata 0805
Footprint	A footprint name and footprint file pair.	No	Genesys layout association table	MDLX_CAP_MUR_0805_001 @Modelithics.LIB
Category	Semicolon-delimited list of categories.	No	Member of the All category	Lumped;Capacitors;Modelithics

Description	Friendly part description.	No	None	CAP - Murata - 0805 10-470 pF <GRM215C1H>
Default	A list of default values when the part is placed. Delimited by "|".	No	Model defaults	C=10|QC=1000
RefInfo	Reference information is accessible using the RefInfo button on the Part Selector toolbar. See note below.	No	None	Murata|http://www.murata.com
Sections	The container for the Model and Symbol sections.	Yes	None	
Section *n*	One container per section.	At least 1	None	
- Model	The model name.	Yes	None	CAP_MUR_0805_001_S
- ModelSet	A set of possible models, semicolon-delimited.	No	None	CAP_MUR_0805_001_S;CAP
- Symbol	The symbol and symbol file names.	Yes	None	CAP_MDLX@Modelithics_CLR.wsx
- SymbolSet	A set of possible symbol and file pairs.	No	None	CAP_MDLX@Modelithics_CLR.wsx
- ShortSymbol	The short symbol used when you place a part with the Shift key held down.	No	The long symbol	CAP_MDLX@Modelithics_CLR.wsx
SimType	Simulation type. Possible values are P, S, O, N, and D for parameters (default), short, open, network, or datafile.	No	P	P
Subnet	If the simulation type is N, this is the name of the subnetwork.	No		Network_A
Datafile	If the simulation type is D, this is the name of the S-data file.	No		SampleSData.s2p
NumPorts	If the simulation type is N, this is the number of ports in the subnetwork.	No	2	2

Note: Normally, you do not edit the XName, UserName, and Description properties directly. You edit these properties by changing the part name and description in the properties window for the part.

Note: The RefInfo string is composed of |-delimited "name|reference-link" pairs of substrings. Each pair consists of a menu item and a command, usually a URL, directory path, or file (.doc, .txt, .htm, etc.). These strings are for hooking up whatever documentation the user wants to the RefInfo button.

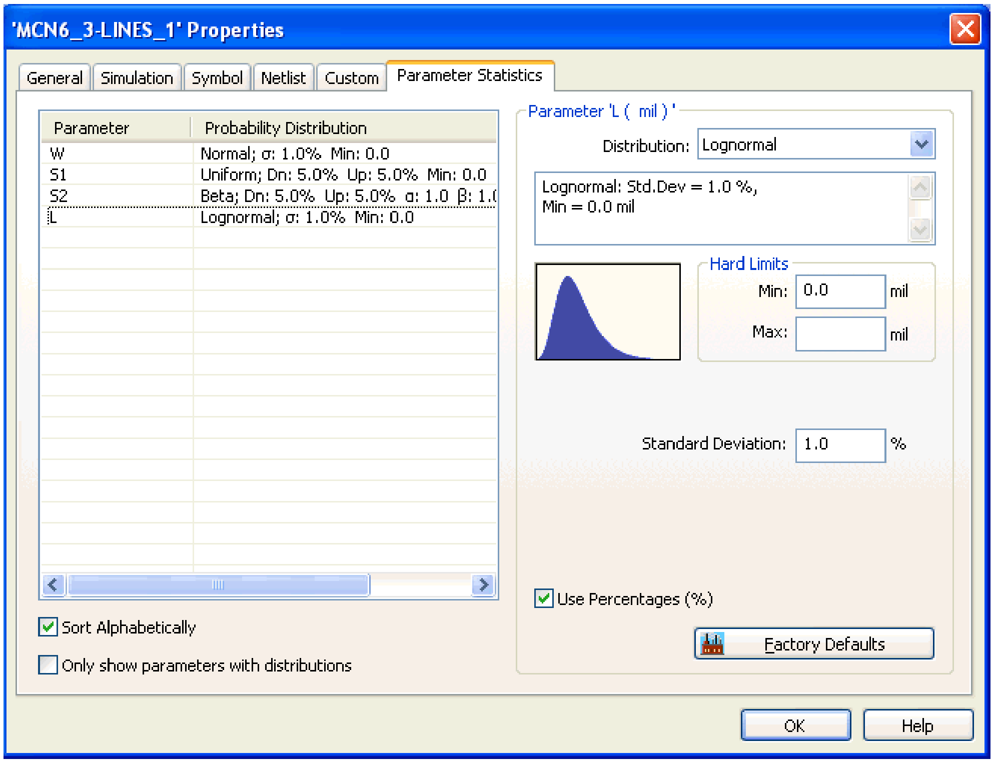
Part Parameter Distributions

Parts often come with known precision (10% resistors or 2% capacitors). In the **Part Properties** dialog you can set the precision of parts by specifying the probability distribution of each parameter. This distribution will be used by default for Yield and Monte Carlo evaluations.

⚠ Warning: part parameter distributions are not compatible with versions of Genesys prior to 2007.08!

To set a parameter distribution, edit the part properties and select the **Parameter Statistics** tab.

In this example image we have 4 parameters with different distributions. The probability distribution column shows a summary of each parameter's settings.



'MCN6_3-LINES_1' Properties

Parameter	Probability Distribution
W	Normal; σ: 1.0% Min: 0.0
S1	Uniform; Dn: 5.0% Up: 5.0% Min: 0.0
S2	Beta; Dn: 5.0% Up: 5.0% α: 1.0 β: 1.(
L	Lognormal; σ: 1.0% Min: 0.0

☑ Sort Alphabetically

☐ Only show parameters with distributions

Parameter 'L (mil) '

Distribution: Lognormal

Lognormal: Std.Dev = 1.0 %,
Min = 0.0 mil

Hard Limits
Min: 0.0 mil
Max: mil

Standard Deviation: 1.0 %

☑ Use Percentages (%)

To change a parameter, click the parameter in the Parameter column. The right hand side will be populated with that parameter's settings. Change the distribution, limits and settings then click OK or change parameters.

Distribution Settings	
Distribution	Select from the possible set of distributions governing a parameter, such as Normal or Uniform.
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.
Factory Defaults	Click this button to use reasonable default settings for a specific distribution.
Optional settings	
Standard Deviation	The standard deviation of the distribution for those parts which use one. This can be in Percent (related to the parameter value) or a united value.
Alpha / Beta	For the Beta distribution, unit less values to set Alpha and Beta
Up / Down	For the uniform distribution the amount allowed to perturb down and up.Up is a percent (or value) added to the current parameter. Down is a percent (or value) subtracted from the nominal parameter value.

To create a library part with parameter distributions

You may want a library of parts with 5% precision normally distributed, or perhaps set the precision to an equation variable. To do this,

- Place the part on a schematic.
- Set the part parameter distributions.
- Right-click the part and select **Copy To Library** (and a library) to save it to your custom part library.
- When you place that saved library part, it will use your set distributions.

Distribution List

The distributions are as follows:

None Distribution

Distribution	This implies there is no distribution and the parameter will not perturb.

Uniform Distribution

Distribution	Flat distribution with flat (but maybe unequal) pdf on left and right of the mean.
Up / Down	For the uniform distribution the amount allowed to perturb down and up. Up is a percent (or value) added to the current parameter. Down is a percent (or value) subtracted from the nominal parameter value. In order to keep the mean equal to the parameter value the uniform distribution pdf is a step function with two steps (flat left and flat right of the mean). To have this be standard flat uniform, ensure that the Up/Down values are the same.
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether the up/down settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.

Normal Distribution

Distribution	A standard normal (Gaussian) distribution.
Standard Deviation	This can be in Percent (related to the parameter value) or a united value.
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.

Lognormal Distribution

Distribution	The standard lognormal distribution such that x = log(Normal distribution).
Standard Deviation	This can be in Percent (related to the parameter value) or a united value.
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.

Beta Distribution

Distribution	The standard beta distribution with pdf proportional to $x^{\text{Alpha-1}} * (1 - x)^{\text{Beta-1}}$
Alpha / Beta	For the Beta distribution, unit less values to set Alpha and Beta
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.

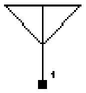
Discrete Distribution

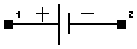
Distribution	This is the discrete equivalent of the uniform distribution
List of Values	Semicolon separated list of parameter values in the parameters unit of measure
Hard Limits	Absolute minimum and maximum values for the parameter. Leave these blank to have no minimum/maximum. A minimum value of 1e-10 or so will stop analyses from choking on negative part parameters.
Use Percentages	Determines whether settings are as a percent of the parameter value or absolute. The units shown on the dialog will change to match.

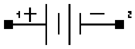
Symbol Reference

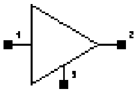
Commonly used Symbols

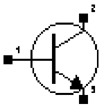
To use a symbol not otherwise available, place a part and change its symbol.

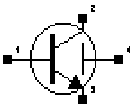

ANTENNA

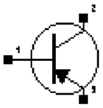

BATTERY

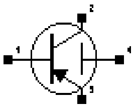

BATTERY_MULTI_CELL

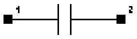

BLOCK

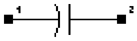

BIP_NPN

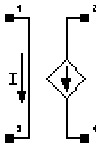

BIP_NPN4

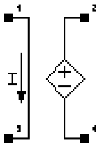

BIP_PNP

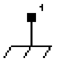

BIP_PNP4

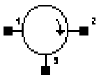

CAPACITOR

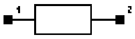

CAP_POLARIZED

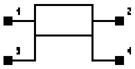

CCCS

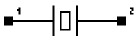

CCVS

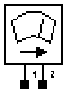

CHASSIS_GROUND

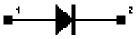

CIR3

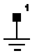

CLI

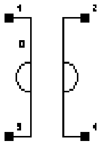

CLI_FOUR

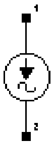

CRYSTAL

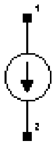

CURRENT_PROBE

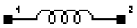

DIODE


GROUND

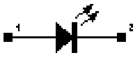

GYRATOR

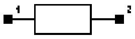

IAC

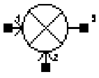

IDC

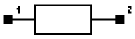

INDUCTOR

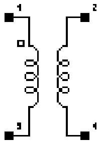
INPUT(1)

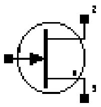

LED

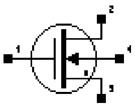

MCURVE

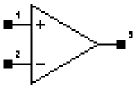

MIXER


MLI_SLI

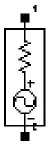

MUTUAL

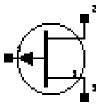

N_FET

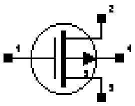

N_MOSFET

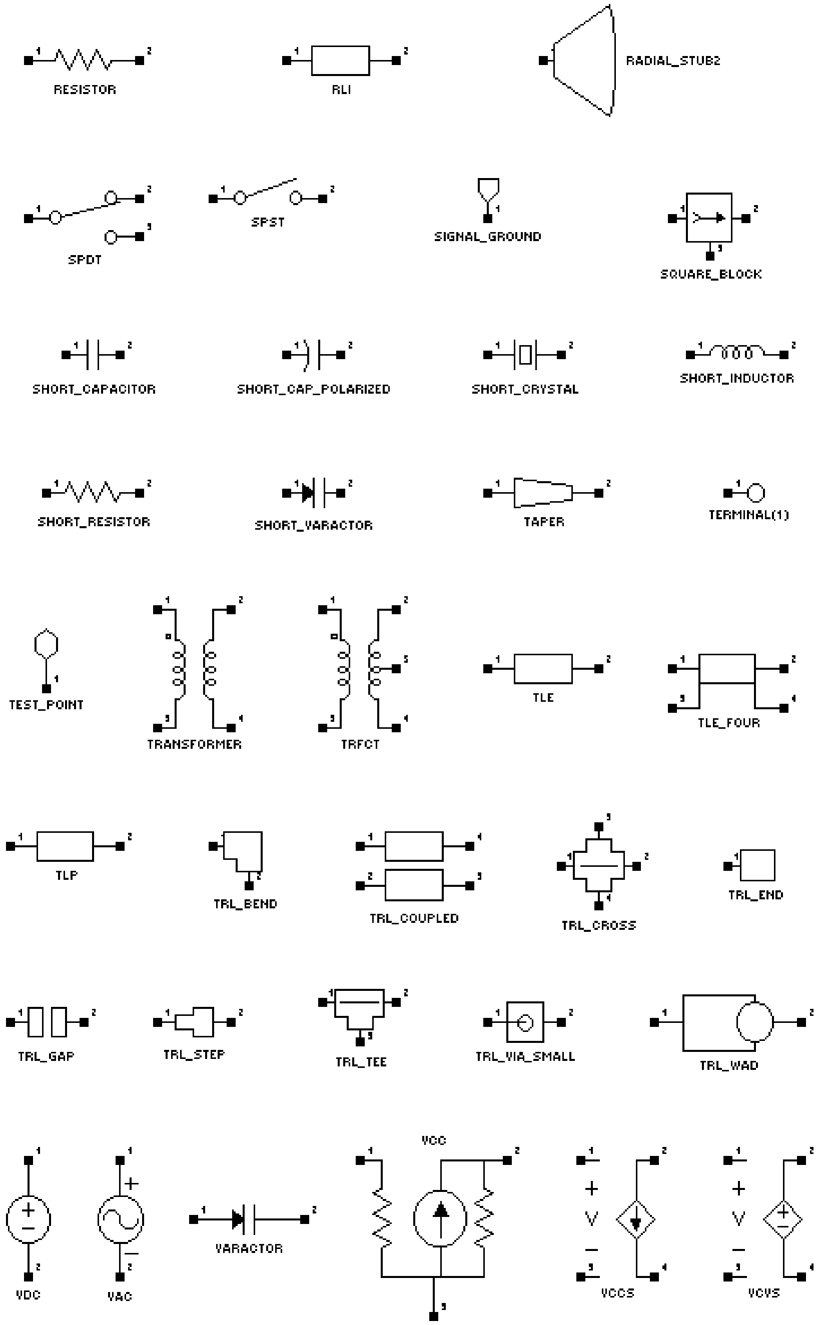

OP_AMP

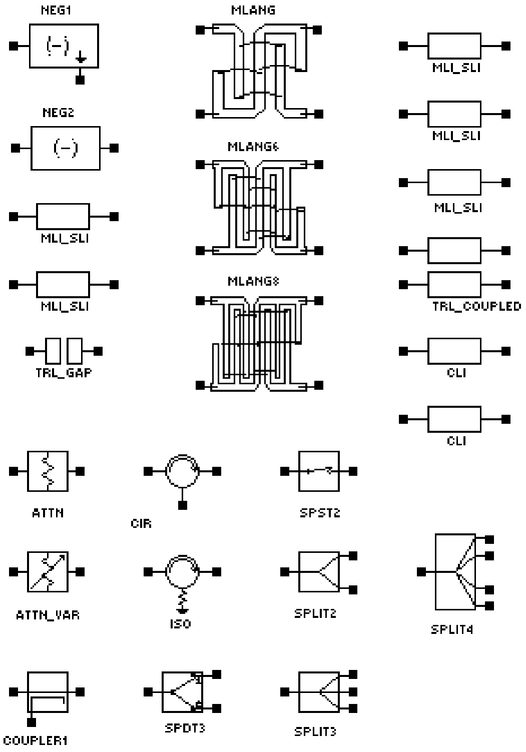
OUTPUT(2)

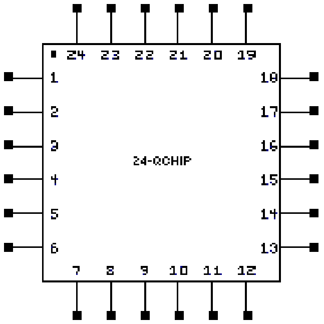

PAC


P_FET

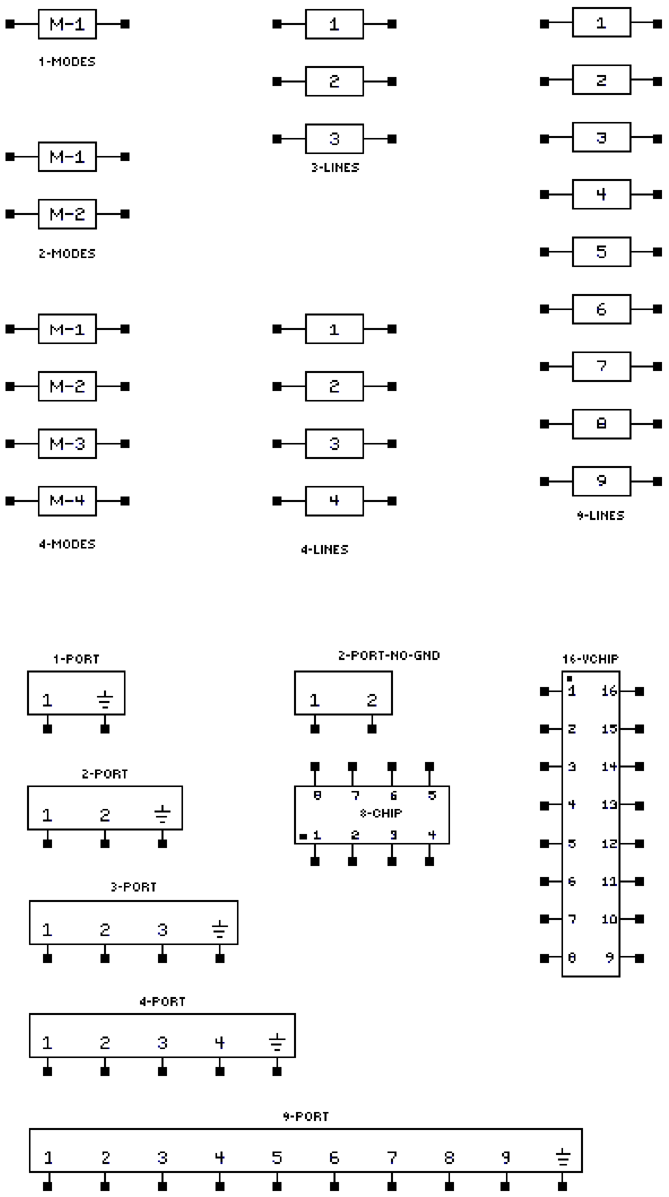

P_MOSFET


RESISTOR
RLI
RADIAL_STUB2
SPDT
SPST
SIGNAL_GROUND
SQUARE_BLOCK
SHORT_CAPACITOR
SHORT_CAP_POLARIZED
SHORT_CRYSTAL
SHORT_INDUCTOR
SHORT_RESISTOR
SHORT_VARACTOR
TAPER
TERMINAL(1)
TEST_POINT
TRANSFORMER
TRFCT
TLE
TLE_FOUR
TLP
TRL_BEND
TRL_COUPLED
TRL_CROSS
TRL_END
TRL_GAP
TRL_STEP
TRL_TEE
TRL_VIA_SMALL
TRL_WAD
VDC
VAC
VARACTOR
VCC
VCCS
VCVS



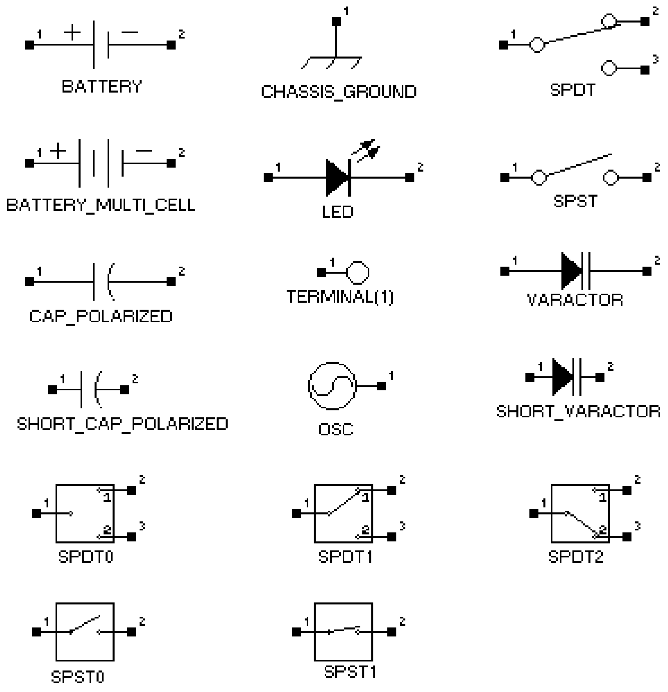


The following are a some representative samples of the plethora of internally generated parts that are available:



Extra Symbols

These extra symbols may be used by placing a similar part on the schematic and double-clicking the part to display the Parts dialog box. Then click the "Symbols..." button to select a custom symbol.

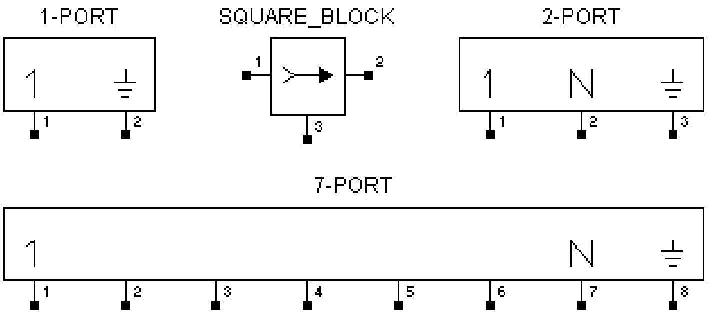


Usage:
The best way to use these symbols is to place a part with a similar shape and then change the visual symbol. For CAP_POLARIZED try starting with a CAPACITOR; for SPST try using a RESISTOR with R=.001 ohms. For LED start with a DIODE and for CHASSIS_GROUND use a true GROUND. MIXER and VARACTOR are best represented by a user model.

You can duplicate a custom symbol by selecting it and pressing Ctrl_D.

Net Block

This part allows a network or EM simulation to be reused. A NET block is an internally generated model that has many symbol variations based on the number of terminal connections. This symbol is available in the main *Schematic Toolbar* (users). Here are a few examples:



Netlist Syntax

In a netlist, simply use the name of the network followed by the node numbers

Parameters

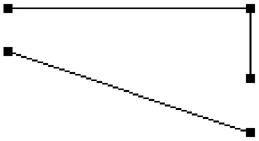
Network to Reuse The name of another design or EM simulation (in the current project)

Examples

NETLIST1 1 2 0

Standard wire connection

(*LINE)
The standard connection between schematic parts, which represents a "short circuit." This symbol is available in the main *Schematic Toolbar* (users) or by pressing the "W" key for 90 lines or Shift+W for a line at any angle. Once a line has been placed, you can drag an end point to move it. You can also move a line (or any component) by dragging the center and the lines will stay "connected" if Keep Connected is enabled. Holding the ALT key down will toggle the current keep connect setting.



Netlist Syntax

In a netlist, a wire connection is represented by a node number. All connections which use the same node number are directly connected by the same circuit trace.

Parameters

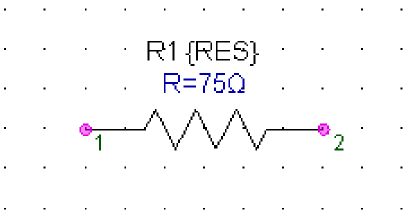
None

Ports, Connection Lines, and Nets

Contents

Part Ports (Terminals)

On a schematic, the Ideal Resistor part is depicted as follows.



Every part port is assigned a unique net name when placed unconnected on the schematic.

Connection Terminology

A **connection line** is a drawn line on the schematic that can be used to connect among ports and nodes of schematic parts. A **net** is a group of simple connection lines that share a unique **net name** and a common value at any instant.

Connection Line Net Labels

Connection lines by default have no net label, so they inherit the net of part terminals or other connection lines that they are connected to. If a connection line is given a Net Label, then that label becomes the net that the connection line resides on.

A net label could be a simple name or number which represents a single net.

To assign a net name:

1. Double-click on the connection line OR Right-click on the connection line and select **Net -> Edit Net Name...**
2. Enter a net name and click OK.

> If two connection lines share a common net from their Net Label, but they do not look visually connected, they are still connected for simulation purposes.

When an unconnected connection line is created, it does not have a net name. When the connection line is connected to net, it may gain a net name from the net. If the net has context and the net name is not set, an implicit net name is generated which may change with the schematic. This net name is an integer that is shown at the ends of the net. When a net name is explicitly assigned, the net name becomes persistent. While a persistent net name can be an integer, begin the net name with an alphabetic character.

Connection Lines and Ports

If no Net Labels are given to connection lines, they are automatically assigned nets in an intelligent manner. (Automatic assignment is the typical mode of operation- user intervention is not required)

Mapping Nets to Ports

The mapping from connection line nets to a particular part port (terminal) can be seen and modified by looking at the Terminal Mapping dialog, accessible by right-clicking on the terminal and selecting the *Edit Terminal Mapping* menu entry, if it exists. The menu entry will not exist if there is no ambiguity in the ordering of the net to part terminal mapping, as is the case when there is a single net connected to the part terminal.

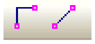
The Netlist for a part (all terminals) can be viewed in the Netlist tab of the Advanced properties of a part. See *Part Properties* (users) for details.

There is no net name or ordering ambiguity in a connection between a standard port and a simple connection line.

Connecting Parts in Genesys

Connection lines are used to connect part terminals and other connection lines. If a part terminal or connection line end is unconnected, the arrow tail or tip is marked with a pink dot. The pink dot disappears after a connection is made.

To draw a connection line:

1. Click one of the two **Draw Connection** buttons from the main toolbar: 
2. Click and hold the the start point on the schematic and drag the line to its end point on schematic.

OR

1. Hover over a part or connection line terminal and see the mouse cursor change into connection line mode. Click and drag the connection line.

Running Scripts

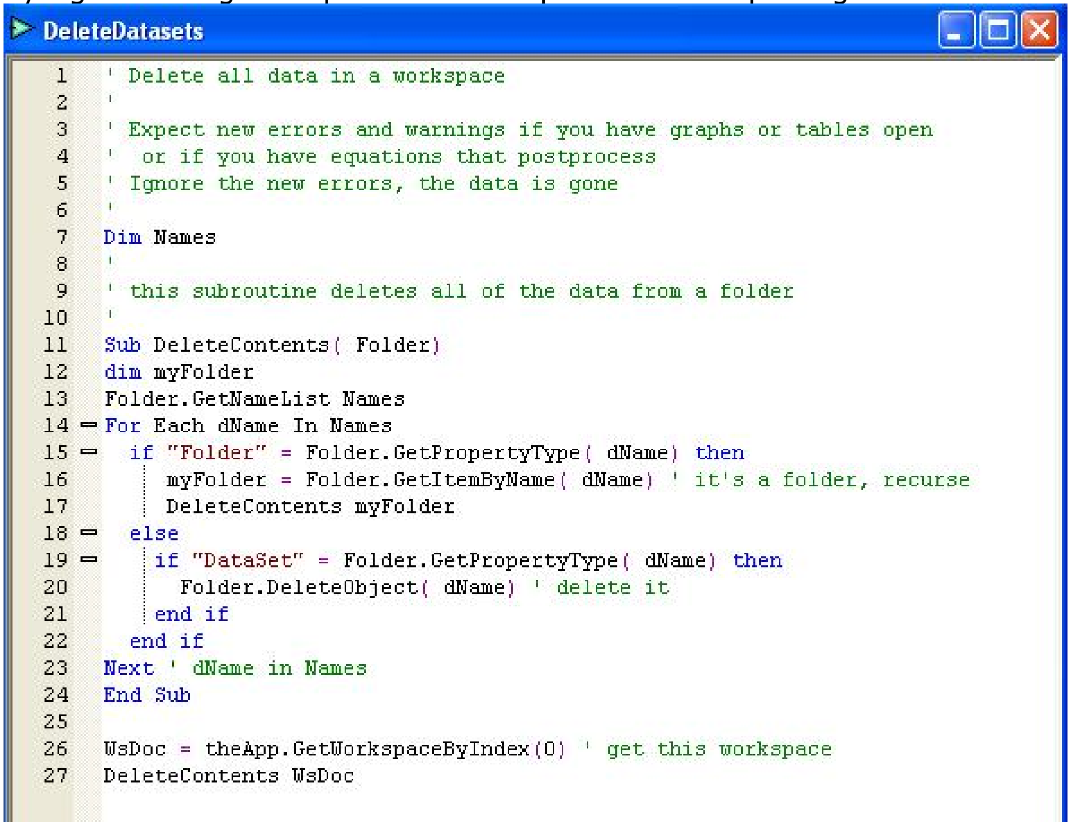
Scripts can be used to perform a variety of functions in Genesys. Some pre-written script have been included with Genesys and can be found in the Library Selector by setting the Library Type to "Scripts".

Contents

Add a Script

To add a script to Genesys:

- Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Script**.
- Once the script is added, edit script text in the window just like a Notes window.
- The toolbuttons at the top let you run the script or copy the script to the script processor (if you want to edit it there).
- You can also run a script by using an Annotation Button with an embedded script or by right-clicking a script in the workspace tree and picking Run.



```
' Delete all data in a workspace
'
' Expect new errors and warnings if you have graphs or tables open
'  or if you have equations that postprocess
' Ignore the new errors, the data is gone
'
Dim Names
'
' this subroutine deletes all of the data from a folder
'
Sub DeleteContents( Folder)
dim myFolder
Folder.GetNameList Names
For Each dName In Names
  if "Folder" = Folder.GetPropertyType( dName) then
     myFolder = Folder.GetItemByName( dName) ' it's a folder, recurse
     DeleteContents myFolder
  else
    if "DataSet" = Folder.GetPropertyType( dName) then
      Folder.DeleteObject( dName) ' delete it
    end if
  end if
Next ' dName in Names
End Sub

WsDoc = theApp.GetWorkspaceByIndex(0) ' get this workspace
DeleteContents WsDoc
```

Scripts have been color enhanced to improve their readability.

Creating Script Objects

In Genesys, all designs and their components are objects that you can refer to by name. In the following example, there is a design named Script Test and it has a substrate named FR4 in the Substrate folder.

Typically, most scripts start out by defining a variable to be the workspace object. In the example below the workspace object was defined by WsDoc = theApp.GetWorkspaceByIndex(0)

To create a sample script object:

1. Set the dielectric constant for your FR4 substrate to 12.
   WsDoc.[Script Test].Substrate.FR4.Er.Set(12)
2. Define an object pointing to the FR4 substrate.
   MySub=WsDoc.[Script Test].Substrate.FR4
3. Set the dielectric constant to 12.
   MySub.Er.Set(12)
4. Set the height to 14.
   MySub.Height.Set(14)

   > *Set* uses the parameter's defined unit of measure.

   > There is an object browser example using Visual Basic in the Genesys Examples\VBBrowser directory. This example shows you how to:
   > - Connect to Genesys from Visual Basic.
   > - Browse objects in Genesys.
   > - Execute any method in Genesys.
   >
   > See the doc on VBBrowser for more details.

Example: Running a Script from Microsoft Excel

Microsoft Excel has a VB Script engine that one can use to script other applications that support a COM interface. In the case of Genesys, this means that a Script can be written in Microsoft Excel that opens Genesys, does something such as load a workspace and run simulations, collects data, and processes the data. For information on accessing the VBScript development editor in Microsoft Excel, see your version of Excel's Help.

The global Windows name for Genesys's COM server is GENESYS. When Genesys runs, it registers itself with the Windows operating system by name so that a script can access it (including run an instance of it).

The first thing one must do to be able to access the Genesys COM server in Excel is to make it visible to Excel by setting it as a "Reference". In the Microsoft Visual Basic editor in Excel, you must declare "GENESYS" as a reference, and this is normally done by accessing the References dialog box via "Tools/References... "

> Note: GENESYS will only appear in the References list only if Genesys has been installed and run at least once.

Now, the Genesys COM server can be accessed in a VBScript module by the name "GENESYS". Create a new VBScript module by right-clicking on your VBA Project in the Project explorer and selecting "Insert... / Module".

The following code snippet shows the simplest possible script which simply opens an instance of Genesys:

```
Sub myScript()
```

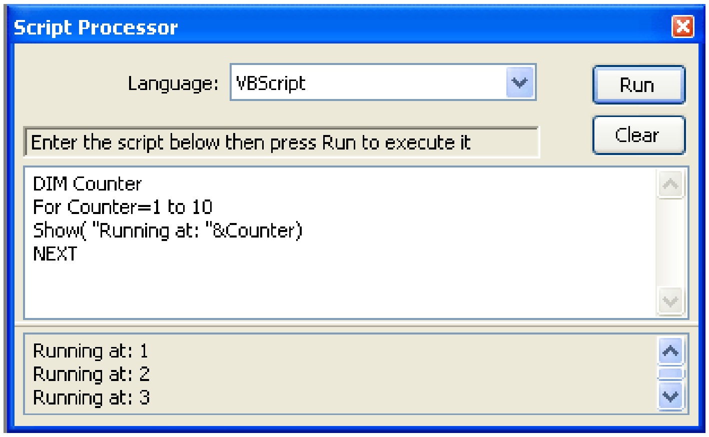
```
 Dim comServer As GENESYS.Application  ' Declare variable that references our COM server
 Set comServer = CreateObject("Genesys.Application")   ' Open an instance of the application
End Sub
```

For illustrative purposes, here is a more involved VBScript which opens Genesys, opens a workspace named "MyWorkspace.wsx", Runs a particular analysis that is located in the workspace, gets data from the dataset, and sets the data into an excel spreadsheet:

```
Sub myScript()
   Dim oGen As GENESYS.Application
   Dim WsDoc As GENESYS.Workspace
   Dim Stri As String

   ' Open Genesys
   Set oGen = CreateObject("Genesys.Application")
   ' Load workspace
   oGen.Manager.OpenWorkspace ("C:\Workspaces\MyWorkspace.wsx")
   ' Get Workspace
   Set WsDoc = oGen.Manager.GetWorkspaceByIndex(0)
   ' Run Analysis called Analysis1
   WsDoc.Designs.Analysis1.RunAnalysis
   ' Get V1 variable from Dataset named MyData
   arr = WsDoc.Designs.MyData.Eqns.VarBlock.V1.GetValue()
   ' Save the workspace
   oGen.Menu.File.Save.Execute
   ' Exit(optional)
   oGen.Menu.File.Exit.Execute
   Dim oXL As Excel.Application
   Dim oWB As Excel.Workbook
   Dim oSheet As Excel.Worksheet
   Dim oRng As Excel.Range
   Dim iNumQtrs As Integer

   Set oXL = Excel.Application  ' Activate Excel
   oXL.Visible = True
   ' Set active Workbook
   Set oWB = oXL.Workbooks.Application.ActiveWorkbook
   ' Set active Sheet
   Set oSheet = oWB.ActiveSheet

   ' output first 201 datapoints to excel
   For i = 0 To 200
       oSheet.Cells(i + 1, 1).Value = arr(i)
   Next i

   Exit Sub
Err_Handler:
     MsgBox Err.Description, vbCritical, "Error: " & Err.Number
End Sub
```

Script Processor

A script contains objects that let you control Genesys using industry-standard scripting languages. Scripts specifically control Genesys operations and are very different from equations, which relate variables in Genesys. Use scripts to load files, save files, save data sets, and change object parameters. Create and run scripts using the Script Processor window. Add the scripts to Genesys or to a specific design.



Language: VBScript

Enter the script below then press Run to execute it

```
DIM Counter
For Counter=1 to 10
Show( "Running at: "&Counter)
NEXT
```

```
Running at: 1
Running at: 2
Running at: 3
```

Genesys supports scripts written in both VBScript and JScript. These are standard programming languages not written by Agilent. Documentation for VBScript and JScript is widely available on the Web.

> Note: The latest version of scripting allows scripting from Visual Basic or C___, access to all Genesys menu items, customization of menus, and custom optimization. For more information on using any of these features, please contact Agilent directly or check the latest Help files at Agilent EEsof EDA Documentation
> .

To run a script:

1. Click **Tools** on the Genesys menu and select **Script Processor**.
2. Type or copy a script in the box.
3. Click the **Run** button.

To add a script to Genesys:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Script**.
2. Once the script is added, edit script text in the window just like a Notes window.

Script Verbs

Some Genesys objects have verbs you can use, including a few global verbs that are applicable to the program.

This table contains a list of all the available functions to use in scripts. The functions are organized by what type of object or item they can be called on. For example, functions in the Dataset table can be used off of datasets in your workspace. The VBBrowser is also very helpful in showing what functions can be used on what objects.

The examples were created using the Bridge-T.wsx as the opened workspace. Bridge-T.wsx can be found in the Examples folder of your Genesys directory. These examples work with the Bridge-T.wsx example workspace, but can be applied to any workspace of your choosing. If you want to see what is return you may need to use the "Show" function to see the result of the example. Any new file created from one of these function can be found in the same directory as Bridge-T.wsx.

Some sample scripts have been included with Genesys and can be used as a reference in writing your own. These scripts are located in the Library Selector under the Library Type "Script".

For all the examples below w = Application.Manager.GetWorkspaceByIndex(0). This sets the variable "w" to the current workspace.

All Main Genesys Objects

Syntax	Description	Example
ExportToLibrary(bstr LibName)	Export the object to a library.	w.ExportToLibrary("Test")
ImportFromLibrary(bstr LibType, bstr LibName, bstr PartName)	Import an object from the library.	w.ImportFromLibrary("Dataset", "Test", "myData")
GetLibrary(bstr LibType, bstr LibName)	Get the library specified by its type and its name. Use the Library Selector as a reference for the inputs.	result = w.GetLibrary("Design", "SymbolsQtr")
GetRegisteredModels	Gets a list of the registered models	w.GetRegisteredModels result
ImportFromLibrary(bstr LibType, bstr LibName, bstr PartName)	Import a part from a library. The parameters are the library type, the library name, and the name of the part. Use the Library Selector in Genesys as a reference to find all these inputs.	w.ImportFromLibrary "Design", "Symbols", "OSC"
OpenWindow	Open a view of the object.	w.Graphs.Graph1.OpenWindow()
CloseWindow	Close any open views.	w.Graphs.Graph1.CloseWindow()
SelectAll	Select all 'parts' in the object	w.Designs.BRIDGE_T.SelectAll
SelectNone	Deselect all 'parts' in the object	w.Designs.BRIDGE_T.SelectNone

Analysis

Syntax	Description	Example
ClearModelCache()	Clear the model cache from this analysis.	w.Design.Frequencies.ClearModelCache
GetDataName()	Get the name of the dataset. If an equation, this is parsed.	result =w.Design.Frequencies.GetDataName()
RunAnalysis()	Run this analysis.	w.Design.Frequencies.RunAnalysis()
SetDataName(bstr Name)	Set the dataset name the analysis will use.	w.Design.Frequencies.SetDataName("Dataset Name")

Application

Syntax	Description	Example
Application()	Returns the current version of Genesys.	Application()
Create(bstr Type, bstr Name)	Create a new object with the specified type and name.	result = Create("Notes", "ThisNote")
FileNewFromTemplate(bstr FileName)	Open a template from the template directory.	FileNewFromTemplate("Default")
FileOpen(bstr FileName)	Open a file from the last opened directory.	FileOpen("Bridge-T")
FileOpenExample(bstr FileName)	Open an example from the last opened directory.	FileOpenExample("Bridge-T")
FileOpenRecent(int FileNumber)	Open a recent file. 1 represents the most recent file.	Application.Manager.FileOpenRecent(1)
GetToolbarSet()	Gets the toolbar set as an object.	result = Application.Manager.GetToolbarSet()
GetWorkspaceByIndex(int iNum)	Returns the workspace object at index iNum	w=theApp.GetWorkspaceByIndex(0)

GetWorkspaceCount()	Returns the number of workspaces opened the instance of Genesys	result =Application.Manager.GetWorkspaceCount()
OpenWorkspace(bstr strFile)	Loads the specified workspace without a open window prompt	OpenWorkspace("Bridge-T.wsx")
PostCommand(bstr CmdMsg)	Display a command message in the current view.	Application.Manager.PostCommand("fit_windows")
SaveTextToFile(bstr FileName, bstr ToShow)	Save text to a file.	SaveTextToFile "File.txt", "HelloWorld"
SetActivePart(bstr PartName, int ID)	Set up part so that the next click in the Schematic places a new copy of the specified part.	none
SetNetworkReuse(int iPorts)	Displays dialog to re use a design as a part with the specified number of ports. A part of iPort number of ports is created representing the design you select.	Application.Manager.SetNetworkReuse 2
Show(bstr ToShow)	Display the text in the Status box below the Edit box of a window.	Show("HelloWorld")
ShowContextMenu(bstr MenuName)	Display the menu.	none
ShowToolBar(bstr ToolBarName, int Show)	Toggle, show, or hide a toolbar by name. 0 = Off, 1 = On	Application.Manager.ShowToolBar "Schematic", 1
Update()	Runs all pending analyses.	Application.Manager.Update()
ViewDesignSelector(int Flags)	Toggle (0), show (1), or hide (2) the Library Selector.	Application.Manager.ViewDesignSelector(0)
ViewPartPicker(int Flags)	Toggle (0), show (1), or hide (2) Part Selector A.	Application.Manager.ViewPartPicker(0)
ViewPartPickerB(int Flags)	Toggle (0), show (1), or hide (2) Part Selector B.	Application.Manager.ViewPartPickerB(0)
ViewSimulationStatus(int Flags)	Toggle (0), show (1), or hide (2) the Simulation Status window.	Application.Manager.ViewSimulationStatus(0)
ViewTuneWindow(int Flags)	Toggle (0), show (1), or hide (2) the Tune window.	Application.Manager.ViewTuneWindow(0)
ViewWorkspaceWindow(int Flags)	Toggle (0), show (1), or hide (2) the Workspace window.	Application.Manager.ViewWorkspaceWindow(0)

Atom

Syntax	**Description**	**Example**
ExportXML(bstr Path)	Save an object's XML stream to file. Path needs file extension.	w.Notes.ExportXML "test.xml"
GetName()	Get the external name of an object.	result = w.GetName
FromXML(bstr XMLStream)	Convert an XML stream into an object.	none
ToString()	Convert an object into a string (text) representation.	result =w.Notes.ToString
ToXML()	Convert an object into XML.	result =w.Notes.ToXML
SetName(bstr bsName)	Set the object name	w.Notes.SetName "HelloName"

Dataset

Syntax	Description	Example
ExportS(bstr FileName)	Export data as an S-parameter data file.	w.Design.Frequencies_Data.ExportS("Bridge-T.s2p")
SnapShotToData	Creates a snapshot of a dataset or equation. This can be used to make a checkpoint dataset.	w.Design.Frequencies_Data.Eqns.SnapShotToData("New_Dataset")
DeleteAnalysisVars	Delete all calculated-by-analysis variables from a dataset	w.Design.Frequencies_Data.DeleteAnalysisVars

Folder

Syntax	Description	Example
ConvertToDesign(bstr SchematicName)	Convert a schematic to a design.	none
DeleteObject(bstr ObjectName)	Delete a Genesys object.	w.Design.DeleteObject("Frequencies_Data")
GetNameList(variant* ItemList)	Get the list of names in the folder. List is a collection of names.	w.Graphs.GetNameList result
GetObjectCount()	Count the number of sub objects in the folder.	result = w.Design.GetObjectCount()
GetObjectList(variant* ItemList)	Get the list of objects (as pointer).	w.GetObjectList result
GetObjectType(bstr ObjectName)	Gets the type of an object called ObjectName inside a folder.	w.Graphs.GetObjectType("Graph1")
LoadSParameters(bstr DataSet, bstr FileName, int Ports, bool HotLink)	Load S-parameter to a design.	none

Item

Syntax	Description	Example
AddProperty(IDispatch* Property)	Insert this property. Input must be an item.	result = w.GetItemByName("Notes") w.Designs.AddProperty(result) 'adds the note to the designs folder
DeleteProperty(bstr Property)	Delete this property.	w.DeleteProperty("Notes")
GetItemByIndex(int Index)	Get items by index starting with index 0.	result = w.GetItemByIndex(1)
GetItemByName(bstr ItemName)	Get item by name.	result = w.GetItemByName("Graphs")
GetItemCount()	Count the number of items.	result = w.GetItemCount()
GetMethodList()	Get the list of methods from the GDISP entries.	result = w.GetMethodList()

GetParentOfItem(IDispatch* Child)	Get the parent item of given item.	child = w.Design.Frequencies.DataName
		result = w.GetParentOfItem(child)

GetPropertyList(variant* ItemList)	Get the property list of an item.	w.Design.BRIDGE_T.PartList.GetPropertyList result
GetPropertyType(bstr PropName)	Get property type by name.	result=w.Design.GetPropertyType("Frequencies")
GetType()	Gets the type of an item.	result = w.Graphs.Table1.GetType
GetPropertyAsArray(bstr name, variant* ItemList)	Gets the contents of a property as a list	w.GetPropertyAsArray "Notes", result
GetVarCount()	Count the number of variables.	result = w.Design.BRIDGE_T.Schematic.Width.GetVarCount()
GetVarName(int Index)	Get the variable name at given index.	result = w.Design.BRIDGE_T.Schematic.Width.GetVarName(2)
GetVarType(int Index)	Get variable type at given index.	result = w.Design.BRIDGE_T.Schematic.Width.GetVarType(2)
GetVarValue(int Index)	Get the variable value at a given index.	result = w.Design.BRIDGE_T.Schematic.Width.GetVarValue(2)
GetVarXMLName(int Index)	Get the XML name of a variable at a given index.	result = w.Design.BRIDGE_T.Schematic.Width.GetVarXMLName(2)

HasProperty(bstr PropName)	Returns -1 is the Item has the property and 0 if it does not	if w.HasProperty("IsOpen") then
		Show "yes" 'is the workspace open?
		end if

SetProperty(bstr Property, variant* Value)	Set a property to a value.	number ="74" w.Design.BRIDGE_T.PartList.C1.ParamSet.C.SetProperty "DataEntry", number

Library

Syntax	Description	Example
GetPartList(variant* ItemList)	Get the part list of a library.	dim Symbols Library = w.GetLibrary("Design", "SymbolsQtr") Library.GetPartList Symbols For each part in Symbols Show part next

Menu

Syntax	Description	Example
Execute()	Execute a menu entry.	Application.Menu.File.New.Execute()
InsertItem(int Pos, bstr Text, bstr Name, bstr Script)	Insert a menu item inside any menu. The action of this new menu item is based on the script passed in.	Application.Menu.Run.InsertItem 0, "Open", "Open", "Application.Manager.OpenWorkspace(""Bridge-T.wsx"")"
InsertMenu(int iPosition,bstr bsText,bstr bsName)	Insert a menu tab on the top of the window.	Application.Menu.InsertMenu 8, "Run Script", "Run Script"
InsertSeparator(int Pos)	Insert a separator (bar). 0 is the initial position.	Application.Menu.Tools.InsertSeparator(2)

Parameters

Syntax	Description	Example
Get()	Get the parameter entry (what the user typed).	result = w.Design.Frequencies_Data.Eqns.VarBlock.F.Get
GetData	Get the formatted value of data.	bsResult = w.Design.Frequencies_Data.Eqns.VarBlock.F.GetData
GetValue	Get the value of the data. This will always be in MKS for united Parameters.	bsResult = w.Design.Frequencies_Data.Eqns.VarBlock.F.GetValue
Set(bstr NewValue)	Set the parameter entry (as if you typed it).	w.Design.Frequencies_Data.Eqns.VarBlock.CS.Set("12e6")
SetValue(variable)	Set the value of the data to the variable value.	a=12 w.Design.Frequencies_Data.Eqns.VarBlock.CS.SetValue (a)

Part

Syntax	Description	Example
ChangeModel(bstr strName)	Change the Model of a part.	w.Design.BRIDGE_T.PartList.R1.ChangeModel("CAP")
ChangeSymbol(bstr strName)	Change the Symbol of a part.	w.Design.BRIDGE_T.PartList.R1.ChangeSymbol("CAPACITOR")
SetCustomValue(bstr ParamName, variant* piParamValue, bstr bsUnit, bstr bsValidate, bool vbShow)	Adds a custom value to a part, which will appear in the custom tab of the part properties.	dim val val = 35 w.Design.BRIDGE_T.PartList.C1.SetCustomValue "C", val, "pF", "Error", 1

Schematic

Syntax	Description	Example
AddAnnotationBox(bstr bsTag, bstr bsText)	Adds a text box named bsTag containing the text bsText to a schematic.	w.Design.BRIDGE_T.AddAnnotationBox "Hello", "World"
ExportIFF(bstr FileName)	Export to ADS/IFF format.	w.Design.BRIDGE_T.ExportIFF "IFF.iff"
GetIntent()	Get intent of the design. Integer value returned.	
result = w.Design.BRIDGE_T.GetIntent()		

Equations

Syntax	Description	Example
SnapShotToData	Convert the equation variable values into a fixed dataset.	w.Equations.SnapShotToData
Calculate	Calculate an equation set. Designed for non-auto-calc equations. This is useful for running communications, for example.	result = w.Equations.Calculate

Script

Syntax	Description	Example
RunScript(int Language)	Executes a script in the specified language. 0==VBScript, 1==JScript	w.Script1.RunScript(0)

Workspace

Syntax	Description	Example
ClearCompareLog()	Clears the Run and Compare error log.	w.ClearCompareLog()
ClearErrorLog()	Clears the error log at the bottom of the window.	w.ClearErrorLog()
GetCompareLog()	Gets the compare as generated by the RunAndCompare function.	SaveTextToFile "test.txt",w.GetCompareLog
DeleteOldDatasets()	Deletes all but the most recent dataset. Works if you have used the RunAndCompare function.	w.DeleteOldDatasets
ImportSpiceFile(bstr bsFileName, bstr bsLibName, int iReverseNodes)	Import a SPICE file into a workspace and, optionally, into a model library. If LibraryName is blank, you are prompted for a library name.	none
IsAnalysisDone()	Returns 1 if the analysis is done and 0 if it is not.	result = w.IsAnalysisDone()
RunAndCompare(int numErrors, double dTolerance)	Runs and creates a new dataset for each analysis. Compares the two datasets on a tolerance of dTolerance and reports errors stopping after numErrors errors have been found.	w.RunAndCompare 2, 0.02
SaveErrorLog(bstr FileName)	Save the error log into a file.	w.SaveErrorLog("ErrorLog.txt")
Save()	Save a workspace.	w.Save()
SaveAs(bstr FileName)	Save a workspace with new name.	w.SaveAs("name.wsx")

Using Scripts in Programs

Supported Languages: C#, C++, Visual Basic.

A program can be written in any one of the supported languages to communicate with Genesys using our COM interface. Scripts and commands can be executed in the Genesys

Script Processor from your program. Your program needs to contain the proper COM reference and include the proper header for our COM Interface.

The VBBroswer is an example of a program that communicates to Genesys through the COM interface. The source code for the VBBrowser is located in Examples\Scripting\VBBrowser\MainForm.vb for your viewing.

Using the COM Interface for Genesys

1. Add Interop.GENESYS.dll as a COM Reference to your project. Interop.GENESYS.dll is found under Examples\Scripting\VBBrowser in your Genesys directory.
2. Import, Use, or Include GENESYS as a header in your program depending on what language you are using.
3. Create an Instance of the GENESYS.Application

Running Scripts from COM Interface

A script can be run from either the RunScript function or the RunScriptFromFile function.

RunScript Function

To use the RunScript function the context of the script you wish to run must be contained in a string variable. The Script Processor works line by line, so the string variable will need to contain a line return character after each line in your script.

> For Example, in VB this is one way you could format a string variable strScript to contain a script that opens a workspace and runs an analysis.
>
> ```
> strScript = "OpenWorkspace("C:\Program Files\GENESYS2008.07\Examples\Bridge-T.wsx")"
> strScript = strScript & vbCrLf & "WsDoc = theApp.GetWorkspaceByIndex(0)"
> strScript = strScript & vbCrLf & "WsDoc.Design.Frequencies.RunAnalysis()"
> ```

Once you have formulated a string containing the script that you want to execute within Genesys, then use the command RunScript to send the script through Genesys to the script processor. For example, if the GENESYS.Application was instantiated as GenesysApp and the string containing the script was called strScript:

For VB script
GenesysApp.RunScript(strScript, ScriptLanguage.genLangVBScript).

For J Script
GenesysApp.RunScript(strScript, ScriptLanguage.genLangJScript).

RunScriptFromFile Function

An easier method for running a script in Genesys from your program is to use the RunScriptFromFile function which runs a script from a text file. Simply copy the cotents of a script in Genesys to a text file and save the file.

> For example, a text file name MyScript.txt contains:
>
> ```
> OpenWorkspace("C:\Program Files\GENESYS2008.07\Examples\Bridge-T.wsx")
> WsDoc = theApp.GetWorkspaceByIndex(0)
> WsDoc.Design.Frequencies.RunAnalysis()
> ```

Use the RunScriptFromFile to load the script text file and execute the script. For example, if the GENESYS.Application was instantiated as GenesysApp and the string containing the path to MyScript.txt was called strPath:

For VB script
GenesysApp.RunScriptFromFile(strPath, ScriptLanguage.genLangVBScript).

For J Script
GenesysApp.RunScriptFromFile(strPath, ScriptLanguage.genLangJScript).

VBBrowser

(Genesys Browser)

The VBBrowser is used to browse objects in Genesys. This is an interactive program that allows a user to see what functions are available to call within the script processor. The program communicates with one active instance of the Genesys program. The browser looks at the current workspace and retrieves objects and items from it.

Running the VBBrowser

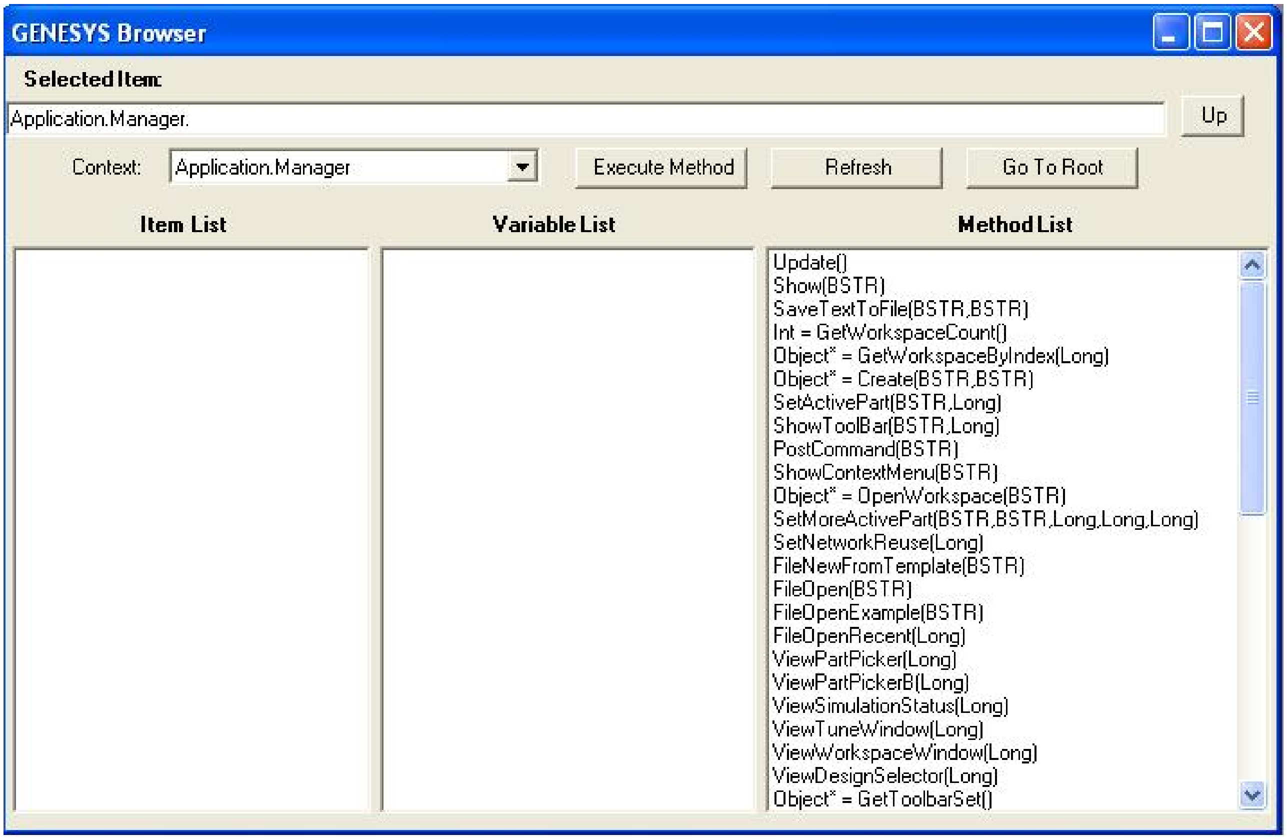
The VBBrowser is located in the Examples\Scripting\VBBrowser folder of your Genesys directory. Source code for the VBBrowser can be found in this same folder in the file called MainForm.vb. The files Interop.GENESYS.dll and VBBrowser.exe were created following the instructions found in the ReadMe.txt located in the same folder.

There are two ways to launch the VBBrowser

1. Run the VBBrowser while you have a Genesys Running.
2. Launch the VBBrower without Genesys Running. The VBBrowser will launch as well as Genesys.

> ℹ If you load another workspace in Genesys while the VBBrowser is running it is best to click the Go To Root button to avoid errors. Clicking the Refresh or Up button will throw an error and then load the root.

Contents of the VBBrowser



Selected Item: Application.Manager.

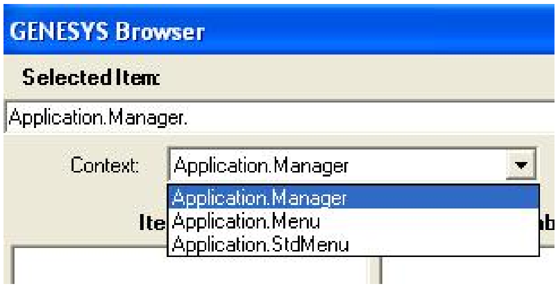
Context: Application.Manager

```
Update()
Show(BSTR)
SaveTextToFile(BSTR,BSTR)
Int = GetWorkspaceCount()
Object* = GetWorkspaceByIndex(Long)
Object* = Create(BSTR,BSTR)
SetActivePart(BSTR,Long)
ShowToolBar(BSTR,Long)
PostCommand(BSTR)
ShowContextMenu(BSTR)
Object* = OpenWorkspace(BSTR)
SetMoreActivePart(BSTR,BSTR,Long,Long,Long)
SetNetworkReuse(Long)
FileNewFromTemplate(BSTR)
FileOpen(BSTR)
FileOpenExample(BSTR)
FileOpenRecent(Long)
ViewPartPicker(Long)
ViewPartPickerB(Long)
ViewSimulationStatus(Long)
ViewTuneWindow(Long)
ViewWorkspaceWindow(Long)
ViewDesignSelector(Long)
Object* = GetToolbarSet()
```

General

The Selected Item box contains the syntax for the script that you can execute by clicking the Execute Method button.

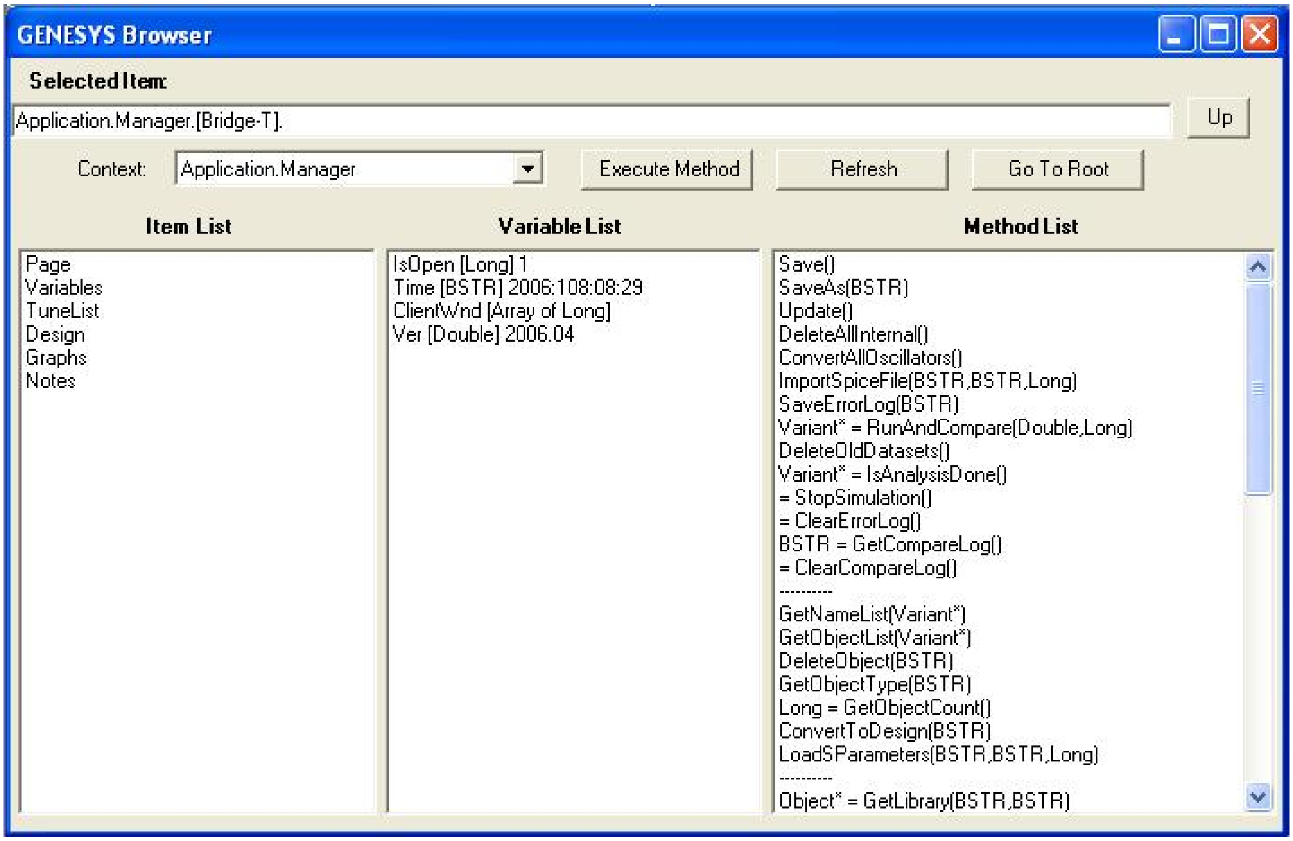
The Context drop box contains three items



Selected Item: Application.Manager.

1. Application.Manager (default) Sets the Item List to the context of the workspace tree.
2. Application.Menu Sets the Item List to the context of the current Menu Bar in Genesys
3. Application.StdMenu - Sets the Item List to the context of the standard Menu Bar in Genesys

Lists



Selected Item: Application.Manager.[Bridge-T].

Context: Application.Manager

Item List:

Page
Variables
TuneList
Design
Graphs
Notes

Variable List:

IsOpen [Long] 1
Time [BSTR] 2006:108:08:29
ClientWnd [Array of Long]
Ver [Double] 2006.04

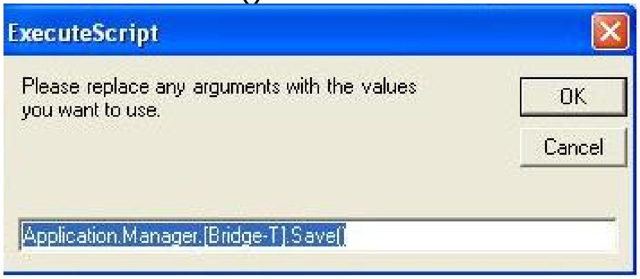
Method List:

```
Save()
SaveAs(BSTR)
Update()
DeleteAllInternal()
ConvertAllOscillators()
ImportSpiceFile(BSTR,BSTR,Long)
SaveErrorLog(BSTR)
Variant* = RunAndCompare(Double,Long)
DeleteOldDatasets()
Variant* = IsAnalysisDone()
= StopSimulation()
= ClearErrorLog()
BSTR = GetCompareLog()
= ClearCompareLog()
----------
GetNameList(Variant*)
GetObjectList(Variant*)
DeleteObject(BSTR)
GetObjectType(BSTR)
Long = GetObjectCount()
ConvertToDesign(BSTR)
LoadSParameters(BSTR,BSTR,Long)
----------
Object* = GetLibrary(BSTR,BSTR)
```

Item List - The window contains a list of all the items found in the current context. If nothing appears in the window you can click the Refresh button to refresh the context. Clicking on an item in this list will show you a list of sub items. Note that the sub items correspond to the items inside the opened workspace. Notice that as you click items, the text in the selected item box changes. The first thing you should see (in the default context) in the Item list is the name of the workspace(s) that are loaded in Genesys. In the example above you would see Bridge-T as the first item in the list.

Variable List - The window contains a list of properties, variables, or parameters that are associated with the current item. Items in this list can be called as a property to an item.

Method List - The window contains a list of the methods that can be used with the current item. Notice that by double clicking on a method the ExecuteScript window pops up with current syntax of the method youve selected. This syntax is generated from the Selected Item text box and the method you have clicked. This is what would pop up if you double clicked the Save() method.

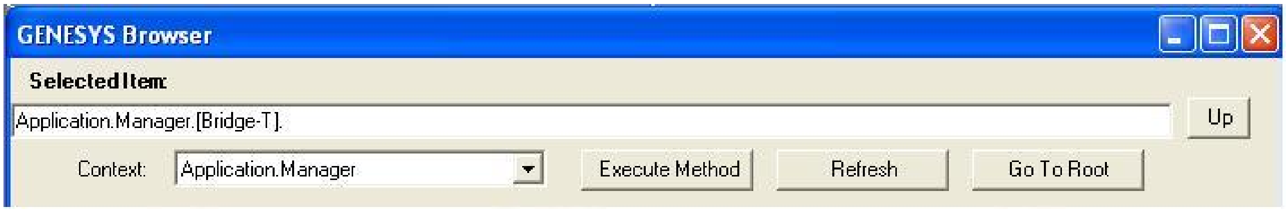


Please replace any arguments with the values you want to use.

Application.Manager.[Bridge-T].Save()

You can execute this one line script by clicking on OK. A script processor window will not pop up in Genesys, so you may not always know if it worked or not. If you need to execute many lines it is suggested to use a script. The ExecuteScript window is best used as a guide to get the correct syntax for writing your own script.

Buttons



Up - The button sets the Item List to the parent item of the current Item List window

Execute Method The button will bring up the ExecuteScript window that shows the syntax for the current Selected Item and gives the option to run it or not.

Refresh - The button reloads the items in the three lists.

Go To Root - The button sets the Item List to the top most parent.

Schematics

This section describes how to create and use a schematic. A schematic is a graphical way of describing a network of parts connected together through schematic symbols. These parts also contain models that are simulated. The schematic symbols and wiring show the connectivity between models.

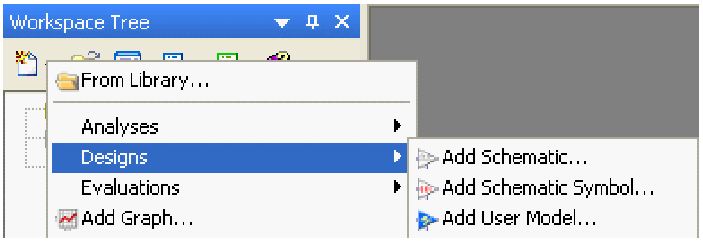
Contents

Creating a Simple Schematic

There are two different ways to to create a design in Genesys. One is the by clicking on the New Item button ( ) on the Workspace Tree toolbar or by right clicking on a folder in the workspace tree.

Method 1 - **Clicking on the New Item Button**

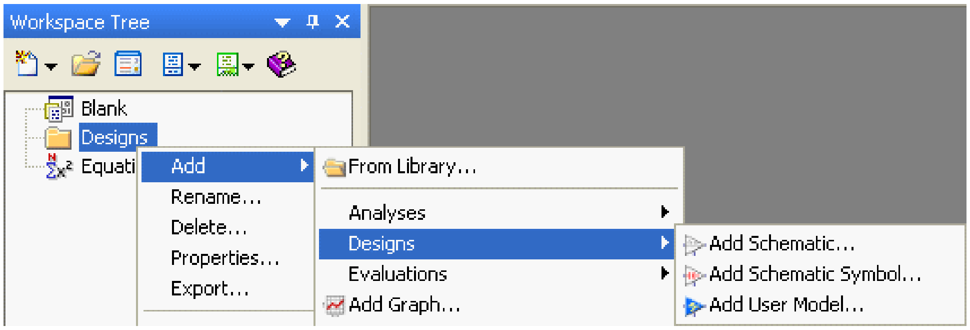
1. Click the New Item button ( ) on the Workspace Tree toolbar
2. Select the **Designs >** submenu
3. Now select **Add Schematic...**



Or

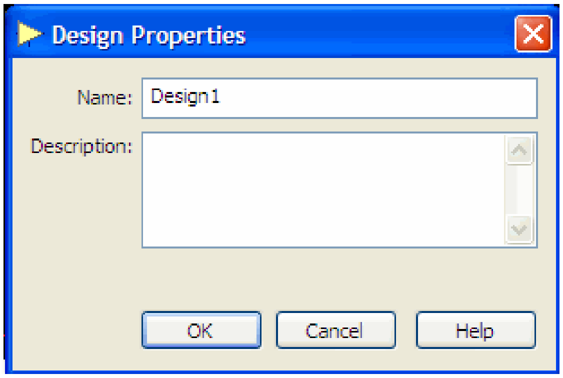
Method 2 - **Right Clicking on a Workspace Folder**

1. Right click on a folder in the workspace tree to bring up the right click menu.
2. Select the **Add >** submenu.
3. Select the **Designs >** submenu
4. Now select **Add Schematic...**



The name of the schematic can then be entered along with an optional description.

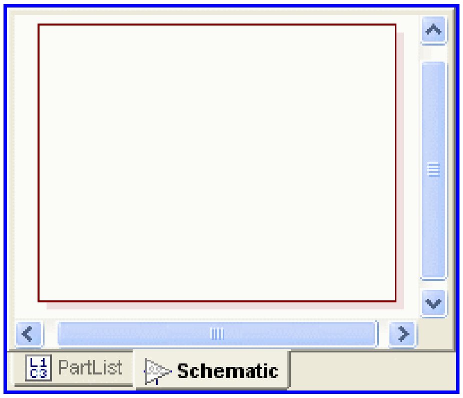
Note: The schematic will be added under the folder that was last selected in the workspace tree. (ie. whatever is the "current" folder.) If you want to move it to a different directory simply drag and drop it in the new folder.


Design Properties

Name: Design1

Description:

A blank schematic will appear.



Placing Parts on a Schematic

Only parts and annotations can be placed on a schematic. Annotation objects are things like text, title blocks, and other drawing objects like polygons, and rectangles. Only parts are connected together through wires or buses.

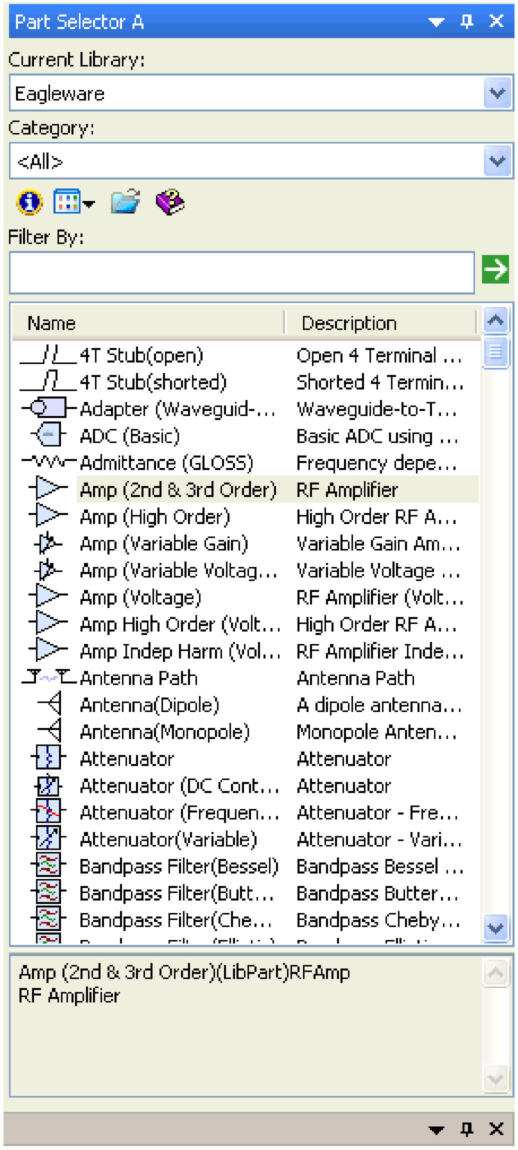
Parts can be placed on the schematic in either of two ways. Through a **part selector** or through **part toolbars**.

Hint
Some parts can be placed with keyboard short cuts. See *Appendix A Keystroke Commands* (users) for more information.

Hint
Some parts can be placed by dragging them from the workspace tree to the schematic. When a schematic or sub-network model is dragged in this way a sub-network part is created with an auto-generated schematic symbol. S-parameter datasets can also be placed on the schematic by dragging them.

Method 1 - **Part Selector**

1. Bring up the part selector by clicking on the **Part Selector** button ( ) on the **Schematic** toolbar.



Part Selector A

Current Library: Eagleware

Category: <All>

Filter By:

Name	Description
4T Stub(open)	Open 4 Terminal ...
4T Stub(shorted)	Shorted 4 Termin...
Adapter (Waveguid-...	Waveguide-to-T...
ADC (Basic)	Basic ADC using ...
Admittance (GLOSS)	Frequency depe...
Amp (2nd & 3rd Order)	RF Amplifier
Amp (High Order)	High Order RF A...
Amp (Variable Gain)	Variable Gain Am...
Amp (Variable Voltag...	Variable Voltage ...
Amp (Voltage)	RF Amplifier (Volt...
Amp High Order (Volt...	High Order RF A...
Amp Indep Harm (Vol...	RF Amplifier Inde...
Antenna Path	Antenna Path
Antenna(Dipole)	A dipole antenna...
Antenna(Monopole)	Monopole Anten...
Attenuator	Attenuator
Attenuator (DC Cont...	Attenuator
Attenuator (Frequen...	Attenuator - Fre...
Attenuator(Variable)	Attenuator - Vari...
Bandpass Filter(Bessel)	Bandpass Bessel ...
Bandpass Filter(Butt...	Bandpass Butter...
Bandpass Filter(Che...	Bandpass Cheby...

Amp (2nd & 3rd Order)(LibPart)RFAmp
RF Amplifier

2. **Click** on a part.
3. **Move** the mouse over the schematic. The cursor will change to a plus sign when placed over the schematic.
4. **Click** the schematic where the part is to be placed.

Method 2 - **Part Toolbars**

Changing Part Orientation

The user can change the part orientation using keystrokes, a part right click menu, and the Main menu.

Hint
When placing the part on the schematic with the mouse the direction of its travel on the left mouse click will determine the initial orientation of the part. For example, if the mouse were being dragged slightly from left to right when the left mouse button is clicked the part would be oriented from left to right on the schematic.

To change the part orientation:

1. **Select** the part by clicking on it. A red selection rectangle will appear around the part.
2. Press **F3** to rotate the part clockwise, **Shift + F3** for counter clockwise, and **F6** for a mirror.
3. Repeat step 2 as necessary.

Manipulating Parts

Connecting Parts

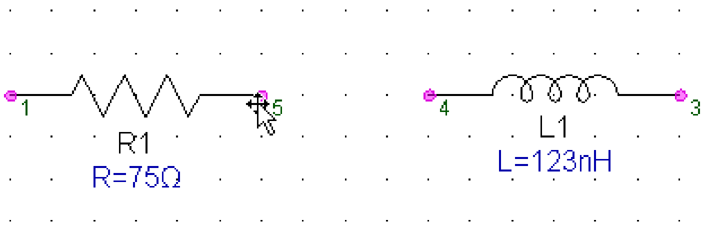
There are two methods that can be used to connect parts together.

Method 1- **Wire Toolbar Buttons:**

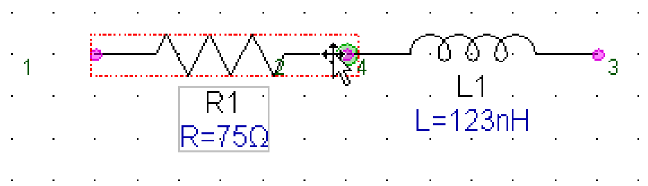
1. **Click** on the **right angle** ( ) or **angled** ( ) toolbar buttons contained on the schematic toolbar.
2. **Click** and drag to draw the wire on the schematic.

Method 2- **Dragging the Part Terminal:**

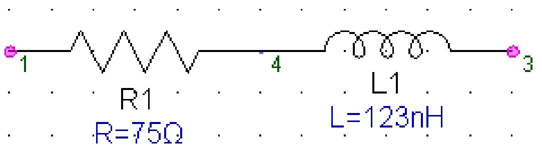
1. **Click** the part terminal to be connected. A connection highlight dot (green circle) will appear on the terminal nearest the mouse cursor. This marks the terminal which will snap to the grid and to other connection nodes.



2. **Drag** the terminal over the terminal of the part to be connected.



3. **Move** the part to the desired location.



Moving Parts

There is a global option to *keep parts connected* (users) when they are moved or they will be unconnected when they move. The **Alt** key toggles this behavior.

+To move a part:

1. **Click** the part to select it.

   > **Hint**
   > A small green circle appears on the terminal closest to the mouse. This is the reference terminal for part alignment, connections, and snap points.

2. **Drag** the part to the location of interest.

> **Hint**
> Multiple parts can be selected and manipulated at the same time. Click and drag a selection rectangle around the parts of interest in a schematic. Holding down the **Ctrl** key during part selection will select / de-select parts from the group.

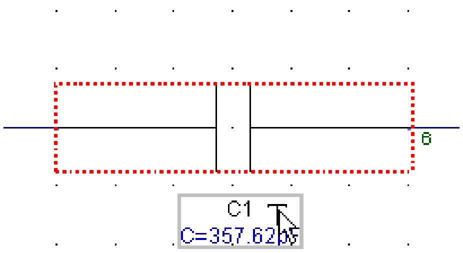
> **Advanced Tip**
> Net names (node numbers) may be re-assigned during a move; when that happens, the first priority is to retain the existing nets attached to ports. Parts which are stationary have the the next highest priority and parts which moved have the lowest priority. This means that if you would like to retain certain existing net/node names (perhaps because you are referencing them in graph measurements), make sure you move the *other* parts of the schamatic, instead of the area you care about.

Moving Part Text

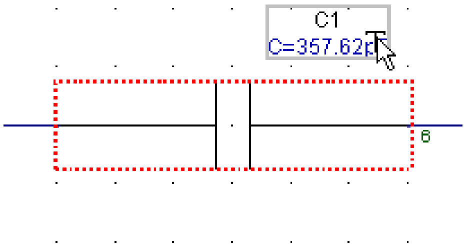
There are three ways to the move the part text.

Method 1- **Drag to Location:**

1. **Click** the part to select it.
2. **Move** the mouse over the part text selection rectangle. The cursor will change to a **T** and a pointer indicating text will be moved on a mouse drag.



3. **Drag** the text block to a new location.



Method 2- **Keyboard Shortcut:**

1. **Click** the part to select it.
2. **Press F4** to rotate through standard text locations of: Top, Bottom, Left, Right, and Center.

Method 3- **Right Mouse Menu:**

1. **Right Click** the part.
2. Select the **Text** menu.
3. Select the desired text location.

Deleting Parts

Parts can be deleted from the schematic.

To delete a part:

1. **Click** the part(s) to be deleted.
2. **Press** the **Del** key

Modifying Part Parameters

To modify part parameters directly on the schematic see *Editing Part Parameters On a Schematic* (users) in the User's Guide.

Changing the Schematic View

Many times schematics can become so large that the entire schematic is not visible. In these cases panning and zooming helps change the current schematic view.

Panning a Schematic

Panning can be used to move the position of the schematic.

To pan:

- Use the scroll bars to move the page up and down or left and right.
  Or
- Select the Pan Tool (keyboard **P**) from the schematic toolbar. Click and drag the schematic to pan with the mouse.

Zooming a Schematic

Use the zooming features to change the viewing area of the schematic.

Ways to zoom a schematic:

- Click one of the following buttons on the Schematic toolbar:

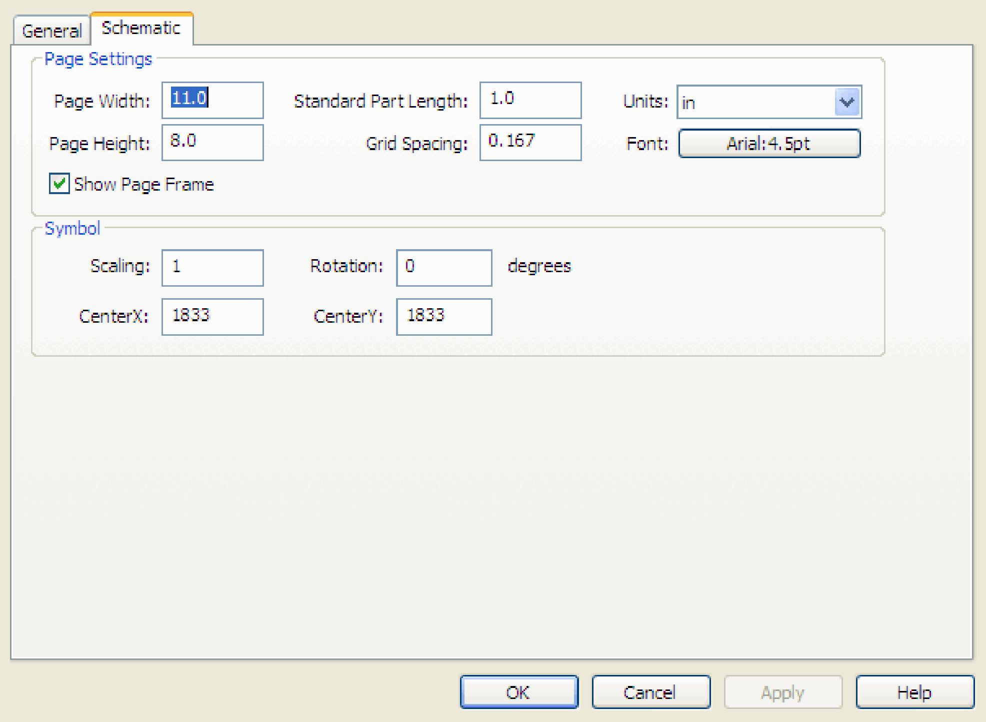
Button	Description	Keyboard	Details
🔍	Zoom an arbitrary area.		Click the schematic and drag the mouse to set the zoom selection rectangle. All items within this rectangle will be zoomed.
🔍	Zoom the schematic to page.	Ctrl+End	Zoom to the page frame.
⛶	Zoom to fit selected parts.		Only selected parts will be zoomed as a group.
✥	Zoom to fit all schematic objects.	Z	Zoom to include all parts in the schematic.

- Move the **mousewheel** in/out to zoom the schematic in/out
- Use the keyboard **+** and **- keys** to zoom in and out.

Changing Schematic Properties

To change the properties of a schematic:

1. Double-click any empty area of a schematic.
2. Click the **Schematic** tab.
3. Make any changes.
4. Click **OK**.



Page Settings

Page Width: 11.0
Standard Part Length: 1.0
Units: in
Page Height: 8.0
Grid Spacing: 0.167
Font: Arial:4.5pt
☑ Show Page Frame

Symbol

Scaling: 1
Rotation: 0 degrees
CenterX: 1833
CenterY: 1833

Page Settings

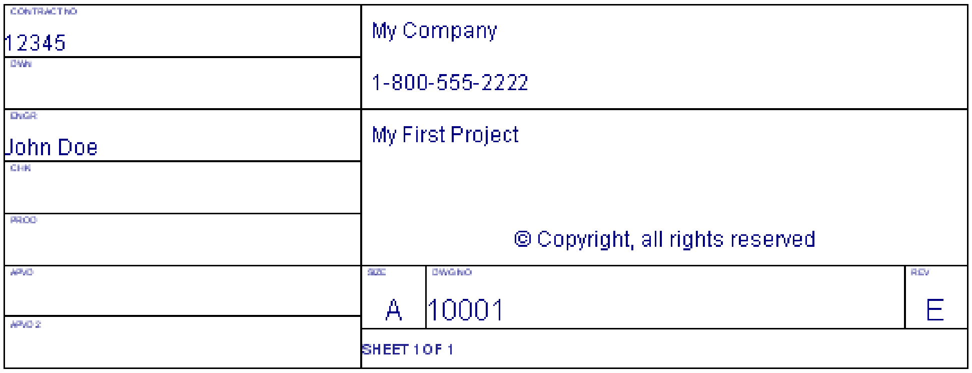
- **Page Width & Height** - The size of the paper (in current units).
- **Standard Part Length** - The length of a resistor part. Defaults to 1 inch. This setting controls the schematic scaling. (If standard part length is set to 0.5, all parts on the schematic will be half-size.)
- **Grid Spacing** - The distance between grid dots.
- **Units** - The units used by the schematic for its settings.
- **Font** - Default font use when text is placed on the schematic.
- **Show Page Frame** - When checked shows the page outline on the schematic.

Symbol

- **Scaling -**
- **Rotation -**
- **CenterX and Y -**

Title Blocks

A title block is used to document a schematic. It often contains information regarding the name of the schematic, the name of the person who drew it, copyright information, etc. A library of common title blocks ship with the product.



CONTRACT NO 12345

DWN

ENGR John Doe

CHK

PROD

APVD

APVD 2

My Company

1-800-555-2222

My First Project

© Copyright, all rights reserved

SIZE A

DWG NO 10001

REV E

SHEET 1 OF 1

Adding a Title Block

There are two ways to add a title block.

Method 1 - **From the Main Menu:**

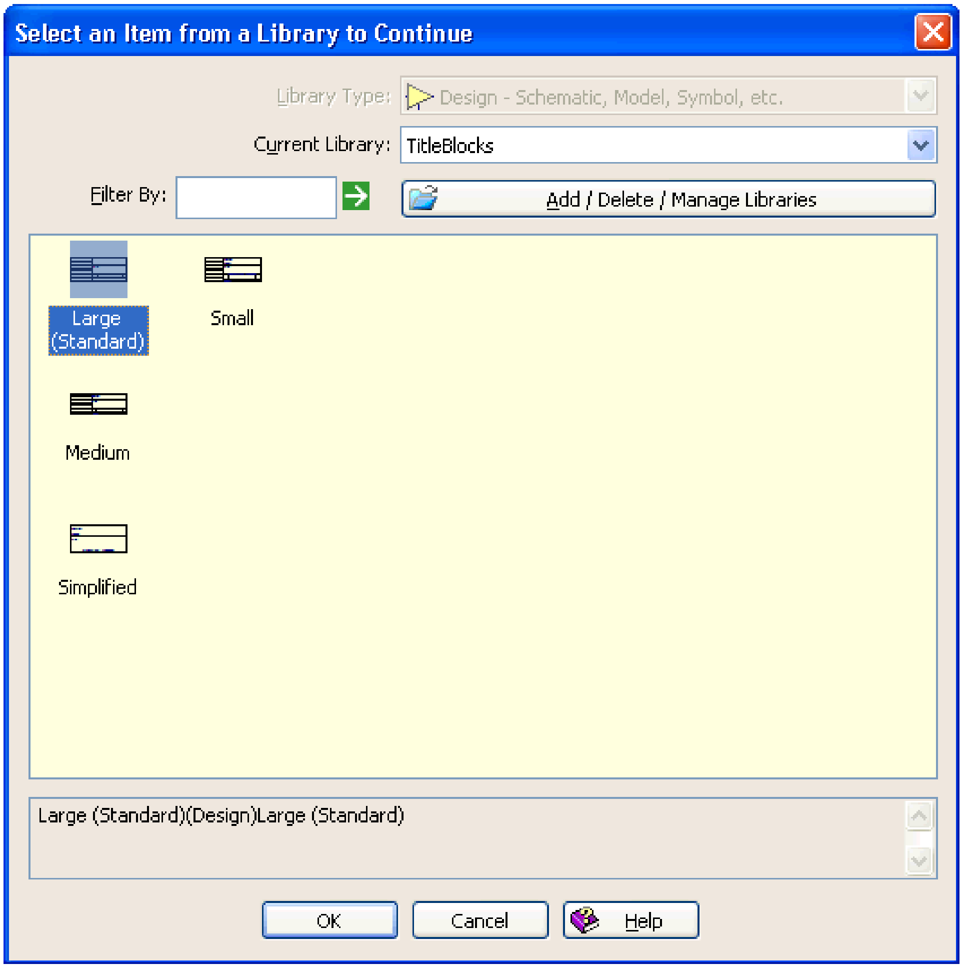
1. **Click** the Schematic menu and select **Add Title Block...**.
2. **Select** the desired title block from the library selector.
3. **Drag** the title block to the location of interest.

Or

Method 2 - **From the Schematic Right Click Menu:**

1. **Right Click** the Schematic and select **Add Title Block...**.
2. **Select** the desired title block from the library selector.
3. **Drag** the title block to the location of interest.

Title Blocks in the Library Selector:



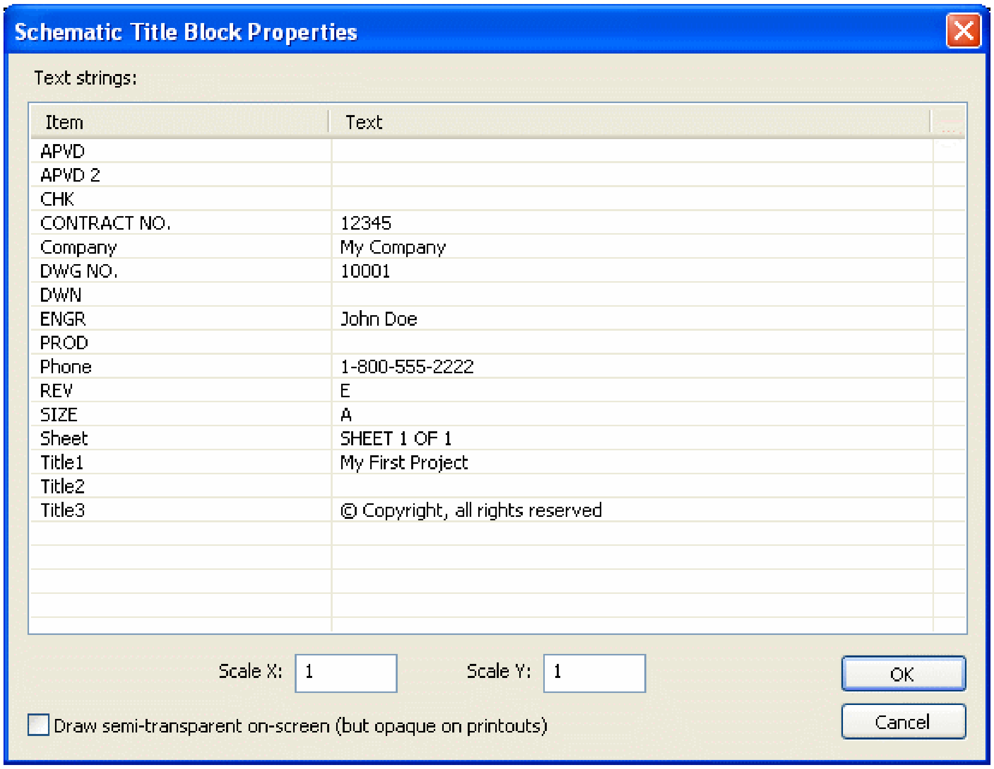
Select an Item from a Library to Continue
Library Type:
Design - Schematic, Model, Symbol, etc.
Current Library:
TitleBlocks
Filter By:
Add / Delete / Manage Libraries
Large (Standard)
Small
Medium
Simplified
OK
Cancel
Help

Large (Standard)(Design)Large (Standard)

Editing the Title Block

To Edit a Title Block:

1. **Double Click** the title block.
2. **Enter** the desired information.
3. **Click** Ok.



Text strings:

Item	Text
APVD	
APVD 2	
CHK	
CONTRACT NO.	12345
Company	My Company
DWG NO.	10001
DWN	
ENGR	John Doe
PROD	
Phone	1-800-555-2222
REV	E
SIZE	A
Sheet	SHEET 1 OF 1
Title1	My First Project
Title2	
Title3	© Copyright, all rights reserved

Scale X: 1 Scale Y: 1

□ Draw semi-transparent on-screen (but opaque on printouts)

- **Item** - Name or titles of information.

  Note: These titles can only be changed on a custom title block symbol.

- **Text** - The actual text string to be drawn in the title block.
- **Scale X & Y** - Scales the title block. For example, 0.5 is half-size.
- **Draw semi-transparent** - When checked draws a faded title block.

Hint
For Advanced users: An equation can be used for the text. For example, if the workspace contains an equation block with a text variable named **CompanyName**, place **=CompanyName** in the Text field of the title block. The **leading = sign** indicates that the text string is actually an expression. When the title block is drawn, the variable will be evaluated and the result will be displayed in the title block.

Creating a Custom Title Block

The easiest way to create a custom title block is to start with an existing one.

1. Open up the **Library Selector**.
2. Set the **Library Type** to **Design**.
3. Change the **Current Library** to **TitleBlocks**.
4. **Double click** on the title block to be modified.

   Note: This will add the title block as a schematic symbol on the workspace tree.

5. **Edit** title block symbol as needed, using annotations:
   - The text annotation **Name** property will be used as the **Item Name**. The **Show Name** option must be checked in this annotation dialog box to see this name on the schematic.
   - The text annotation **Enter 1 or more lines of text** property will be the text value of the field that appears in the title block.

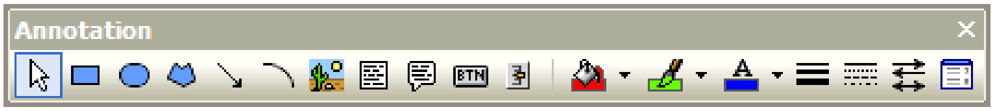
     Hint
     For best results, only use 1 line of text and keep it fairly short.

   - **Images**, like a company logo, or any other annotation can be placed on the custom symbol.

6. Save the workspace.
7. On workspace tree, **right-click** the symbol and use **"Copy To"** to place the symbol in a new (or existing) library.
8. To use your new custom title block on a schematic, use **"Add Title Block..."** and select the custom title block from the library it was saved in.

Annotating Schematics

The Annotation button ( ) on the Schematic toolbar gives you access to the Annotation toolbar.



The Annotation toolbar provides tools like lines, circles, and text that you can use to point out details of interest on a schematic, draw a box around a group of components, etc.

> **Hint**
> Double-click a text annotation to set the horizontal and vertical justification (text alignment).

> **Hint**
> **For Advanced users:** An equation can be used for the text. For example, if the workspace contains an equation block with a text variable named **CompanyName**, place **=CompanyName** in the Text field of the title block. The **leading = sign** indicates that the text string is actually an expression. When the title block is drawn, the variable will be evaluated and the result will be displayed in the title block.

Text annotations can display model and parameter info when used within a custom symbol. This is implemented via macro-text-substitution. When symbol text is drawn on a schematic, the displayed text is modified prior to output. For example, Name=%Model% would be displayed as "Name=Resistor" on a symbol using a resistor model. The recognized macro strings are:

1. **%Des%** - Displays the part's designator.
2. **%Model%** - Displays the name of the model attached to the part.
3. **%MODEL%** - Displays the model name in UPPERCASE.
4. **%ParameterName%** - Displays the value of the specified model parameter attached to the part.

Adding Text

Text can be placed directly on a schematic.

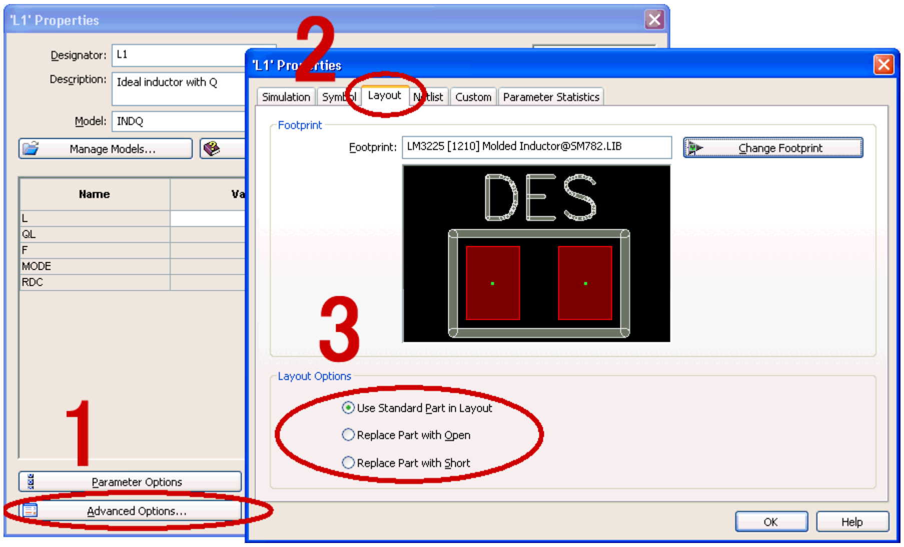
To add text:

1. **Click** the Annotation button ( ) to display the Annotation toolbar.
2. **Click** the **Text** button ( ).
3. **Click** in the Schematic window where you want to place the text.
4. **Type** the **text** into the **Enter 1 or lines of text** field.

Specifying Schematic Part Layout Options

Often in RF circuits, you want to model packaging or component parasitics. You do this by placing lumped parts in series or parallel with the actual part. However, you do not want these parts to display in the layout.

To prevent a schematic part from displaying in a layout:



'L1' Properties

Designator: L1

Description: Ideal inductor with Q

Model: INDQ

Name	Va
L	
QL	
F	
MODE	
RDC	

Footprint: LM3225 [1210] Molded Inductor@SM782.LIB

Layout Options

● Use Standard Part in Layout

○ Replace Part with Open

○ Replace Part with Short

1. Double-click a part in the schematic and click the Advanced Options button.
2. Click the Layout tab.
3. Click an option. For capacitors, select **Replace Part with Open**. For inductors and resistors, select **Replace Part With Short**.
4. Click **OK**.

Using DisCos

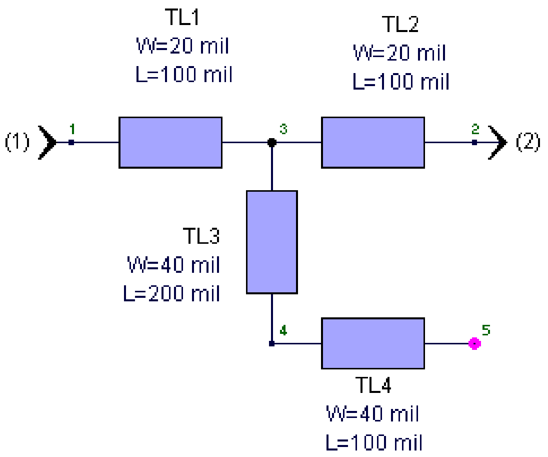
Inserting discontinuities such as bends and tees for distributed parts is simplified with parts called DisCos. These small symbols are easily added to any schematic that uses microstrip lines or similar distributed parts. This avoids the need to manually add these models by disconnecting the parts, selecting the discontinuity from toolbars, reconnecting the parts, and defining parameters.

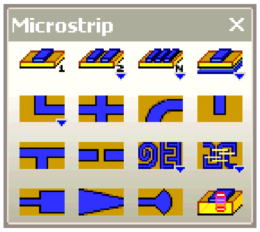
Using Distributed Parts

You can use distributed parts such as microstrip lines as schematic objects. Place distributed parts in a schematic the same way as lumped parts such as resistors and capacitors. Also, it is now easier to insert discontinuities such as bends and tees in a schematic.

To add a microstrip:

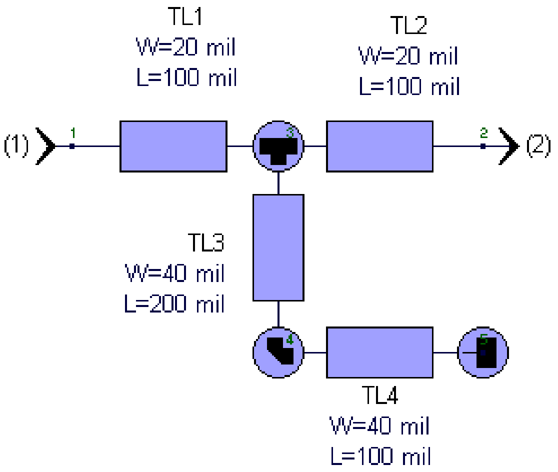
1. Click the New Item button ( ) on the Workspace Tree toolbar and select **With a Schematic** from the Designs menu.
2. Click the Microstrip button ( ) on the Schematic toolbar.
3. Click the Microstrip Line button ( ) on the Microstrip toolbar, and then click in the Schematic window to place the part.
4. Press the spacebar to repeat the selection.
5. Click the right terminal of the first part to place the second part.
6. Add input and output ports.
7. Continue to add microstrip parts as shown below. Double-click each part to enter values (width and height).





To add discontinuities:

1. Right-click the node in the schematic that connects the three line segments.
2. Select **Add Tee DisCo**. A small symbol appears. This DisCo (discontinuity) functions the same as the tee part from the Microstrip window.
3. Right-click the bend.
4. Click the **Chamfered Corner DisCo** check box.
5. Place an End DisCo on the open terminal. The resulting schematic is shown below.



> Note: For this example, clear the Absorb DisCos check box to preserve circuit response when possible. This option adjusts the line lengths connected to each DisCo and is useful if the design is optimized.

Reviewing Rules for Using DisCos

The following rules apply for using DisCos:

- DisCos can connect only to a node with one or more transmission lines.
- All connected transmission lines must be the same type and use the same substrate.
- The number of transmission lines connected to a node is one factor in determining the DisCos to apply.

Number of Lines	Possible DisCos
1	End Effect
2	Step or Bend
3	Tee
4	Cross

- A bend is inserted at a node where two transmission lines meet at a right angle in a schematic.
- A step is applied at a node between two transmission lines of unequal widths.

Reviewing Tips for Using DisCos

The following tips apply for using DisCos:

- When generating the original schematic, be aware that the orientation of parts affects the application of DisCos. For example, transmission line parts meeting at right angles result in a "bend" DisCo when converted. The same orientation is important when layouts are generated from the schematic.
- If more than one line type is used in a schematic, you can still use DisCos by right-clicking each node. Note that DisCos can connect only to a node with one of more transmission lines, still applies.
- The option of preserving circuit response when possible is useful if the circuit response is already optimized. This minimizes the need for major changes in components to compensate for the discontinuities.

Using Substrates

If your schematic contains any parts that use a substrate (such as microstrip, slabline, stripline, coax, and waveguide), you must add one or more substrates to your design before you can analyze it. Genesys lets you add and change substrates using the Substrate Properties window. You can also add a substrate from the Genesys library or a user library.

Adding and Changing Substrates

A substrate is required whenever a physical component is used in a schematic or a netlist. Components that require substrates are:

- *Coaxial* (users)
- *Microstrip* (users)
- *Slabline* (users)
- *Stripline* (users)
- *Waveguide* (users)

You can also use substrates to provide metal and substrate characteristic information to EMPOWER by selecting the substrate name in the EMPOWER Layer table.

To add a new substrate:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Substrate**.
2. Type substrate and metal layer parameter values.
3. Click **OK**.

To add a library substrate:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **From Library**.
2. Select the Substrate type of library
3. Open a substrate library
4. Locate the substrate you want to add, and then click **Open** or double-click the substrate.

To change substrate properties:

1. Right-click the substrate in the Workspace Tree and select **Properties**.
2. Make the changes you want.
3. Click **OK**.

To change substrate properties of ALL the parts in a schematic (to use a tuned variable, etc.):

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **From Library**.
2. Select the Script type of library
3. Add the ReplaceSubstrates script to your workspace
4. Double-click to open the script, edit it according to the instructions in the script.
5. Run the script.

Substrate Parameters for Microstrip and Stripline

For an overview of substrate parameters, see *Substrate Parameters for Microstrip and Stripline* (users).

Substrate Parameter Tables

For a listing of substrate parameter tables, including:

- Loss Tangent
- Metal Thickness
- Relative Dielectric Constants
- Relative Permeability
- Resistivity
- Surface Roughness
  see *Substrate Parameter Tables* (users).

Substrate Parameter Tables

Loss Tangent

The dielectric loss tangents of some common materials are:

Material	**tanD at 100 MHz**	**tanD at 10 GHz**
Air	0.0	0.0
Polyolefin, irradiated	3E-4	3E-4
PTFE (Teflon)	2E-4	1.5E-4
RT/Duroid 5880, PTFE microglass	5E-4	9E-4
Teflon, glass microfiber	5E-4	9E-4
Teflon, woven quartz	6E-4	6E-4
Teflon, woven glass	1.5E-3	2E-3
Polystyrene, cross linked	2E-4	7E-4
Polystyrene, glass microfiber	4E-4	2E-3
Quartz, fused	2E-4	6E-5
G10 Epoxy glass	8E-3	No Data
Pyrex glass	3E-3	7E-3
Alumina, 99.5%	1E-4	1E-4
RT/Duroid 6010.5, PTFE ceramic	2E-3	2.3E-3

The dielectric loss tangents for some materials commonly used in coaxial cables are:

Material	**tanD at 100 MHz**	**tanD at 3 GHz**
Air	0.0	0.0
Teflon	2E-4	15E-4
PolyEthylene, DE-3401	2E-4	3.1E-4
Polyolefin, irradiated	3E-4	3E-4
Polystyrene	1E-4	3.3E-4
Polyvinal formal (Formvar)	1.3E-2	1.1E-2
Nylon	2E-2	1.2E-2
Quartz, fused	2E-4	6E-5
Pyrex Glass	3E-3	5.4E-3
Water, distilled	5E-3	1.6E-1

ℹ This data is for solid materials. Foamed materials have lower loss tangents. These data are approximate. Consult manufacturer for critical applications.

Metal Thickness

The thickness of the conductor metalization for planar structures. The algorithms for microstrip are most accurate for thin metalization, but both loss and Zo are corrected for thickness.

For stripline, the algorithms are more accurate to thicker metalization. Thickness to 0.1*b or to the width is permissible.

Commonly used thicknesses:

Metallization Type	Thickness (mm)	Thickness (mils)
½ ounce copper	0.018	0.71
1 ounce copper	0.036	1.42
2 ounce copper	0.072	2.83

Relative Dielectric Constants

Er is the substrate or dielectric constant, relative to free space. Following are constants of some common materials:

Material	Dielectric Constant
Air	1.0
Alumina, 99.5%	10
G10/FR4 Epoxy glass	4.8 (varies)
PolyEthylene, DE-3401	2.26
Polyhexamethyleneadipamide (Nylon)	2.9
Polyolefin, irradiated	2.32
Polystyrene	2.53
Polystyrene, cross linked	2.53
Polystyrene, glassed cross linked	2.62
PolyTetraFluoroEthylene (Teflon)	2.10
Polyvinal formal (Formvar)	2.8
PTFE, glass microfiber	2.35
PTFE (Teflon)	2.10
PTFE, woven glass	2.55
PTFE, woven quartz	2.47
Pyrex glass	4.84
Quartz, fused	3.8
RT/Duroid	2.20
RT/Duroid, PTFE ceramic filled	10.5
Water, distilled	77

> ℹ Foamed materials have lower dielectric constants. These data are approximate; consult manufacturer for critical applications.

Relative Permeability

Ur is the substrate permeability, relative to free space. Most line types do not allow substrates or dielectrics with magnetic properties.

Resistivity

Rho is the lines metalization resistivity relative to copper.

Some common values are:

Material	Resistivity Relative to Copper
Copper, annealed (1.7e-8 ohm meters)	1.00
Copper, hard drawn	1.03
Silver	0.95
Gold	1.42
Aluminum	1.64
Tungsten	3.25
Zinc	3.4
Brass	3.9
Cadmium	4.4
Nickel	5.05
Phosphor-bronze	5.45
Platinum	6.16
Stainless Steel, 18-8	52.8

Surface Roughness

Sr is the metalization surface roughness, using the current units.

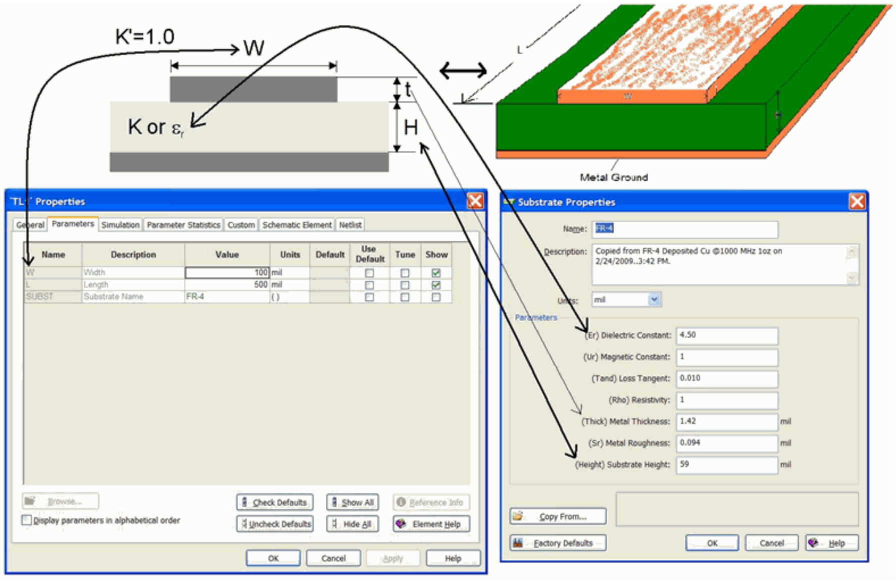
Conductor losses increase with larger values of surface roughness. Approximate values for copper PWB surface roughness:

Electrodeposited Copper	Sr value (mm)	Sr value (mils)
½ ounce	0.0019	0.075
1 ounce	0.0024	0.094
2 ounce	0.0029	0.114

Rolled Copper	Sr value (mm)	Sr value (mils)
½ ounce	0.0014	0.055
1 ounce	0.0014	0.055
2 ounce	0.0014	0.055

Substrate Parameters for Microstrip and Stripline

Microstrip Overview



'TL1' Properties

General | Parameters | Simulation | Parameter Statistics | Custom | Schematic Element | Netlist

Name	Description	Value	Units	Default	Use Default	Tune	Show
W	Width	100	mil		□	□	☑
L	Length	500	mil		□	□	☑
SUBST	Substrate Name	FR-4	( )		□	□	□

□ Display parameters in alphabetical order

Substrate Properties

Name: FR-4

Description: Copied from FR-4 Deposited Cu @1000 MHz 1oz on 2/24/2009..3:42 PM.

Units: mil

Parameters

(Er) Dielectric Constant: 4.50

(Ur) Magnetic Constant: 1

(Tand) Loss Tangent: 0.010

(Rho) Resistivity: 1

(Thick) Metal Thickness: 1.42 mil

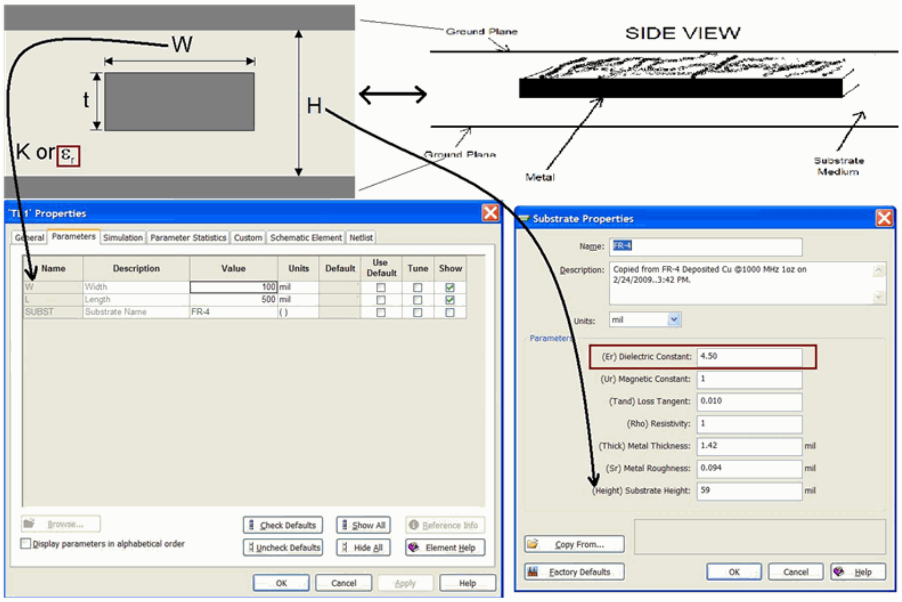
(Sr) Metal Roughness: 0.094 mil

(Height) Substrate Height: 59 mil

The substrate height "H" is the entered value in the Substrate properties field "(Height) Substrate Height"

More information regarding substrate properties can be found in the *Substrate Parameter Tables* (users) section.

Stripline Overview



'TL1' Properties

General | Parameters | Simulation | Parameter Statistics | Custom | Schematic Element | Netlist

Name	Description	Value	Units	Default	Use Default	Tune	Show
W	Width	100	mil		□	□	☑
L	Length	500	mil		□	□	☑
SUBST	Substrate Name	FR-4	( )		□	□	□

□ Display parameters in alphabetical order

Substrate Properties

Name: FR-4

Description: Copied from FR-4 Deposited Cu @1000 MHz 1oz on 2/24/2009..3:42 PM.

Units: mil

Parameters

(Er) Dielectric Constant: 4.50

(Ur) Magnetic Constant: 1

(Tand) Loss Tangent: 0.010

(Rho) Resistivity: 1

(Thick) Metal Thickness: 1.42 mil

(Sr) Metal Roughness: 0.094 mil

(Height) Substrate Height: 59 mil

If the Stripline will be manufactured by purchasing two substrates, each substrate will have H/2=H' height. Length "L" is the length of the transmission strip. This dimension is "looking into the page" toward the "z" axis.

More information regarding substrate properties can be found in the *Substrate Parameter Tables* (users) section.

Parameter List

Er	Dielectric Constant of substrate
Ur	Magnetic Constant of substrate
Tand	Loss Tangent of substrate
Rho	Resistivity of substrate
Thick	Metal thickness
Sr	Metal roughness
Height	Substrate height (top ground place to bottom ground plane)

Using S-Parameters in Genesys

This section shows how S-Parameter data can be incorporated into Genesys designs and exported to other programs.

S-Parameters are commonly used in RF circuits to represent incident and reflected traveling waves. S-Parameters are created by a linear simulation and are generally available from component manufactures. Several libraries of S-Parameter data ship with Genesys. S-Parameters in Genesys are imported and exported in the Touchstone format.

Contents

Creating S-Parameter Data

1. Create the schematic for which the S-Parameter data will be represented
2. Add a linear analysis and point it to the desired schematic
3. Set the frequency range and step size of the linear analysis to the desired resolution of the S-Parameter data
4. Run the linear analysis
5. Export linear analysis data as a S-Parameter file

Using S-Parameters in a Simulation

The use model for S-Parameters manually imported into the workspace versus file based S-Parameter is slightly different. The model used in the schematic determines how the S-Parameters will be managed.

Displaying S-Parameter Data

The easiest way to display S-Parameter data is to open up the S-Parameter dataset and the right click on the **S** variable. Then select **Create Table** or **Graph** and the type of graph. The data will automatically be displayed.

Files with No Noise Parameters

Noise parameters can be added to an S-Parameter file so noise characteristics of active devices can be simulated. For passive devices the noise characteristics are determined directly from the S-parameters so noise parameters are not needed.

⚠ **Caution**
If an S-Parameter file represents an active device **no** noise will be created and a warning will be displayed.

File Based S-Parameters

File based S-Parameter import the S-Parameters from a file into a dataset providing simulation cache. This dataset is used when reloading the workspace to re-cached the data. If the dataset is deleted the S-Parameters will be re-imported the next time a simulation needs the data.

1. Place a S-Parameter file based part in the schematic (*1-port* (part), *2-port* (part), or *N-port* (part). This can be done from the Linear Toolbar or the Part Selector
2. Double click the part to bring up the part properties
3. Click the **Browse** button to browse to the S-Parameter file
4. Add an analysis and point it to the desired schematic
5. Run the analysis

Physical S-Parameters

S Parameters can be taken or formed in such a way that they represent non physical parts like negative resistors. Realistic real world answers only come when S-Parameters are physical. If S-parameters are physical, then the corresponding Y-parameters will meet **all** of the following requirements:

1. The real part of every diagonal entry must be positive. i.e. **Real.Yp[i,i] > 0**
2. The real part of every non-diagonal entry must be negative. i.e. **Real.Yp[i,j] < 0** where i is not equal to j
3. The absolute value of the row real summation, excluding the diagonal, must be less than real value of the diagonal in that row. i.e. **abs (sum(Real.Yp[i,j])) < Real.Yp[i,i] where i is not equal to j**
4. The absolute value of the column real summation, excluding the diagonal, must be less than the real value of the diagonal in that column. i.e. **abs (sum(Real.Yp[i,j])) < Real.Yp[j,j] where i is not equal to j**

Note: It is assumed the Y parameters are in Real _ j Imaginary format.

Examples:
Here are some typical Y-parameters (which is converted from S-parameters):

The Y parameter matrix for F = 3000 is:

0.077 - j0.122   -0.078 + j0.123

-0.078 + j0.123   0.078 - j0.121

This matrix meets items 1 and 2 but not 3 and 4, because abs(Real.Y[1,2]) > Real.Y[1,1] or abs(Real.Y[2,1]) > Real.Y[1,1], so these S parameters are **non physical**.

Sub-Network Models

A sub-network model is used to abstract a model or group of models to something easier to use and manage from a users perspective. This type of model hides implementation details that may confuse the user or distract from the readability of a simulation topology.

For example, a user may want to simulate the effects of a non-linear filter. Models exist for filters and non-linear blocks. However, there is no non-linear filter model. A new sub-network model can be created out of the two existing models. The parameters from these two models can be abstracted to only reveal parameters that user would be interested in entering for this type of sub-network model.

The abstraction of the Sub-Network model in Genesys is really an object called a design. The Sub-Network model is really a design object with the following attributes:

- PartList
- Schematic
- Equations
- Parameters
- Notes

All of these attributes are interrelated except for the Notes which serve as documentation or help for the Sub-Network model.

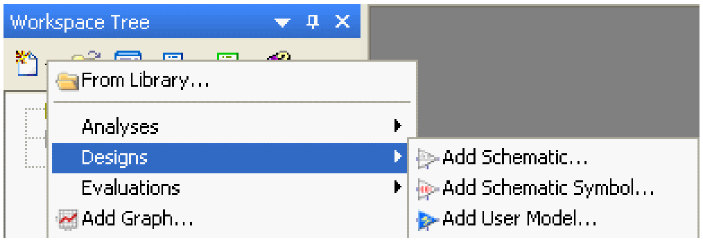
Contents

Creating a Parameterized Sub-Network Model

There are two different ways to to create a sub-network model in Genesys. One is by clicking on the New Item button ( ) on the Workspace Tree toolbar. The other is by right clicking on a folder in the workspace tree.

Method 1 - **Clicking on the New Item Button**

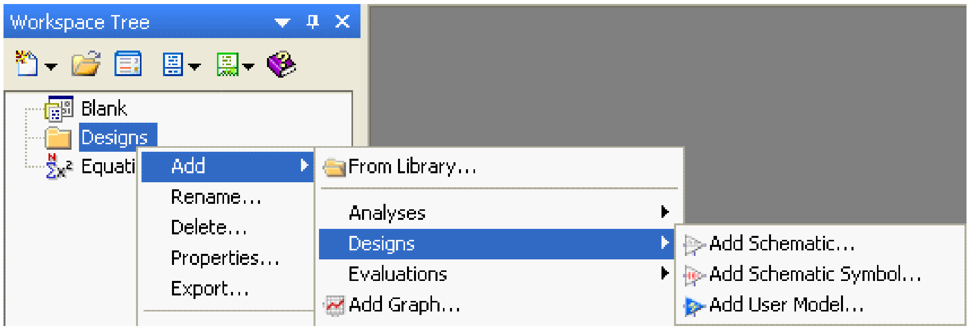
1. Click the New Item button ( ) on the Workspace Tree toolbar
2. Select the 'Designs >' submenu
3. Now select 'Add User Model...'
4. This model will be added under the folder that last selected in the workspace tree



Or

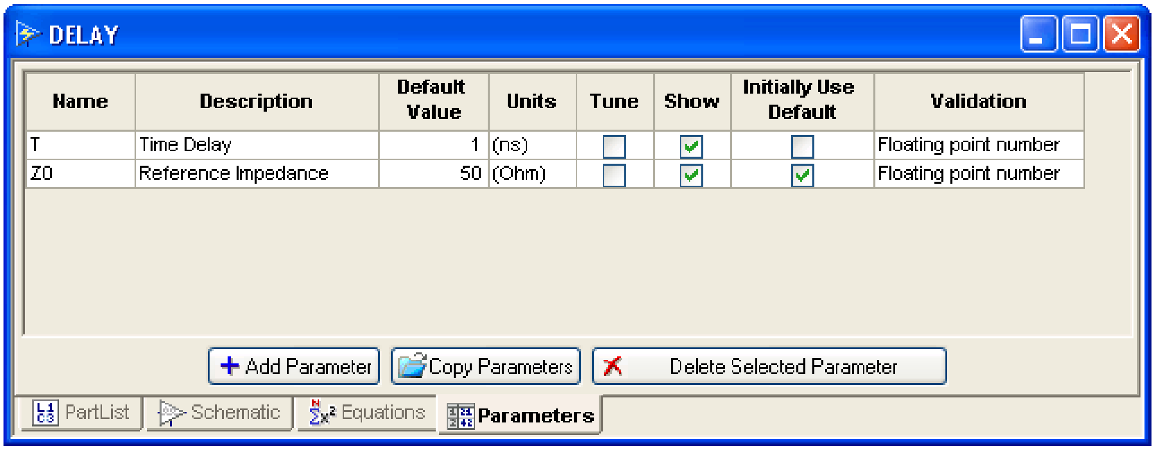
Method 2 - **Right Clicking on a Workspace Folder**

1. Right click on a folder in the workspace tree to bring up the right click menu.
2. Select the 'Add >' submenu.
3. Select the 'Designs >' submenu
4. Now select 'Add User Model...'
5. The design will be added under the folder that was initially right clicked



Note
If you create the new design in the wrong folder, simply drag it to the folder of interest.

Entering User Parameters



Name	Description	Default Value	Units	Tune	Show	Initially Use Default	Validation
T	Time Delay	1	(ns)	□	☑	□	Floating point number
Z0	Reference Impedance	50	(Ohm)	□	☑	☑	Floating point number

1. Add new parameters by clicking the Add Parameter button.
2. Copy selected parameters (from the parts in the design) into the list by clicking the Copy Parameters button. A selection dialog box is displayed. Select individual parameters (to copy from the base design) by placing a checkmark beside each parameter you wish to copy. Then click OK.
   Tip: This is the recommended way of adding parasitics (etc.) to an existing part (a "user model").
3. Delete unwanted entries with the Delete Selected Parameter button.

- **Name** - The name of the parameter.
- **Description** - A short description of the parameter
- **Default Value** - The normal, standard value for this parameter
- **Units** - The units-of-measure for this parameter
- **Tune** - Is it normally tuned?
- **Show** - Is it normally shown on a schematic?
- **Initially Use Default** - Should the Default value be used when the part is placed on a schematic?
- **Validation** - Usage rules that determine if a parameter value is valid and in-range.

See details below.

- **Hide Condition -** Dependence of the activity and visibility of the parameter on values of other parameters of the design. See details below.

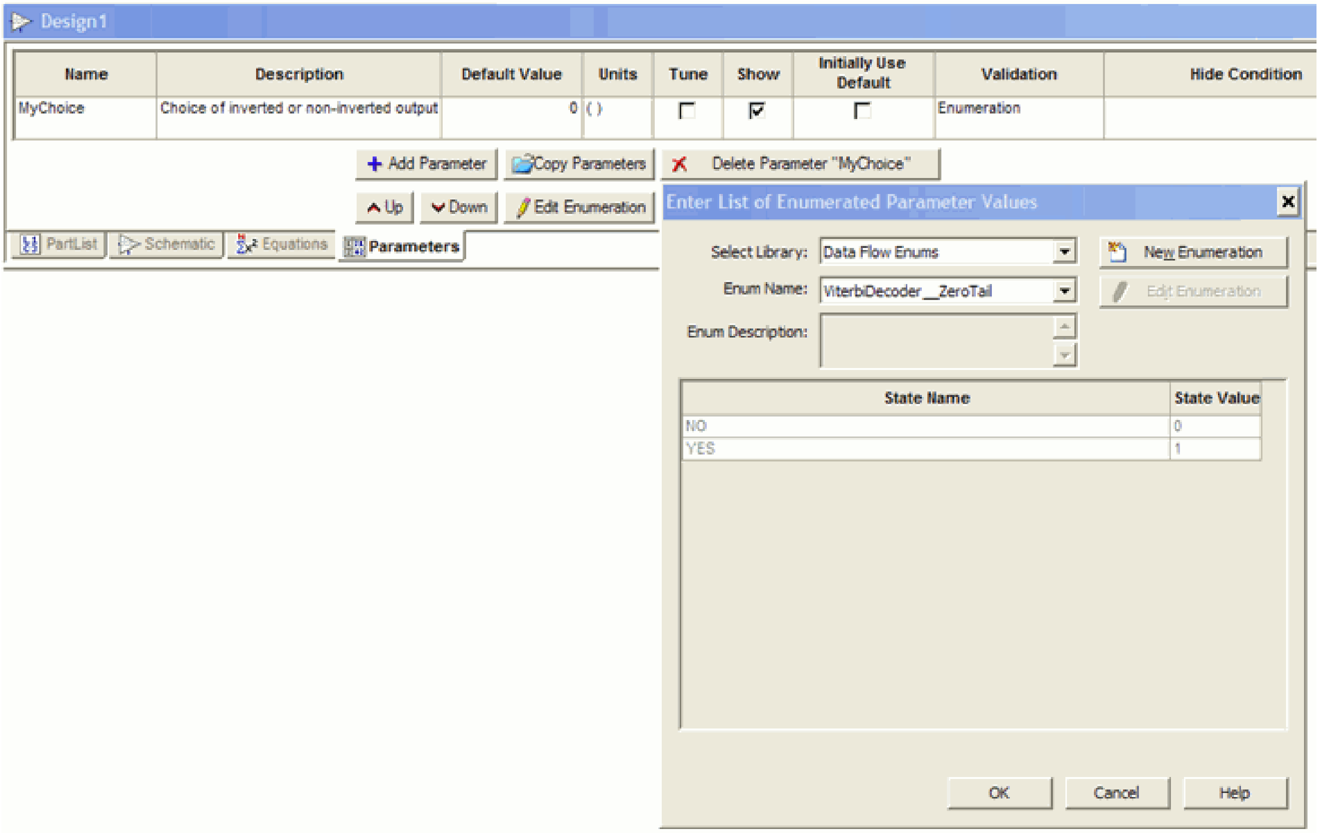
Validation Types

Type	**Comment**
Floating point number	1.0, 1e-6, etc. are valid entries
Warn if negative	Posts a warning if the value is < 0
Warn if non-positive	Posts a warning if the value is < 1
Positive integer	Only numbers like 1, 2, 3, ... are allowed
<None>	No validation will be performed
Text	The parameter is a string; any text is valid
Warning	Always generates a warning
Error if negative	Posts an error if the value is < 0
Error if non-positive	Posts an error if the value is < 1
Error	Always generates an error
Filename	Brings up a browse button for file selection as well option for manual a text entry
Integer	Any integer value
Complex number	Complex number in RI MathLang syntax, e.g. X + j*Y. Real and integer values supported by default
Integer array	Fully defined MxN array of integers with comma delimited columns and semi-colon delimited rows
Floating point array	Fully defined MxN array of integer or real numbers with comma delimited columns and semi-colon delimited rows
Complex array	Fully defined MxN array of integer, real or complex numbers with comma delimited columns and semi-colon delimited rows
Enumeration	Allows definition of arbitrary user-defined labels and options for assigning values to the parameter of interest

Note
An array parameter may be specified as a scalar number without any delimiters as in MyArray0=2.345. It may be a one-dimensional vector as in MyArray1=[2, 3, -4].
Two-dimensional arrays are specified row over column as MyArray2=[1, 2, 3; 4, 5, 6] where the first three parts form the first row.
Higher dimensions are created by appending nested versions of 2-D representations separated by semi-colons e.g.
MyArray3=[[1, 2; 3, 4; 5, 6]; [-1, -2; -3, -6; -5, -4]].
This is a 3-D array consisting of 2 separate 3x2 2-D arrays such that matrix size is 2x3x2.

Using enumerated parameters

The process of defining an enumerated parameter starts with the selection of this validation type followed by selection of the context sensitive **Edit Enumeration** button which appears adjacent to the other parameter editing buttons. Clicking this button will invoke the **Enter List of Enumerated Parameter Values** dialog box.



Name	Description	Default Value	Units	Tune	Show	Initially Use Default	Validation	Hide Condition
MyChoice	Choice of inverted or non-inverted output	0	( )	□	☑	□	Enumeration	

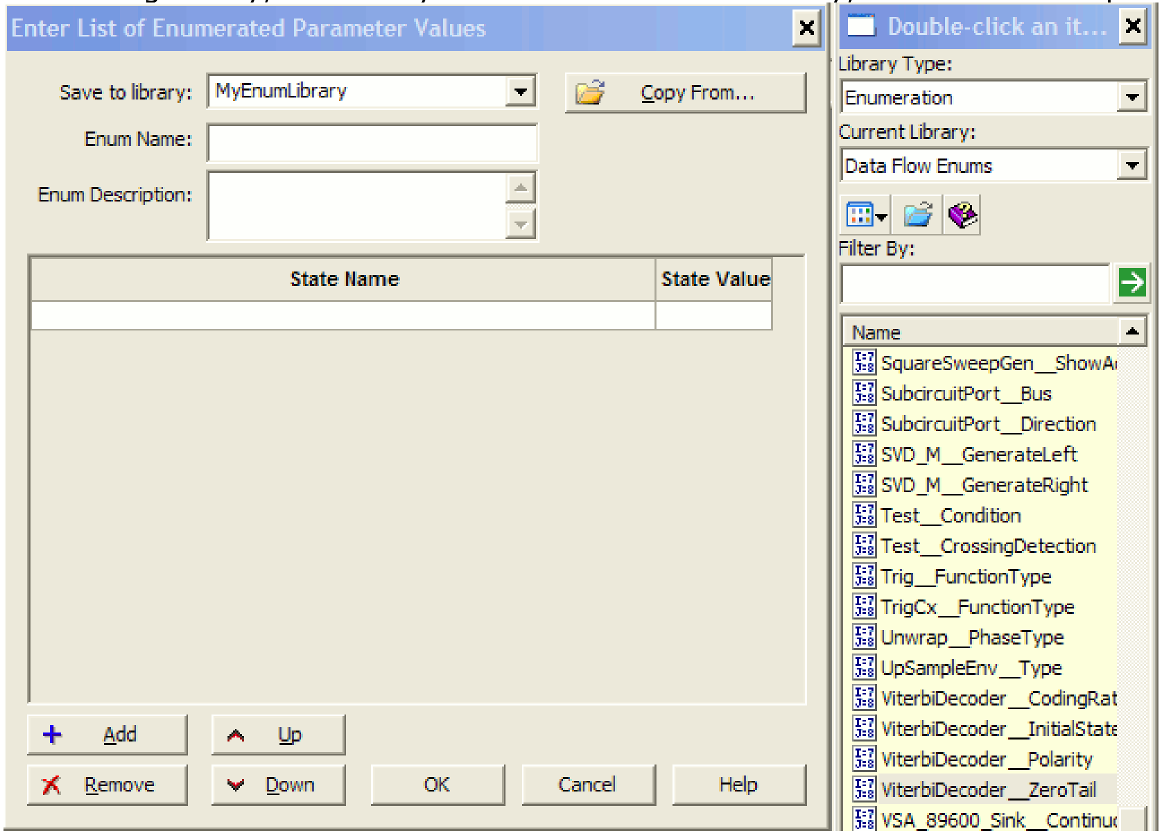
Select Library: Data Flow Enums

Enum Name: ViterbiDecoder__ZeroTail

Enum Description:

State Name	State Value
NO	0
YES	1

It is possible to choose a pre-defined enumeration template by selecting the library and enumeration name. Customizations can be performed by clicking the **New Enumeration** button which is transformed into a **Copy From ...** button to allow graphical selection from an existing library, or a newly created enumeration library, name and description.



Save to library: MyEnumLibrary

Enum Name:

Enum Description:

State Name	State Value

Library Type: Enumeration

Current Library: Data Flow Enums

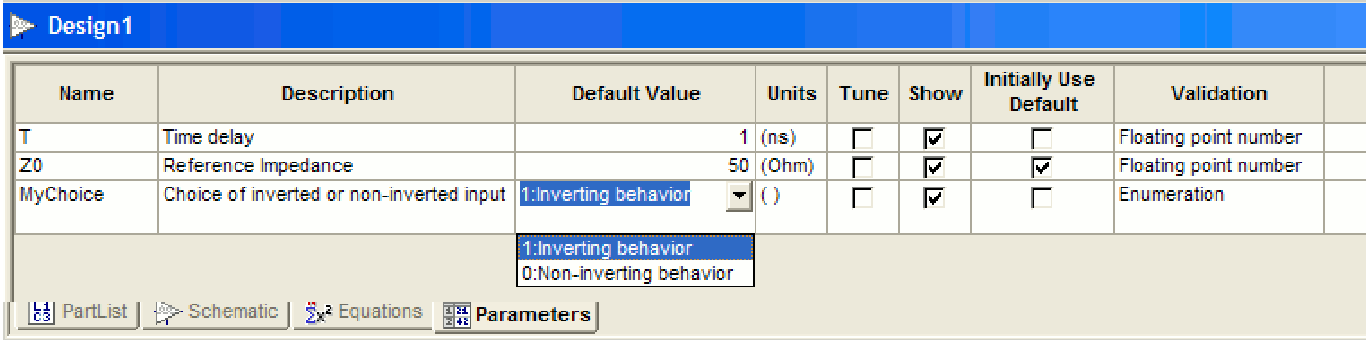
Filter By:

Name
SquareSweepGen__ShowA
SubcircuitPort__Bus
SubcircuitPort__Direction
SVD_M__GenerateLeft
SVD_M__GenerateRight
Test__Condition
Test__CrossingDetection
Trig__FunctionType
TrigCx__FunctionType
Unwrap__PhaseType
UpSampleEnv__Type
ViterbiDecoder__CodingRat
ViterbiDecoder__InitialState
ViterbiDecoder__Polarity
ViterbiDecoder__ZeroTail
VSA_89600_Sink__Continu

Names and states of manually created enumerations or modifications of existing enumerations can be directly entered into the table and organized using the **Add**,

Remove, **Up** and **Down** buttons.

Upon accepting the enumeration list, the corresponding drop-down menu is created under the **Default Values** column for this parameter in the main **Parameter** tab. Note that the first entry of the enumeration table will be treated as the initial default value regardless of the state number associated with it. In this example, the first entry was *1:Inverting Behavior*, which despite its state number being 1, not 0, was picked as the default in the **Parameters** main tab. The user can set a different default state prior to leaving this tab.

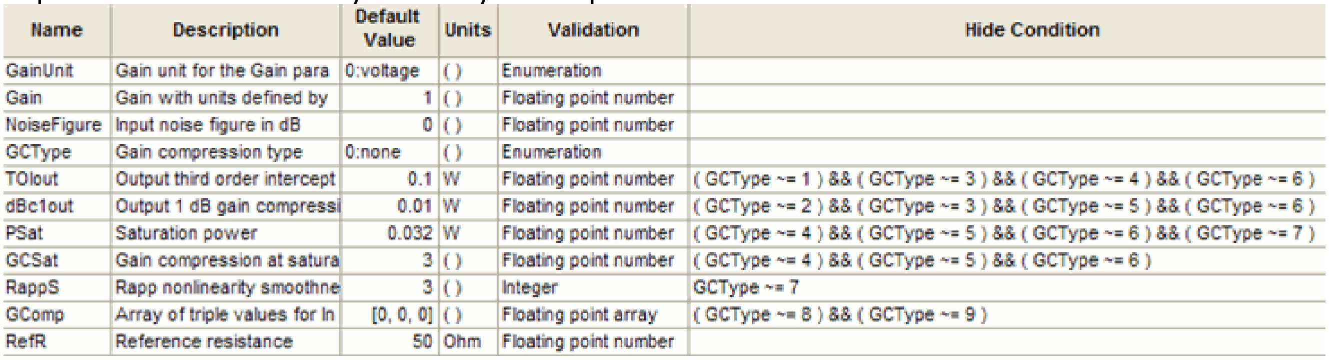


Name	Description	Default Value	Units	Tune	Show	Initially Use Default	Validation
T	Time delay	1	(ns)	□	☑	□	Floating point number
Z0	Reference Impedance	50	(Ohm)	□	☑	☑	Floating point number
MyChoice	Choice of inverted or non-inverted input	1:Inverting behavior	( )	□	☑	□	Enumeration

Setting Hide Condition

The final column of the Parameter entry table allows the user to set up Boolean conditions for establishing a clean functional approach to parameter definitions - by allowing conditional deactivation and hiding of parameters.
MathLang syntax can be used to set up conditions of indivisibility of one parameter based on current values assigned to other parameters.

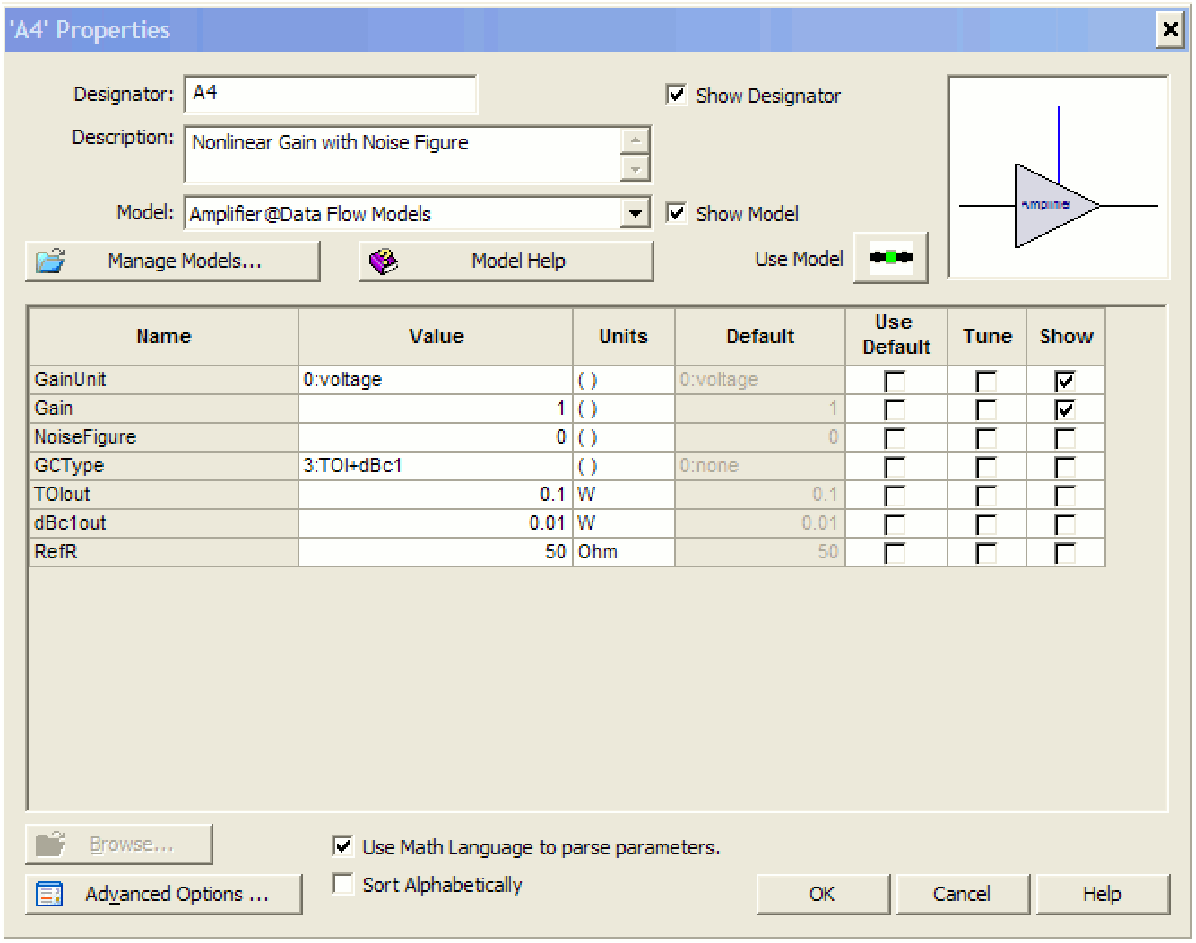
One example is shown in the *Amplifier* part of the *Algorithm Design* library. This built-in component has a total of 11 parameters as shown in the model view which can be imported from the library into any workspace.



Name	Description	Default Value	Units	Validation	Hide Condition
GainUnit	Gain unit for the Gain para	0:voltage	( )	Enumeration	
Gain	Gain with units defined by	1	( )	Floating point number	
NoiseFigure	Input noise figure in dB	0	( )	Floating point number	
GCType	Gain compression type	0:none	( )	Enumeration	
TOIout	Output third order intercept	0.1	W	Floating point number	(GCType ~= 1) && (GCType ~= 3) && (GCType ~= 4) && (GCType ~= 6)
dBc1out	Output 1 dB gain compressi	0.01	W	Floating point number	(GCType ~= 2) && (GCType ~= 3) && (GCType ~= 5) && (GCType ~= 6)
PSat	Saturation power	0.032	W	Floating point number	(GCType ~= 4) && (GCType ~= 5) && (GCType ~= 6) && (GCType ~= 7)
GCSat	Gain compression at satura	3	( )	Floating point number	(GCType ~= 4) && (GCType ~= 5) && (GCType ~= 6)
RappS	Rapp nonlinearity smoothne	3	( )	Integer	GCType ~= 7
GComp	Array of triple values for In	[0, 0, 0]	( )	Floating point array	(GCType ~= 8) && (GCType ~= 9)
RefR	Reference resistance	50	Ohm	Floating point number	

Observe that the parameters *TOIout*, *dBc1out*, *PSat*, *GCSat*, *RappS* and *GComp* all have conditions of inactivity defined based on the value of *GCType*. For instance, *TOIout* is to be hidden and its assigned value ignored if *GCType* is not in the set {1, 3, 4, 6}. The corresponding behavior can be observed when placing an instance of the part on a schematic and invoking its **Properties** dialog box.

When *GCType* is selected to be a member of the above mentioned set, e.g. to 3:TOI+1dBc, the *TOIout* parameter is displayed and enabled for editing.



'A4' Properties

Designator: A4 ☑ Show Designator

Description: Nonlinear Gain with Noise Figure

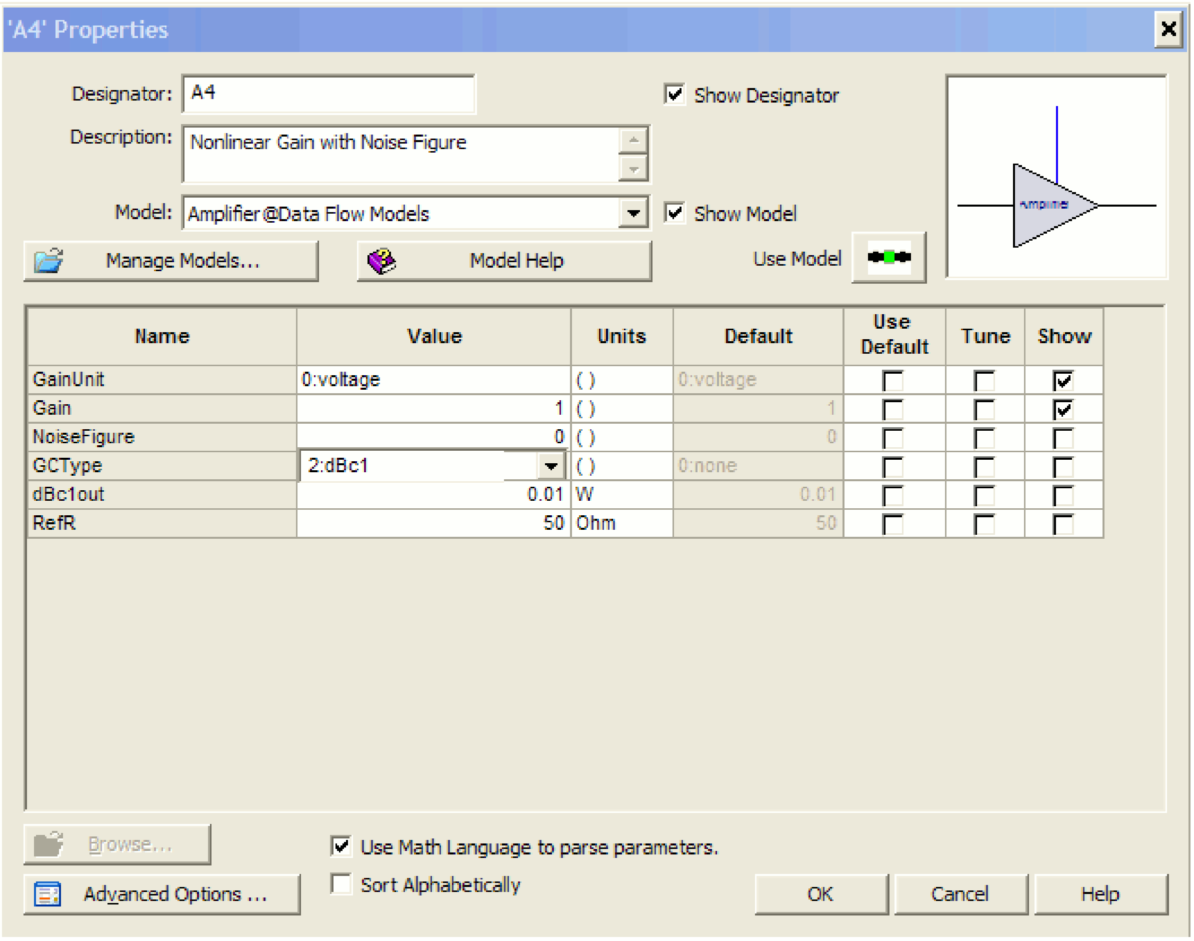
Model: Amplifier@Data Flow Models ☑ Show Model

Name	Value	Units	Default	Use Default	Tune	Show
GainUnit	0:voltage	( )	0:voltage	□	□	☑
Gain	1	( )	1	□	□	☑
NoiseFigure	0	( )	0	□	□	□
GCType	3:TOI+dBc1	( )	0:none	□	□	□
TOIout	0.1	W	0.1	□	□	□
dBc1out	0.01	W	0.01	□	□	□
RefR	50	Ohm	50	□	□	□

☑ Use Math Language to parse parameters.

□ Sort Alphabetically

Setting *GCType* to a non-member e.g. 2:1dBc hides the parameter from view even on the **Properties** table.



'A4' Properties

Designator: A4

☑ Show Designator

Description: Nonlinear Gain with Noise Figure

Model: Amplifier@Data Flow Models

☑ Show Model

Manage Models...

Model Help

Use Model

Name	Value	Units	Default	Use Default	Tune	Show
GainUnit	0:voltage	( )	0:voltage	□	□	☑
Gain	1	( )	1	□	□	☑
NoiseFigure	0	( )	0	□	□	□
GCType	2:dBc1	( )	0:none	□	□	□
dBc1out	0.01	W	0.01	□	□	□
RefR	50	Ohm	50	□	□	□

Browse...

☑ Use Math Language to parse parameters.

□ Sort Alphabetically

Advanced Options ...

OK

Cancel

Help

Note

Defining **Hide Conditions** refers strictly to the table view of parameters and not the visibility of selected parameters on the schematic. Parameters that are hidden by condition are barred from schematic display even if they had the **Show** button checked prior to concealment.

As you simulate this design in the workspace, the default parameter values will be used in the simulation. When you use this design as a model in a part, the part parameters override these default parameter values.

Roles of Sub-Network Model Attributes

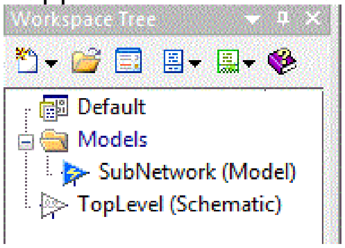
- **Parameters** - These are the parameters the user sees when entering values for this model.
- **Equations** - These are commonly used to manipulate data entered by the users to a format needed by models that appear in the schematic
- **Schematic** - This shows how existing models are visually and electrically connected together and their relationships with each other. The model parameters in the schematic can also use the top level parameters as well as any variable created in the equation block.
- **PartList** - This shows connectivity and part information in a table format.
- **Notes** - This is used for documentation or help for this sub-network.

Run-time Hierarchy - How Parameters get passed

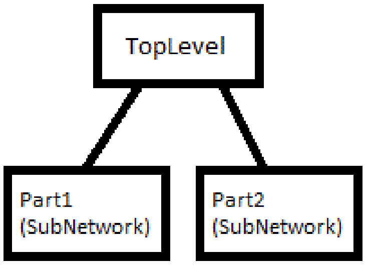
When a simulation is run, a model tree is instantiated that corresponds to the topology of the network you are simulating. This is called the *run-time hierarchy*. In contrast, when you are editing designs in the workspace, you are working in *design-time*. The difference will become apparent shortly.

Each part in your top level design references a model, and an instance of that model is created and set as a "child" of the top-level design when a simulation is run. If one of these children corresponds to a subnetwork model, then each model inside the subnetwork design is instantiated as well, recursively. It is easy to see why this sort of instantiation is necessary - you can have two parts in your top-level design that point to the same model, and they may have different values for their parameters.

Suppose we have a workspace as shown here (ie. this is the design-time hierarchy):



and suppose that TopLevel contains 2 instances of SubNetwork, ie. TopLevel has 2 parts called Part1 and Part2 whose models are both "SubNetwork". When you run a simulation on TopLevel, the following run-time model hierarchy is constructed:



Note that the Equations and Parameters of TopLevel are visible to the model instances of Part1 and Part2, but only at run-time!

It is important to note that when you are looking at the design called SubNetwork (ie. in design-time), and in its schematic you are using parameters defined in the Parameters tab of SubNetwork, the values you see at design-time will correspond to the "Default" values of the parameters as defined in the Parameters tab. This is because you are editing the Model called SubNetwork, but that model can be instantiated many times in your top-level network, and each instance can have different values for the parameters. Since you are editing the design-time model, it has no way of knowing what the values passed to it will be at run-time, and thus just shows the default values that are defined at design-time.

Sweeps

Sweeps are used to create analysis results that are functions of a parameter tuned at several values. A sweep is an evaluation object that controls a specific analysis. It also contains a single tuned parameter that is swept across a range of values specified by the user.

Once a parameter is made tunable (part parameter 'Tune' checkbox has been checked or a question mark '?' has been placed in front of the equation value) it can be selected in the **Parameter Sweep Properties** dialog box. The user specifies hard **Start** and **Stop** values as well as the number of swept points.

The number of swept points can be specified in 1 of 4 ways:

- **Linear: Number of Points**
- **Log: Points / Decade**
- **Linear: Step Size**
- **List**

Once the **analysis** has been selected and the **swept parameter** has been defined the **sweep** will tune the parameter to the first point, run the analysis, and then save the data in a sweep dataset. The swept parameter is tuned to the next value, the analysis is re-ran and the new data will be appended to prior values. This process repeats until the last swept point is reached.

✓ **Hint**
Remember, an **analysis** and **tuned variable** must exist in the workspace before a sweep can be created.

Note
A sweep can control another sweep. Sweeps are also considered an **analysis** and will appear in the **Analysis to Sweep** list. A good example of this are IV curves for a transistor. One sweep will point to the transistor circuit and sweep its supply voltage the other sweep will control the first sweep as the bias current is swept across its range.

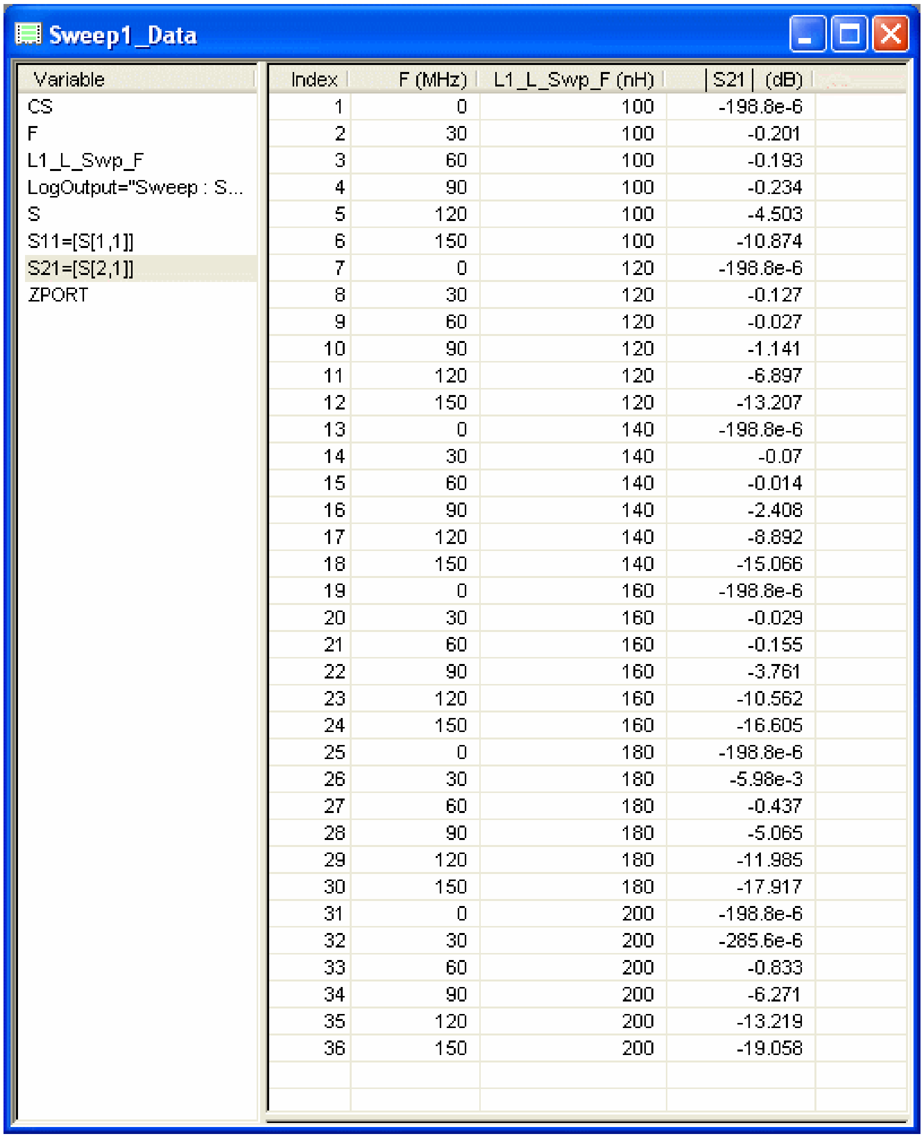
Note
For additional information on sweeps in Spectrasys see *Sweeps of a Path* (sim).

Contents

Understanding Swept Data

The following figure shows the dataset for a modified version of the *Getting Started with Parameters Sweeps* (users) example. This example was modified to reduce the number of linear simulation points from 101 to 6 so the results could be seen in a single figure.

In summary, there is a linear simulation of simple LC low pass filter over 6 frequency points (0, 30, 60, 90, 120, and 150 MHz). A sweep has been created that tunes the filter inductor parameter **L** across 6 values (100, 120, 140, 160, 180, and 200 nH).



Variable
- CS
- F
- L1_L_Swp_F
- LogOutput="Sweep : S...
- S
- S11=[S[1,1]]
- S21=[S[2,1]]
- ZPORT

Index	F (MHz)	L1_L_Swp_F (nH)	|S21| (dB)
1	0	100	-198.8e-6
2	30	100	-0.201
3	60	100	-0.193
4	90	100	-0.234
5	120	100	-4.503
6	150	100	-10.874
7	0	120	-198.8e-6
8	30	120	-0.127
9	60	120	-0.027
10	90	120	-1.141
11	120	120	-6.897
12	150	120	-13.207
13	0	140	-198.8e-6
14	30	140	-0.07
15	60	140	-0.014
16	90	140	-2.408
17	120	140	-8.892
18	150	140	-15.066
19	0	160	-198.8e-6
20	30	160	-0.029
21	60	160	-0.155
22	90	160	-3.761
23	120	160	-10.562
24	150	160	-16.605
25	0	180	-198.8e-6
26	30	180	-5.98e-3
27	60	180	-0.437
28	90	180	-5.065
29	120	180	-11.985
30	150	180	-17.917
31	0	200	-198.8e-6
32	30	200	-285.6e-6
33	60	200	-0.833
34	90	200	-6.271
35	120	200	-13.219
36	150	200	-19.058

Column Name	Description
Index	Row number in the table.
F (MHz)	First independent variable of swept data. In this example, this is the frequency range of the linear analysis. It is repeated for each tuned value of the swept parameter.
L1_L_Swp_F (nH)	Second independent variable of swept data. In this example, this is the tuned value of the inductor parameter **L** at each frequency point of the linear analysis.
|S21| (dB)	Dependent variable for each tuned parameter value (inductor) at each frequency.

Note
The independent variables (in this example **F** and **L1_L_Swp_F**) are also stored in the dataset.

Hint
For additional information on indexing into sweeps see *Using Engineering Language* (users) or *Using Math Language* (users).

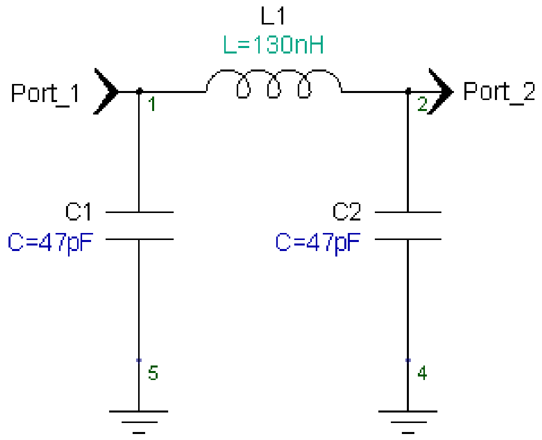
Getting Started with Parameter Sweeps

The steps for creating a parameter sweep are:

1. Create a schematic (or design)
2. Add an analysis
3. Make a variable tunable
4. Create a parameter sweep
5. Run the sweep
6. Plot the data

Create a Schematic

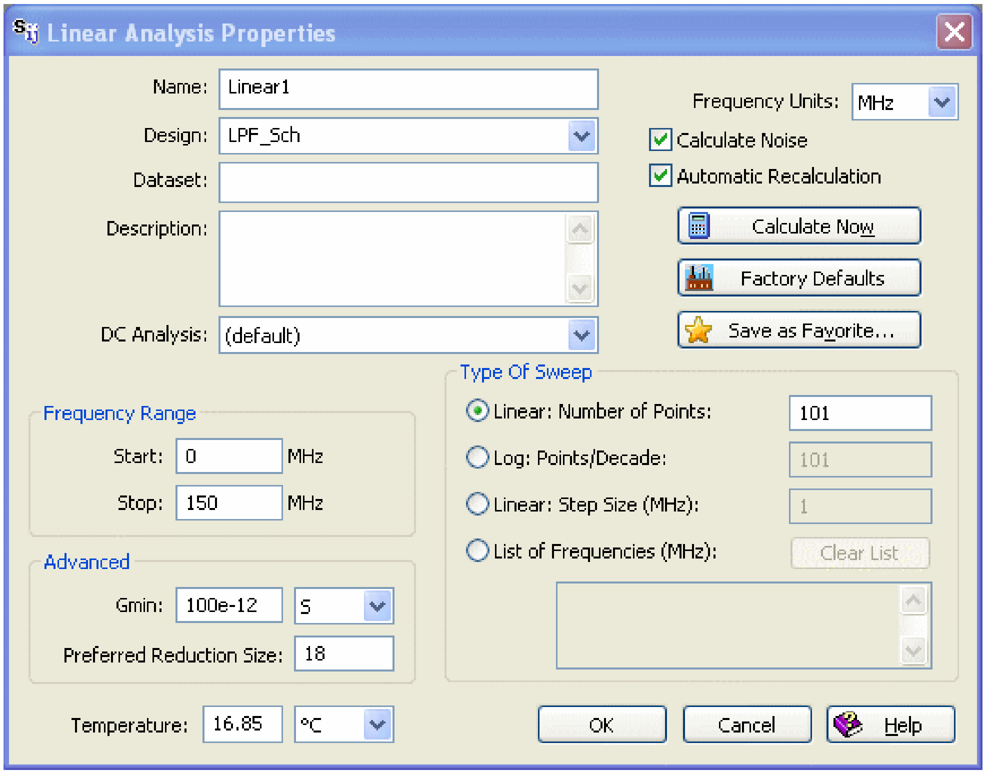
- A simple Lowpass filter can be created as follows (the schematic is named **LPF_Sch**):



Note
See *Creating a Simple Schematic* (users) in the User's Guide for details on schematic creation.

Add an Analysis

- Create the linear analysis



Linear Analysis Properties

Name: Linear1

Design: LPF_Sch

Dataset:

Description:

DC Analysis: (default)

Frequency Units: MHz

☑ Calculate Noise

☑ Automatic Recalculation

Frequency Range

Start: 0 MHz

Stop: 150 MHz

Advanced

Gmin: 100e-12 S

Preferred Reduction Size: 18

Type Of Sweep

● Linear: Number of Points: 101

○ Log: Points/Decade: 101

○ Linear: Step Size (MHz): 1

○ List of Frequencies (MHz):

Temperature: 16.85 °C

- Set the values as shown in the above dialog

Note
See the *Analysis* (users) section in the User's Guide for details on creating an analysis.

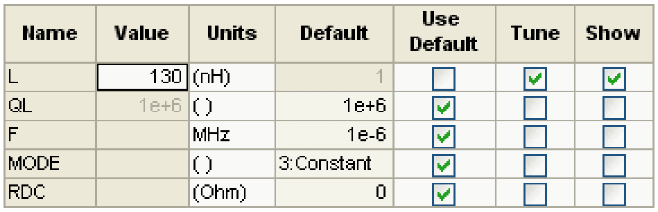
Make Variables Tunable

There are two ways to make variables tunable:

1. Through part properties
2. As an equation variable

Method 1 - **Part Properties**

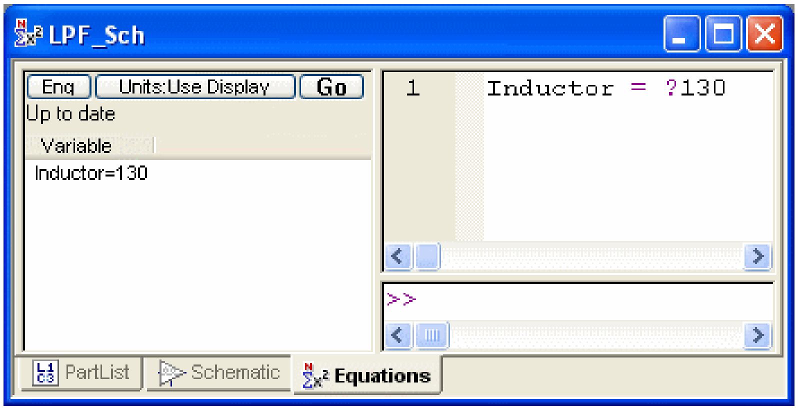
- Check the **Tune** checkbox for the parameter to be tuned



Name	Value	Units	Default	Use Default	Tune	Show
L	130	(nH)	1	☐	☑	☑
QL	1e+6	( )	1e+6	☑	☐	☐
F		MHz	1e-6	☑	☐	☐
MODE		( )	3:Constant	☑	☐	☐
RDC		(Ohm)	0	☑	☐	☐

Method 2 - **Equation Variable**

- Create the variable in an equation block and place a **?** before the value



Inductor=130

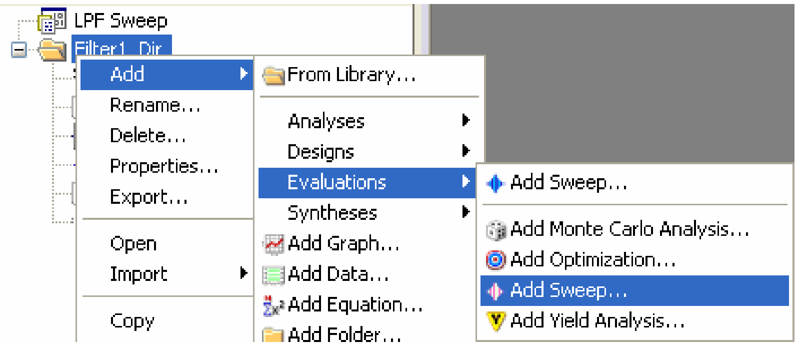
```
1    Inductor = ?130
```

```
>>
```

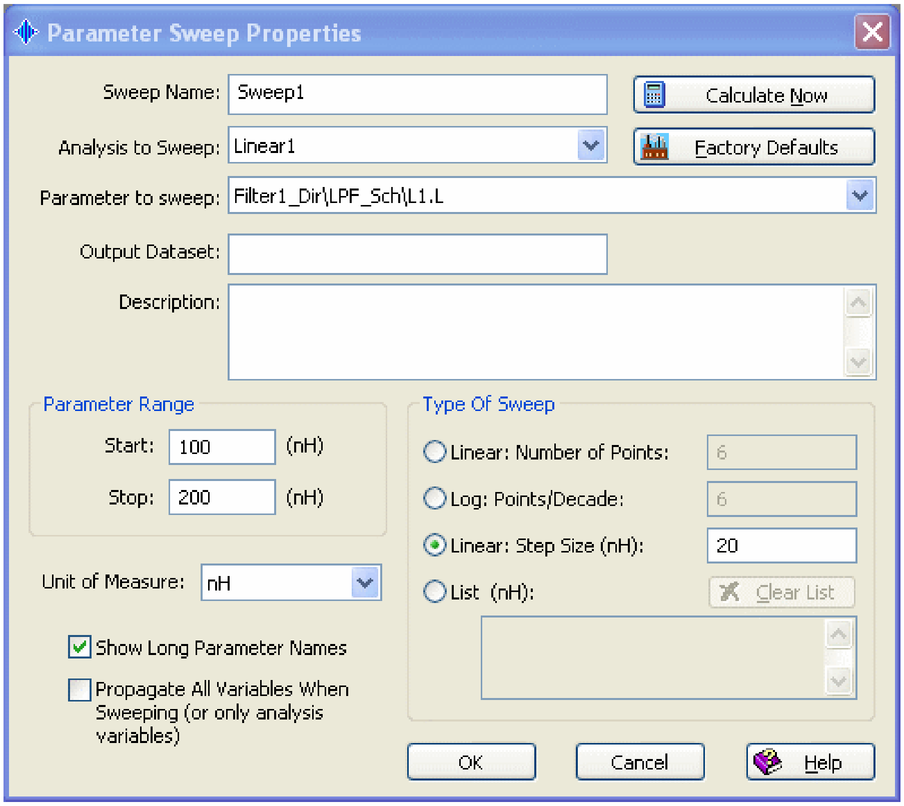
Hint
Remember to set the parameter **value** to the equation **variable name** to link the equation to the part parameter. In this example the variable name **Inductor** would be used for the part parameter **Value** of **L** instead of **130**.

Create the Parameter Sweep

- Add a new sweep to the workspace tree by right clicking on a folder and selecting **Add Sweep** from the submenus.



- Set the sweep properties as follows:



Sweep Name: Sweep1

Analysis to Sweep: Linear1

Parameter to sweep: Filter1_Dir\LPF_Sch\L1.L

Output Dataset:

Description:

Parameter Range

Start: 100 (nH)

Stop: 200 (nH)

Unit of Measure: nH

☑ Show Long Parameter Names

☐ Propagate All Variables When Sweeping (or only analysis variables)

Type Of Sweep

○ Linear: Number of Points: 6

○ Log: Points/Decade: 6

● Linear: Step Size (nH): 20

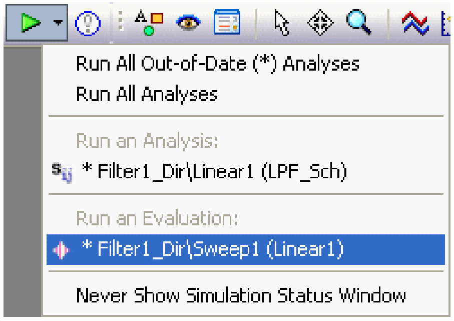
○ List (nH):

Run the Sweep

The sweep can be ran in one of two ways:

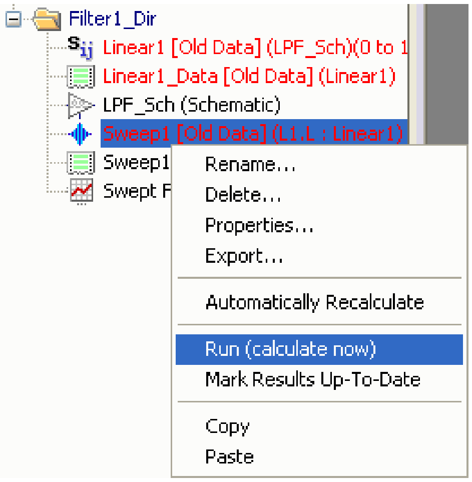
- From the **Play** button on the main toolbar
- From a **right click menu** selection on the workspace tree

Method 1 - **Main Menu Play Button**


Run All Out-of-Date (*) Analyses
Run All Analyses
Run an Analysis:
* Filter1_Dir\Linear1 (LPF_Sch)
Run an Evaluation:
* Filter1_Dir\Sweep1 (Linear1)
Never Show Simulation Status Window

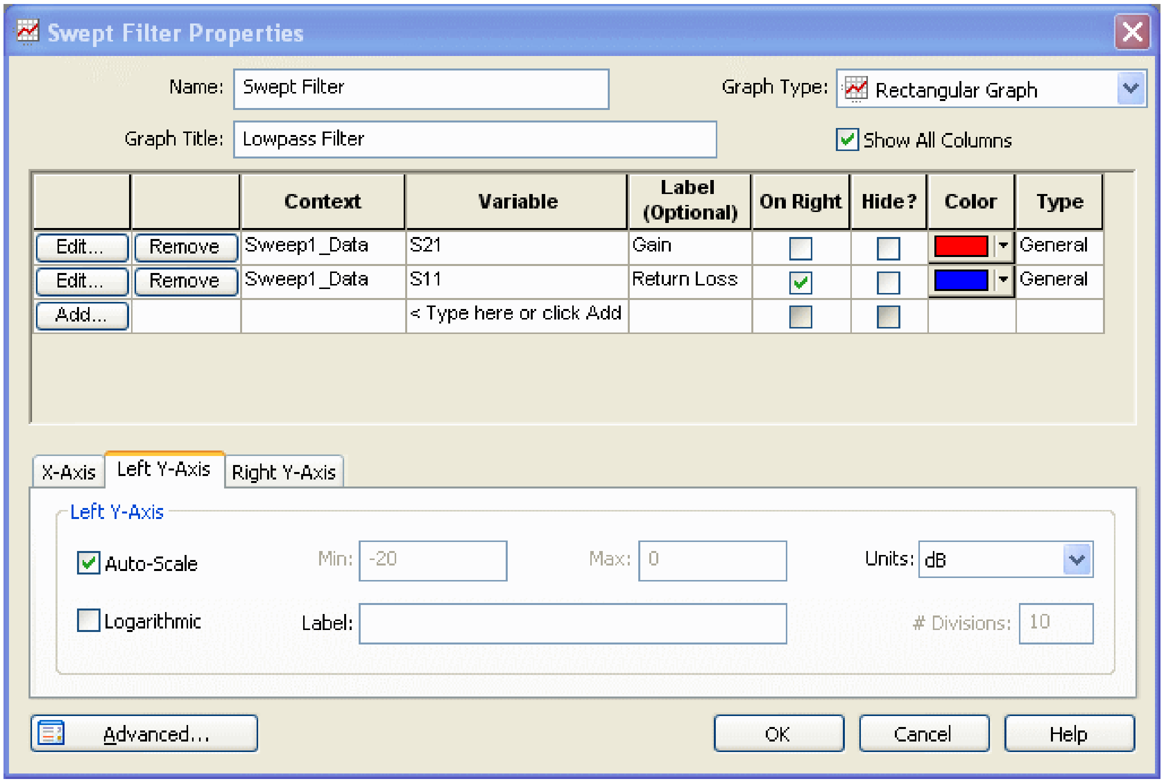
Hint
The **asterisk** * next to the analysis name indicates the analysis data is **out of date**.

Method 2 - **Workspace Right Click Menu**



Plot the Data

- Add a graph to the workspace tree



Name: Swept Filter

Graph Type: Rectangular Graph

Graph Title: Lowpass Filter

☑ Show All Columns

		Context	Variable	Label (Optional)	On Right	Hide?	Color	Type
Edit...	Remove	Sweep1_Data	S21	Gain	□	□	🟥	General
Edit...	Remove	Sweep1_Data	S11	Return Loss	☑	□	🟦	General
Add...			< Type here or click Add		□	□		

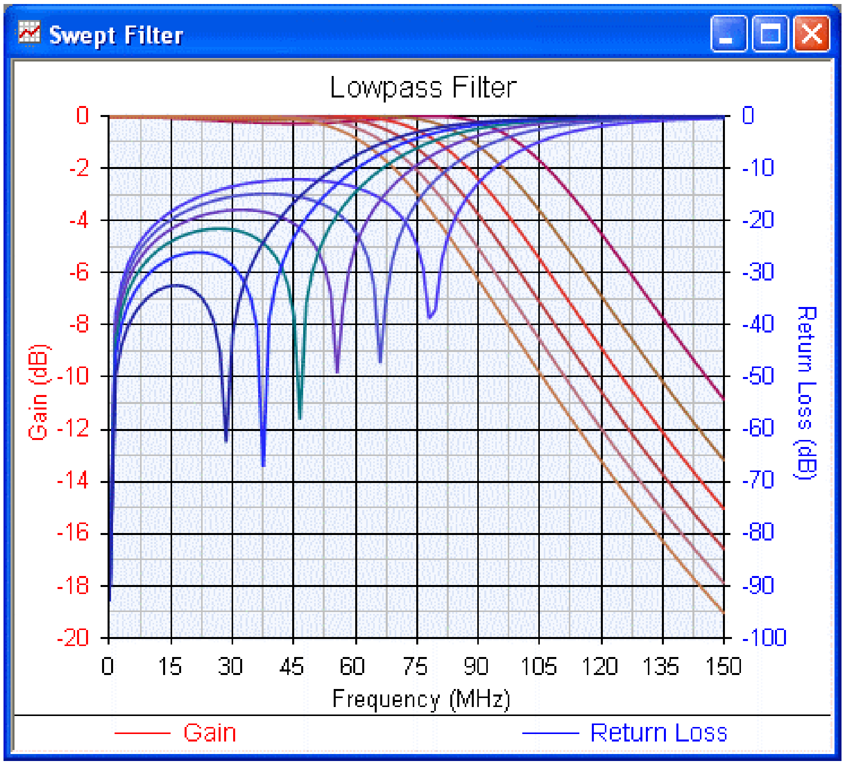
Left Y-Axis

☑ Auto-Scale Min: -20 Max: 0 Units: dB

□ Logarithmic Label: # Divisions: 10

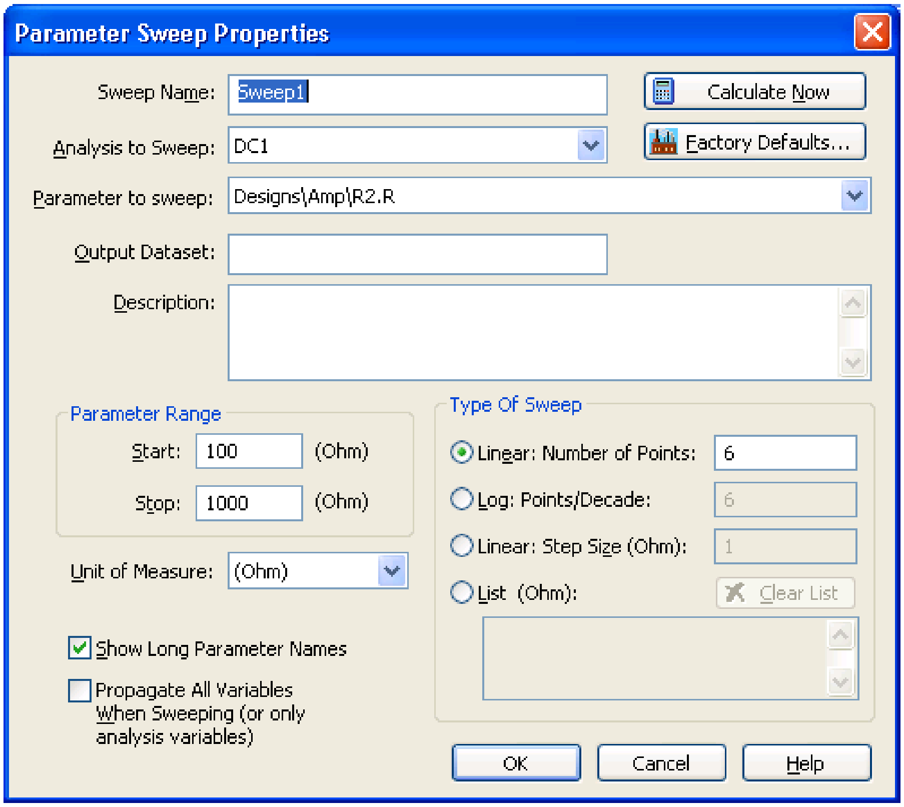
Note
See *Creating Graphs* (users) in the User's Guide for additional information.

- The swept results are shown below



Note
Graphs that contain swept data display a trace for each swept value.

Parameter Sweep Properties



Sweep Name: Sweep1

Analysis to Sweep: DC1

Parameter to sweep: Designs\Amp\R2.R

Output Dataset:

Description:

Parameter Range

Start: 100 (Ohm)

Stop: 1000 (Ohm)

Unit of Measure: (Ohm)

☑ Show Long Parameter Names

□ Propagate All Variables When Sweeping (or only analysis variables)

Type Of Sweep

● Linear: Number of Points: 6

○ Log: Points/Decade: 6

○ Linear: Step Size (Ohm): 1

○ List (Ohm):

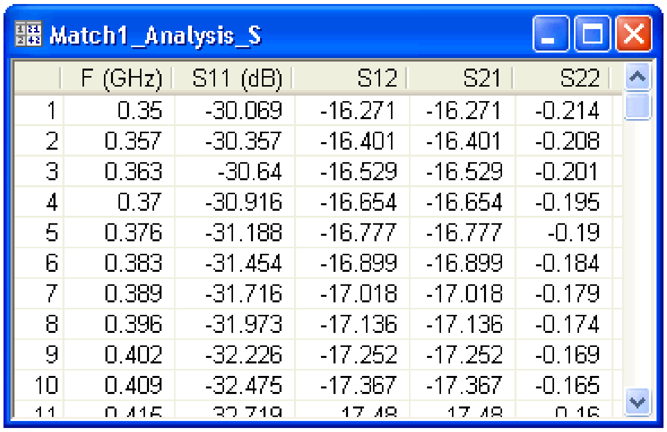
Name	**Description**
Sweep Name	Name of Sweep Evaluation
Analysis to Sweep	Analysis used for the parameter sweep. The selected analysis will be recalculated for each different value of the swept parameter.
Parameter to Sweep	Parameter that gets changed to create the sweep. All parameters defined as tunable are available to be swept.
Output Dataset	Dataset file in which the data is saved. If not specified, the dataset name will be the name of the analysis with "_Data" appended.
Description	Description of the evaluation being run. For documentation purposes only, not otherwise used by Genesys.
Calculate Now	Run the evaluation. Always runs the analysis, regardless of whether or not any changes were made.
Propagate All Variables When Sweeping	User created variables in the source dataset will be swept and aggregated into the sweep dataset.
Show Long Parameter Name	Display the full parameter name (with path) in case you have multiple parameters with the same short name (such as C1.C).
Factory Defaults	Reset all values to their default
Parameter Range	Start. The lower bound (minimum frequency) of the sweep.
Stop	The upper bound (maximum frequency) of the sweep.
Unit of Measure	Unit of measure used for the evaluation
Type of Sweep	Linear: Number of Points. Number of points in entire sweep.
Log: Points/Decade	Number of points in each decade of the sweep.
Linear: Step Size (MHz)	Allows specification of start and stop frequencies, and space between points.
List of Frequencies (MHz)	Allows the explicit specification of analysis frequencies. These points are entered into the List of Frequencies box separated by spaces.

Tables

Tables provide text-based tabular output instead of graphical output. There is only one type of table in Genesys. You can place any measurement in a table. Change the properties of a table using the Table Properties window.

➡ Use Ctrl_MouseWheel to zoom in and out on tables.

Any type of alpha-numeric data (such as S-parameters) may be displayed in a table:



Match1_Analysis_S

	F (GHz)	S11 (dB)	S12	S21	S22
1	0.35	-30.069	-16.271	-16.271	-0.214
2	0.357	-30.357	-16.401	-16.401	-0.208
3	0.363	-30.64	-16.529	-16.529	-0.201
4	0.37	-30.916	-16.654	-16.654	-0.195
5	0.376	-31.188	-16.777	-16.777	-0.19
6	0.383	-31.454	-16.899	-16.899	-0.184
7	0.389	-31.716	-17.018	-17.018	-0.179
8	0.396	-31.973	-17.136	-17.136	-0.174
9	0.402	-32.226	-17.252	-17.252	-0.169
10	0.409	-32.475	-17.367	-17.367	-0.165
11	0.415	-32.719	-17.48	-17.48	-0.16

Contents

- *Creating Tables* (users)
- *Table Toolbar* (users)

Creating Tables

- The easiest way to **create** a new graph is using the *Instagraph Feature* (users).
- The easiest way to **add** an arbitrary measurement to an existing table is via the *Measurement Wizard* (users).
- Tables can also be created manually:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Table**.
2. Enter the measurements you want to display in the Table Properties dialog..

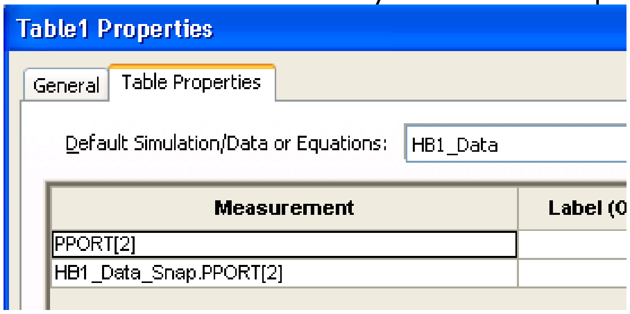
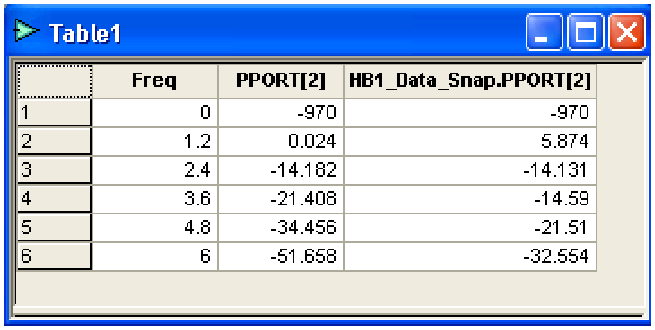


Table1 Properties

Default Simulation/Data or Equations: HB1_Data

Measurement	Label (O
PPORT[2]	
HB1_Data_Snap.PPORT[2]	

3. Click **OK**.
(**note** that the Independent Variable for PPORT2 in the Snap dataset was set to the same as PPORT[2] to remove a second Freq column)



	Freq	PPORT[2]	HB1_Data_Snap.PPORT[2]
1	0	-970	-970
2	1.2	0.024	5.874
3	2.4	-14.182	-14.131
4	3.6	-21.408	-14.59
5	4.8	-34.456	-21.51
6	6	-51.658	-32.554

The ***Label*** entry determines the column labels.

If the data is complex it will often display as dB or magnitude by default. To see the full complex data select a format from the ***Complex Format*** column.

If you want to print a table, copy it to a notes object by using the ***Copy To Notes*** right-click menu entry. This copies the headings and data into an HTML table which you can then copy to Word or other HTML editor or you can just print the Notes. Modify the Notes manually to change fonts or formatting.

➡ New Right-click in the table header to get a popout menu that lets you turn off columns.

Templates

Templates are a very convenient way to get started quickly with a new design. They give you a complete circuit as your starting point. You can also modify templates for your specific program. Many templates are included with Genesys, and you can easily add your own.

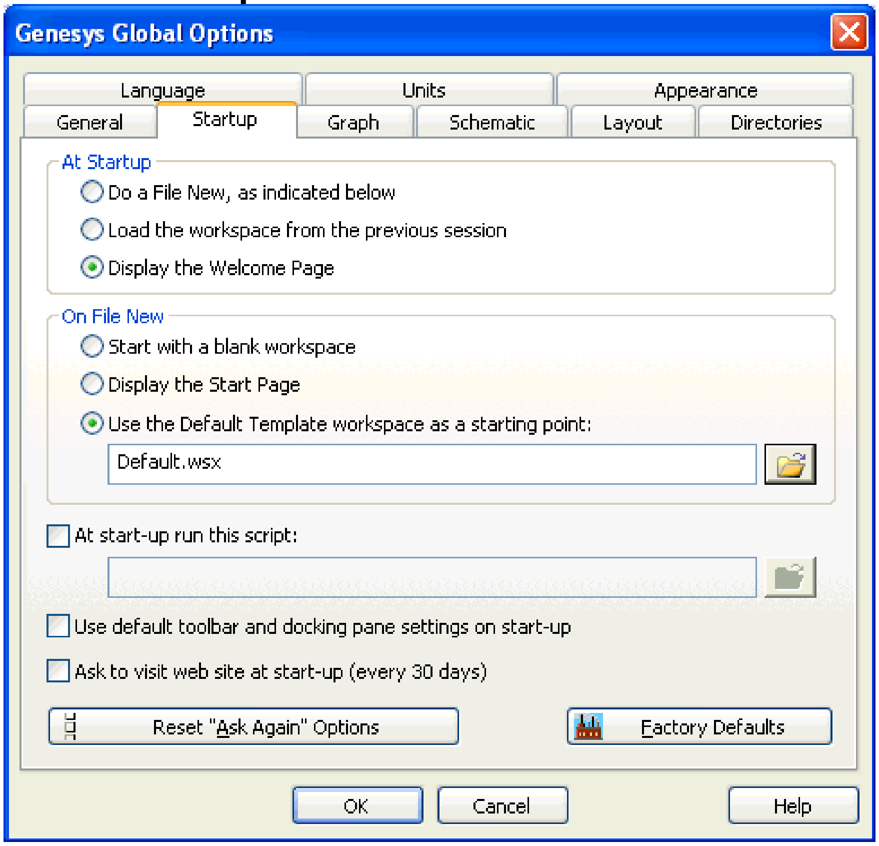
- *Selecting a Genesys Template* (users)
- *Reviewing the Genesys Templates* (users)

Selecting a Genesys Template

The Default.wsx template is automatically loaded whenever a new workspace is created. You can use it or select a different Genesys template.

To select a template to always start with:

1. Click **Tools** on the Genesys menu and select **Options**.
2. Click the **Startup** tab.



Genesys Global Options

At Startup

○ Do a File New, as indicated below
○ Load the workspace from the previous session
● Display the Welcome Page

On File New

○ Start with a blank workspace
○ Display the Start Page
● Use the Default Template workspace as a starting point:

Default.wsx

□ At start-up run this script:

□ Use default toolbar and docking pane settings on start-up

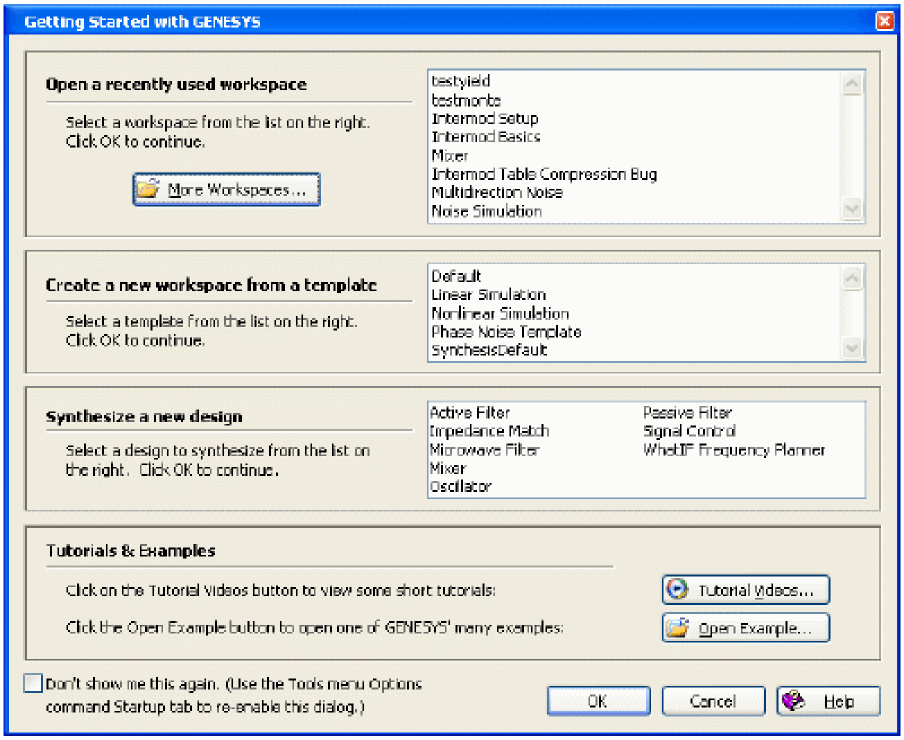
□ Ask to visit web site at start-up (every 30 days)

3. Click the button: **Use the Default Workspace as a Starting Point**
4. Click the folder button and select a template.
5. Click **Open**, and then click **OK**.

> The default templates are workspaces stored in a folder at (default) C:\program files\Genesys20xx.xx\Template. You can place your own template workspaces there, as well.

To select a template once:

1. Click the **Start Page** button ( ) on the Genesys toolbar.



Open a recently used workspace

Select a workspace from the list on the right. Click OK to continue.

Create a new workspace from a template

Select a template from the list on the right. Click OK to continue.

Synthesize a new design

Select a design to synthesize from the list on the right. Click OK to continue.

Tutorials & Examples

Click on the Tutorial Videos button to view some short tutorials:

Click the Open Example button to open one of GENESYS' many examples:

□ Don't show me this again. (Use the Tools menu Options command Startup tab to re-enable this dialog.)

2. In the templates area double-click a template name, such as "Linear Simulation".

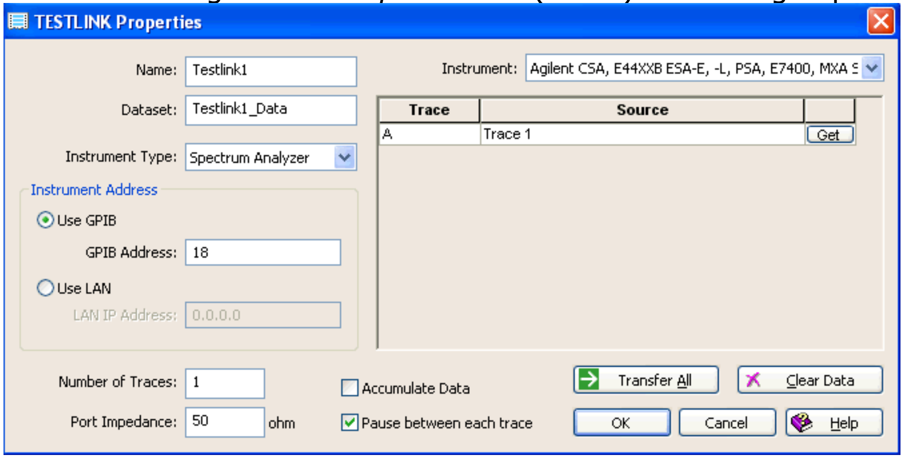
Reviewing the Genesys Templates

The table below lists all of the workspace templates included in Genesys.

File Name	Description
Default.wsx	A blank schematic.
Linear Simulation.wsx	A linear analysis template.
Nonlinear Simulation.wsx	A nonlinear analysis template.
Oscillator Template.wsx	An oscillator analysis template
Phase Noise Template.wsx	A workspace which helps determine the phase noise of an oscillator.
SynthesisDefault.wsx	The base synthesis template, used by Filters, WhatIF, etc.
System Template.wsx	A System Analysis (Spectrasys) template

Testlink

Data from Network Analyzers, Spectrum Analyzers, and Oscilloscopes can be read into the Genesys environment for analysis and display. The interface is called "Testlink" and may be added using the *Workspace Tree* (users).This brings up the Testlink dialog box:



Name: Testlink1

Dataset: Testlink1_Data

Instrument Type: Spectrum Analyzer

Instrument Address

● Use GPIB

GPIB Address: 18

○ Use LAN

LAN IP Address: 0.0.0.0

Number of Traces: 1

Port Impedance: 50 ohm

□ Accumulate Data

☑ Pause between each trace

Instrument: Agilent CSA, E44XXB ESA-E, -L, PSA, E7400, MXA S

Trace	Source	
A	Trace 1	Get

1. **Name** - Name of Testlink analysis.
2. **Dataset -** The name of dataset.
3. **Instrument Type -** Available choices are "Network analyzer" for reading S-Parameters, "Spectrum Analyzer" for reading a power spectrum, and "Oscilloscope" for reading a time-domain waveform.
4. **Instrument** - Specific instrument make and model for the specified instrument type.
5. **GPIB Address** - GPIB address of the instrument. By default Testlink will use the address of the first listener found on the bus.
6. **Lan Address -** LAN address of the instrument.
7. **Number of Ports** - For network analyzers, specifies the number of ports to be measured.For example, if number of ports is 2, then S11, S21, S12, and S22 can be measured.For other instrument types, specifies the number of traces to measure.
8. **Port Impedance** - Port impedance of the selected instrument.
9. **Accumulate Data** - When checked will append an retrieved data into the existing dataset.
10. **Pause between each trace** - When checked with prompt the user to setup the instrument between getting each measurement.
11. **Transfer All** - Transfers all instrument traces into the dataset.This is the equivalent of pressing all of the "Get" buttons in sequence.
12. **Clear Data** - Erases all data from the dataset.
13. **Trace Setup Grid**
    - **Trace** - For Network analyzers, always shows S11, S22, etc.For other instrument types, this field can be edited and is used to give convenient names to your data.
    - **Source** - Allows to specify which instrument channel contains the data for this trace. Additionally, for Network Analyzer, options are provided to make one S-Parameter equal to another, or to make an S-Parameter always zero.This is convenient for passive networks, for example, making S21=S12.
    - **Get** - Transfers one measurement at a time.Often, it will be more convenient to press the "Transfer All" button at the bottom of this dialog.

Contents

- *Testlink Walkthrough* (users)
- *Testlink Hardware Interface* (users)

- *Testlink Registration* (users)

Testlink Hardware Interface

Set-up

1. Connect PC to measurement instrument, such as an oscilloscope, network analyzer, or spectrum analyzer using a GPIB card or RS-232 port. A partial list of supported instruments is found below. There is no limit to the number of instruments. The instruments can be daisy-chained or connected in a "star" configuration. Verify the proper operation of the PC / instrument communication using the software supplied by the interface card manufacturer. A list of available GPIB interface cards is also provided.
2. Load the Genesys software on the PC with Testlink option. Then insert a Testlink icon on the workspace tree and transfer the data into the Genesys workspace. Then compare simulated versus measured data or use the data as models in larger simulations. See the Testlink walk-through for details on using Testlink data in simulations, equations, and other analyses.

Hardware Requirements

PC Requirements:

Minimum System Requirement: Same as Genesys

GPIB card:

National Instruments: type PC-IIA, PC-AT, PC-PCMCIA, PC-USB
Agilent/Hewlett-Packard: HP82335, HP82340, HP82341, HP82350
ComputerBoards Inc.: GPIB card, type ISA-GPIB, ISA-GPIB/LC, ISA-GPIB-PC2A, PCI-GPIB, PCM-GPIB
ines: GPIB-PCMCIA, GPIB-PCI card

> ℹ TEST LINK can also use RS-232 port if the instrument supports it.

Network Analyzers:

Advantest: R3753H, R3764/65/66/67H, R3765/67G
Agilent: E5061/62/70/71A/B ENA Series, E835XA/B, E836XA/B PNA Series, E4991 Impedance Analyzer
Anritsu: Wiltron 360 Series, Wiltron 371/372/373xxA Series, MS462XX Series, MS3401A/B, MS4630B, 54xxA / 541xxA / 56100A Scalar Analyzers, Site Master SxxxA/B
Hewlett-Packard: HP 35660 Dynamic Signal Analyzer, HP 35665/35670A Dynamic Signal Analyzer, HP 3577A, HP 3589A (Network Analyzer Mode), HP 4145A/B Semiconductor Parameter Analyzer, HP 4155/56 Semiconductor Parameter Analyzer, HP 4192 Impedance Analyzer, HP 4194A Impedance Analyzer, HP 4195 (Network Analyzer Mode), HP 4280 Capacitance Meter, HP 4284A Precision LCR Meter, HP 4291A/B Impedance Analyzer, HP 4294A Impedance Analyzer, HP 4352B VCO / PLL Signal Analyzer, HP 4395A Network/Spectrum/Imped Analyzer, HP 4396A Network/Spectrum Analyzer, HP E5100A/B HP 8510, HP 8711-14B/C Series VNA, HP 8720 series VNA (8753E, 8720, etc.), HP 8751 Series VNA, HP 8752A, 8753A/B/C, HP 8756 Analyzer + 8340,83640/650/750 Source, HP 8757 Analyzer + 8340,83640/650/750 Source
IFR: 6800 series (Scalar/Fault Loc'n)
Marconi Instruments: 6210, 6200 (Scalar)
Rohde & Schwarz: ZVR/ZVC/ZVM/ZVK Series, ZVA/ZVB/ZVT Series

Wiltron: 560A Scalar Analyzer / 6600 Source

Spectrum Analyzers:

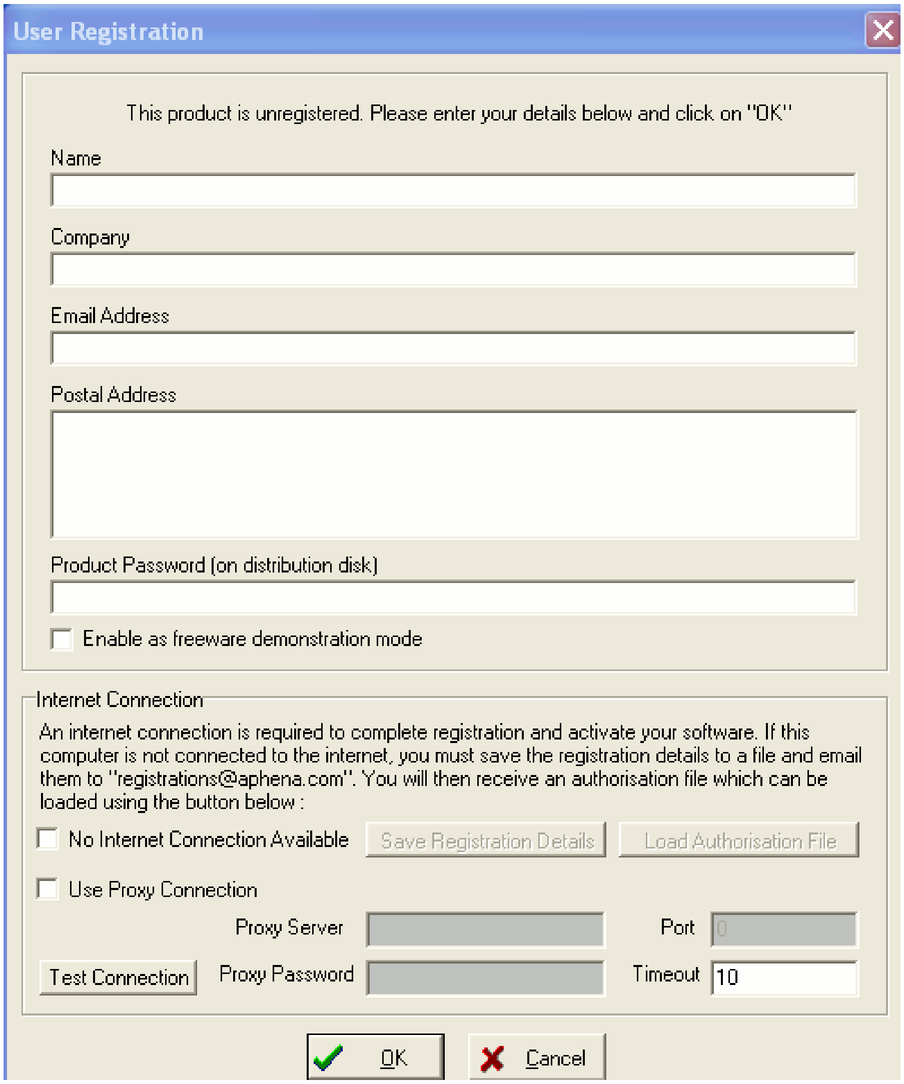
Advantest: R3131 / R3132 / R3162 Series, R3261/3361 Series, R3265/3271 Series, R3267/3273, R3463/3465 Series, R3681 Series, U3641, U4941, U3751, U3771 and U3772, R4131A/B/C/D/N Series,TR4135
Agilent: E44XXB ESA-E, -L, PSA, E7400 Series, N8972/3/4/5A NFA Series Noise Figure Meter, 8960 Wireless Comms Test Set (GSM), 89600 Series Spectrum Analyzer
Ando: AQ6317
Anritsu: MS2602A, MS2650/60, MS2380 series, MS2702A, MS2802A, MS612A, MT8220 UMTS Master, MT8801B Radio Comms Analyzer
Hewlett-Packard: HP 3561A Dynamic Signal Analyzer, HP 3562A Dynamic Signal Analyzer, HP 35660 Dynamic Signal Analyzer, HP 35665/35670A Dynamic Signal Analyzer, HP 3585, HP 3588A/3589A (Spectrum Analyzer Mode), HP 4195 (Spectrum Analyzer Mode), HP 4395A Network/Spectrum/Imped Analyzer, HP 4396A Network/Spectrum Analyzer, HP 8542E / HP 8546A EMI Receiver, HP 8566B, 8568B, HP 8566A, 8568A, HP 8569B HP 8560..65, 8590 Series, L1500A, 70000, HP 85671A Phase Noise Card in HP8560/90, HP 85719A Noise Figure Card in HP859XE, HP 8920/22 Wireless Comms Test Set, HP 8970A Series Noise Figure Meter, HP 8970B Series Noise Figure Meter
IFR: AN940 Series, 2398, 2394, 2395, 2397, 2399, 2399A, 2399B, 6800 series (Spectrum mode)
LG Precision: SA-9270 / SA-7270
Marconi Instruments: 2380 Series, 2390 Series, 2945 Series, 2965 Series
Rohde & Schwarz: CMD55/65 Series, CMS50 Series, CMU200 Series (3GPP FDD), CMU200 Series (Bluetooth), CMU200 Series (400 GSM), CMU200 Series (850 GSM), CMU200 Series (900 GSM), CMU200 Series (1800 GSM), CMU200 Series (1900 GSM), CMU300 Series (400 GSM), CMU300 Series (850 GSM), CMU300 Series (900 GSM), CMU300 Series (1800 GSM), CMU300 Series (1900 GSM), ESCS EMI Test Receiver, FSA/B/M, ESMI Series, FSE, FSIQ, FSP, FSU, FSQ Series, FSH3/6 Series with FSH-K1 option
Scientific Atlanta: SD385
Tektronix: 2711/2712, 2714/2715, RSA3303A/08A Real-Time Spectrum Analyzer, 492P/AP/BP, 494P
Willtek: 9100 Series

Oscilloscopes:

Agilent: EPM-P Series Power Meter, 548XXA, 80000 Infiniium Series, 6000 Series
Boonton: 4400/4500 Power Analyzer
Fluke/Philips: PM3350/55/65/75 Series, PM338XA/PM339XA Series
Hewlett-Packard: HP54120 Series, HP54111D / HP54112D, HP54200A Series, HP54502A Series, HP54520C / HP54540C Series, HP54600/1/2/3, 54610, 54615/6 Series, HP54621/2/4A/D, 54641/2/4A/D Series, HP54645A/D Series, HP54750, HP83480 Series , HP 8990A/8991A Peak Power Analyzer
LeCroy: WaveRunner/WaveMaster/WavePro, SDA, DDA, LC300/LC500/9300
Tektronix: 11000 / DSA60x / CSA Digitiser, 2220 / 2230 / 2232A, 2432A / 2440, 7D20 Digitiser Plug-in, 7854,TDS 200 Series, TDS 300 - 800 Series, TDS 1000 / 2000 Series, TDS 3000 Series, TDS 5000 Series, TDS 8000 / CSA 8000 Sampling Scope
Yokogawa: DL1520 / DL1540 Series, DL1740 / DL7100 / DL7200 Series

Testlink Registration

We use an outside program for Testlink to communicate with instruments. The first time you add a Testlink Analysis you will have to register your copy of Testlink. You must enter your Name, Company, Email Address, Postal Address, and Product Password before you can select **OK**. If you did not receive a Product Password for Testlink when you purchased the product, then you should contact support to get one. The registration form will look like this:



User Registration

This product is unregistered. Please enter your details below and click on "OK"

Name

Company

Email Address

Postal Address

Product Password (on distribution disk)

□ Enable as freeware demonstration mode

Internet Connection

An internet connection is required to complete registration and activate your software. If this computer is not connected to the internet, you must save the registration details to a file and email them to "registrations@aphena.com". You will then receive an authorisation file which can be loaded using the button below :

□ No Internet Connection Available

□ Use Proxy Connection

Proxy Server

Port: 0

Proxy Password

Timeout: 10

Active Internet Connection

To check if you can communicate with the registration server, you can select **Test Connection** and find out. If you have an active internet connection, you must enter all of your information into the window and select **OK** to register your copy of Testlink.

Use a Proxy Connnection

If you are behind a firewall you have the option to setup a proxy connection. To do this, you will need to check **Use Proxy Connection** and enter your proxy information along with all of your registration information. You can select **Test Connection** to see if you are communicating with the server or not. If you are, then you can hit **OK** to register.

No Internet Connection

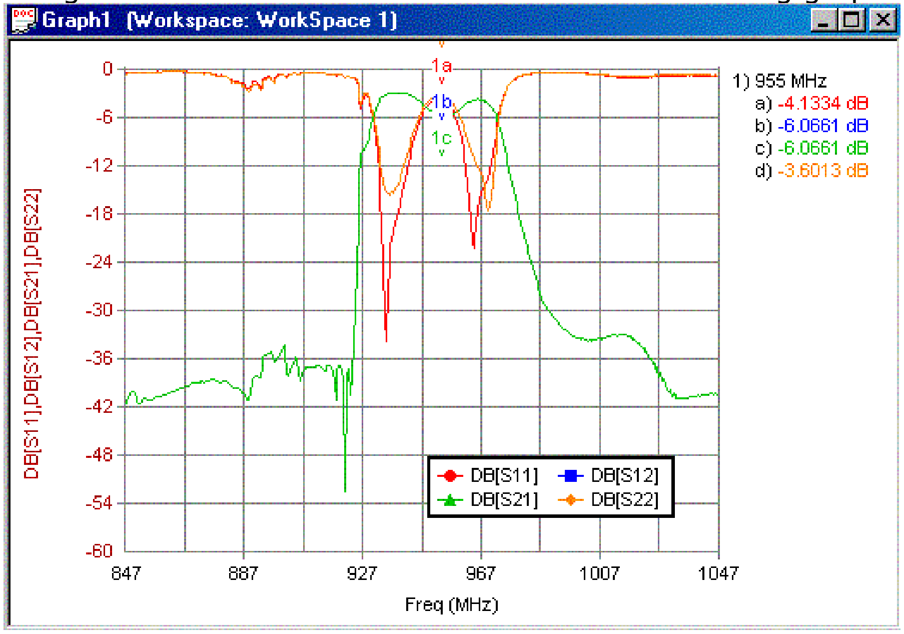
Once you have entered all of registration information, then you must check **No Internet Connection Availible** and then select **Save Registration Details**. Your information will be save as a .rgn file which you must send to the email address registration@aphena.com. Use the subject title "Registration" in your email.

After you send your .rgn file, you will receive an email containing a .aut file that has the same name as your .rgn file. Bring the registration window back up for Testlink and check **No Internet Connection Availible**. Then select **Load Authorization File** and find your .aut file. When you load the .aut file all of your information will be loaded into the entry fields. After you've successfully loaded your .aut file you can select **OK** to complete the registration.

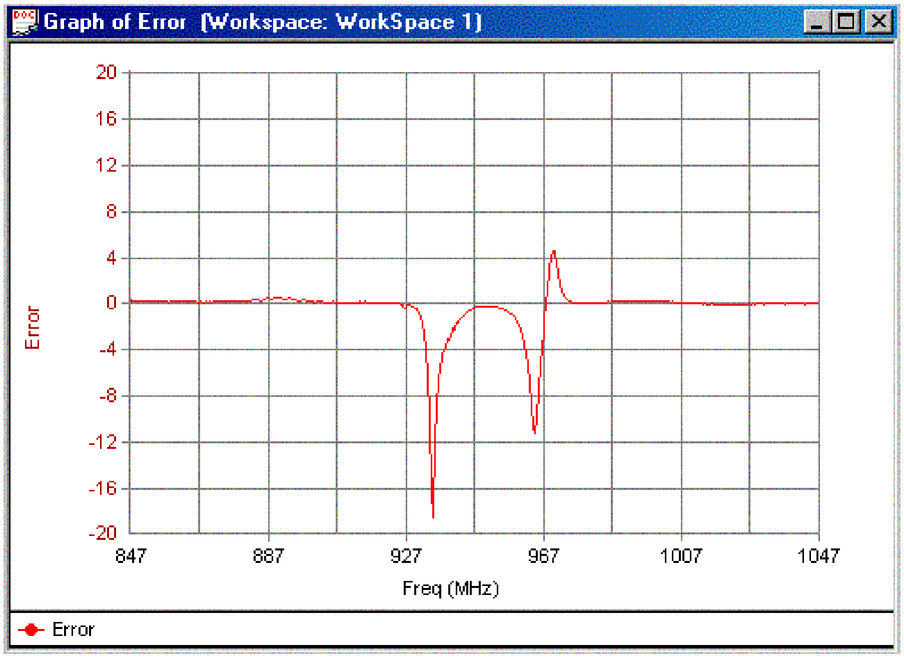
Getting Started with Testlink

The purpose of the Testlink feature of Genesys is to import data from instruments. This allows measurements to be compared with the models which may have been used to develop the network being tested. Consider the case of a two port network being tested with a network analyzer. Typical device data is available in the Testlink demo mode. If the demo mode is selected when Testlink is installed, then Testlink can be exercised as if the PC were connected to an instrument.

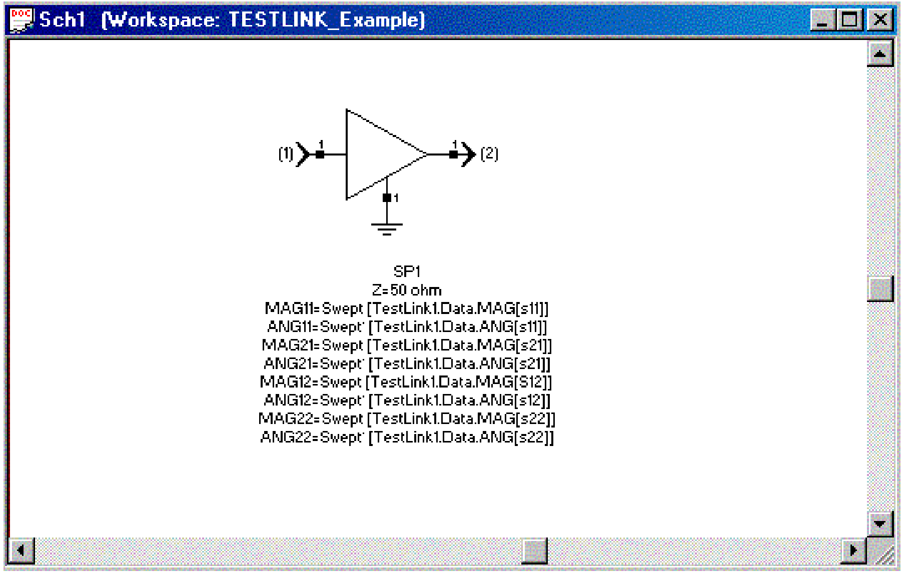
In the add menu select "Add Testlink"., using the name "TestLink_Data". For instrument select: Network Analyzer. For Instrument, use any model. Then under Outputs, add a Rectangular Graph. Enter: S11, S12, S21, S22. Right click on "TestLink_Data" on the workspace tree and select "Transfer All". A dialog box appears for each of the four traces, click OK for each. Another way of transferring data is to open the TestLink dialog box and using the "Get" or "Transfer All" buttons. The resulting graph is as shown below.



This data can also be used in Equations or in Optimization Targets, such as: db(TestLink_Data.s11) for the magnitude of s11 in db. Add a Rectangular Graph, enter a name. In the dialog box, make sure the Data Source is Testlink_Data, then enter "db(S11) - db(S22)" on the first line. Select OK The error between s11 and s22 is then plotted.

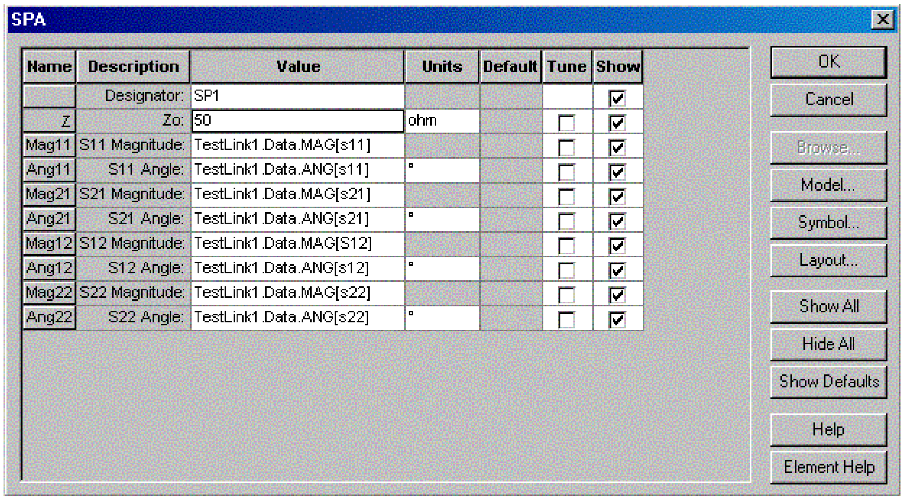


Another use is in a schematic as two port part. There are currently two approaches. The first is to create an SPA part where each of the components is an part of the Testlink data. This approach is shown in the schematic below.

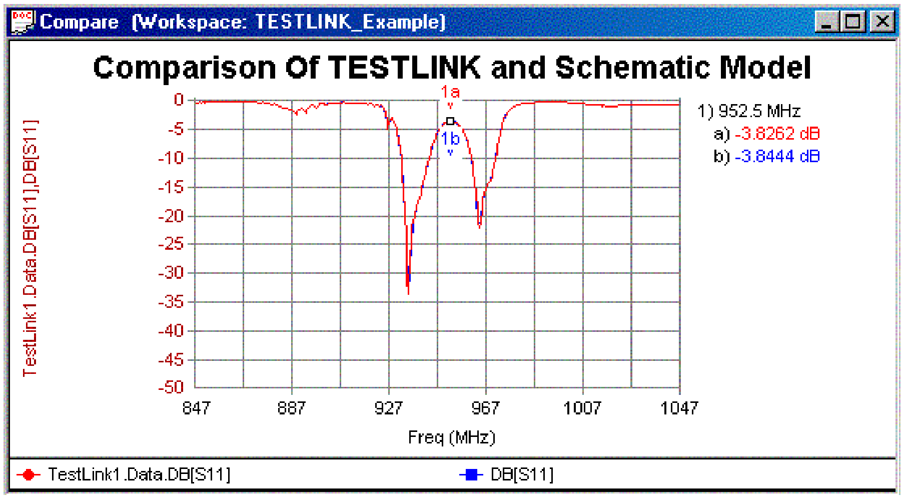


SP1
Z=50 ohm
MAG11=Swept [TestLink1.Data.MAG[s11]]
ANG11=Swept' [TestLink1.Data.ANG[s11]]
MAG21=Swept [TestLink1.Data.MAG[s21]]
ANG21=Swept' [TestLink1.Data.ANG[s21]]
MAG12=Swept [TestLink1.Data.MAG[S12]]
ANG12=Swept' [TestLink1.Data.ANG[s12]]
MAG22=Swept [TestLink1.Data.MAG[s22]]
ANG22=Swept' [TestLink1.Data.ANG[s22]]

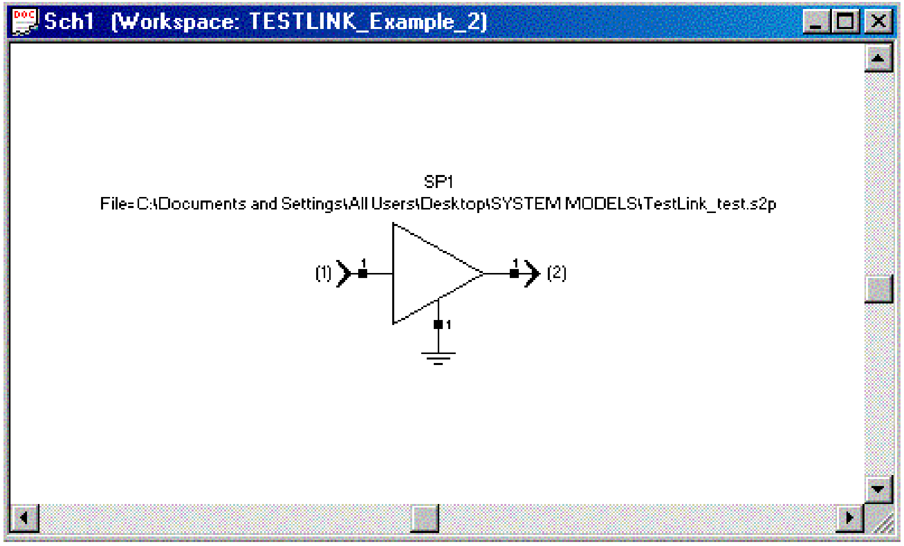
If the data is updated by getting a new set of data from the instrument, the data in this part will be automatically updated. The data from the schematic is plotted along with the original Testlink data. Note that the two plots are essentially the same.



Name	Description	Value	Units	Default	Tune	Show
	Designator:	SP1				☑
Z	Zo:	50	ohm		☐	☑
Mag11	S11 Magnitude:	TestLink1.Data.MAG[s11]			☐	☑
Ang11	S11 Angle:	TestLink1.Data.ANG[s11]	°		☐	☑
Mag21	S21 Magnitude:	TestLink1.Data.MAG[s21]			☐	☑
Ang21	S21 Angle:	TestLink1.Data.ANG[s21]	°		☐	☑
Mag12	S12 Magnitude:	TestLink1.Data.MAG[S12]			☐	☑
Ang12	S12 Angle:	TestLink1.Data.ANG[s12]	°		☐	☑
Mag22	S22 Magnitude:	TestLink1.Data.MAG[s22]			☐	☑
Ang22	S22 Angle:	TestLink1.Data.ANG[s22]	°		☐	☑



The second approach is to generate an s-parameter data file with the Testlink data. From the FILE menu, choose "Write an s-parameter file", and enter an appropriate name such as "Measured Data". Under Source of data select: TestLink_Data". In a new schematic (Sch2), select "two port" from the Linear Toolbar, and add to the schematic as shown. Under file, "Browse" and find the "Measured Data" file previously generated. Enter in the space provided.


Sch1 (Workspace: TESTLINK_Example_2)
SP1
File=C:\Documents and Settings\All Users\Desktop\SYSTEM MODELS\TestLink_test.s2p
(1)
1
1
(2)
1

The resulting s-parameters are again the same as the measured data.

The only drawback to the second approach is that if new data is downloaded from the instrument, the file must be regenerated.

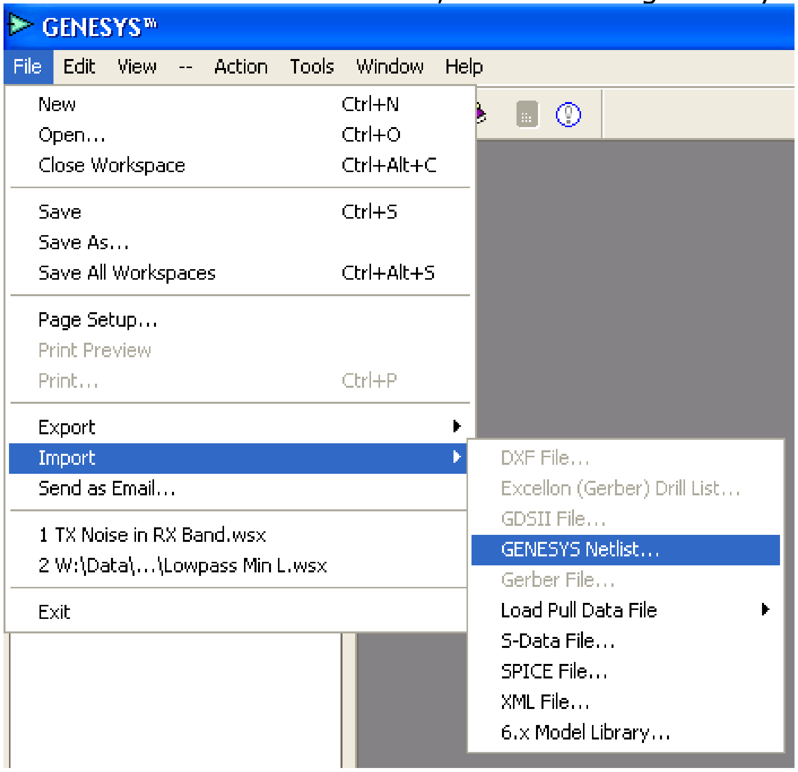
Text Netlists

In Genesys, text files can be used instead of schematic files. These text files are often referred to as a "netlist" or "nodal netlist", since they describe a network's nodal connections. Text netlist files may be imported into the workspace (see the section entitled *Importing Genesys Netlists*).

Contents

Importing Genesys Netlists

Importing old Genesys Netlist files (*.ckt) can be accomplished by selecting the Import submenu under the File menu, then selecting Genesys Netlist... This is illustrated below:



Circuits described in netlists will be converted to Designs containing appropriate components in their part lists. The netlist text corresponding to each design is placed in a Notes tab in the design window.

> The netlist text is for reference only. Changing the netlist text does not change the part list. Changes to the circuit can be made by using the part list user interface.

Seeing Text Equivalents for Schematics

The examples shipped with Genesys include schematic designs, not text netlists. However, you can very easily see the netlist for any of the included example schematics: right-click on the schematic node and select "Place Netlist in Notes". For instance, the text for the AMP schematic in "Examples\Amplifier\Balanced Amp.wsp" is:

```
TERM(50,50)
TWO 1 2 3 FILENAME=AT41586.825 'Q1
SUB Default
MLI 5 2 W=40 L=200 '!1
MCR 7 1 8 9 WT=W6 WC=W2 '!2
MVH 3 0 R=20 T=1 '!3
MVH 3 0 R=20 T=1 '!4
CAP 14 5 C=27 'C1
RES 8 14 R=680 'R1
MLI 2 15 W=W3 L=L3 '!7
MTE 15 16 17 WT=W3 WS=W4 '!8
MLI 17 18 W=W4 L=L4 '!9
CAP 16 19 C=27 '!10
MEN 18 0 W=W4 '!11
MLI 20 9 W=W2 L=L2 '!12
CAP 21 7 C=27 '!13
MEN 20 0 W=W2 '!14
DEF2P 21 19 AMP
```

Shunt-C Coupled Netlist Example

```
TERM(25,75)
```

For complex or frequency dependent terminations, a filename can be used in place of a number. For example, to use 50 ohms at the input of a two-port and ANTENNA.RX at the output, use the following statement:

```
TERM(50,ANTENNA.RX)
```

This assumes that ANTENNA.RX is in the same directory as the workspace. If it is not, you must specify a complete path in the TERM statement.

The next line in the netlist is:

```
CAPQ 1 0 C=47 QC=1000
```

This line describes a *capacitor with Q (CAPQ)* (part) connected between nodes 1 and 0 (0 being ground). The schematic above shows a picture of these connections. The capacitor has a value of 47pF (C=47) and a Q of 1000 (QC=1000). (QC is optional and defaults to 1 million if not specified.) Components can be named by specifying a NAME= parameter, but this is optional. They can be useful when making multiple components with identical parameters as was done later in this file.

```
SLC 1 2 L=1340 C=2 QL=120 QC=1000
```

This line specifies a *Series L-C network (SLC)* (part) connected between nodes 1 and 2. Since this network defines both the inductor and the capacitor, node 3 shown in the schematic above is not needed. The inductor is 1340nH (L=1340) with a Q of 120 (QL=120) and the capacitor is 2pF (C=2) with a Q of 1000 (QC=1000). This component is not named since it will not be reused later.

```
CAPQ 1 0 C=47 QC=1000
```

This line defines another capacitor with Q.

```
DEF2P 1 2 RESONATE
```

This line finishes the description of the resonator network and names the circuit. It reduces the components described above to a two-port network. Node 1 is the input of the circuit and node 2 is the output. The defined network is assigned the name "RESONATE".

Each network that you define must end with a DEF *n* P line, where *n* is the number of ports on the network. If you want to reuse (make a duplicate of) a network, you can use the name on a line with three node numbers (input, output, and reference ground). This is similar to named components but reuses the entire network. Our example did not reuse the RESONATOR network.

For information on plotting simulation results, see the Outputs section.

Substrates

To use substrates in a Netlist, use a SUB statement:
SUB *substrateName*
This statement applies to all parts below it until another SUC statement. *substrateName* refers to a substrate in the Workspace Window tree. See the Substrates section in this manual for more information on entering substrates.

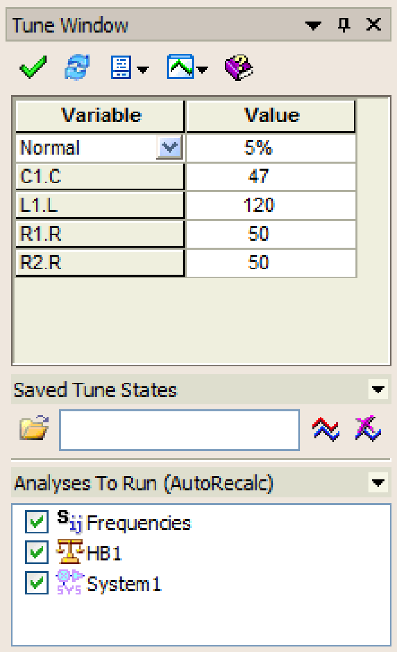
Units In Netlists

The units used in Netlists are:

- Resistance: ohms
- Inductance: nanohenries
- Capacitance: picofarads
- Conductance [1/Resistance (G)]: Siemens (mhos)
- Time: Nanoseconds
- Frequency: Megahertz
- Electrical Length: Degrees at the specified frequency
- Physical Length: Specified in substrate (default: mm)

Tuning Variables

One of the most powerful features of Genesys is real-time tuning of values in variables. You can use tuned variables almost anywhere in Genesys, including part parameters. See almost any of our examples for tuned variables. Tuned variables are listed in the Tune Window as shown:



Variable	Value
Normal	5%
C1.C	47
L1.L	120
R1.R	50
R2.R	50

☑ Frequencies
☑ HB1
☑ System1

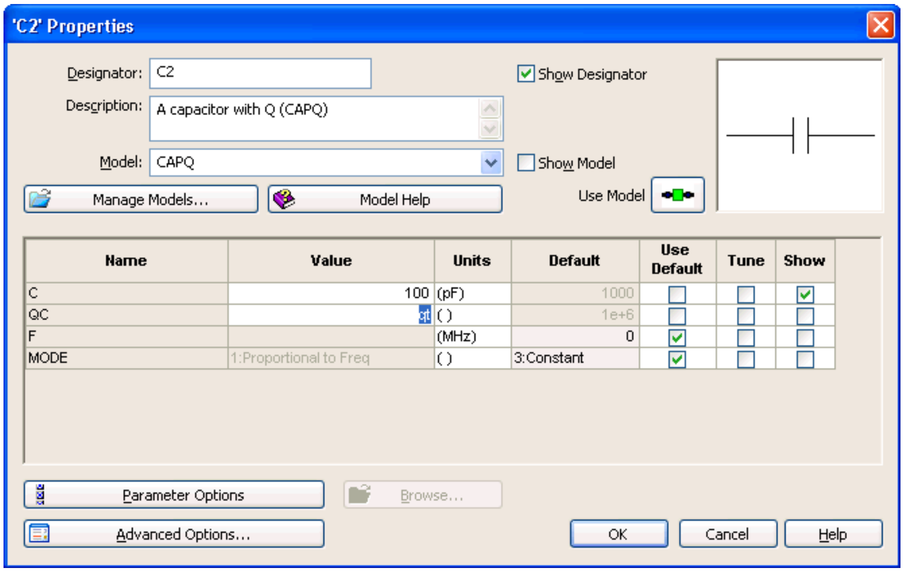
Any numeric parameter in a part can be made tunable. You can tune the value of a variable or use *Gang Tuning* (users) to adjust a value which is used in more than one place. Genesys lets you dynamically tune variables to determine whether your design meets its requirements. You can do this by entering different values for a specific variable and simultaneously viewing the response in a graph. Continue adjusting values and viewing the graph until you get the desired response.

Contents

Gang Tuning

Another common task in tuning is to adjust more than one parameter at the same time. This is called Gang Tuning and the easiest way to do it is with an equation variable.

In the following figure, an equation variable for the Capacitor Q named *qt* has been setup.



Designator: C2

☑ Show Designator

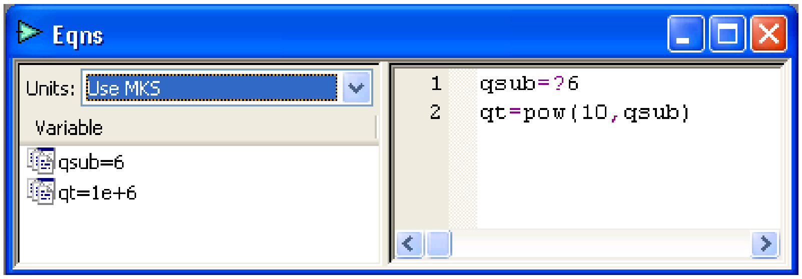
Description: A capacitor with Q (CAPQ)

Model: CAPQ

☐ Show Model

Name	Value	Units	Default	Use Default	Tune	Show
C	100	(pF)	1000	☐	☐	☑
QC	qt	( )	1e+6	☐	☐	☐
F		(MHz)	0	☑	☐	☐
MODE	1:Proportional to Freq	( )	3:Constant	☑	☐	☐

By selecting the equation from the workspace tree you can see the details of the variable.



Units: Use MKS

Variable

qsub=6

qt=1e+6

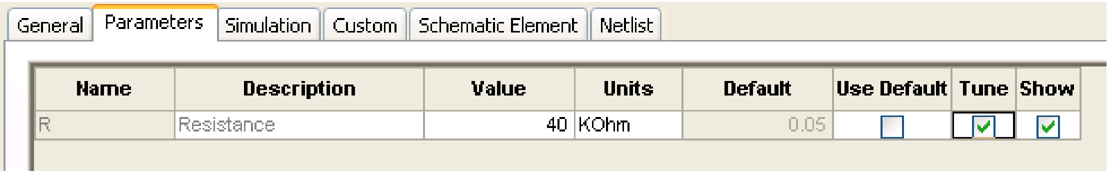
```
qsub=?6
qt=pow(10,qsub)
```

The variable qsub is defined with a ?6. The 6 is the starting value and the ? syntax makes the variable tuneable. The variable qt is defined as qsub to the 10th power which will tune in an exponential form.

For more information about Equations and setting variables tunable from Equations, please refer to the *Using Equations* (users) section.

Making a Part Parameter Tunable

1. Double-click a part on any schematic; this will bring up the Part Properties dialog.
2. Click the **Tune** check box next to any parameter you wish to be tunable.
3. Click **OK**



Name	Description	Value	Units	Default	Use Default	Tune	Show
R	Resistance	40	KOhm	0.05	☐	☑	☑

4. Notice that the variable(s) to be tuned have now changed color on the schematic and appear in the Tune window. To tune this parameter, use any of the methods discussed in *Setting Tuned Values* (users).

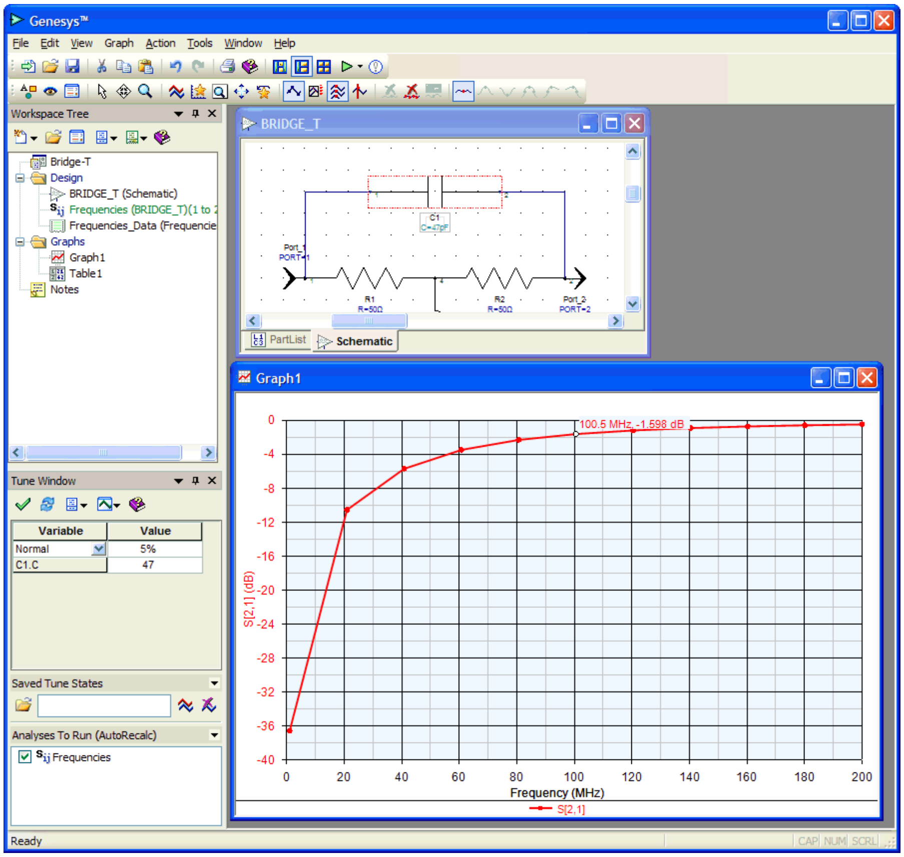
Note that part parameters can actually be selected for tuning in a number of ways:

- Double-click the part and check the Tune box next to a parameter value (as described above).

- Click the part to select it, then select **Make Components Tunable** from the Schematic menu. This sets the first parameter of the part to be tunable.
- Click the part to select it, then click the **Make Tunable** schematic toolbar button, which **toggles** the tunable setting of the first parameter.
- Parameters can be marked for tuning via on-screen editing: click a part parameter, unfold the parameters window (if necessary, use the "unfold" « button), and click in the first cell column. A 'T' indicates that a parameter is tunable.
- To mark many items tunable at once, try this approach:
  1. In the Tune Window, click the **Variable Options** ▾ button
  2. Select **Select Variables** from the drop down menu, which will show a comprehensive list of **everything** which can be tuned.
  3. Check the items you wish to tune.
  4. Click **OK**.

An Example: Making C1.C tunable

1. First, load the **Bridge-T** example and rearrange your screen so that it looks like the screenshot below. Hide the blue group delay trace on the graph.
2. Double-click part C1 and check the Tune box next to a parameter value.
3. Once C1 is tunable, the C=47pf line should turn teal-colored and the Tune Window should now have a C1.C entry. All of the analyses turn red because the schematic has changed.



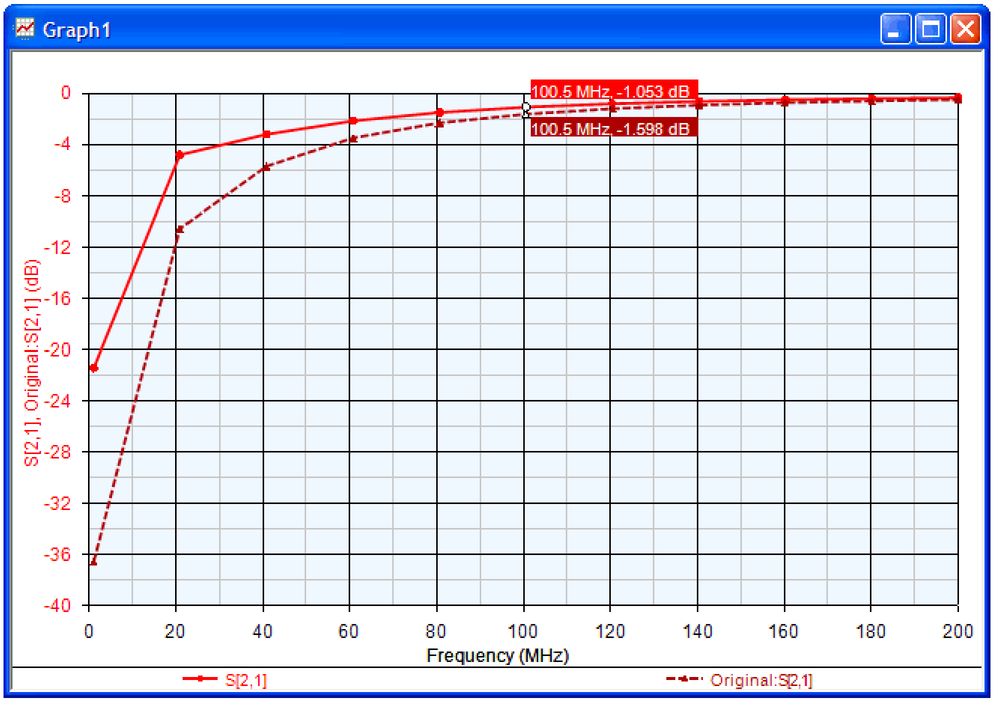
Variable	Value
Normal	5%
C1.C	47

Actually changing a value

Tune the C1.C value by selecting it; click in the grid box where it says 47. Then, do any of the following:

- Roll the mouse-wheel to tune up or down

- Press the PgUp or PgDown key to **tune** up or down
- Click the up/down arrows in the grid box to **move** up or down
- Or type a new value and press enter
-
  - After typing a new C value of 500 and pressing enter, we see the following display:
  - The dashed, dark red trace is the original simulation result S21. It is dashed because it is a "checkpointed" trace.
  - The bright red is the new tuned simulation result.



Save often

This is always a good idea. Saved workspaces remember all the tune settings, including what you were tuning, what runs when tuned, and which graphs to update. Click the save button (the diskette icon in the main toolbar) to save.

Tuning Options

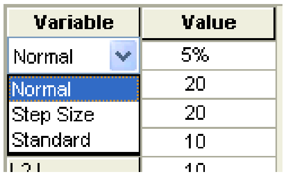
To assist with the tuning of all the various types of variables you may need to adjust, there are a number of tuning options, which control how and what is tuned, when and what is updated, etc.

Specifying how values are tuned up/down

The tuning percentage controls the amount values are stepped when tuning. Whenever a variable is tuned up or down, a tuning ratio is used to calculate the new value. You can use the Tune window to adjust the Normal and Standard options by percentages. The Step Size option uses decimal values for adjusting.

1. Click the first box in the Variable column in the Tune window.
2. Select an option from the list. There are three options which control tuned variables values:
   - **Normal** – This option is steps the tuned value by the specified percentage, and is unrestricted. For lumped parts, such as resistors and capacitors, values between zero (0) and infinity are possible. You can use this option to determine

the theoretical optimum values. This typically increments the value by 5%.
- **Step Size** – This option adds or subtracts the specified step-size to the parameter. For example, if the step-size value equals 0.5, then the allowable parameter values are 0.5, 1.0, 1.5, and so on.
- **Standard** – This option uses only standard values for lumped circuit elements, such as 1.2, 3.3, 4.7, 5.6, and so on. The tuning percentage is shown in the first box in the Value column. This controls the amount values are stepped when tuning.



Variable	Value
Normal	5%
Normal	20
Step Size	20
Standard	10

Tip: You can press F6 to decrease or F7 to increase the tuning percentage by a factor of 2.

Setting Tuned Values

You can set tuned values in a number of ways. Begin by selecting the value you want to change (click in the grid box cell holding the value), then

Click the scroll arrows:

- Click the up arrow to increase the value, the down arrow to decrease the value.
- The analyses you've selected will run will run automatically.

Scroll the mouse wheel:

- Roll the mouse wheel to scroll up and down to tune the value up and down.
- The analyses you've selected will run will run automatically.

Use the keyboard:

- Press the PgUp or PgDown key to **tune** up or down
- The analyses you've selected will run will run automatically.
- Click the up/down arrows in the grid box to **move** up or down to an adjacent grid cell.

Direct entry

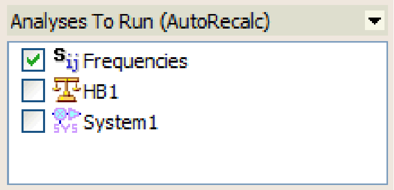
- Type a new value in the Value box for the variable.
- When you press the enter key it will be entered and the analyses you've selected will run.

When values have been set and you want to save a set under a new name just type the name into the name entry field and click the save settings (diskette) icon. If you want to save the current state of graphs, click the graph checkpoint button to create a named checkpoint.

Quicker Tuning: don't tune more than you need

- In the Tune Window, only enable the Analyses that you want automatically recalculated after you tune.
- Check / uncheck analyses. When checked, they will automatically recalculate (so they

will run while tuning).

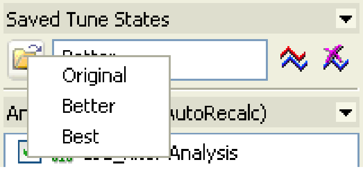


Analyses To Run (AutoRecalc)

☑ Frequencies
☐ HB1
☐ System1

Reverting Tuned Values

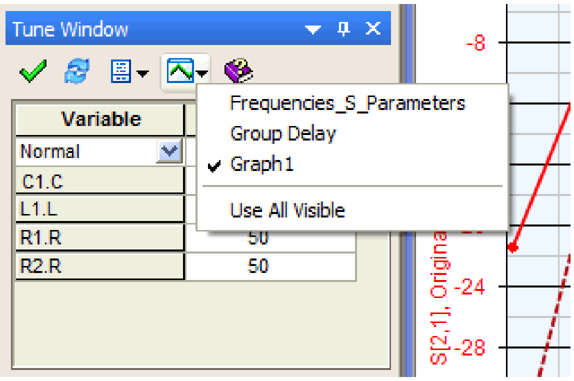
- Genesys lets you revert to your original values if you do not want to keep the newly tuned values.
- Click the **Use These Settings** button and select a named set. The set named Original is the original settings.



Checkpoints

A checkpoint is a saved intermediate point. In a graph, it is usually a dashed trace showing potentially good values.

- Click the **Select Graphs** dropdown button and select (check) only the graphs you want checkpointed while tuning, as shown below

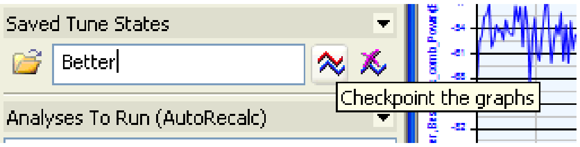


Variable	
Normal	
C1.C	
L1.L	
R1.R	50
R2.R	50

- Or, check **Use All Visible** to use all visible graphs. The graph list dynamically changes as you open and close graphs.

To establish checkpoints in a graphed analyses:

1. If the **Saved Tune States** panel is not currently visible, click its "unfold" button on the right.
2. Type a name into the Named Setting entry field (such as *Better*).

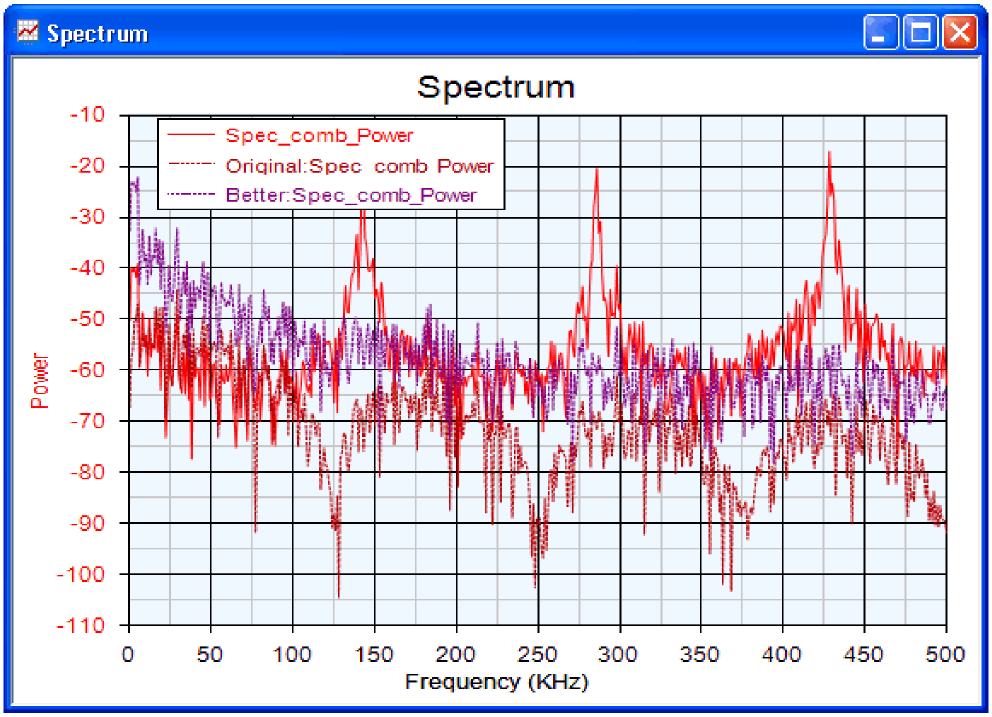


Better

3. Click the Checkpoint button.

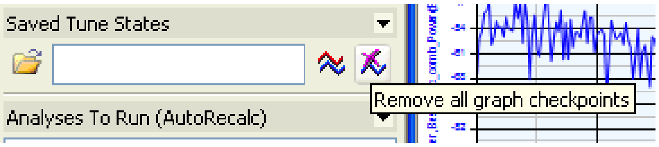
As you tune you will see an echo left behind of the original settings, this is the checkpoint.

You can add as many checkpoints as you like. Each new checkpoint will have a dashed trace and be in a darker color.



To remove all checkpoint traces from all graphs:

1. Click the Remove Checkpoint button in the tune window. This will remove checkpoints from the graphs listed in your graph checkpoint list. As you then tune a checkpoint will not be created.



Using X-Parameters in Genesys

This section shows how X-Parameter data can be incorporated into Genesys designs.

The X-parameter model is a generalized **circuit model** that includes **nonlinear** effects. The data for this model is contained in a Generalized MDIF file. A non-linear circuit simulation technique called **Harmonic Balance** is needed to make sense of X-parameter data.

X-Parameters are used in RF circuits to represent non-linear incident and reflected traveling waves.

Contents

Convergence Issues

The X-parameter model is a circuit level component. A non-linear circuit simulation technique called **Harmonic Balance** is needed to make sense of X-parameter data. Under high nonlinear conditions harmonic balance may be unable to converge to an accurate solution. In these cases, convergences parameters can be tweaked to optimize convergence for the given circuit problem.

By default when **XPARAMS** models are combined with system behavioral models in the same design each of the XPARAMS models will use the same generic default convergence criteria. This model provides no mechanism for the user to change the convergence criteria. When XPARAMS model(s) are placed in a Circuit_Link design the the entire design will all have common convergence criteria that can be controlled by the user.

Getting X-Parameters into the Workspace

X-parameters file data will automatically be imported and cached into memory when a simulation runs that contains X-parameter parts. All X-parameter data used in a workspace will remain in cached memory until the workspace is close or another workspace is opened.

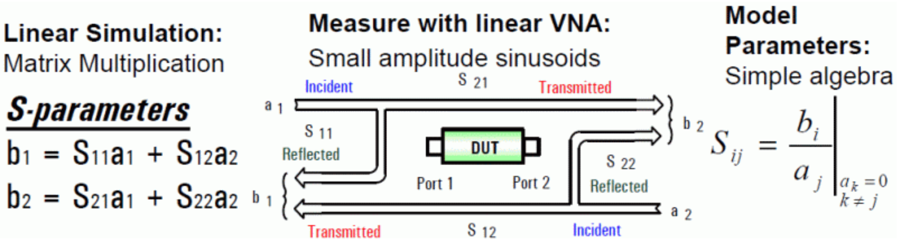
> **Note**
> Neither datasets nor any other type of workspace tree object is created during this automatic import process. X-parameter file data is cached to improve simulation performance.

For more information on the X-parameter file format see *X-parameter GMDIF Format* (users).

Theory of Operation

Traditional S-Parameters

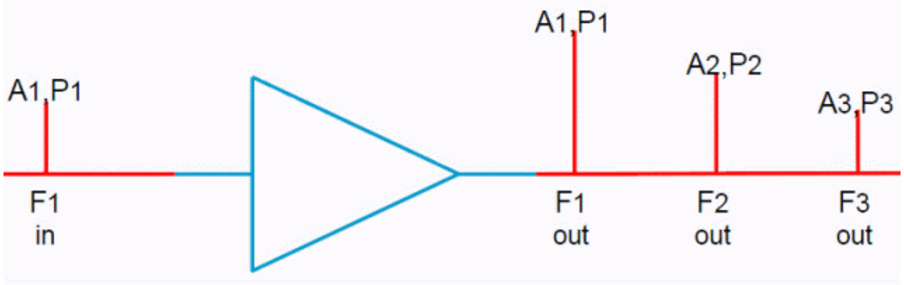
At high RF frequencies terminal voltages and currents are difficult to measure. Scattering parameters, or S-parameters are ratios of power flow amplitudes and phases in a circuit which are much easier to measure at these frequencies. However, S-parameters only characterize the linear behavior of RF devices.



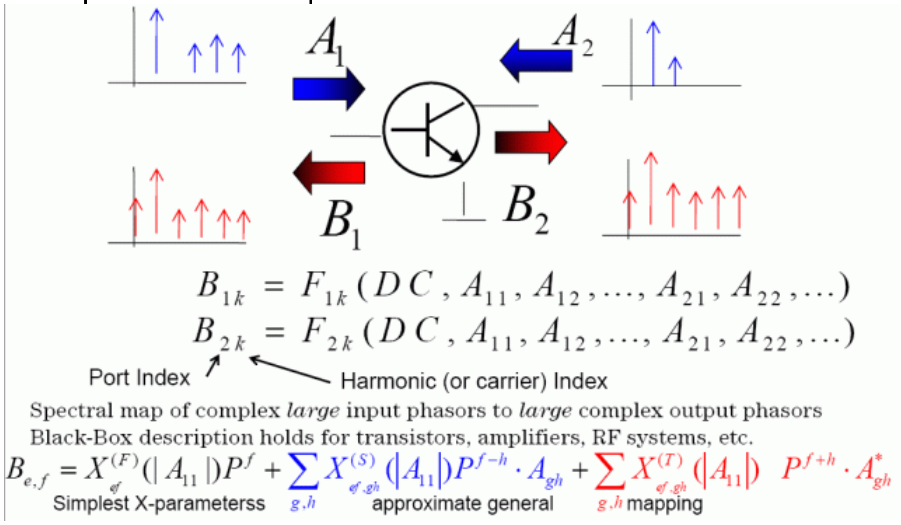
X-Parameter Basics

Unlike S-parameters, X-parameters characterize the linear and non-linear circuit behaviors of RF components in a more robust and complete manner. In effect, X-parameters are the mathematically correct super-set of S-parameters, applicable to both large-signal and small-signal conditions, for linear and nonlinear components. X-parameters are cascade-able just like S-parameters so higher levels of integration can be simulated or characterized.

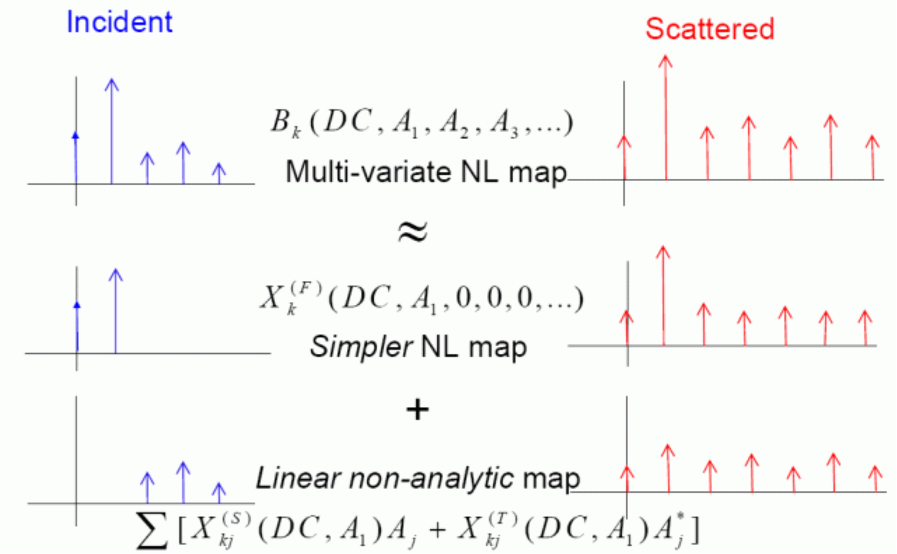
A simplified non-linear output spectrum from a single input spectrum is shown in the following figure.



The incident waves A1 and A2 and the resultant reflected B1 and B2 waves are shown for a simple nonlinear 2 port device.


A_1
A_2
B_1
B_2
$B_{1k} = F_{1k}(DC, A_{11}, A_{12}, \ldots, A_{21}, A_{22}, \ldots)$
$B_{2k} = F_{2k}(DC, A_{11}, A_{12}, \ldots, A_{21}, A_{22}, \ldots)$
Port Index
Harmonic (or carrier) Index
Spectral map of complex large input phasors to large complex output phasors
Black-Box description holds for transistors, amplifiers, RF systems, etc.
$B_{e,f} = X^{(F)}_{ef}(|A_{11}|)P^{f} + \sum_{g,h} X^{(S)}_{ef,gh}(|A_{11}|)P^{f-h}\cdot A_{gh} + \sum_{g,h} X^{(T)}_{ef,gh}(|A_{11}|)\ P^{f+h}\cdot A^{*}_{gh}$
Simplest X-parameterss
approximate general
mapping

The X-parameter approach is similar to various nonlinear mapping techniques as shown.


Incident
Scattered
$B_k(DC, A_1, A_2, A_3, \ldots)$
Multi-variate NL map
$\approx$
$X_k^{(F)}(DC, A_1, 0, 0, 0, \ldots)$
Simpler NL map
+
Linear non-analytic map
$\sum [X_{kj}^{(S)}(DC, A_1)A_j + X_{kj}^{(T)}(DC, A_1)A_j^*]$

File Extraction Basics

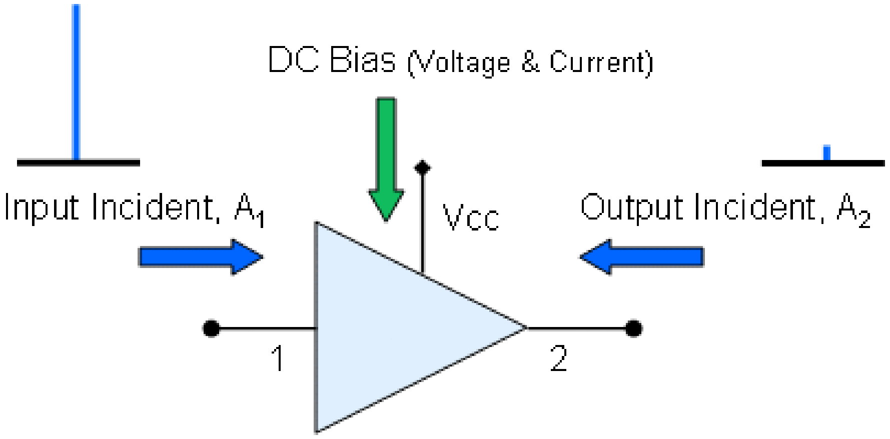
X-parameter data can either be extracted by special network analyzers such as Agilent's NVNA network analyzer or specialized simulation software such as Agilent's Advanced Design System (ADS). When an X-parameter file is extracted from a nonlinear device the user must supply the following extractions parameters and boundaries:

1. The number of characterization carriers (large signal).
2. The frequency of each carrier (*fund_k*).
3. The power level range of each carrier (*AN_p_n*).
4. The phase range of each carrier (*AP_p_n*).
5. The characteristic impedance.
6. DC voltage or current bias ranges (*VDC_p* & *IDC_p*).
7. Load characteristics that may be in the form of either reflection coefficients or impedance's (*GM_p_n, _GP_p_n*, etc).
8. User specified variables may also be used

Notation:

- **k** - fundamental frequency index
- **p** - port index
- **n** - harmonic index
- **m** - minus sign i.e. *_m2* = -2

Example of setup for 1 characterizing tone (Output Incident, A $_2$ is optional):



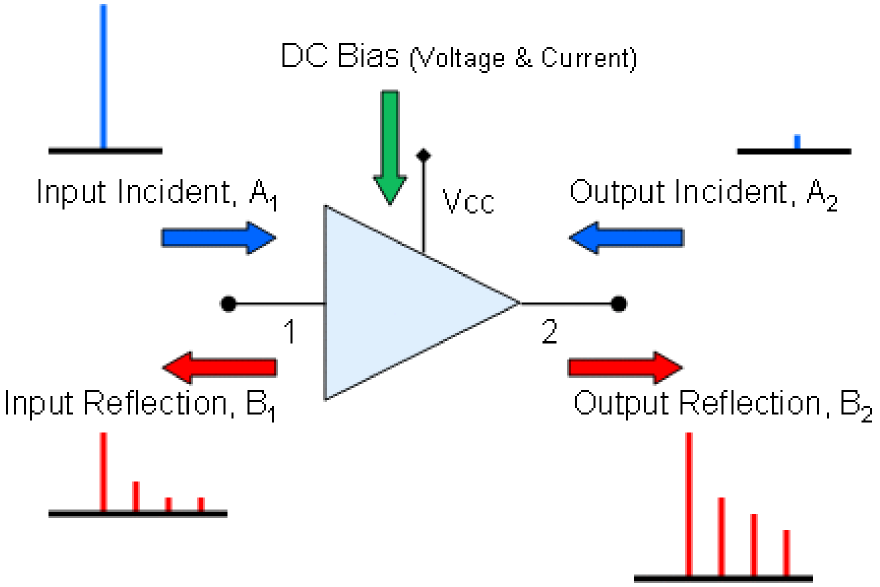
Extracted Data

Specialized hardware or simulation software extracts a text file containing the dependent data based on the independent input parameters listed in the prior section. The extracted output consists of several pieces of information for each input carrier. Every port is examined across a specified range of harmonics of the input carriers. Each piece of the contributing resultant output spectrum is characterized and saved in the extracted file.

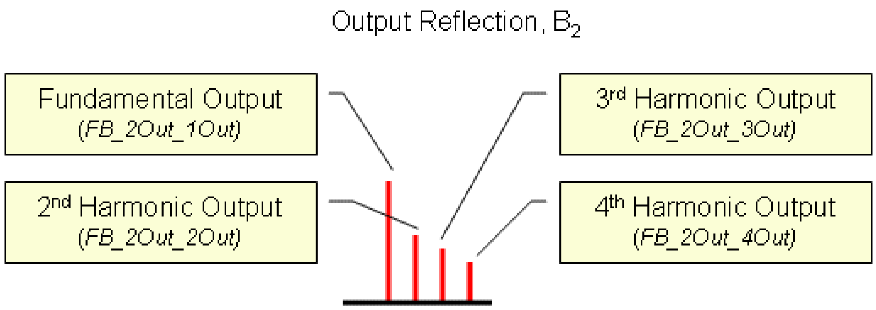
The extracted output consists of the following data:

1. Carrier reflected wave at the output (*FB_pOut_nOut*)
2. DC output current (*FI_pOut*)
3. DC output voltage (*FV_pOut*)
4. Small signal added output contribution due to a small signal input (S_pOut_nOut_pIn_nIn)
5. Small signal added output contribution due to phase-reversed small signal inputs (T_pOut_nOut_pIn_nIn)
6. DC current added output contribution due to small signal inputs (*XY_pOut_pIn_nIn*)
7. DC voltage added output contribution due to small signal inputs (*XZ_pOut_pIn_nIn*)

Example of extracted data from a single large signal characterizing tone:

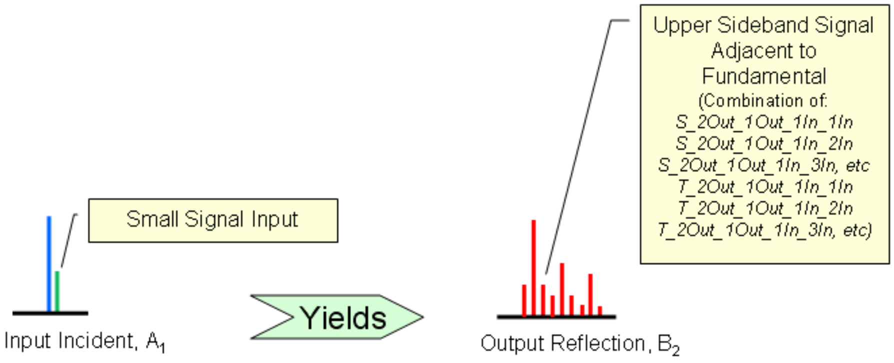


Examining the **Reflected B** $_2$ spectrum for the 1 characterizing input tone we get:



To account for large and small signal effects a 'Quasi-Linear' system is created by internally generating a small signal at frequencies slightly different than the characterizing carrier frequencies. These small signals combined with the large characterizing signals produce new frequencies. By linear superposition the output frequencies and amplitudes can be determined for all small signal inputs in a real system.

The following figure illustrates the resulting spectrum from a single large signal characterization tone and small signal at the input.



For more information see *X-parameter Variables* (users).

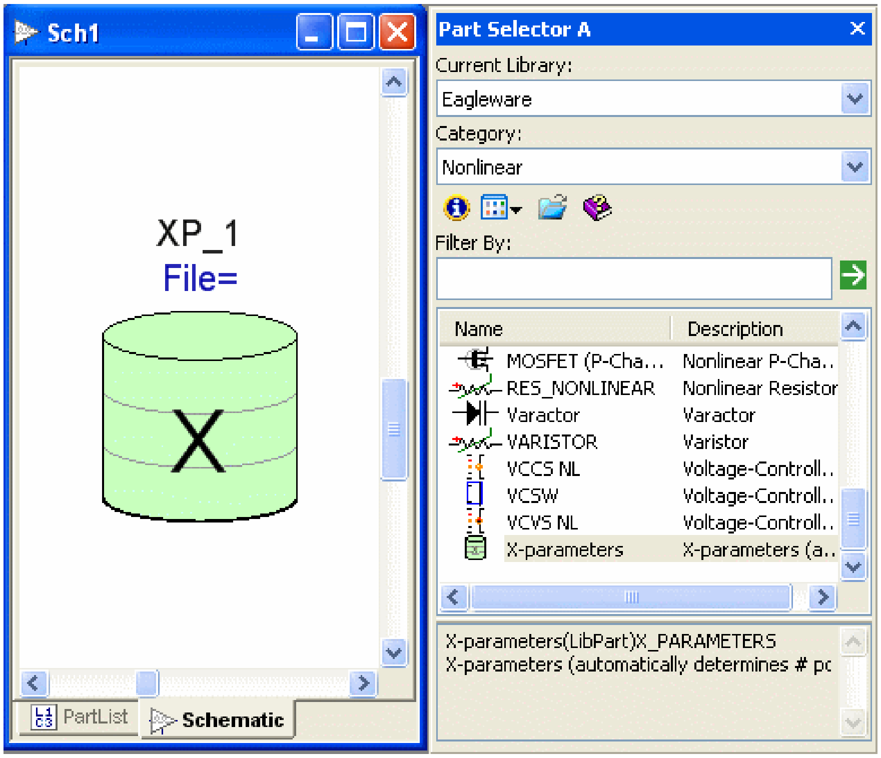
Using X-Parameters in a Design

To use an X-parameter file in a design follow these steps:

1. Place an X-Params Part
2. Browse to the X-Parameter File
3. Finish the Design
4. Add an Analysis

Place an X-Params Part

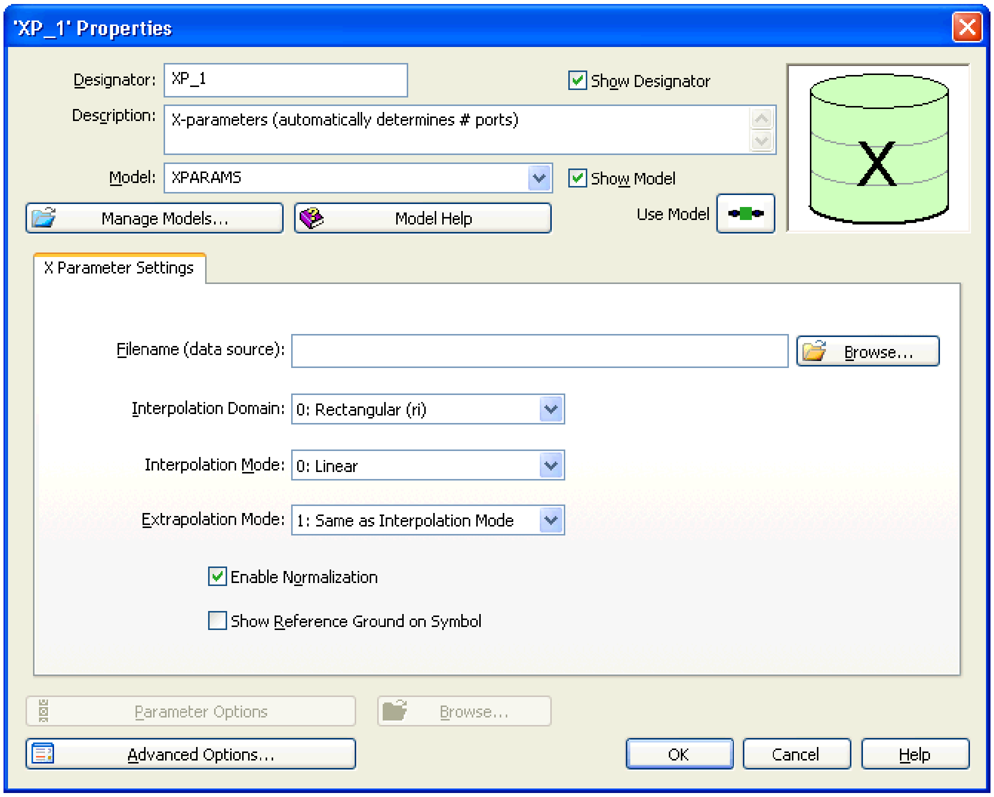
Select the **X-Params** part from the part selector located in the **Eagleware Nonlinear** Design library.



> **Note**
> When the X-Params model is placed the schematic symbol contains no pins. This is because the X-parameter file has not yet been selected (and of course, has not been read); the number of ports cannot be determined until the file is actually read.

Browse to the X-Parameter File

Double click the X-Params part to bring up the part properties.


'XP_1' Properties

Designator: XP_1

☑ Show Designator

Description: X-parameters (automatically determines # ports)

Model: XPARAMS

☑ Show Model

X Parameter Settings

Filename (data source):

Interpolation Domain: 0: Rectangular (ri)

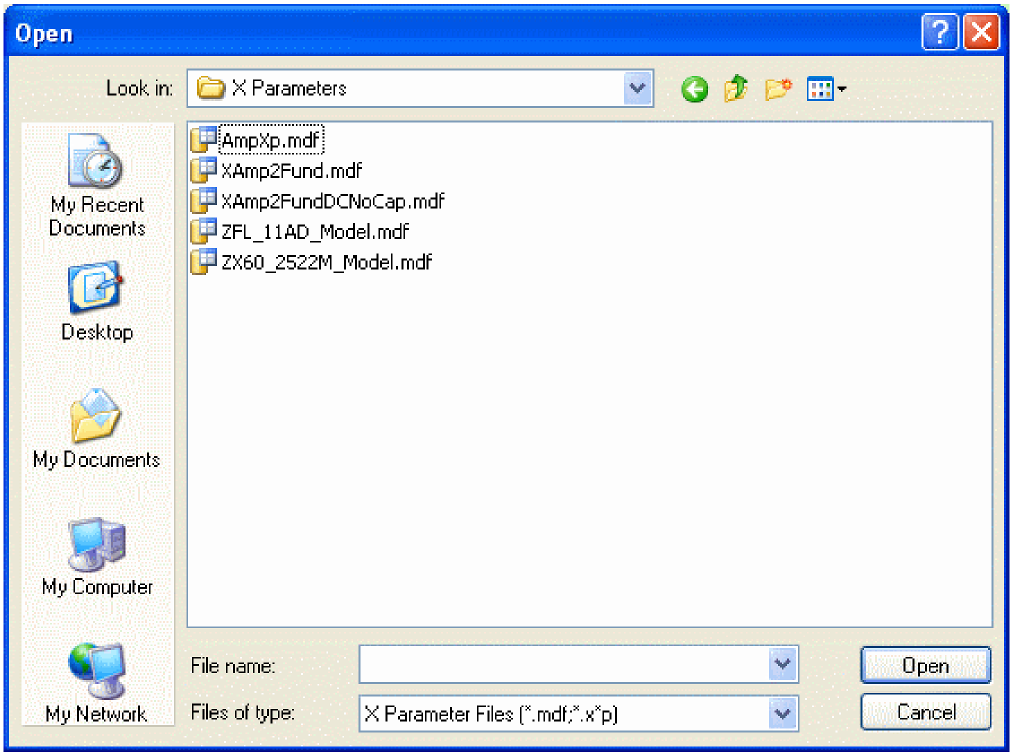
Interpolation Mode: 0: Linear

Extrapolation Mode: 1: Same as Interpolation Mode

☑ Enable Normalization

☐ Show Reference Ground on Symbol

Click the Browse (Browse...).



Look in: X Parameters

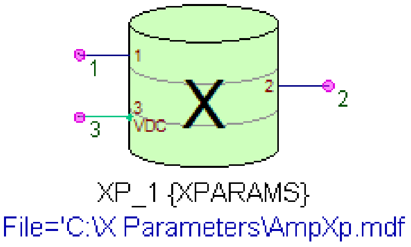
- AmpXp.mdf
- XAmp2Fund.mdf
- XAmp2FundDCNoCap.mdf
- ZFL_11AD_Model.mdf
- ZX60_2522M_Model.mdf

File name:

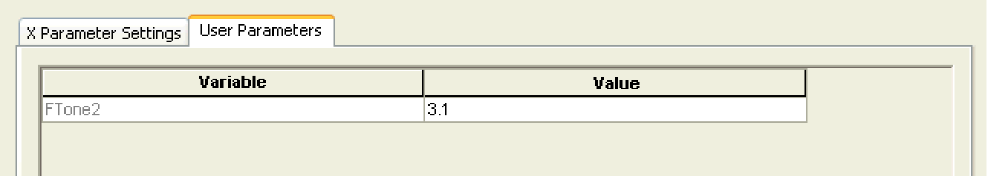
Files of type: X Parameter Files (*.mdf;*.x*p)

Select the desired X-parameter file.

At this time the number of ports is resolved and the schematic symbols changes appropriately because a specified X-parameter file has been selected.



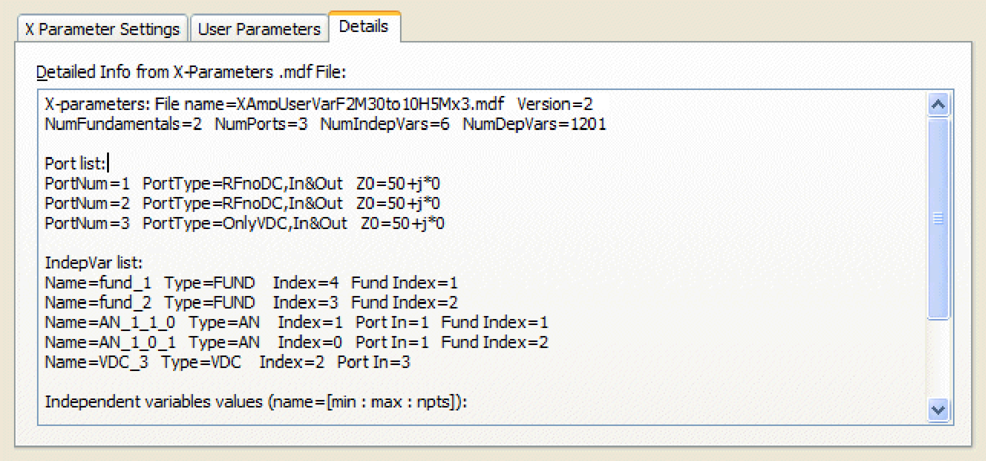
Also, one or two additional (optional) tab pages may appear: A User Parameters tab may appear, if the X-parameter file has any User Variables defined.



Variable	Value
FTone2	3.1

These parameters are defined by the file. You may NOT add, delete, or rename the User Variables, but you can change their values to any floating point number. (Equations are not permitted for values.)

A Details page displays a summary of the info from the X-Paramters .mdf file.



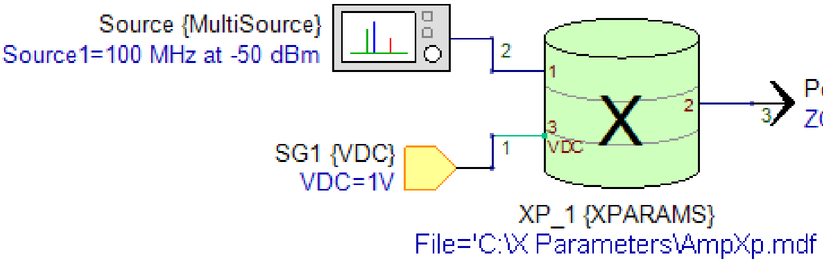
Detailed Info from X-Parameters .mdf File:

```
X-parameters: File name=XAmpUserVarF2M30to10H5Mx3.mdf  Version=2
NumFundamentals=2  NumPorts=3  NumIndepVars=6  NumDepVars=1201

Port list:
PortNum=1  PortType=RFnoDC,In&Out  Z0=50+j*0
PortNum=2  PortType=RFnoDC,In&Out  Z0=50+j*0
PortNum=3  PortType=OnlyVDC,In&Out  Z0=50+j*0

IndepVar list:
Name=fund_1  Type=FUND   Index=4  Fund Index=1
Name=fund_2  Type=FUND   Index=3  Fund Index=2
Name=AN_1_1_0  Type=AN   Index=1  Port In=1  Fund Index=1
Name=AN_1_0_1  Type=AN   Index=0  Port In=1  Fund Index=2
Name=VDC_3  Type=VDC   Index=2  Port In=3

Independent variables values (name=[min : max : npts]):
```

For more information on the X-Params model properties see X-Parameter Part .

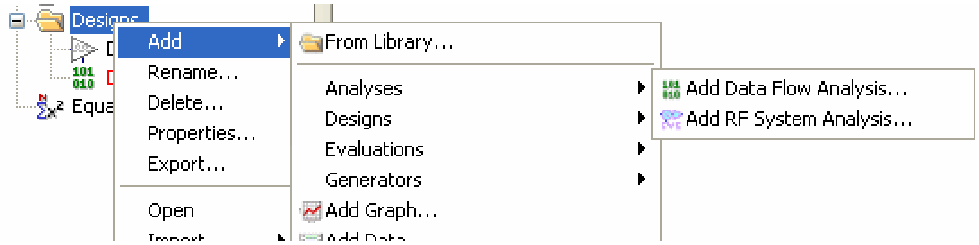
Finish the Design

Place the desired components to finish the design. (In this particular example a Multisource, Output Port, and Signal Ground parts are used)



Add an Analysis

Add the desired analysis.



Run the analysis and plots the results.

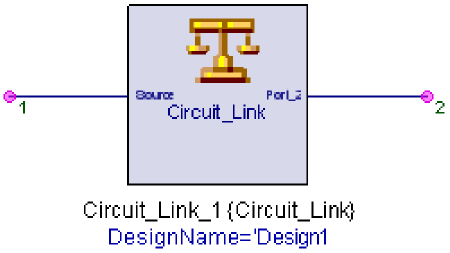
Using DC Bias Voltage

X-parameters can be characterized with various DC bias voltages and can even support multiple DC bias ports.

⚠ **Caution**
If the X-parameter file has been characterized with a single DC voltage the internal interpolation and extrapolation algorithms can only use this **single** bias point so all interpolated or extrapolated DC bias voltages will all be at the same DC bias voltage. Consequently, specifying a DC bias voltage on the part becomes irrelevant.

Using X-Parameters in the Circuit Link

The **Circuit Link** component is used as a bridge between circuit and system level components.



This bridge points to a design and contains parameters most often needed to control circuit level convergence criteria. A nonlinear circuit simulation technique called harmonic balance uses this criteria to simulate the linear and nonlinear characteristics of the circuit. These results are passed to the Spectrasys for spectral creation and path measurement calculations.

For more information see *Circuit_Link* (part)

⚠ **Caution**
The accuracy of cascaded circuit components will be increased when all circuit level components are combined in a single *Circuit_Link* (part) component.

Using X-Parameters in Spectrasys

The X-Params model predicts the circuit level output currents and voltages given specific characterization characteristics contained in the X-parameter file. A common nonlinear simulation technique called **Harmonic Balance** is used to extract the necessary information needed by the system simulator called Spectrasys. Spectrasys is a nonlinear behavioral simulator and the simulation approach is drastically different than that used for nonlinear circuits.

RF system simulation is used to determine optimum RF architecture as well as requirements for each of the behavioral blocks or sub-systems in a system that has a common characteristic impedance. Circuit simulations can be oblivious to characteristic impedance's and users are generally more interested in circuit input and output characteristics rather than cascaded parameters are some internal intermediate nodes.

⚠ **Note**
Highest circuit simulation accuracy will be achieved when all circuit level components such as X-parameters are placed together in a single **Circuit Link** component. Complex circuit level interactions between cascaded circuit components may be missed in during the system simulation.

Validation Limits

1. Spectrasys simulation using single X-parameters part with single tone or 2-tone stimulus has been compared with equivalent simulation in ADS, all results are consistent.
2. Spectrasys simulation using cascaded X-parameters parts with single tone or 2-tone stimulus has been compared with equivalent simulation in ADS, results are consistent with reasonable (negligible) difference (e.g. less than a few tenths of a dB at fundamental frequency and can be slightly higher for mixing terms < -50dBm) due to the difference in underlying computational algorithms (e.g convergence criteria).

Performance Limits

If simulation speed becomes an issue (most likely due to convergence), use Circuit_Link with *X-Parameters Part* (part) part to control the convergence criteria directly.

Operational Limits

> ⚠ **Caution**
> Currently, X-parameter models are not allowed in the LO chain for **RF LINK** simulations only.

Noise

> ℹ **Note**
> Currently, the X-parameter parts do not support self generated device noise. However, any external noise appearing at the X-parameter ports will be amplified by the small-signal gain specified in the X-parameter file.

Frequency and Power Limits

X-parameter files are extracted across a user specified power range with a fixed number of input tones at user specified frequencies. During a simulation frequency and power values will be interpolated if the simulation frequencies and power levels reside within the characterization limits otherwise the values will be extrapolated.

> ⚠ **Caution**
> If the characterizing tones are not swept in frequency or power there will be noting to interpolate or extrapolate since all frequency and power levels will appear to be constant.

Tone Characterization and Mapping

Along with frequency and power level characterization a non-linear circuit is characterized by a fixed number of input tones (carriers) specified by the user. Furthermore, these tones can be swept or fixed in frequency and power level. During a simulation three simulation scenarios exist with regard to the number of tones used in the simulation versus the number of tones the X-parameter file was characterized with. They are:

1. Number of Simulation Tones **=** Number of X-parameter Characterization Tones
2. Number of Simulation Tones **<** Number of X-parameter Characterization Tones
3. Number of Simulation Tones **>** Number of X-parameter Characterization Tones

> ℹ **Note**
> Highest accuracy will only be achieved when the X-parameters are extracted with the exact number of carriers, frequencies, and power levels of interest.

X-parameters are simulated using a **large-signal-small-signal analysis** technique. In this technique a certain number of tones are designated as large signal all other input signals are considered small signal. When the large and small signal analysis techniques are combined distortion (intermod) products can be determined at all distortion frequencies.

During an X-parameter simulation all input carriers are sorted by power level. The largest input signal maps to the 1st X-parameter tone and the 2nd largest input signal maps to the 2nd X-parameter tone, etc. until all the large signal tones have been mapped. For example, if an X-parameter file was characterized with two tones, the first one fixed in frequency and power, and the second swept in power and frequency then during the simulation the largest power input tone would map to the fixed X-parameter tone and the next input carrier would map to the swept characterizing tone.

If the number of simulation tones equals the number of X-parameter characterization tones then each input tone is considered a large signal tone. If the number of simulation tones is less than the number of X-parameter characterization tones then the extra X-parameter characterization tones are ignored. If the number of simulation tones is greater than the number of X-parameter characterization tones then all the unmapped tones become small signal input tones.

> ⚠ **Caution**
> If the X-parameter file was only characterized with one tone and two tones are being used in the simulation the resulting simulation will be a one tone large signal analysis with a single small signal not the traditional two tone analysis.

For more information on **large-signal-small-signal analysis** see Mass, Stephen A, *Nonlinear Microwave Circuits.* Norwood, MA: Artech House, 1988, Chapter 3.

Yield/Monte Carlo

Monte Carlo and Yield Analysis are probabilistic ways to simulate component variations caused by the production process. These methods can be used to determine which components need to be low tolerance (usually more expensive components) or to help create designs that are able to accommodate parameter variation.

One method of gaining confidence is to consider worst case scenarios. The circuit response is computed with each component stepped up or down in value by the appropriate tolerances. The response is observed while all components are stepped in the direction resulting in the worst possible outcome for the parameter being considered. This process is fast and insightful with a real-time simulator such as Genesys. However, the outcome is generally pessimistic. Redesign, to fit worse case scenarios into desired specifications, may result in greater cost than rejecting or repairing a few units which fail test.

Monte Carlo analysis evaluates circuit behavior for a sample run size with a random distribution of component values within specified limits. It is a statistical process. It does not tell us with certainty what will happen with a particular unit, but it gives us confidence that production results will fall within acceptable limits.

Monte Carlo and Yield Analysis are virtually identical functions. The only difference between them is that Yield has Target functions that it tracks. As the component stepping occurs Yield Analysis retains a pass-fail value for each round (step) based on the targets. Monte Carlo simply retains the stepped values at each round along with the desired measurements for graphical or tabular comparison.

Contents

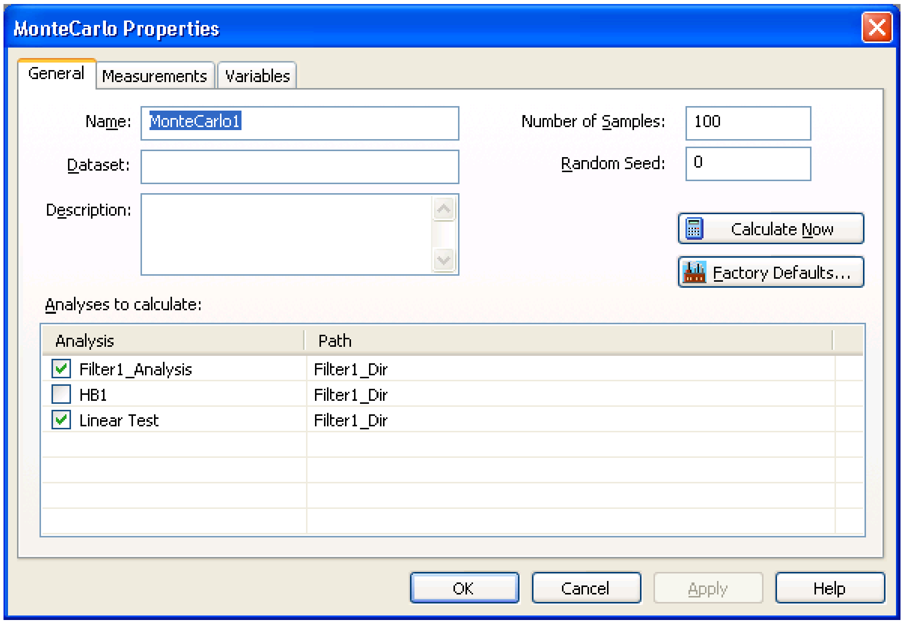
Creating a Monte Carlo Analysis

To create a Monte Carlo Analysis:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Monte Carlo Analysis** from the Evaluations submenu.

Select Analyses to Run

1. Determine which analyses will run when you do the statistical run by checking them on or off.



Name: MonteCarlo1

Dataset:

Description:

Number of Samples: 100

Random Seed: 0

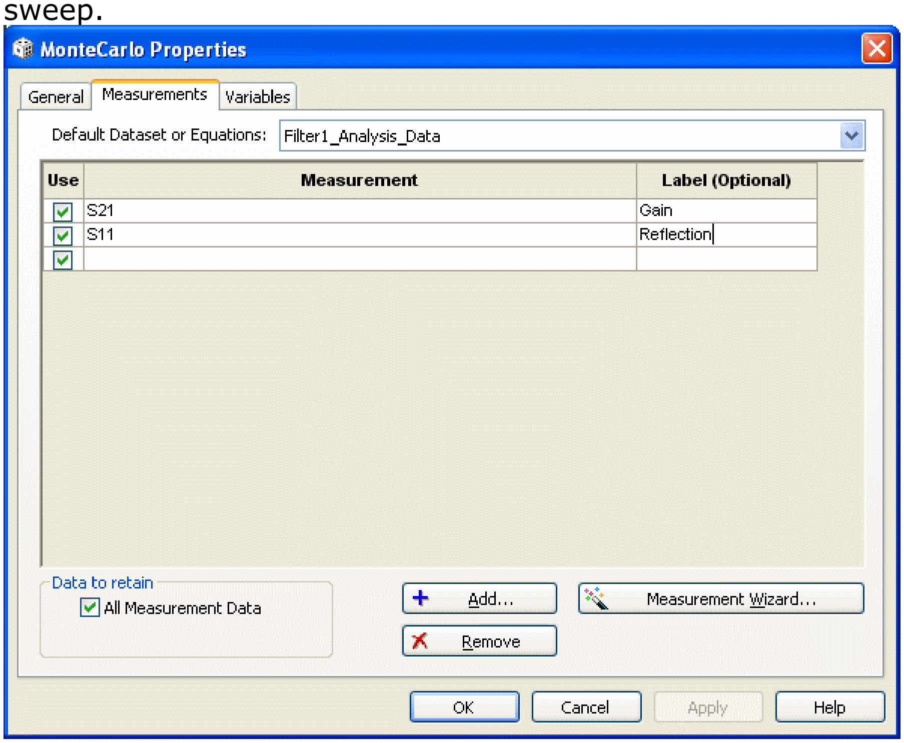
Analyses to calculate:

Analysis	Path
☑ Filter1_Analysis	Filter1_Dir
□ HB1	Filter1_Dir
☑ Linear Test	Filter1_Dir

In this page you can also set the Random Seed value (the Monte Carlo run is pseudo-random) and the number of samples. Large numbers of samples can generate a *lot* of data. If you leave the Dataset name blank Genesys will use *MonteName* _Data as the default name.

Select Measurements to Retain

While the Monte Carlo analysis runs, Genesys stores whatever measurements you have listed here into the output dataset. This data has the same format as a typical Parameter sweep.



Default Dataset or Equations: Filter1_Analysis_Data

Use	Measurement	Label (Optional)
☑	S21	Gain
☑	S11	Reflection
☑		

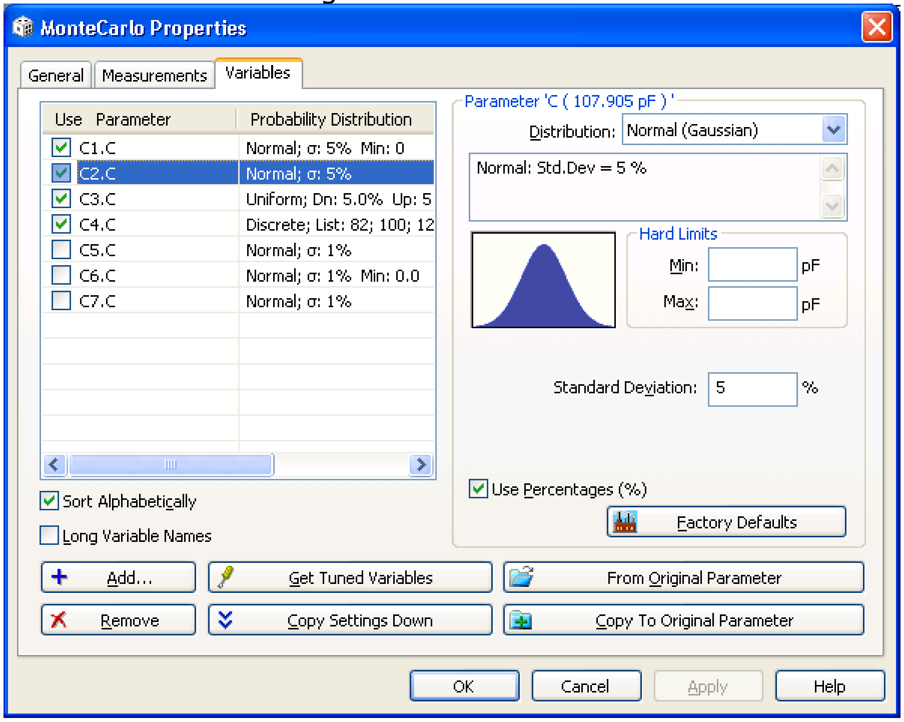
Data to retain

☑ All Measurement Data

For a Monte Carlo analysis you will almost always want to retain the Measurement Data (for Yield you may prefer to only save pass-fail status (for speed and storage savings). The measurement Label is how the data is identified within the dataset. You can individually turn Measurements on and off for better selectivity.

Select Variables to Perturb

A Monte Carlo analysis perturbs (statistically varies) variables. Select these on the 'Variables' tab. The parameter variables used in this dialog box are copies of the original values and not the original values themselves.



MonteCarlo Properties

Use	Parameter	Probability Distribution
☑	C1.C	Normal; σ: 5% Min: 0
☑	C2.C	Normal; σ: 5%
☑	C3.C	Uniform; Dn: 5.0% Up: 5
☑	C4.C	Discrete; List: 82; 100; 12
☐	C5.C	Normal; σ: 1%
☐	C6.C	Normal; σ: 1% Min: 0.0
☐	C7.C	Normal; σ: 1%

☑ Sort Alphabetically

☐ Long Variable Names

Parameter 'C (107.905 pF)'

Distribution: Normal (Gaussian)

Normal: Std.Dev = 5 %

Hard Limits

Min: pF

Max: pF

Standard Deviation: 5 %

☑ Use Percentages (%)

Once you select variables to perturb you can select how they are perturbed. The distribution selection is exactly the same as for Part Distributions.

Buttons

- **Sort Alphabetically** - When checked the parameters names are sorted alphabetically.
- **Long Variable Names** - When checked the full path from the design to the parameter is shown in the parameter list.
- **Add** - Click this button to bring the 'Select Variables to Tune' dialog box. Select desired variables for analysis and click OK. These selected variables will be added to the parameter list.
- **Remove** - Click this button after selecting a parameter from the parameter list to remove it from the parameter list.
- **Get Tuned Variables** - Click this button to select which tuned variables to get.
- **Values with Distributions** - This will only get the tuned variables that have been assigned a distribution. For example, this option will ignore tuned variables created in the equation block that have no assigned distribution.
- **All Values** - The will get any variable that is tunable. Including those created in an equation block.
- **Copy Settings Down** - Click this button to have the settings on the currently selected parameter copied down to all the parameters below it. Convenient for repeating entries.
- **From Original Parameter** - Click this button to copy the parameter statistics of the selected parameter from the part.
- **Copy to Original Parameter** - Click this button to copy the current parameter settings to parameter statistics of the original part.

Distributions

The following distributions are supported. The *distribution settings* (users) are the same as for Part Distributions.

Distribution	**Parameters**
Uniform (users)	Up / Down
Normal (users)	Standard Deviation
Lognormal (users)	Standard Deviation
Beta (users)	Alpha / Beta
Discrete (users)	List of Values

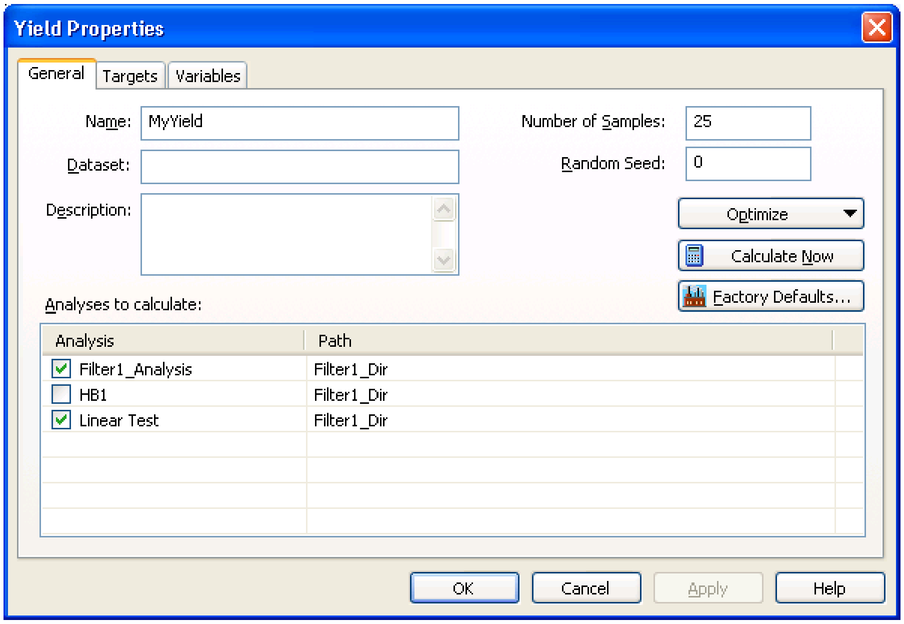
Creating a Yield Analysis

To create a Yield Analysis:

1. Click the New Item button ( ) on the Workspace Tree toolbar and select **Add Yield Analysis** from the Evaluations submenu.

Select Analyses to Run

1. Determine which analyses will run when you do the statistical run by checking them on or off.



Yield Properties

Name: MyYield

Dataset:

Description:

Number of Samples: 25

Random Seed: 0

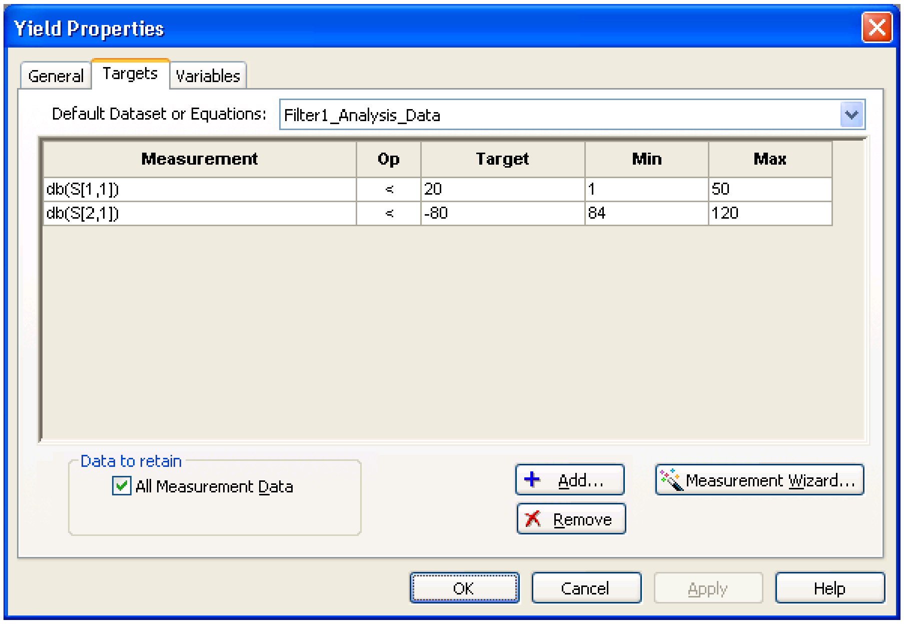
Analyses to calculate:

Analysis	Path
☑ Filter1_Analysis	Filter1_Dir
□ HB1	Filter1_Dir
☑ Linear Test	Filter1_Dir

In this page you can also set the Random Seed value (the Monte Carlo run is pseudo-random) and the number of samples. Large numbers of samples can generate a *lot* of data. If you leave the Dataset name blank Genesys will use *YieldName* _Data as the default name.

Select Targets

While theYield analysis runs, Genesys stores whatever measurements you have listed here into the output dataset. This data has the same format as a typical Parameter sweep. In addition the Error from the target comparisons is stored in the dataset for each round.



Default Dataset or Equations: Filter1_Analysis_Data

Measurement	Op	Target	Min	Max
db(S[1,1])	<	20	1	50
db(S[2,1])	<	-80	84	120

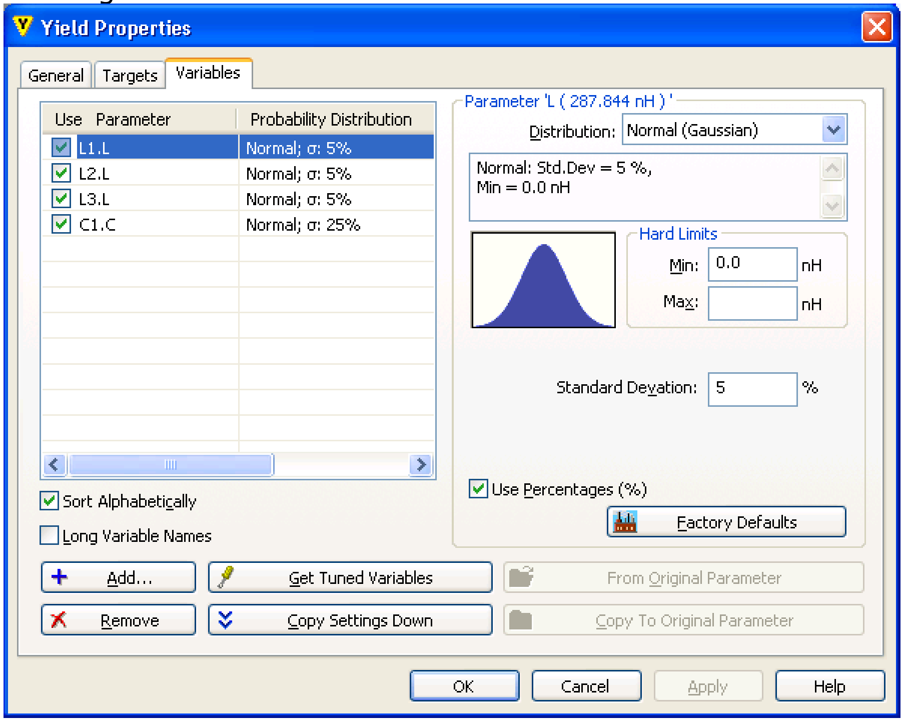
Data to retain

☑ All Measurement Data

For a Monte Carlo analysis you will almost always want to retain the Measurement Data, for Yield you may prefer to only save pass-fail status (for speed and storage savings). The Min and Max values are in the unit of measure of the independent (here, Frequency) variable. If you leave them blank, it calculates error over the entire range.

Select Variables to Perturb

A Yield analysis perturbs (statistically varies) variables. Select these on the 'Variables' tab. The parameter variables used in this dialog box are copies of the original values and not the original values themselves.



Use	Parameter	Probability Distribution
☑	L1.L	Normal; σ: 5%
☑	L2.L	Normal; σ: 5%
☑	L3.L	Normal; σ: 5%
☑	C1.C	Normal; σ: 25%

☑ Sort Alphabetically

□ Long Variable Names

Parameter 'L (287.844 nH)'

Distribution: Normal (Gaussian)

Normal: Std.Dev = 5 %,
Min = 0.0 nH

Hard Limits

Min: 0.0 nH

Max: nH

Standard Devation: 5 %

☑ Use Percentages (%)

Once you select variables to perturb you can select how they are perturbed. The distribution selection is exactly the same as for Part Distributions.

Buttons

- **Sort Alphabetically** - When checked the parameters names are sorted alphabetically.
- **Long Variable Names** - When checked the full path from the design to the parameter is shown in the parameter list.
- **Add** - Click this button to bring the 'Select Variables to Tune' dialog box. Select desired variables for analysis and click OK. These selected variables will be added to the parameter list.
- **Remove** - Click this button after selecting a parameter from the parameter list to remove it from the parameter list.
- **Get Tuned Variables** - Click this button to have all tuned variables placed into the parameter list.
- **Copy Settings Down** - Click this button to have the settings on the currently selected parameter copied down to all the parameters below it. Convenient for repeating entries.
- **From Original Parameter** - Click this button to copy the parameter statistics of the selected parameter from the part.
- **Copy to Original Parameter** - Click this button to copy the current parameter settings to parameter statistics of the original part.

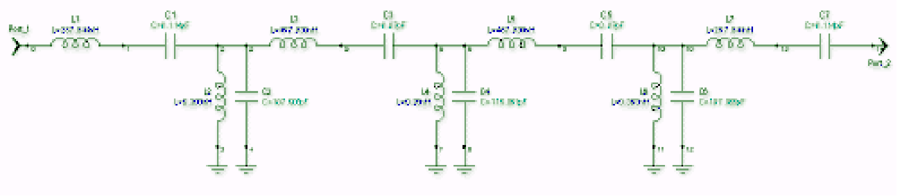
Distributions

The following distributions are supported. The *distribution settings* (users) are the same as for Part Distributions.

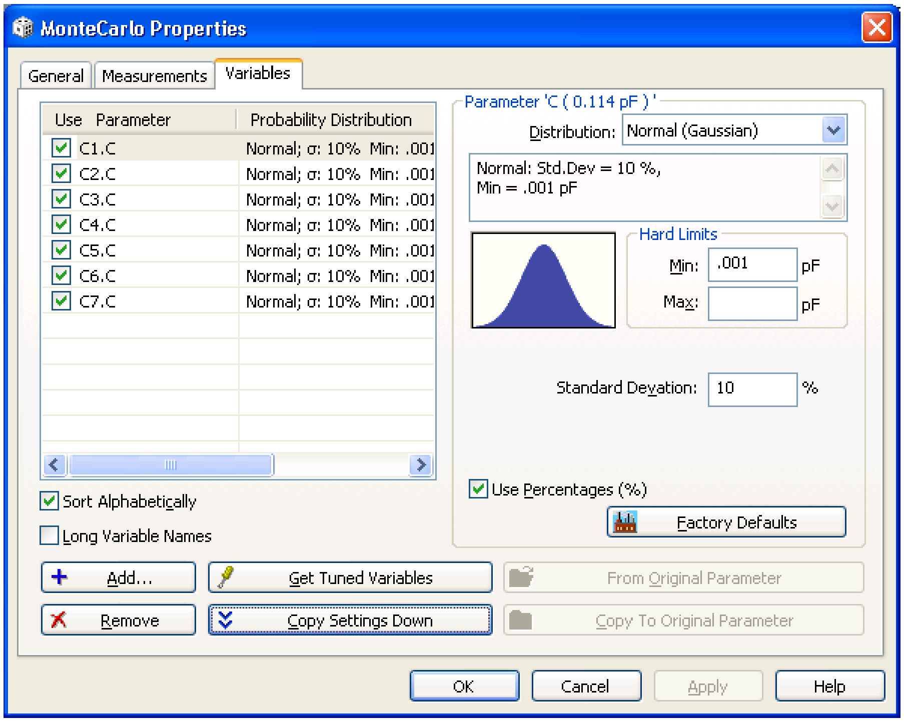
Distribution	Parameters
Uniform (users)	Up / Down
Normal (users)	Standard Deviation
Lognormal (users)	Standard Deviation
Beta (users)	Alpha / Beta
Discrete (users)	List of Values

Monte Carlo Example

The figure below shows a simple 7th order bandpass filter designed in FILTER. After creating the schematic, the capacitors were all selected (using Shift-Click with the mouse) and then set to tunable using the Schematic / Make Components Tunable command.



Next a Monte Carlo Analysis was created that perturbed each of the capacitors by a normal distribution with Sigma of 10% (as if they were 10% parts). The Variables page looks like this:



Use	Parameter	Probability Distribution
☑	C1.C	Normal; σ: 10% Min: .001
☑	C2.C	Normal; σ: 10% Min: .001
☑	C3.C	Normal; σ: 10% Min: .001
☑	C4.C	Normal; σ: 10% Min: .001
☑	C5.C	Normal; σ: 10% Min: .001
☑	C6.C	Normal; σ: 10% Min: .001
☑	C7.C	Normal; σ: 10% Min: .001

☑ Sort Alphabetically

☐ Long Variable Names

Parameter 'C (0.114 pF) '

Distribution: Normal (Gaussian)

Normal: Std.Dev = 10 %,
Min = .001 pF

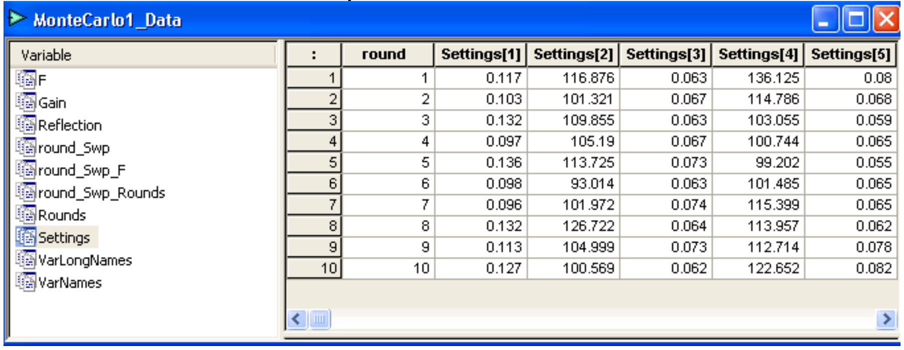
Min: .001 pF

Max: pF

Standard Devation: 10 %

☑ Use Percentages (%)

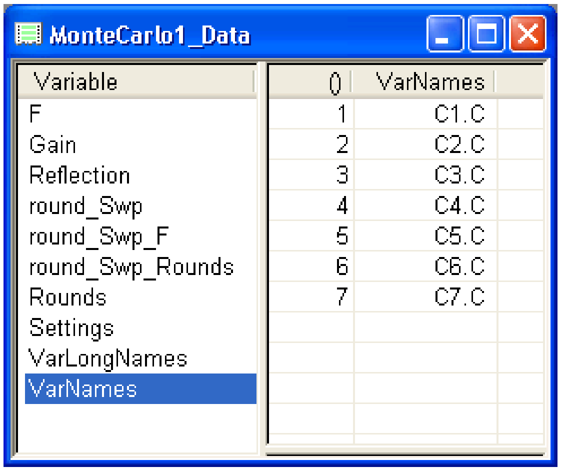
The Hard Limit Min value of .001 ensures no capacitor gets a negative value during the runs. Once the run completes we examine the dataset.



Variable
F
Gain
Reflection
round_Swp
round_Swp_F
round_Swp_Rounds
Rounds
Settings
VarLongNames
VarNames

:	round	Settings[1]	Settings[2]	Settings[3]	Settings[4]	Settings[5]
1	1	0.117	116.876	0.063	136.125	0.08
2	2	0.103	101.321	0.067	114.786	0.068
3	3	0.132	109.855	0.063	103.055	0.059
4	4	0.097	105.19	0.067	100.744	0.065
5	5	0.136	113.725	0.073	99.202	0.055
6	6	0.098	93.014	0.063	101.485	0.065
7	7	0.096	101.972	0.074	115.399	0.065
8	8	0.132	126.722	0.064	113.957	0.062
9	9	0.113	104.999	0.073	112.714	0.078
10	10	0.127	100.569	0.062	122.652	0.082

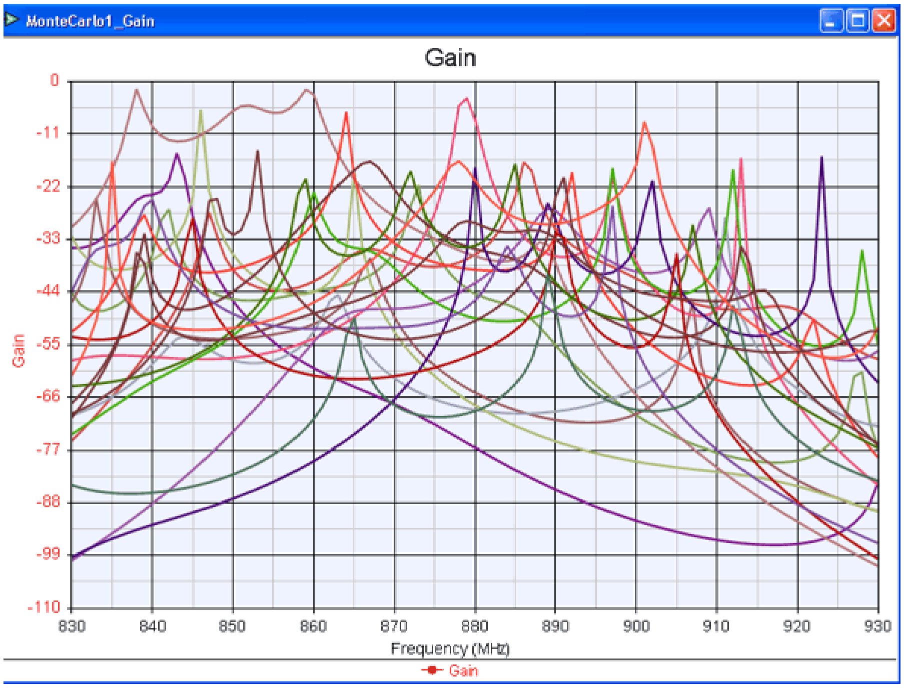
The Gain and Reflection variables contain our measured db(S21) and db(S11). Settings shows the values of each of the components for each round. VarNames is the names of the variables in the same order as Settings.



Variable
F
Gain
Reflection
round_Swp
round_Swp_F
round_Swp_Rounds
Rounds
Settings
VarLongNames
VarNames

()	VarNames
1	C1.C
2	C2.C
3	C3.C
4	C4.C
5	C5.C
6	C6.C
7	C7.C

We plot the Gain curves by right-clicking the Gain variable and selecting Graph / Rectangular Graph to see this:



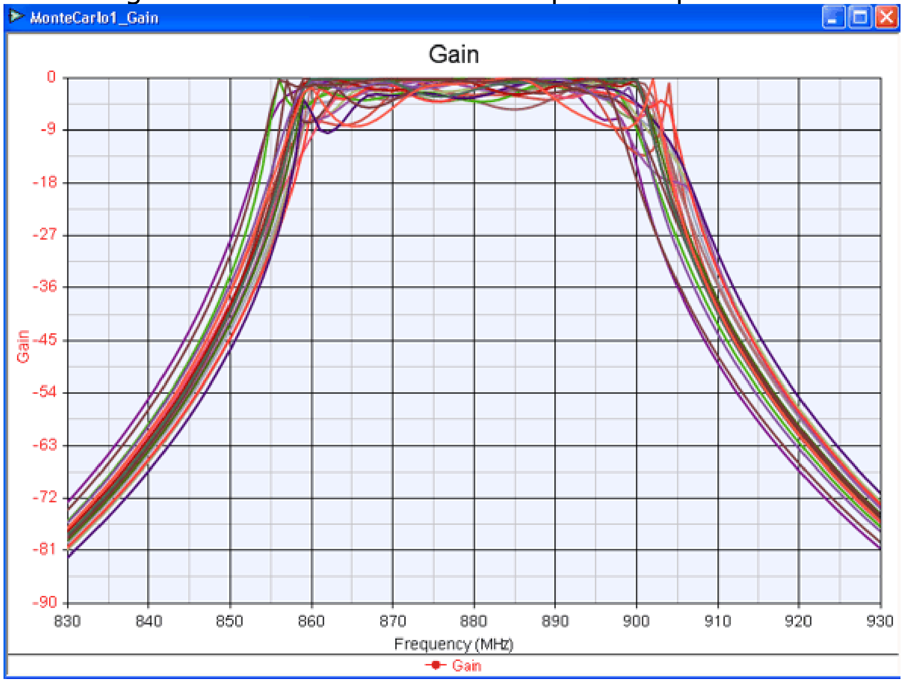
This looks like a mess - indicating that 10% capacitors may play havoc with the design.

Switch the Monte Carlo simulation to 1% components by:

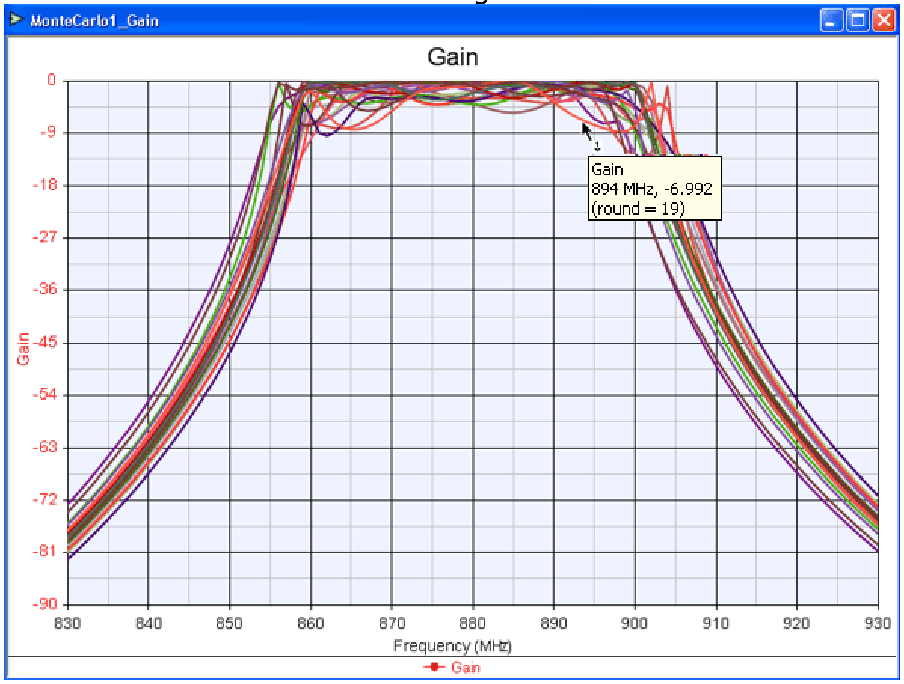
1. Double-click the Monte Carlo Analysis in the Workspace Tree (or bring up properties some other way).
2. Switch to the Variables tab
3. Type a 1 in the Sigma % column in the top variable.
4. Click Copy Settings Down to copy the 1 to all the variables.
5. Click OK

> We could have more easily just set the Sigma value of each part to an equation variable and then changed the variable from 10 to 1.

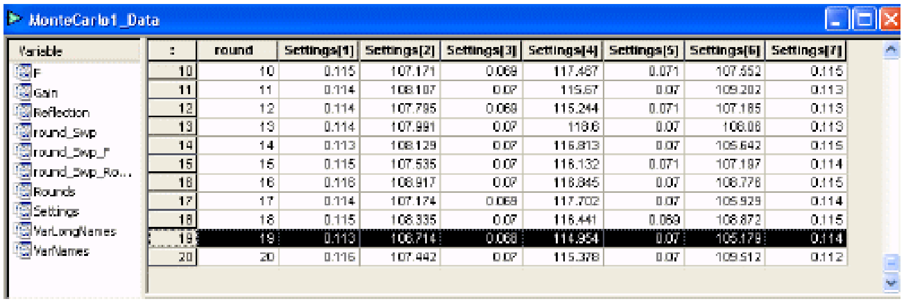
Switching the Monte Carlo to 1% components produces the following graph:



Now as we mouse-over the orange trace we see



So the Orange outlier was calculated at Round 19. Examining the Monte Carlo result dataset we see that round 19 was done with...



:	round	Settings[1]	Settings[2]	Settings[3]	Settings[4]	Settings[5]	Settings[6]	Settings[7]
10	10	0.115	107.171	0.069	117.467	0.071	107.552	0.115
11	11	0.114	108.107	0.07	115.67	0.07	109.202	0.113
12	12	0.114	107.795	0.069	115.244	0.071	107.185	0.113
13	13	0.114	107.991	0.07	116.6	0.07	106.08	0.113
14	14	0.113	108.129	0.07	116.813	0.07	105.642	0.115
15	15	0.115	107.535	0.07	116.132	0.071	107.197	0.114
16	16	0.116	106.917	0.07	116.845	0.07	106.778	0.115
17	17	0.114	107.174	0.069	117.702	0.07	105.929	0.114
18	18	0.115	106.335	0.07	116.441	0.069	106.872	0.115
19	19	0.113	106.714	0.068	114.954	0.07	105.179	0.114
20	20	0.116	107.442	0.07	115.378	0.07	109.512	0.112

Note that here both C4 and C6 are slightly "out of range".

Process Capability

This section is a brief overview of process capability and how process capability measurements can be incorporated into Genesys designs.

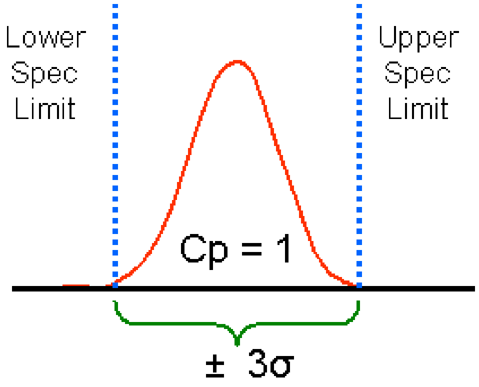
Basics

The outputs of any process will vary and it is not uncommon that the performance of a circuit of system falls outside their specification limits. When a device falls outside its requirements in a manufacturing environment additional costs are accrued to either scrap or rework the product both of which increase manufacturing costs and reduce the ROI (Return On Investment). Much can be done at the design level to maximize the production success rate.

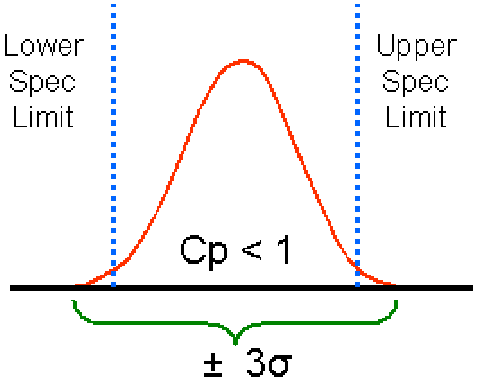
Statistical results are commonly used to determine product variation. The difference between the upper and lower specification is know as the tolerance. Capability indices have been developed to portray the relationship between specification limits and the design variation. These capability indices determine whether a process is capable of meeting some pre determined specifications.

Cp

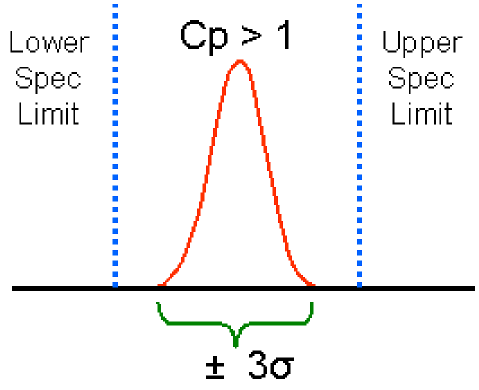
The **Cp** index relates the spread of the design relative to the specification width. The definition is:
***Cp* (users)** = (USL - LSL) / (6 x Std Dev)
The results are illustrated in the following figures.



For a design perfectly centered with a **Cp** = 1 there would be a 99.7% success rate.



The success rate of the design would decrease as **Cp** becomes less than 1.



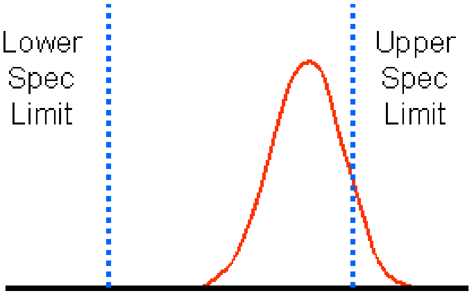
The success rate will increase for the design as **Cp** is greater than 1.

> That the **Cp** index assumes a design is perfectly centered which is often not the case. However, **Cp** is often referred to as the process 'potential'.

EXAMPLE: If Std Dev = 1.5, lower spec limit = 48, upper spec limit = 60 then the tolerance would be 60 - 48 = 12. The six sigma process spread would be 6 x 1.5 = 9. Therefore, Cp would equal 12 / 9 = 1.33.

Cpk

***Cpk* (users)** not only measures the design variation with respect to the specifications, it also accounts for the mean value. It considers only the variation half that is closest to the specification limit.



$$Cpk = \min\left(\frac{\overline{X} - LSL}{3\sigma}, \frac{USL - \overline{X}}{3\sigma}\right)$$

When the design is centered with the limits **Cp** and **Cpk** are equal.

EXAMPLE: If Std Dev = 1.5, lower spec limit = 48, mean value = 57, upper spec limit = 60 then distribution is closer to the upper spec limit than the lower. Cpk lower = (57 - 48) / (3 x 1.5) = 2 and Cpk upper = (60 - 57) / (3 x 1.5) = 0.67. Cpk = minimum of Cpk lower or Cpk upper = 0.67.

Using Cp and Cpk in Genesys

A 'Process Capability' equation library is provided with Genesys. To get access to these equations the '**MoreEquations.xml**' library must be loaded into the Library Manager. To accomplish this do the following:

Load the Library containing Cp and Cpk ('MoreEquations.xml')

1. Bring the library manager ('Tools' then 'Library Manager...' from the menu or Click the Library Manager (📂) button in the Library Selector)
2. Click the 'Add From File...'
3. Browse to the 'MoreEquations.xml' library located in the '\Models' installation directory (i.e. C:\Program Files\GENESYS20xx.yy\Models)

Using the **Cp** and **Cpk** functions

1. Open up the Library Selector (Ctrl+Shift+L or 'View', 'Docking Windows', then 'Library Selector')
2. Set the 'Library Type' to 'Equation'
3. Set the 'Current Library' to 'More Equations'
4. Double click the '**Process Capability**' library to add it to the workspace tree
5. Use the **Cp** and **Cpk** functions in local equation block in the workspace

Targets

Yield Targets are identical to Optimization Targets, except the "=" and "%" operators are almost never used because the yield would be zero (the measurements will almost never be *exactly* equal to the targets). The ">" and "<" operators are used to specify the range of output parameters which constitute a successful unit. The Optimization Targets are used to find component values which result in the desired nominal responses. The Yield Targets are used to set acceptable limits for definition of what is a successful unit during Monte Carlo analysis. See Entering Targets in the optimization section for the exact steps for entering targets. An example amplifier Yield Target Set might be:

Measurement	**Op**	**Target**	**Weight**	**Min**	**Max**
db(S21)	>	11.5	1	50	950
db(S21)	<	12.5	1	50	950
db(S11)	<	-10	1	50	950
db(S22)	<	-10	1	50	950

This specifies that all trials with a gain between 11.5 and 12.5 dB, and with better than 10 dB return losses, are considered to successful. The Target unit of measure is the measurement unit of measure. The Min/Max frequencies are in the unit of measure of the measurement independent (usually frequency).

When the workspace includes yield targets, the number and percentage of passes which meet the targets are displayed during the run. If yield targets do not exist in the evaluation, the display occurs, but the error is computed as zero and the yield is 100%.

The rules and options for use of yield targets are nearly identical to optimization targets. The philosophy of the optimization block is different in Yield analysis versus circuit optimization. If the above goals were used to optimize an amplifier, the response would likely be close to either 11.5 or 12.5 dB, at least at some frequencies. Using this same block in Yield would then result in a poor yield. Instead, using these targets for circuit optimization:

Measurement	**Op**	**Target**	**Weight**	**Min**	**Max**
db(S21)	=	12	1	50	950
db(S11)	<	-16	1	50	950
db(S22)	<	-16	1	50	950

would tend to center the gain response at 12 dB. Then, using yield targets for Monte Carlo with limits of 11.5 to 12.5 dB would produce a better yield.

The error values reported during Optimization and Yield Analysis are all computed by the same algorithm.

Yield Optimization

Yield Optimization is typically far more effective and its use is now recommended over Design Centering.

Yield Optimization finds part values based on the following equality:

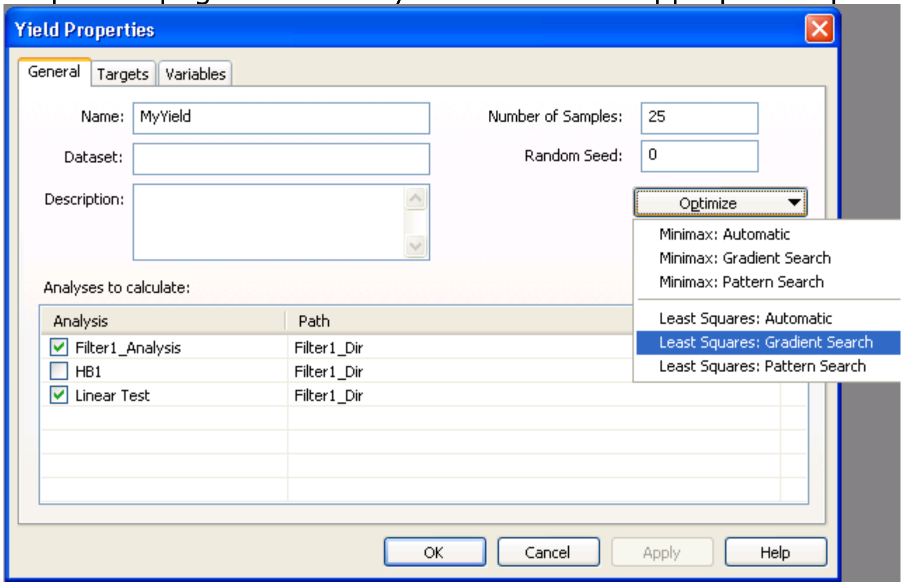
$$\forall_{j}, V_{j} = \frac{1}{n}\sum_{m=1}^{n} V_{j,m}$$

where

- V *j = the new value of the j* th tunable variable.
- V *j,m* = the *j* th tunable variable value at successful sample *m*
- n = number of samples where the yield is successful.

The principle of Yield Optimization is simple. It simply averages the part values for all the samples which satisfy the yield conditions. This simple principle is the most effective technique we have found for improving yield.

To perform a yield optimization, Create a Yield Analysis then create an Optimization that runs the Yield Analysis and whose goals are based on the Yield results. Genesys has a wizard to create the Yield Analysis for you. Just select from the Optimize menu in the Yield Properties page and Genesys will create an appropriate optimization.



Name: MyYield

Dataset:

Description:

Number of Samples: 25

Random Seed: 0

Analyses to calculate:

	Analysis	Path
☑	Filter1_Analysis	Filter1_Dir
□	HB1	Filter1_Dir
☑	Linear Test	Filter1_Dir

After this is selected, allow Genesys to create an optimization. By default the optimization will perturb the same variables as the Yield Analysis and it will minimize the sum of the Error results.

Appendix A - Keystroke Commands

- *General Keystroke Commands* (users)
- *Graph Keystroke Commands* (users)
- *Layout Keystroke Commands* (users)
- *LiveReport Keystroke Commands* (users)
- *Schematic Keystroke Commands* (users)

ℹ The availability of keystroke commands depends on the active design window.

General Keystroke Commands

- **Space** – Place another copy of the most recently placed item (schematics and layouts)
- **Escape** – Cancel current mode
- **Delete** – Delete current selection
- **Ctrl+A** – Select all
- **Ctrl+C** – Copy
- **Ctrl+D** – Duplicate
- **Ctrl+N** – File new
- **Ctrl+Shift+N** – Select none
- **Ctrl+O** – File open
- **Ctrl+P** – Print
- **Ctrl+S** – Save
- **Ctrl+V** – Paste
- **X** – Zoom – use the zoom tool (zoom to mouse rectangle)
- **Ctrl+X** – Cut
- **Ctrl+Y** – Redo
- **Ctrl+Z** – Undo
- **Z** – Zoom to fit all objects (Maximize)
- **Shift+Z** – Zoom to fit with extra margin
- **Ctrl+Shift+Z** – Redo
- **+** – Zoom in
- **–** – Zoom out
- **Ctrl+End** – Show entire page (maximize)
- **Ctrl+Home** – Zoom to fit
- **Ctrl+PageUp** – Zoom in
- **Ctrl+PageDown** – Zoom out
- **LeftArrow**, **RightArrow**, **UpArrow**, **DownArrow** – Move the current selection in the direction indicated (use the Enter key to drop parts in schematic after moving with the arrow keys)
- **Ctrl+LeftArrow**, **Ctrl+RightArrow**, **Ctrl+UpArrow**, **Ctrl+DownArrow** – Pan (scroll) the view (when nothing is selected)
- **F3** – Rotate item clockwise
- **Shift+F3** – Rotate item counterclockwise
- **Ctrl+F3** – Reset rotation angle to 0
- **F5** – Does an Action / Run All Out-of-Date Analyses and Sweeps (calculates simulations/sweeps)
- **Shift+F5** – Run all optimizations
- **F6** – Mirror an item
- **Ctrl+F6** – Reset mirror state to unmirrored
- **Alt+F7** – Print/export entire screen
- **F7** – Hide/Show docker windows (tree and tune windows)
- **F8** – Fit Windows to Frame - resize the windows to fit the non-docker area

- **Ctrl+F8** – Next editor
- **Alt+F8** – Print/export active window

Graph Keystroke Commands

- **C** – Checkpoint – Create a graph checkpoint or remove existing checkpoints
- **F** – Favorite – save a graph axis favorite
- **B** – Back – use a graph axis favorite
- **V** – Vertex – Hide / Show vertex symbols
- **R** – Right – Show markers on right / floating
- **M** – Mark – mark all traces with markers
- **L** – Legend – hide/show the legend
- **P** – Pan – use the pan (scrolling) tool
- **X** – Zoom – use the zoom tool (zoom to mouse rectangle)
- **Z** – Zoom to fit – Maximize the view
- **Tab** – Select the next marker.
- **Shift+Tab** – Select the previous marker.
- **Enter** – Bring up the Marker Properties window. If no marker is selected, it brings up the Graph Properties instead.
- **Delete** – Delete the currently selected marker.
- **Shift+Delete** – Delete all markers (you are asked to confirm the deletion before deleting the markers).
- **Arrow Keys** – The up, down, left, and right arrow keys have several functions, based on the currently selected marker's style.
    - **Standard Marker** – Move the reference frequency left or right on the graph.
    - **Peak Marker** – Move to the next peak (if any).
    - **Valley Marker** – Move the marker to the next valley (if any).
    - **Bandwidth Marker** – Move the relative markers to increase or decrease the bandwidth. This changes the delta values of the child relative markers, so each arrow key action does not always move the marker by a single data point.
    - **Delta Marker** – Increase or decrease the relative delta. This changes the dB Down value of the marker, so each arrow key action does not always move the marker by a single data point.
- **Ctrl+Arrow Keys** – Pan (scroll) the chart up, down, left, or right.
- **Shift+Arrow Keys** – Move the marker up or down to the next trace on the graph (if any).
- **Ctrl+Shift+S** – Change the current marker's style to Standard.
- **Ctrl+Shift+P** – Change the current marker's style to Peak.
- **Ctrl+Shift+V** – Change the current marker's style to Valley.
- **Ctrl+Shift+B** – Change the current marker's style to Bandwidth.
- **Ctrl+Shift+L** – Change the current marker's style to Delta Left.
- **Ctrl+Shift+R** – Change the current marker's style to Delta Right.

Layout Keystroke Commands

- **A** – Arc
- **B** – Box (rectangle)
- **C** – Component
- **D** – Redraw
- **E** – Empower port
- **F** – Flip selected line or arc
- **G** – Group
- **L** – Line (Wire)
- **O** – Toggle orthogonal line mode
- **P** – Pour
- **Shift+P** – Polygon
- **R** – Toggle pounded (pads or lines)

- **Shift+R** – Switch to ruler tool
- **T** – Text
- **U** – Ungroup
- **V** – Via
- **Shift+W** – Wire (line)
- **Z** – Zoom to fit
- **0** (Zero) – Pad
- **F11** – Show layout preferences

LiveReport Keystroke Commands

- **A** – All Zoom - zoom to page
- **P** – Pan - use the pan (scrolling) tool
- **X** – Zoom - use the zoom tool (zoom to mouse rectangle)
- **W** – Zoom to Width
- **Z** – Zoom to fit - Maximize the view
- **Tab** – switch to next window
- **Shift+Tab** – switch to previous window
- **1, 2, 3, 4, 5, …** – switch to nth window (zooms to fit specified window)

Schematic Keystroke Commands

- **Enter** – Bring up part properties or place parts moved using the arrow keys
- **A** – Places an ammeter (CURRENT_PROBE)
- **B** – BLOCK (two-port)
- **C** – CAPQ (capacitor with Q)
- **Shift+C** – CAPACITOR (ideal)
- **D** – DIODE
- **G** – GROUND
- **I** – INPUT Port
- **L** – INDQ (inductor with Q)
- **Shift+L** – INDUCTOR (ideal)
- **M** – MLI_SLI
- **N** – BIP_NPN
- **O** – OUTPUT port
- **P** – Use the Pan (scrolling) tool
- **Q** – SQUARE_BLOCK (attached to a design)
- **R** – RESISTOR
- **S** – SIGNAL_GROUND
- **TEE** – Microstrip T
- **V** – Voltage TEST_POINT
- **W** – 90 degree WIRES (Shift+W for any angle wires)
- **Shift+W** – Angled WIRE
- **X** – Zoom - use the zoom tool (zoom to mouse rectangle)
- **Y** – CRYSTAL
- **Z** – Zoom to show all parts (zoom to fit)
- **Shift+Z** – Zoom to show all parts (with extra margin)
- **F4** – Rotate the text origin of part parameters
- **1**, **2**, **3**, …, **0** – Place 1-port, 2-port, …, 10-port

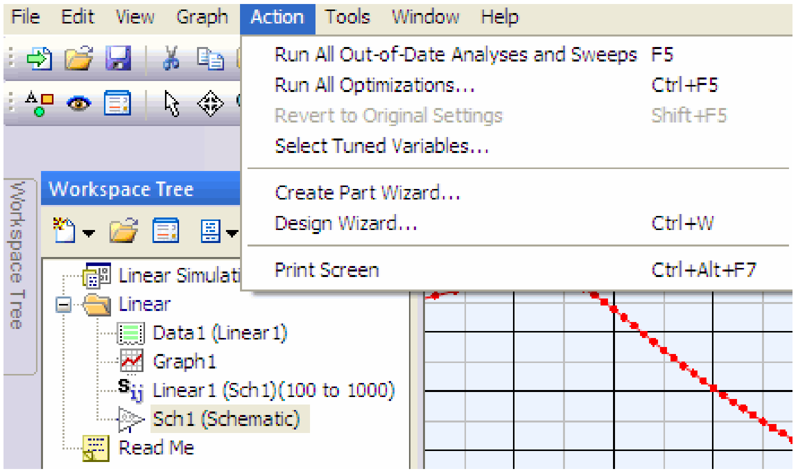
Appendix B - Menus

- *Action Menu* (users)
- *Edit Menu* (users)
- *Equations Menu* (users)
- *File Menu* (users)
- *Graph Menu* (users)
- *Help Menu* (users)
- *Layout Menu* (users)
- *LiveReport Menu* (users)
- *Notes Menu* (users)
- *PartList Menu* (users)
- *Schematic Menu* (users)
- *Scripts Menu* (users)
- *Tools Menu* (users)
- *View Menu* (users)
- *Window Menu* (users)

Action Menu

Use this menu to calculate variables or to access the Create Part, Design, or Source wizards.

To open: Click the Action button on the menu.



1. **Calculate** – Calculate the out-of-date simulations.
2. **Calculate All Optimizations** – Run all the optimizations.
3. **Revert to Original Settings** – Return all tuned variables to their "Original" values.
4. **Select Tuned Variables** – Make any parameter from a master list tunable.
5. **Create Part Wizard** – Run the part creation wizard. Use this to create a new part based on existing parts or from scratch by defining the model and symbol for the part.
6. **Design Wizard** – Run the design creation wizards. Use this to create Models, Symbols, and general Designs.
7. **Print Screen** – print the current screen.

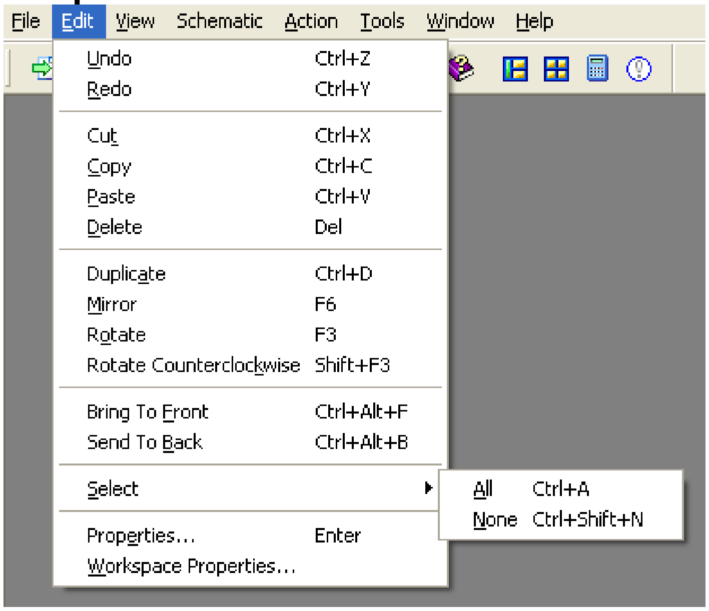
Design Menu

Designs do not have a menu of their own, instead the current tab page menu is displayed (Schematic, PartList, etc.)

Edit Menu

Use this menu to perform basic editing functions, such as undo, redo, cut, paste, copy, and delete.

To open: Click the Edit button.

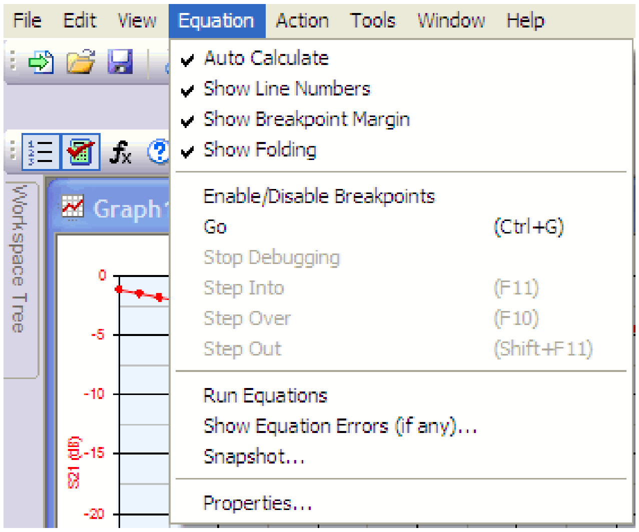


1. **Undo** – Reverse previous editing. Multi-level undo is available in a schematic or layout.
2. **Redo –** Put back changes that were previously reversed with Undo.
3. **Cut –** Copy the selected object and delete it.
4. **Copy –** Copy the selected object. The selection is not deleted.
5. **Paste** – Paste the last copied object into the current schematic, layout, text, etc.
6. **Delete –** Delete the selected object.
7. **Duplicate** – Duplicate the selected object. This is equivalent to a copy-and-paste sequence.
8. **Mirror** – Flip the selected object about its horizontal or vertical axis. Mirror is not available for layouts, because it yields backward parts.
9. **Rotate** – Rotate the selected object by the Part Constrain angle specified in the Global Schematic Options window.
10. **Bring To Front** – Moves the selected item(s) in front of the other items in the window.
11. **Send To Back –** Moves the selected item(s) behind the others.
12. **Rotate Counterclockwise** – Rotate the selected object counterclockwise.
13. **Select** – Display a submenu allowing easy access to commonly used objects
14. **All** – Select all objects in the schematic or layout.
15. **None** – Turn off all selected objects.
16. **Properties** – Open properties for active object.
17. **Workspace Properties** – Open workspace properties.

Equations Menu

Use this menu to access equations commands.

To open: Click the Script button on the menu.



1. **Active** – When checked, this equation is available for use.
2. **Auto Calculate** – When checked, these equations will recalculate while typing

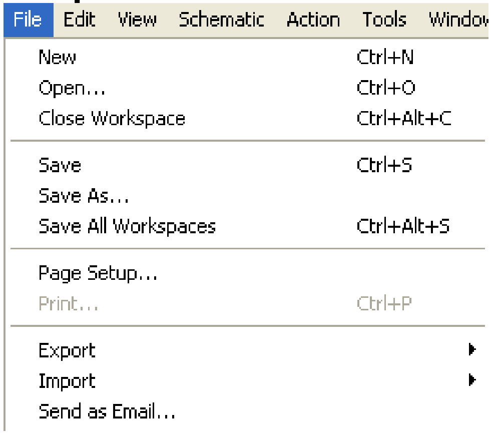
⚠ Caution: be careful not to write infinite loops if this option is checked

3. **Show Line Numbers** – Shows / hides line numbers in the equations window.
4. **Show Folding** – Shows / hides the folding bar in the equations window (next to line numbers). When enabled, the folding bar can be used to expand/contract blocks of code, such as if / then / else sections.
5. **Equation Wizard** – Runs the Equation Wizard.
6. **Run Equations** – Executes the equation block.
7. **Show Equation Errors** – Helps diagnose equation errors.
8. **Snapshot** – Create a dataset with static variables that capture the current state of the equation block. Use it save reference variables, such as when the equation block is dependent on an analysis that gets re-run and you want to keep around old results in the workspace.
9. **Properties** – Shows the Equation's Properties dialog box

File Menu

Use this menu to open, close, save, or print designs. You can also import or export files, and exit.

To open: Click the File button on the menu.



1. **New –** Close the current workspace and open a new workspace. If you select the Allow Multiple Open Workspaces option on the General Global Options page, the current workspace remains open.

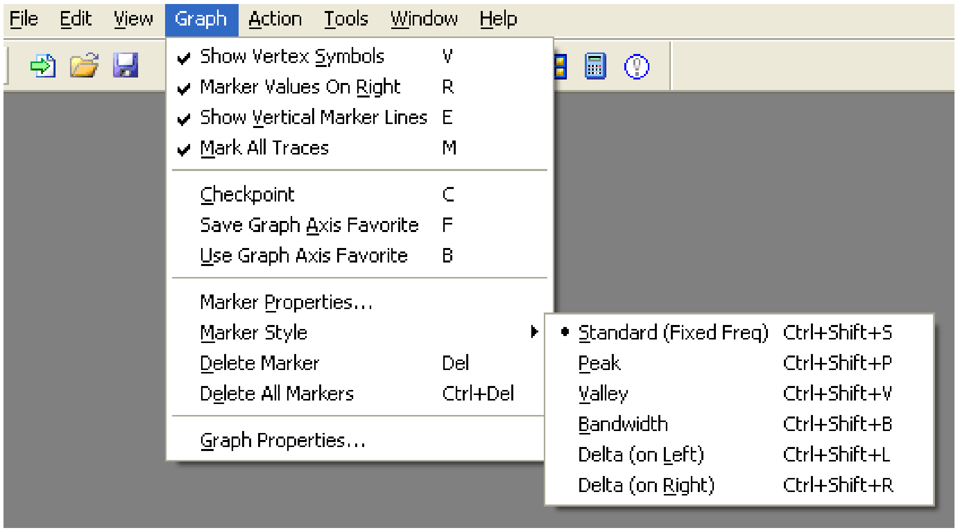
2. **Open –** Opens a new workspace.
3. **Close Workspace** – Close the current workspace.
4. **Save –** Save the current workspace. If the current file has not been previously saved, you will be prompted for a file name.
5. **Save As** – Save the current workspace into a new file.
6. **Save All Workspaces –** Save all loaded workspaces.
7. **Page Setup** – Select printer and settings.
8. **Print –** Print the active window.
9. **Export** – Display a submenu allowing access to all of the Export options.
   - **ASCII Drill List –** Export an ASCII (Text) X-Y drill hole list from the current layout.
   - **Export Schematics to ADS –** Export a schematic directly ADS.
   - **Bill of Materials –** Export a bill of materials from the current layout.
   - **Bitmap (Active Window)** – Export the active window.
   - **Bitmap (Entire Screen)** – Export the entire screen, including any applications outside the window.
   - **DXF/DWG File** – Export a DXF/DWG File from the current layout.
   - **GDSII File** – Export a GDSII File from the current layout.
   - **Gerber File** – Export a Gerber File from the current layout. Available formats are Gerber 274-D and 274-X.
   - **HPGL File** – Export a HPGL File from the current layout.
   - **IFF Schematic File** – Export an ADS compatible IFF schematic file.
   - **Part Placement List** – Export a part placement list from the current layout.
   - **S-Parameters** – Export a device data file from a simulation.
   - **SPICE File** – Export a SPICE file from the current schematic.
   - **Touchstone File** – Export a Touchstone file from the current schematic.
   - **XML File** – Export the published properties to an XML file.
10. **Import** – Display a submenu allowing access to all of the Import commands.
    - **DXF/DWG File** – Import a DXF/DWG File into the current layout using Set Layer Mapping.
    - **GDSII File** – Import a GDSII File into the current layout using Set Layer Mapping.
    - **Genesys Netlist** –
    - **Gerber File** – Import a Gerber 274-X File into the current layout.
    - **LoadPull Data File** – Display a submenu of Load Pull import options.
    - **M-File** – Import an M-file.
    - **Directory of M-Files** – Import all M-files in a directory
    - **S-Data file** - Import an S Parameter file in Touchstone format.
    - **SPICE File** – Import a SPICE file.
    - **XML** – Import an XML file.
    - **6.x Model Library** – Import a model library that was created in Genesys Version 6.5B or earlier.
    - **Old Genesys S-Data file** - For 3+ ports Genesys S-parameters were saved transposed from the Touchstone standard. Import this old format.
    - **CITI File** – Import a Common Instrumentation Transfer and Interchange (CITI) file.
11. **Send as Email** – Send the current workspace as an email attachment using your email program.

Graph Menu

The **Graph** menu and toolbar appear and disappear on whether or not a graph is the principal focus. The work carried out by the menu items or toolbar buttons are common to all types of graphs including accessing a particular graph's *Graph Properties* (users). Another way to get to **Graph Properties** is by right-clicking on the graph and selecting it on the pop-up menu.

Use this menu to specify various graph settings.

To open: Click the Graph button on the menu. (This menu appears only when a graph window is active.)

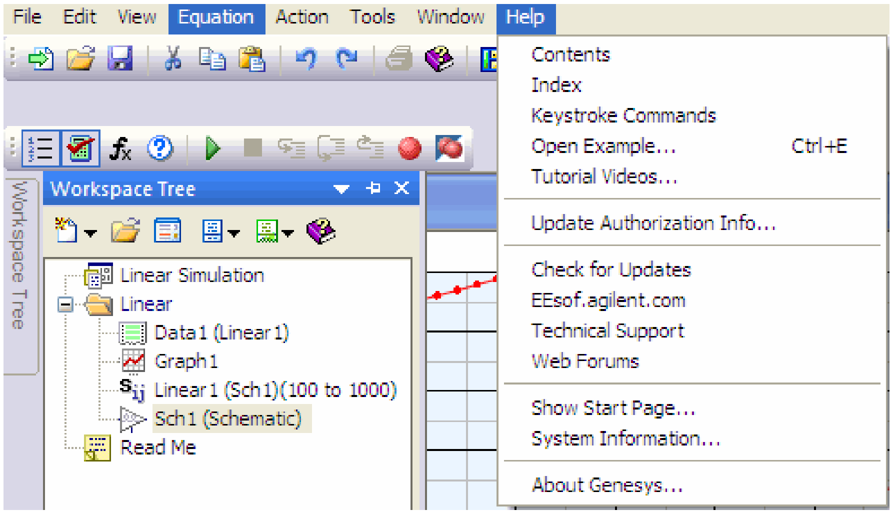

File
Edit
View
Graph
Action
Tools
Window
Help
✓ Show Vertex Symbols V
✓ Marker Values On Right R
✓ Show Vertical Marker Lines E
✓ Mark All Traces M
Checkpoint C
Save Graph Axis Favorite F
Use Graph Axis Favorite B
Marker Properties...
Marker Style
Delete Marker Del
Delete All Markers Ctrl+Del
Graph Properties...
• Standard (Fixed Freq) Ctrl+Shift+S
Peak Ctrl+Shift+P
Valley Ctrl+Shift+V
Bandwidth Ctrl+Shift+B
Delta (on Left) Ctrl+Shift+L
Delta (on Right) Ctrl+Shift+R

1. **Show Vertex Symbols** – Show or hide the vertex symbols on the trace.
2. **Marker Values On Right --** Place marker values on the right of the graph.
3. **Show Vertical Marker Lines** – Show or hide the vertical marker lines.
4. **Mark All Traces --** Place markers on all traces.
5. **Checkpoint --** Remove all current checkpoint traces if there are any. Create one if there are none.
6. **Marker Properties** – Open the Marker Properties window.
7. **Marker Style** – Display a submenu allowing easy access to commonly used marker styles.
8. **Standard (Fixed Frequency)** – Place a maker on the graph at the sport where you clicked.
9. **Peak** – Place a marker at the highest point on the trace.
10. **Valley** – Place a marker at the lowest point on the trace.
11. **Bandwidth** – Placer a marker on the trace to indicate bandwidth.
12. **Delta (On Left)** – Place a marker left of the trace to indicate the relative offset specified in the Marker Properties window.
13. **Delta (On Right)** – Place a marker right of the trace to indicate the relative offset specified in the Marker Properties window.
14. **Delete Marker** – Delete the currently selected marker.
15. **Delete All Markers** – Delete all the markers on the current graph; it prompts yes/no before actually deleting the markers.
16. **Graph Properties** – Open the Graph Properties window.

Help Menu

Use this menu to check for the latest update, get quick access to the Agilent Web site, or get help.

To open: Click the Help button on the menu.

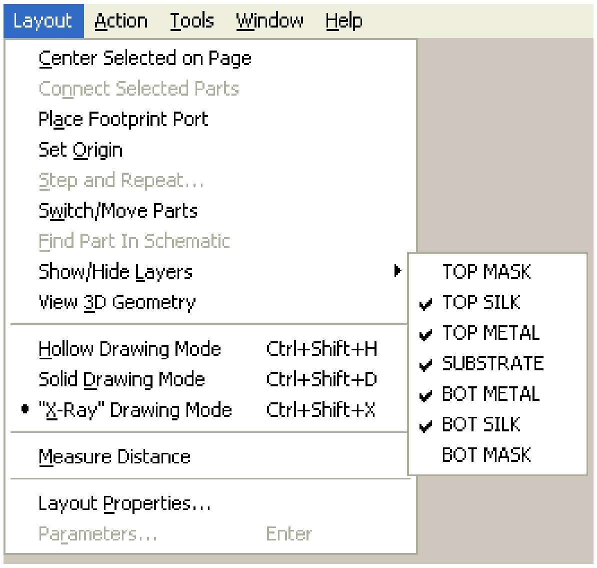


1. **Contents** – Open the Help contents.
2. **Index** – Open the Help index.
3. **Keystroke Commands** – Open a Help topic containing information about all of the keystroke commands.
4. **Open Example** – Open an example workspace.
5. **Tutorial Videos** – Select and watch a collection of short, helpful videos.
6. **Update Authorization Information** – Open a page where you can start the authorization process.
7. **Check for Updates** – Open a Web page to check for updates.
8. **Agilent.com** – Open the Agilent Web site.
9. **Technical Support** – Open the technical support Web page.
10. **Web Forums** – Open the Web page to access one of the forums.
11. **Show Start Page** – Open the Start page.
12. **System Information** –
13. **About** – Open a page with information about the program.

Layout Menu

Use this menu to specify drawing modes or change layout properties.

To open: Click the Layout button on the menu. This menu appears only when a layout window is active.



1. **Center Selected on Page** – Center the selected parts on the layout page.
2. **Connect Selected Parts** – Connects the selected parts, by placing them together (so that they are touching).
3. **Place Footprint Port** – If the layout is a footprint, place a port.
4. **Set Origin** – Place the origin on a layout. (The origin is initially placed in the lower left corner of the layout.)
5. **Step and Repeat** – Creates a row/column matrix, by copying the current selection and pasting it repeatedly. Use this feature to create rows of evenly spaced pads, DIP packages, etc.
6. **Switch/Move Parts** – Used with packages containing parts from more than one schematic, to move parts from one schematic to another.
7. **Find Part In Schematic** – Searches the schematic for currently selected Layout component.
8. **Show/Hide Layers** – Checked Layers (on the popout menu) will be displayed; unchecked layers will be hidden.
9. **View 3D Geometry** – Displays a 3-Dimensional view of your layout. For more details, see note below.
10. **Hollow Drawing Mode** – Emphasize edges of lines, polygons, and other objects.
11. **Solid Drawing Mode** – Use classic opaque fill of lines, polygons, and other objects. This is the fastest drawing mode.
12. **"X-Ray" Drawing Mode** – Use semi-transparent fill of lines, polygons, and other objects. This technique shows overlapping items best.
13. **Enable Adjustable TLines** – Allow transmission lines to be resized by dragging handles.
14. **Measure Distance** – Measure the distance between two points.
15. **Layout Properties** – Open the Layout Properties window.
16. **Parameters** – Open the properties window for the selected part.

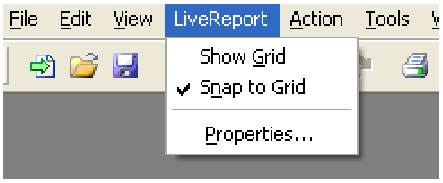
Note
To see currents or farfield data displayed in a 3D view:

- set up a Momentum Analysis to generate desired data
- run the anakysis
- right-click it on the workspace tree
- select **View 3D Geometry** from the menu

LiveReport Menu

Use this menu to set LiveReport options. (A LiveReport is a *living* notebook page that collects live views of schematics, graphs, equations, notes, and tables into a single page.)

To open: Click the Schematic button on the menu. This menu appears only when a schematic window is active.

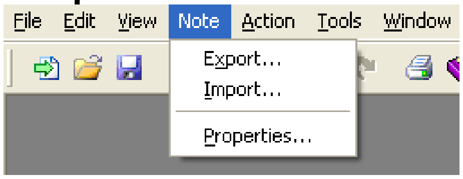


1. **Show Grid** – Show or hide the background grid.
2. **Snap to Grid** – Toggles (enables / disables) mouse cursor snap-to-grid (constrains mouse coordinates to the grid).
3. **Properties** – Shows the LiveReport Properties dialog box, which allows you to specify settings such as Page Width and Height, Paper Orientation, Margins, Headers, and Footers.

Notes Menu

Use this menu to access Note commands.

To open: Click the Action button on the menu.



1. **Export** – Export the Note's text.
2. **Import** – Import text into the note.
3. **Properties** – Shows the Note's Properties dialog box

> In order for the **Note** menu to reveal, the Notes page must be the current selected window (either open or minimized) in the Genesys workspace area.

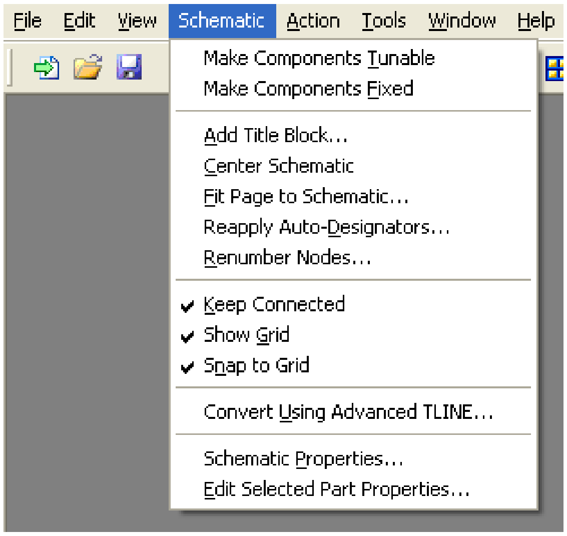
PartList Menu

The PartList has a single item:
Properties – Open the Properties window.

Schematic Menu

Use this menu to set component and schematic options.

To open: Click the Schematic button on the menu. This menu appears only when a schematic window is active.



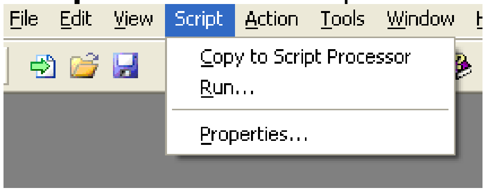
1. **Make Components Tunable** – Force selected components to be tunable or optimizable by adding question marks (?) to the first value of each component. This only adds question marks to part values with a numerical value. If a variable is used for a particular value, it is not made tunable.
2. **Make Components Fixed** – Force selected components to be non-tunable by removing any question marks that were added to the first value of each component. This only removes question marks on part values with a numerical value.
3. **Add Title Block** – Adds a schematic title block to the page, so that the schematic can be documented.
4. **Center Schematic** – Center the schematic on the page.

5. **Fit Page to Schematic** – Resize the page to fit all the parts within it. Note that you can also change the standard part length in a schematic to have parts shrink to fit a specific page size.
6. **Reapply Auto-Designators** – Reassign standardized designators to selected components. A designator is a part name like R1 or C3. The Auto-Designator feature builds component names by using the appropriate designator prefix (like R for a resistor or C for a capacitor) and appending a unique sequence number to the end. When you use this command, the designators are applied in geometric order, from left to right.
7. **Renumber Nodes** – Renumber all nodes in the schematic, regardless of any selection. When you use this commend, the nodes are numbered in geometric order, from left to right. Nodes that connect to a port are set to match the port number (if that option is enabled). This is primarily useful before exporting a SPICE file.
8. **Bring to Front** – Move the selected objects to the front.
9. **Send to Back** – Move the selected objects to the back.
10. **Keep Connected** – Allow wires to remain connected to components as they are moved. The ALT key temporarily toggles this function as long as the key is held down.
11. **Show Grid** – Show or hide the schematic grid.
12. **Snap to Grid** – Toggles (enables / disables) mouse cursor snap-to-grid (constrains mouse coordinates to the grid).
13. **Convert Using Advanced TLine** – Convert all electrical transmission line parts to physical transmission line parts using Advanced TLine (for example, microstrip, stripline, coplanar, or coax). This allows discontinuities to be added and automatically compensated for. Also, substrates can be converted from one to another.
14. **Schematic Properties** – Shows the Schematic Properties dialog box, which allows you to specify settings such as Page Width and Height, Title, Company Name, and Company Address.
15. **Edit Selected Part Properties** – Shows the Part Properties dialog box, which allows you to specify parameters and settings for the selected part.

Scripts Menu

Use this menu to access scripting commands.

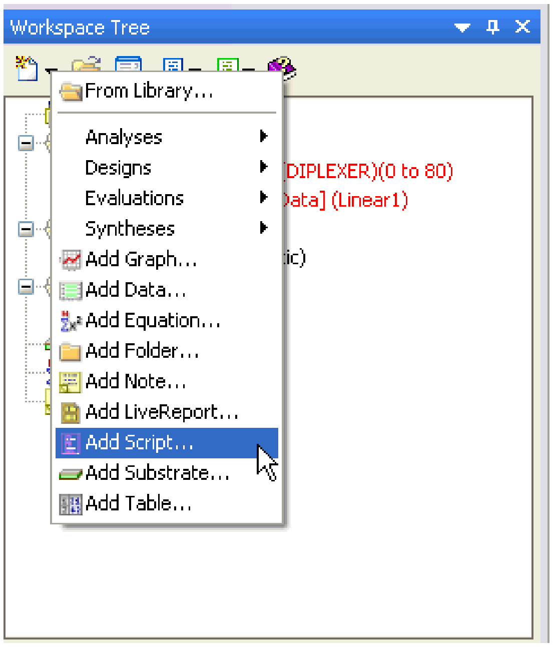
To open: Click the Script button on the menu.

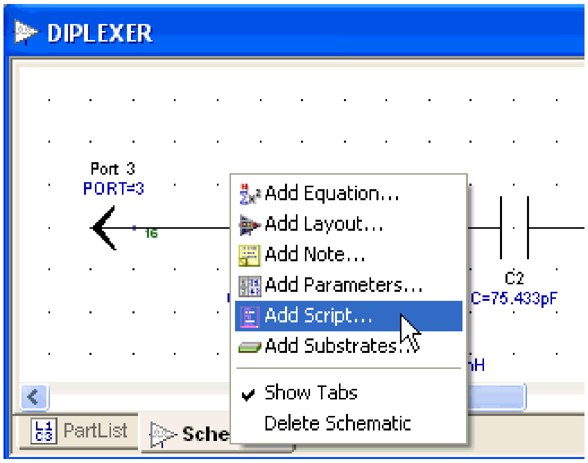


1. **Copy to Script Processor** – Copies the script to the Script Processor window.
2. **Run** – Executes the script.
3. **Properties** – Shows the Script's Properties dialog box

The script menu shows only when a script page is present.

To add a script page, use the "New Item" button or Right-Click an existing schematic tab. See images below.

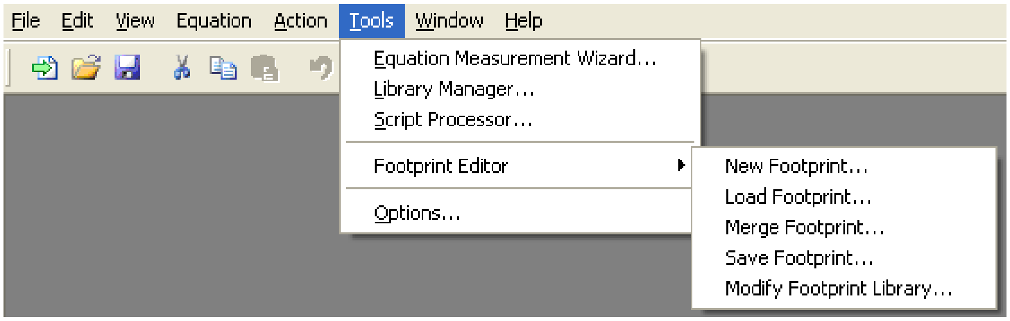




Tools Menu

Use this menu provides access some common design tools or change the global options.

To open: Click the Tools button on the menu.



1. **Equation Measurement Wizard** – Open the Equations Measurement wizard. This option is available when the Equation window is active.
2. **Library Manager** – Open the Library Manager window.
3. **Script Processor** – Open the Script Processor window.
4. **Footprint Editor** – Edit layout component footprints.
    - **New Footprint** – Create a new, blank footprint.

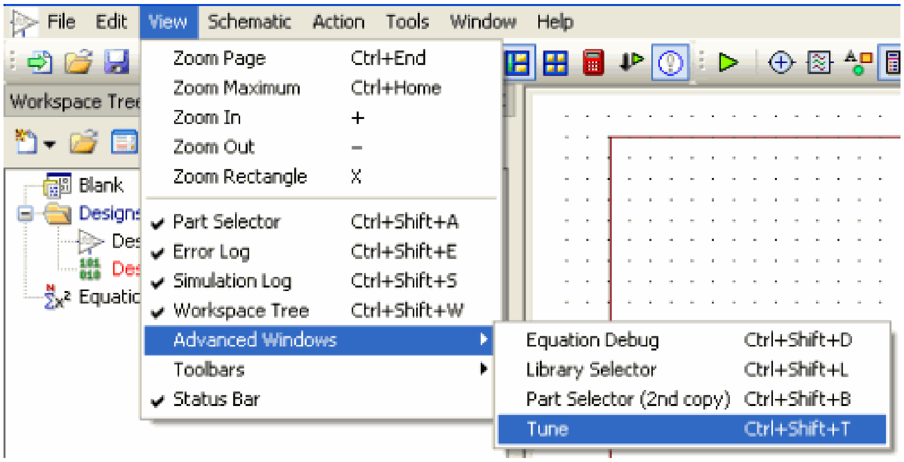
   - **Load Footprint** – Load an existing footprint for editing.
   - **Merge Footprint** – Adds an additional footprint to the footprint that is currently being edited.
   - **Save Footprint** – Saves the current footprint into a footprint library. Choose <New Object> to add a new footprint to the library -or- select an existing footprint to replace.
   - **Modify Footprint Library** – Allows footprints in a library to be renamed or deleted.

5. **Options** – Open the Global Options window.

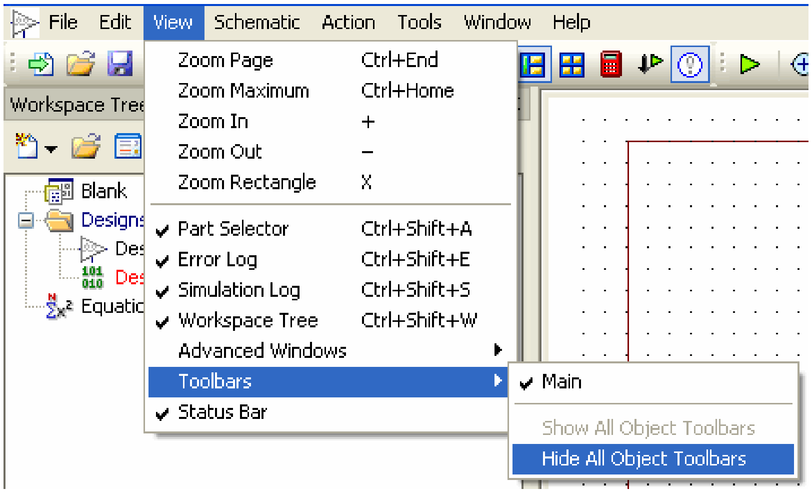
View Menu

Use this menu to adjust the size of your window. This menu can also be use to show or hide docking windows or toolbars.

To open: Click the View button on the menu.



- **Zoom In** – Zoom in on the center of the window.
- **Zoom Out** – Zoom out from the center of the window.
- **Zoom Page** – Zoom to fit the page.
- **Zoom Maximum** – Zoom to fit all objects or traces.
- **Zoom Rectangle** – Allow you to draw a rectangle to zoom in on.
- **Part Selector** – Show or hide the Part Selector.
- **Error Log** – Show or hide the Error Log.
- **Simulation Log** – Show or hide the Simulation Log.
- **Workspace Tree** – Show or hide the Workspace tree.
- **Advance Windows** – Show a secondary list of docking windows.
  - **Equation Debug** – Show or hide the Equation Debug window.
  - **Library Selector** – Show or hide the Library(Design) Selector.
  - **Part Selector (2nd copy)** – Show or hide a second copy of the Part Selector.
  - **Tune** – Show or hide the Tune window, which lists and controls tune variables.

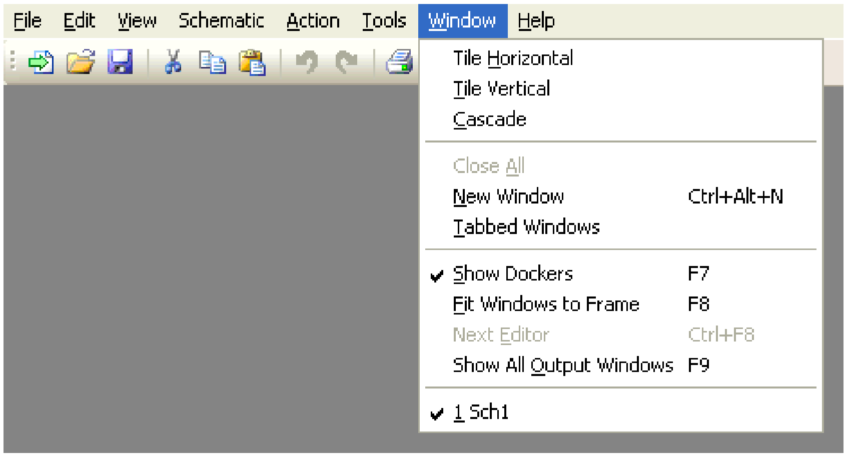


- **Toolbars** – Choose how toolbars are shown.
    - **Main** – Show or hide the Main toolbar.
    - **Show All Object Toolbars** – Show toolbars for the active object.
    - **Hide All Object Toolbars** – Hide toolbars for the active object.
- **Status Bar** – Show or hide the status bar at the bottom of the main window.

Window Menu

Use this menu to organize or open a window. You can also use this menu to close all open windows at the same time.

To open: Click the Window button on the menu.



1. **Tile Horizontal** – Tile open windows above each other.
2. **Tile Vertical** – Tile open windows beside each other.
3. **Cascade** – Arrange open windows in an overlapping style.
4. **Close All** – Close all open windows.
5. **New Window** – Open a new design window.
6. **Tabbed Windows** – Switches between tabbed and overlapping document window styles.
7. **Show Dockers** – Show / hide vertical dockers (Tune, Workspace Tree, Part Selector, etc.).
8. **Fit Windows to Frame** – Resizes the open windows to fit the non-docker area.
9. **Next Editor** – Toggle between editor windows (schematics, layouts, equation editors).
10. **Show All Output Windows** – Open all output windows (graphs, tables, variable

viewers).

11. **Numbered Window List** – A pick-list of all open document windows. Select one to make it active (current).

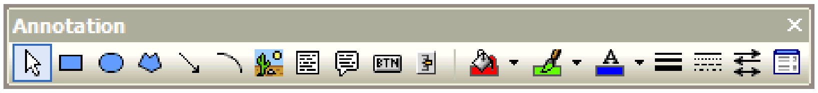
Appendix C - Toolbars

- *Annotation Toolbar* (users)
- *Basic Toolbar* (users)
- *Coax Toolbar* (users)
- *CoPlanar Toolbar* (users)
- *Dataset Toolbar* (users)
- *Equation Toolbar* (users)
- *Graph Toolbar* (users)
- *Layout Toolbar* (users)
- *Linear Toolbar* (users)
- *LiveReport Toolbar* (users)
- *Lumped Toolbar* (users)
- *Main Toolbar* (users)
- *Microstrip Toolbar* (users)
- *Nonlinear Toolbar* (users)
- *Notes Toolbar* (users)
- *Part Groups Toolbar* (users)
- *Schematic Toolbar* (users)
- *Script Toolbar* (users)
- *Slabline Toolbar* (users)
- *Spectrasys Toolbar* (users)
- *Stripline Toolbar* (users)
- *Table Toolbar* (users)
- *TLine Toolbar* (users)
- *Wave Toolbar* (users)

Annotation Toolbar

Use this toolbar to add basic drawing objects, such as lines, circles, or arrows, to a design or to modify the selected annotations by changing the color, dashed line style, etc.

To open: Click the Annotation button ( ) from any design window toolbar, e.g. schematic window toolbar.



1. **Select** – Select an object.
2. **Rectangle** – Draw a square or rectangle.
3. **Ellipse** – Draw a circle or ellipse.
4. **Polygon** – Draws a filled polygon or unfilled polyline.
5. **Arrow** – Draw a line or arrow. Change the arrow style by selecting a line and picking an arrow type from Arrows button menu.
6. **Arc** – Draw an arc.
7. **Picture** – Insert a picture. Use this annotation to add a company logo to a graph, for example. Double-click the new object and select a JPG, GIF, or BMP image file to be displayed. To allow all users to see the image, the bitmap file should reside on a network server.
8. **Text** – Place text. Text has a number of settings. Double-click a text annotation to set the horizontal and vertical justification (text alignment). The name of the text item can be changed and shown on-screen, which simplifies building a schematic title block.
9. **Text Balloon** – Draw a text balloon. This annotation has a "tail" which can be

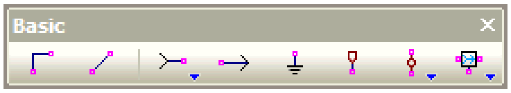
anchored to a data point on a graph, to the page, or not anchored (using the right-button menu).

10. **Button** – Draw a user button (widget). This annotation can be "clicked" to run a custom script, which is specified by double-clicking the outer EDGE of the button control. The middle of the button runs the script.
11. **Slider** – Draw a slider control (widget). This annotation is linked to a tunable parameter and functions much like the Tuning Window.
12. **Fill Color** – Set the fill color. Use the 3 color buttons to change the colors of the selected annotations. New annotations will be created using the current colors. The bottom-right color swatch (with a diagonal slash) is transparent, which specifies an unfilled object.
13. **Line Color** – Set the line color. The bottom-right color swatch (with a diagonal slash) is transparent, which specifies a object with no outline.
14. **Text Color** – Set the text color.
15. **Line Thickness** – Set the width of borders and lines.
16. **Line Style** – Set the drawing style of borders and lines (dash pattern, etc.).
17. **Arrows** – Set the arrow style of lines.
18. **Properties** – Display the properties window for the selected part.

Basic Toolbar

Use this toolbar to place basic components.

To open: Click the Basic button on the Schematic Toolbar.



Draw a 90-degree Connection Line (W)

90-degree Connection Line (W)

Draw a Angled Connection Line (Shift+W)

Connection Line (Shift+W)

Place an Input (I)

*Input: Standard (*INP)* (part)
Input: AC Power (PAC) (part)
Input: DC Voltage (part)
Input: DC Current (part)
Input: AC Voltage (part)
Input: AC Current (part)
Input: Pulsed Voltage (part)
Input: Pulsed Current (part)
Input: Custom Voltage Waveform (part)
Input: Custom Current Waveform (part)
Input: AC Power (PAC) with uniform Noise (part)
Input: Noise Power (uniform) (part)
Input:AC Power (PAC) with Custom Noise (part)
Input: Custom Noise Power (part)

Place an Output (O)

Output (O) (part)

Place a Ground (G)

Ground (G) (part)

Place a Signal Ground/DC Source (S)

Signal Ground/DC Source (S) (part)

Place a Source, Current Probe or Test Point

Source: AC Power (PAC) (part)
Source: DC Voltage (part)
Source: DC Current (part)
Source: AC Voltage (part)
Source: AC Current (part)
Source: Pulsed Voltage (part)
Source: Pulsed Current (part)
Source: Custom Voltage Waveform (part)
Source: Custom Current Waveform (part)
Source: Noise Voltage (part)
Source: Noise Current (part)
Current Probe (Ammeter) (part)
Voltage Test Point (part)
Oscillator Port (part)

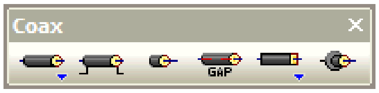
Place a Reused Schematic

Subcircuit 1-Port
Subcircuit 2-Port
Subcircuit 3-Port
Subcircuit 4-Port
Subcircuit 5-Port
Subcircuit 6-Port
Subcircuit 7-Port
Subcircuit 8-Port
Subcircuit 9-Port
More Subcircuit Models...

Coax Toolbar

Use this toolbar to insert Coaxial parts in the schematic.

To open: Click the Coax button on the Schematic Toolbar.



CLI: Coaxial Line

Coaxial Line (CLI) (part)
Coaxial Cable (part)
RG6 Cable (part)
RG8 Cable (part)

RG9 Cable (part)
RG58 Cable (part)
RG59 Cable (part)
RG214 Cable (part)

CLI4: 4 - Terminal Coaxial Line

CLI4: 4 - Terminal Coaxial Line (part)

CEN: Coaxial End Effect

CEN: Coaxial End Effect (part)

CGA: Coaxial Gap

CGA: Coaxial Gap (part)

CSQLI,CSQLX: Square Coaxial Lines

Square Coaxial Line (Round Inner Conductor) (part)
Square Coaxial Line (Square Inner Conductor) (part)

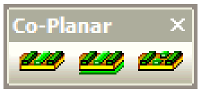
CST: Coaxial Step

CST: Coaxial Step (part)

Co-Planar Toolbar

Use this toolbar to place co-planar parts.

To open: Click the Co-Planar button on the Schematic Toolbar.



CPW: Coplanar Line

CPW: Coplanar Line (part)

CPWG: Coplanar Line with Ground

CPWG: Coplanar Line with Ground (part)

GPWCGAP: Coplanar Line with Gap

GPWCGAP: Coplanar Line with Gap (part)

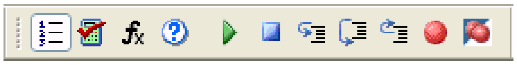
Dataset Toolbar

Use this toolbar to interact with the active *Dataset* (users) and adjust its settings.

1. **Properties** - Brings up the Dataset Properties dialog.
2. **Save** - Export/save the dataset to a file.

Equations Toolbar

Use this toolbar to change the Equation window display options and debug your equations. This toolbar automatically displays when you have an Equation window active.



The icons are:

Show or hide the Line Numbers margin

Turn autocalculate on/off

Bring up the Equation Wizard

Display Errors for this set of equations

Go - run the equations, or continue on from a breakpoint (F5 or Ctrl_G)

Stop - stop debugging (abort execution). This button is only enabled while breakpointed.

Step Into - step inside a function and break at the first line of execution in the function (F11). This button is only enabled while breakpointed.

Step Over - execute statements on the current line (F10). This button is only enabled while breakpointed.

Step Out - run until the current function ends, then break at the next line (the caller) (Shift_F11). If there is no function call at the current line, or the equation processor cannot step into the function, then all statements on the current line are simply executed. This button is only enabled while breakpointed.

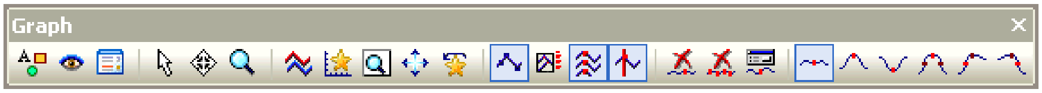
Add or Remove a breakpoint from the current line (F9 or Ctrl_B)

Toggle all existing breakpoints to either the "enabled" or "disabled" state

Graph Toolbar

Use the toolbar for Graph functions.

To open: Open a graph window.



1. **Annotation** – Display the Annotation Toolbar toolbar.
2. **Eye** – Hide/Show graph traces in a pulldown menu
3. **Graph Properties** – Display the Graph Properties window.
4. **Select** – Select an object.
5. **Pan** – use the Pan tool to pan the graph (left-right for rectangular graphs, free for polar and smith charts).
6. **Zoom** – zoom in on a selected part of the graph.
7. **Checkpoint** – Add a checkpoint if there is none. Remove all current checkpoint traces if there are any
8. **Add Axis Favorite** – Save the current axis settings into the Axis Favorite list.
9. **Zoom to Page** – Zoom the graph data attractively to fit the page.
10. **Maximize** – Zoom the graph data exactly to fit the page..
11. **Use Axis Favorite** – Set the axis settings to the last favorite in the list. Click again

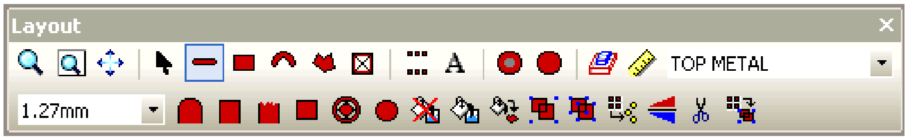
to cycle through the axis favorites.

12. **Toggle Vertex Symbols** – Show or hide trace vertex symbols (large dots on traces).
13. **Marker Values On Right** – Place marker text in right margin of graph, or inline in graph.
14. **Mark All Traces** – Mark all traces on the graph.
15. **Toggle Vertical Marker Lines** – Show or hide dashed vertical marker lines at every marker position.
16. **Delete Marker** – Delete the selected marker.
17. **Delete all Markers** – Delete all markers on the current graph.
18. **Marker Properties** – Display the Marker Properties window.
19. **Standard Marker** – drop a standard marker or convert a selected marker to standard.
20. **Peak Marker** – Change marker style to Peak.
21. **Valley Marker** – Change marker style to Valley.
22. **Bandwidth Marker** – Change marker style to Bandwidth and insert two Delta markers.
23. **Delta Marker (On Right)** – Place a new Delta marker on the left side of the selected marker.
24. **Delta Marker (On Left)** – Place a new Delta marker on the right side of the selected marker.

Layout Toolbar

Use the toolbar to place components and edit in Layout. The Layout toolbar automatically displays when a Layout is active. Note that many of these buttons are tool specific; they will only appear when it is appropriate.

To open: Open a layout window.



1. **Select** – Select an object.
2. **Line** – Draw a line.
3. **Rectangle** – Draw a rectangle.
4. **Arc –** Draw an arc.
5. **Polygon** – Draw a polygon.
6. **Port** – Place an EMport in the layout window. (In the Footprint editor, this button places a footprint port.)
7. **Component** – Place a component footprint from the library.
8. **Text** – Place text.
9. **Viahole** – Draw a viahole, including pads.
10. **Pad** – Draw a round, square, or wagonwheel pad.
11. **View 3D Geometry** – Displays a 3-Dimensional view of your layout. For more details, see note below.
12. **Ruler** – Measure the distance between two specified points.
13. **Layer** – Displays layer name.
14. **Line Width** – Displays line width.
15. **Round End** – Switch to line round ends.
16. **Square End** – Switch to line square ends.
17. **Ambiguous Ends** – Revert multiple lines (within a selection) to back to the original line endings.
18. **Square** – Change pad type to wagon wheel.
19. **Wagon Wheel** – Change pad type to circle.

20. **Circle** – Change pad type to square.
21. **Unpour** – Unpour polygon.
22. **Pour** – Pour/repour the polygon.
23. **Convert Pour** – Convert pour to polygon.
24. **Group Objects** – Place the selected objects into a group.
25. **Ungroup Objects** – Ungroup the objects in the selection.
26. **Save as Footprint** – Save the selection as a footprint.
27. **Mirror** – Flip by 180 degrees along an axis (toggles between X and Y axis during use).
28. **Scissors** – Cut a hole in the polygon.
29. **Footprint to Group** – Convert footprint to group, to allow editing.

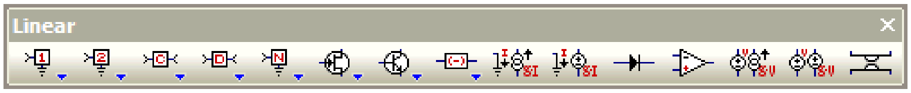
Note
To see currents or farfield data displayed in a 3D view:

- set up a Momentum Analysis to generate desired data
- run the anakysis
- right-click it on the workspace tree
- select **View 3D Geometry** from the menu

Linear Toolbar

Use this toolbar to insert linear parts.

To open: Click the Linear button on the Schematic Toolbar.



Each of the n-Port blocks has an option to load an s-parameter file (from the item's properties, available by double-clicking it in schematic view)

1-Port (S-Parameter)

1-Port (S-Param w/1 Terminal) (part)
1-Port (S-Param w/2 Terminal) (part)

2-Port (S-Parameter)

2-Port (S-Param w/2 Terminal) (part)
2-Port (S-Param w/BIP_NPN) (part)
2-Port (S-Param w/BIP_PNP) (part)
2-Port (S-Param w/N_FET) (part)
2-Port (S-Param w/P_FET) (part)
2-Port (S-Param w/BLOCK and Gnd) (part)
2-Port (S-Param w/Generic and Gnd) (part)
Manual ABCD-Parameters (BLOCK) (part)
Manual S-Parameters (BLOCK) (part)

Subcircuit (N-Port)

Subcircuit (w/NET2) (part)
Subcircuit (w/2-PortNoGnd) (part)

Dataset (S-Parameter N-Port)

Dataset 1-Port (S-Param) (part)
Dataset 2-Port (S-Param) (part)

Multi-Port (S-Parameter)

3-Port Data File (S-Param) (part)
4-Port Data File (S-Param) (part)
5-Port Data File (S-Param) (part)
6-Port Data File (S-Param) (part)
7-Port Data File (S-Param) (part)
8-Port Data File (S-Param) (part)
9-Port Data File (S-Param) (part)
10-Port Data File (S-Param) (part)
More File (S-Param) models...

FET Models

FET (S-Param w/N_FET) (part)
FET (S-Param w/P_FET) (part)
FET (N_FET) (part)
FET (P_FET) (part)

Bipolar Transistor

Bipolar (S-Param w/BIP_NPN) (part)
Bipolar (S-Param w/BIP_PNP) (part)
Bipolar (BIP_NPN) (part)
Bipolar (BIP_PNP) (part)

Negation /1 or 2 Port

Negation (NEG1) (part)
Negation (NEG2) (part)

Current Controlled Current Source

Current Controlled Current Source (part)

Current Controlled Voltage Source

Current Controlled Voltage Source (part)

PIN Diode

PIN Diode (part)

Op Amp

Op Amp (part)

Voltage Controlled Current Source

Voltage Controlled Current Source (part)

Voltage Controlled Voltage Source

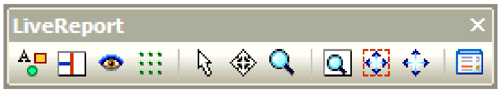
Voltage Controlled Voltage Source (part)

Gyrator

Gyrator (part)

LiveReport Toolbar

Use this toolbar to change the LiveReport and adjust its settings. The LiveReport toolbar automatically displays when you have a LiveReport active.

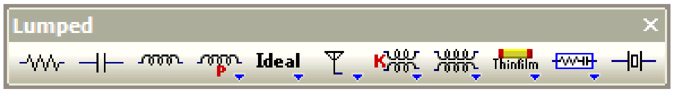


1. **Annotation** – Show/Hide the Annotation Toolbar.
2. **Arrange** – Brings up the Arrange Views dialog box, which repositions all the sub-objects.
3. **Eye** – Use this pull down menu to turn on/off text displays such as Titles, Headers, Footers, etc. on the LiveReport.
4. **Grid Snap** – enable/disable the grid snap
5. **Select** – Use the select tool to select views or annotations..
6. **Pan** – Use the pan (scrolling) tool to pan the schematic around (press the tool button and drag the LiveReport).
7. **Zoom** – Use the zoom tool to zoom into a rectangular region of the LiveReport (press the tool button and drag a rectangle).
8. **Zoom to Page** – Zoom to fit the page.
9. **Zoom to Fit Selection** – Zoom to fit the currently selected objects.
10. **Zoom to Fit All** (Maximize) – Zoom to fit all objects.
11. **Properties** – Opens the LiveReport properties dialog box.

Lumped Toolbar

Use this toolbar to place lumped parts.

To open: Click the Lumped button on the Schematic Toolbar.



Resistor (R)

Resistor (R) (part)

Capacitor (C)

Capacitor (C) (part)

Inductor (L)

Inductor (L) (part)

Physical Inductors

Air core inductor (AIRIND1) (part)
Inductor with Q (INDQ) (part)
Spiral inductor (SPIND) (part)
Toroidal core inductor (TORIND) (part)

Ideal parts

Ideal Gain Block (GAIN) (part)
Ideal Impedance Inverter (INV) (part)
Ideal Isolator (ISOLATOR) (part)
Ideal Phase Shift (PHASE) (part)
Ideal Three Port Circulator (CIR3) (part)
Ideal Time Delay Block (DELAY) (part)

Antenna parts

DIPOLE Antenna (part)
MONOPOLE Antenna (part)

Mutually Coupled Inductors

Two Mutually Coupled Inductors (part)
2 Mutually Coupled Coils (part)
3 Mutually Coupled Coils (part)
4 Mutually Coupled Coils (part)
5 Mutually Coupled Coils (part)
6 Mutually Coupled Coils (part)
7 Mutually Coupled Coils (part)
8 Mutually Coupled Coils (part)
9 Mutually Coupled Coils (part)
10 Mutually Coupled Coils (part)

Ideal Transformer

Transformer (part)
Center-Tapped Transformer (part)

Thin Film Resistor/Capacitor

Thin film capacitor (part)
Thin film resistor (part)

Lumped part Networks

Parallel L-C Resonator: Enter: f,C (PFC) (part)
Parallel L-C Resonator: Enter: f,L (PFL) (part)
Parallel Network: L-C (PLC) (part)
Parallel Network: R-L (PRL) (part)
Parallel Network: R-C (PRC)
Parallel Network: R-L-C (PRX) (part)
Series L-C Resonator: Enter: f,C (SFC) (part)
Series L-C Resonator: Enter: f,L (SFL) (part)
Series Network: L-C (SLC) (part)
Series Network: R-L (SRL) (part)

Series Network: R-C (SRC) (part)
Series Network: R-L-C (SRX) (part)
Frequency-Independent Impedance (IMP) (part)

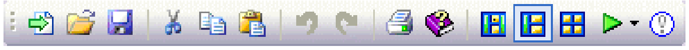
Crystal (Y)

Crystal (Y) (part)

Main Toolbar

Use this toolbar for global functions, like File Save, Print, and Undo.

To open: Click View on the menu and select Main from the Tools menu.

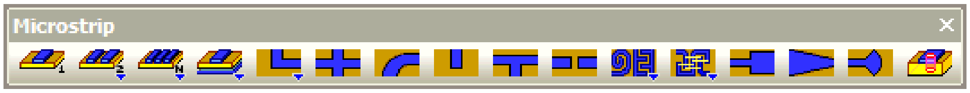


1. **Start Page** – Create a new workspace.
2. **Open** – Open an existing document.
3. **Save** – Save the active document.
4. **Cut** – Cut the selection to the clipboard.
5. **Copy** – Copy the selection to the clipboard.
6. **Paste** – Paste the contents of the clipboard.
7. **Undo** – Undo the last action. Available only for schematics and layouts.
8. **Redo** – Redo the previously undone action. Available only for schematics and layouts.
9. **Print** – Print the active window.
10. **Help** – Open the Help file.
11. **Docker View Menu** – Drop down menu to allow dockers to be toggled hidden or shown.
12. **Hide/Show Dockers** – Hide or show the Tree and Tune windows. (Hide them for more work area).
13. **Fit Windows To Frame** – Resize all of the object windows to fit into the frame.
14. **Run Analysis** – Run one or more analyses (calculate simulations).
    - When the active document window is a design/schematic and there is only one analysis associated with it, the analysis will be run.
    - If there are several associated analyses then a list containing all the associated analyses (and evaluations) will be displayed, so that the appropriate one may be selected.
    - If no design/schematic is active, or the **drop-down arrow to the right** of the button is clicked, a list containing all the analyses (and evaluations) of the workspace will be displayed, along with options to run all the out-of-date analyses or every analysis in the workspace.
15. **Stop Analyses** – The ■ button is shown instead of the Run Analysis button when any analysis or Evaluation is currently running. Click the button to stop the running Analyses / Evaluations.
    - The drop-down on the right side of the button displays options to Show or Hide the Status Window and to Stop Running Analyses/Evaluations.
16. **Errors Window** – Open the Errors window.

Microstrip Toolbar

Use this toolbar to place microstrip parts.

To open: Click the Microstrip button on the Schematic Toolbar.



MLI: Microstrip Line (Shift+M)

MLI: Microstrip Line (Shift+M) (part)

MCP, MCN4A: 2 Microstrip Coupled Lines

2 Symmetrical Coupled Microstrip Lines (MCP) (part)

2 Asymmetrical Coupled Microstrip Lines (MCN4A) (part)

MCN: Multiple Coupled Microstrip Lines

3 Coupled Microstrip Lines (MCN6) (part)

4 Coupled Microstrip Lines (MCN8) (part)

5 Coupled Microstrip Lines (MCN10) (part)

6 Coupled Microstrip Lines (MCN12) (part)

7 Coupled Microstrip Lines (MCN14) (part)

8 Coupled Microstrip Lines (MCN16)

9 Coupled Microstrip Lines (MCN18) (part)

10 Coupled Microstrip Lines (MCN20) (part)

More`MCN`models... (part)

Inverted/Suspended Microstrip

Suspended Microstrip (MSUS) (part)

Inverted Microstrip (MINV) (part)

MBN, MBN3: Microstrip Bends

Bend: Square or Chamfered (MBN) (part)

Bend: Optimal Miter (MBN3) (part)

Bend: Arbitrary Angle (MBNA) (part)

MCR: Microstrip Cross (Shift+X)

MCR: Microstrip Cross (Shift+X) (part)

MCURVE: Curved Microstrip Line

MCURVE: Curved Microstrip Line (part)

MEN: Microstrip End Effect (Shift+E)

MEN: Microstrip End Effect (Shift+E) (part)

MTE: Microstrip Tee (Shift+T)

MTE: Microstrip Tee (Shift+T) (part)

MGA: Microstrip Gap (Shift+G)

MGA: Microstrip Gap (Shift+G) (part)

Microstrip Capacitor and Inductors

Interdigital Capacitor (MIDCAP)\tShift+C (part)
Spiral Inductor (MSPIND)\tShift+L (part)
Rectangular Inductor (MRIND) (part)

Lange Coupler

Lange Coupler, 4 lines (MLANG)
Lange Coupler, 6 lines (MLANG6) (part)
Lange Coupler, 8 lines (MLANG8) (part)

MST: Microstrip Step (Shift+S)

MST: Microstrip Step (Shift+S) (part)

MTAPER: Microstrip Tapered Line

MTAPER: Microstrip Tapered Line (part)

MRS: Microstrip Radial Stub (Shift+R)

MRS: Microstrip Radial Stub (Shift+R) (part)

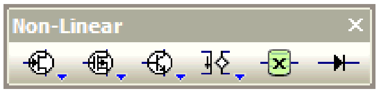
MVH: Microstrip Via Hole (Shift+V)

MVH: Microstrip Via Hole (Shift+V) (part)

Nonlinear Toolbar

Use this toolbar to place non-linear parts.

To open: Click the Nonlinear button on the Schematic Toolbar.



Nonlinear FETs

Angelov N-Channel (sim)
Curtice2 Quadratic N-Channel (part)
Curtice2a Advanced Quadratic N-Channel (sim)
Curtice3 Cubic N-Channel (part)
Parker-Skellern N-Channel (sim)
Statz N-Channel (part)
TOM N-Channel (part)
TOM2 N-Channel (part)
TOM3 N-Channel (sim)
Angelov P-Channel (sim)
Curtice2 Quadratic P-Channel (part)
Curtice2a Advanced Quadratic P-Channel (sim)
Curtice3 Cubic P-Channel (part)

Statz P-Channel (part)
TOM P-Channel (part)
TOM2 P-Channel (part)
TOM3 P-Channel (sim)
JFET N-Channel (part)
JFET P-Channel (part)
JFET2 N-Channel (part)
JFET2 P-Channel (part)

Nonlinear MOSFETs

MOS1 N-Channel (part)
MOS1 P-Channel (part)
MOS2 N-Channel (part)
MOS2 P-Channel (part)
MOS3 N-Channel (part)
MOS3 P-Channel (part)
MOS9 N-Channel (sim)
MOS9 P-Channel (sim)
BSIM1 N-Channel (part)
BSIM1 P-Channel (part)
BSIM2 N-Channel (part)
BSIM2 P-Channel (part)
BSIM3 N-Channel (part)
BSIM3 P-Channel (part)
BSIM4 N-Channel (sim)
BSIM4 P-Channel (sim)
EKV N-Channel (sim)
EKV P-Channel (sim)
LDMOS MET Model N-Channel (part)
LDMOS MET Model P-Channel (part)
TFT_ASIA2 N-Channel (sim)
TFT_ASIA2 P-Channel (sim)
TFT N-Channel (sim)
TFT P-Channel (sim)

Nonlinear BJTs

NPN Bipolar Gummel-Poon (BIPNPN) (part)
NPN Bipolar Gummel-Poon with Substrate Node (BIPNPN4) (part)
NPN Bipolar VBIC (VBICNPN) (part)
NPN Bipolar VBIC with Substrate Node (VBICNPN4) (part)
NPN Bipolar VBIC with Substrate and Temp. Nodes (VBICNPN5) (part)
NPN Bipolar MEXTRAM (MEXTRAM_NPN) (sim)
NPN Bipolar MEXTRAM with Substrate Node (MEXTRAM_NPN4) (sim)
NPN Bipolar MEXTRAM with Substrate and Temp. Nodes (MEXTRAM_NPN5) (sim)
PNP Bipolar Gummel-Poon (BIPPNP) (part)
PNP Bipolar Gummel-Poon with Substrate Node (BIPPNP4) (part)
PNP Bipolar VBIC (VBICNPN) (part)
PNP Bipolar VBIC with Substrate node (VBICNPN4) (part)
PNP Bipolar VBIC with Substrate and Temp. Nodes (VBICPNP5) (part)
PNP Bipolar MEXTRAM (MEXTRAM_PNP) (sim)
PNP Bipolar MEXTRAM with Substrate Node (MEXTRAM_PNP4) (sim)
PNP Bipolar MEXTRAM with Substrate and Temp. Nodes (MEXTRAM_PNP5) (sim)

Nonlinear Power Sources

Current-Controlled Current Source (part)
Current-Controlled Voltage Source (part)
Voltage-Controlled Current Source (part)
Voltage-Controlled Voltage Source (part)

X-Parameters

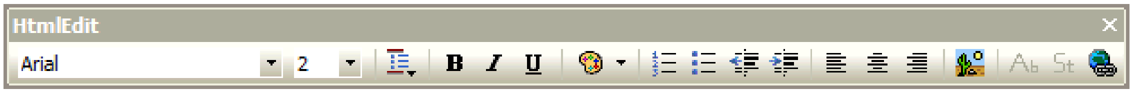
X-Parameters Part (part)

Nonlinear Diode (D)

Standard Nonlinear Diode (part)

Notes Toolbar

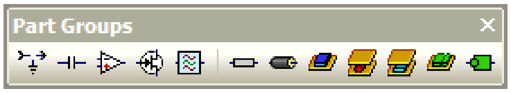
Use this toolbar to edit/modify the Note its text settings. The Notes toolbar automatically displays when you have an active Note.



1. **Font** - select a font for the selection or for typing
2. **Size** - select a font size in html units (3 = average) for the selection or typing
3. **Style** - click the pulldown to pick from standard html styles
4. **Bold** - embolden selected characters
5. **Italic** - italicize selected characters
6. **Underline** - underline selected characters
7. **Color** - select font color
8. **Number** - number the selected paragraphs
9. **Bullet** - bullet the selected paragraphs
10. **Exdent** - exdent a paragraph (reduce indent)
11. **Indent** - indent a paragraph
12. **Left Justify** - left justify the paragraph
13. **Center Justify** - center justify the paragraph
14. **Right Justify** - right justify the paragraph
15. **Image** - Insert a picture into the notes. This picture is specified by a URL.
16. **Absolute** - position part as absolute
17. **Static** - position part as static
18. **Hyperlink** - add a hyperlink (this is currently disabled)

Part Groups Toolbar

Use this toolbar to bring up sub-toolbars with palettes of parts on them. This toolbar may automatically display when you have a schematic active or use the Schematic Toolbar to show it.

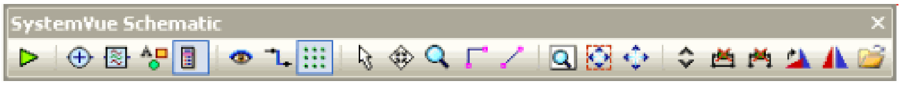


1. ***Basic* (users)** - Bring up the basic toolbar, which contains ports and sources.
2. ***Lumped* (users)** - Open or close the Lumped Toolbar, which contains lumped parts.
3. ***Linear* (users)** - Open or close the Linear Toolbar, which contains component models (such as diodes and op-amps).

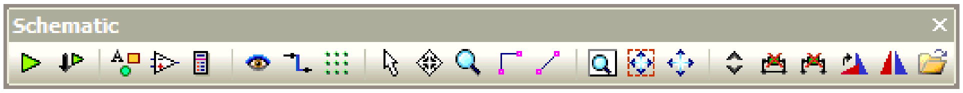
4. ***Non-Linear* (users)** - Open or close the Nonlinear Toolbar, which contains nonlinear models (such as transistors and diodes).
5. ***System* (users)** - Open or close the System Toolbar, which contains RF system models.
6. ***T-Line* (users)** - Open or close the T-Line Toolbar, which contains ideal transmission line models.
7. ***Coax* (users)** - Open or close the Coax Toolbar, which contains physical coaxial line models.
8. ***Microstrip* (users)** - Open or close the Microstrip Toolbar, which contains physical microstrip line and discontinuity models.
9. ***Slabline* (users)** - Open or close the Slabline Toolbar, which contains physical slabline models.
10. ***Stripline* (users)** - Open or close the Stripline Toolbar, which contains physical stripline line and discontinuity models.
11. ***Co-Planar* (users)** - Open or close the Coplanar Toolbar, which contains physical coplanar line and discontinuity models.
12. ***Wave* (users)** - Open or close the Waveguide Toolbar, which contains rectangular waveguide and waveguide-to-TEM adapter models.

Schematic Toolbar

Use this toolbar to change a schematic or to bring up another toolbar with commonly used parts.



OR

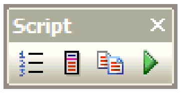


1. **Run** - Runs the analyses.
2. **Part Group** - Show/Hide the part group toolbar.
3. **Annotation** - Show/Hide the Annotation toolbar.
4. **Part Selector** - Show/Hide the Part Selector.
5. **Eye** - Use this pull down menu to turn on/off text displays such as Part Parameters, Net Names, etc. on the schematic.
6. **Keep Connect** - Enable/disable automatic line connections when dragging parts.
7. **Grid Snap** - Enable/disable the grid snap.
8. **Select** - Use the select tool to select parts or annotations.
9. **Pan** - Use the pan tool to pan the schematic around. Press the tool and drag the schematic.
10. **Zoom** - Use the zoom tool to zoom into a rectangular region of the schematic.
11. **Line** - Use the line tool to draw horizontal, vertical or right angled line connections.
12. **Angled Line** - Use the angled line tool to draw line connections of any orientation.
13. **Zoom to Page** - Zoom to fit the page.
14. **Zoom to Fit Selection** - Zoom to fit the currently selected parts/objects on a schematic.
15. **Zoom to Fit All** - Zoom to fit all object.
16. **Tune** - Make the selected parts tunable or fixed.
17. **Disable to short** - Disable/enable the selected parts and simulate them as short circuit.
18. **Disable to open** - Disable/enable the selected parts and simulate them as an open circuit.

19. **Rotate** - Rotate the selected parts by 90 degrees.
20. **Mirror** - Mirror the selected parts.
21. **Open Model or Symbol** - Open part models/symbols. For a single part, this button can open its model/symbol library.

Script Toolbar

Use this toolbar to interact with the active Script and adjust its settings.

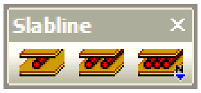


1. **Line Numbers** – Hide/Show line numbers on the display.
2. **Script Processor** – Hide/Show the script processor window.
3. **Copy** – copy the script to the script processor.
4. **Run** – copy the script to the script processor and run it.

Slabline Toolbar

Use this toolbar to place Slabline components.

To open: Click the Slabline button on the Schematic Toolbar.



RLI: Single Slabline

RLI: Single Slabline (part)

RCP: 2 Coupled Slablines

RCP: 2 Coupled Slablines (part)

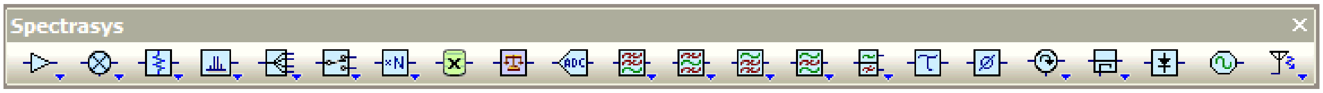
RCN: Multiple Coupled Slablines

3 Coupled Slablines (RCN6) (part)
4 Coupled Slablines (RCN8) (part)
5 Coupled Slablines (RCN10) (part)
6 Coupled Slablines (RCN12) (part)
7 Coupled Slablines (RCN14) (part)
8 Coupled Slablines (RCN16) (part)
9 Coupled Slablines (RCN18) (part)
10 Coupled Slablines (RCN20) (part)
More`RCN`models...

Spectrasys Toolbar

Use this toolbar to place system parts.

To open: Click the System button on the Schematic Toolbar.



RF Amplifiers

RF Amplifier - 2nd 3rd Order (part)
RF Amplifier - High Order (part)
Variable Gain Amplifier (part)

RF Mixers

Basic Mixer (part)
Double Balanced Mixer (part)
Table Mixer (part)

Attenuators

Attenuator (part)
DC Controlled Attenuator (part)
Frequency Dependent Attenuator (part)
Variable Attenuator (part)

Sources

Multi Source (part)
CW Source (part)
CW Source with PhaseNoise (part)
Wideband Source (part)
Multicarrier Source (part)
Intermod Source (part)
Receiver Intermod Source (part)
Continuous Frequency Source (part)
Noise Source (part)

Splitters/Combiners

2 Way 0 Degree Splitter/Combiner (part)
2 Way 90 Degree Splitter / Combiner (part)
2 Way 180 Degree Splitter / Combiner (part)
3 Way 0 Degree Splitter / Combiner (part)
4 Way 0 Degree Splitter / Combiner (part)
5 Way 0 Degree Splitter / Combiner (part)
6 Way 0 Degree Splitter / Combiner (part)
8 Way 0 Degree Splitter / Combiner (part)
9 Way 0 Degree Splitter / Combiner (part)
10 Way 0 Degree Splitter / Combiner (part)
12 Way 0 Degree Splitter / Combiner (part)
16 Way 0 Degree Splitter / Combiner (part)
24 Way 0 Degree Splitter / Combiner (part)
48 Way 0 Degree Splitter / Combiner (part)

RF Switches

Switch - SPST (part)
Switch - SPDT (part)
Switch - SP3T (part)
Switch - SP4T (part)

Switch - SP5T (part)
Switch - SP6T (part)
Switch - SP7T (part)
Switch - SP8T (part)
Switch - SP9T (part)
Switch - SP10T (part)
Switch - SP11T (part)
Switch - SP12T (part)
Switch - SP13T (part)
Switch - SP14T (part)
Switch - SP15T (part)
Switch - SP16T (part)
Switch - SP17T (part)
Switch - SP18T (part)
Switch - SP19T (part)
Switch - SP20T (part)

Frequency Multiplier

Frequency Multiplier (part)
Frequency Divider (part)
Digital Frequency Divider (part)

Analog / Digital Converters

Analog / Digital Converters (part)

Lowpass Filters

Lowpass Butterworth Filter (part)
Lowpass Bessel Filter (part)
Lowpass Chebyshev Filter (part)
Lowpass Elliptic Filter (part)
Lowpass Pole Zero Filter (part)

Bandpass Filters

Bandpass Butterworth Filter (part)
Bandpass Bessel Filter (part)
Bandpass Chebyshev Filter (part)
Bandpass Elliptic Filter (part)
Bandpass Pole Zero Filter (part)

Highpass Filters

Highpass Butterworth Filter (part)
Highpass Bessel Filter (part)
Highpass Chebyshev Filter (part)
Highpass Elliptic Filter (part)
Highpass Pole Zero Filter (part)

Bandstop Filters

Bandstop Butterworth Filter (part)

Bandstop Bessel Filter (part)
Bandstop Chebyshev Filter (part)
Bandstop Elliptic Filter (part)
Bandstop Pole Zero Filter (part)

Duplexers

Chebyshev Duplexer (part)
Elliptic Duplexer (part)

Time Delay

Time Delay (part)

Phase Shifter

Phase Shifter (part)

RF Circulator / Isolator

Circulator (part)
Isolator (part)

Couplers

Single Directional Coupler (part)
Dual Directional Coupler (part)
90 Degree Hybrid Coupler (part)
180 Degree Hybrid Coupler (part)

Log Detector

Log Detector (part)

Oscillator

Oscillator (part)

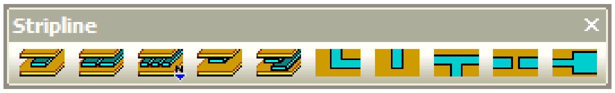
Antennas

Coupled Antenna (part)
Antenna Path (part)

Stripline Toolbar

Use this toolbar to place Stripline components.

To open: Click the Stripline button on the Schematic Toolbar.



SLI: Single Stripline

SLI: Single Stripline (part)

SCP: 2 Coupled Striplines

SCP: 2 Coupled Striplines (part)

SCN: Multiple Coupled Striplines

3 Coupled Striplines (SCN6) (part)
4 Coupled Striplines (SCN8) (part)
5 Coupled Striplines (SCN10) (part)
6 Coupled Striplines (SCN12) (part)
7 Coupled Striplines (SCN14) (part)
8 Coupled Striplines (SCN16) (part)
9 Coupled Striplines (SCN18) (part)
10 Coupled Striplines (SCN20) (part)
More`SCN`models...

SLIO: Offset Stripline

SLIO: Offset Stripline (part)

SBCP: Offset Coupled Striplines

SBCP: Offset Coupled Striplines (part)

SBN: Stripline Bend

SBN: Stripline Bend (part)

SEN: Stripline End Effect

SEN: Stripline End Effect (part)

STE: Stripline Tee

STE: Stripline Tee (part)

SGA: Stripline Gap

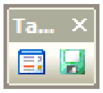
SGA: Stripline Gap (part)

SSP: Stripline Step

SSP: Stripline Step (part)

Table Toolbar

Use this toolbar to interact with the active Table and adjust its settings.



1. **Properties** - Bring up the properties dialog.

2. **Save** - export/save the table to a file.

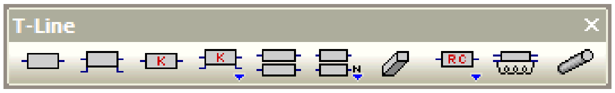
See Also

- *Tables* (users)

T-Line Toolbar

Use this toolbar to place transmission line components.

To open: Click the T-Line button on the Schematic Toolbar.



TLE: Transmission Line

TLE: Transmission Line (part)

TLE4: 4 - Terminal Transmission Line

TLE4: 4 - Terminal Transmission Line (part)

TLP: Physical Transmission Line

TLP: Physical Transmission Line (part)

TLP4: 4 - Terminal Physical Transmission Line

Physical transmission line (part)
Open 4 Terminal Stub (part)
Shorted 4 Terminal Stub (part)

CPL: 2 Coupled Transmission Lines

CPL: 2 Coupled Transmission Lines (part)

CPNx: Multiple Coupled Line Models

3 Coupled T-Lines (CPN6) (part)
4 Coupled T-Lines (CPN8) (part)
5 Coupled T-Lines (CPN10) (part)
6 Coupled T-Lines (CPN12) (part)
7 Coupled T-Lines (CPN14) (part)
8 Coupled T-Lines (CPN16) (part)
9 Coupled T-Lines (CPN18) (part)
10 Coupled T-Lines (CPN20) (part)
More`CPN`models...

RIBBON: Ribbon Wire

RIBBON: Ribbon Wire (part)

RLGC: Ideal Distributed Line Models

Distributed RC T-Line (RCLIN) (part)
Distortionless TEM T-Line (TLRLDC) (part)
Uniform TEM T-Line (TLRLGC) (part)
Exponential TEM T-Line (TLX) (part)

TRFRUTH: Ruthroff Transformer

TRFRUTH: Ruthroff Transformer (part)

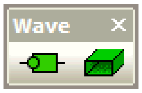
WIRE: Round Wire

WIRE: Round Wire (part)

Wave Toolbar

Use this toolbar to place waveguide components.

To open: Click the Wave button on the Schematic Toolbar.



WAD: Waveguide Adapter

WAD: Waveguide Adapter (part)

WLI: Rectangular Waveguide

WLI: Rectangular Waveguide (part)