

## Chapter 1

# Welcome to PSCAD

Whether you are a seasoned veteran upgrading to PSCAD X4, or a novice just starting out, you have joined a community comprised of over 30,000 users at over 1000 sites in nearly 80 countries – welcome to the family!

## What is PSCAD?

PSCAD (Power Systems Computer Aided Design) is a powerful and flexible graphical user interface to the world-renowned, EMTDC electromagnetic transient simulation engine. PSCAD enables the user to schematically construct a circuit, run a simulation, analyse the results, and manage the data in a completely integrated, graphical environment. Online plotting functions, controls and meters are also included, enabling the user to alter system parameters *during* a simulation run, and thereby view the effects while the simulation is in progress.

PSCAD comes complete with a library of pre-programmed and tested simulation models, ranging from simple passive elements and control functions, to more complex models, such as electric machines, full-on FACTS devices, transmission lines and cables. If a required model does not exist, PSCAD provides avenues for building custom models. For example, custom models may be constructed by piecing together existing models to form a module, or by constructing rudimentary models from scratch in a flexible design environment.

The following are some common models found in the PSCAD master library:

- Resistors, inductors, capacitors
- Mutually coupled windings, such as transformers
- Frequency dependent transmission lines and cables (including the most accurate time domain line model in the world!)
- Current and voltage sources
- Switches and breakers
- Protection and relaying
- Diodes, thyristors and GTOs
- Analog and digital control functions
- AC and DC machines, exciters, governors, stabilizers and inertial models
- Meters and measuring functions
- Generic DC and AC controls
- HVDC, SVC, and other FACTS controllers
- Wind source, turbines and governors

PSCAD, and its simulation engine EMTDC, have enjoyed close to 40 years of development, inspired by ideas and suggestions by its ever strengthening, worldwide user base. This development philosophy has helped to establish PSCAD as one of the most powerful and intuitive CAD software packages available.

# A Quiet Revolution in Simulation

PSCAD was first conceptualized in 1988, and began its development as a graphical interface for the EMTDC electromagnetic transient simulation program. In its pre-commercial form, PSCAD was largely experimental; nevertheless, it represented a giant leap forward in productivity, since EMTDC users could design their systems schematically, rather than entering data through text listings. The graphical aspects of PSCAD enhanced the overall perceptual comprehension of the simulated system, dramatically accelerating circuit assembly and minimizing error.

Prior to its release, PSCAD went through extensive testing in North America, Japan, Australia and Europe, and upon completion was publicized in the fall of 1992 under the trademark PSCAD/EMTDC Version 3. The released UNIX-based version of PSCAD, which accompanied EMTDC, was eventually referred to as PSCAD V2, and consisted of a suite of associated software tools that performed circuit drafting, runtime plotting/control and off-line plotting. PSCAD V2 enjoyed tremendous success leading up to the new millennium. During this same period, desktop personal computers running Microsoft Windows became immensely popular, and the rapidly expanding PSCAD user base demanded a version that supported this platform. Development on a Windows based version of PSCAD commenced immediately.

## Migration to the Windows Environment

When PSCAD V3 for Windows finally arrived in 1999, it sought to push the envelope by introducing an environment where system schematics could be built in a modular form. Systems could thus be constructed using interconnected page modules (or sub-pages), which were compiled individually and possessed their own private data space. In addition, PSCAD V3 merged both the drafting and runtime systems of its predecessor, resulting in a comprehensive environment harbouring both design and simulation analysis.

In 2001, development of the next major release commenced. One of the primary goals for PSCAD V4 was to enhance the robustness of the software, mainly by migrating many of the custom toolsets over to a more standardized design. This included the incorporation of *Microsoft Foundation Class (MFC)* architecture, as well as a completely new library of online control and plotting tools. The V3 Component Workshop utility was integrated directly as part of the drafting editor, in order to achieve a fully unified design environment; all aspects of component definition and circuit design could be performed using a single view.

PSCAD V4 was released in 2002, to much success. It included many other new features that have since become indispensable to project design including, single-line representations, xy-plotting, wire mode, undo/redo, drag and drop, docked windows and enhanced navigational features.

## A Paradigm Shift

As with many software products, the need to continually improve PSCAD creates pressure to add newer and better features that make the users' experience an enjoyable one. Continuous development over time however, can lead to a situation where the application code body becomes a complicated mess of patch work and shoe-horned feature implementations. Further development of the product becomes more and more difficult as new mechanisms are put in place that perhaps do not fit well in the overall architecture. A smart development team will recognize when its software reaches this juncture and take effective action. In 2006, this time came when the PSCAD user base began to demand a simulation environment where they could perform studies of different types, such as load flow, in addition to EMTDC-based electromagnetic transients studies.

Development of a completely new design for PSCAD was embarked upon, specifically to make this multiple study environment a possibility. Following many hours of research and planning, a new architecture was chosen, based on a single database model, it is a design that places the entire application focus on a data core. The result is an environment where different solver engines (ex. EMTDC, Load Flow, etc.) can dynamically share information between themselves and the simulation environment itself. This new implementation is referred to as the *neXus engine*, and represents a new generation of PSCAD products.

PSCAD X4 is the first product release to include the neXus engine. Although still exclusively an electromagnetic transients study environment, its primary new feature will act as a foundation for a multiple simulation environment; page modules that may be multiple-instanced based on the same definition. It also includes many other new features, such as parallel simulation runs, module black boxing, transmission line mutual coupling, and many others. In 2011, the *Microsoft Foundation Class (MFC)* architecture was upgraded to the latest version. This resulted in an application face lift, which includes a modern ribbon control bar and tabbed window interface.

## Who Uses PSCAD, and for What?

The PSCAD users' spectrum includes engineers and scientists from utilities, manufacturers, consultants, as well as research, military and academic institutions. It is used in planning, operation, design, commissioning, preparing of tender specifications, teaching and research. The following are examples of the studies routinely conducted using PSCAD:

- Contingency studies of AC networks consisting of rotating machines, exciters, governors, turbines, transformers, transmission lines, cables, and loads
- Relay coordination
- Transformer saturation effects
- Insulation coordination of transformers, breakers and arrestors
- Impulse testing of transformers
- Sub-synchronous resonance (SSR) studies of networks with machines, transmission lines and HVDC systems
- Evaluation of filter design and harmonic analysis
- Control system design and coordination of FACTS and HVDC; including STATCOM, VSC, and cycloconverters
- Optimal design of controller parameters
- Investigation of new circuit and control concepts
- Lightning strikes, faults or breaker operations
- Steep front and fast front studies
- Electric naval vessel design
- Investigation of the pulsing effects of diesel engines and wind turbines on electric networks

## What's New in PSCAD X4?

PSCAD X4 is classified as a minor upgrade. The number of enhancements and features it provides however are far from minor. The following is a general overview of what is new in PSCAD X4.

### Important Points of Note

PSCAD X4 represents a complete refurbishment of the internal architecture of the software. Although it appears quite similar to the previous version on the surface, under the hood it is vastly different. Due to these changes, there are a few important points of note.

- **New File Formats:** PSCAD X4 project file extensions have been changed to reflect the switch-over to XML-based file storage. The extensions are now *\*.pslx* and *\*.pscx* for library and case projects respectively. Component definition file extensions have been changed from *\*.cmp* to *\*.psdx*.
- **Upwards Compatibility:** PSCAD X4 supports the import of *\*.psc* and *\*.psl* file formats that have been generated by PSCAD v4.2 only. Older component definition files with extension *\*.cmp* may be imported as well.
- **Downwards Compatibility:** The PSCAD X4 release is not backwards compatible. That is, X4 format project files (*\*.pscx* and *\*.pslx*) cannot be converted back to *\*.psc* or *\*.psl* format.
- **New Look and Feel:** The application window framework has been upgraded from MFC 6 (c. 1998) to MFC 10. This has resulted in a brand new look and feel for the software:

- **Ribbon Control Bar:** A modern ribbon control bar has been added that provides easier accessibility to most features and components. Included with the ribbon is an inherent quick access bar, which is fully customizable by the user for placement of favoured and well used button actions. The ribbon control bar is featured prominently across the top of the application environment.
- **Tabbed Document Interface (TDI):** A modern working environment including customizable docked windows and window pinning and hiding has been added. The new MFC framework also incorporates a new tabbed document interface, which enhances convenience in inter-project navigation.
- **64-bit PSCAD Product:** A 64-bit product is now here! A separate software product, the PSCAD 64-bit application directly addresses the 'Out of memory' issue that some power users have experienced when attempting to run very large simulations: If a simulation exceeds the allocated process memory of 2 GB (imposed by the Windows 32-bit operating system) then the simulation will crash, resulting in lost time. The 64-bit PSCAD will open up an enormous memory space (8 TB or 8,000 GB), but the speed of execution is not affected.
- **New Component Wizard:** A revamped component wizard has been included in this release. Functioning internally in a similar manner to the older version, the new wizard possesses a much different interface on the surface. New components (both native and module), transmission lines and cables may be created from this utility.
- **New Search (Query) Utility:** A revamped search utility has been included in this release. Functioning internally in a similar manner to the older version, the new utility possesses a different interface on the surface.
- **Temporary Directories:** The old temporary directory (\*.emt) naming convention is obsolete. A unique temporary directory is now created depending on the Fortran compiler used to build the project. For example, if your project is called *vdiv\_1.pscx*, then the temporary directory will be named *vdiv\_1.gf42* if you have selected the GFortran compiler.

The temporary directory extensions are as follows:

- **GFortran:** \*.gf42
- **Intel Visual Fortran 9 to 11:** \*.if9
- **Intel Visual Fortran Composer XE 2011:** \*.if12
- **Active Project Concept is Obsolete:** In previous versions of the software it was necessary to set an 'active' project for compiling and running the simulations. Due to changes in the way the software navigates and displays status information, the concept of an 'active' project is no longer necessary. It is now possible to run multiple cases in the environment simultaneously. All ribbon control buttons and status bar messages are based on the project currently in focus. See Simulation Sets/Multiple EMTDC for more details.

There are other important issues to consider both before and after importing older projects into PSCAD X4. For more details on this, please see Converting PSCAD v4.1 and v4.2 Projects to X4 in Chapter 13 of this manual.

## New and Enhanced Features

The following is a general overview (not exhaustive) of new and enhanced features to be aware of:

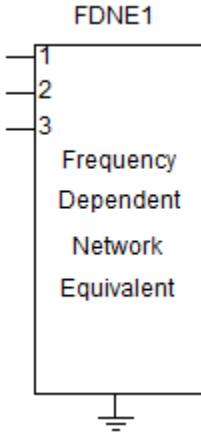
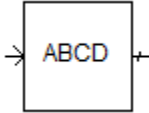

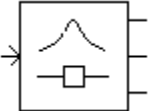
- **Multiple Instance Modules (MIM):** This is the primary new feature in this release. See Multiple Instance Modules (MIM) in Chapter 5 of this manual for more details.
- **EMTDC Runtime Configuration:** Runtime configuration is a term used to describe a collection of changes to both the EMTDC system dynamics structure and methods in the design of components, in order to ensure support for MIM.
  - **#BEGIN/#ENDBEGIN Directive Block:** This directive provides access to the BEGIN outer process in EMTDC, and is required for supporting MIM in custom components. See The BEGIN Subroutine in Chapter 2 of the EMTDC manual or #BEGIN/#ENDBEGIN in Chapter 10 of this manual for more details.

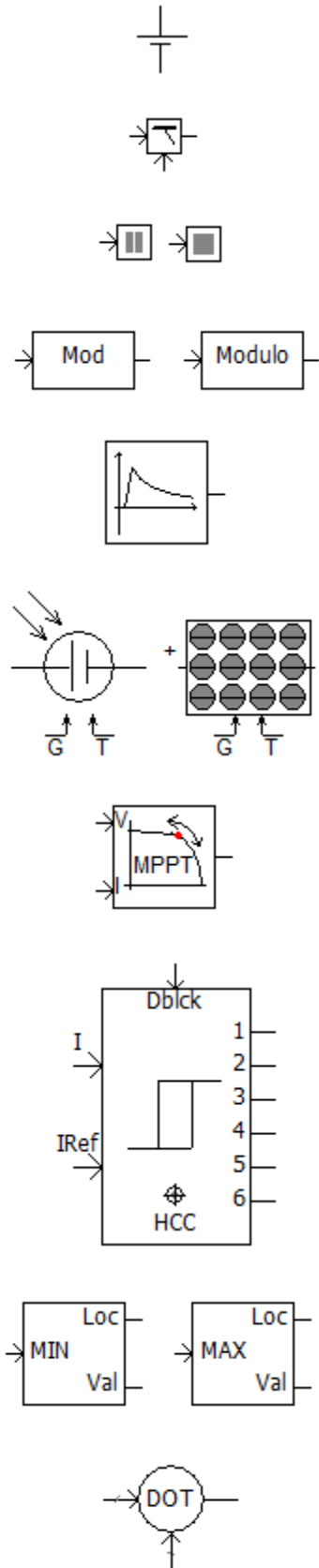
- **New #STORAGE Arrays:** New storage arrays have been added to EMTDC specifically for the transfer of data from the new BEGIN section to the DSDYN/DSOUT sections in the EMTDC system dynamics. See #STORAGE in Chapter 10 of this manual for more details.
- **High Performance Computing:** Two unique parallel computing functions were incorporated that both utilize multiple processor cores: A parallel solution of transmission lines and cables, and a ability to launch multiple EMTDC simulation runs simultaneously.
- **Electric Network Interface (ENI):** Multiple case projects representing multiple parts of a complete electric network, may be run simultaneously as a single simulation. A single electric network may be split so that each electric subsystem is represented by a separate case project, and thereby run using a separate EMTDC process. Each EMTDC process is linked together via an *Electric Network Interface (ENI)* to form a cohesive simulation that is run from within a single workspace.
- **Certificate Licensing:** All PSCAD products now support cloud-based, Certificate Licensing, in addition to the intranet-based legacy licensing, which requires a License Manager on the client's premises. Certificate licensing provides a time-limited, license certificate from MyCentre, an internet-based user portal. This allows users to work off-site and off-grid without the need to connect to their License Manager, license files, nor a hardware lock.
- **Multiple Workspaces:** PSCAD now supports multiple workspaces: What this means from the user's perspective is that entire workspaces may be loaded, saved and unloaded without having to close the application. A single workspace may house multiple projects, including both libraries and cases, as well as possessing its own unique setting options.
- **Workspace-Level Control:** Workspace-level, multi-project, multiple-run control support has been added to the application. Communicating inter-project signal values via Radio Link components to and from a common value table in the workspace, a master project can control a number of slave projects to simultaneously perform necessary calculations and report them back to the master. See Workspace-Level Control for more details.
- **Blackboxing Modules:** With a simple click, this feature will convert any page module containing purely control-based systems into an equivalent, non-module component, complete with generated source files and/or compiled binary files. Blackboxing allows users to design their systems graphically, and then quickly black box the system, thereby protecting their intellectual property when distributing their models to clients.
- **Simulation Sets/Multiple EMTDC:** It is now possible to simultaneously launch and run multiple, simultaneous EMTDC simulations. Both sequential and parallel simulation runs are possible via the defining of what are referred to as 'simulation sets' in the workspace.
- **Enhanced Searching:** The searching facilities have been enhanced. The background search engine is now based on XPath, a query language for selecting nodes in an XML document. See Searching in Chapter 11 of this manual for details.
- **Fortran Compiler Support:** A new free Fortran 95 compiler called GFortran now accompanies PSCAD X4. Note that the former free compiler GNU Fortran 77 is no longer supported. The Compaq Fortran Compiler is also no longer supported as of v4.6.0. Since 2001, this compiler product has been developed by the Intel corporation, who have released many updated versions since. The Compaq version of the compiler (v6) is no longer sold or maintained. See The Getting Started Brochure and Converting PSCAD v4.1 and v4.2 Projects to X4 in Chapter 13 of this manual for more details.
- **Mutual Coupling:** This feature enables users to mutually couple individual line or cable segments with identical lengths. Multiple segments can be merged into a single Right-Of-Way (ROW) without affecting the individuality of the each segment. See Mutual Coupling in Chapter 8 of this manual for more details.
- **Enhanced User Petition Request:** The former Support Petition Request dialog has been improved with additional features. New enhancements include the ability to manually attach files, as well as automatic collection and attachment of key files necessary for debugging of customer installation and licensing problems.
- **Oscilloscope:** A new meter utility has been added as yet another avenue for viewing data online. See Oscilloscopes in Chapter 6 for more details.
- **Comparator Tool:** The schematic comparator tool allows for quick and convenient visual differentiation between module component definitions.

- **Parameter Grid Pane:** The component parameter grid pane provides a convenient means to display the parameters for all instances of a given component or module definition. More importantly, it enables the ability to modify multiple parameter values in multiple component instances simultaneously.
- **Bird's Eye View Pane:** The Bird's Eye View navigation pane provides an overview of the entire Schematic or Graphic canvas and indicates what is currently in view with a blue box.
- **Layers Pane:** The layers pane is the interface to the schematic canvas drawing layers feature. Drawing layers provide the ability to efficiently enable or disable components on the canvas, or to toggle the visibility of any objects that appear on the canvas.
- **Display Voltage on Buses:** This option allows for the dynamic display of RMS voltage directly on Bus components. See Runtime in Chapter 7 of this manual for more details.
- **Saving Graphics to File:** Graphic objects used in the Graphics section of the component design environment can now be stored in and imported from files. See The Graphic Section in Chapter 9 of this manual for more details.

## New Master Library Models

The following table comprises a list of all new models added to the master library since the PSCAD v4.2.1b release in 2007.

Graphic	Name
	Frequency-Dependent Network Equivalent (FDNE)
	Frequency Dependent Transfer Function (FDTF)
	Rank Number
	Statistical Breaker



Battery

Dynamic Data Tap

Programmable Pause

Programmable Stop

Mod Function and Modulo Function

Surge Generator (CIGRÉ, IEC or IEEE Standard)

Photovoltaic Source

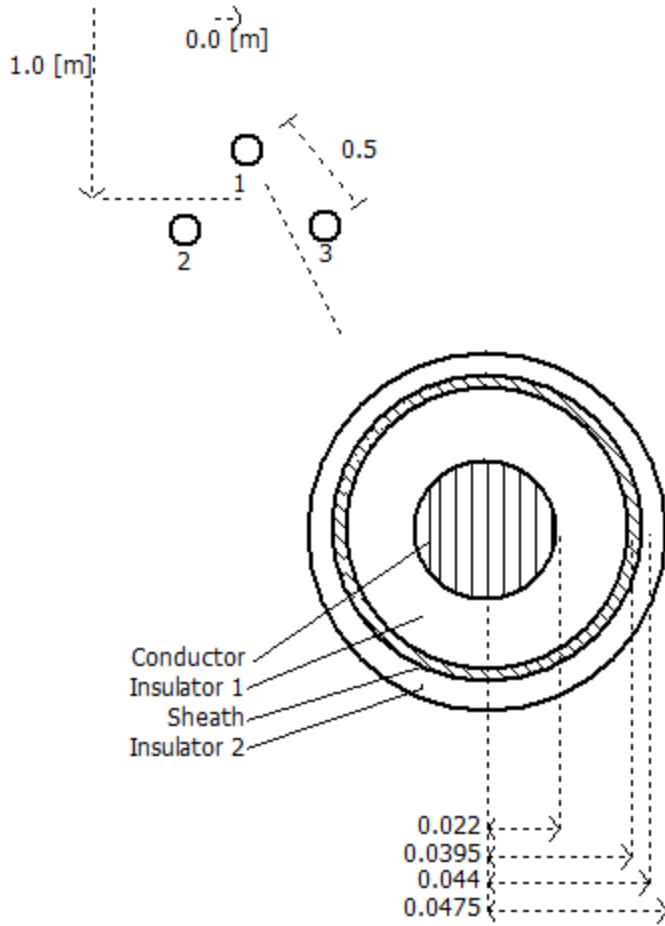
Maximum Power Point Tracker

Hysteresis Current Control PWM Generator

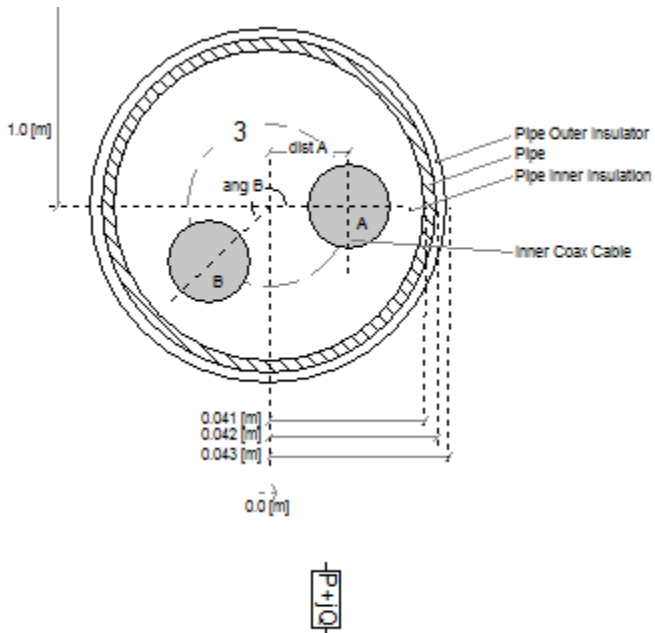
Maximum/Minimum Array Value and Location

Array Dot Product

Simplified Underground Cable

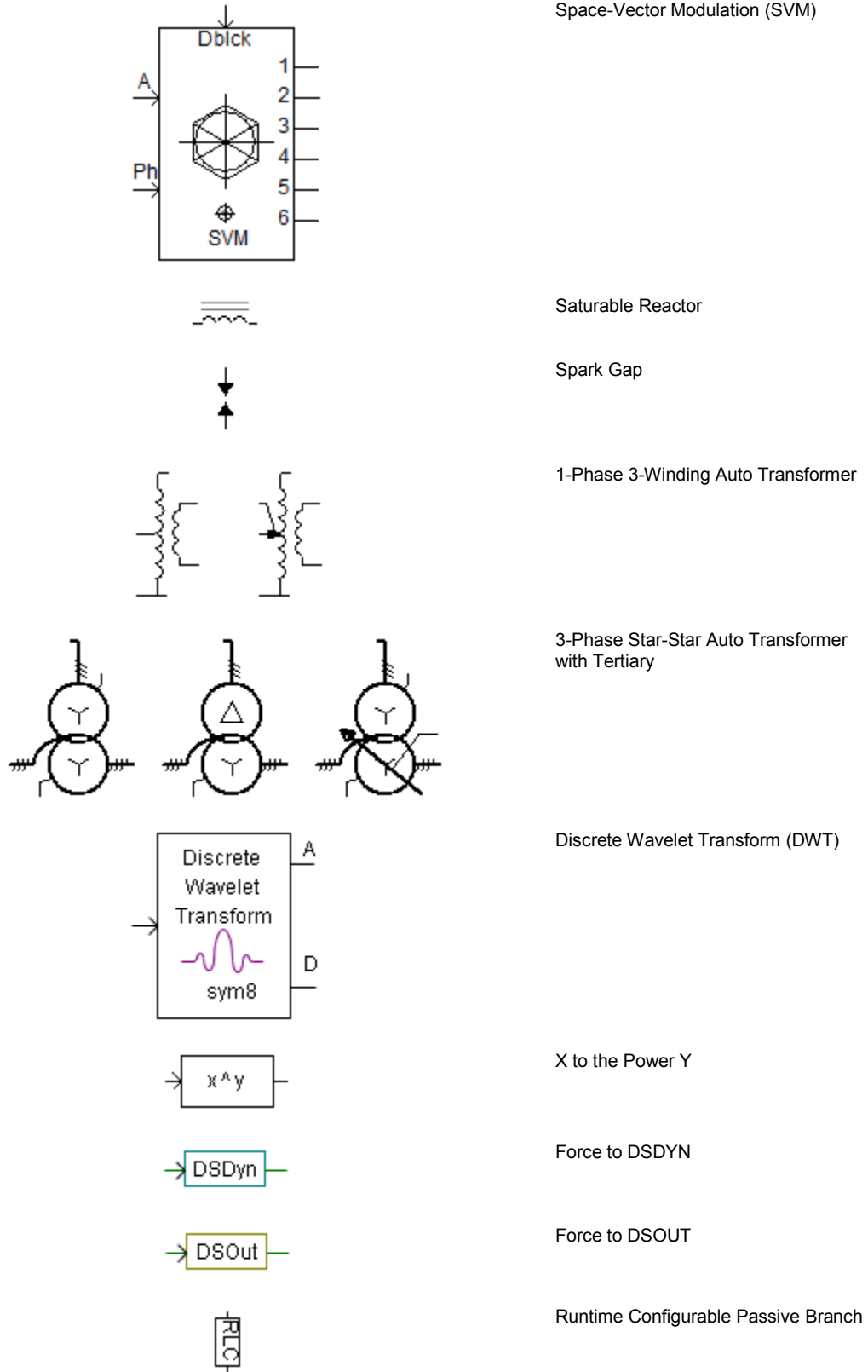


Pipe-Type Cable



1-Phase, L-L Fixed Load



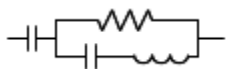




XY Table



Variable Series Impedance Branch



C-Type Filter



Multiple Run Additional Recording

## Line Constants Program (LCP)

There have been significant enhancements to the Line Constants Program (LCP) since the previous released version (August 26, 2005). The new DC Correction algorithm has made the Frequency Dependent (Phase) model even more accurate than ever.

- **Aerial Cable Support:** It is now possible to combine both underground and aerial cables in the same right-of-way. The new feature affects mainly the ground plane component, where you must specify the formulae used for aerial, underground and mutual (underground/aerial) earth return representation. You must also specify in the coax or pipe cable components, whether or not they are aerial or underground.
- **DC Correction:** Two unique DC correction algorithms have been added, which ensure perfectly accurate DC parameters for time domain simulations.
- **Passivity Checking:** The LCP now checks for passivity violations and if found, warns the user. New input parameters have been added to the frequency-dependent (phase) model for control of this feature.
- **Total Number of Conductors Increased:** The total allowable conductors per transmission line or cable have been increased from 20 to 30.
- **Unique Ground Wires in Overhead Towers:** If there are 2 ground wires in a tower, they may now be entered with unique parameters.
- **Hollow Conductor Support in Overhead Towers:** All conductors in a tower may now be selected as hollow core.
- **Bundled Sub-Conductor Limit Increased:** Conductor bundles may now include up to 15 sub-conductors.
- **Conductor/Ground Wire Permeability:** There is now an input for relative permeability of both conductors and ground wires.
- **Conductor Library Format Change:** The addition of conductor relative permeability and hollow conductor support has forced a change in the format of Conductor Library files (additional two parameters).
- **Specific Conductor Layer Elimination in Cables:** Users may now select which conductors are to be eliminated (not just the outer layer).

- **Enhanced Log File Output:** The \*.log file format for the Frequency Dependent (Phase) model has been updated to make it more readable.
- **Additional Detailed Output Files:** Added additional detailed output files for calculated versus fitted values when using the Frequency Dependent (Phase) model. Users may now compare calculated versus fitted responses for the first time for this model.
- **PI Section Auto-Creation:** PI-sections may now be created when using the Manual Entry of Y,Z component. PI-section component automatic creation was flawed when solving single-phase transmission systems. This has been fixed.
- **Overlapping Cables:** A check to ensure cable cross-sections do not overlap is now performed. This is accomplished by comparing the centre-point modulus with the sum of the radii.
- **Cable Depth and Conductor Height:** Checks are performed to ensure these parameters are entered positive.
- **Conductor Permeability:** The relative permeability input parameter value entered in the Ground Plane component was used as the relative permeability value for all ground wires and conductors in overhead towers. Conductor and ground wire permeabilities are now unique.
- **Total Number of Cables:** The LCP now supports up to 12 cables, including all concentric conductors

See Chapter 9 in the EMTDC manual and Chapter 8 of this manual, both entitled *Transmission Lines and Cables*, for more details on the above enhancements.

## Contacting Us

The Manitoba HVDC Research Centre (MHRC), a division of Manitoba Hydro International Ltd., and its representatives are committed to providing you with the best sales and technical support available.

Contact your local PSCAD representative first for fast and efficient service. If you do not have their contact address from the time you purchased PSCAD, you can get it either from the PSCAD Web Site ([www.pscad.com](http://www.pscad.com)) or by contacting the Centre directly. To make the best use of our technical support and sales facilities you should have a maintenance contract arranged through your local PSCAD supplier.

## PSCAD Support Services

For non-sales related technical support, precedence is given to commercial users with valid software maintenance contracts. We can be reached at:

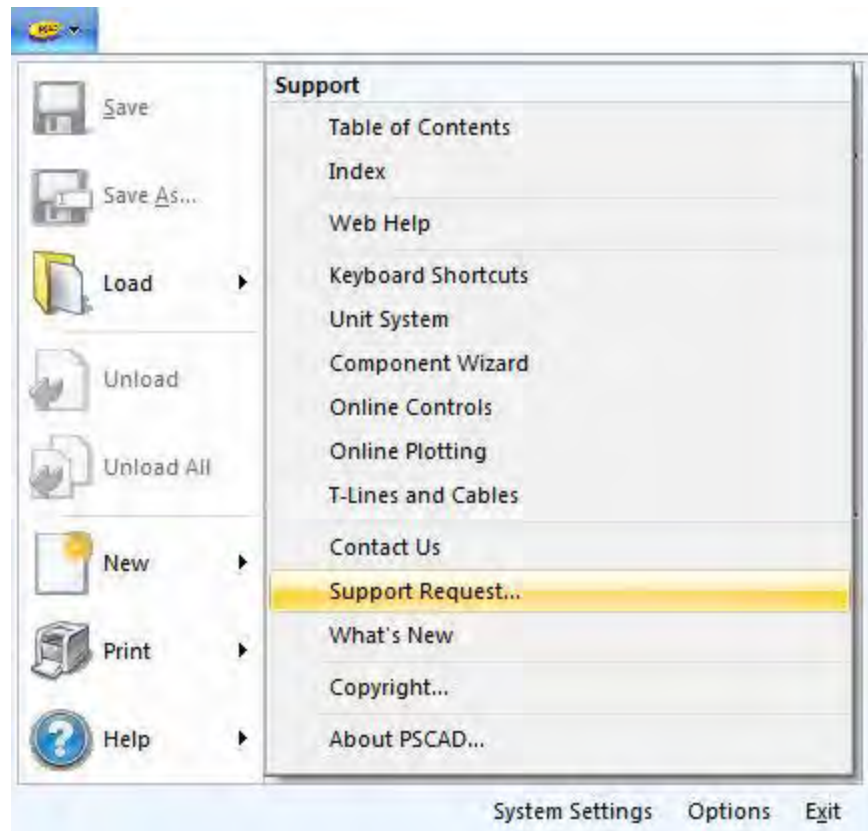
E-mail: [support@pscad.com](mailto:support@pscad.com)

Web Site: [www.pscad.com](http://www.pscad.com)

## Support Petition Request

In order to provide faster, more efficient support service, we recommend that users issue a *Support Petition Request* directly from PSCAD, rather than e-mailing support directly. A *Support Petition Request* will automatically contain information regarding your PSCAD version, compiler version and license data, appended to the request. This will minimize the need for 'back-and-forth' communications, which can be cumbersome especially for our Asian and European users.

To send a support query, go to the Main Menu bar and select **Help | Support Request...**



This will bring-up the *Support Request* pane. Simply add your comments in the spaces provided. Please ensure that you select a **Problem Description** from the drop list, which most adequately describes your problem type. When finished, press the **Submit** button. An email will be sent directly to *PSCAD Support Services*.

Support Request

# PSCAD Support

Welcome to support. Using this pane you may send a quick question to the support staff, or add more detail using the optional tools below.

## Basic Message

### Problem Description

### Categories

Please Choose ▾

## More Details

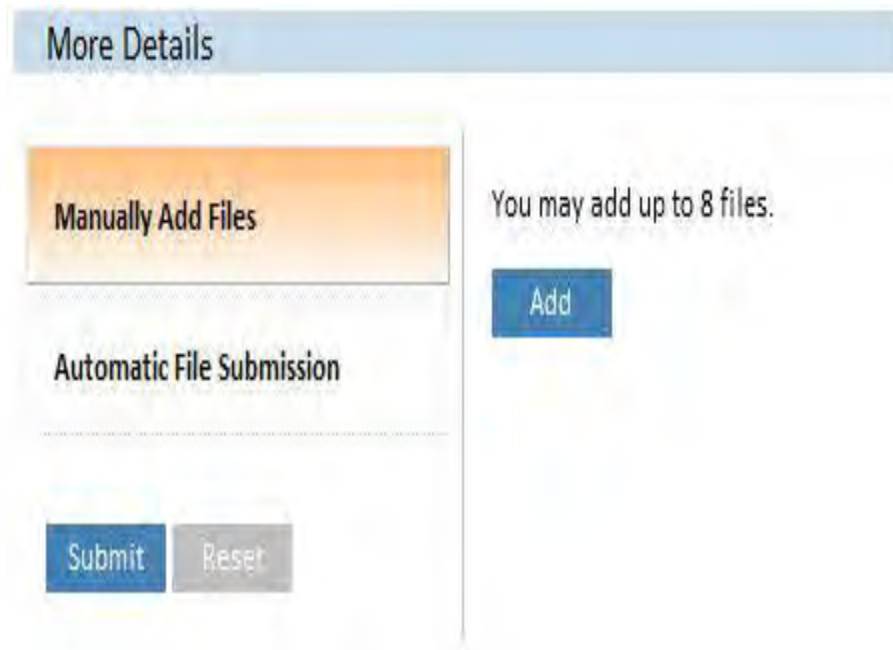
### Manually Add Files

### Automatic File Submission

SubmitReset

## Manually Add Files

To attach specific files to your request, simply press the **Manually Add Files** area and then the **Add** button. This action will invoke a file explorer dialog, where you can choose up to 8 files. Once selected, press the **Open** button to attach the files and close the dialog. Press the **Submit** button to send the request to the PSCAD Support Desk with the files attached.



The screenshot shows a web interface titled "More Details". On the left, there is a vertical panel with two sections. The top section is titled "Manually Add Files" and contains an orange button labeled "Manually Add Files". Below this is a section titled "Automatic File Submission" which contains a blue "Submit" button and a grey "Reset" button. To the right of this panel, the text "You may add up to 8 files." is displayed above a blue "Add" button.

## Automatic File Submission

This feature allows you to quickly add most important files needed by *PSCAD Support Services* to diagnose and solve your installation and/or licensing issue. Simply press the **Automatic File Submission** area and then select the files you would like to automatically attach. Press the **Submit All** button to select all files. Once selected, press the **Submit** button to send the request to the PSCAD Support Desk with the files attached.

**More Details**

**Manually Add Files**

**Automatic File Submission**

**Submit** **Reset**

Please choose which files you would like to send automatically to PSCAD Support.

- License Manager Files
- Fortran Medic Results
- Current Project File
- Workspace File

**Submit All**

## PSCAD Q&A Forum

A new PSCAD Q&A forum went live in April 2014.

To access it, you must first register a MyCentre account. If you already have a *MyCentre* account, then simply log in and click the PSCAD Q&A tab in the tab bar.



Logging in to *MyCentre* automatically logs you in to the forum.

## Forum Details

Users can submit their questions/comments/examples regarding PSCAD by posting under the appropriate forum. To ask a question, simply type your question in the ask box and wait for an answer. You may also browse existing questions.

Your post will immediately be viewable to all registered members. If you are responding to an existing topic, the forum will automatically send an e-mail to the member who posted the question.

**NOTE:** If you have queries regarding problems simulating with PSCAD, bug reports, etc., please contact the PSCAD Support desk by using either the Support Petition Request, or directly at [support@pscad.com](mailto:support@pscad.com). Contacting support services will ensure a prompt response to your query.

## PSCAD Sales

For sales related inquiries (i.e. quotations, etc.), we can be reached at:

E-mail:	<a href="mailto:sales@pscad.com">sales@pscad.com</a>
Phone:	+1 (204) 989-1240
Fax:	+1 (204) 989-1277
Web Site:	<a href="http://www.pscad.com">www.pscad.com</a>
Address:	PSCAD Sales Manitoba HVDC Research Centre (a division of Manitoba Hydro International Ltd.) 211 Commerce Drive Winnipeg, Manitoba R3P 1A3 Canada



## Chapter 2

### License Management

## Licensing Overview

All PSCAD software editions must be licensed to run. There are a few different licensing modes, as well as a few different license types.

The following are the currently available license types:

- **Lock-Based License:** This license type uses a combination of hardware lock and license database file to authenticate. There are two types of lock-based licenses:
  - **Single-User License:** A single license is installed and used locally on a workstation.
  - **Multi-User Licenses:** One or more seats (i.e. instances of the PSCAD application) may be used locally, in addition to being shared with other workstations on the Local Area Network (LAN).
- **Lockless Trial License:** This time-limited license type uses a license database file, but no hardware lock to authenticate. A lockless trial license is used primarily for the evaluation of our fully-featured software editions.
- **Certificate License:** A certificate is an electronic license that is acquired via a cloud-based license server. Certificate licenses are authorized through the customer's MyCentre account (MyCentre Licensing). A temporary, renewable license certificate is granted locally on the user's workstation upon request from within the PSCAD application. Certificate licensing does not require a hardware lock or a license database file.

The following are the currently available licensing modes:

- **Standalone License Manager:** The standalone license manager, installed as a separate product, manages licensing for one or more seats (i.e. instances of the PSCAD application). The standalone license manager grants licenses to either the host workstation or to other workstations on the network, and is used only for lock-based licenses.
- **Self-Licensing:** Under this licensing mode, the PSCAD software can license itself. Self-licensing is available only on the local workstation, and cannot be shared with other workstations over the network.
- **MyCentre Licensing:** This licensing mode functions through a member's MyCentre web portal account.

The licensing mode for each license type is summarized in the following table:

License Types	Licensing Modes		
	Standalone License Manager	Self-Licensing	MyCentre Licensing
Multi-User Lock-Based License	✓		
Single-User Lock-Based License	✓ <sup>1</sup>	✓	
Lockless Trial License		✓	
Certificate License			✓

<sup>1</sup>The standalone license manager is not recommended in combination with a single-user lock-based license, since it can potentially overcomplicate product licensing..

# Lock-Based Licenses

With lock-based licenses, validation is achieved through the installation of a database file and hardware lock. Licenses are organized and controlled using license manager software, which can exist as either a separate, standalone service, or embedded within the PSCAD application itself.

Lock-based licenses come in two different styles: Multi-user and single-user. The style that is best for your purposes depends on how PSCAD is to be utilized, as well as on what type of license(s) you purchased. The following sections describe each type of lock-based license in detail.

## Multi-User License (MUL)

Multi-user licenses (MUL) were first introduced with PSCAD V3 and were, at the time, the only license type available. With this configuration, the license manager software is installed as a standalone service, which can be accessed by any workstation (including the host) on a Local Area Network (LAN).

**NOTE:** The license manager can only license client workstations that reside on the same network as itself, and where the network is constrained by its network class (the term *network class* is defined by RFC-960, which is part of the Internet Protocol Standards). The subnet masks on the license manager and client workstations may need to be adjusted, so that the client can contact the license manager workstation. A new configuration, called *Expanded Licensing*, is available to allow a license manager to license clients across two networks. Expanded licensing is available for free, and requires License Manager v1.30 or later. This licensing is temporary but renewable, and expires after three years for an Educational Edition, or at the end of a maintenance contract for a Professional Edition.

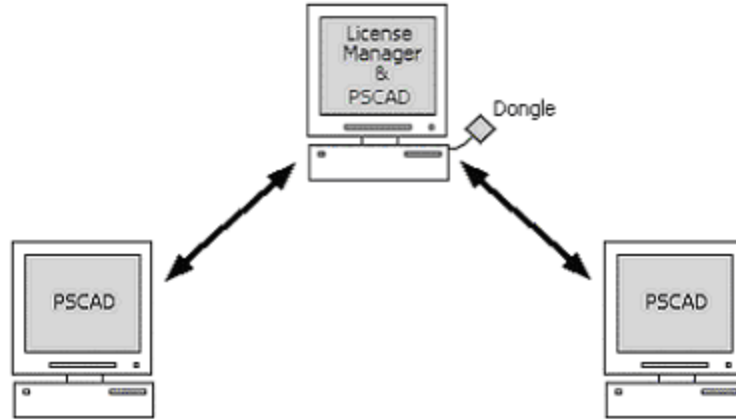
A multi-user license set-up is capable of providing multiple users on multiple workstations the ability to run multiple combinations of various licensable editions of PSCAD. In other words, if a user is sufficiently licensed, this method has the potential to afford a lot of freedom when operating over a LAN, and should be used if there will be two or more users running PSCAD simultaneously.

The license manager is used in conjunction with a hardware lock (also known as a USB lock or dongle), which contains licensing information for the purpose of validation. When an instance of PSCAD is started somewhere on the LAN, a license will be requested from the license manager. The license manager determines whether a license is available, and checks the information on the dongle for verification.

The number of users that can access PSCAD simultaneously over the LAN depends on the number of licenses purchased. For example, if there are a total of two Professional Edition licenses, then only two users may use the Professional Edition at the same time, or one user may use two instances of PSCAD on his or her workstation. Each time a user opens or closes the application, it will request or relinquish a license respectively. Each licensable edition possesses a separate product number, so that several products can be licensed by the same license manager.

**NOTE:** The number of licenses owned does not limit the number of PSCAD installations on the LAN. You can install PSCAD on every workstation in your organization if you wish.

The license manager and dongle need be installed only on one workstation in a network. This workstation will act as the license manager server, which will grant licenses to other workstations on the network, as requested. PSCAD may also be installed on the license manager server workstation if desired.



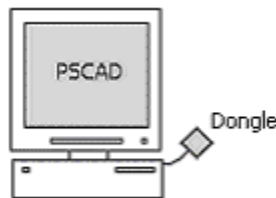
**EXAMPLE:** A user owns one Professional Edition license containing three PSCAD seats, all of which are programmed onto a single hardware lock. The three PSCAD seats refer to a maximum of three instances of PSCAD that may be run at the same time. The above diagram illustrates how the user could install the license manager on a LAN, so that up to three users can run PSCAD simultaneously.

Multi-user licenses can be time limited or permanent (no time limit).

## Single-User License (SUL)

A single-user license (SUL) involves the use of a hardware lock, but a standalone license manager service is not required. An embedded license manager within the PSCAD application itself manages the single-user license.

A single-user license will only allow a single user on a single workstation to run only one instance of a licensable software edition. A single-user license is designed specifically for use on standalone workstations, and will NOT allow access from others.



**EXAMPLE:** A user owns one Professional Edition single-user license programmed onto a single hardware lock. The above diagram illustrates how the user would install PSCAD as a single-user configuration. Note the absence of the license manager as a separate program.

Single-user licenses can be time limited or permanent (no time limit).

## Hardware Locks (Dongles)

The License Manager software for both single-user and multi-user licensing supports the following hardware lock in lock-based licensing:



Sentinal SuperPro™ USB, USB Port Hardware Lock

The USB hardware lock is compatible with all Windows platforms on which PSCAD is supported.

## License Key/License Database File

The license key is an ASCII text file named *license.txt*. The license key is used with both single-user and multi-user licenses, and is required for both the Professional and Educational Editions. This file contains encrypted information designed to act as a key to adding or upgrading a license on a workstation, specifically by creating and updating a license database file. Together, the hardware lock and license database file are used directly by either the standalone license manager or self-licensing to validate a request for a license.

## Certificate Licenses

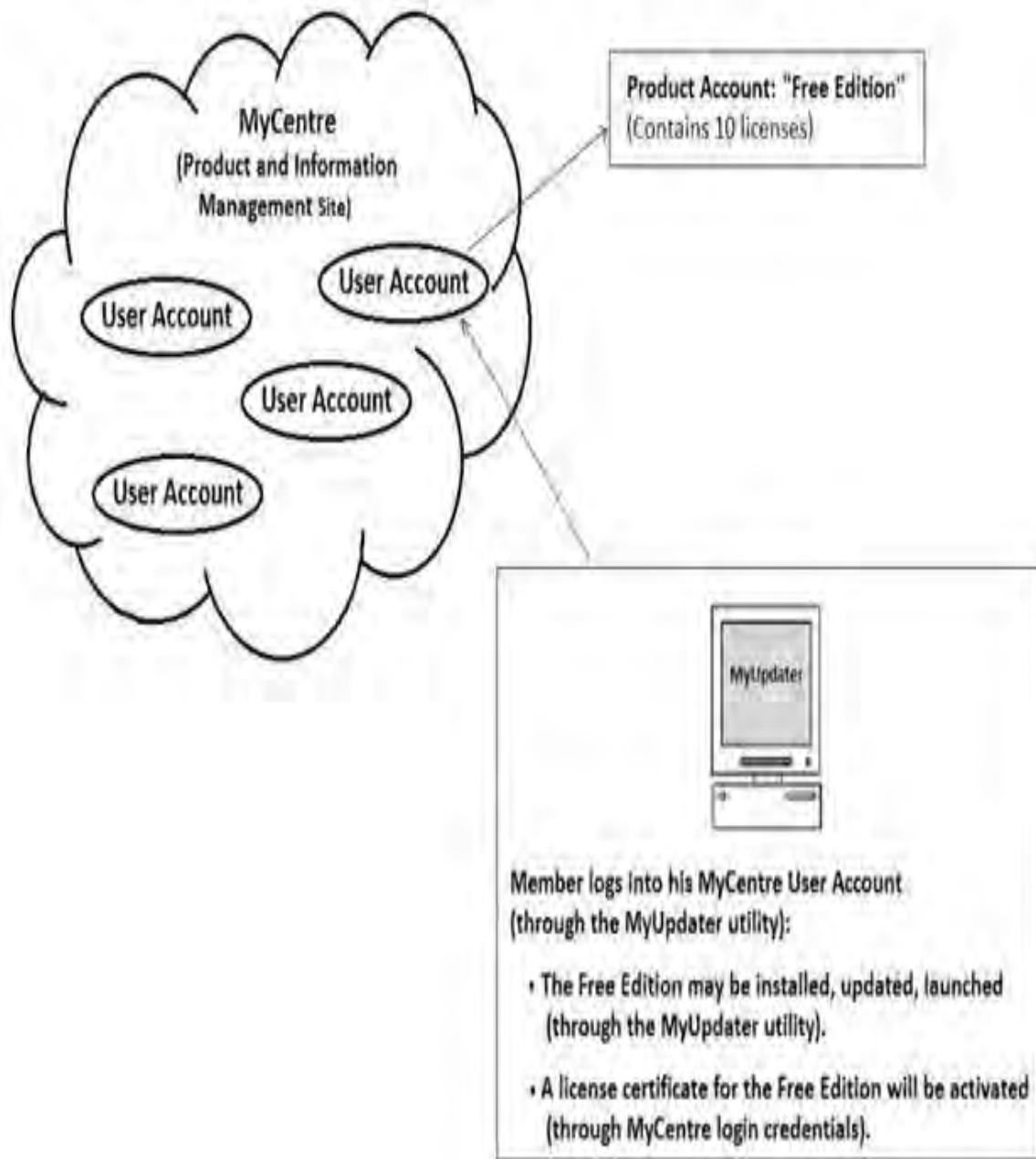
Certificate licenses (available starting in 2013), are the preferred method for licensing PSCAD. Unlike lock-based licenses, this method does not require a hardware lock and database file. A license certificate is a temporary authorization to run PSCAD, and is granted through a user's MyCentre login credentials.

The following sections contain an overview of MyCentre and certificate licenses. For more detailed information and instructions, refer to the document *Managing MyCentre*, located on the following web page: <https://hvdc.ca/knowledge-library/reference-material>.

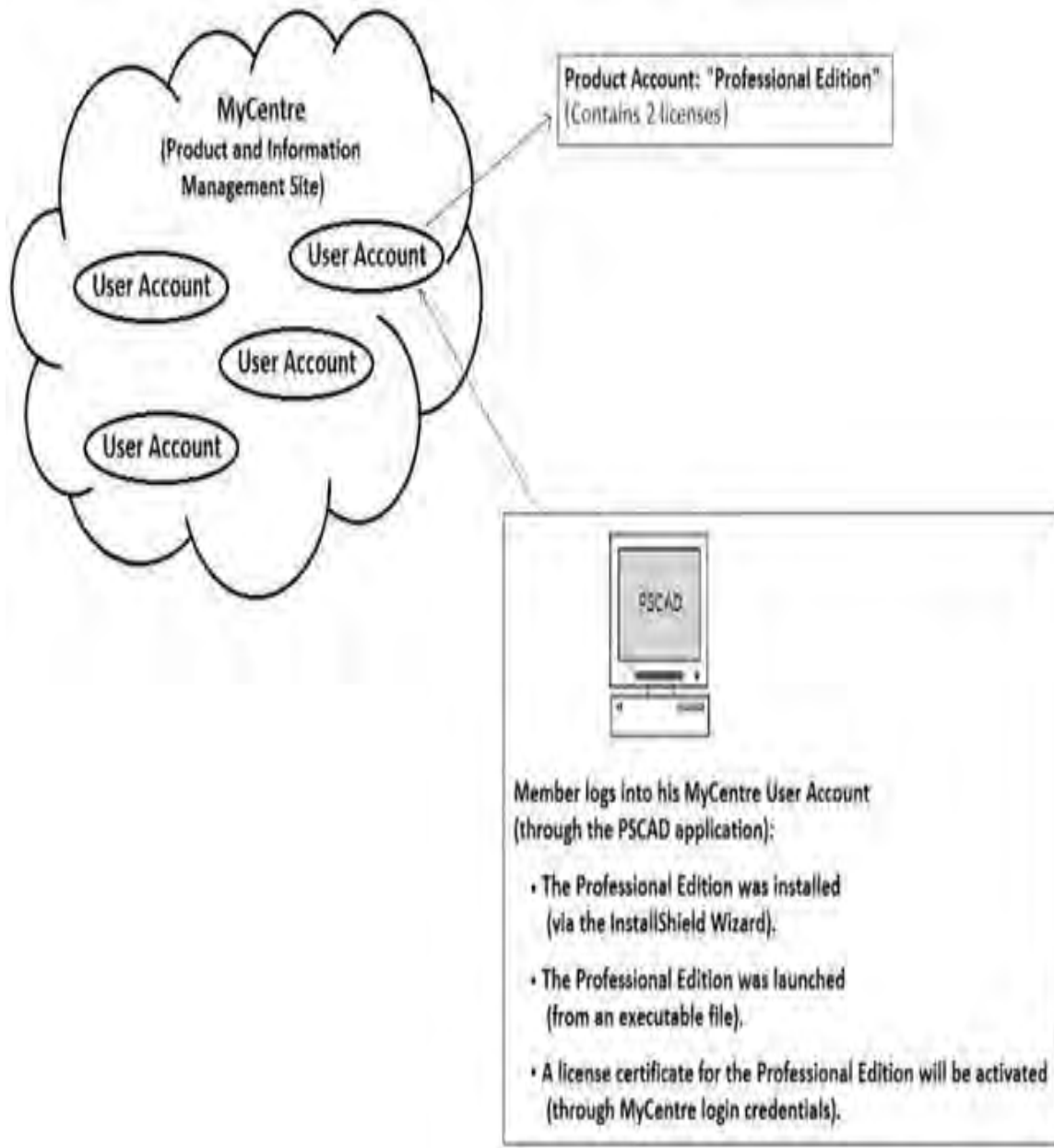
### Overview

Members register a MyCentre user account, in which their licenses are stored. A member's login credentials will authorize one of their licenses, and in turn activate a license certificate in the form of a file on their workstation. Licenses and products, exclusive to a member, are held in product accounts. In some cases, a software deployment utility, called MyUpdater, is used to install, uninstall, update and launch PSCAD. In others, software is installed via the *InstallShield Wizard*, and launched from an executable file.

The following figure illustrates the elements for running software using MyCentre and MyUpdater.



The following figure illustrates the elements required for running software using MyCentre, but not MyUpdater.



Certificate licensing is applicable to the following editions and license types:

- The Professional Edition of v4.6 and up (permanent, temporary and trial licensing)
- The Educational Edition of v4.6 and up (trial licensing)

**NOTE:** Permanent and temporary licensing for the Educational Edition is not available in v4.6.0.

- The Professional and Educational Editions of v4.5 (trial licensing)

**NOTE:** Alternatively, a temporary licensing is also available through a lockless trial License.

- The Free Edition and Beta Edition

## MyCentre

A certificate license is authorized through a web portal called MyCentre (<https://mycentre.hvdc.ca>). MyCentre is a software and information management portal designed to provide assistance to users of PSCAD. In addition to hosting current and relevant reference materials and a user forum, MyCentre provides software products and licensing.

### User Account

Members register a user account to access their MyCentre products. A user account contains all the products for which the member is eligible. For example, every member is automatically granted access to the Free Edition and given ten licenses for it.

### Product Account

Members' products, such as licenses, are all held in product accounts. Accounts are used to control the availability of products to members by controlling access. An educational facility, for example, may make its Educational Edition available only to select students by providing them with access to an account containing that edition.

An account may contain one or more licenses. The product account for the Free Edition, for example, typically contains ten licenses.

An account may be shared between members. A commercial facility with five licenses for the Professional Edition, for example, may have one product account containing all five licenses. Staff members have access to this account through their user account.

An account may contain licenses for one or more different editions. An educational facility, for example, may have a product account containing licensing for its Professional and Educational Editions. Conversely, the facility may split its licensing into two separate product accounts, one for its graduate students, and the other for its undergraduate students.

### License

Product accounts contain licenses. Licenses dictate the software edition and version, type of license, and licensing duration. Some licenses are permanent, while others are temporary. Licenses for the Free Edition and for permanent Educational and Professional Editions have no expiry date. Licenses for the Beta Edition, and for software used for training or a Trial License, typically have expiry dates applied.

### License Certificate

A license certificate is a temporary and renewable authorization for running PSCAD on a machine. A user may obtain a license certificate on a machine by using their MyCentre login credentials. This will access his or her product account containing the license.

A license certificate is temporary, and expires according to the following conditions:

- It will automatically be set to expire approximately four weeks following the initial activation date, so long as the license does not expire during that period. This permits a period of offline usage.

**NOTE:** If a member intends to work offline with a license, he must ensure that he configures the licensing for offline usage for the full 28-day period.

**NOTE:** If, after using a license certificate on Machine A, the license will next be used on Machine B, the member must ensure to release the license certificate from Machine A. Otherwise, the license certificate may not be activated on Machine B.

- It may be released (expired) at any time by the member, and returned to the account. It may then be re-activated for an additional four weeks, so long as the license does not expire during that period.

Once a license certificate is activated, multiple instances of PSCAD may be run on a workstation using a single license certificate. The number of instances run on a workstation is only limited by the memory size. This is a major advantage of a certificate license over a lock-based license; in the latter, each instance of PSCAD consumes one seat (the equivalent of a license certificate).

## MyUpdater

One additional consideration when using a certificate license is that certain software requires the use of a software deployment utility, called *MyUpdater*. MyUpdater is installed on a member's workstation, and is used to install, uninstall, update and launch software. Specifically, the Free and Beta Editions require its use. Occasionally, a pre-release version may be provided to select members via MyUpdater to deploy required fixes or to test new features. MyUpdater is optional for the deployment of the Professional Edition, and is not yet available for the Educational Edition.

For more detailed information and instructions for using MyUpdater, refer to the document "Managing MyCentre", located on the following web page: <https://hvdc.ca/knowledge-library/reference-material>

## Obtaining a MyCentre User Account

Anyone may join MyCentre. Following registration, a MyCentre user account will be created, and will contain the member's unique product set. The Free Edition is automatically made available in each user account (see below for instructions on obtaining this edition). A user account is required for obtaining the following:

- Running the Free Edition
- Running the Beta Edition
- Running a Professional Edition using certificate licensing (permanent)

**NOTE:** A MyCentre user account is not required when running a professional edition using lock-based licenses.

- Running a temporary Professional or Educational Edition using certificate licensing for either a training session or a trial license

**NOTE:** A MyCentre user account is not required when running a trial license using the *trial.txt* file.

See Section 2.2 of the *Managing MyCentre* document on the following web page for instructions on registering a User Account: <https://hvdc.ca/uploads/ck/files/ManagingMyCentre.pdf>

## Obtaining the Free Edition

The PSCAD X4 Free Edition is available to any member with a MyCentre user account. The Free Edition is a limited-feature edition, and allows members to become familiar with PSCAD tools and capabilities. The MyUpdater utility is used to deploy the software, and licensing is authorized with your MyCentre login credentials.

See Section 4 of the *Managing MyCentre* document on the following web page for instructions on obtaining the Free Edition: <https://hvdc.ca/uploads/ck/files/ManagingMyCentre.pdf>

## Obtaining the Beta Edition

The Beta Edition is available to qualified members. It provides a means for users to directly affect the development of PSCAD, by using and critiquing new features and functionalities before they are released. The MyUpdater utility is used to deploy the software, and licensing is authorized with your MyCentre login credentials.

See Section 5 of the *Managing MyCentre* document on the following web page for instructions on obtaining the Free Edition: <https://hvdc.ca/uploads/ck/files/ManagingMyCentre.pdf>

## Running the Professional Edition

The Professional Edition is available using a certificate license as of PSCAD v4.6 and later. Alternatively, it is available using a lock-based license.

To run a Professional Edition with certificate licensing, please contact our *Sales Desk* ([sales@pscad.com](mailto:sales@pscad.com)).

## Running a Temporary Edition

A temporary Professional or Educational edition is available using certificate licensing with PSCAD v4.6 and later. Alternatively, trial licensing is also still available using a *trial.txt* file.



To run a temporary edition with certificate licensing, please contact our *Sales Desk* (sales@pscad.com).

## Viewing Active License Information

Open the System Settings dialog. Select the **Certificate Licensing** tab.

In the middle of the dialog, there is an area entitled **Status**. Some preliminary information is given directly within this area. For more detailed information, click the **View...** button. The **License Certificate** dialog will display, and provide all available information regarding your current licensing.



Clicking in a field will display a corresponding description in the bottom (General) section.

## Standalone License Manager

The standalone license manager is typically used only with a multi-user lock-based license (MUL). The standalone license manager can operate on a dedicated server, or on any workstation running PSCAD. Any combination of products may be included and used interchangeably. Installation instructions are provided later in this chapter.

### Maintenance and Support

Once the standalone license manager is installed and running, it is recommended that the person operating the designated license manager server maintains the software and ensures proper operation.

The standalone license manager usually starts automatically when the workstation is booted and runs as a background process. All standalone license manager events and transactions are recorded in a log file entitled *Lmgrd-log.txt*, which is located as indicated below. When troubleshooting, this file contains important information about the cause of any problems.

- **On Windows XP:** The *Lmgrd-log.txt* file is located as follows:
  - Documents and Settings\All Users\Application Data\Manitoba HVDC Research Centre\License Manager
- **On Windows 7:** The *Lmgrd-log.txt* file is located in one or more of the following locations (please send in all files as available):
  - C:\Users\All Users\Manitoba HVDC Research Centre\LicenseManager
  - C:\Users\YOUR-USER-ID\AppData\Local\Manitoba HVDC Research Centre\LicenseManager
  - C:\ProgramData\Manitoba HVDC Research Centre\LicenseManager

If problems arise with the operation of your license manager software, first please try to address the problems using the clues found in the log file. If the solution remains elusive, contact us at PSCAD Support Services with a detailed description of the problem. To ensure efficient response time, please also attach the following:

- The log file from the server (i.e. *Lmgrd-log.txt*), which is found on the C-drive, as indicated above.
- The log file from the client workstation experiencing the problem, *PscadLmgr.txt*, which is found on the C-drive, as indicated above.
- A Get Info log file from both the server and the client workstation. Go to **START | Programs | PSCAD | Utilities**, run the GetInfo utility, and attach the generated file (*GetInfo.txt*) to the e-mail.

### PSCAD Usage Log

For the standalone license manager service, the usage of all licensed PSCAD editions is logged to a file. This information is saved in a comma-separated variable (\*.csv) file called *PscadUsage.txt*, which is located in the folders as listed in Maintenance and Support on the workstation running the license manager software. This file can be directly imported into a spreadsheet program.

The following is a sample line from a *PscadUsage.txt* file:

lha, 2014	Pro, Remote,	7327,	177, Mon Aug 22 10:45:42 2014, Mon Aug 22 10:48:39
lha, 2014	Pro, Remote,	7328,	176, Mon Aug 22 10:46:21 2014, Mon Aug 22 10:49:17

The format of the file is as follows:

<user id>, <Pscad Edition>, <user location>, <LicenseID>, <#seconds of use>, <start time>, <finish time>

Description:

- **<userid>**: This is the login userid of the PSCAD user.
- **<Pscad Edition>**: Can be *Pro* (Professional) or *Edu* (Educational). Usage of the Free and Beta Editions is not logged, as these are not licensed editions, and do not require the license manager (see Certificate Licenses).
- **<user location>**: *Remote* for users using PSCAD on workstations other than the license manager host. *Local* for uses using PSCAD on the same workstation as the license manager host.
- **<LicenseID>**: This is the license ID of the granted license, and appears in the license manager log file (*Imgrd-log.txt*), and in the PSCAD usage log file (*PscadLmgr.txt*)
- **<#seconds of use>**: The number of seconds that the license was in use.
- **<start time>**: The time that the license manager granted the PSCAD license to the user.
- **<finish time>**: The time that the user quit the PSCAD application

## Adding/Modifying Licenses

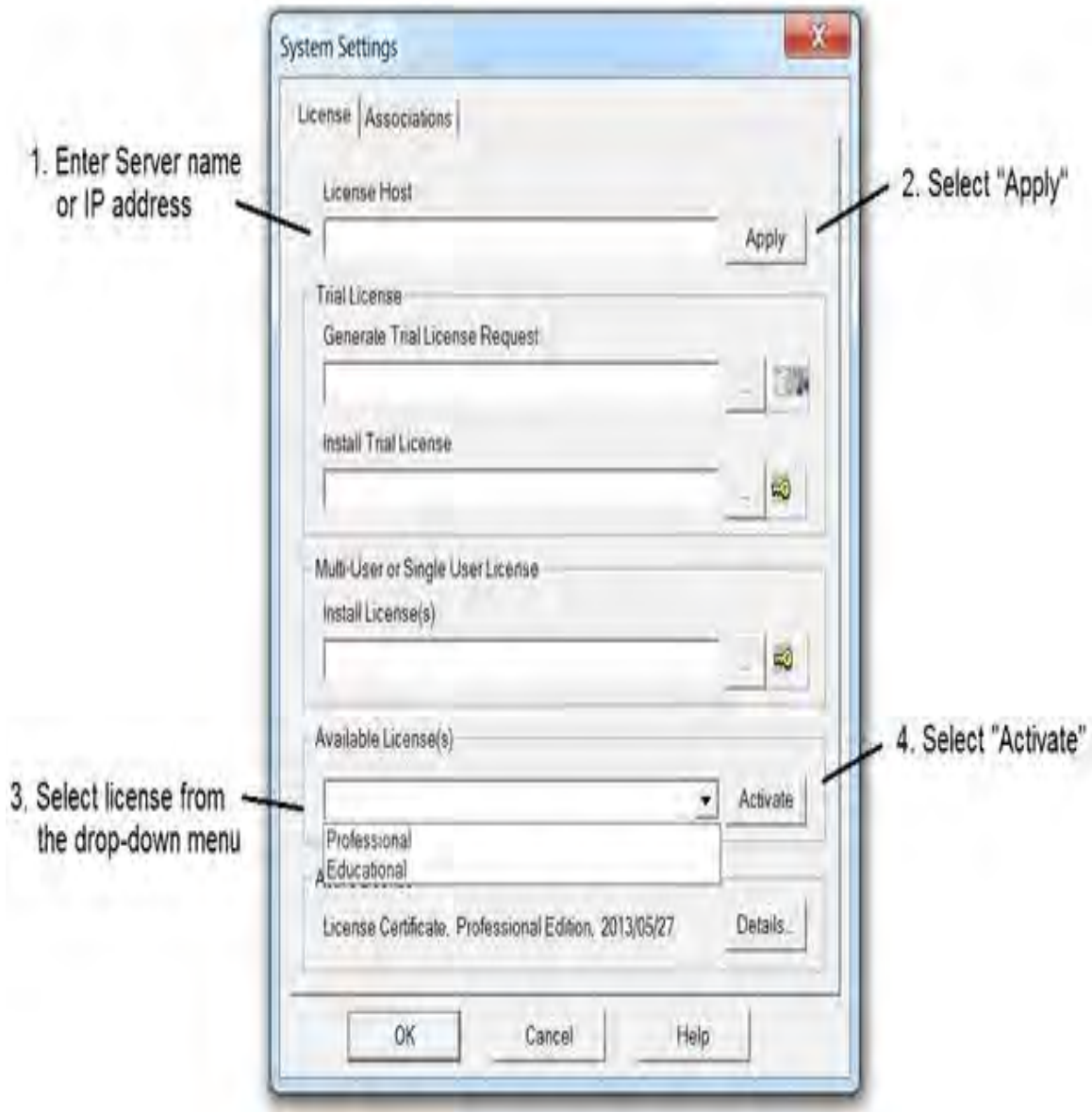
When using a standalone license manager, either a multi-user lock-based license (MUL) may be added as a new installation on a workstation, or a pre-existing multi-user license may be modified. A license is modified in the following circumstances:

- **Adding Products to a License**: This is typically performed when adding another PSCAD seat, or adding other products such as LiveWire, or expanded licensing. In this case, there is no change to the hardware lock; only the license database file, residing on the workstation's hard drive, is modified.
- **Renumbering the License**: In this case, both the hardware lock and the license database file are modified, and the license is renumbered to a new license number. This is typically performed in the following situations:
  - When removing products from a license (e.g. a PSCAD seat, or other products such as LiveWire or expanded licensing).
  - When updating a license for a significant software update (e.g. for updating a license from V4 to X4 (i.e. v4.3 or later)).

## Adding or Modifying a License

To add a new license or to modify a license that will not be renumbered:

- You will be provided with a new *License.txt* file. Save this file to a convenient directory (e.g. C:\temp).
- Ensure the hardware lock is plugged in.
- Display the License Update Utility (from the Windows Start menu, go **All Programs | PSCAD**, then browse to **Enter License Key**).
- From the **Actions** menu, select **Enter license keys**.
- When prompted, browse to the *License.txt* file, then select **Open**.
- The license will be installed or modified as applicable.
- For new license installations, ensure that all client workstations that will obtain a license from this server are configured as shown (launch PSCAD and display the System Settings dialog from the PSCAD Start button):



## Renumbering the License

When renumbering a license, it may be required to update the license manager to the latest version, determined as follows:

- When removing products from a license, the license manager is not required to be updated.
- When performing a significant software update, the license manager may be required to be updated.

To renumber an existing license:

- You will be provided with an *Upgrade.txt* file and a new *License.txt* file. Save these files to a convenient directory (e.g. C:\temp).
- Ensure the hardware lock is plugged in.
- If required, update the standalone license manager software.

- Display the License Update Utility (from the Windows Start menu, go **All Programs | PSCAD**, then browse to **Enter License Key**).
- From the **Actions** menu, select **Upgrade license keys and lock**.
- When prompted, browse to the *Update.txt* file, then select **Open**.
- When prompted, browse to the *License.txt* file, then select **Open**.
- Both the license database file and the hardware lock will be updated.

If you encounter any issues during this process, contact us at PSCAD Support Services.

## Changing Active Licensing Settings

For lock-based licenses, PSCAD allows you to alter your present (i.e. 'active') license settings without the need to close and reopen PSCAD in a different mode. You can change the license manager host workstation, and/or the licensed edition directly from the System Settings dialog.

### Changing License Manager Host

To change the license manager host workstation:

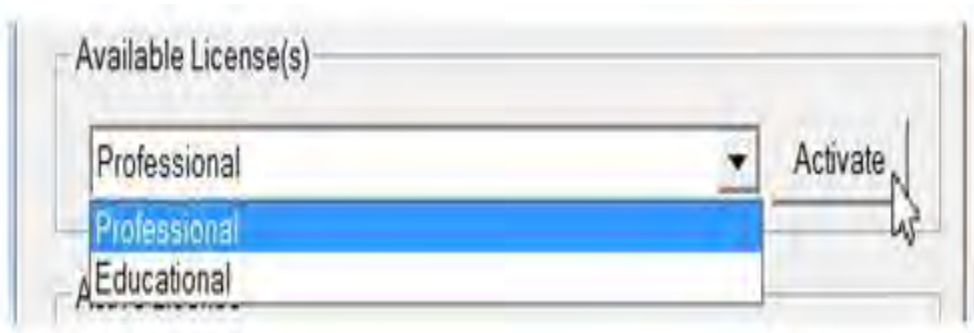
- Enter the name of the new host (along with 2053), and click the **Apply** button.



- Select an available license (see below).

### Changing Active License Type

To change the active license type, click the down arrow in the selection box under the **Available License(s)** area and select the required license. Click the **Activate** button.



**NOTE:** You must be properly licensed in order to change license type!

## Viewing Active License Information

At any time, you can review the status of your active license.

1. Open PSCAD and open the System Settings dialog.
2. Near the bottom of the dialog, there is an area entitled **Active License**. Some preliminary information is given directly within this area. For more detailed information, click the **Details...** button. The *License Certificate* dialog will display, and show all available information regarding your current licensing.



## Getting Licensing Information

If you would like information about your current licensing set-up, it may be obtained by running the *Get Info* utility (see *Licensing Information*, below). This information is also helpful in troubleshooting if a problem arises, and is used by *PSCAD Support Services* to help solve licensing issues.

## Licensing Information

To view your licensing information for either a single-user or multi-user lock-based license, use the *Get Info* program:

1. We recommend always using our latest *Get Info* utility, which is frequently updated and improved. It may be downloaded from the following link:

<http://updater.pscad.com/utilities/GetInfo32.zip>

2. Alternatively, a version of this utility is installed when PSCAD is installed, and may be run from the Windows Start menu (go **All Programs** | **PSCAD**, then browse to **GetInfo**).



```

-----
Manitoba HUDC Research Centre Inc.
Licence Manager Information V1.31
(c) 1998-2012 Manitoba HUDC Research Centre Inc.
-----

System Info
Windows O/S: Windows 7 (6.01.7601 Service Pack 1)
IEExplorer:  9.11.9600.17041
Win64:       yes
WOW64:       yes
-----

Lock Info (direct access)
Expected on: <no port specified, or Lmgrd.ini does not exist => assume USB>
Expected on: USB <default>
customer ID 111111 (expected)
customer ID unknown
cause(s) <Dongle absent, not communicating, or wrong port in Lmgrd.ini>
-----

NOTE: This information is also saved in:
      C:\ProgramData\Manitoba HUDC Research Centre\LicenseManager\getinfo.txt

If you are experiencing PSCAD licensing issues or Licence Manager problems,
then please attach the Getinfo.txt file when e-mailing support@pscad.com

Please do NOT send a screen shot as it may not show all the information.

Press [Enter] to exit

```

When you run this program, a DOS-based window should appear similar to that shown above.

A file will also be created at the same time called *getinfo.txt*, the location of which will be listed in the utility itself, as shown in the previous snapshot. This is an important file to include when contacting PSCAD Support Services.



## Manually Starting

The method for starting the standalone license manager may differ slightly for the supported Windows platforms:

### Windows XP/Vista/7

- Right-click on your **My Computer** desktop icon and select **Manage**.
- Double-click on the **Services and Applications** entry and then double-click **Services**.
- Highlight **HVDC License Manager**, right-click and select **Start**.

## Manually Stopping

The method for stopping the standalone license manager may differ slightly for the supported Windows platforms:

### Windows XP/Vista/7

- Right-click on your **My Computer** desktop icon and select **Manage**.
- Double-click on the **Services and Applications** entry and then double-click **Services**.
- Highlight **HVDC License Manager**, right-click and select **Stop**.

## Self-Licensing

### Adding/Modifying a License

The self-licensing mode is typically utilized only when running under a single-user lock-based license (SUL). Under the self-licensing mode, multiple products, such as PSCAD and LiveWire may be licensed.

The self-licensing mode does not support multiple instances of any individual PSCAD product. For example, attempting to start a second instance of a certain product (e.g. Professional Edition), will result in the licensing request being denied.

**NOTE:** Alternatively, with Certificate Licensing, multiple instances of PSCAD may be run using one license certificate.

## Adding/Modifying a License

For self-licensing, either a single-user lock-based license (SUL) may be added as a new installation on a workstation, or a pre-existing single-user license may be modified. A license is modified in the following circumstances:

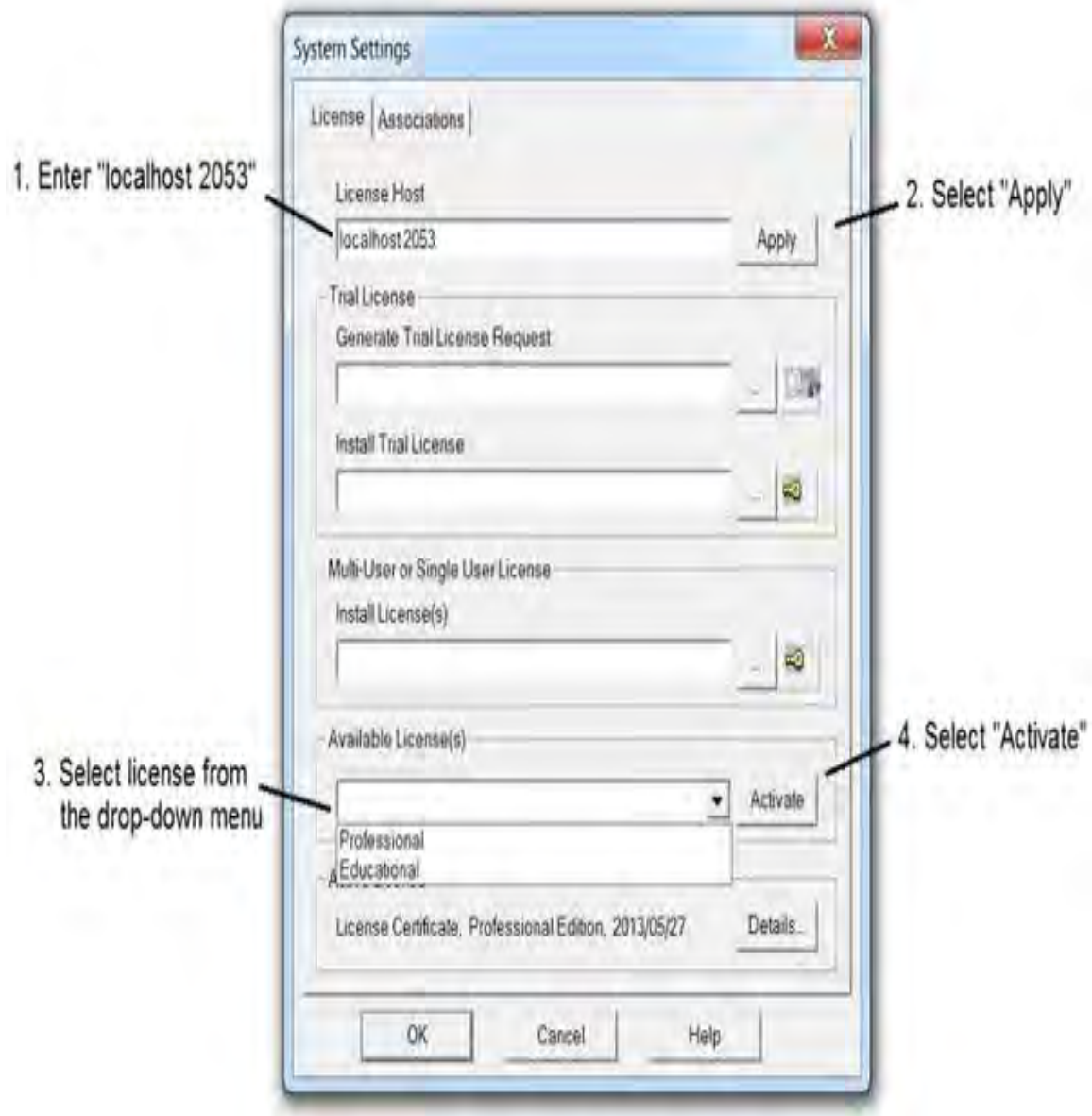
- **Adding Products to a License:** This is typically performed when adding LiveWire. In this case, there is no change to the hardware lock; only the license database file residing on the workstation hard drive is modified.
- **Renumbering the License:** In this case, both the hardware lock and the license database file are modified, and the license is renumbered to a new license number. This is typically performed in the following situations:
  - When removing LiveWire from a license.
  - When updating a license for a significant software update (e.g. for updating a license from V4 to X4 (i.e. v4.3 or later)).

### Adding or Modifying a License

To add a new license or to modify a license that will not be renumbered:

- You will be provided with a new *License.txt* file. Save this file to a convenient directory (e.g. C:\temp).
- Ensure the hardware lock is plugged in.

- Display the License Update Utility (from the Windows Start menu, go **All Programs | PSCAD**, then browse to **Enter License Key**).
- From the **Actions** menu, select **Enter license keys**.
- When prompted, browse to the *License.txt* file, then select **Open**.
- The license will be installed or modified as applicable.
- For a new license installation, configure the PSCAD licensing as shown (launch PSCAD and display the System Settings dialog from the PSCAD Start button):



#### Renumbering the License

To renumber an existing license:

- You will be provided with an *Upgrade.txt* file and a new *License.txt* file. Save these files to a convenient directory (e.g. C:\temp).
- Ensure the hardware lock is plugged in.
- Display the License Update Utility (from the Windows Start menu, go **All Programs | PSCAD**, then browse to **Enter License Key**).
- From the **Actions** menu, select **Upgrade license keys and lock**.
- When prompted, browse to the *Update.txt* file, then select **Open**.
- When prompted, browse to the *License.txt* file, then select **Open**.
- Both the license database file and the hardware lock will be updated.

If you encounter any issues during this process, contact us at PSCAD Support Services.



## Chapter 3

### Application, Project and Workspace Options

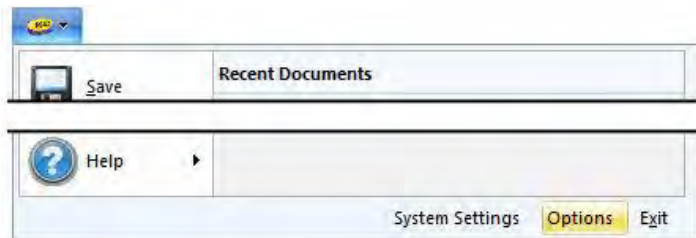
# Application, System and Workspace Options Overview

Now that the application has been installed successfully, you may want to adjust some of the work environment options and preferences that are available.

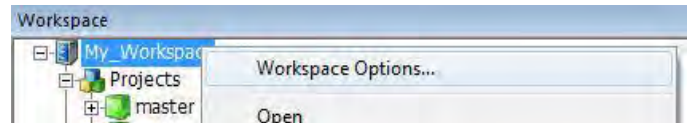
The user-controlled options available are divided into different levels, depending on functionality:

- **Application Options/System Settings:** Effect all parts of the PSCAD environment, including all workspaces and projects.
- **Workspace Options:** Effect only the current workspace and all projects within.

Click on either the **Workspace Options** or **System Settings** buttons in the ribbon control file tab to access either of these dialogs.



For *Application Options* or *System Settings*, click on the PSCAD tab in the ribbon and then press the corresponding button (**Options** or **System Settings**).



For *Workspace Options*, right-click on the workspace branch and select **Workspace Options**.

The settings available are organized into several categories:

#### Application Options

- Workspace
- Environment
- Blackboxing
- Dependencies
- External Tools
- Advanced Licensing

#### System Settings

- License
- Associations

## Workspace Options

- Projects
- Build
- Runtime

## Application Options

# Workspace

This category contains options that relate to all workspaces loaded into the application.

## Display

- **Namespace:** Use this option to control how the module names are displayed in the workspace hierarchy tree. By default, module namespace will be displayed only if the module definition is stored in an external project.
- **Transmission Segment Instances:** Use this option to control whether or not transmission lines and/or cables are displayed in the workspace hierarchy tree.
- **Simulation Stop Warning:** Enable this option to ensure that a confirmation dialog is displayed on a stop action of a simulation run.
- **Runtime Progress Bar:** Superimpose a progress bar atop the project name in the workspace project tree during runtime of a single project.
- **Workspace Pane Scaling:** Use this option to enable scaling (zoom) of the workspace pane contents. Note that graphical artifacts may appear during scaling or scrolling when this option is enabled.

## New Session

- **On Startup:** Choose the preferred method for loading the workspace when PSCAD is started.
- **Start Page:** Controls whether or not the start page is opened on application start up.
- **Workspace Path:** This field simply displays the workspace file path for the last loaded workspace. This path is updated when the application is closed and the user has selected to *Restore the last loaded workspace* under the *On-Startup* application option (above).

## Projects

- **My Projects:** Enter a default directory path for the *Load Project* dialog. Once a path is entered in this field, the *Load Project* dialog will always start in the specified folder. If the field is left blank, then PSCAD will default to the folder last accessed by either the *Load Project* or *Save As...* dialog. You can either enter the path directly, or use the browse button to select the directory. This path may be changed at any time.
- **Change Tracking:** Enables clipboard undo and redo if set to *Use Undo/Redo Stack*.
- **Folder Access Tracking:** If set to normal, file dialogs will open to the folder of the focused project. If last folder accessed is selected, file dialogs will open as such.
- **On Unload Library:** Select whether or not you wish to be notified when unloading a library project that has inter-project dependencies (i.e. has definitions that are instantiated in other projects).

## v4.x Legacy Import/Load

- Obsolete 'datatap2' components:** When this application option is selected, PSCAD will replace any existing obsolete 'datatap2' components with their modern equivalent, and issue a navigable message so that each replacement in the project can be manually checked. This is an important step, as the actual coordinate of the 'datatap2' signal will be shifted by one grid point (down and to the left) on the Schematic canvas, so as to accommodate the 'datatap' component. This may lead to signal source contention errors that should, for the most part, be detected by the compiler when you build the case.
- Parameter Synchronization:** This option is important for ensuring that component parameters are kept up to date when new versions of the associated component definition become available. Component parameters may be added or subtracted between versions of a component definition (ex. master library components). PSCAD cannot automatically detect and update these changes, so this option forces PSCAD to scan each component on load to synchronize the parameters. If you are sure there are no changes to any components used in your project, you can set this option to *No action*.
- Obsolete #DEFINE Directives:** When this setting is enabled, all obsolete #DEFINE script directives will be replaced by corresponding, updated syntax (i.e. #LOCAL) on project import or load. This also includes #DEFINE FUNCTION and #DEFINE SUBROUTINE statements (will be changed to #FUNCTION and #SUBROUTINE respectively). Note that this setting can have detrimental effects on import/load speed. Enable only when necessary.
- Parameter/Port to Import/Export Tag Name:** When this setting is enabled, all case mismatches between module parameters and ports and their corresponding import or export tags will be detected and corrected. In past versions, name matching was case insensitive due to the case insensitivity of the Fortran language. However, other programming languages, such as C, are case sensitive, which is the reason for the change.
- Exported Signal Distinction:** This setting is used to close a long standing compilation loophole. In previous versions, an export tag could share its name with a locally declared signal. This issue has been found to cause problems with the Fortran compiler in certain instances. When enabled, this setting will automatically correct this situation.
- Orphaned Global Substitutions:** Enable this option to detect and repair orphaned global substitutions (i.e. globals that exist in the top-level module instance, but not its definition). Global substitutions will become orphaned following their addition and then deletion from the global list. This is caused by a PSCAD issue inherent to the module parameter mechanism, which was fixed in v4.5.2. This repair only needs to be performed once; henceforth, the definition and instance global substitution lists will be kept synchronized. Enable only when necessary.
- Illegal Characters in Script Segments:** When enabled, this option will ensure that apostrophes ('), existing in component definition script segments, are replaced by underscores (\_). This is a necessary replacement, the apostrophe character is illegal for use in Xml.

## Simulations

This category contains options that relate to how simulation processes are managed.

### Simulation Sets

- Process Execution:** Leave this option as *Use Local Machine* at all times. The *Use Grid Engine* option is used with the Xoreax Grid Engine (XGE) feature, licensed under *Centralized License Manager (CLM)*. Contact the PSCAD Support Desk for more information on both the XGE and the CLM features.
- Maximum Namespace Count:** Defines the maximum number projects that can be included in a single simulation set for execution in parallel.

## Environment

The environment section contains a variety of user preferences pertaining to the overall working environment.

## Build and Run

- **Show Build and Run Windows:** Launch build and run executables in separate windows.

## Keyboard

- **Cut/Copy/Paste key:** Select *Ctrl+x,c,v* to cut/copy/paste with the *Ctrl* key. Select *x,c,v* to enable just the corresponding key.

## Mouse

- **Controls and Curve Creation:** This option is directly related to Drag and Drop. By default, all drag and drop functions are invoked using the **Ctrl** key in combination with the left mouse button. Enable this option if you would prefer to designate the **Shift** key (in place of **Ctrl**) to place curves in graphs or control interfaces/meters in control panels.
- **Control Interfaces Wheel Action:** Enable this feature if you would prefer that control interfaces (such as sliders and switches) respond to mouse wheel action. Disabling this setting ensures that control interfaces are not inadvertently modified during navigation using the mouse wheel.

## Navigation

- **Drill Down:** Use this option to revert to the older module navigation style (i.e. *Double-Click* opens the canvas, bypassing any input parameters). In versions X4 and greater, accessing the circuit canvas of a module is considered editing its definition (i.e. *Ctrl + Double-Click*); this combined with the fact that modules can possess input parameters, lead to the change.
- **Tab Colouring:** This option controls the colouring of the project tabs in The Definition Editor.
- **Tab Closure:** Provides options for placement of tab close button.
- **Tab Style:** Provides options for tab appearance.

## Paper Formatting

- **Definition Colour:** Choose a colour to help visually differentiate the instance domain from the definition domain. This option becomes helpful especially when working with modules having multiple instances. See Multiple Instance Modules in this manual for more details.

## Graphics

This section contains a variety of user preferences pertaining to the visual appearance of various graphics.

### Panels

- **Panel Appearance:** Select a preferred panel style for control panels and graph frames. The choices are *Plain Paper* and *3D Shadow Filled*.
- **Panel Color:** Sets the color of the panels when using 3D style.

### Schematic

- **Rendering Quality:** This option can improve the visual quality of all graphics (*GDI+*), including circuits, graphs and meters. The aesthetic improvement comes at a cost in terms of processor time, so for large cases this option should be left as *Best Speed* until runtime has ended. *GDI+* can then be enabled and the canvas refreshed.
- **Tool-tip Appearance:** Select *Animated Popup* to enable animated tool tip pop-up windows.



- **Hovering Appearance:** If set to *Show Bounding Box*, a dashed box will appear surrounding the component over which the mouse pointer is situated.
- **Wire Junction Appearance:** Set this option to *Display Solder Joint* to show connection symbols at all overlapping signal junctions. This feature is used mostly to detect overlapping wires and possible short circuits. Once debugging is complete, this option should be turned off on larger projects, as it can add significant build time.
- **Control Panel Auto-Size:** This option controls the reaction of control panels to the deletion of control interfaces within it.
- **Component Scaffolding:** Draws a bounding box and diagonal to show the footprint of components on the screen.
- **Component Sequencing:** During the compile sequencing, this option will flash highlight each component that includes dependencies to aid in debugging.

## Blackboxing

This category contains all the controlling options for the blackboxing feature. For more information on blackboxing, see Blackboxing Modules and Blackbox a Module.

### File Creation

- **Custom Folder:** Choose a destination folder for all generated source and compiled object files. Compiled files will appear in a sub-folder, sorted according to the compiler used to create them. If left blank, the default destination is the project folder.
- **Object (\*.obj or \*.o):** Use this setting to control whether or not the generated source code is compiled into a binary file. A corresponding sub-folder will be created for each compiler within the *source file destination folder* to house the corresponding binary.



- **Linking:** When enabled, a link to the generated file will be automatically specified in the target project settings (Fortran or Link tab). If an object file is created, only the object file will be added to the corresponding project setting.
- **Clear Existing Paths:** Use this option to help keep either the Additional Source files (\*.f, \*.for, \*.f90, \*.c, \*.cpp) or the Additional Library (\*.lib) and Object (\*.obj/\*.o) Files fields in the project settings from getting unruly. Unruliness can occur when creating many blackboxed modules in a single session. Select *Leading to the custom folder* to keep the field clear, excluding dependent paths not related to your Blackbox folder. Select *All* to clear all contents.

### New Component

- **Target Namespace:** Choose from a list of loaded projects in the workspace, a destination project for the new component and associated file links. Source or object file links will be appended to this project if the corresponding option is enabled. If set to local project, then the destination will be the current working project.
- **Generation:** This option controls to what extent the algorithm should go to in the creation of the component. Options are to create only a definition, or both a definition and first instance of the component.

- **Parameters:** Choose whether or not to include the parameter values of the source module instance when creating and instantiating the new component. If disabled, the new component instance will use the default parameter values.

## Dependencies

This area contains settings related to dependent files and folders:

### Fortran Compiler

- **Version:** Select the FORTRAN compiler to be used for compilation of projects. If you have more than one compiler installed, you may freely switch between compilers without having to restart PSCAD or reload the project (provided they are compatible of course). See the section entitled FORTRAN Compilers in the *Getting Started* brochure for a list of compatible FORTRAN compilers.
- **Number Format (Locale):** Setting this option to *Use English (USA) number format* can eliminate runtime errors where the default number format does not use periods as decimal points.

### Matlab

- **Version:** If you intend to make use of the MATLAB/Simulink interface, then choose which version of MATLAB you will be using in this field.
- **Configuration File:** Selects the MATLAB information file. This file includes configuration information about compatible versions.

**NOTE:** Enabling the MATLAB/ Simulink interface is also a project-specific option. See the section entitled Link for more details.

### Models

- **Master Library:** The master library is used to support core operations and provide model interfacing with the build in simulation.
- **User Libraries Folder:** The path entered here is automatically appended to files entered into the Additional Library (\*.lib) and Object (\*.obj/\*.o) Files input field in the *Project Settings* dialog.

### Resources

- **Local Help:** This option sets the path to the local help system included with your PSCAD software.
- **Online Help:** This option sets the web URL for the web-based help system.
- **Examples Folder:** Enter a default directory path to the folder containing the example projects included with the released software. By default, this path will be set to where the example cases placed during installation. This path is referred to when loading a case and selecting the *Examples* option. See Ribbon Control Bar for more details.
- **Custom Help Folder:** Enter a path to a specific folder that will be used to house all custom component help files (\*.html). If left blank, it will be assumed that the custom files will be located in the folder corresponding to the project where the component definition is defined.

## External Tools

The tools section is used to adjust properties related to external tools..

## Configuration Medic

- **Version:** Display only.
- **Folder:** Specifies the binary executable folder for the configuration medic tool. This is an optional add-on.

## Electromagnetic Transients (EMTDC)

- **Version:** Display only.
- **Folder:** This is the root source folder for all EMTDC binaries. Sub-folder referencing is done by compiler selection.

## Line Constants Program (LCP)

- **Version:** Display only.
- **Path:** Specifies the binary executable folder for the line constants program (LCP).
- **Maximum Concurrent Execution:** Transmission line and cable segments are solved in parallel in PSCAD. Projects containing large numbers of transmission segments can stress system resources. This option controls the maximum number of transmission segments that can be solved concurrently.

## Xoreax Grid Engine (XGE)

- **Version:** Display only.
- **Folder:** Specifies the binary executable folder.
- **Console Window:** Show/hide the output console window.
- **Build Monitor:** Agent activity can be shown in a console window on this machine. This indicates the total amount of available work power.

## ZSystems LiveWire (LW)

- **Folder:** Specifies the binary executable for the LiveWire post-analysis tool. This is an optional add-on.

## Certificate Licensing

### Prompt

- **Prompt on 'Log Out':** Display a warning message when logging out.

### Proxy Server (HTTP)

- **Mode:** Licensing can contact the server, using the default Internet Explorer proxy settings (if configured), or using a user specified proxy
- **Credentials:** Select Default if your proxy uses the same credentials as the Internet Explorer proxy (if used), None if no credentials are used, or Windows if your proxy uses your Windows login credentials.
- **Address:Port:** Enter the address:port of your network proxy if it is different than the Internet Explorer default proxy. Examples: <http://SomeNet.com:8080/>, <http://10.1.1.1:80/>

- **Expect 100 Continue:** This behavior is supported by most HTTP proxy servers. Disable this behaviour if you encounter (417) Expectation Failed warnings.

## Server

- **HTTP Address:** Enter the full HTTP address of the Central License Manager.
- **TCP Address:** Enter the full TCP address of the Central License Manager.

## Startup Behaviour

- **Licensing Service:** Intranet licensing has been used since PSCAD 4.0 (2001). Advanced licensing is a new license certificate-based service being introduced to provide portable and reliable license acquisition.

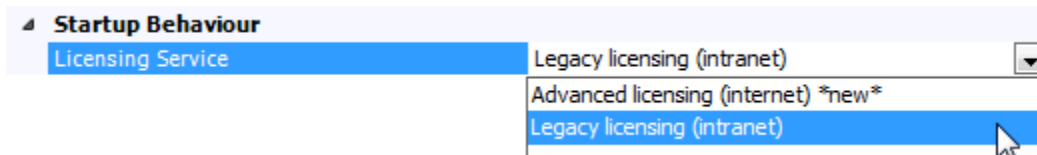
## Termination Behaviour

- **Certificate Behaviour:** On exit, the active certificate can be retained for further use on this machine, or returned to the server for use on another machine.

## System Settings

# Licensing

The *Licensing* section is devoted to licensing preferences and features. Depending on your selection of licensing type (i.e. *Certificate* or *Legacy*), the contents of this page will change. The control to allow to switch back and forth can be found in the application options dialog within the Advanced Licensing tab:

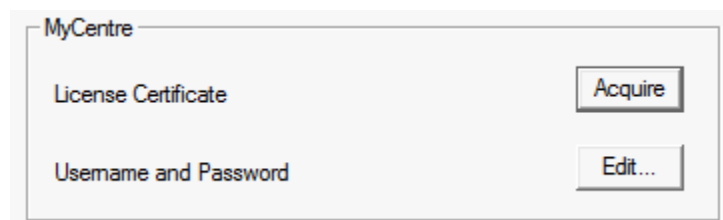


## Certificate Licensing

These options will be displayed only if *Licensing Service* | *Certificate Licensing* is selected in the Application Options.

## MyCentre

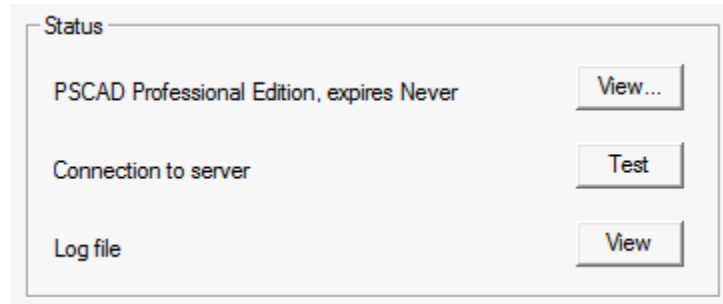
A certificate license is authorized through a web portal called MyCentre (<https://mycentre.hvdc.ca>).



If your MyCentre account is already setup, simply press the **Acquire** button to get your license. To setup a MyCentre account, see Certificate Licenses for details. Your MyCentre username and password may also be edited from this dialog.

## Status

This section provides status and testing facilities.

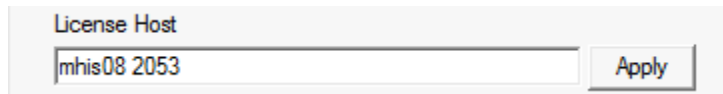


## Legacy Licensing

These options will be displayed only if *Licensing Service | Legacy Licensing* is selected in the Application Options.

### License Host

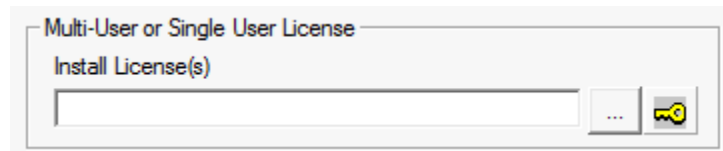
The hostname of the license manager server machine on your network.



This field is automatically configured during installation and should not need to be adjusted unless your license manager server hostname is changed. The additional '2053' number is always required (specifies the port number). This field is not used if you are using a trial license. For more details on how to change the license manager host directly from this dialog, see Standalone License Manager.

## Multi-User or Single User License

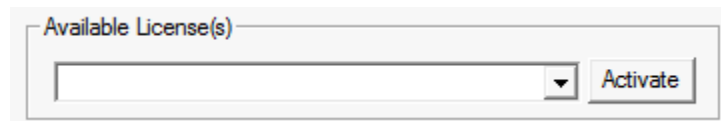
The multi-user or single user license area is used only when installing and activating these types of licenses.



See Standalone License Manager for more details.

## Available License(s)

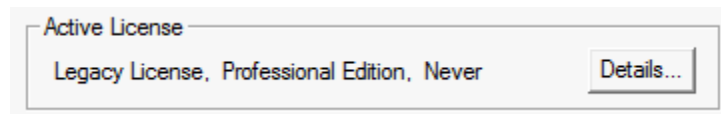
The Available License(s) area is used for switching between available licenses.



See Standalone License Manager for more details.

## Active License

The Active License area is used only when requesting and installing a trial license.



## Associations

The associations section is used to set-up file association settings. All file associations specified here, will enable the user to start-up external applications from within PSCAD, by using the File Reference component.

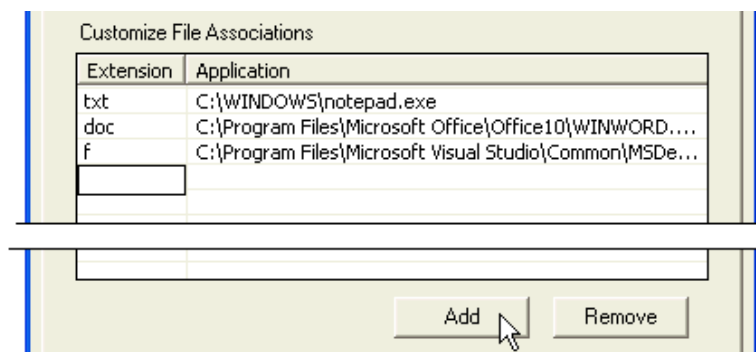
### Customize File Associations

File associations are entered in two parts: The file extension (with no period), and the path to the application executable file.

Extension	Application
txt	C:\WINDOWS\notepad.exe
doc	C:\Program Files\Microsoft Office\Office10\WINWORD....
f	C:\Program Files\Microsoft Visual Studio\Common\MSDe...

### Creating a New Association

To enter a new file association, press the **Add** button. The next available input box in the **Extension** column will appear as selected.



Enter the file extension (without the period) and press **Enter**. Either double left-click the box directly to the right of the new extension, or press the **Browse (...)** button to enter a path to the corresponding application executable file. The browse button will bring up a dialog window by which you can navigate to the file.

Extension	Application
txt	C:\WINDOWS\notepad.exe
doc	C:\Program Files\Microsoft Office\Office10\WINWORD....
f	C:\Program Files\Microsoft Visual Studio\Common\MSDe...
xls	

Browse Button

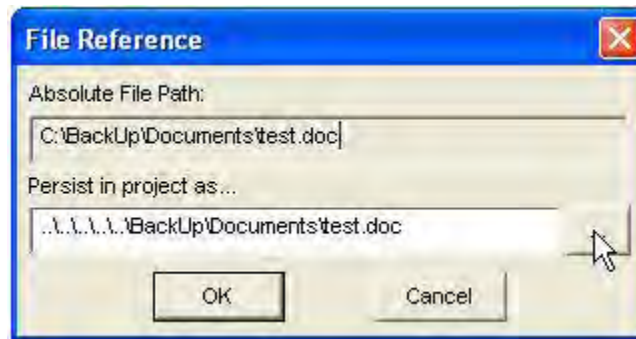
## Removing an Association

Simply select the association to be removed and then press the **Remove** button.

## Invoking an Associated External Application

To make use of the file associations entered here, the File Reference component must be linked to a file with a specified extension as follows:

1. Add a File Reference component by right-clicking over a blank part of the Circuit canvas and selecting **Add Component | File Reference**.
2. Either double-click on the component or right-click on it and select **Properties...** to bring up the component properties dialog.
3. Enter the path (relative to the project file) to the file either directly or by pressing the **Browse (...)** button.



That's all there is to it! The next time the File Reference component is double-clicked it will start the associated application and open the specified file.

## Workspace Options

## Projects

The projects section contains a variety of user preference settings.

### Auto-Save

- **Before building:** If set to *Save all changes*, case projects are automatically saved whenever they are compiled.

- **Save document every:** PSCAD will save all project documents loaded in the workspace at the intervals specified here.

## Display

- **Project file name is:** Select either *Hidden* or *Visible*. If *Visible* is selected, both the project namespace and filename will be displayed in the workspace window.

## Build

The build section is used to adjust compiler check settings and messages.

### Code Generation

- **Type Conversion:** Set to *No action* if you do not want to view signal type conversion warnings.
- **Unit System Converter:** Set to *No action* if you do not want to be notified when the unit system converter is disabled.
- **Import/Export Tag Matching:** Case sensitivity is important for the support of case sensitive languages, such as C and C++, when source code from project schematics.

### Compiler

- **Environment Variables:** Using private process-based environment settings can eliminate conflicts that occur when multiple FORTRAN compilers are installed. This setting should be set as *Private to process only* in order to avoid compiler setup issues.

### Electric Network

- **Maximum Electrical Subsystems:** This setting should be adjusted with caution. There is a trade-off between simulation efficiency and memory usage. If there are too many subsystems defined, unrecoverable out-of-memory errors may occur.

### IEC Compatibility

- **CIM 61970 Compatibility:** Enable this setting to ensure project conformity to the Common Information Model (CIM) 61970 standard. When enabled, certain compile errors may occur. For example, non 3-phase buses are not allowed in CIM.

### Wireless Radio Links

- **Maximum Radio Links:** Large numbers of radio links can slow overall performance. Increase this value if your workspace requires larger storage.

## Runtime

The runtime section is used to adjust compiler check settings and messages related to system memory and runtime related issues.

### General

- **Maximum Concurrent Execution:** Specify the maximum number of simulations that can be executed concurrently.



- **Communication Port Base Value:** This value defines the base range for opening communication ports to each simulation. This value can be changed whenever multiple instances of PSCAD are running on the same machine, avoiding address contention.
- **Storage Notification Level:** If the storage required for a simulation exceeds the selected value, a warning will be issued before the simulation is run.

## Console Messages

- **Maximum Duplicates:** During runtime, it is possible for messages to be sent to the output window from EMTDC every time step. Use this option to disable this from occurring. If set to None, PSCAD will only display the first message and ignore the rest. In some instances, it may be prudent to output duplicate messages, which is why this setting is available.

## Output Channel Checks

- **No output channels present:** PSCAD will issue a warning message upon project run indicating that there are no Output Channels (and hence no graphical feedback).
- **Excessive channels present:** Performs a sanity check to ensure that the user has not requested an excessive number of output channels. If the number of output channels is very large, the user should send channels on demand rather than all of them (otherwise, the simulation may perform very slowly). See the section entitled Runtime in Chapter 7 of this manual for more details.
- **Sample density is very high:** Performs a sanity check to ensure the user has not selected a sample count that is too big for the system display and storage. This can occur when the time step is very small and the run duration is long. For example, a one second run plotted at every microsecond will have 1 million data points per trace. These are upper bound to ensure that the plot and graphs continue to function with reasonable performance. See the section entitled Runtime in Chapter 7 of this manual for more on setting simulation time step and plot step.

## Output File Checks

- **Waveform Overwrite:** This option provides a warning dialog when a project is run, indicating that the project output file is about to be overwritten. The option to enable writing of an output file can be found under the Runtime tab in the *Project Settings* dialog.
- **Snapshot Overwrite:** This option provides a warning dialog when a project is run, indicating that the project snapshot file (or files) is about to be overwritten. The option to enable writing of a snapshot file can be found under the Runtime tab in the *Project Settings* dialog.

## Schematic

- **Prohibit Canvas Modification:** Enable this option to prevent the accidental modification on focus circuits during simulation.

## Project Settings

# Project Settings

Many features and settings related to simulation control in PSCAD are contained within the *Project Settings* dialog. Important parameters, such as total simulation time and time step, are included here along with more advanced, project specific PSCAD and EMTDC options.

See Editing Project Settings for information on accessing the *Project Settings* dialog.

The options available in the *Project Settings* dialog are divided into seven specific areas:

- **General:** Options such as revision tracking and other file related information.

- Runtime: Project related simulation options.
- Simulation: EMTDC related simulation options.
- Dynamics: PSCAD signal control and other options.
- Mapping: PSCAD electric network related options.
- Fortran: Fortran compiler and debug settings.
- Link: Fortran and MATLAB linking options.

## General

The properties contained within this section are related to the project file and version tracking.

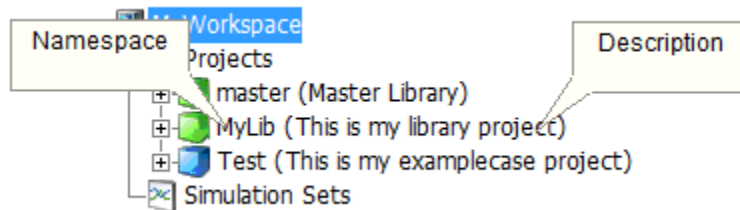
### Namespace

The *namespace* is a project attribute, through which all component definitions are linked to the project. The *namespace* may only be modified for library projects. In case projects, the *namespace* is kept in synch with the project file name, and can only be modified when the case project is saved as another filename.

Using a project attribute to ensure proper linking to component definitions ensures that these links are not dependent on the actual filename. See Definition Referencing for more details.

### Description

This field allows for the entry of a single-line description of the project. This description will be displayed beside the project filename in the workspace window. Do not use quotations (") or apostrophes (') in this field.



### Full Path

Full Path displays the path and filename information for the project file. This field is for display only.

### Relative Path

Relative path displays the project filename only. This field is for display only.

### Revision Tracking

These settings display particular information regarding the project file revision history.



## File Version

The PSCAD release version used to create the project.

## First Created

The date and time at which the project was first created.

## Last Modified

The date and time at which the project last modified.

## Author

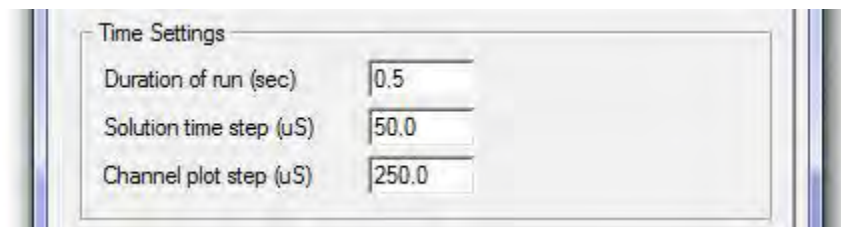
The name (userid) of the person who created and/or modified the project.

# Runtime

The properties contained within this section are the most commonly accessed project parameters. Note that most of the settings found here can also be accessed via the *Project* tab of the ribbon control bar.

## Time Settings

These settings are very important and are used quite often in every day simulation studies.



## Duration of Run (sec)

This is the total length of the simulation run, entered in seconds. If you start from time zero, this is the finish time of the run. If you start from a snapshot file (pre-initialized state), this is the length of run from the snapshot time.

## Solution Time Step (us)

This is the EMTDC simulation time step, entered in microseconds. The default is  $50 \mu\text{s}$ , which is an optimum step for most practical circuits. However, users should make sure that the time step selected is suitable for their simulation. This input sets the value of the EMTDC internal variable DELT.

## Channel Plot Step (us)

This is the time interval at which EMTDC sends data to PSCAD for plotting as well as writing data to output files. It is always an integer multiple of the EMTDC simulation time step. Usually a  $250\ \mu\text{s}$  plot step provides a reasonable resolution and speed.

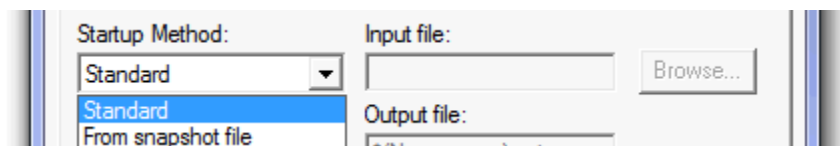
Smaller sampling intervals (higher sampling rate) can decrease the simulation speed considerably due to an excessive transfer of data from EMTDC to PSCAD (without adding much to the plot resolution). Users can experiment with this number for a given project. If the sampling interval is too large, the waveforms may appear 'choppy'. If you are debugging the case, it is a good practice to plot every point in the simulation; that is, plot sampling time as equal to EMTDC simulation time step.

A trap that even the most experienced engineers can readily fall into is the setting of plot step too broad with respect to the level and period of noise in the signal. If a signal is periodic at a frequency similar to the plot step interval, the perceived output may be quite different to the actual signal. As a basic rule: If you are puzzled by the results observed from a plotted simulation output, run the case with the plot step equal to the EMTDC time step and compare the results.

**NOTE:** The plot step can be modified during a run (or after starting from a snapshot). You can change the value from the *Project Settings* dialog or via the ribbon control bar.

## Start-up Method

There are two ways to start a simulation in PSCAD: The standard method (i.e. from time = 0.0 seconds) or from a snapshot file.



### Standard

The standard method to start an EMTDC simulation run is to simply start from an un-initialized state (i.e. from time  $t = 0.0$ ). This is how simulations are started most of the time.

### From Snapshot File

There may be instances when you would like to start your simulation from a pre-initialized state. Initial conditions are not available as direct entry into specific components, but it is possible to run a case to steady-state, and then take a snapshot at a specified instant during the run. All relevant network data will be saved to a snapshot file, from which you may start your case already pre-initialized.

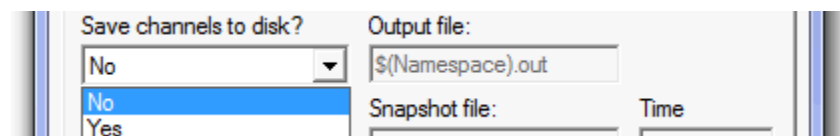
An input field is included directly beside this field called **Input File**. Enter a name for the snapshot file to be used here. See Starting from a Snapshot for more details on how to create and start from a snapshot file.

**NOTE:** When you start from a snapshot file, make sure that you have not altered the circuit from which the snapshot was taken, otherwise an error may occur.

## Save Channels to Disk?

You can save all output channel signals in your project to a file for post processing. Output files are saved in standard ASCII format, and all data is stored in columnar format in steps of time according to the set plot step.

An input field is included directly beside this field called **Output File**. Enter a name for the file here. The output file will by default be located in the PSCAD Temporary Folder.

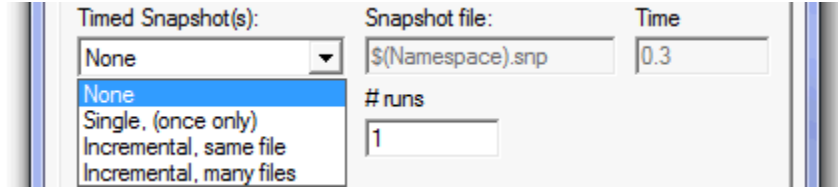


See Saving Output to File for more details.

## Timed Snapshot

There are two ways to utilize a snapshot file: Single and incremental snapshots.

An input field is included directly beside this field called **Snapshot File**. Enter a name for the snapshot file to be created here. Another input field is included called **Time**. Enter the time in seconds at which the snapshot is to be taken.



See Taking a Snapshot and Starting from a Snapshot for more details.

### Single (Once Only)

A single snapshot will be taken at the time specified in the **Time** field.

### Incremental (Same File)

An incremental snapshot in the same file will take the first snapshot at the specified time and subsequent snapshots at equal intervals equal to this time. The snapshot file will be over-written every time, so you will end-up with a single file taken at the last time interval.

### Incremental (Many Files)

If you would like to save all snapshot files, select this option. You can save up to 10 distinct snapshots. If the number of files exceeds 10, the names will be reused starting from the beginning.

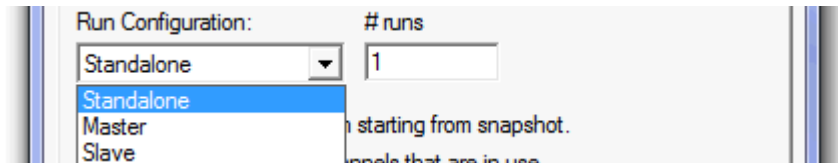
The file naming convention is:

base\_name\_##.snp

You are required to provide only the 'base\_name' (in the **Snapshot File** field). The extension will be added automatically.

## Run Configuration

There are currently two methods for performing multiple runs in PSCAD – this one is the more basic of the two. This method is used in conjunction with the Current Run Number and Total Number of Multiple Runs components, which are available in the master library.

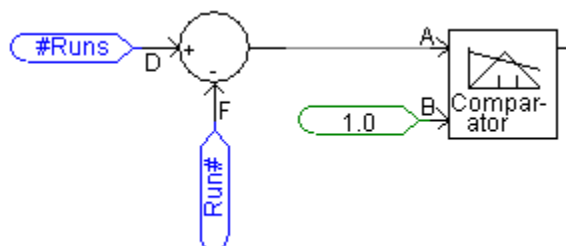


Another field is included directly beside the **Multiple Run** field called **Output File**. Enter a name for the multiple run output file here. Another input field is included called # runs. Enter the total number of runs here (this value is used to set the Total Number of Multiple Runs component).

**NOTE:** The other multiple run method involves the Multiple Run component, which is available in the master library. DO NOT enable this multiple run feature when using the Multiple Run component.

#### EXAMPLE 7-1:

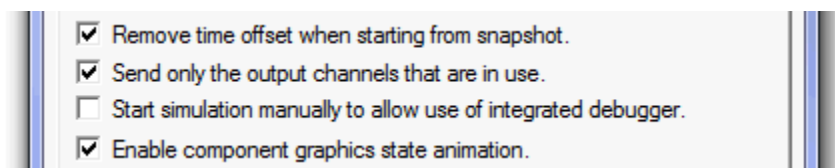
Consider the simple comparator below. The **Multiple Run** parameter is enabled in the Project Settings dialog and set to 10 runs.



In this case, the comparator output is set to go *low* (0) when input A is less than or equal to 1.0 (i.e. the last two runs).

## Miscellaneous

The remaining Runtime parameters are outlined below:



### Remove Time Offset When Starting From a Snapshot

This parameter is used in correlation with starting the simulation from a snapshot file. Enabling this parameter will force the initial start time on all plots to display a 0.0 start time - regardless of what time the snapshot was taken. If disabled, the initial start time will be displayed as the time at which the snapshot file was created.

#### EXAMPLE 7-2:

A user runs a case project to steady state, and then takes a snapshot at 0.5 seconds. The user then re-starts the simulation from the snapshot file created, and sets the **Duration of Run** parameter to 0.05 s.

If **Remove Time Offset When Starting from a Snapshot** is enabled, the simulation run will be displayed from 0.0 to 0.5 seconds. If disabled, from 0.5 to 0.55 seconds.

It is important to note here that this option changes displayed start time only. That is, if you are controlling breakers or faults for example, you must use the actual time. In Example 7-2 above, if the user wanted to open a breaker 0.01 seconds after starting from the snapshot, the breaker logic must indicate 0.51 seconds (not 0.01 seconds).

## Send Only the Output Channels that are In Use

Selecting this option will turn off all output channels that are not being plotted in graphs, or monitored in meters. Enabling this option has the potential to greatly reduce the amount of storage for the simulation, as well as a slight simulation speed improvement.

**NOTE:** Unused output channel data will still be written to EMTDC output files if Save Channels to Disk? is enabled.

## Start Simulation Manually to Allow Use of an Integrated Debugger

This option allows you to start a PSCAD run that can be connected with a manually started EMTDC case. See Deployment of an Integrated Debugger in Chapter 11 of this manual for more details.

## Enable Component Graphics State Animation

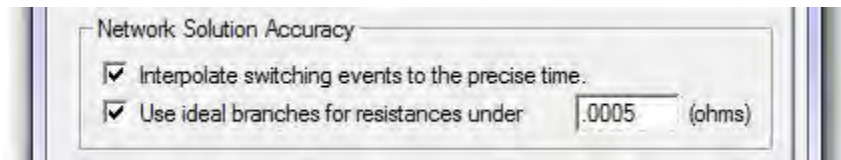
This option will enable/disable Active Graphics in all components. Since the active graphics algorithm is processor intensive, disabling it will help speed up those cases that contain a lot of animation.

# Simulation

The properties contained within the *Simulation* section provide some control over EMTDC operation.

## Network Solution Accuracy

The following input parameters can affect solution speed.



### Interpolate Switching Events to the Precise Time

To account for inter-time step switching of switching devices (i.e. those components whose electrical conductance change during a run), the EMTDC Electric Network Solution uses a linear interpolation algorithm to solve at the exact switching instant. This is the default behavior of EMTDC and is essential for the accurate simulation of frequently switching devices, (such as FACTS models).

### Use Ideal Branches for Resistances Under

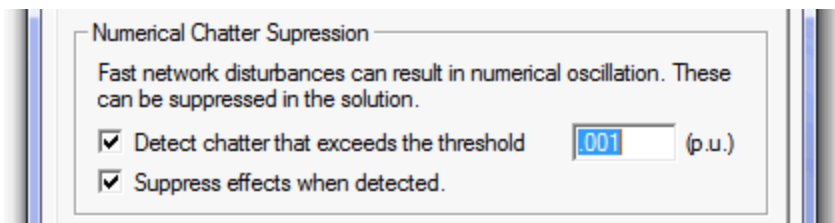
The Ideal Branch algorithm allows for zero resistances and true infinite bus voltage sources in EMTDC.

With this option enabled, you can create an infinite bus by either entering a value less than the threshold (set in the threshold field directly to the right of this check box) or  $0.0 \Omega$  for the source resistance. Similarly, for a zero resistance branch, enter a value less than the threshold or  $0.0 \Omega$  for the ON resistance of a diode, close resistance of a breaker, etc.

The default threshold value is preset to  $0.0005 \Omega$ . As this algorithm involves extra computations and so a non-zero value greater than this threshold is recommended, unless ideal like results are paramount.

## Numerical Chatter Suppression

The following input parameters are related to the detection and removal of numerical oscillations called chatter.



## Detect Chatter that Exceeds the Threshold

Chatter is a numerical oscillation phenomenon inherent to the trapezoidal integration method used in EMTDC, and is usually caused by sudden network disturbances (either voltage or current). EMTDC continuously looks for chatter and removes it if required.

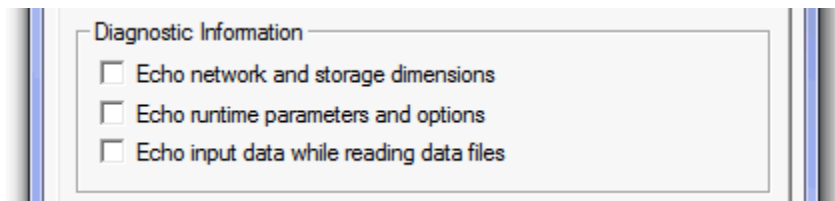
An input field is included directly beside this option. Enter a threshold value for chatter detection, below which chatter will be ignored (the default is  $0.001 pu$ ).

## Suppress Effects When Detected

When chatter is detected, a chatter suppression procedure is invoked if this option is enabled.

## Diagnostic Information

The following input parameters are important during the debugging process of your simulation project and are recommended for more advanced users. The information generated by these settings will appear under the Non-Standard Messages branch in the Runtime Messages Pane.



### Echo Network and Storage Dimensions

Prints network and storage array dimensions.

### Echo Runtime Parameters and Options

Prints runtime related options.

### Echo Input Data While Reading Data Files

Prints all data read from the data and map files.

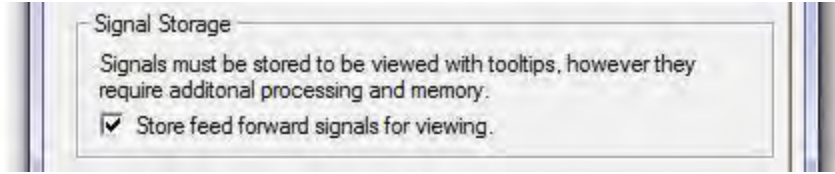
## Dynamics

The properties contained within the *Dynamics* section are related to the EMTDC system dynamics. These are explained below.

### Signal Storage

The input parameters involved here are outlined below.







## Store Feed Forward Signals for Viewing

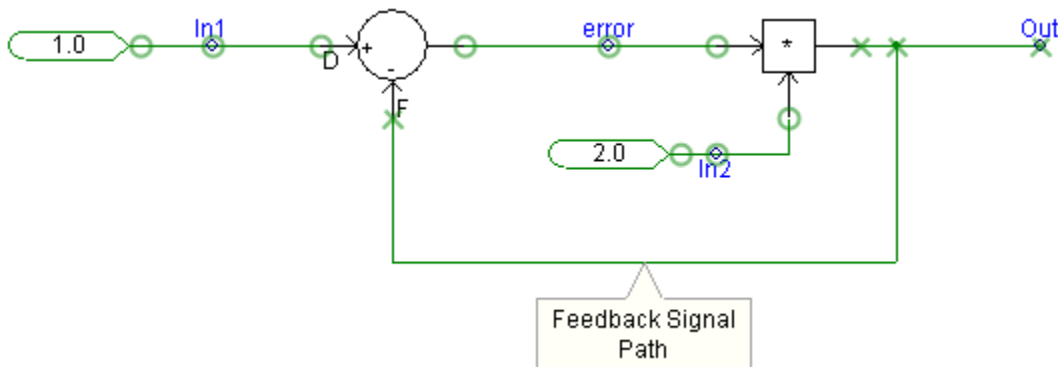
When this control is enabled, all declared data signal variables (be they feed-forward or feed-back signals), will be transferred to and from their designated EMTDC storage array each time step. If this control is disabled, only those declared variables that must be transferred to storage (such as feed-back signals) are considered; all other variables are deemed temporary and are discarded at the end of each time step.

**NOTE:** It is important to note that tool tips (flybys extract the value of the signal being monitored from storage. Therefore, if this control is disabled, all feed-forward signal values will not appear in the tool tips!

In smaller projects, toggling this control will have very little effect on the simulation speed. As projects become larger and contain many control signals, this feature may help to speed things up.

### EXAMPLE 7-3:

In the simple control system shown below, there will be a total of four, type REAL variables declared in the project Fortran file when this project is compiled: *In1*, *error*, *Out* and *In2*. Three are feed-forward variables (indicated by the  symbol), and one is a feedback variable (indicated by the  symbol).



**NOTE:** Users can display these symbols at any time by enabling the **Show Signal Locations** parameter in the Module Settings dialog. For a more detailed description of these symbols, see Show Signal Locations in Chapter 11 of this manual.

If *Store Feed Forward Signals for Viewing* is **enabled**, all four of the above variables will be stored every time step, as indicated in the following excerpt from the project FORTRAN file:

```
!-----
SUBROUTINE DSDyn ( )
.
.
.
```

```

!-----
! Transfers from storage arrays
!-----

    In2      = STOF(ISTOF + 1)
    In1      = STOF(ISTOF + 2)
    error    = STOF(ISTOF + 3)
    Out      = STOF(ISTOF + 4)

.
.
.
!-----
! Feedbacks and transfers to storage
!-----

    STOF(ISTOF + 1) = In2
    STOF(ISTOF + 2) = In1
    STOF(ISTOF + 3) = error
    STOF(ISTOF + 4) = Out

```

If *Store Feed Forward Signals for Viewing* is **disabled**, only the feedback variable Out will be written to storage every time step as indicated in the following excerpt from the project FORTRAN file:

```

!=====
SUBROUTINE DSDyn()
.
.
.
!-----
! Transfers from storage arrays
!-----

    Out      = STOF(ISTOF + 4)

```

```

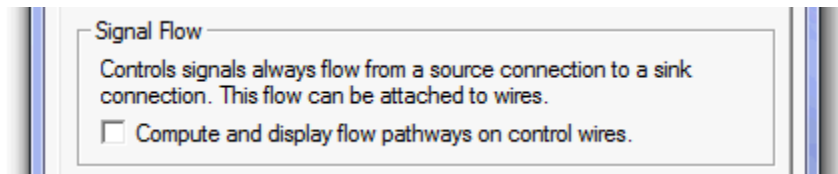
.
.
.
! -----
! Feedbacks and transfers to storage
! -----

STOF(ISTOF + 4) = Out

```

## Signal Flow

The input parameters involved here are outlined below.

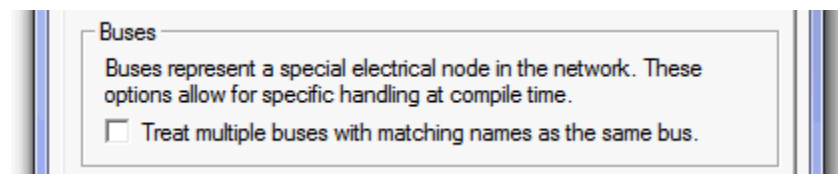


### Compute and Display Flow Pathways on Control Wires

Select this option to show signal flow direction on control signal Wires (i.e. Wires carrying REAL, INTEGER or LOGICAL data types). See Control Signal Pathways for more details on this.

## Buses

The input parameters involved here are outlined below.

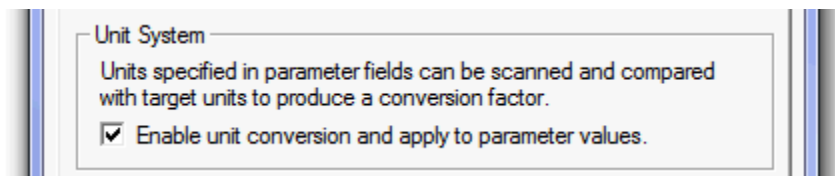


### Treat Multiple Buses with Matching Names as the Same Bus

Select this option to treat multiple buses with the same name as the same bus. That is graphically separated bus points with the same name will represent the same EMTDC node points in the electrical grid. See the Bus component for more.

## Unit System

The input parameters involved here are illustrated below:



### Enable Unit Conversion and Apply to Parameter Values

Select this option to enable the Unit System.

**NOTE:** Enabling the Unit System may cause compilation failures in previously created case projects. This is normally due to input parameter units in existing components that are not recognized by the Unit System compiler. If compilation failure does indeed occur:

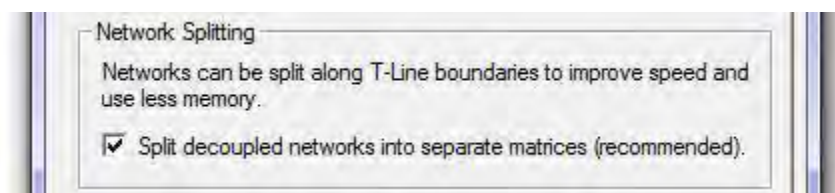
1. Consult the Output Window for error messages: Point to the error source by Locating the Problem Source.
2. View the parameters of the component.
3. Scan through the *Value* column and look for a #NaN (Not a Number) symbol. This usually indicates that the entered unit does not match the Target Unit defined for this input parameter.

## Mapping

The properties contained within the *Mapping* section are related to EMTDC network solution conductance matrix optimization.

### Network Splitting

The input parameters involved here are outlined below.



### Split Decoupled Networks into Separate Matrices

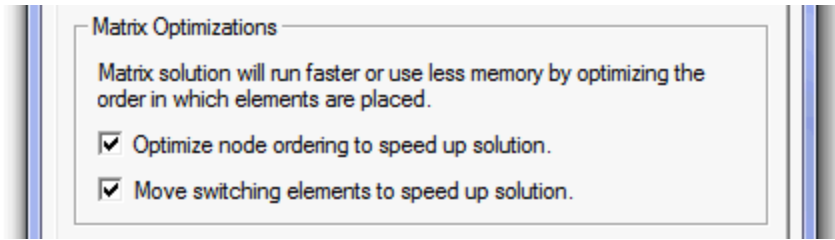
Matrix solution methods can be intensive, requiring a large amount of computing power, especially if frequently switching branches are involved. If this option is enabled, larger networks will be reduced into smaller sub-networks, or subsystems.

Once the main electric network has been split into subsystems, each subsystem can be solved independently. Say for example a large FACTS device (with many frequently switched branches) is located within its own subsystem, but part of a much larger network. By splitting the main network into subsystems, only the local subsystem, harbouring the FACTS device, need be resolved when a switching event occurs. The speed benefits here can be tremendous.

See Subsystems in Electric Networks in the EMTDC manual for more details.

### Matrix Optimizations

The input parameters involved here are outlined below.



## Optimize Node Ordering to Speed Up Solution

Optimize Node Ordering is a PSCAD based algorithm, which re-numbers nodes in the EMTDC electric network conductance matrix, so as to optimize solution speed. The electric network conductance matrix is optimized using a Tinney algorithm to exploit matrix sparsity. For more information on matrix optimization, please see [7], [8], [9] and [10].

## Move Switching Devices to Speed Up Solution

Frequently switching branches are identified and re-ordered, so as to optimize conductance matrix re-triangularization following a change in branch conductance (a switch). The switch-ordering algorithm splits nodes into two types:

1. Nodes not attached to switching elements or nodes that have switching elements connected, but do not switch frequently (i.e. breakers and faults).
2. Nodes that have switching elements connected that switch frequently (i.e. thyristor, GTO, IGBT, diode, arrester, Variable RLC, etc.).

**NOTE:** Electrical connections in user-defined components are set as node type 2 by changing the electrical connection port type to *Switched*.

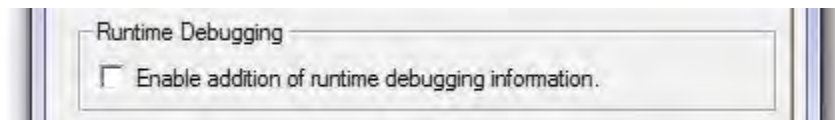
The switch-ordering algorithm moves the frequently switched nodes (i.e. type 2) to the bottom of the conductance matrix. The benefits to EMTDC solution speed are proportional to the number of nodes and the number of switching branches in a given electric network. For more information on matrix optimization, please see references [7], [8], [9] and [10].

## Fortran

The parameters contained within the FORTRAN section are used for the control of compiler based error and warning messages. Also, any additional source files, required for the compilation of the project, are indicated here as well.

## Runtime Debugging

The input parameters involved here are outlined below.



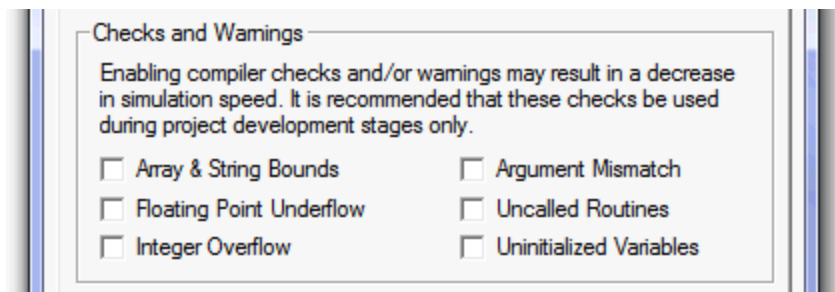
## Enable Addition of Runtime Debugging Information

This option will add some additional information to the build, so as to allow for the effective use of a Fortran debugger. If you have selected this option, you should also select all of the checks available in the Checks section (see below).

When a program crashes on a PC, the operating system brings up a dialog asking if you would like to debug the case. If you select Yes and if you had this option selected, the FORTRAN debugger can load the source file and point to the line of code that is causing the crash.

## Checks

The input parameters involved here are outlined below.



## Array & String Bounds

With this option selected, the program will stop if you are accessing an invalid array index. For example, if the array  $X$  is declared to be of dimension  $10$ , and you have a line of source code that has  $X(J)$ . If  $J$  ever goes greater than  $10$ , the program will stop with a proper message. If this option is de-selected, the program will continue execution and eventually may crash, adding difficulty to tracing the cause.

This option should be used when testing your new models. It will slow down the simulation and the speed penalty will vary from machine to machine – you can disable this option if speed is a concern. Please read your FORTRAN compiler documentation you are using for more details.

## Floating Point Underflow

This option is useful for debugging but has a speed penalty that is again architecture dependent. Please read your Fortran compiler documentation for more details.

## Integer Overflow

This option is useful for debugging but has a speed penalty that is again architecture dependent. Please read your Fortran compiler documentation for more details.

## Argument Mismatch

The Fortran compiler issues a warning if the argument type (REAL, INTEGER, etc.) of the CALL statement does not match with the subroutine statement (this is applicable to functions as well).

Do not ignore this warning as it may have the potential to create unpredictable and hard to trace results.

## Uncalled Routines

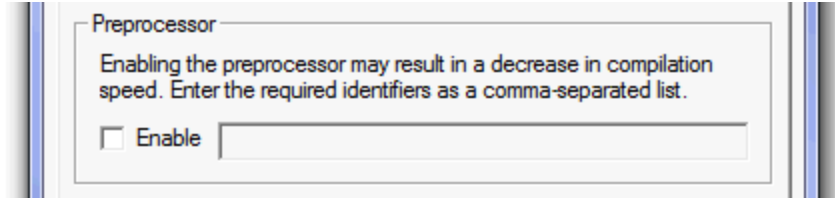
The Fortran compiler issues a warning if a routine is not called in the program.

## Uninitialized Variables

A warning is issued if a variable is used before it is assigned a value. Normally, the cause of this is a typographical mistake in the source code.

## Preprocessor

This option provides the ability to enable/disable the Fortran preprocessor. Disabled by default, this option is supported on both the Intel and GFortran compilers. An additional text field is also provided to allow the user to define a comma-separated list of preprocessor identifiers.



## Additional Source files (\*.f, \*.for, \*.f90, \*.c, \*.cpp)

This input field allows you to link one or more external source code files, so as to be included during compilation of the associated project. An external source code file could contain one or more subroutines, which must be called from within the FORTRAN segment of the component that uses it.

Files may be referenced in this field either by an absolute, or relative path specification. For example, a file referenced with an absolute path may appear as follows:



If only the filename is entered, PSCAD will assume that the source file is located in the same directory as the project file itself, and will append this path to the filename. You may also use standard directory navigational features when working with relative paths. For example, if a source file called *test.f* is located in the directory directly above the project file, then the file entry would appear as follows:

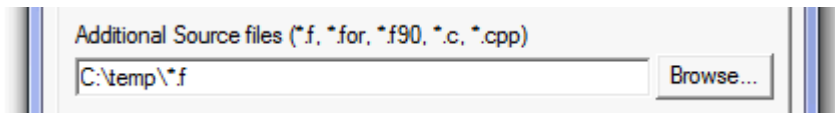


When entering multiple source files, each entry must be separated either by a blank space, a comma or a semicolon:



## Wildcard Character

The wildcard character (i.e. \*) is also supported. For example, entering the following:



Will ensure all \*.f files in the folder *temp*, will be included in the project build and make.

## Supported File Types

This input field will only accept the following source file types:

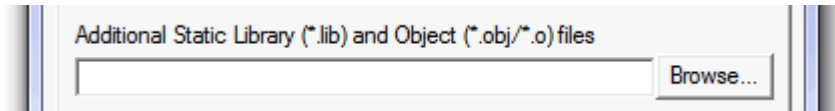
- **Fortran:** \*.f, \*.f90, \*.for
- **C:** \*.c, \*.cpp

# Link

The parameters contained within the *Links* section are used for the linking of pre-compiled libraries (including MATLAB related libraries) and object files.

## Additional Library (\*.lib) and Object (\*.obj/\*.o) Files

This input field allows you to specify external static library (\*.lib) and object (\*.obj) files, which must be linked to the project before compilation. Object and library files would normally be used if you have a lot of custom models associated with user written subroutines, where it would be more efficient to create a library or object file rather than maintain a collection of Fortran source files.



Object and library files provide a way to avoid the compilation of user written subroutines every time they are used in different projects. They are also useful for sharing custom models with others, where you do not want to share the source code itself.

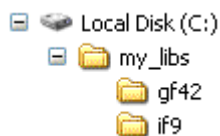
## User Libraries Folder Method

A compiler specific folder convention exists for the purpose of specifying compiler specific object and library files. Object and library files can be compiled with one or more different Fortran compilers, and then placed in a named sub-folder within the specified **User Libraries Folder**. Then, the user may freely switch between compilers without the need to re-specify the files.

The user must manually add these sub-folders to the main folder specified by the **User Libraries Folder** option (in the Dependencies category of the *Application Options* dialog). Each sub-folder represents a specific Fortran compiler, and the same library or object file (compiled by the corresponding compiler) may be placed. Since there are different types of Fortran compilers that may be used with PSCAD, there can be different sub-folders specified. These must be named as follows:

- gf42 (GNU GFortran 95 compiler)
- cf6 (Compaq Fortran 90 compiler)
- if9 (Intel Visual Fortran compiler versions 9, 10 & 11)
- if12 (Intel Visual Fortran compiler version 12, 13 & 14)
- if15 (Intel Visual Fortran compiler version 15 (64-bit))
- if15\_x86 (Intel Visual Fortran compiler version 15 (32-bit))

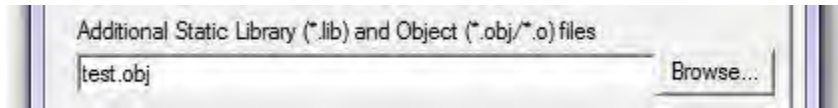
For example, if a user specifies *C:\my\_libs* as the **User Libraries Folder**, and intends to use either the *GFortran* compiler or the *Intel Visual Fortran 9* compiler, then two sub-folders should be added to 'my\_libs' as shown below:



Any library or object files required should be created by using each compiler, and then added to the corresponding sub-folder.

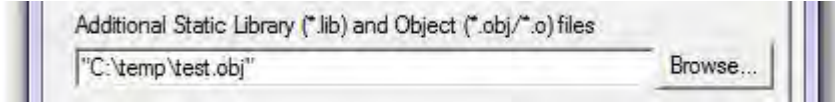
If only the filename is entered, PSCAD will assume that the library or object file is located in the specified sub-folder in the **User Libraries Folder**. For example, if a user specifies *C:\my\_libs* as the User Libraries Folder and is using the *Intel Visual Fortran* compiler, entering *test.obj* as shown below would indicate that the specified file is located at *C:\my\_libs\if9\test.obj*.





## Absolute Path Method

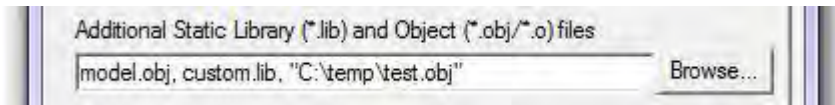
Files may also be referenced by either an absolute or relative path specification. If this method is chosen, then the folder specified in the User Libraries Folder will be overridden. For example, a file referenced with an absolute path may appear as follows:



**NOTE:** Although it is not a necessity, the absolute path should appear within quotes. This will ensure that if any spaces exist in a path directory or the filename itself, it will be parsed properly.

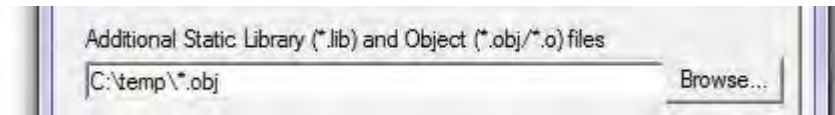
## Entering Multiple Files

When entering multiple files into this field, they must be separated with a space, a comma or a semicolon. Both *User Libraries Folder* and *Absolute Path* methods may be used simultaneously:



## Wildcard Character

The wildcard character (i.e. \*) is also supported. For example, entering the following:



Will ensure all \*.obj files in the folder temp, will be included in the project build and make.

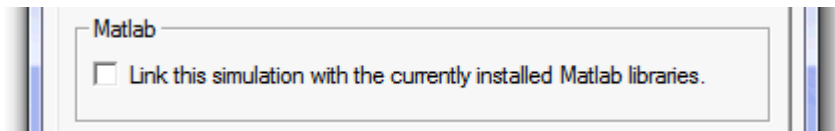
## Supported File Types

This input field will only accept the following compiled file types:

- \*.o, \*.obj, \*.lib

## MATLAB

The input parameters involved here are outlined below.



## Link This Simulation with the Currently Installed MATLAB Libraries

Select this option if you intend to utilize the MATLAB®/Simulink® interface with this project. Note that this option is disabled unless you have specified a MATLAB® version in the Dependencies category of the *Application Options* dialog.



## Chapter 4

# The Application Environment

The term *Application Environment* refers not only to how PSCAD is organized visually, but also to naming conventions, utilities and other features that facilitate its use. A lot of effort is continuously expended on the work environment –the primary goals being the perseverance of both the look and feel of previous versions, as well as the enhancement of existing features and the timely addition of new ones.

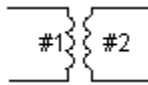
This chapter discusses a wide range of introductory topics, from simple terms and definitions to a general overview of the most important elements that make up the environment. The tutorial section My First Simulation is a must read for all users new to PSCAD.

## Terminology

The following comprises the most popular terms used when discussing the PSCAD environment. It is important to become familiar with this terminology, as it will help greatly in the comprehension of topics discussed hereafter.

### Components and Modules

A component (sometimes referred to simply as a block) is a graphical representation of a device model, and is the basic building block of all circuits created in PSCAD. Components are usually designed to perform a specific function, and can exist as either electrical, control, documentary or simply decorative in type.



Single-Phase Transformer component in PSCAD

Components usually possess input and output connection ports and can be pieced together with other components to form larger system models. The user may interface directly to the model through input parameters.

Modules (sometimes referred to as *sub-pages* or *page components*) are a special type of component, where the functionality of the component model is defined by constructing a circuit, using a combination of other components. Modules possess their own canvas, as opposed to a hard-coded script interface in regular components. Modules can even contain other modules within their canvas, thus providing a hierarchical modeling capability.

All components and modules possess a single definition, from which many instances may be created.

### Definitions

A definition is the underlying 'blueprint' of a component or module, and is where all design aspects of the associated model are defined. This may include graphical appearance, port connections, input parameters and model code and/or circuit layout. Only one definition can exist for every unique component or module.

Definitions are normally stored in library projects, and are a basis from which to create multiple instances (or copies) of a component or module to be used in any project loaded in the workspace.

### Instances

An instance is a graphical 'copy' of the definition, and is normally what is seen and manipulated by the user. An instance is not exactly a copy; each instance is its own entity, and may have different input parameter settings, or even appear graphically different from other instances.

Since all instances are based on a single definition, any design changes to a definition will affect all instances.

## Projects

Everything involved in a particular simulation (except output files) is harboured within a single file called a project. Projects can contain stored definitions, on-line plots and controls, and of course the schematically constructed system itself. There are two types of projects in PSCAD: Library and case projects.

### Case

Case projects (or simply 'cases') are where most work is performed in PSCAD. In addition to performing the functions of a library project, cases may be compiled, built and run. Simulated results can be viewed directly within the project through on-line meters and/or plots. Case projects are saved with the file extension \*.pscx.

### Library

Library projects are used primarily to store definitions. Instances of definitions stored in a library, can be used within any case project. Library projects are saved with the file extension \*.pslx.

### Namespace

The namespace is a project attribute, used to provide a stable source name for functions such as definition referencing. The namespace is separate from the project filename, which can be modified outside of the application (in File Explorer for example).

Namespace and filename may be different in library projects only. In case projects, the namespace and filename are kept synchronized in order to avoid confusion. For example, if you save a case project as another name, the namespace attribute will be modified as well.

The namespace is what is displayed in the workspace projects branch.

## Workspace

The workspace is the central operating hub within the PSCAD environment. It not only provides an overview of all projects currently loaded, but also organizes simulation sets, data files, signals, controls, transmission line and cable objects, display devices, etc. within an easily navigable environment.

Although only a single workspace can be present in the application, workspaces may be exchanged at any time. See Loading a Workspace for details.

## Tutorial: My First Simulation

### My First Simulation

This tutorial is designed to give new users a 'jump start' in learning how to use PSCAD, as well as to provide a chance to try out a PSCAD simulation before proceeding further in this chapter.

In this tutorial you will learn:

- How to load a case project
- How to run a simulation
- How to print

If you can successfully run the example project at the end of this chapter, it will also mean that your installation was successful. So let's begin!

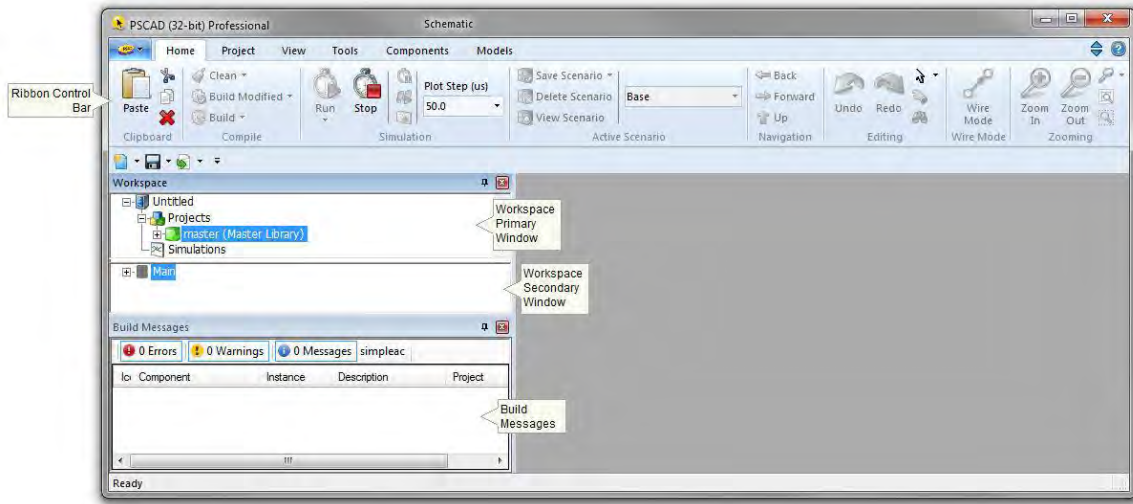
## Starting PSCAD

If PSCAD is installed locally on your machine, go to **Start | All Programs | PSCAD | PSCADX4 | PSCAD <Version>** in the Windows Start menu to start it.

If PSCAD is installed through MyUpdater:

1. Go to **Start | All Programs | PSCAD | MyUpdater** in the Windows Start menu to launch MyUpdater.
2. Log in to MyUpdater.
3. Select the **Install** or **Update** link associated with the software, if applicable.
4. Select **Run** to launch PSCAD.

You will see the main PSCAD environment as shown below.



Most functions available in the PSCAD application are represented by a button in the ribbon control bar.

## Ribbon Control Bar and Schematic Tabs

The ribbon control bars and schematic tabs are essential elements to consider when using the PSCAD application.

### Ribbon Control Bar

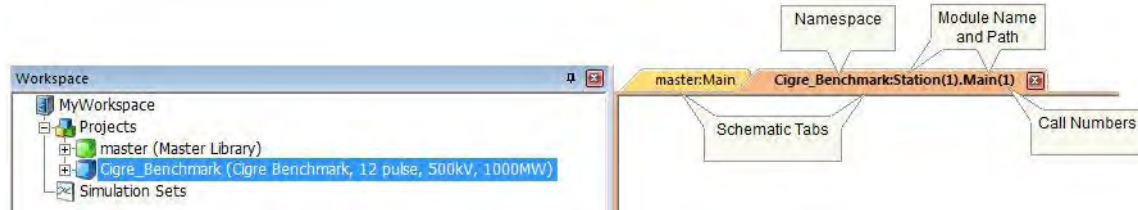
The area directly under the application title bar is called the ribbon control bar. Most functions available in the PSCAD application can be found here.



After you become more familiar with the program, you may also begin to use the many keyboard shortcuts available. See Keyboard Shortcuts for more details.

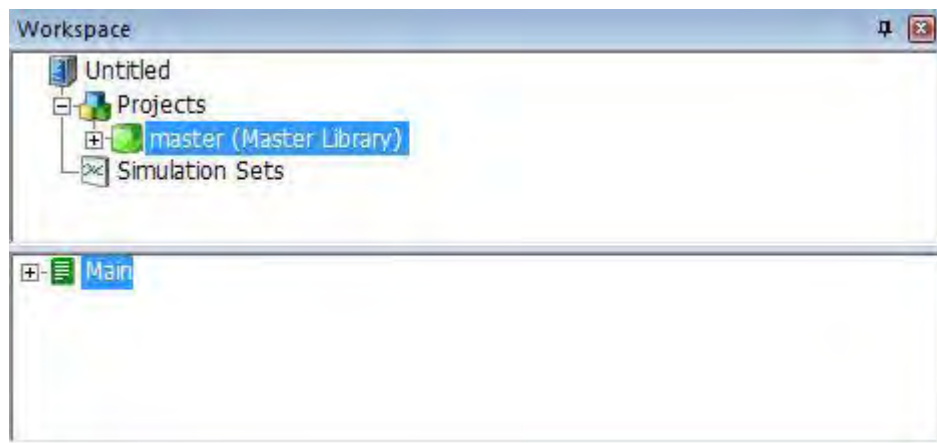
## Schematic Tabs

Each project loaded in the workspace will be represented by a schematic tab. The tab itself displays the project name, along with the name, path and call number of the module whose canvas is currently being viewed. A call number of -1 indicates that the current view is a module definition (i.e. non-white schematic canvas background).



## Workspace and Messages Windows

In the top left-hand corner of the environment, you should see a docked window referred to as the Workspace window (it has the master library and perhaps other projects displayed within it). If it is not visible, go to the ribbon control bar, click on **View** tab and then select **Workspace** in the **Panes** drop list button.



The workspace gives you an overall view of any library and/or case projects loaded. You can use it to perform a wide variety of activities, such as navigation or accessing files.

Directly below either the workspace or the definition editor, is another docked window referred to as the Build Messages pane. If it is not visible, go to the ribbon control bar, click on the **View** tab and then select **Messages** in the **Panes** drop list button.

All the navigable status (informational) warning and error messages, involved in both *Build* and *Runtime* procedures, are logged in this window – so, it is recommended to keep this window visible at all times.

## Opening a Case Project

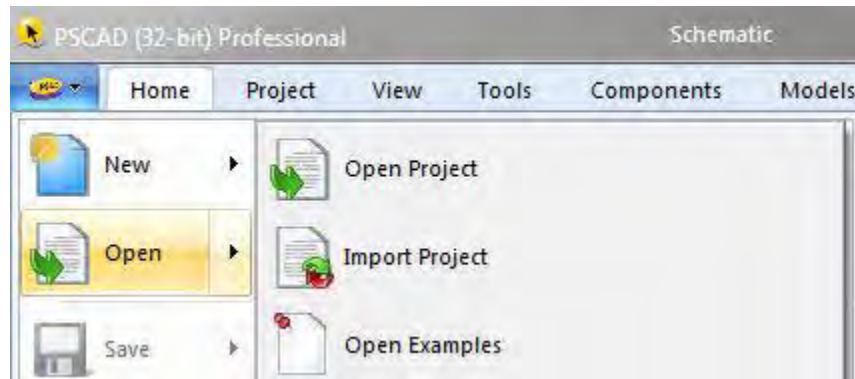
We will start with the most simple of example cases for this tutorial. This exercise will help us to ensure that both PSCAD, and any FORTRAN compilers being used, are installed correctly. We will learn to create a case from scratch in the tutorial entitled Creating a New Project.

To load an existing case project, perform one of the following operations:

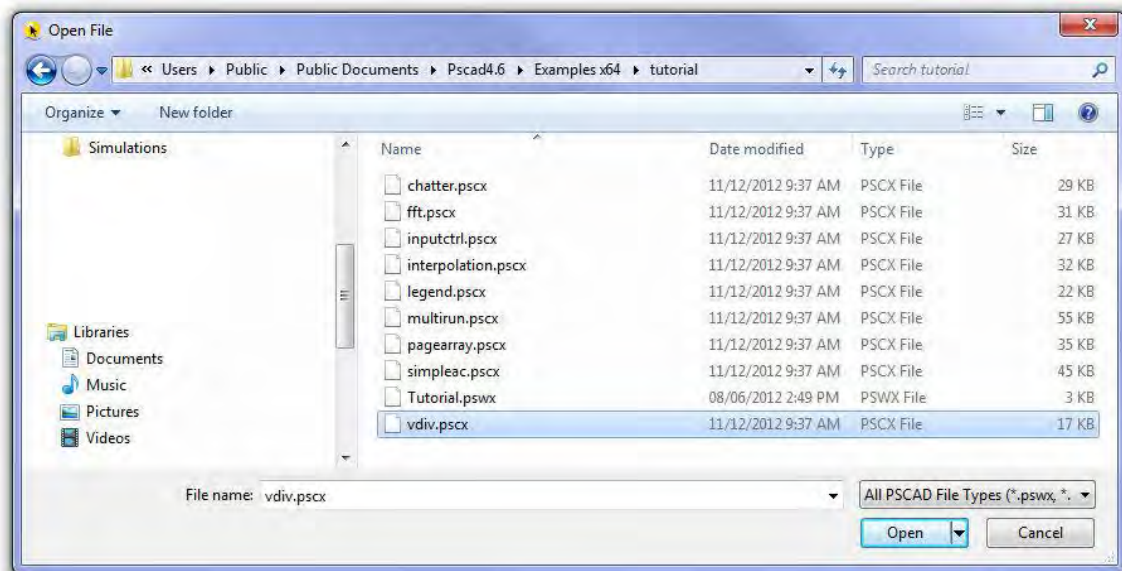
- Click the **Open Project** button in the quick access bar:



- Press the **Ctrl + o** keyboard shortcut.
- Click the PSCAD tab in the ribbon control bar and select **Open** (performs **Open | Open Project** by default):



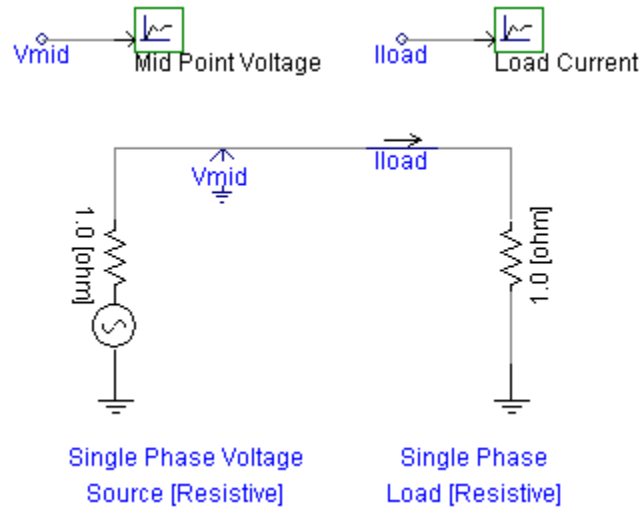
You should see the **Open File** dialog appear on your screen. By default, the selected file type is **Cases and Libraries (\*.pscx, .pslx)** at the bottom of the dialog.



Navigate to the *tutorial* directory inside your example project folder (C:\Users\Public\Public Documents\PSCAD[version]\Examples). Click on the *vdiv.pscx* file and then click on the **Open** button to load this project.

**NOTE:** You can navigate directly to the examples folder by clicking the PSCAD tab in the ribbon control bar and selecting **Open | Open Examples**.

The workspace window will now list an additional project entitled *vdiv* (*Single Phase Voltage Divider*) directly under the master library project. The main schematic canvas for the project will open automatically in the Schematic window tab. You should see the assembled voltage divider circuit as shown below, which is located at the top left corner of the main page in the project that you just opened. The plots are situated directly to the right of the circuit.



The circuit consists of a single-phase resistive voltage source connected to a resistive load. Since the magnitude of the source resistance ( $1 \Omega$ ) and the load resistance are the same, the voltage at the load terminal is half that of the voltage behind the source resistance. This voltage is measured using a voltmeter called *Vmid* connected to the node between the source and the load. The current in the circuit should be:

$$I_{load} = \frac{E}{R_S + R_L}$$

The plot and graphs will contain the values of the voltage at the mid-point of the circuit, and the current flowing through the circuit when the project simulation is run.

## Running a Simulation

Before we run the simulation we will do a simple calculation to find out what load current and mid-point voltage we should be expecting. Double-click on the source component to open and view its parameters – note that the source voltage magnitude is  $70.71 \text{ kV RMS}$  (or  $100 \text{ kV peak}$ ). Close this dialog by clicking on the **Cancel** button at the bottom of the dialog and left-click anywhere in an empty space on the page, to de-select the selected source component. For a  $100 \text{ kV}$  source voltage, we know that the mid-point voltage should then be  $50 \text{ kV peak}$ , and the load current should be  $50 \text{ kA peak}$ . Now let us run the simulation and actually verify the current and voltage waveforms.

To run a case, simply click on the **Run** button in the **Home** tab of the ribbon control bar.

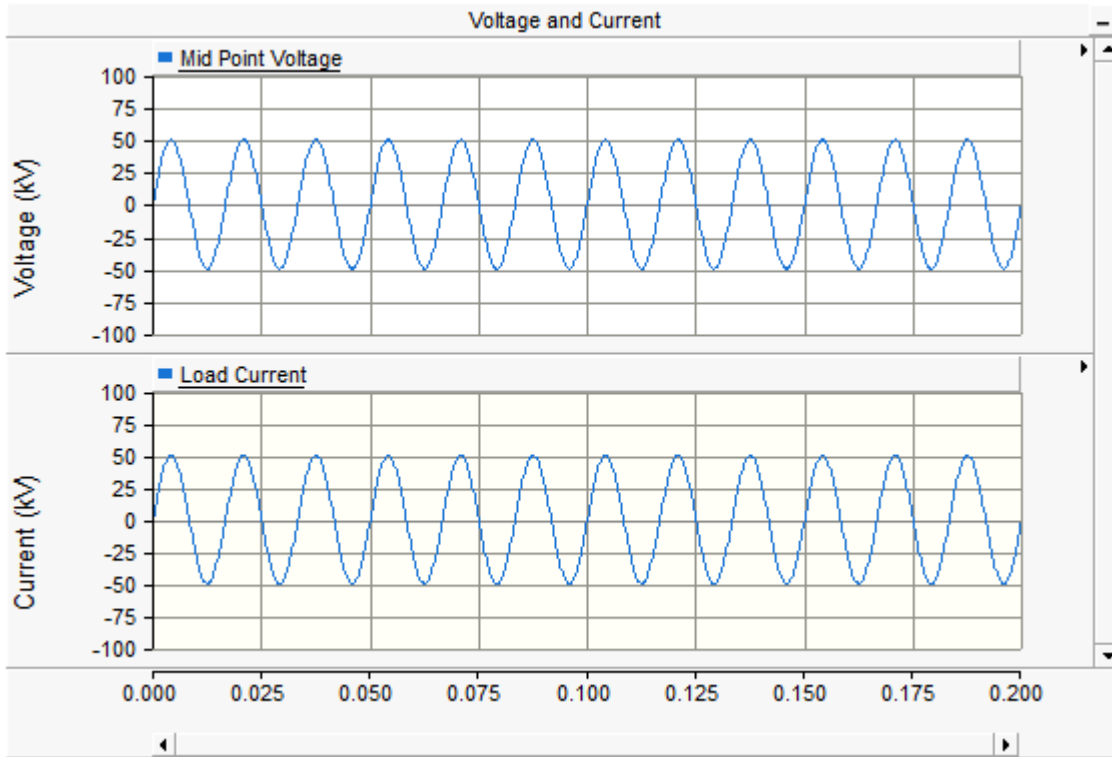


When this button is pressed, PSCAD will go through several stages of processing the circuit before starting the simulation. You should see messages in the status bar, at the bottom of the PSCAD window, related to various stages of the process. Depending on how fast your computer is, you may not be able to read these.



Watch the graphs as the simulation progresses. If you look near the bottom-right corner of the environment, you will see a message *xx%* complete where *xx* represents the percentage of the total simulation length. To the right of it you will also see the current simulation time, which changes with the simulation. Once again, depending on the speed of your computer, the simulation may finish almost instantaneously.

This tutorial case is set up to run for 0.2 seconds. At the end of the run you will see the message EMTDC run completed in the status bar. Your plots should look similar to the following, depending on your plot settings:



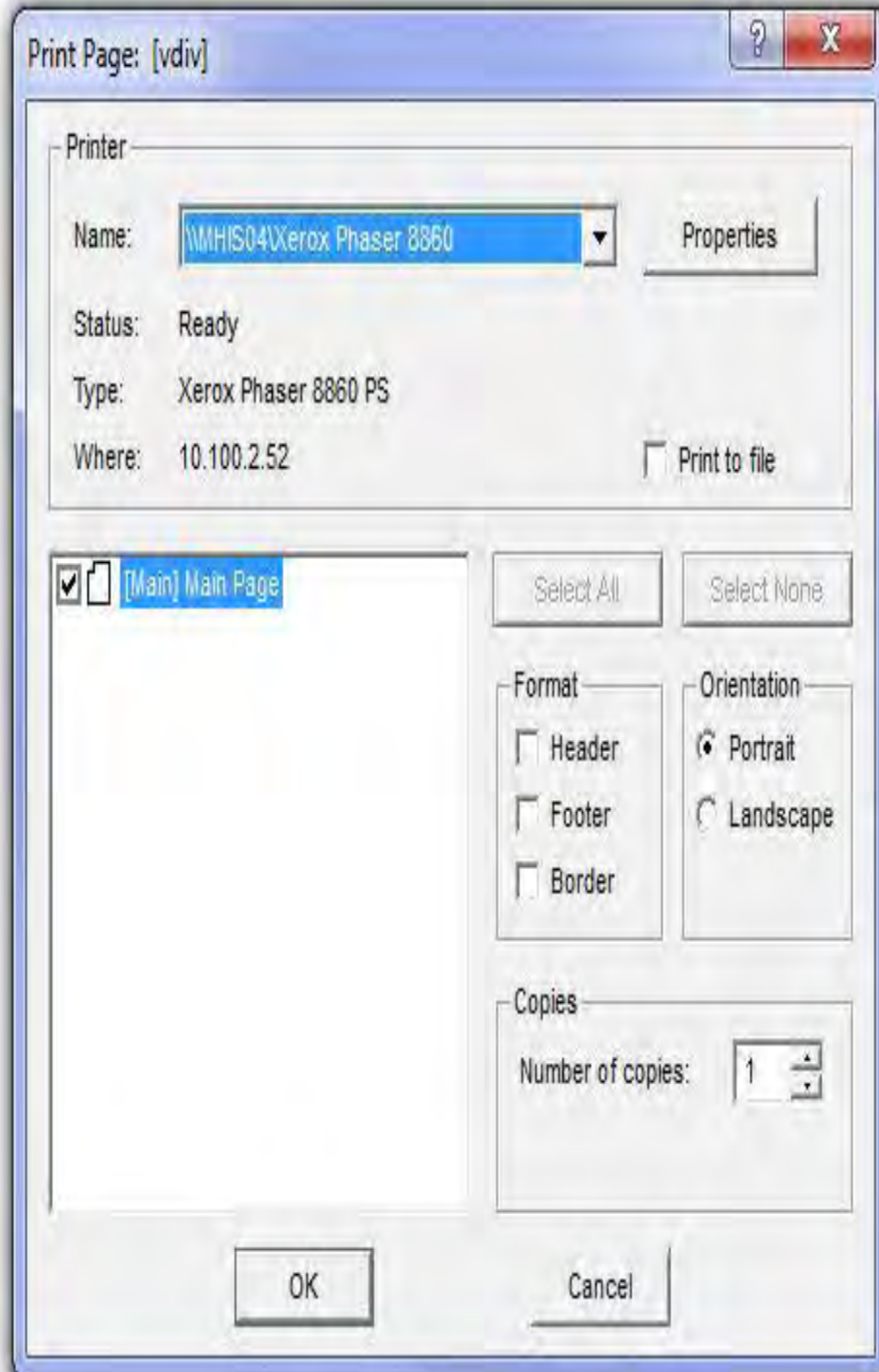
Make sure that your simulation produces the same result as shown here. This is one step towards ensuring that your PSCAD is installed correctly.

Click again on the **Run** button to see the run once again. PSCAD will go through all three stages (i.e. compile, build and run); however, you may not be able to detect the first two stages, as they pass by very quickly. This is because PSCAD performs them only if changes have been made to the circuit.

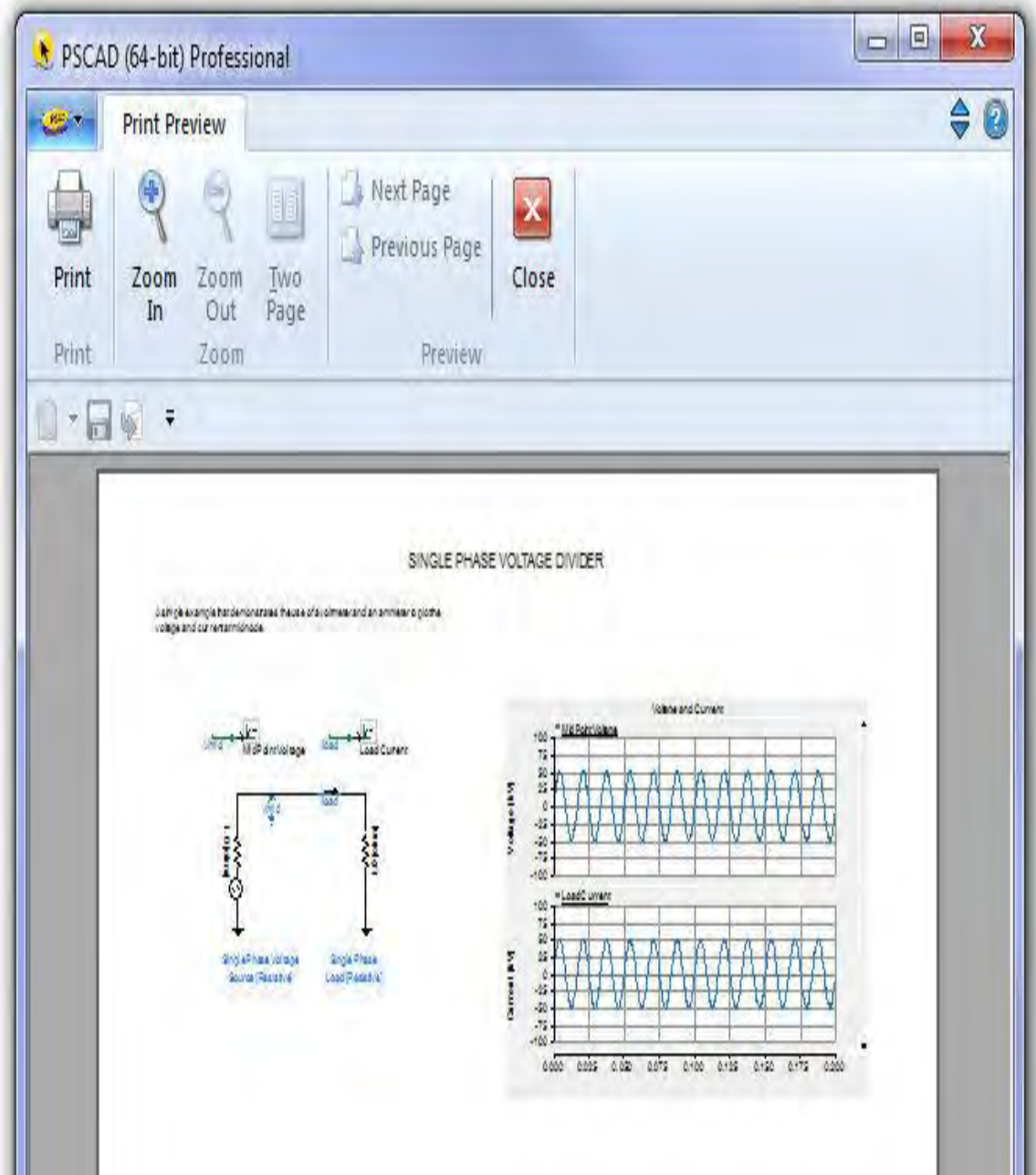
## Printing the Circuit

To print the circuit along with the graph you just simulated, click the right mouse button on the background of the Schematic and select either the **Print Page** or **Print Preview Page** items.

If you selected **Print Page**, the *Print Page* dialog will display. The contents of the *Print Page* dialog depend on what you are printing – click the **OK** button to proceed.



If you selected **Print Preview Page**, all graphics and text in the project will be displayed in the *Print Preview* screen (see below). Select **Print** to display the *Print Page* dialog (see above).



## Workspace Pane

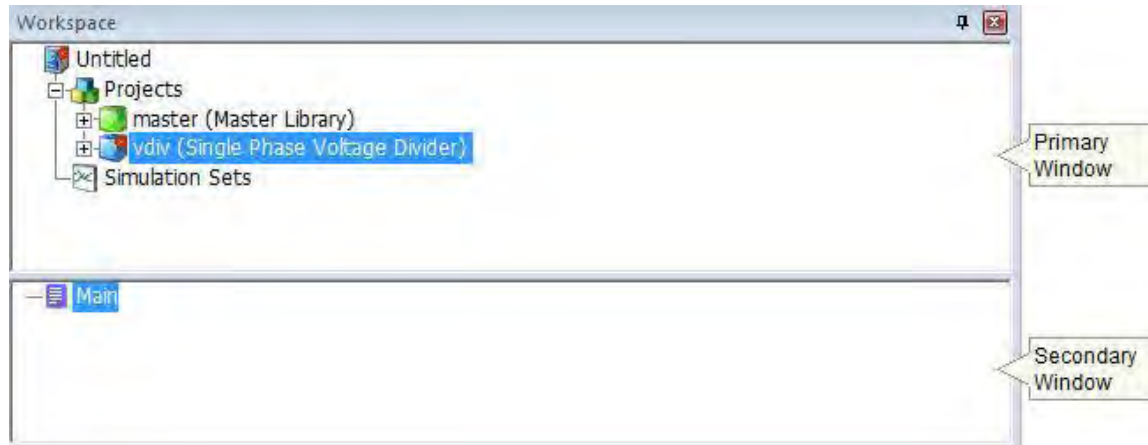
# The Workspace Pane

A workspace is the central operating hub within the PSCAD environment. It not only provides an overview of all projects currently loaded, but also organizes simulation sets, data files, signals, controls, transmission line and cable objects, display devices, etc. within an easily navigable environment.

**NOTE:** The master library is always the first project loaded into the *Projects* list. The master library cannot be unloaded.








In versions previous to v4.5, the workspace and the application itself were inextricably linked together as one complete unit. Now, the application and the workspace have been divided into separate entities. What this means from the user's perspective is that an entire workspace may be loaded, saved and unloaded without having to close the application. A single workspace may house multiple projects, including both libraries and cases, as well as possessing its own unique setting options.

The workspace pane is divided into two sub-windows. The primary window contains a list of loaded projects, and the secondary window context is based on whatever project is selected in the primary window. Its primary function is to provide a navigation tree for ease in jumping from module to module.



## The Primary Window

The primary window is used mainly for inter-project navigation, viewing project related data files and organization of simulation sets. Icons are included for visual differentiation between case and library projects. The main icons used here are listed below:

-  Workspace Branch
-  Projects Branch
-  Library Project
-  Case Project
-  Simulation Sets Branch
-  Simulation Set
-  Simulation Task

## Projects Branch

When a case or library project is loaded, the project namespace and description will appear by default in the primary workspace window, under the *Projects Branch*. It is of course possible to load multiple projects, and these will be listed in the order in which they are loaded. Each *Project* in the *Projects Branch* includes two additional branches to house data specific to that project. These are the *Definitions* and the *Temporary Folder* branches.

See Operations and Feature Overview for more on creating, loading and saving projects.

## Definitions Branch





The definitions branch contains a list of all definitions that are stored locally within that project. Definitions for component or module instances that exist in a project, but are stored in other projects (ex. master library components), will not appear in this local definitions branch.

The image below shows the list of component definitions for the case project called *test*. In addition to the definitions of each module, the existence of a component definition (called *user\_comp*) is also indicated. The number in parenthesis affixed to the definition name indicates the total number of local instances of this definition in the project.

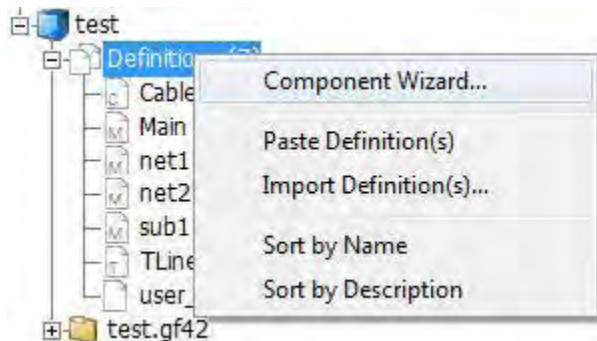


**NOTE:** It is not recommended to store user-defined component definitions within case projects. All component definitions should exist exclusively in library projects to avoid having to maintain multiple versions of the same definition.

The main icons used here are listed below:

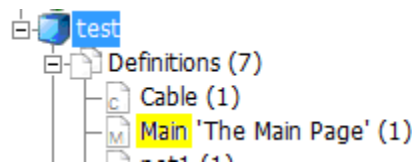
-  Component Definition
-  Module Definition
-  Transmission Line Definition
-  Cable Definition

Right-clicking directly over the *definitions* branch will bring up a menu as shown below:

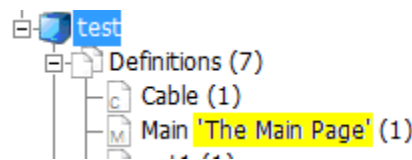


The following list describes the functions of this pop-up menu:

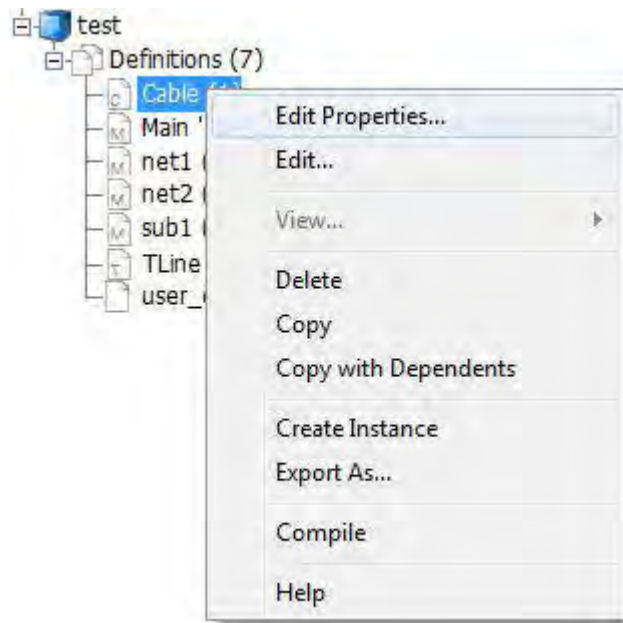
- **Component Wizard:** This menu item will display the Component Wizard pane if not already displayed. The Component Wizard is used to create both **Module and Non-Module Component, TLine or Cable** definitions. Once created, the definition will appear at the bottom of the definitions list, and the instance may be dropped onto the Schematic canvas by dragging the cursor to the appropriate location and left-clicking.
- **Paste Definition:** This function may be used to paste a definition that has been copied from this project or from another project loaded in the workspace primary window (see the *Copy* menu function described below).
- **Import Definition(s)...:** This function is used to import stored definition files (\*.psdx or the older \*.cmp format). See Importing/Exporting Definitions in the next chapter for more details.
- **Sort by Name:** Select this option to sort the definitions list by name (see below). Continually selecting this option will toggle the alphabetical order from A to Z or Z to A.



- **Sort by Description:** Select this option to sort the definitions list by description (see below). Continually selecting this option will toggle the alphabetical order from A to Z or Z to A.



If you right-click with the mouse pointer over a specific component definition in the list, the following menu will appear:



The following list describes the functions of this pop-up menu:

- **Properties...**: This brings up the *Definition Settings* dialog window for editing the definition *Name*, *Description* and *Label(s)*. See Editing Definition Properties for more details.
- **Edit...**: Select this option to edit the definition of the component (opens the Graphic window for components, or the Schematic window for modules, transmission lines and cables). See Editing a Component or Module Definition for more details.
- **View...**: This menu item is used to populate (or view) a list of any runtime objects that happen to exist within a particular module definition. The user may choose to view all objects, or a filtered list as per the sub-menu below. See Runtime Objects in the Definitions Branch for more details.
- **Delete**: Select this option to delete the definition. You will be prompted whether to delete the definition as well as all existing instances (select **Yes**), to delete only the definition (select **No**), or to cancel the action (select **Cancel**).
- **Copy**: Select this option to copy a definition, which may then be pasted (see *Paste Definition*, above), then instantiated (see *Create Instance*, below).
- **Copy with Dependents**: Select this option to copy this definition and all of its dependents. This means that if this is a module definition, and it contains other modules as part of its definition, then these other modules will be included in the copy. See Copy with Dependents in the next chapter for more details.
- **Create Instance**: Select this option to create an instance of the definition. Note that once this option has been selected, go to a blank part of the Schematic canvas, right-click and select **Paste**. This will paste the newly created instance directly onto the page. Once the first instance has been created, subsequent copies may be made by directly copying the original instance. Holding down the Ctrl key and using Drag and Drop will also perform this feature.
- **Export As...**: This option brings up the *Save Definition As* dialog so that you can save the definition to a Definition File (\*.psdx). See Importing/Exporting Definitions in the next chapter for more details.
- **Help**: If the definition is a regular component, selecting this option will open the associated help file.

#### Module Components Only:

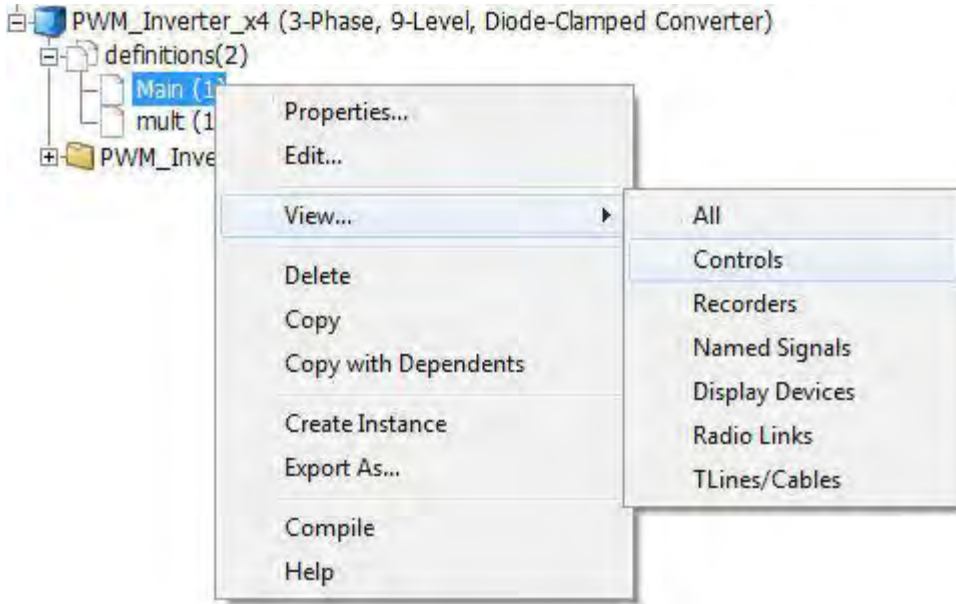
- **Compile**: Selecting this option will compile only the module selected. All module instances linked to this definition will of course be affected.

A double-click on a module definition will bring you directly to its definition canvas in the Schematic canvas, whereas double-clicking on a regular component definition will bring you directly to its definition in the Graphic view. Double-clicking on a transmission line or cable will bring you to the definition Editor canvas.

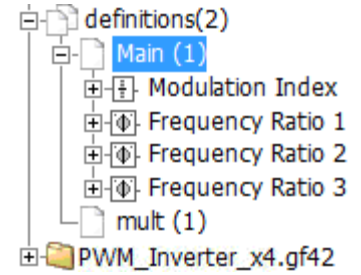
## Runtime Objects in the Definitions Branch

Runtime related objects, such as Output Channels, Controls, Graphs, etc., are themselves specialized components, which are placed on the canvas of a module. As such, they help define the definition of the module, and are therefore organized under the definitions branch on a per definition basis.

Note that in order to avoid needless, continuous population of these lists (and thereby conserve processor operations), the user must manually choose to view each of these lists.



Manually Populating the List



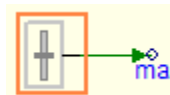
Resulting List of Controls-Type Runtime Objects in definition *Main*

Each module definition branch can contain a list of all runtime objects that exist in that module. Whenever a runtime object is added, a record of it is immediately added under the appropriate module definition in the workspace.

Runtime objects can be categorized into several main groups:

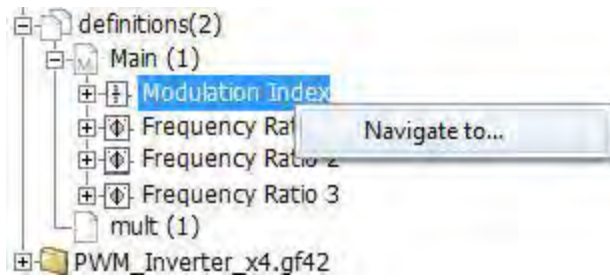
- Controls
- Recorders
- Display Devices
- Named Signals
- Radio Links
- TLines/Cables

A left double-click on any runtime object in the list will highlight that object directly in the corresponding module definition canvas. For example, double-clicking on the *Modulation Index* object in the above diagram will result in the following in Schematic view:







This may also be achieved by right-clicking on a module listing and selecting **Navigate To....**







Icons are included in this branch for visual distinctness:







Controls:

-  Slider Component Instance
-  Two State Switch Component Instance
-  Dial Component Instance
-  Push Button Component Instance


Recorders:

-  Output Channel Instance
-  RTP/COMTRADE Recorder Component Instance

Display Devices:

-  Control Panel Instance
-  Plot Frame Instance
-  XY Plot Instance
-  PolyMeter Instance
-  PhasorMeter Instance
-  Oscilloscope Instance



Named Signals:

-  Data signal defined with a Data Label

Radio Links:

-  Transmitter
-  Receiver

TLines/Cables:

-  TLine
-  Cable

Transmission Lines and Cables

Transmission segment wires are themselves very similar to modules. As such, they are treated like a module instance when they are placed in the circuit.

If the line is an overhead line with *Termination Style* in **Remote Ends** mode, then a [+] box will appear beside the branch name. Expanding this branch (press the [+] symbol) will show the two interface components representing both ends of the transmission line. Note that cables will always have remote ends, as direct connection mode cables are not supported.

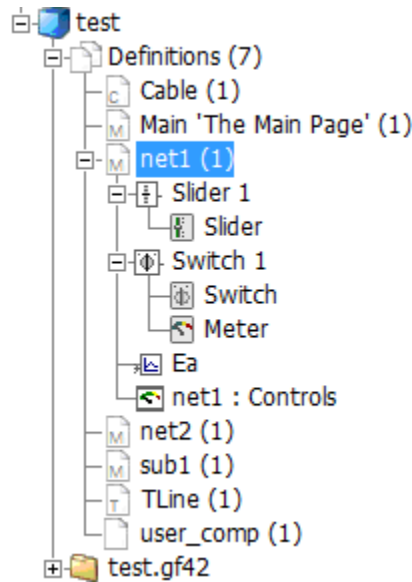
TLine/Cable Interface Components:

- → Remote End Interface Component Instance

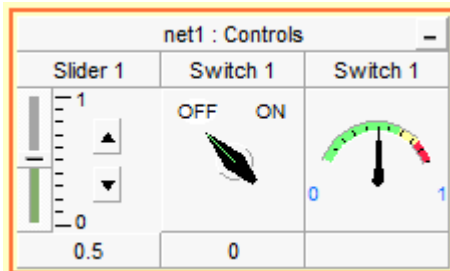
Observers

Output channels and control objects can exist in a project without being associated with a curve, meter or control interface (i.e. the data is not plotted or monitored). If any of these objects are added as a curve to a graph, meter to a control panel, or added directly to a PolyMeter, PhasorMeter or Oscilloscope, an entry referred to as an observer will be added as a sub-branch to the corresponding recorder or control branch. If the object is displayed in more than one display device, an observer will be added for each occurrence.

The figure below shows the existing controls and recorders in the *Network #1* module definition for a test project. The module contains a slider component called *Slider 1*, a two-state switch component called *Switch 1*, as well as two output channels entitled *Ea* and *Switch 1*. Each object possesses an observer to indicate that the output of the objects is being sent to a display device.








A simple double-click on any observer will point you directly to that occurrence of the runtime object within the display device. For example, double-clicking on the *Slider 1* observer in the above diagram will result in the following in Schematic view:







Icons are included for all observer types. These are listed below:

Recorders:

-  Curve
-  Meter
-  PolyMeter
-  PhasorMeter
-  Oscilloscope

Controls:

-  Slider
-  Dial
-  Two State Switch
-  Push Button

#### Group Name Based Sorting

If any of the runtime objects are part of a runtime group, then the group name will precede the object name. This automatically categorizes the objects according to group name:

$$V_1 =$$

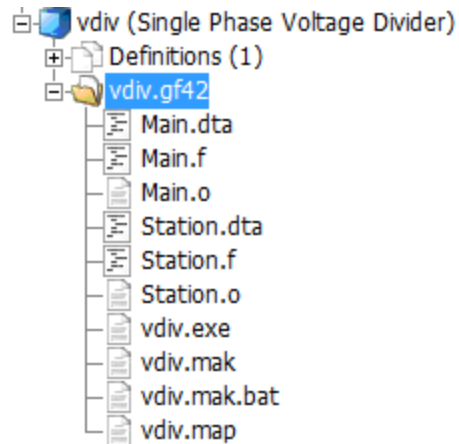
In the above example, controls *Slider 1* and *Switch 1* have been given the group name *Control Group*, whereas output channels *Ea* and *Switch 1* have been given the group name *Output Channels*. For more information, see Grouping of Runtime Objects.

## Temporary Folder Branch

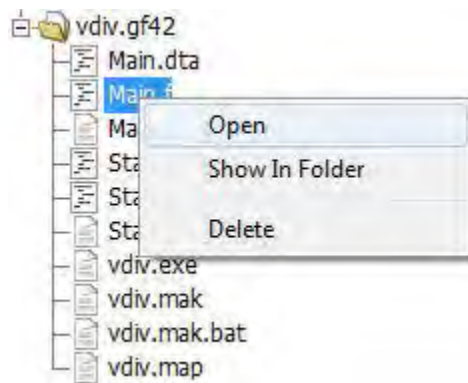
The Temporary Folder Branch is used specifically for convenient access to files found in the project temporary folder, such as FORTRAN, Data, Output, etc. files.

**NOTE:** The files will only be visible if you compile the project first (see Compiling and Building a Project in the next chapter).

To view the contents of the *temporary folders branch*, left-click the [+] box beside the project name:



A simple double-click on any file will open that file in the main viewing area of the environment. Right-clicking on the file will invoke a pop-up menu:



The functions in this menu are as described below:

- **Open:** Open the selected file for viewing in the main window.
- **Show in Folder:** Select this option to open Windows file explorer at the folder location where this file resides.
- **Delete:** Delete the selected file from the temporary folder.

## The Master Library

The master library is always the first project listed in the workspace primary window whenever PSCAD is started. It contains most of the components required to build almost any circuit. To open the master library, simply left-click the title in the workspace. The main master library canvas will open (Schematic window), giving access to all of its components.

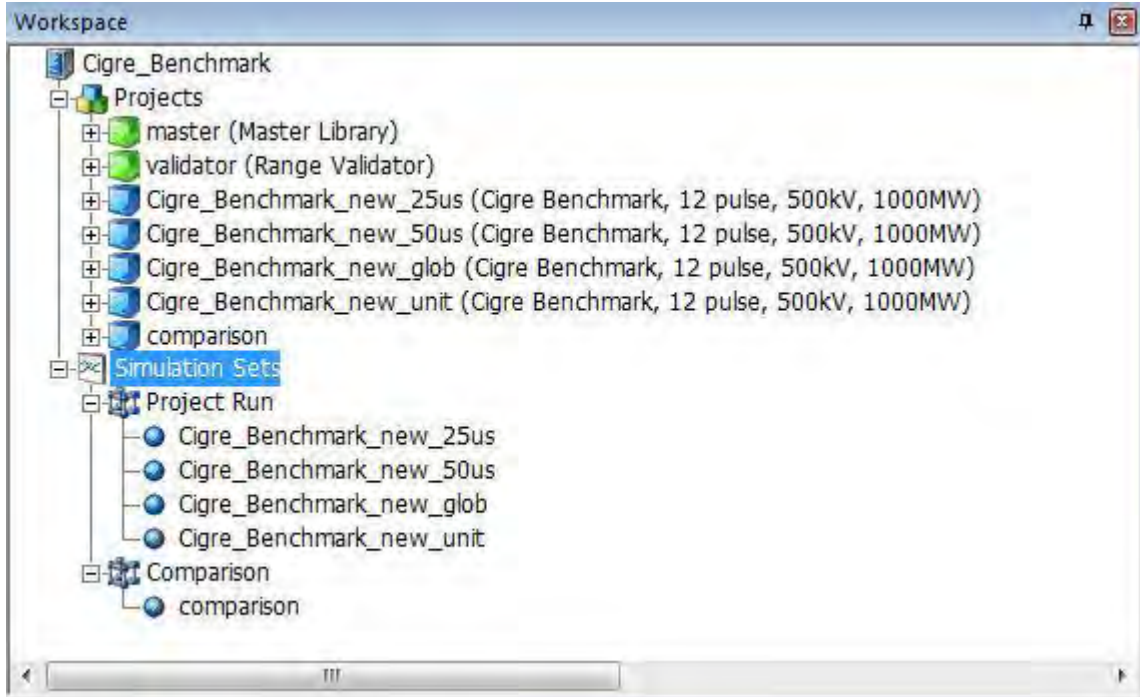
Components stored in the master library are categorized into several modules (located on the main page), according to the functionality of the component. For example, all transformer components are stored in the *Transformers* module. In addition to these categories, most master library components are also located on the main page. Some users may find this format allows for easier navigation to the correct component.

**NOTE:** Component definitions stored in the master library can be viewed, but not directly modified. If you wish to modify one of these definitions, copy the definition into another library project and rename it first..

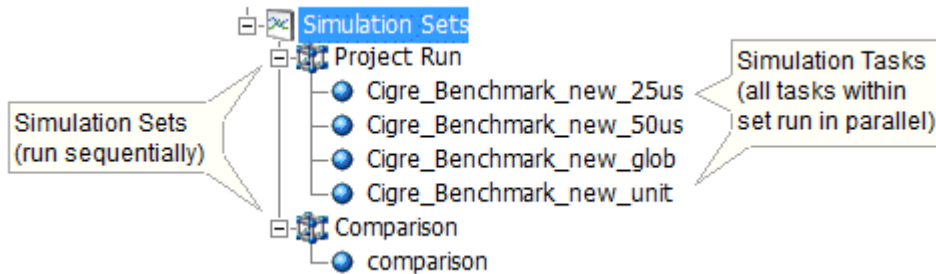
If you want to add any of these components to your own case, simply copy the instance directly from the master library and paste it into your own project. You can also use the **Models** tab on the ribbon control bar or the Library Pop-up Menu system.

## Simulation Sets Branch

It is possible to simultaneously launch and run multiple project simulations. Both sequential and parallel simulation runs are possible via the defining of *Simulation Tasks* within what are referred to as *Simulation Sets*. Only projects loaded under the Projects branch may be added as a task in a simulation set.



All simulations in a particular set will be launched simultaneously in parallel, utilizing all processor resources available on the local machine. All sets may be run sequentially: In the image above for example, if the user chooses to run all simulation sets, the set *Project Run* will launch and run all of its tasks in parallel. When these are complete, the simulation set *Comparison* will launch its tasks immediately after, and so on.



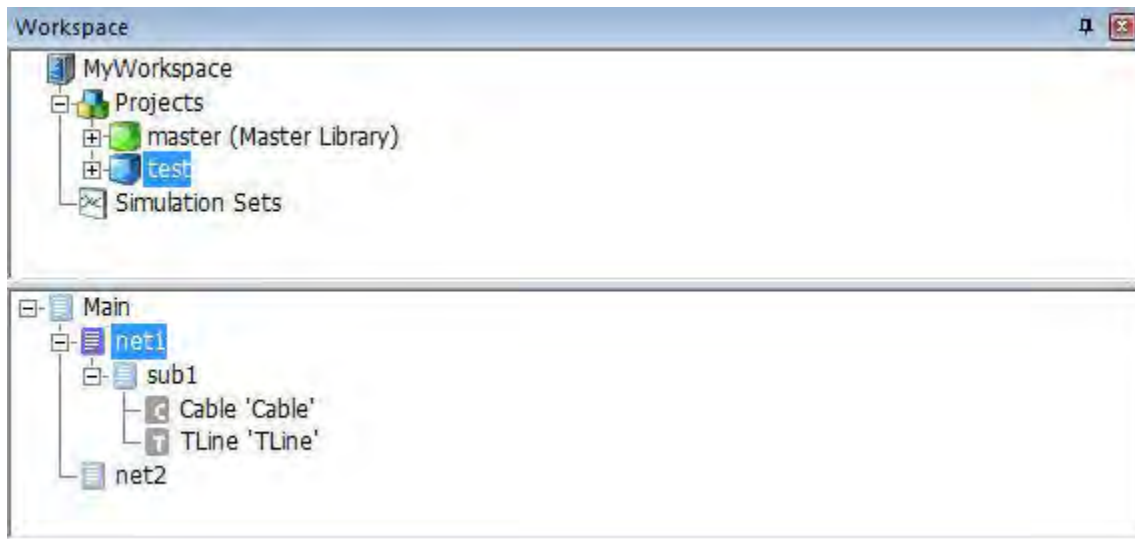
See Simulation Sets/Multiple EMTDC for more on creating and running simulation sets.

## The Secondary Window

The secondary window displays data in a project-specific context. That is to say that its contents are based on whatever project is selected in the primary window. The secondary window also acts as an intra-project navigation tool.

## Module/TLine/Cable Hierarchy Branch

This branch lists all module, transmission line and cable instances in the project. Starting with the main page, modules are organized according to their hierarchy, which helps to simplify navigation, as well as provide a complete overview of how the project is structured. Transmission lines and cables will appear inside their parent module.



For example, the case project above (with namespace name *test*) contains a main page with two modules called *net1* and *net2*. The module *net1* contains another called *sub1*. The display format of the names in this branch is adjustable via the option called *Namespace* in the *Workspace* category of the *Application Options* dialog. Depending on the setting of this option, the format is as follows:

Namespace display enabled:

```
<namespace_name>:<definition_name> '<instance_name>'
```

#### NOTES:

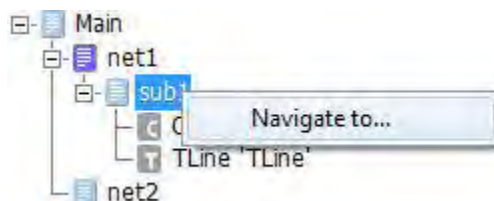
- If the **Namespace** option is set to **Show if definition as external**, the namespace will display if the definition originates from a different project.
- If the **Namespace** option is set to **Show always**, the namespace will display regardless of whether the definition originates from the local or from a foreign project.

Namespace display disabled (**Namespace** option is set to **Hidden**):





```
<definition_name> '<instance_name>'
```

The namespace name identifies the project where the definition resides. The definition name is of course the name of the definition. The instance name is the name given to that specific instance of the module (refers to the *segment* name for lines/cables). In the example above, all definitions reside in the local project *test*. None of the three modules have been given instance names, so therefore none are displayed. *TLine* and *Cable* are the segment names given to these particular instances of the transmission line and cable (by default the segment name is the same as the definition name when a line or cable is created).

A left double-click on any module, transmission line or cable listing in the hierarchy will bring you directly to canvas of that instance in the Schematic window. This may also be achieved by right-clicking on a module listing and selecting **Navigate To....**



Icons are included in this branch for visual distinctness:

-  Module Instance (local)
-  Module Instance (foreign)
-  Transmission line instance
-  Cable instance

The icon of the instance that is currently being viewed will appear as a darker shade.

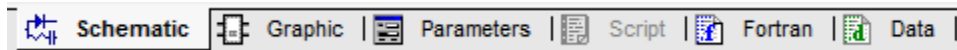
## The Definition Editor

The *Definition Editor* is the most important part of the application environment. It is where the majority of project design work is performed, including the graphical construction of circuits (i.e. Schematic view), component graphic design, as well as scripting.

The definition editor window is invoked (in Schematic view by default) by a left-click on a project in the workspace window.

## Viewing Windows

The definition editor is divided into six sub-windows. Each of the sub-windows can be accessed by simply clicking on the specific tab on bar at the bottom of the editor window. The tab bar is shown below for reference:



As you can see, the *Script* tab is initially disabled (greyed-out). This tab is used exclusively for non-module component design, and will not be enabled unless editing a non-module component definition. See Editing a Component or Module Definition in Chapter 9 for details on this.

### Schematic

The Schematic tab is always the default view when a project is first opened. This is where all control and electrical circuits are constructed.

### Graphic

This Graphic window is used for editing the graphics of a component definition or module.

### Parameters

This Parameters window is used for editing the parameters of a component definition.

### Script

This Script window is used for editing non-module component definition code.

### Fortran

The Fortran tab is a simple text viewer that allows easy access to the EMTDC Fortran file corresponding to the module definition currently being viewed. For example, if you are viewing the main page of a project in Schematic view, the Fortran window (click the **Fortran** tab) will show the EMTDC Fortran file corresponding to the main page.




## Data

The Data window is a simple text viewer that allows easy access to the EMTDC input data for any existing electric network, corresponding to the module being viewed in the Schematic window. For example, if you are viewing the main page of a project in Schematic view, the Data window (click the **Data** tab) will show the EMTDC input data for the main page.

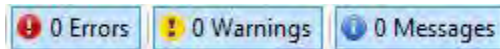
# Build and Runtime Message Panes

## Build Messages Pane

The primary purpose of the *Build Message* and *Runtime Message Panes* is to provide an interface for viewing simulation feedback, and for troubleshooting your simulation. All informational, error and warning messages, either issued by a component, PSCAD, or EMTDC, can be viewed in one of these two panes. In the *Build Messages Pane*, the messages are divided into errors, warnings and informational message categories. A distinction can be made between error and warning messages simply by the color of the symbol preceding the message. The color code is as follows:

-  Warning
-  Error
-  Information

The message types may be filtered by selecting the appropriate buttons in the *Build Messages Pane* title bar. All messages are displayed by default.



Build messages are errors and warnings related to the compilation and building of FORTRAN, data and map files for the project. PSCAD has the capability of detecting a number of different types of system inconsistencies related to this. See Error and Warning Messages for more information. Note that any warning or error messages defined in the Checks segment of any component definition will be displayed as a build message.

## Warning Messages

Warning messages are not considered detrimental to the simulation run, and PSCAD will continue to build and run the project regardless of any warnings. However, warnings can indicate areas of the system that, although not technically illegal, will still affect the simulation results (e.g. a mistakenly disconnected node). It is therefore important to study the output window every time the project is built and run if there are any warning messages.

## Error Messages

If an error is reported, the simulation build will be halted immediately. The user must then study any error messages reported and attempt to determine the problem source. See Locating the Problem Source for more details.

## Runtime Messages Pane

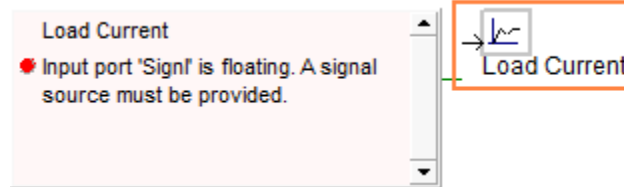
Runtime messages provide error and warning messages related to the simulation run – that is, messages sourced from EMTDC. Runtime messages are usually more serious in nature and can involve numerical instabilities and other problems of this type.

It is important to study the messages thoroughly. In some cases, PSCAD will direct you towards the subsystem and node number in the electrical system where the problem is occurring. The Search utility can help point you toward the problem area. See Error and Warning Messages for more information.



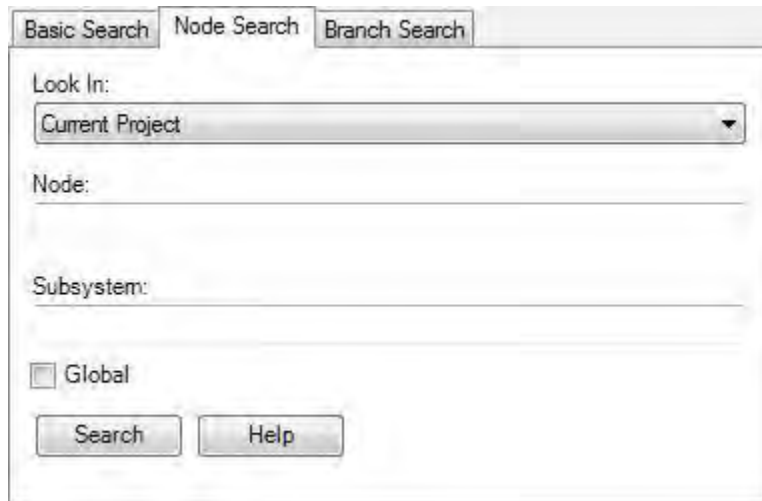
## Locating the Problem Source

The output window provides an easy method of locating the source of any displayed message: Simply left-click the hyperlink associated with the message. PSCAD will then automatically open the source page in Schematic view and point directly at the problem with a message box:



## Search

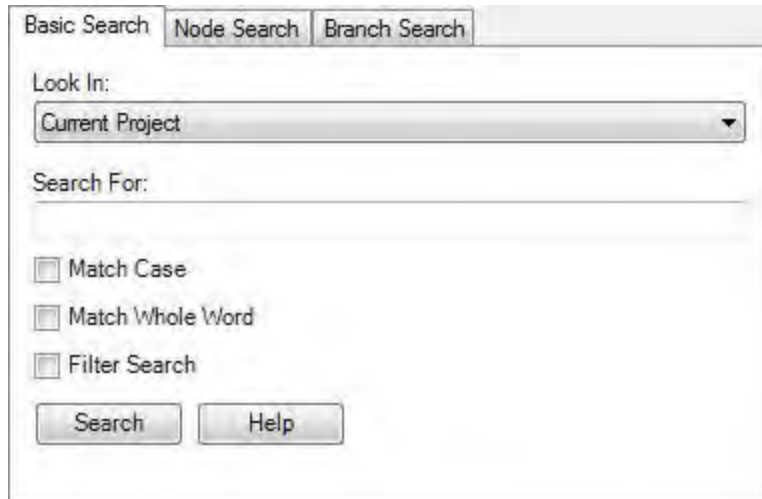
If a message indicates that the problem is arising at a designated subsystem, node and/or branch number, then you can utilize the *Search Pane* to search for this exact location. Simply invoke the Search Pane. Enter the subsystem and node, or branch number indicated in the runtime message and select the **Search** button.



The feedback from this search will be displayed in the Search Results pane (will appear automatically). Simply left-click the hyperlink associated with the search result and PSCAD will automatically open the source page in Schematic view and [identify the item using a colored enclosure, similar to that shown in Locating the Problem Source above.](#)

## Search Pane

If the user wishes to search a project for a particular object, the search utility may be utilized. To invoke the Search dialog window, click on the **View** tab in the ribbon control bar and select **Search** from the **Panes** drop list, press the **Search** button in the **Home** tab in the ribbon control bar or press **Ctrl + Shift + f** on your keyboard.



The search window is divided into three main parts: **Basic**, **Node**, and **Branch Search**.

## Basic Search

Basic searching is for just about anything within the workspace, except for electrical node and subsystem numbers. The following sections describe the input parameters needed for your search.

### Look In

This is the primary scoping mechanism. Select to search in the either current module or project, or all projects loaded in the workspace.



### Search For

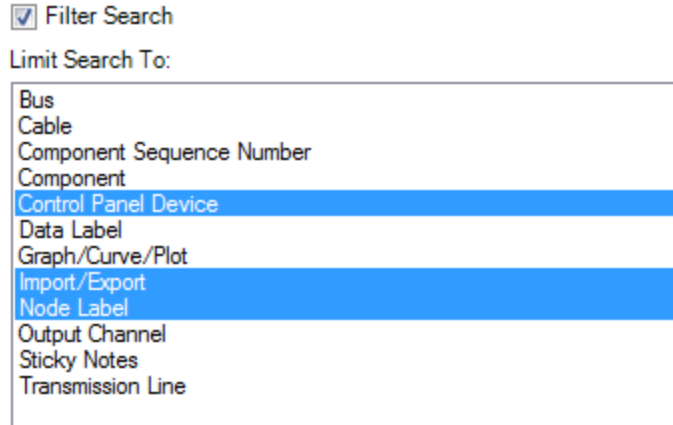
This is the string of text you would like to search for. Note that PSCAD maintains a history of previously searched items. Simply enter the first letter and whatever similar searches were performed in the past will appear in a drop list.

### Match Case and Match Whole Word

These are standard searching devices to further scope your search.

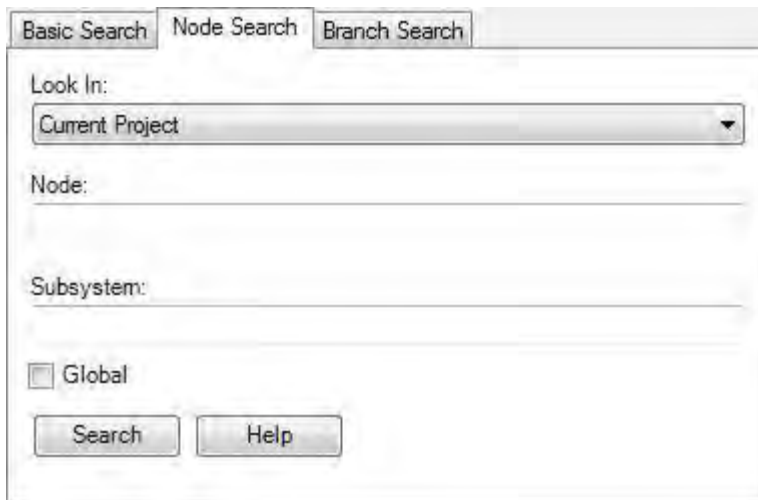
### Filter Search

This is a secondary scoping mechanism that works in conjunction with **Look In** above. Here you select the type of objects you would like to limit your search to.



## Node Search

A node number search is a different animal, in that it requires the project be compiled first, thereby generating the node numbers.



A node number search is a special type, where you can look directly for an electrical node, by node and subsystem number, as opposed to a character string. Note that project must be compiled first in order to get any node search results!

Some additional selections and fields become enabled if this option is selected:

- **Look In:** Select *Current Project* or *Current Module*. *Current Project* is selected by default, but you can narrow your search by confining it to the current module canvas (*Current Module*).
- **Node:** The actual node number.
- **Subsystem:** The electrical subsystem number (if desired)
- **Global:** Select this option if the node you are searching for is a global node. Otherwise, the utility will search for local nodes.

## Branch Search

Like the node search, branch number search requires the project be compiled first, so that the branch numbers are generated.

A branch number search is also a special type, where you can look directly for a specific electrical branch number, based only on subsystem number. Note that project must be compiled first in order to get any branch search results!

## Viewing Search Results

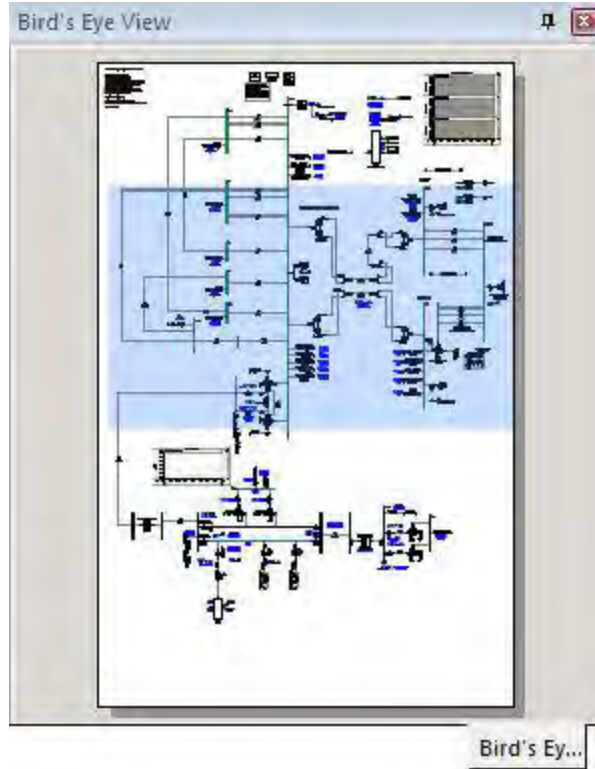
Search results are located in the Search Results window. This window will automatically open upon performing a search. Any result matching the search criteria will appear in a list form as shown below:

	Instance	Object	Definition	Symbol	Module	Namespace	Text
▶ 1	<a href="#">2084520790</a>	Control Frame	ControlFrame		Main	chopper_optimization	
2	<a href="#">1639522068</a>	User Component	UserCmp	En3	Main	chopper_optimization	

Simply left click the hyperlink in the *Instance* column and PSCAD will automatically highlight the source in the Schematic canvas.

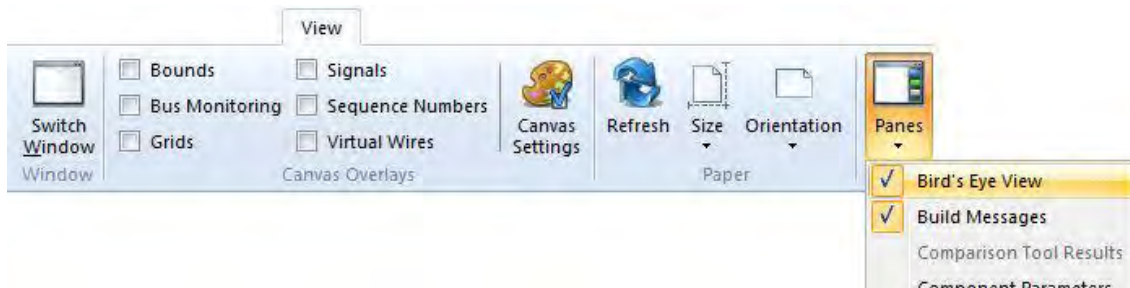
## Bird's Eye View Navigation Pane

The *Bird's Eye View Pane* provides an overview of the entire Schematic or Graphic canvas and indicates what is currently in view. This tool is an important part of the collection of navigational tools in PSCAD, and is used to easily zoom and navigate a Schematic. This pane is particularly helpful when working with very large projects.



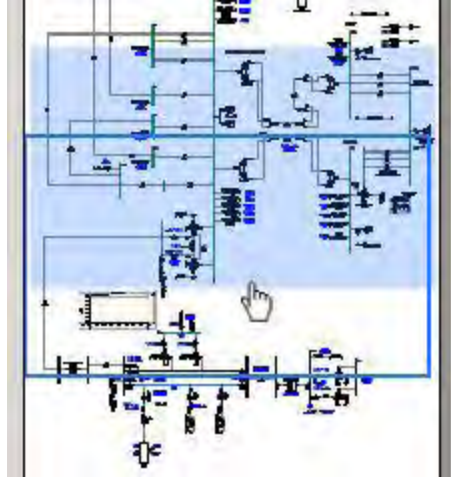
## Displaying the Bird's Eye View Pane

The *Bird's Eye View Pane* is not displayed by default, but may be invoked via the **View** tab in the ribbon control bar by selecting it from the **Panes** drop-down box.

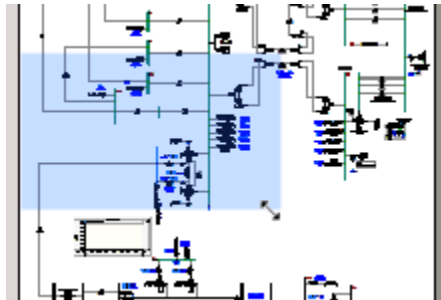


## Navigating a Zooming

A shaded box is used to represent the portion of the canvas that is currently in view. Using a left mouse drag action in the *Bird's Eye View Pane*, the shaded rectangle may be moved about. The Schematic or Graphic canvas in view will move position simultaneously according to the position of the shaded box. During movement, a rectangle outline will show the current position until the left mouse button is released.



Moving the mouse pointer near the edge of the shaded box will allow you to re-size it, effectively providing a zoom functionality on the canvas being viewed. Shrinking the shaded box will zoom in on the canvas, whereas expanding the box will zoom the canvas out.



## Parameter Grid Pane

The component parameter grid pane provides a convenient means to display the parameters for all instances of a given component or module definition. More importantly, it enables the ability to modify multiple parameter values in multiple component instances simultaneously.

Id	Name	Group	enab	Display	Scale	Units	mrn	Pol	Max	Min	Layer
85194344	q_inst		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
85213928	p_inst		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
85214336	p_inst filtered		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
85215560	q_inst filtered		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
95928792	IaRef		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
95929608	IbRef		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
95930424	IcRef		Yes	Yes	1.0		Last run only	No	2.0	-2.0	Default
95931240	VaRef		Yes	Yes	1.0		Last run only	No	2	-2	Default

## Viewing the Parameter Grid Pane

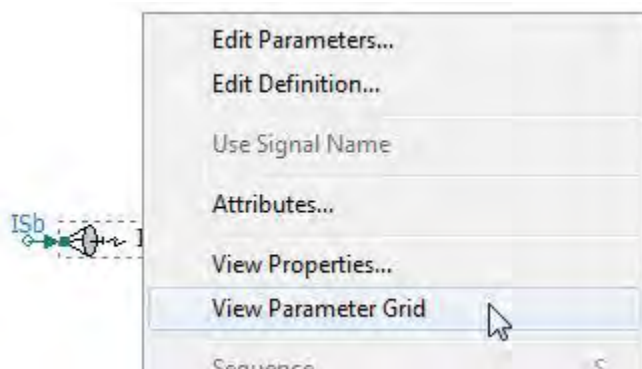
The parameter grid pane can be viewed by simply selecting it in the **Panes** drop list button, under the **View** tab of the ribbon control bar.



Once selected, dock the pane in a preferred location in the application environment.

## Initializing the Parameter Grid

Before the parameter grid can be used, it must be initialized for first use. Select a component type for which to view the parameters of all instances of that component, right-click and select **View Parameter Grid**.



The resulting parameter grid will immediately display inside the pane. There will be one row of parameters for each existing instance of that unique component definition. In the above image, the parameter grid is initialized for all radio link component (definition name 'radiolink'). The resulting parameter grid is shown below:

Parameter Grid

Select Module All Series\_AF: 'radiolink'

Id	Source	Name	Type	dim	rank	Mode	Min	Max	Layer
<a href="#">55675626</a>	Main	V	Real	3	0	Receive	-1.0	1.0	Default
<a href="#">1148526055</a>	Main	ISa	Real	1	0	Receive	-1.0	1.0	Default
<a href="#">1026448552</a>	Main	ISb	Real	1	0	Receive	-1.0	1.0	Default
<a href="#">1340931987</a>	Main	ISc	Real	1	0	Receive	-1.0	1.0	Default
<a href="#">1719973804</a>	Main	g1	Real	2	0	Transmit	-1.0	1.0	Default
<a href="#">1782575342</a>	Main	g2	Real	2	0	Transmit	-1.0	1.0	Default
<a href="#">1715606506</a>	Main	g3	Real	2	0	Transmit	-1.0	1.0	Default
<a href="#">1226877125</a>	Main	g4	Real	2	0	Transmit	-1.0	1.0	Default

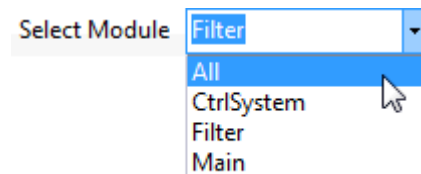
Parameter Grid

The title bar displays both the namespace and the definition name of the instances currently being viewed.

Series\_AF: 'radiolink'

## Sorting According to Module

The component instances displayed can be sorted by module. In this example project, there are a total of three modules. Choices for sorting are either of the three modules or all of them.



## Modifying Parameter Values

The power of the parameter grid lies in its allowance for modifying parameter values en masse. Multiple parameters (columns) and multiple component instances (rows) can be modified simultaneously by simply selecting the appropriate cells, right-clicking and selecting **Modify...**

Id	Source	Name	Type	dim	rank	Mode	Min	Max
<a href="#">766132655</a>	CtrlSystem	g1	Real	2	0	Receive	-1.0	1.0
<a href="#">928719668</a>	CtrlSystem	g2	Real				-1.0	1.0
<a href="#">214254178</a>	CtrlSystem	g3	Real				-1.0	1.0
<a href="#">1661827020</a>	CtrlSystem	g4	Real					
<a href="#">769326887</a>	CtrlSystem	g5	Real	2	0	Receive	-1.0	1.0

Modify...  
Write to file (.csv)  
Use S... Modify all the selected cell values

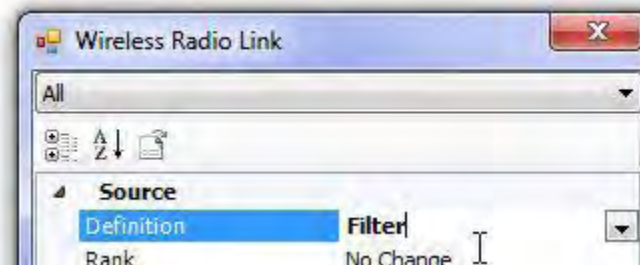
Upon selection of **Modify...**, a special parameter dialog will be dynamically constructed depending on which parameters (columns) were selected. For the above example, the resulting dialog will appear as follows:



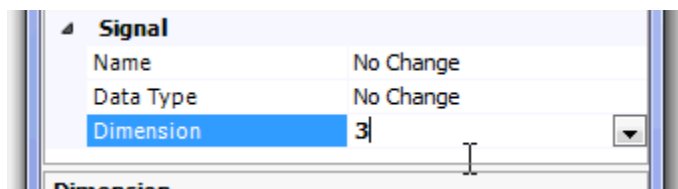


The default setting for each parameter is **No Change**. If the dialog is closed at this point no parameters will be modified. Let us say that we want to modify both the *Source* of each radio link receiver, as well as the input signal *Dimension*.

Perhaps the corresponding transmitters were moved in the project to the *Filter* module, thereby changing the *Source* for each of these radio link instances:



At the same time, we will also adjust the signal dimension for each radio link instance selected:



We now press the OK button, and the parameter values will change accordingly.

Id	Source	Name	Type	dim	rank	Mode	Min
<a href="#">766132655</a>	Filter	g1	Real	3	0	Receive	-1.0
<a href="#">928719668</a>	Filter	g2	Real	3	0	Receive	-1.0
<a href="#">214254178</a>	Filter	g3	Real	3	0	Receive	-1.0
<a href="#">1661827020</a>	CtrlSystem	g4	Real	2	0	Receive	-1.0

## Navigating to Component Instances

You can easily navigate to each instance of a particular component definition by simply clicking the navigable link in the Id column.

## Writing Parameter Values to File

Any selected block of cells can be written to file in comma separated variable (\*.csv) format. Simply right-click the selection and choose **Write to file (\*.csv)**.

Id	Source	Name	Type	dim	rank	Mode	Min	Max
<a href="#">766132655</a>	Filter	g1	Real	3	0	Receive	-1.0	1.0
<a href="#">928719668</a>	Filter	g2	Real	3	0	Receive	-1.0	1.0
<a href="#">214254178</a>	Filter	g3	Real	3	0	Receive	-1.0	1.0
<a href="#">1661827020</a>	CtrlSystem	g4	Real	2	0	Receive	-1.0	1.0
<a href="#">765326087</a>	CtrlSystem	g5	Real	2	0	Receive	-1.0	1.0

Modify...  
 Write to file (\*.csv)  
 Use Signal Name  
 Write this table to a '.csv' file

You will be presented with a dialog to save the file to a specified folder.

## Layers Pane

The layers pane is the interface to the schematic canvas drawing layers feature. Drawing layers are used to provide the ability to efficiently enable or disable components on the canvas, or to toggle the visibility of any objects that appear on the canvas.



Layers are a property of the project. Therefore, once a layer is created and defined, it may be used on any schematic canvas (i.e. module) that exists in that project.

## Viewing the Layers Pane

The layers pane can be viewed by simply selecting it in the **Panes** drop list button, under the **View** tab of the ribbon control bar.



Once selected, dock the pane in a preferred location in the application environment.

## Adding a Layer

To add a layer, simply give the layer a name in the layers pane and then press the **Add** button. A new layer will be created immediately.

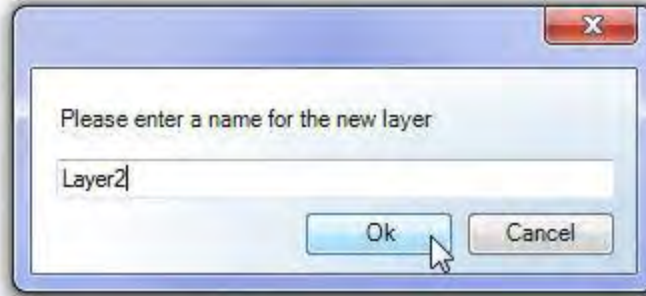


Enter Name and Press **Add**



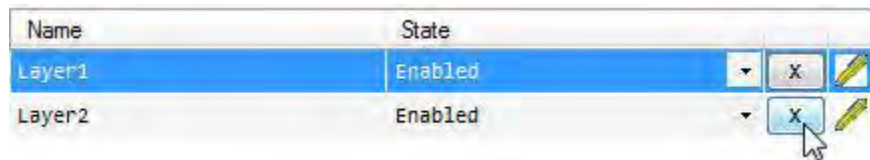
New Layer Created

Alternatively, simply press the Insert key while the layers pane is in context, or right-click anywhere within the pane and select **Add**. A dialog will pop-up asking you to name the layer. Enter an name and press **OK**.



## Removing a Layer

To remove a layer, simply press the **X** button on the layer to be removed.



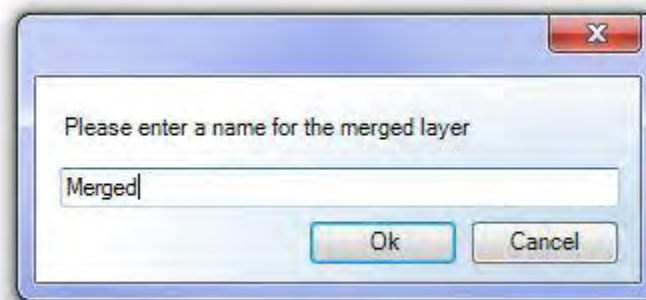
Alternatively, select the layer you want to remove and press the **Delete** key, or right-click on the layer and select **Delete**.

## Rename a Layer

To rename a layer, simply select the layer, right-click and select **Rename**.

## Merge Layers

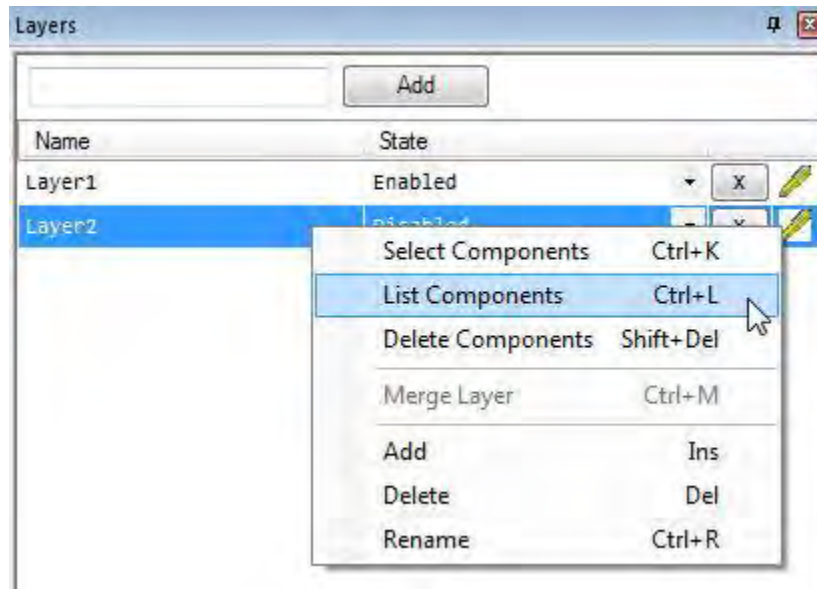
To merge two or more layers, simply select multiple layers, right click and select **Merge Layers**. A dialog will pop-up asking you to name the merged layer. Enter an name and press **OK**.



## Selecting, Listing, Deleting or Highlighting Components in a Layer

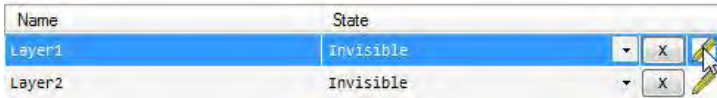
Once components have been moved into a layer, you can keep track of them by either selecting, listing or highlighting all components in that layer. You may also delete all components in a layer.

To select, list or delete all components in a layer, select the layer in the layers pane, right-click and choose **Select**, **List** or **Delete Components**:

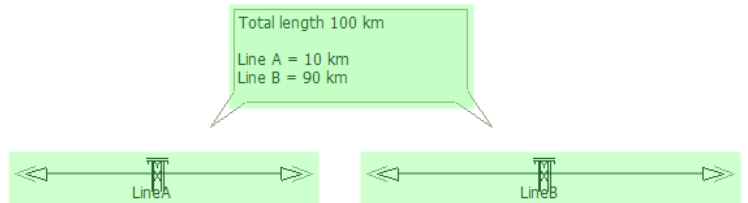


If **Select Components** is chosen, all components in the selected layer will appear selected on the schematic canvas. If **List Components** is selected, the components in the selected layer will be listed in the Search Results Pane.

To highlight all components in a layer, simply hold the mouse pointer over the high lighter icon. All components in that layer will appear selected on the schematic canvas.



Mouse Held Over Icon



Highlighted Components on the Canvas

## Tool bars, Menus and Shortcuts

### Menus

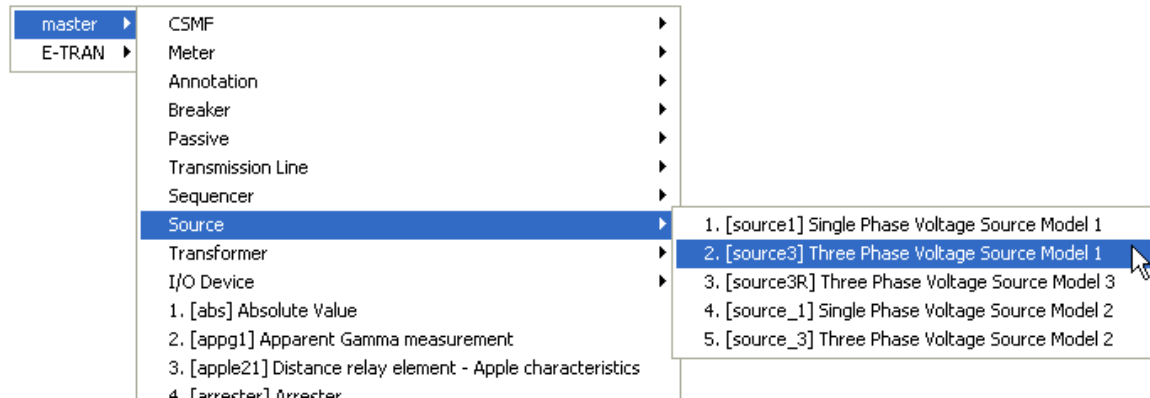
There are a few different types of menus available in PSCAD. The following sections provide a description of each.

#### Right-Click Pop-Up Menus

PSCAD contains an extensive set of right-click pop-up menus for support of location specific commands and tasks. Depending on the location of your mouse pointer, you can invoke pop-up menus almost anywhere in the environment by simply pressing your right mouse button. The menus that pop-up will always contain commands, which are specific to the area you are currently working in.

## Library Pop-Up Menus

Library pop-up menus are designed to provide an organized method of accessing desired component and module instances located in the master library, or any other user-defined library project. This menu system is invoked by pressing **Ctrl + right mouse button**, while the mouse pointer is over a blank section of any canvas in Schematic view.



**NOTE:** It is recommended to always provide an adequate description in user-defined components, so as to avoid confusion while using these menus. If a description is not provided, the menu will display the actual filename of the component.

Library pop-up menus display the definition name, followed by whatever is entered into the description field of the definition. In the above image, the user has two library projects loaded in the workspace: The master library and another library called *E-TRAN*. The mouse pointer is pointing toward the Three Phase Voltage Source Model 1 component in the *Source* module, in the master library. A left-click on this component will select it, and another left-click will drop it onto whichever canvas is open in Schematic view.

In general, library pop-up menus construct a system of menus according to the hierarchical structure of all library projects in the workspace. As a default, the master library will always be displayed first, and any other library projects will then be placed in the order as they appear in the workspace.

## Grouping of Components within the Menu

Components may be grouped into sub-menus if desired. For example, the menu shown above contains several sub-menus (CSMF, Meter, etc.). This is accomplished by setting the definition *Labels* parameter in the definition settings editor. See Editing Definition Settings for details.

## Ribbon Control Bar

PSCAD Tab

Home

Project

View

Tools

Utilities

Components

Models

T-Lines

Cables

Shapes

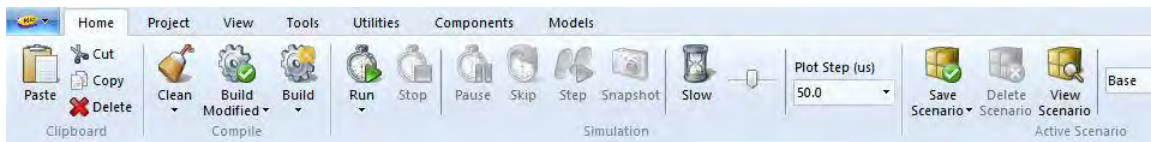
Filtering

Script

Minimizing the Ribbon






Quick Access Bar

The ribbon control bar provides easy accessibility to most features and components in PSCAD. Included with the ribbon is an inherent quick access bar, which is fully customizable for placement of favoured and well used button actions. The ribbon control bar is featured prominently across the top of the application environment.



## PSCAD Tab

The PSCAD tab includes functions related to project file organization and manipulation, including standard buttons for saving, loading, etc.




Button	Description
	Creates a new case project (*.pscx file) by default. See Creating a New Project.
	Opens the <i>Open File</i> dialog (*.pscx file) by default. See Loading a Project.
	Saves changes to the project (*.pscx or *.pslx file) currently in focus. See Saving Project Changes.
	Unloads the project currently in focus. See Unloading a Project.
	Print.



Launches the online help. See PSCAD On-Line Help System.






## New Menu

Hover the mouse pointer over the **New** button to invoke this menu.

Button	Description
	Creates a new case project. See Creating a New Project.
	Creates a new library project. See Creating a New Project.
	Creates a new workspace. See Creating a New Workspace.

## Open Menu





Hover the mouse pointer over the **Open** button to invoke this menu.

Button	Description
	Opens the Open File dialog. See Loading a Project.
	Opens the <i>Open File</i> dialog. See Importing a Project.
	Opens the <i>examples</i> folder containing example projects released with the software.
	Opens the <i>Open File</i> dialog at the <i>user source</i> folder. This folder path is set via the <i>My Folder</i> application option. See the Workspace category of the <i>Application Options</i> dialog.
	Opens the <i>Open File</i> dialog. See Loading a Workspace.

## Save Menu



Hover the mouse pointer over the **Save** button to invoke this menu.



Button	Description
	Saves changes to the project (*.pscx or *.pslx file) currently in focus. See Saving a Project.
	Opens the <i>Save Project As</i> dialog. See Saving a Project.
	Saves changes to the workspace (*.pswx file). See Saving a Workspace.
	Opens the <i>Save Workspace</i> dialog. See Saving a Workspace.





## Unload Menu
















Hover the mouse pointer over the **Unload** button to invoke this menu.









Button	Description
	Unloads the project (*.pscx or *.pslx file) currently in focus. See Unloading a Project.
	Unloads all projects in the workspace, except the master library.

## Home

The buttons in this tab consist of the most commonly used features. Some of the less straightforward buttons are described below:

Button	Description
	Clean the temporary folder.
	Compile (build) modified modules. See Compiling and Building a Project.
	Compile (build) entire project.
	Run simulation. See Running a Single-Project Simulation.

	Stop simulation. See Stopping a Simulation.
	Pause simulation. See Pausing a Simulation.
	Skip simulation. See Skipping a Simulation.
	Advance run by one time step (while pause is invoked).
	Take a snapshot.
	Slow simulation. See Slowing a Simulation.
<p><b>Plot Step (us)</b></p> <input type="text" value="0.01"/>	Change plot step.
	Save scenario. See Scenarios (Control Templates).
	Delete scenario.
	View scenario.
<input type="text" value="Base"/>	Scenarios template list.
	Navigate backward.
	Navigate forward.
	Navigate up to parent schematic instance.
	Undo.
	Redo.
	Select mode (includes <b>Select All</b> , as well as <b>Freehand</b> and <b>Point</b> select modes)

-  Pan mode
-  Launch search utility.
-  Invoke wire mode.
-  Zoom in one step. See Zoom.
-  Zoom out one step.
-  Zoom control list box
-  Zoom extents
-  Zoom rectangle

## Project

The project tab simply contains an assortment of fields and buttons that allow for easy access to the project settings options.



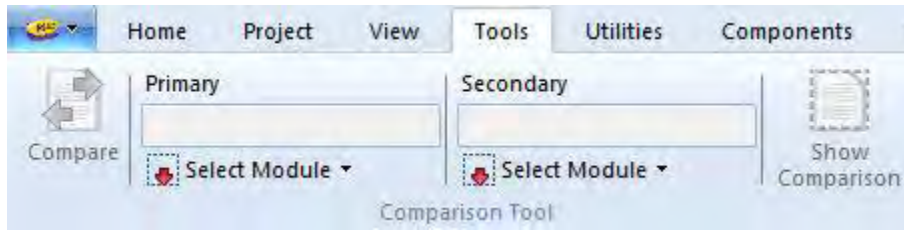
## View

The view tab contains an assortment of buttons and check boxes that are mainly for window control, as well as easy access to the modules canvas settings options.



## Tools

The tools tab provides a place to house button for features inherent to PSCAD. As of PSCAD v4.6.0, the only controls here are that for the Comparison Tool.



## Utilities

The utilities tab provides easy access to utility programs used that complement PSCAD.

### Button

### Description



Fortran Medic. Use this utility to help diagnose and correct installation settings. Contact the support desk for help.



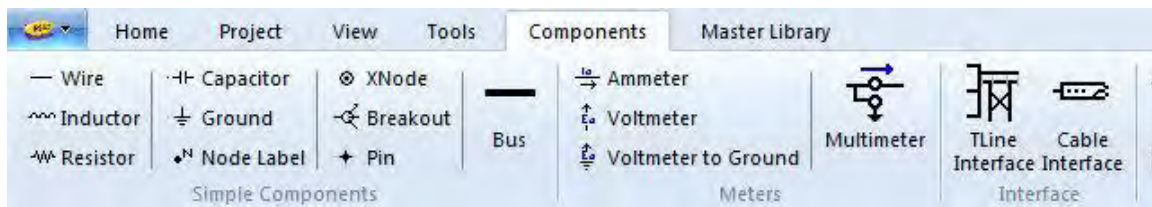
LiveWire. This is an offline plotting program which can be used to study waveform data output by PSCAD. Contact the sales desk for information on how to purchase.



License Keys. Use this button to launch the Licence Update utility (used to enter or upgrade license keys).

## Components

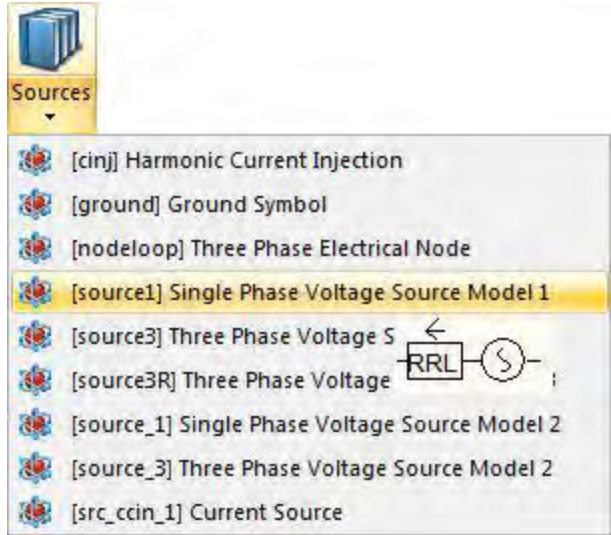
The components tab provides easy access to the most commonly used components in PSCAD.



**NOTE:** This tab is visible only when viewing the Schematic tab.

## Models

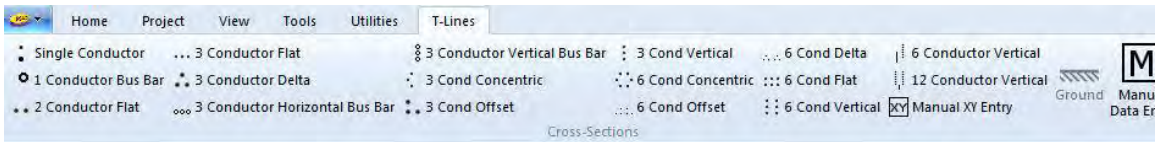
The models tab provides easy access to all components in the master library. The components are sorted into the same categories as they are graphically represented in the master library main page. Simply click on any of the icons to get a list of components. The component graphic is displayed in a flyby window for ease in selection.



**NOTE:** This tab is visible only when viewing the Schematic tab.

## T-Lines

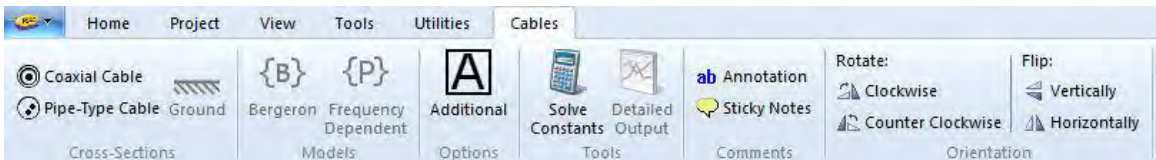
The t-lines tab is specific to overhead transmission lines only.



**NOTE:** This tab is visible only when editing transmission line definitions.

## Cables

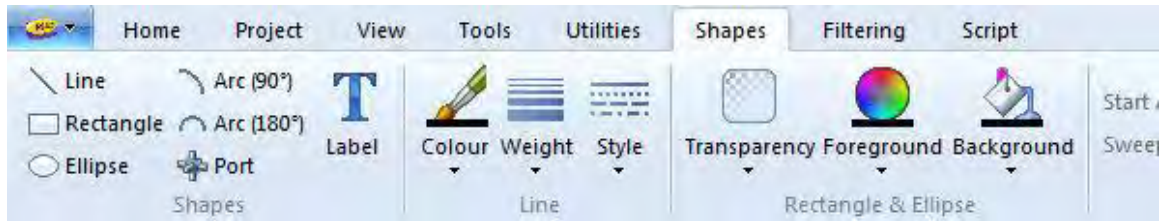
The cables tab is specific to underground cables only.



**NOTE:** This tab is visible only when editing cable definitions.

## Shapes

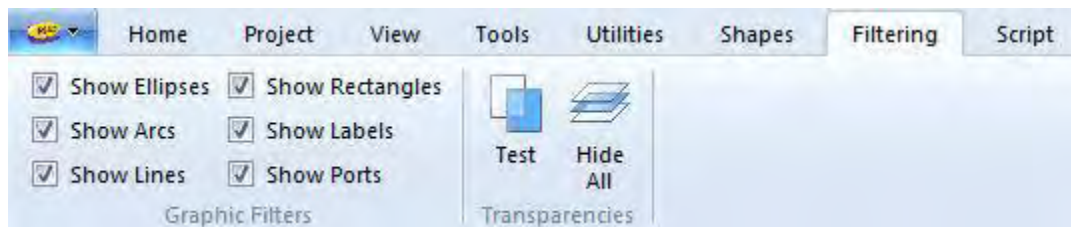
The shapes tab deals with graphical objects in the *Graphics* section of The Definition Editor.



**NOTE:** This tab is visible only when viewing the Graphic, Parameters or Script tabs.

## Filtering

The filtering tab contains functions dealing with graphical layers.



**NOTE:** This tab is visible only when viewing the Graphic, Parameters or Script tabs.

## Script

The script tab is used only when viewing component Script.



**NOTE:** This tab is visible only when viewing the Graphic, Parameters or Script tabs.

## Minimizing the Ribbon

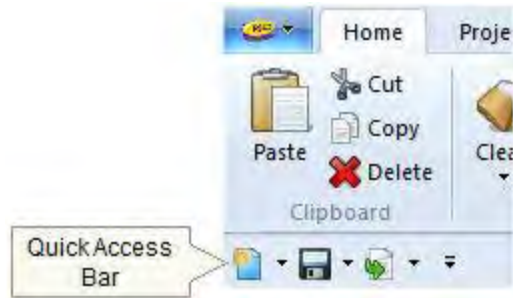
If more canvas space is desired for viewing your circuit, you can minimize or maximize the ribbon by simply pressing the down/up arrow in the top-right corner.



Or, right-click on the ribbon and select **Minimize the Ribbon**.

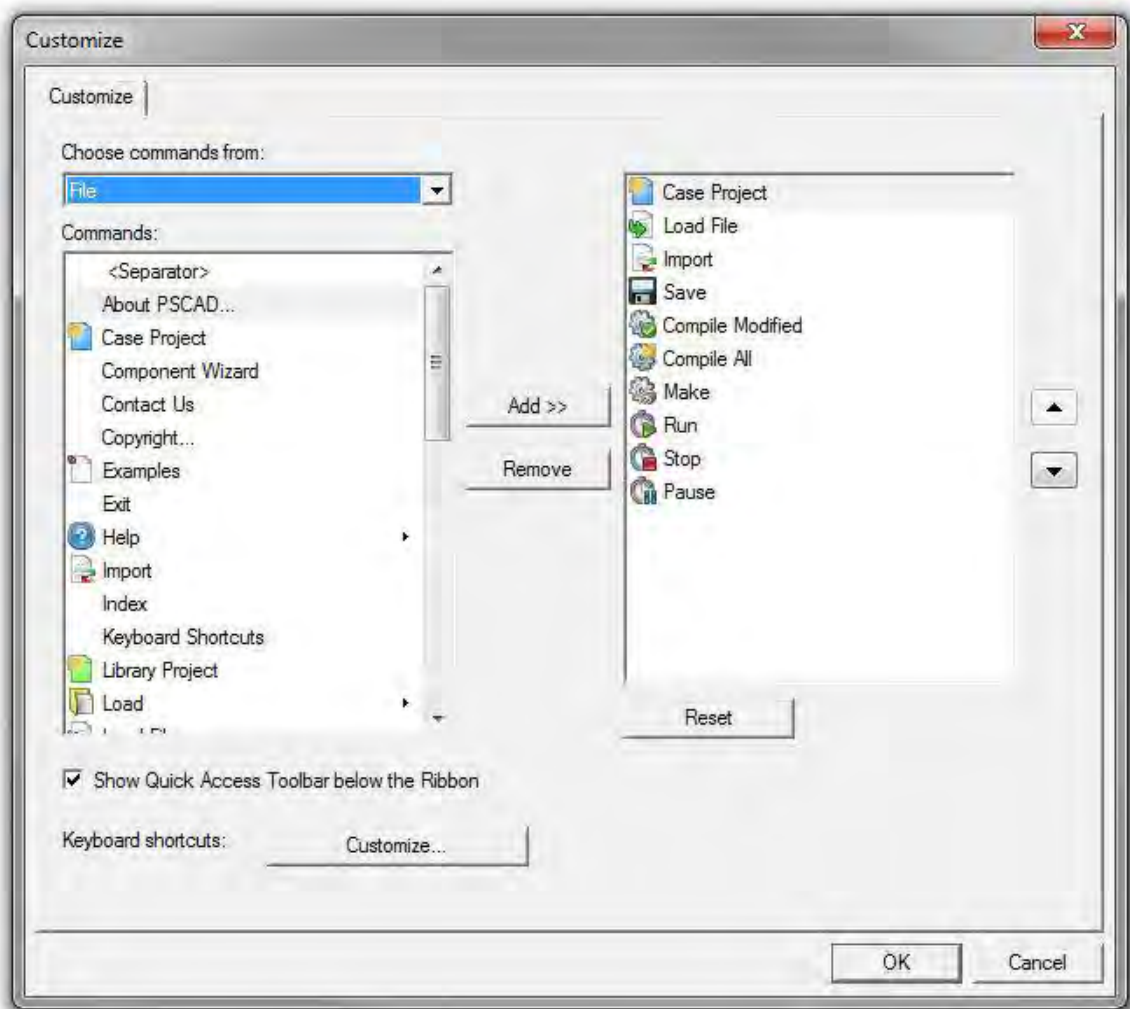
## Quick Access Bar

The ribbon control bar comes complete with a customizable button bar called the *Quick Access Bar*. Any function that is defined within the ribbon control bar can be added to this bar for quick access.



## Adding/Removing Buttons

To add a button to the quick access bar, right-click on the bar and select **Customize Quick Access Toolbar...** This will bring up the Customize dialog:



Click the **Add** or **Remove** buttons to add or remove selected commands.

# Keyboard Shortcuts

There are many keyboard shortcuts (or hotkeys) available. Hotkeys help to reduce the 'amount of clicks' required for performing a specific task. The following tables list the available keyboard shortcuts.

## General

Note that the 'Ctrl' key may be dropped where indicated (i.e. [Ctrl +]) if **Enable Cut/Copy/Paste without 'Ctrl' Key** in the Environment category of the *Application Options* is selected.

Shortcut	Description
[Ctrl +] x	Cut selection
[Ctrl +] c	Copy selection
[Ctrl +] v	Paste selection
r	Rotate selection right
l	Rotate selection left
m	Mirror selection
f	Flip selection
s	Set component sequence. Note that the component must be selected first before pressing this key.
Ctrl + a	Select all
Ctrl + z	Undo
Ctrl + y	Redo
Ctrl + o	Load project
Ctrl + n	New project
Ctrl + s	Save active project
Ctrl + g	Global Substitutions dialog



<b>Ctrl + u</b>	Unload the selected project from the workspace
<b>Ctrl + w</b>	Invoke/cancel wire mode
<b>Esc</b>	Cancel action
<b>+</b>	Zoom in
<b>-</b>	Zoom out
<b>Ctrl + Shift + Left Mouse Hold</b>	Pan (dynamic scroll)
<b>Ctrl + Left Double Click</b>	Edit definition of selected component or module
<b>Ctrl + Right Click</b>	Invoke the library pop-up menu system
<b>Left Double Click</b>	Edit properties of selected component or module
<b>Backspace</b>	Navigate backwards
<b>Ctrl + Backspace</b>	Navigate up to parent canvas
<b>F5</b>	Refresh canvas
<b>F1</b>	Invoke the help system
<b>← → ↑ ↓</b>	Scroll canvas horizontally and vertically

## Wires

To apply any of the following shortcuts, simply move your mouse pointer over a wire.

<b>Shortcut</b>	<b>Description</b>
<b>v</b>	Insert a wire vertex
<b>i</b>	Reverse wire vertexes
<b>d</b>	Decompose wire

## Plotting

To apply any of the following shortcuts, simply move your mouse pointer over a plot area. Note that in some instances, the graph must be selected.

<b>Shortcut</b>	<b>Description</b>
<b>Insert</b>	Insert an overlay graph
<b>+</b>	Zoom in to graphs
<b>-</b>	Zoom out of graphs
<b>p</b>	Zoom previous
<b>n</b>	Zoom next
<b>x</b>	Zoom x-axis extents
<b>e</b>	Zoom x-axis limits
<b>y</b>	Zoom y-axis extents
<b>u</b>	Zoom y-axis limits
<b>r</b>	Reset all extents
<b>b</b>	Reset all limits
<b>Ctrl + Left Mouse Hold</b>	Zoom horizontal aperture
<b>Shift + Left Mouse Hold</b>	Zoom vertical aperture
<b>Left Mouse Hold</b>	Zoom to box (simultaneous horizontal and vertical)
<b>g</b>	Toggle grid lines
<b>i</b>	Toggle tick marks
<b>k</b>	Toggle curve Glyphs
<b>m</b>	Show or hide markers
<b>x</b>	Set X marker
<b>o</b>	Set O marker
<b>l</b>	Toggle marker lock-step

<b>f</b>	Toggle frequency/delta view
<b>q</b>	Show x-intercept
<b>w</b>	Show y-intercept
<b>c</b>	Show cross-hairs (follows curve traces)
<b>Space Bar</b>	Switch curves while in cross-hair mode
<b>← →</b>	Graph frame dynamic aperture adjustment

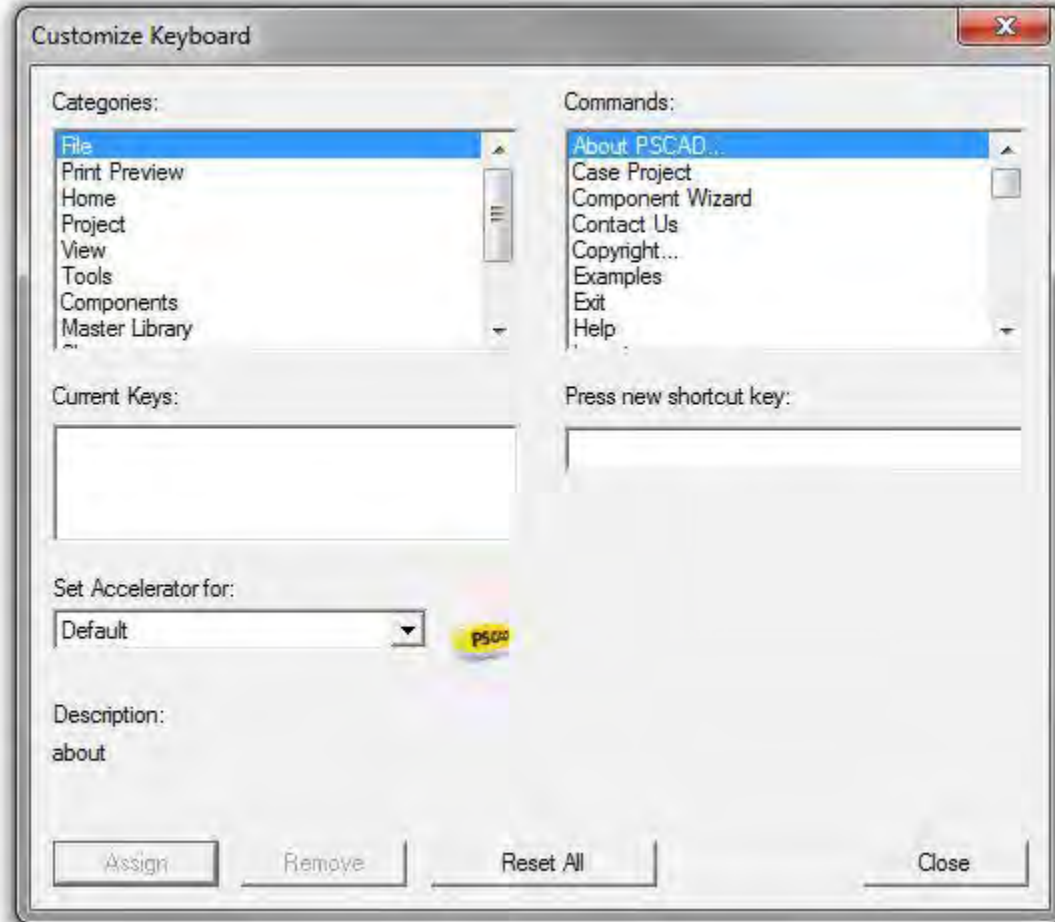
## Layers

To apply any of the following shortcuts, you must be working within the Layers Pane.

<b>Shortcut</b>	<b>Description</b>
<b>Insert</b>	Add layer
<b>Delete</b>	Remove layer
<b>Ctrl + l</b>	List components
<b>Ctrl + k</b>	Select components
<b>Ctrl + m</b>	Merge layers
<b>Ctrl + r</b>	Rename layer
<b>Hold h</b>	Highlight components
<b>↑ ↓</b>	Move the selection within the layers table
<b>Shift + ↑ Shift + ↓</b>	Extend the selection within the layers table
<b>Home and End</b>	Move the selection to the first and last Layers respectively
<b>PageUp and PageDown</b>	Move the selection by the size of the window
<b>e, d and i</b>	Change the state to Enabled, Disabled and Invisible respectively

## Customizing Shortcuts

To customize shortcuts, right-click on the ribbon control bar and select **Customize Quick Access Toolbar....** This will bring up the Customize dialog. Click the **Customize...** button to bring up the **Customize Keyboard** dialog.



## PSCAD Temporary Folder

When a case project is compiled, several files are created and placed into a temporary directory located in the same folder as the project file (\*.pscx) itself. The directory is named by appending an extension to the project filename. This extension is dependent on the compiler used to build the project. For example, if a case project has a filename *test.pscx*, a temporary directory called *test.gf42* will be created when the project is compiled with the GFortran (v4.2.1) compiler. The temporary directory will contain all files created by both the compiler and PSCAD.

The naming convention for the temporary directory is as follows for all supported compilers:

- **GFortran 95 (v4.2.1):** \*.gf42
- **GFortran 95 (v4.6.2):** \*.gf46
- **Intel® Visual Fortran Compiler 9 to 11:** \*.if9
- **Intel® Visual Fortran Composer XE 2011 to 2014:** \*.if12
- **Intel® Visual Fortran Compiler 15:** \*.if15 (64-bit), \*.if15\_x86 (32 bit)

All files located in the temporary directory can be viewed in an organized manner in the primary workspace window Projects branch. You may also clean out the entire temporary directory at any time (see Cleaning the Temporary Directory). Note that both of these features are based on the currently selected compiler. That is, if Intel® is the current compiler, the files viewed in the *Projects* branch, will be those from the \*.if9, \*.if12 or \*.if15 temporary folder. Same goes for cleaning the temporary folder.

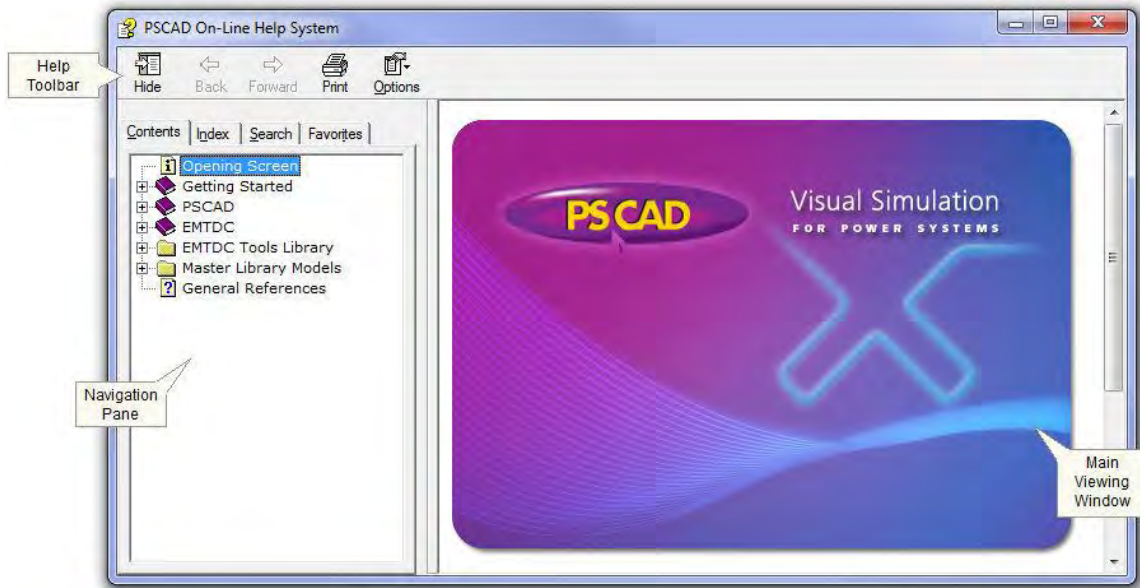
## PSCAD On-Line Help System

The PSCAD on-line help is a single compiled file in *Microsoft HTML Compiled Help* format. This file includes its own browser, and features a full-text search engine and comprehensive index.

In addition to the complete manual set, the on-line help file includes component specific documentation (not included in the manuals). See Accessing the Online Help System details on accessing help.

### Look and Feel



The help system is invoked by pressing the **F1** key. A window similar to that shown below should appear.






There are three main sections of the help system. Across the top is the help toolbar, and below this are the navigation pane on the left, and the main viewing window on the right.

### The Help Toolbar

The help toolbar contains the most commonly used functions. These are summarized below:

Button	Description
 Hide	Hides navigation pane
 Back	Moves backward in the viewed topic list

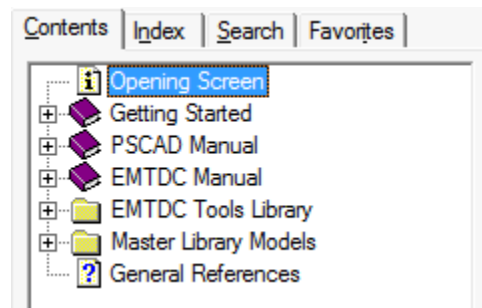
 Forward	Moves forward in the viewed topic list
 Print	Prints currently viewed topic
 Options	Provides some options to allow the user to customize the online help viewer

## The Navigation Pane






The navigation pane is separated into four sections: *The Table of Contents (TOC)*, the *Index*, the *Search* engine and the *Favorites* section. To access any section, click the corresponding tab at the top of the pane.

### Contents

The contents tab contains the help system TOC. Left-click on the corresponding [+] or [-] icons (directly to the left of each branch respectively), to expand or collapse any branch of the tree.



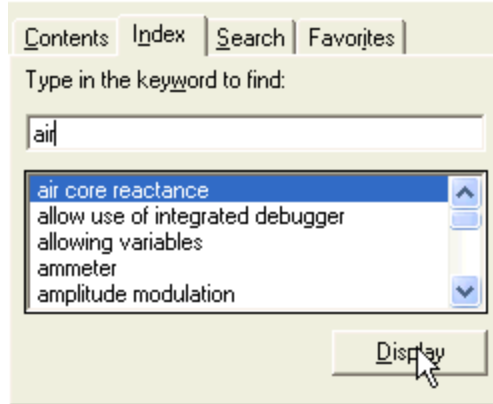
There are various icons associated with the TOC, as described below:

-  **Topic:** A help page containing information about a specific topic.
-  **Book:** A folder that contains one or more topics, which is also part of a printed manual.
-  **Folder:** A folder that contains one or more topics, which exist only in the online help.
-  **New Topic:** A topic that has been added to the online help since the last release.
-  **Unfinished Topic:** A topic that is included in the online help, but is not complete.

### Index

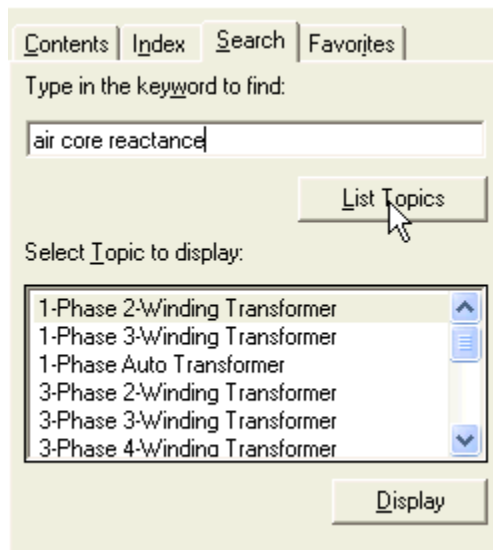
The index contains a large list of keywords, in alphabetical order, that have been associated with various topics within the help system. You have the choice of either scrolling through this list directly, or you may type in a word into the input field near the top of the index pane, in order to directly look up a word.

In the figure below, for example, a user is looking up *air core reactance*. Entering air will bring the list window to the nearest matching word.



## Search

The search section allows you to enter in a single word or a text string, and then list the topics that contain the word or string. Simply enter the text string and click the **List Topics** button. Any topics containing the string will be displayed in the output field as shown below.



**NOTE:** The help system search engine will search all text in each topic. Text that exists within embedded images will not be considered.

To bring up a listed topic into the main viewer, simply left double-click, or select it and click the **Display** button.

## Favorites

The Favorites section allows users to bookmark their favorite help pages. To add a favorite topic to the favorites list, simply left click the **Add** button. To remove a topic from Favorites, select it and left click the **Remove** button. To display a topic from the favorites list, select it and left click the **Display** button.

## The Main Viewer

The contents of the main viewing window are interactive in the same way as in a normal HTML web page. Depending on the page viewed, there may be hyperlinks and other types of functionality, normally seen while navigating the web. Usually, text with associated hyperlinks and other functions are shown in blue color and may or may not be underlined.

## Tutorial Projects

The PSCAD software package is installed complete with a directory of tutorial projects, which contains a variety of simple cases to illustrate various features. This directory is located within the installation directory under *...examples\tutorial*.

The tutorial projects described in this section are mainly meant to illustrate the use of PSCAD and hence, they are simple cases from an electrical engineering point of view. If you are a first time user, go through all the tutorial cases in the order they are listed below.

Once you are familiar with these projects, have a look at the more detailed projects contained within the main examples directory. These include a variety of practical examples ranging in topics from machines to FACTS devices.

### Voltage Divider

vdiv_1.pscx	A simple voltage divider circuit with a resistor and a resistive source. Demonstrates how to assemble a circuit, monitor voltage and current, and run the simulation.
-------------	---

### Fast Fourier Analysis

fft.pscx	Shows the use of Fast Fourier analysis component to perform online fft on signals.
----------	--

### Simple AC System with a Transmission Line

simpleac.pscx	A simple AC system with transmission lines. Introduces transformers, transmission lines, and the concept of subsystems in PSCAD.
---------------	--

### Use of Control Arrays

pagearray.pscx	Demonstrates the use of control arrays and how to export electrical nodes to other pages so that a circuit can be modeled on multiple pages - even if you do not have transmission lines.
----------------	---

### Use of Slider, Switch, Button, and Dial

inputctrl.pscx	Shows the use of dynamic input devices: Slider, Switch, Push Button and the Dial.
----------------	---

### Interpolation

interpolation.pscx	A simple case illustrating the use of interpolation.
--------------------	--



## Generating a Legend and using PSCAD Macros

legend.pscx                      Shows how to generate a legend. Also illustrates the use of PSCAD macros in a legend.

## Chatter Elimination

chatter.pscx                      Defines chatter and gives examples on chatter elimination techniques.

## Multiple Run

multirun.pscx                      A simple example to demonstrate the use of multiple run features to find the maximum overvoltage due to a fault in a 3-phase power system. The point on wave and type of fault is varied to determine the worst-case overvoltage.



## Chapter 5

# Operations and Feature Overview

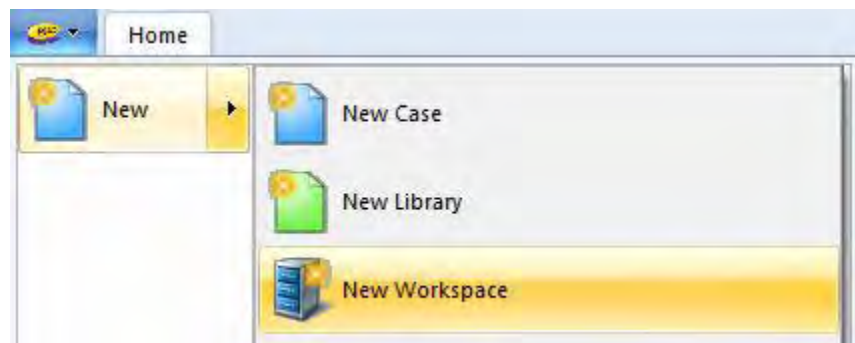
This chapter is intended to be a quick reference for basic operations, as well as to provide an overview of key features in PSCAD. In many cases, the features introduced here are discussed in greater detail in other chapters. If you are a new user, the following topics can be helpful in familiarizing yourself with PSCAD.

This chapter includes a tutorial entitled *Creating a New Project*, which is designed to jump-start the learning process and to get you using the application. If you have not done so already, try the tutorial entitled *My First Simulation* in Chapter 4 of this manual as well.

## Workspace

### Creating a New Workspace

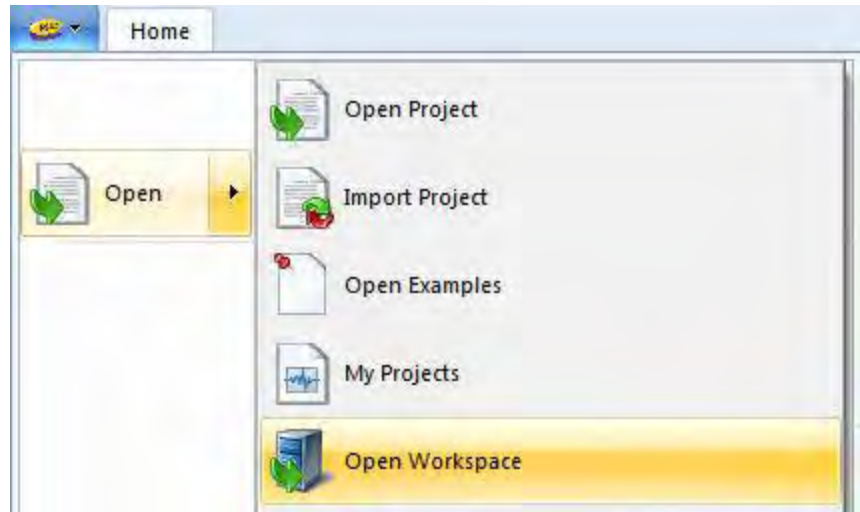
Click the PSCAD tab in the ribbon control bar, hover mouse pointer over **New**, and select **New Workspace**.



Creating a new workspace action will immediately unload the current workspace and create a new one. If the old workspace had been modified, you will be asked to save it. A new workspace called *Untitled* will then appear in the workspace window.

### Loading a Workspace

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Open** and select **Open Workspace**.



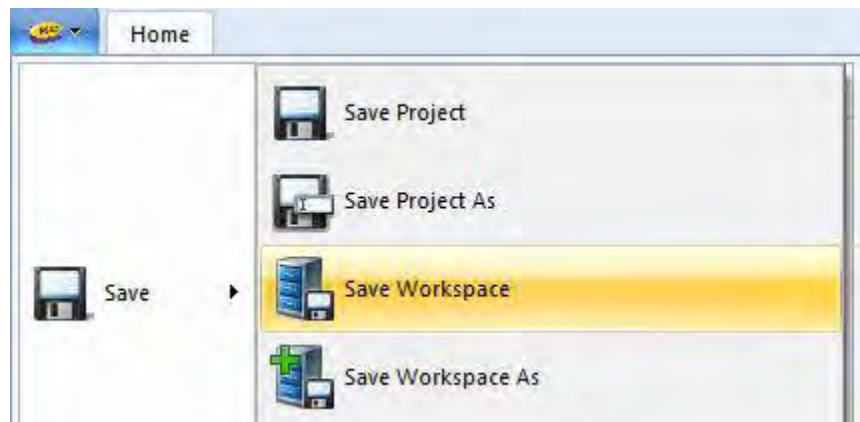
The *Open File* dialog window will open with default file types as **Workspaces (\*.pswx)**. Navigate to the desired workspace file and select it so that its name appears in the **File name** field. Click the **Open** button to open the workspace. If the old workspace has been modified, you will be asked to save it. The old workspace is first unloaded, and then the requested workspace will appear in the workspace window.

## Drag and Drop

Workspaces may also be loaded via drag and drop. See for Loading Projects and/or Workspaces more details.

## Saving a Workspace

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Save** and select **Save Workspace**.



You can also save by right-clicking on the workspace branch within the workspace window, and selecting **Save** from the pop-up menu.

## Save Workspace As...

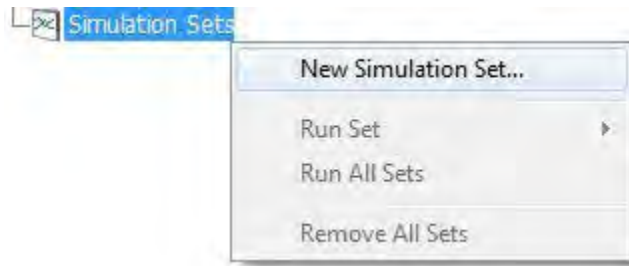
Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Save** and select **Save Workspace As**.

You can also save by right-clicking on the project title within the workspace window, and selecting **Save As** from the pop-up menu.

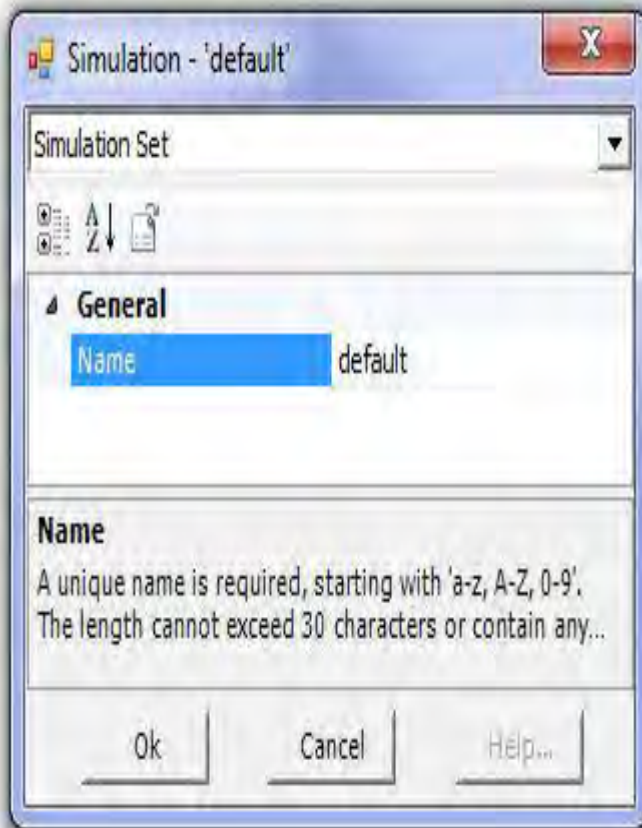
In either case, the *Save Workspace* dialog window will open with a default **Save as type** as **Workspaces (\*.pswx)**. Change the file name in the **File name** field. Click the **Save** button to save the project under a different file name. The workspace name should then change in the workspace window, indicating that the workspace was indeed renamed.

## Adding a Simulation Set

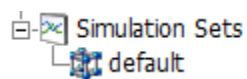
Right-click on the *Simulation Sets* branch in the Workspace Primary window and select **New Simulation Set**.



When the *Simulation Set* dialog displays, enter a name in the *Name* field and then select **OK**.

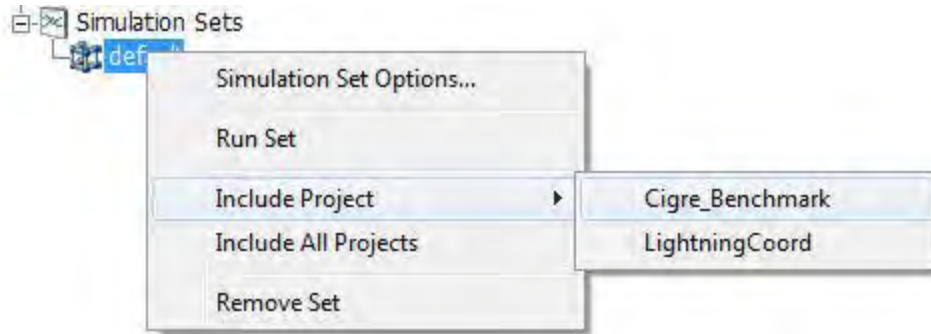


A new simulation set will be created, and will appear under the *Simulation Sets* branch.



## Adding a Project to a Simulation Set

Right-click on a simulation set in the Simulation Sets branch and select either **Include Project** or **Include All Projects**.



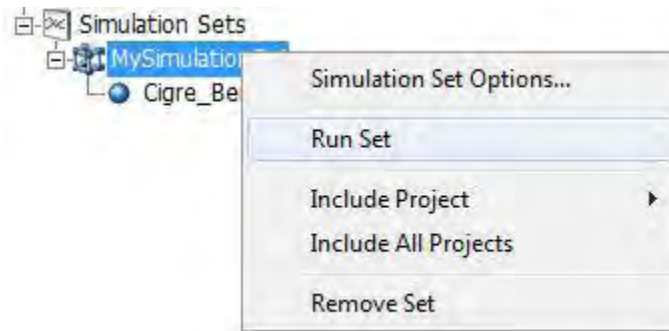
If **Include All Projects** is chosen, all projects loaded under the Projects branch in the workspace will be added as tasks within that given simulation set. Alternatively, you may add individual projects from a drop list of loaded projects.

## Running Simulation Sets

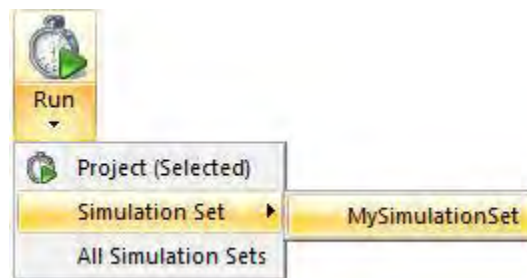
Running simulations sets can be accomplished via a couple of different ways.

To run an individual simulation set, perform one of the following:

- Right-click on the *Simulation Set* and select **Run Set**.



- Click the **Run** button drop list in the ribbon control bar and select **Simulation Set**.



To run all simulation sets, click the **Run** button drop list in the ribbon control bar and select **All Simulation Sets**. When done so, each simulation set will run sequentially according to the order they appear within the Simulation Sets branch.

See also:

Running a Single-Project Simulation

## Pausing a Simulation Set

Press the **Pause** button in the **Home** tab of the ribbon control bar to pause an entire simulation set run.



See also:

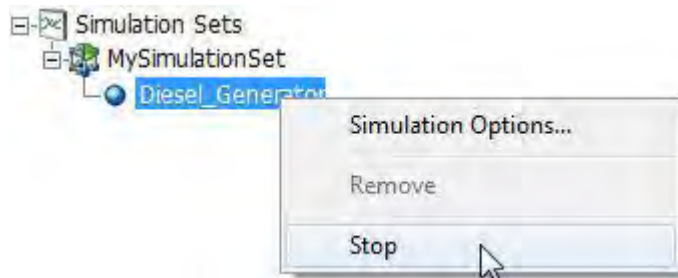
Pausing a Simulation

## Stopping a Simulation Set

Press the **Stop** button in the **Home** tab of the ribbon control bar to stop an entire simulation set run.



Single project simulations inside a simulation set may also be stopped via the simulation context menu. Note however, that only projects with their run configuration set to standalone may be stopped using this method. If set as master or slave, the entire set must be stopped.



See also:

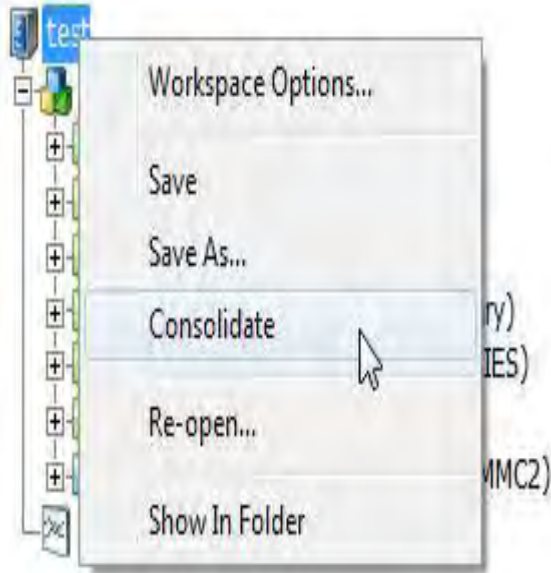
Stopping a Simulation

## Consolidating a Workspace

A workspace can contain multiple projects, which may or may not link with other dependent files that are stored in multiple folders all over the hard drive. This makes it very difficult to transfer an entire workspace between computers or users.

Workspaces may be quickly and efficiently consolidated, so that all projects and associated dependent files will be automatically moved and organized into a specified, local folder. To consolidate a workspace, simply right-click on the workspace branch and select **Consolidate**.

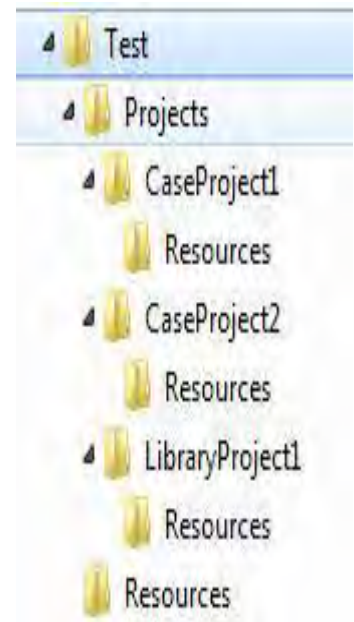




The consolidator will organize all associated files according to dependencies and automatically adjust additional source and linked file paths in all projects. It places a new workspace (\*.pswx) file in a top-level folder along with the creation of a *Projects* and a *Resources* folder.

Name	Date modified	Type	Size
Projects	25/04/2014 10:47 AM	File folder	
Resources	25/04/2014 10:47 AM	File folder	
test.pswx	25/04/2014 10:47 AM	PSWX File	3 KB

Top Level Folder



Folder Hierarchy

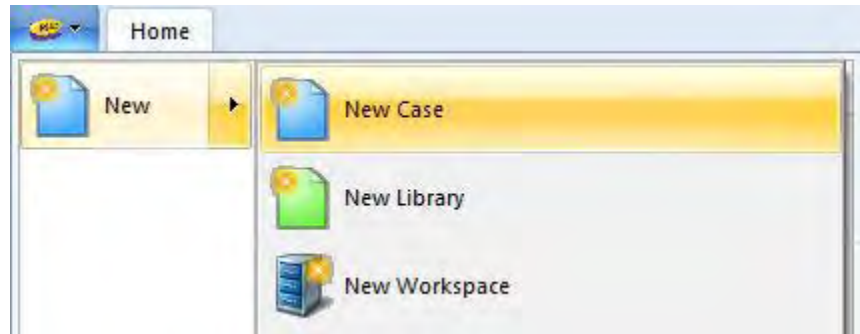
The *Projects* folder will contain a unique sub-folder to house each library and case project that are part of the original workspace. Each of these folders will contain a *Resources* sub-folder to hold any dependent files related to that project, including linked binaries and additional source.

All projects in the workspace can be built and run without any manual adjustment following a consolidation.

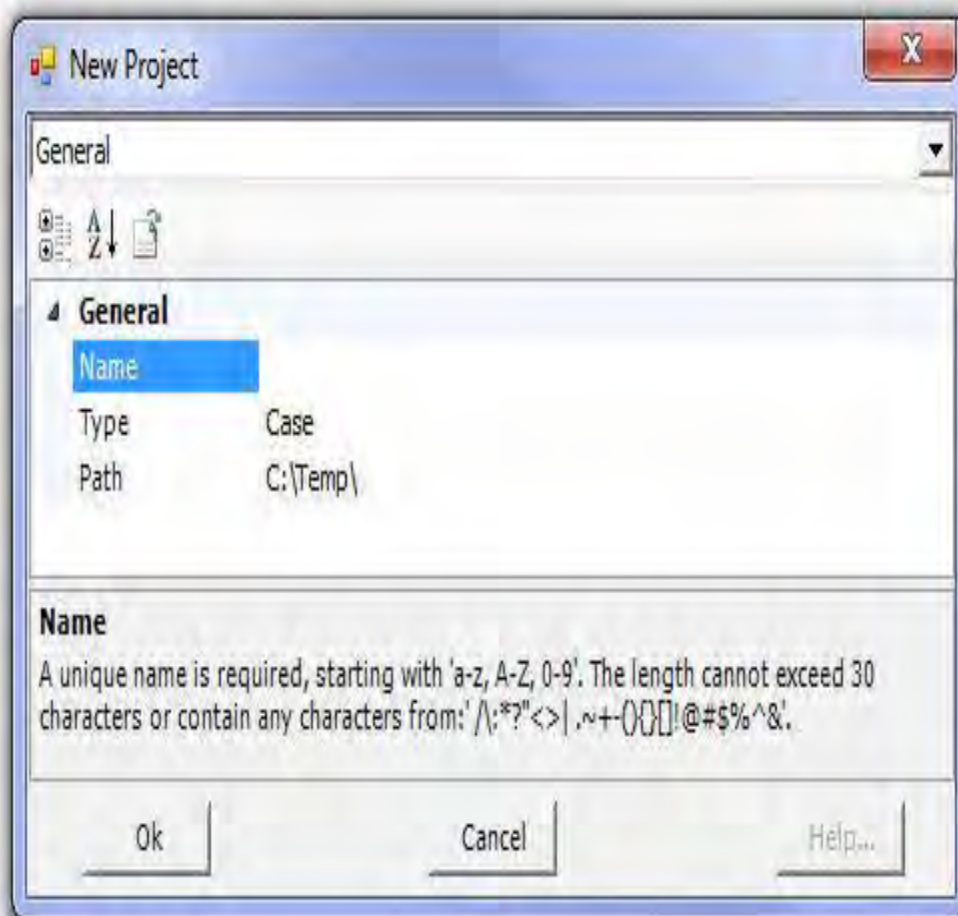
## Projects

# Creating a New Project

Click the PSCAD tab in the ribbon control bar, hover cursor over **New**, and then select **New Case** or **New Library**; or simply pressing **Ctrl + n** on your keyboard.



The *New Project* dialog will appear.



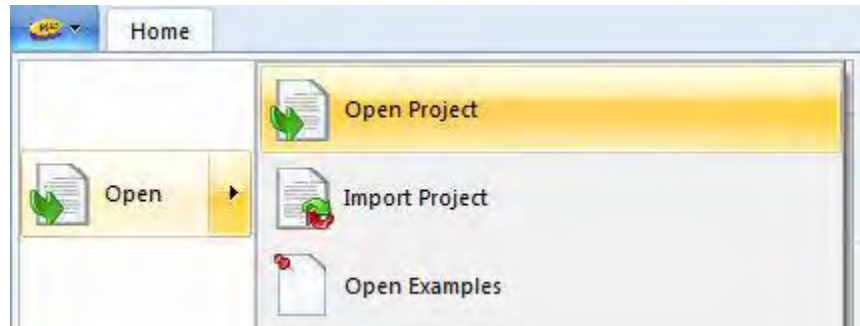
The input fields for this dialog are described as follows:

- **Name:** The name of the project. Initially when a project is created, this name will become both the namespace name and the file name of the project. Only the file name can be modified after the project is created (the project namespace is kept synchronized with the filename for case projects). See Editing Project Settings in this chapter for details.
- **Type:** Select a Case or Library type project.
- **Path:** Select the path to the folder, in which the new project will be created and stored.

Enter a name for the project in the **Name** field. A new project will appear in the workspace window. Note the rules for the name format in the *New Project* dialog help box.

## Loading a Project

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Open** and select **Open Project**.



The *Open File* dialog window will open with the default file types **Cases and Projects (\*.pscx, \*.pslx)**. Navigate to the desired project and select it so that its name appears in the **File name** field. Click the **Open** button to open the project. The project name should appear in the workspace window, indicating that the project was loaded.

You can also bring up the *Open File* dialog by simply pressing the **Open** button in the quick access bar, by pressing **Ctrl + o** on your keyboard or by right-clicking on the Projects branch in the workspace and selecting **Open Project...**

## Recent Files

You can load projects that have been loaded previously by pressing the PSCAD tab in the ribbon control bar. By default the recent files list will display a list of previously loaded and saved projects. Select the desired project from the list to load it.

## Examples Folder

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Open** and select **Open Examples**. The *Open File* dialog window will open directly within the PSCAD application examples folder. Note that examples are located at C:\Users\Public\Public Documents\PSCAD[version]\Examples.

## Open My Projects

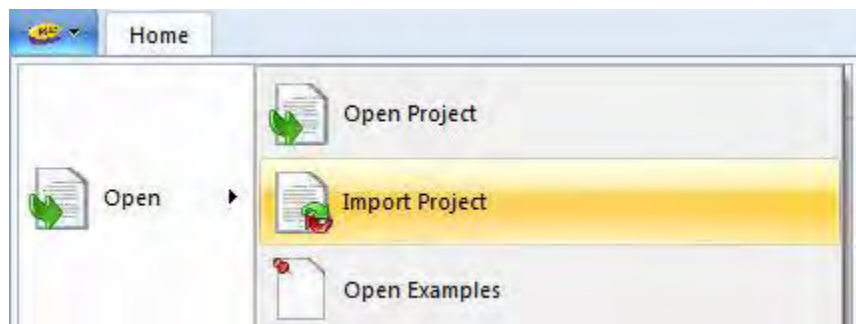
Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Open** and select **Open My Projects**. The *Open File* dialog window will open directly within the user projects folder.

## Drag and Drop

Projects may also be loaded via drag and drop. See for Loading Projects and/or Workspaces more details.

## Importing a Project

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Open**, and select **Import Project**.



The *Open File* dialog window will appear with v4.1 or v4.2 default file types (\*.psc, \*.psl). Navigate to the desired project and select it so that its name appears in the **File name** field. Click the **Open** button to import the project. The project should appear in the workspace window, indicating that the project was indeed imported.

Importing a project, as opposed to loading one, is the act of both loading and converting an older style project (i.e. \*.psc or \*.psl) into the new XML-based file format (\*.pscx/\*.pslx). For more information on the import process and some possible pitfalls see *Converting PSCAD v4.1 and v4.2 Projects to X4* in the chapter entitled *Migrating from Older Versions*.

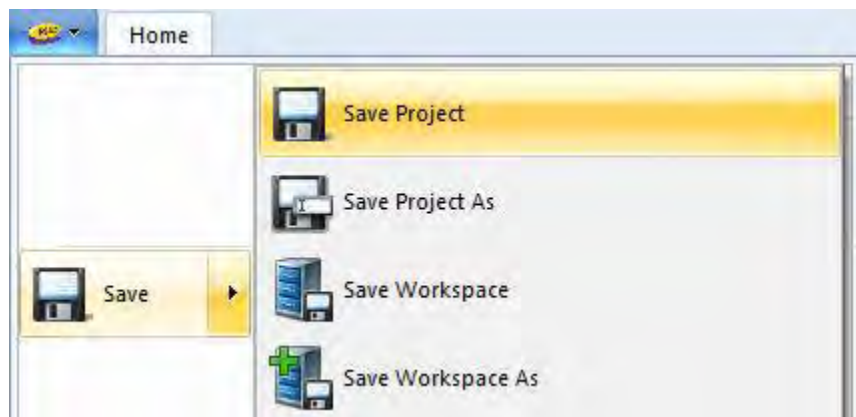
## Drag and Drop

Projects may also be imported via drag and drop. See for Loading Projects and/or Workspaces more details.

## Saving a Project

Save Project As...

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Save** and select **Save Project**.



You can also save the project by pressing **Ctrl + s** on your keyboard, by right-clicking on the project title within the workspace window, and selecting **Save** from the pop-up menu, or by right-clicking on the Projects branch in the workspace and selecting **Save All Projects...**

## Save Project As...

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Save** and select **Save Project As**.

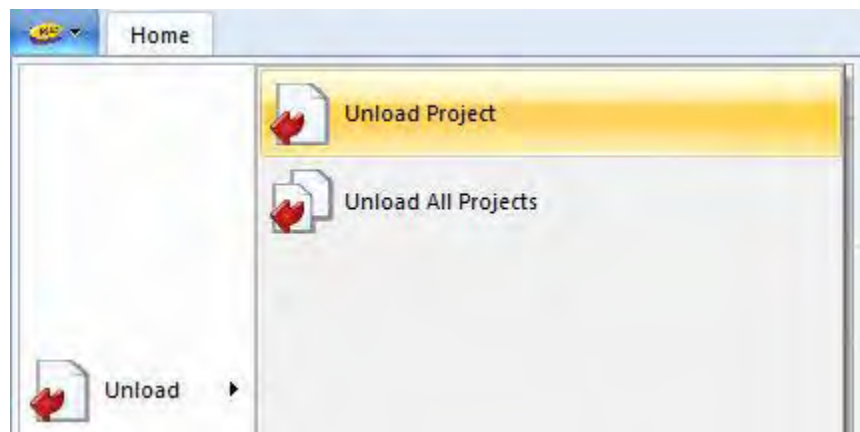
You can also save the project by right-clicking on the project title within the workspace window, and select **Save As** from the pop-up menu.

In either case, the *Save Project As* dialog window will open with a default **Save as type** as **Project File (\*.pscx)** or **Project File (\*.pslx)** for case or library project respectively. Change the file name of the project in the **File name** field. Click the **Save** button to save the project under a different file name. The project name should then change in the workspace window, indicating that the project was indeed renamed. Note that the project namespace is synchronized with the filename at this point if saving a case project.

## Unloading a Project

Unload All Projects

To unload one or more projects, Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Unload** and select **Unload Project**.



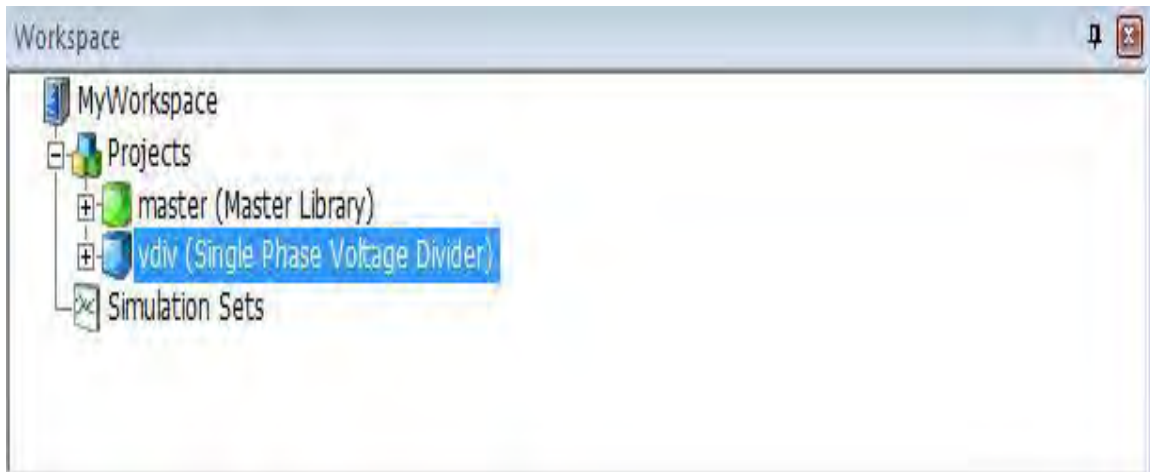
You may also unload a project by either right-clicking on the project title in the workspace window and selecting **Unload** from the pop-up menu, or by selecting the project in the workspace window and pressing **Ctrl + u** on your keyboard.

## Unload All Projects

Click the PSCAD tab in the ribbon control bar, hover the mouse pointer over **Unload** and select **Unload All Projects**.

## Opening and Viewing a Project

To open a project for viewing, simply select the project in the workspace window.



Once opened, the last page viewed when the project was saved will appear in the definition editor (Schematic window).

## Navigating Through an Open Project

Once a project has been opened, there are many navigational features available to help you efficiently navigate about the project.

### Scroll Bars

Standard vertical and horizontal scroll bars are available in all windows. These are located at the right-most and bottom-most edges of the open window respectively.



### Arrow Keys

You can use the arrow buttons on your keyboard to scroll both horizontally and vertically while in Schematic view.

### Mouse Wheel Scroll

Both the schematic and the graphic canvases may be scrolled using the mouse wheel. To vertically scroll, simply roll the mouse wheel. To scroll horizontally, hold down the **Shift** key and roll the mouse wheel.

### Panning (Dynamic Scroll) Mode

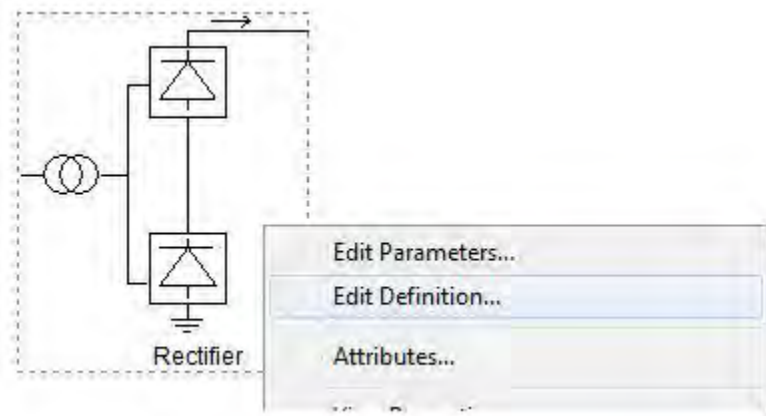
The panning feature allows you to scroll through the *Schematic* or *Graphic* tab windows in a fluid motion. You can invoke the panning mode by one of the following methods:

- On a blank portion of the page, press and hold the *Ctrl* and *Shift* keys at the same time, then click and hold the left mouse button (**Ctrl + Shift + left mouse hold**). Moving the mouse will then allow panning through the page.

- Press the **Pan** button in the **Home** tab of the ribbon control bar to invoke panning mode. To indicate that you are in panning mode, the mouse pointer will change into a hand shape. Press **Esc** on the keyboard or press the **Pan** button again to exit from panning mode.

## Moving In and Out of Modules

Opening the Schematic canvas of a module component (i.e. moving into) is analogous to editing the definition of the module. Therefore, right-click on the module and select **Edit Definition...**



You may also perform the same operation by double-clicking the left mouse button. However, this functionality is dependent on a setting called *Drill Down* (under the Environment category of the *Application Options* dialog) as follows:

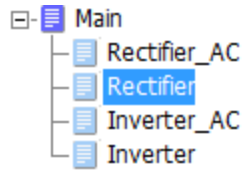
- **Double-Click:** If you want to use the old method of navigating through module components, set the *Drill Down* setting to *Double Click*. Note however that if the module possesses input parameters, the parameter dialog will not open with this action.
- **Ctrl + Double-Click:** If the *Drill Down* setting is set to *Ctrl + Double Click*, you may still navigate into the module by holding down the **Ctrl** key and left double-clicking the module component. A double-click without the **Ctrl** button in this mode opens the parameters dialog.

To move out of the current module (i.e. move back one level), press the Up to parent canvas button in the **Home** tab of the ribbon control bar.



Up Button

Instead of the **Up** button, you can also use the workspace secondary window to specifically select a module to navigate to. See *The Secondary Window* for more.



## Forward/Back Buttons

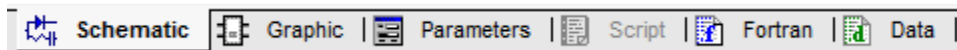
PSCAD maintains a navigational history of the user's module canvas navigation for each session. When a project is unloaded, or when a session is ended, all navigation history is lost.

Navigation history is presented in the form of **Forward** and **Back** buttons in the **Home** tab of the ribbon control bar.



## Tabs

Tabs allow you to jump from one viewing area to another. There is a tab bar included at the bottom of some windows (as shown below), including the Output Window and others.



## Zoom

Zoom features are available when working within either the *Schematic* or the *Graphic* windows. There are a few different methods available:

- In the **Home** tab of the ribbon control bar, select either the **Zoom In** or **Zoom Out** buttons, or select a percentage zoom directly from the **Zoom In/Out** drop down list.



- Press the **+** or **-** keys on your keyboard number pad.

## Zoom Rectangle

The **Zoom Rectangle** button is located in the **Home** tab of the ribbon control bar. Click this button to enter zoom rectangle mode. On the Schematic or Graphic canvas, **hold down the left-mouse button** and drag the pointer to create the desired rectangle. Release the mouse button to zoom.

## Zoom Extents

The **Zoom Extents** button is located in the **Home** tab of the ribbon control bar. Click this button to zoom to the extents of the currently viewed Schematic or Graphic canvas.



## Refresh

You can refresh the page view in either the *Schematic* or *Graphic* windows by one of the following methods:

- In the **View** tab of the ribbon control bar, click the **Refresh** button.
- Press the **F5** key on your keyboard.
- Right-click on a blank part of the page and select **Refresh**.

## Compiling and Building a Project

Running a project without compiling first is perfectly fine. If a **Run** command is issued, PSCAD will perform the **Build Modified** procedure before the run starts. See Running a Simulation for more details.

**Build** refers to the creation of the Fortran, data, and map files required for the building of a project executable file. You may choose to build the entire project (**Build**), or only modules that have been modified since a previous build (**Build Modified**). The latter choice comes in handy for very large projects, which may take minutes to compile in their entirety.

To build the entire project from scratch, select the **Build** button in the **Home** tab of the ribbon control bar:



To build only modules and transmission segments that have been modified since the last compile, select the **Build Modified** button in the **Home** tab of the ribbon control bar:



## Viewing Error and Warning Messages

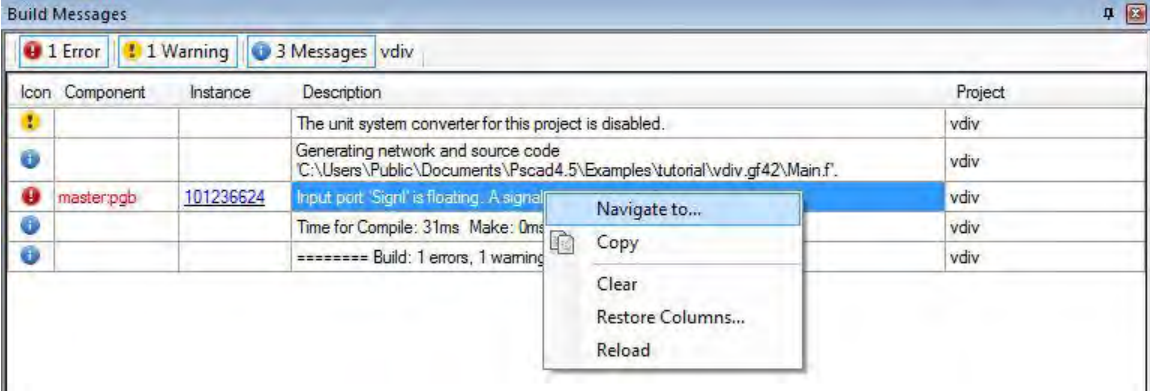
If a problem is detected when a simulation is run, all error and warning messages will appear in the Build Messages pane.

Icon	Component	Instance	Description	Project
!			The unit system converter for this project is disabled.	vdiv
i			Generating network and source code C:\Users\Public\Documents\Pscad4.5\Examples\tutorial\vdiv.gf42\Main.f.	vdiv

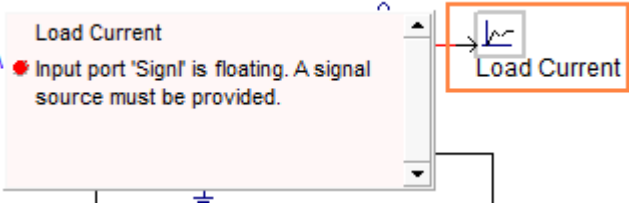
Output window messages are divided into three main groups: *Messages*, *Warnings* and *Errors*. For more on this, see Error and Warning Messages in Chapter 11 of this manual.

## Navigating to the Message Source

If an error or warning message is received, PSCAD can highlight the source of the problem. In the output window, navigate to the error or warning message, move the mouse pointer over the message and either **left-click** the message hyperlink or right-click and select **Navigate to....** The problem source will be highlighted in the Schematic window.



In the example image below, it can be seen that the voltmeter 'Vmid' has been shifted and is not measuring a specific node voltage.



# Running a Single-Project Simulation

Press the **Run** button in the **Home** tab of the ribbon control bar to run a single-project simulation.

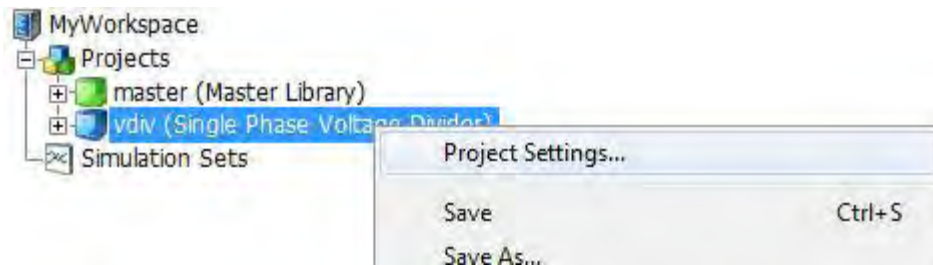


See also:

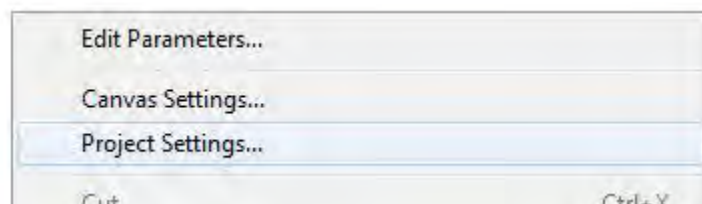
Running Simulations Sets

# Editing Project Settings

Select the desired project in the workspace window, right-click and select **Project Settings...** from the pop-up menu.



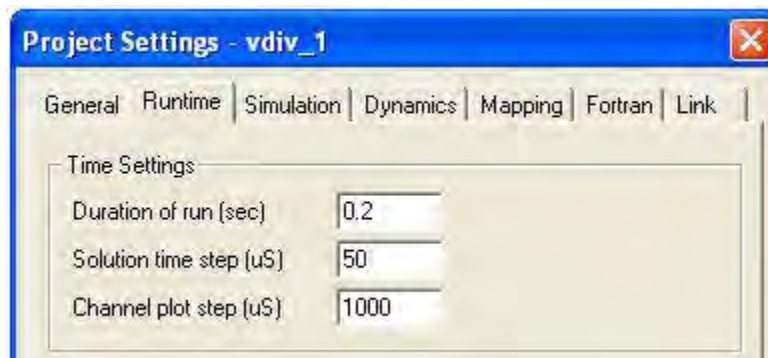
Also, you can right-click over a blank area of any project page in Schematic view and select **Project Settings...** from the pop-up menu.



In either case, a dialog window entitled *Project Settings* will open. For detailed information on this dialog, see the chapter entitled *Project Settings* in this manual.

## Changing the Run Duration

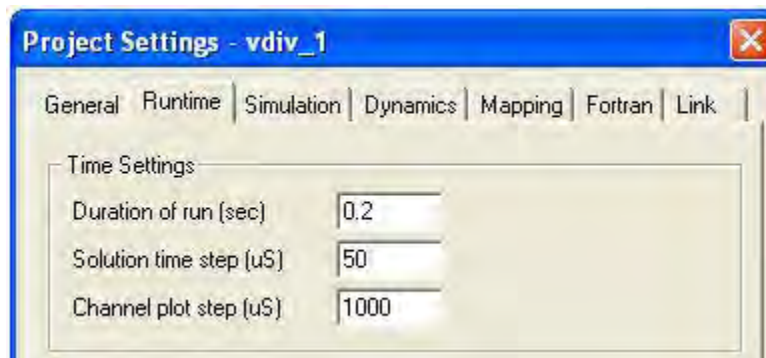
Open the *Project Settings* dialog window as described in *Editing Project Settings* above. Click the **Runtime** tab and, in the **Time Settings** area, enter a new **Duration of Run** time in seconds. Click the OK button to save changes and exit the *Project Settings* dialog.



The run duration may also be varied through the equivalent field in the **Project** tab of the ribbon control bar.

## Changing the Simulation Time Step

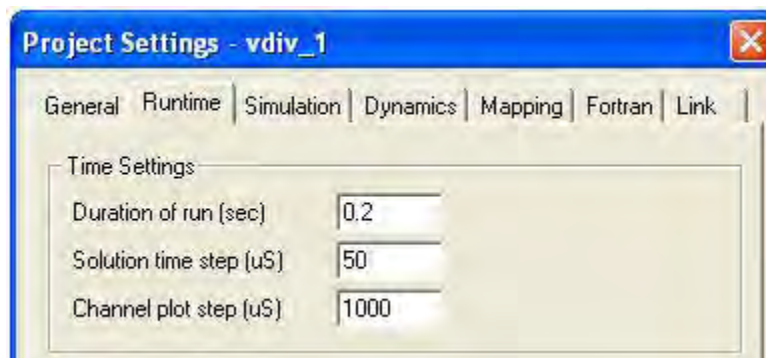
Open the *Project Settings* dialog window as described in *Editing Project Settings* above. Click the **Runtime** tab and, in the **Time Settings** area, enter a new **Solution Time Step** time in microseconds ( $\mu\text{s}$ ). Click the OK button to save changes and exit the *Project Settings* dialog.



The time step may also be varied through the equivalent field in the **Project** tab of the ribbon control bar.

## Changing the Simulation Plot Step

Open the *Project Settings* dialog window as described in Editing Project Settings above. Click the **Runtime** tab and, in the **Time Settings** area, enter a new **Channel Plot Step** time in microseconds ( $\mu\text{s}$ ). Click the OK button to save changes and exit the *Project Settings* dialog.



The plot step may also be varied through the equivalent field in the **Project** tab of the ribbon control bar. The plot step can be changed at any point during a simulation.

## Pausing a Simulation

Press the **Pause** button in the **Home** tab of the ribbon control bar to pause a single-project simulation.



See also:

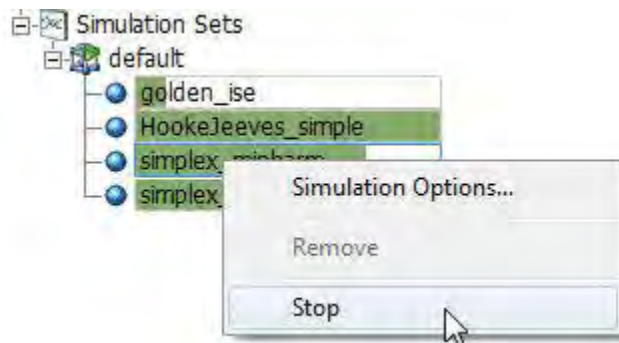
Pausing a Simulation Set

## Stopping a Simulation

Press the **Stop** button in the **Home** tab of the ribbon control bar to stop a single-project simulation.



Individual simulations inside a simulation set may also be stopped. Right click on the simulation and select Stop from the context menu.



See also:

[Stopping a Simulation Set](#)

## Skipping a Simulation

Press the **Skip** button in the **Home** tab of the ribbon control bar to skip a simulation multiple run.



Jumps to the next multiple run iteration or next coordinated run iteration if using simulation sets.

## Slowing a Simulation

Type topic text here.

Reduce plotting speed by using this slide control. This provides plotting delay of 1 ms up to a maximum of 10 ms per time step.

## Taking a Snapshot

Taking a snapshot of a simulation run is presently the only way to start an EMTDC simulation from an initialized condition. See Initialization and Initial Conditions in the EMTDC manual for more details on snapshot files.

### Manual

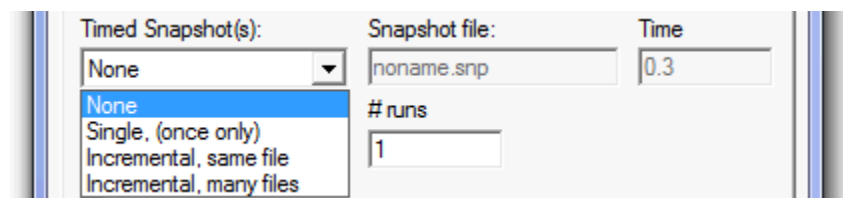
You can take a snapshot manually by pressing the **Snapshot** button in the **Home** tab of the ribbon control bar.



### Pre-Defined

Pre-defined snapshots are set before the run is started. There are a couple of different snapshots of this type available.

Open the *Project Settings* dialog window as described in Editing Project Settings above. In the **Runtime** section of the dialog, choose the snapshot type from the **Timed Snapshot(s)** drop list.



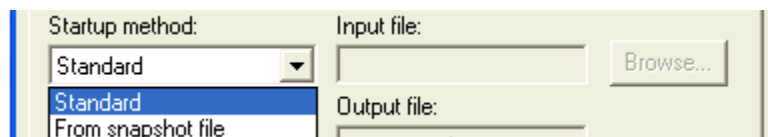
Enter a name for the snapshot file in the **Snapshot file** field, and the exact time at which to take the snapshot during the next run in the **Time** field.

Click the **OK** button to exit the *Project Settings* dialog and save changes. Run the project to completion.

**NOTE:** Make sure that the run duration is greater than the snapshot time, or no snapshot file will be created!

## Starting from a Snapshot

Open the *Project Settings* dialog window as described in Editing Project Settings above. In the **Runtime** section of the dialog, choose **From snapshot file** in the **Startup method** drop list. Enter a name for the snapshot file in the **Input File** field or use the **Browse...** button to select the file.



Click the **OK** button to exit the dialog and save changes. Run the project—it should start initialized from the snapshot time. See Initialization and Initial Conditions in the EMTDC section for more details on snapshot files.

## Saving Output to File

Open the *Project Settings* dialog window as described in Editing Project Settings. In the **Runtime** section of the dialog, choose **Yes** from the **Save channels to disk?** drop list. Enter the name for the output file in the **Output file** field.



Click the **OK** button to exit the *Project Settings* dialog and save changes. Run the case to create the output file (see Running a Simulation above). See EMTDC Output Files later in this chapter for more details on output file format.

## Cleaning the Temporary Directory

### Clean All Temporary Directories

This folder is used to house all temporary files generated and/or needed by PSCAD to run a simulation (i.e. compilation, make, data, etc. files). It is sometimes necessary to clear the contents of this folder to ensure that you are starting from a clean build. See PSCAD Temporary Directories in Chapter 4 of this manual for more details.

To delete all temporary project files, do one of the following:

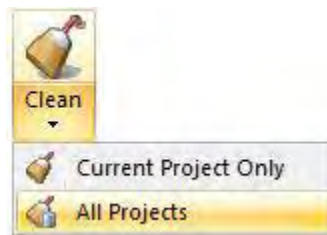
- Select the **Clean** button in the **Home** tab of the ribbon control bar:



- Right-click on the project title in the workspace window, and select **Clean Temporary Directory** from the pop-up menu.

### Clean All Temporary Directories

It is also possible to clean all temporary folders associated with the projects in the current workspace. To do so, select the **Clean** button in the **Home** tab of the ribbon control bar, and then select **All Projects**.

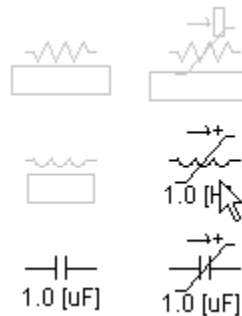


## Components and Modules

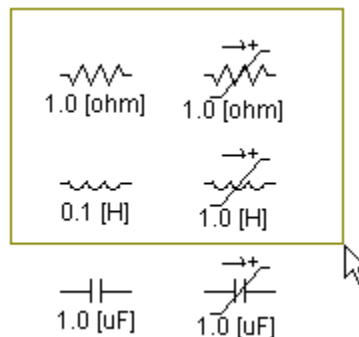
# Selecting Objects

Select an individual object with a left-click on the object graphic. Select a group of objects by one of the following methods:

- Press and hold the **Ctrl** key and then separately select (**left-click**) all objects to be grouped into one selection. Note that you may also use box select in combination with separate select, provided that the **Ctrl** key remains depressed.



- **Press and hold** the left mouse button and then **drag** the mouse pointer so that a box outline appears (box select). Encompass all desired objects to be selected and then **release** the left mouse button.



To un-group individual objects, hold the **Ctrl** key and then select (**left-click**) all objects to be un-grouped. To un-group all objects, left-click on the canvas background.

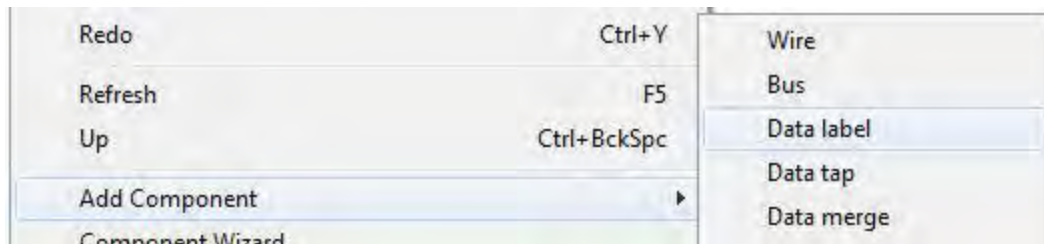
## Adding Components to a Project

### Adding Multiple Instances of a Component

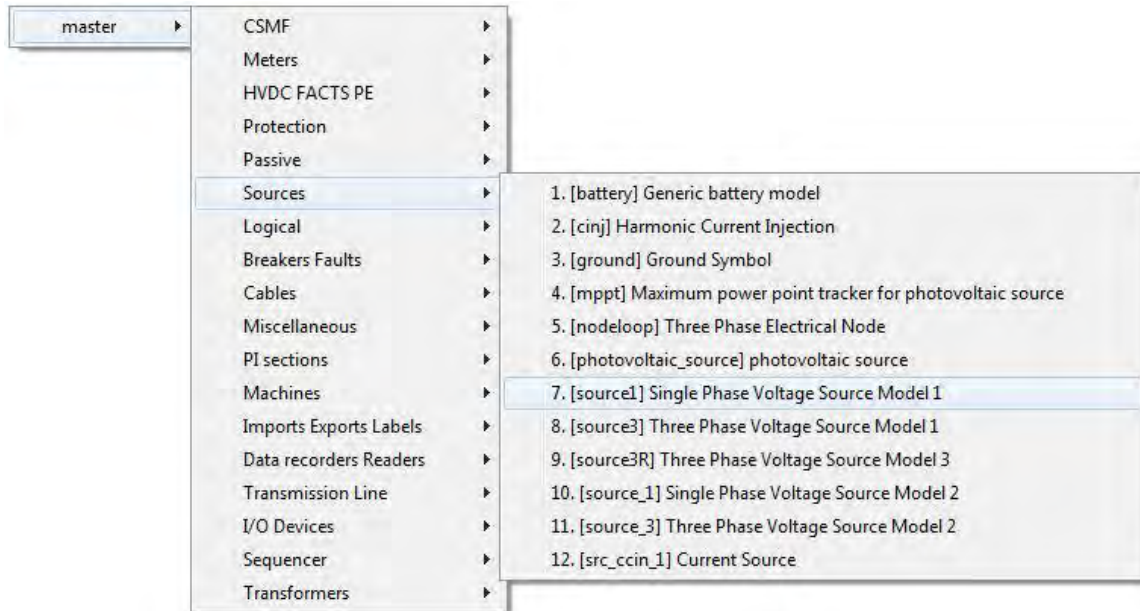
There are a variety of ways to insert components onto the Schematic canvas. Before proceeding, ensure that you are viewing the desired page in the Schematic window.

- **Manual Copy/Paste:** Open the master library and navigate to the area containing the desired component. Right-click on the component and select **Copy**, or select the component and press **Ctrl + c**. Open the project page where you wish to add the component, right-click over a blank area and select **Paste** (or press **Ctrl + v**).
- **Right-Click Menu:** Right-click over a blank area of the page and select **Add Component**. A sub-menu will appear containing the most commonly used components from the master library. Select a component and it will be automatically added.

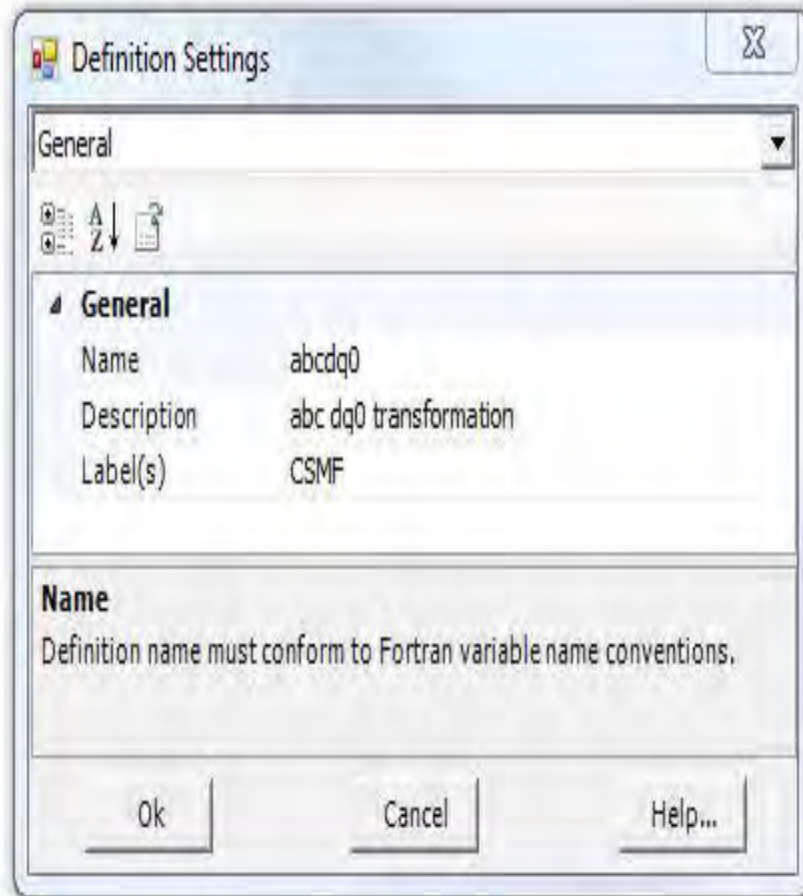




- **Library Pop-Up Menu:** Press **Ctrl + right mouse button** over a blank area of the page to invoke the library pop-up menu system. Select a component and it will be automatically added.



Library pop-up menu systems will include all libraries currently loaded in the workspace. The organization of definitions in this menu is accomplished by using the *Label* definition property. For example, the *abcdq0* component in the master library will appear under the *CSMF* sub-menu above. The definition settings are set as follows:



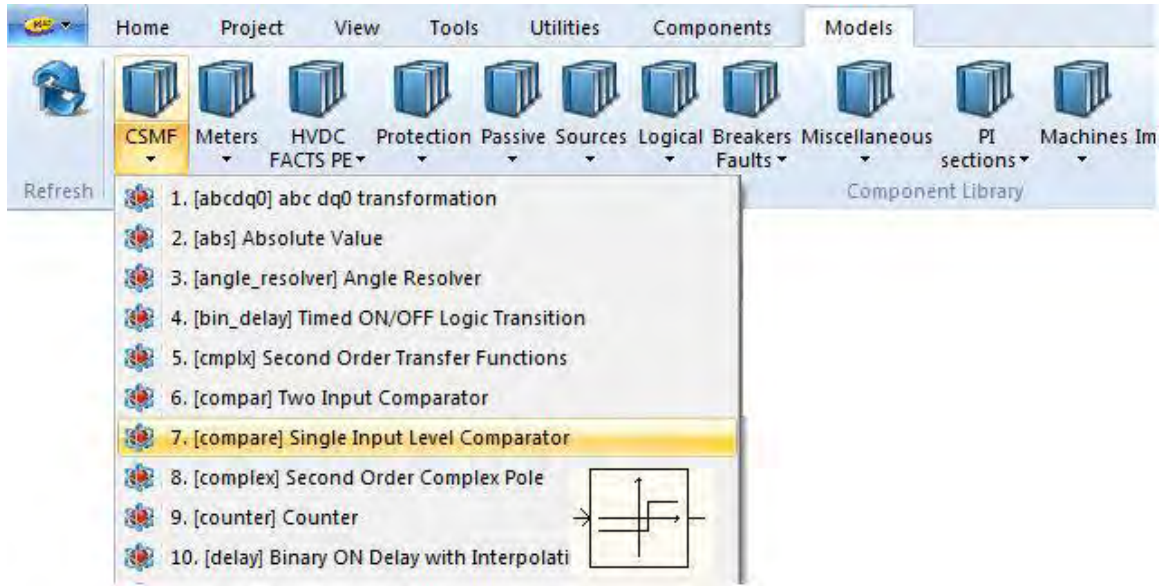
See Editing Definition Settings later in the chapter for more details.

- **Components Tab in the Ribbon Control Bar:** **Left-click** on any of the buttons and then move the mouse pointer over the Schematic canvas. You should see the object attached to the pointer. Continue to move the object to where it is to be placed and then **left-click** again.



Components Tab in the Ribbon Control Bar

- **Models Tab in the Ribbon Control Bar:** **Left-click** on the down arrow associated with any of the categories and then move the mouse pointer down the resulting list. Select the desired component — you should see the object attached to the pointer. Continue to move the object to where it is to be placed on the canvas and then **left-click** again.



Models Tab in the Ribbon Control Bar

## Adding Multiple Instances of a Component

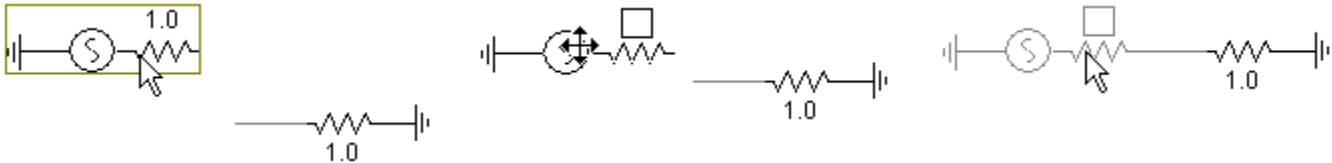
Multiple instances of the same component can be added when using the Components Tab in the ribbon control bar:

1. **Hold down the Ctrl key and left-click** on the desired component from the *Components* tab.
2. **Continue holding the Ctrl key** and move the mouse pointer over the Schematic canvas. **Left-click** again to paste the first instance of the component. Move the mouse pointer to a new position and left-click again to paste a second instance of the component.

You may continue this process and add as many instances as desired, provided the **Ctrl** key remains depressed. To escape from this function, simply release the **Ctrl** key at any time.

## Moving or Dragging an Object

To move an object, place the mouse pointer over the object icon. Press and hold the left mouse button. Now drag the mouse to move the component. When you move a component, it will always snap to the nearest drawing grid, even if the grid dots are not visible.



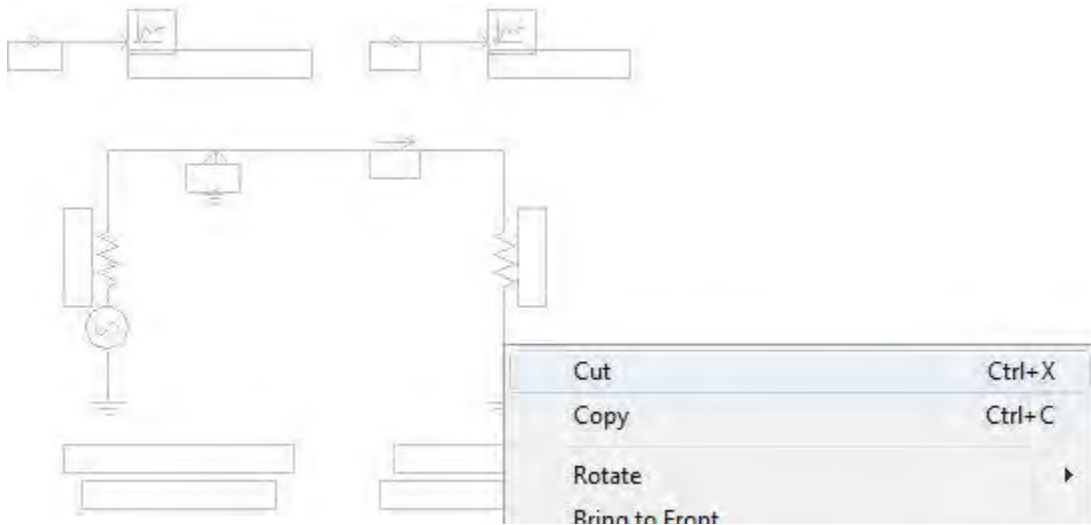
Click the left mouse button and hold

Drag the object

Release the left mouse button

## Cut/Copy/Paste

Objects can be cut, copied and pasted as many times as desired within the Schematic canvas: Right-click on the object and select either **Cut** or **Copy** (or select the object and press **Ctrl + x** or **Ctrl + c** respectively). Multiple objects can be cut, copied and/or pasted simultaneously.

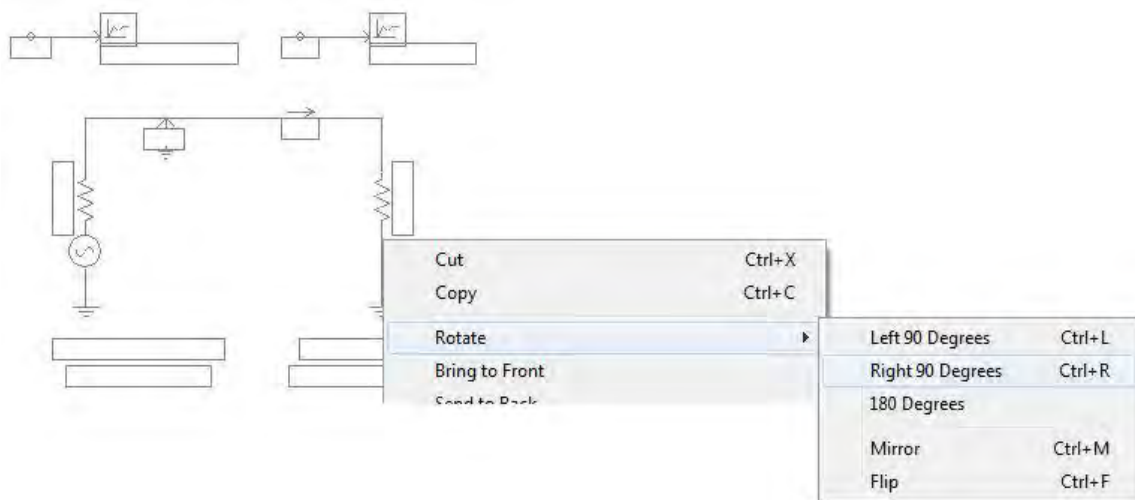


Once cut or copied, you can paste the object by right-clicking over a blank area of the Schematic canvas and selecting **Paste** (or press **Ctrl + v**).

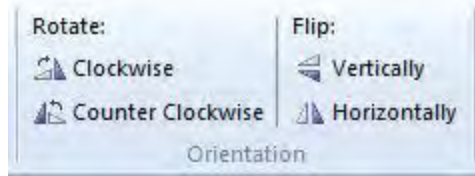
## Rotate/Mirror/Flip

Once added, single or multiple objects can be rotated, flipped or mirrored:

- **Hotkeys:** Select the object and press the **r**, **f** or **m** to rotate, flip or mirror respectively.
- **Right-Click Menu:** Right-click over the object and select **Rotate** or **Flip**. Select the desired option from the resulting sub-menu.



- **Components Bar:** Select the object or group of objects and then press one of the four **Orientation** buttons in the Components ribbon tab.



## Deleting Objects

Select the object (or objects) and press the **Delete** key.

## Undo and Redo

To undo or redo any object manipulations, such as moves, cuts, pastes, deletions, etc., select the **Undo** or **Redo** button in the Home ribbon tab, or press **Ctrl + z** or **Ctrl + y** respectively. The undo and redo features will store most manipulations and changes, however some limitations may apply.

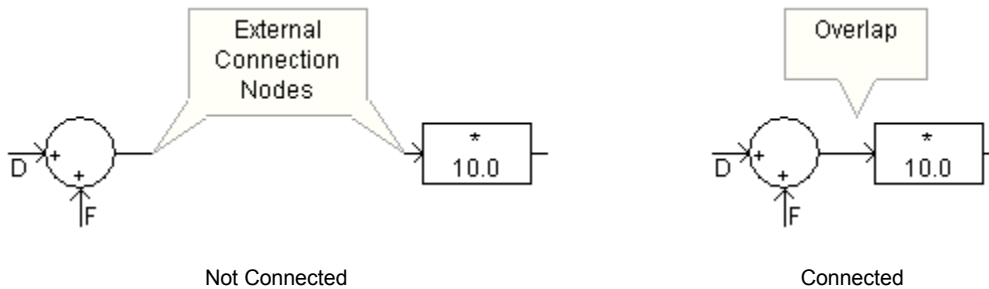
If this feature does not appear to be working, check the Projects section of the *Application Options* dialog to see if *Change tracking* is set to *Use Undo/Redo Stack*.

## Connecting Components Together

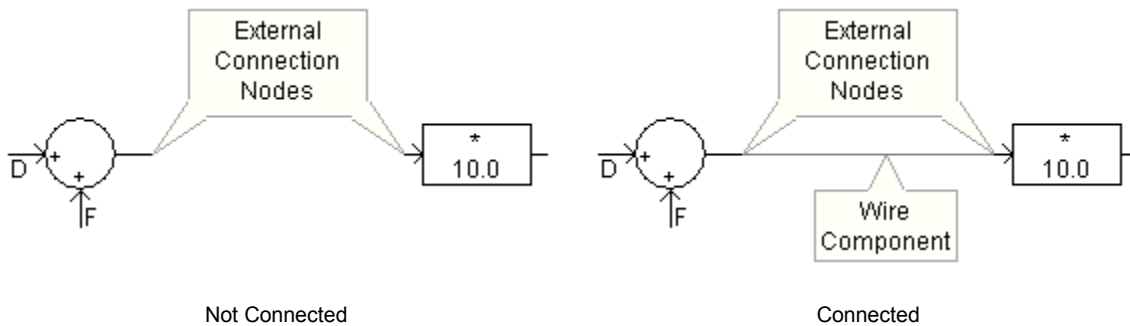
### Invalid Component Connections

A connection between components can be made in one of the following ways:

- Overlapping one component port connection over top another.

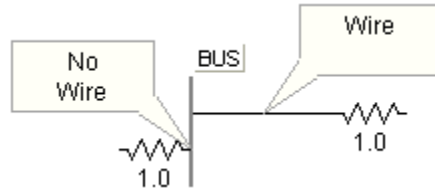


- The endpoint of a Wire component can be used to make contact with port connections, or the endpoint another Wire:



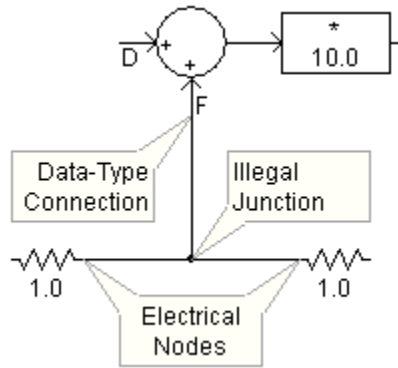
**NOTE:** See Valid Connections in the *Wire* help page for more on connecting objects.

- Any point along a Bus component is considered a valid connection point, to which Wires and component port connections may be connected.



## Invalid Component Connections

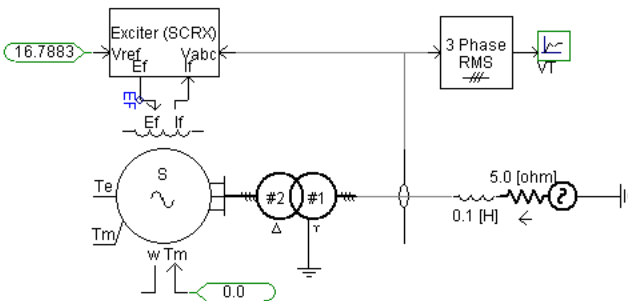
It is important to note the difference between an electrical and a data signal when connecting components together. Wires may be used interchangeably as data signal paths, or for connecting electrical nodes. However, it is illegal to connect data signals to/from electrical nodes. For example, the following connection is invalid:



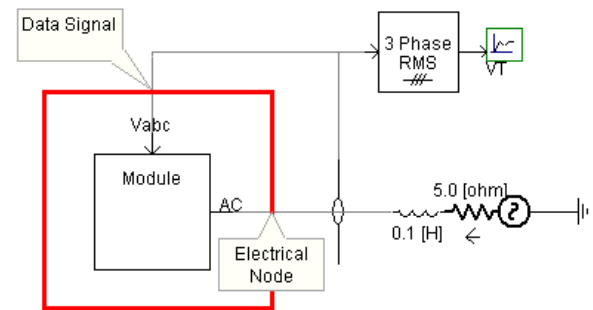
Wires are universal in that they assume the signal type of the nodes they are connected to. If a wire is connected to an electrical node, it will become an electrical wire, for example.

## Porting Signals and Nodes between Modules

At some point during project design, you may want to identify a common set of components, and consolidate them into their own module component. This procedure can be useful in simplifying and organizing the appearance of the overall project. It also helps to reduce duplication if the module can be multiple instanced. In doing this however, data signals and electrical nodes that once connected the module circuit to the greater circuit will sever, and these must be reconnected.



Original Circuit



Sub-Circuit Consolidated into a Module and Connected to Greater Circuit

The process of transporting data and connecting nodes between a module circuit and the greater circuit (i.e. its parent module), is referred to as *porting*. Porting signals and nodes between modules can be accomplished in a few different ways:

- Port Connections
- Component Parameters
- Radio Link Components

## Data Signal Analogies to Fortran Code

When a project is compiled, the application creates a variety of files describing the project, so that an executable file can be generated. Some of these are Fortran files (\*.f), where one Fortran file is created for each unique module in the project (including the main page). Each file is a unique set of subroutines describing the content of the corresponding module.

When a data signal is passed into or out of a module, it will either appear as a subroutine argument (if it is ported as a parameter or port connection), or be transferred via storage array, inline within the subroutine code body, if using Radio Link components.

```
!
SUBROUTINE DSDyn (X)
! . . .
```

Data Signal X as an Argument

```
!
SUBROUTINE DSDyn ()
! . . .
X = STOF(ISTOF + 1)
```

Data Signal X Extracted from Storage Inline

## Porting Data Signals by Port Connection

If you create a new module component by using the *Component Wizard*, the process of adding port connections is fully automated. See *Creating a New Component or Module* later in this section for more details.

**NOTE:** Ported signal names are case sensitive. The port signal name must match its corresponding import/export tag exactly.

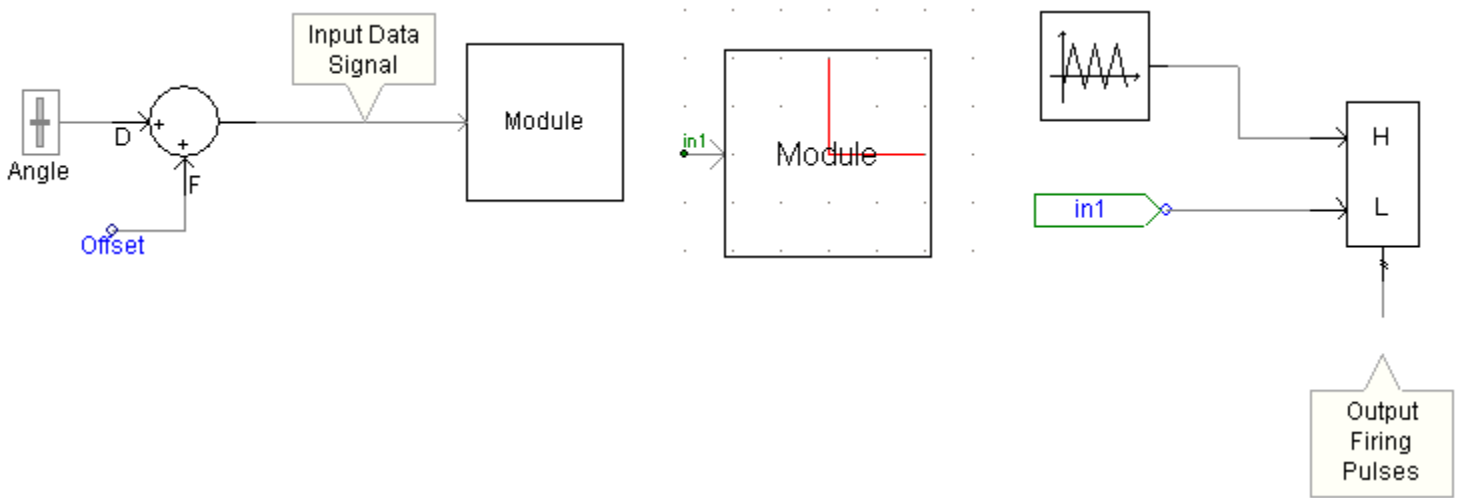
If you are editing an existing module, follow these general steps:

1. Edit the definition of the module. See *Editing a Component or Module Definition* in this section for more details.

2. Define an input or output port connection in the *Graphic* section (Graphic tab) of the module. See The Graphics Section for more details.
3. Add a corresponding Import or Export component on the module Schematic canvas. Each *Import* or *Export* must be named the same as its corresponding port connection.

EXAMPLE 5-1:

Consider a project that consists of a simple phase angle offset control being input into a module. The module Graphic section consists of a single port connection named *in1*, which is to be connected to the incoming data signal from its parent module. The input signal *in1* is used as a reference angle to a firing pulse generator inside the module definition Schematic canvas.



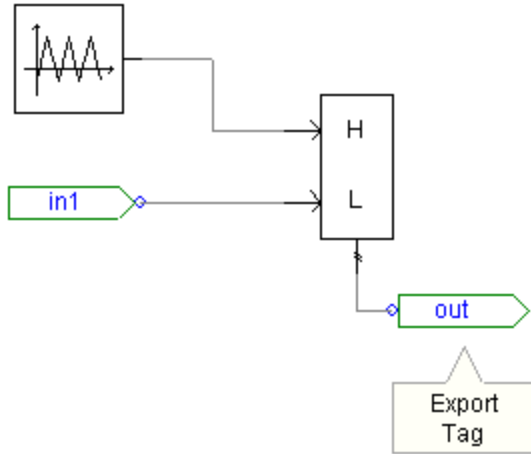
Parent Canvas Containing Module Component and Greater Circuit

Graphic View of Module Definition

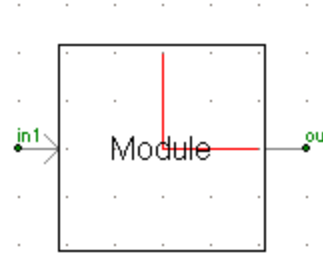
Schematic Canvas of Module

A user wants to bring the firing pulse output signal back out to the main page using an output port connection. The first step is to define an *Export* component and connect it to the firing pulse output signal. The user names the export signal *out*. The final step is to define an output port connection in the module definition *Graphic* section.



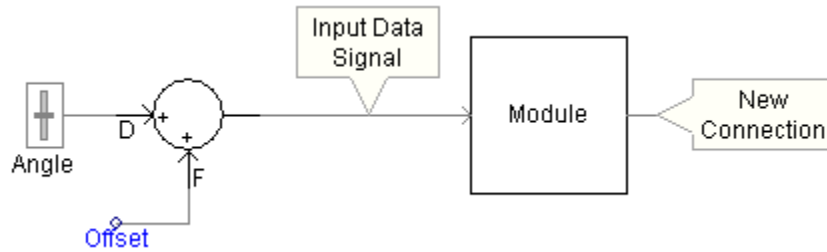


Added Export tag in Canvas of Module



New Output Port Connection in Graphic of Module

The user may now input this new output signal to any other component on the main page.



Parent Canvas with Module and Controller with New Connection

Note that only one instance of a uniquely named *Import* or *Export* component may exist within a module, just as only one uniquely named port connection may exist. If the same ported signal is required at more than one location within the page, you can utilize the Data Label component.

## Porting Data Signals by Component Parameter

Unlike with *port connections*, the *Component Wizard* does not automatically create parameters for you when creating new components. You must therefore create a new module first (if necessary), before editing the definition and defining parameters.

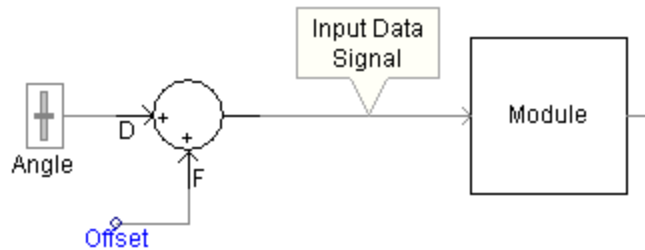
**NOTE:** Ported signal names are case sensitive. The parameter signal name must match its corresponding import/export tag exactly.

Follow these general steps:

1. Edit the definition of the module. See *Editing a Component or Module Definition* in this section for more details.
2. Define an input or an output parameter in the *Parameters* section (Parameter tab) of the module. See *The Parameters Section* for more details.
3. Add a corresponding *Import* or *Export* component on the module Schematic canvas. Each *Import* or *Export* must be named the same as its corresponding input or output parameter.

EXAMPLE 5-2:

Consider the project outlined in *Example 5-1*.



This time, instead of using a *port connection*, the user wants to port the *Input Data Signal* through a parameter on the module. The data wire leading into the module needs to be named –this is accomplished by using a Data Label. Then define a parameter in the module definition *Parameters* section.

The first screenshot shows the module with the 'Angle' signal and 'Offset' parameter connected to the summing junction. The output signal is now named 'in1'. The 'Module' box is shown to the right.

The second screenshot shows the 'CategoryProperties' dialog for the module. A new input parameter 'Input Firing Angle' has been added to the list. The 'Data type' is set to 'Variable' and the 'Intent' is 'Input'. The 'Prompt text' is 'This is the input signal.' and the 'Help mode' is 'Override'. The 'Default value' is '30.0'.

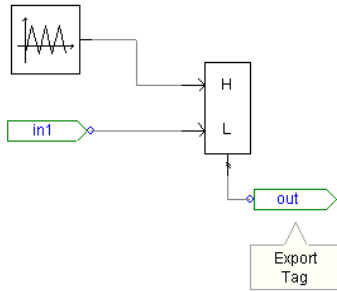
The third screenshot shows the 'Module Component Parameter Dialog' for the module. The 'Name' is 'Input Firing Angle' and the 'parameter value' is 'in1'. A callout box points to the 'in1' value with the text 'Signal name entered as parameter value.'

Data Signal Named and Port Connection Removed from Module

New Input Parameter in Parameters Section of Module Definition

Module Component Parameter Dialog

Say also that the user wants to bring the firing pulse output signal back out to the main page using a parameter. The first step (as before) is to define an *Export* component and connect it to the firing pulse output signal. The user names the export signal *out*. The final step is to define an output parameter in the module definition *Parameters* section.



New Output Parameter

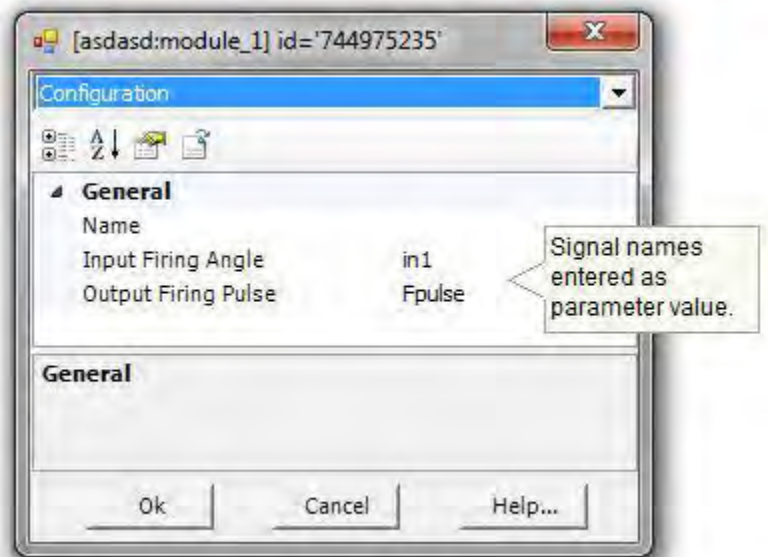
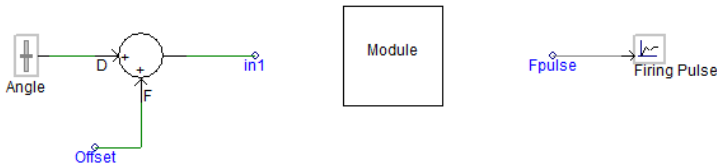
Category Properties	
Name	Configuration
Conditional expression	
<div style="display: flex; justify-content: space-between;"> <span> </span> </div>	
▶ Name	Text
▶ Input Firing Angle	Real
▶ <b>Output Firing Pulse</b>	Real
Description	<b>Output Firing Pulse</b>
Symbol	<b>out</b>
Group label	
Default units	
Minimum value	-1e+308
Maximum value	+1e+308
Data type	<b>Variable</b>
Intent	<b>Output</b>
Prompt text	<b>Output firing pulse.</b>
Help mode	<b>Overwrite</b>
Conditional expression	
Default value	<b>0.0</b>

Specify intent

Added Export tag in Canvas of Module

New Output Parameter in Parameters Section of Module Definition

The user may now input this new output signal to any other component on the main page.

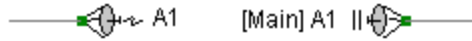


Parent Canvas with Module and Controller with New Connection

Module Component Parameter Dialog

## Porting Data Signals by Radio Links

The aforementioned data signal porting can also be accomplished by means of Radio Link components.

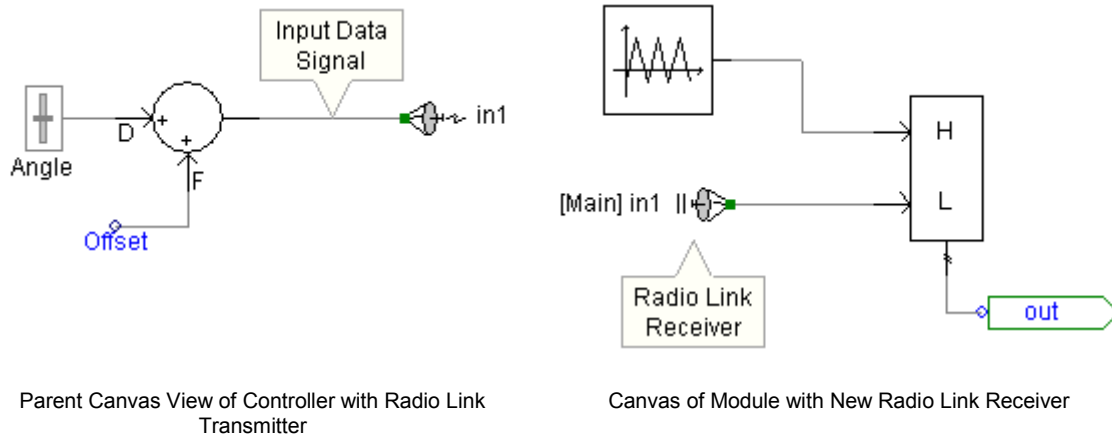


Radio Links offer a 'wireless' method of porting the signals; that is there is no need for 'hard-wired' port connections, input parameters and Import/Export components. In fact, *Radio Links* make it possible to port signals simultaneously to multiple receiving points throughout the project from a single transmit point. This is analogous to a wireless broadcast, where multiple receivers can accept data from a single transmitter. See the Radio Links component help for more details.

#### EXAMPLE 5-3:

Consider again the project in *Example 5-1*. A user wants to modify the system so that the input signal *in1* is transferred into the module by means of *Radio Link* components. The input data signal is first disconnected from the module and then reconnected to a Radio Link transmitter named *in1*.

Secondly, the input port connection is removed from the module definition Graphic section, as it is no longer required. Lastly, the Import component is removed and replaced with a Radio Link receiver. See Radio Links for more details on setting up the component parameters.



Parent Canvas View of Controller with Radio Link Transmitter

Canvas of Module with New Radio Link Receiver

**NOTE:** Radio Link components are not fully supported when using Multiple Instance Modules. Please contact the PSCAD Support Desk for more information.

## Porting Electrical Nodes

If an electrical node needs to be represented within a module, then PSCAD must be informed of this requirement so that the network of electric nodes can be properly mapped. This is accomplished by using port connections and a special electrical node component called an *XNode*.

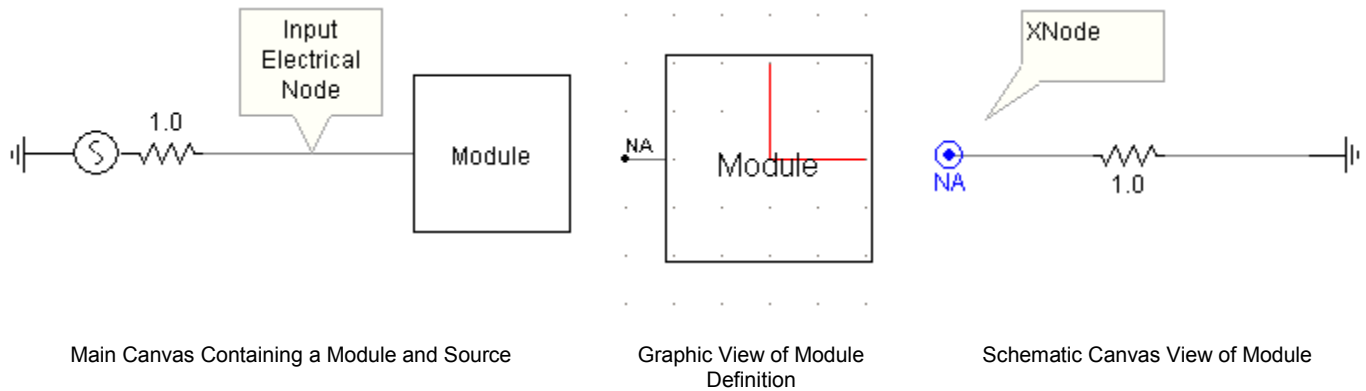
If you create a new module component by using the *Component Wizard*, the process of adding port connections and *XNodes* is fully automated. See *Creating a New Component or Module* later in this section for more details.

If you are editing an existing module, follow these general steps:

1. Edit the definition of the module. See *Editing a Component or Module Definition* in this section for more details.
2. Define an electrical port connection in the *Graphic* section (Graphic tab) of the module. See *The Graphics Section* for more details.
3. Add a corresponding *XNode* component on the module Schematic canvas. Each *XNode* must be named the same as its corresponding port connection.

## EXAMPLE 5-4:

Consider a simple project that consists of a single-phase source bus connected to a module. The module *Graphic* section includes a single port connection named *NA*, which is connected to the source bus on the main Schematic canvas. The module canvas (i.e. Schematic view) consists simply of a Resistor component connected directly to a Ground component.



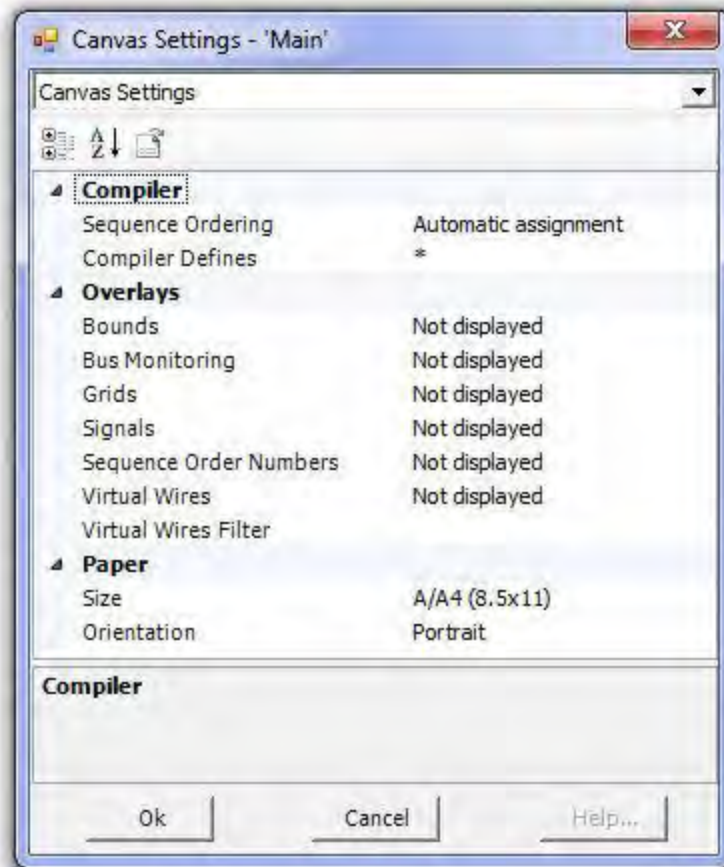
Here are a few important factors to remember when porting electrical signals into and out of modules:

- *XNodes* cannot be placed on the project main page.
- Only one instance of an *XNode* may exist within the module for a single port connection.
- An *XNode* cannot be directly grounded. That is, do not connect a Ground component directly to an *XNode*. A ground may be connected directly to the corresponding electrical port connection however.

## Editing Module Canvas Settings

Right-click on a blank part of the Schematic canvas and select **Canvas Settings...** These settings are specific to a single module definition, and will affect all instances of that module.

A dialog window entitled *Canvas Settings* will open, as shown below:



The *Canvas Settings* dialog inputs are described in the following sections.

## Compiler

These options are related to the way in which system dynamics code is ordered. For more on this option, see Component Ordering in chapter 11 of this manual.

- **Sequence Ordering:** Enable this option to ensure that a PSCAD smart algorithm will automatically order your control components. This algorithm systematically scans all control systems and sub-modules within the module and determines the sequence in which each component should appear in the in EMTDC system dynamics. This option is by default enabled, and should remain that way unless the user wants to reorder these components manually.
- **Compiler Defines:**

## Overlays

These options are related to the page display.

**Bounds:** Set this option to view borders of one paper size smaller than the current size. This is especially convenient when attempting to reduce the page size of a canvas containing components and other objects.

- **Bus Monitoring:** Set this option to view voltages on all buses.
- **Grids:** Set this option to view the major grid points on the module page.

- **Signals:** When this option set, PSCAD will use icons placed on data signal wires and connections, so as to allow for easy, graphical differentiation between feed-forward and feedback signals.
- **Sequence Order Numbers:** When this option is set, PSCAD will label each component/module instance within this specific module with a sequence number. This sequence number represents the sequential placement of the component/module code in the EMTDC system dynamics.
- **Virtual Wires:** Use this option to virtually connect signals on a canvas that are connected by Data Labels, or sourced internally from components. For more on this, see Virtual Control Wires in chapter 11 of this manual.
- **Virtual Wires Filter:** Enter one or more specific signal names (comma separated) into this field in order to limit which virtual wires are drawn. An empty field will assume default behaviour and include all wires; or an asterisk (\*) can be inserted in the list to include all, without the need to delete the signal list. This filter can help immensely in tracking down paths for specific signals.

## Paper

These options are related to the page settings (size and layout).

- **Size:** Select a standard paper size for this module canvas.
- **Orientation:** Select the module canvas orientation.

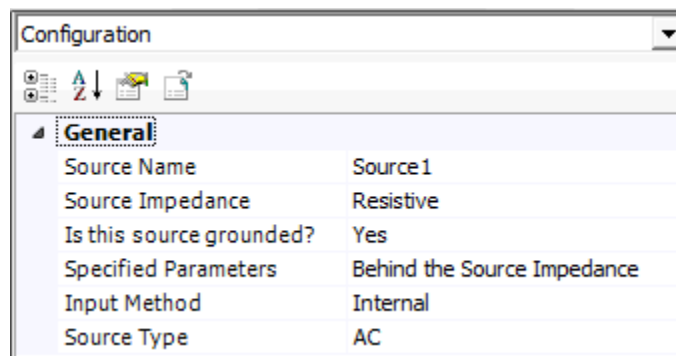
**NOTE:** The *Oversize (34x44)* paper is included for compatibility with the PSCAD V2 large canvas size. However, it can still be used as a valid paper size.

## Editing Component or Module Parameters

Either left double-click the component, or right-click on the component and select **Edit Parameters...** from the pop-up menu, to edit the parameters of a component or module.

**NOTE:** The left double-click operation is affected by the *Drill Down* application option. If this setting is set to *Double Click*, a left double-click on a module component will perform an Edit Definition operation (i.e. will open the module canvas in Schematic view).

The *Edit Parameters* operation will bring up the component instance parameters dialog. The figure below shows the first page of the dialog for one of the single-phase source models in the master library (called *source\_1*).



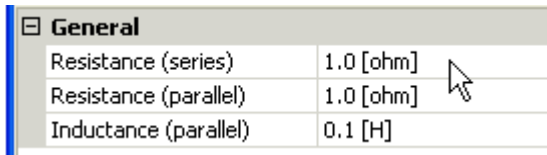
Dialogs for other components will differ, but all include the same basic features and most contain multiple pages: At the top of the dialog is a drop list, which contains a list all of the dialog pages (called *categories*). In this example, the first page is entitled *Configuration*. To move through or view any other pages, left-click the down arrow on the field as shown below:



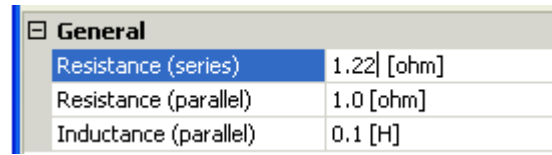
If there are too many items in the list, you will need to scroll to see all the category names. To do this, browse through the list using the scroll bar along the right side of the list. Or, press the up or down arrow on your keyboard while the list is expanded. When the required choice is highlighted, press the **Enter** key to select it.

## Changing Parameter Values

Each category page in the component parameter dialog will usually contain an assortment of input fields. Left-click the field value box and type a new value to modify the parameter value.



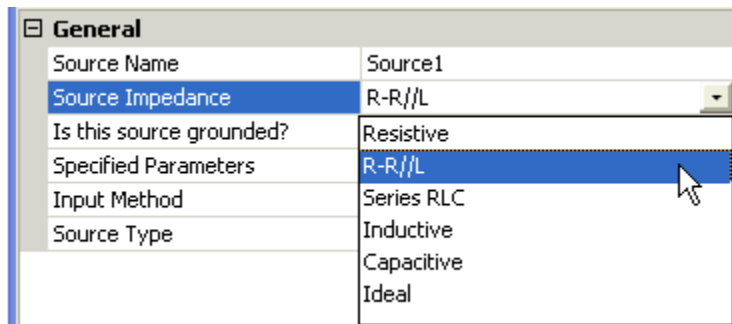
Select the Parameter



Modify the Value Box

Some parameter fields will contain a unit (i.e. [MVA], [sec], [m/sec], etc.), displayed next to the parameter value. See Unit System in this chapter for more details.

Choice list or Boolean type parameters will have a downward pointing arrow on the right hand side. Click on that arrow to see the list of choices and then click again on the required item. The choice list could have more items than visible in the list. To scroll through the list, simply use the up or down arrows on your keyboard while the list is expanded. When the required choice is highlighted, press the **Enter** key to select it



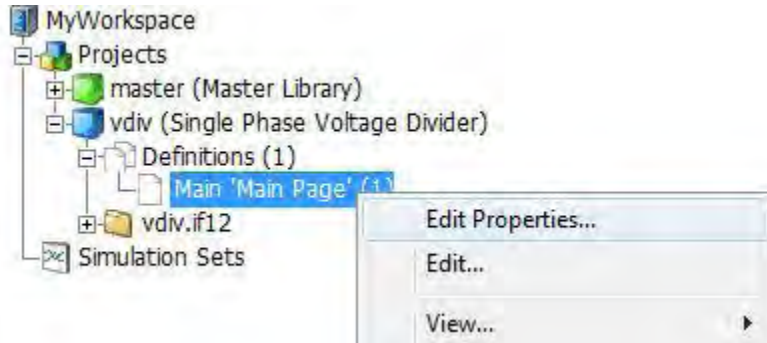
Once you are finished editing, click on the **OK** button to accept the edited values and to exit the dialog (all edited parameter fields will appear in bold). Clicking on the **Cancel** button will exit the form and ignore all changes made.



## Editing Definition Settings

Definition settings are a few special properties for all definition types that may only be accessed from the definition list in the workspace window: These are the definition *Name*, *Description* and *Label*.

In the workspace window, expand the *definitions* branch and right-click on a component definition listed therein. Select **Properties...**:



This will invoke the *Definition Settings* dialog window:



The following list describes the functions of this dialog:

- **Name:** Enter a name for the definition. This name must conform to FORTRAN standards (i.e. it cannot begin with a number or contain any spaces or other illegal characters).
- **Description:** Enter a description of the definition.
- **Label(s):** This property is used to help organize component definitions existing in user defined library projects, for the purpose of display in the Library Pop-Up menu system. See Adding Components to a Project for more details.

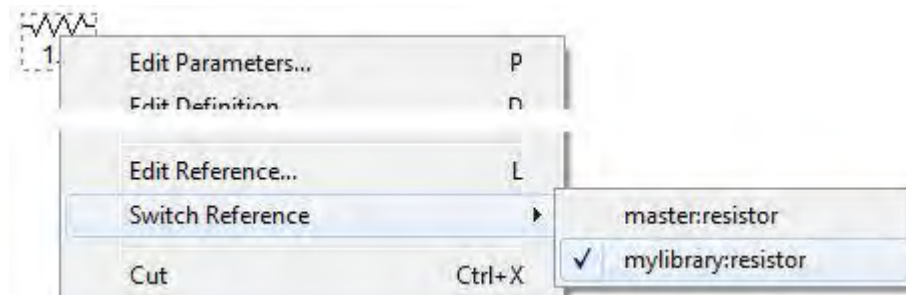
## Definition Referencing

In PSCAD versions X4 or greater, users may reference any definition from a any component instance. Obviously, the definition must be compatible with the instance for this to work, but this feature can be convenient when working with several versions of the same definition. Referencing definitions is also important when copying and pasting module components or transmission segments between projects.

### Reference List

All definitions that share the same name in any loaded project in the workspace will appear in the reference list. For example, a user has loaded the master library and a custom library in the workspace, where a custom component definition called *resistor* is stored. Of course, a definition called *resistor* also exists in the master library. The user wants to switch freely between these two definitions while working in a case project.

Right-click on the component instance in question and select **Switch Reference...**



As can be seen above, both resistor definitions have been detected. From here you may freely switch between definitions.

### Edit Reference

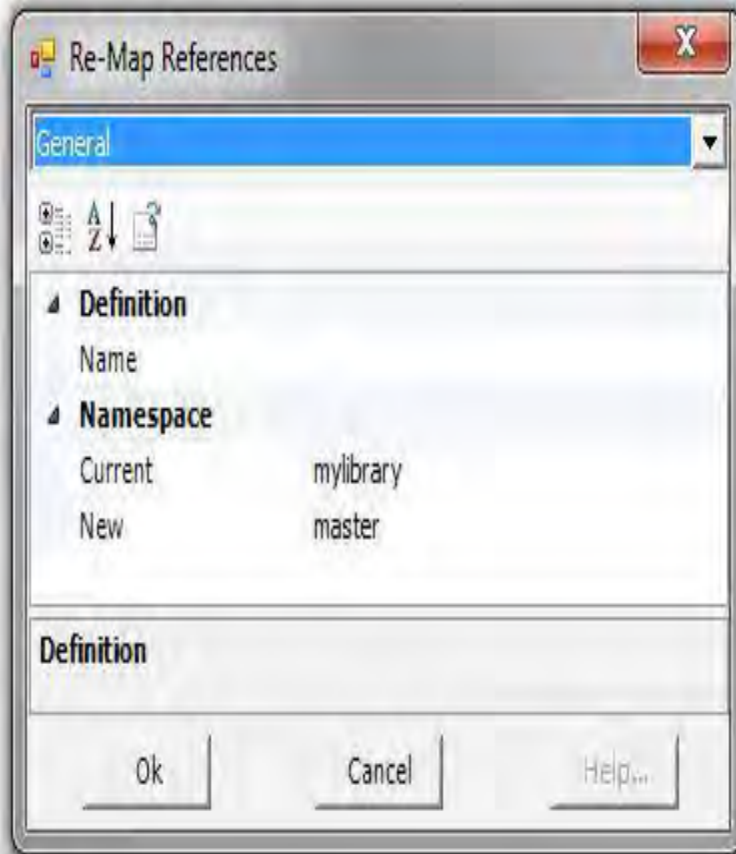
If a component instance is not currently referenced to a definition, or if you want to point to a specific definition of a different name, then use **Edit Reference...** This will invoke the *Edit Reference* dialog in which you can adjust both the namespace and definition name.

### Re-Map References

Sometimes you may need to change the namespace name of a library project. If there are any component instances with definitions referencing this namespace, they will become de-referenced and appear as place cards.



If this happens, you can re-map all component instances in a project simultaneously. Right-click on the project in the workspace window and select **Re-Map References** to bring up the associated dialog:

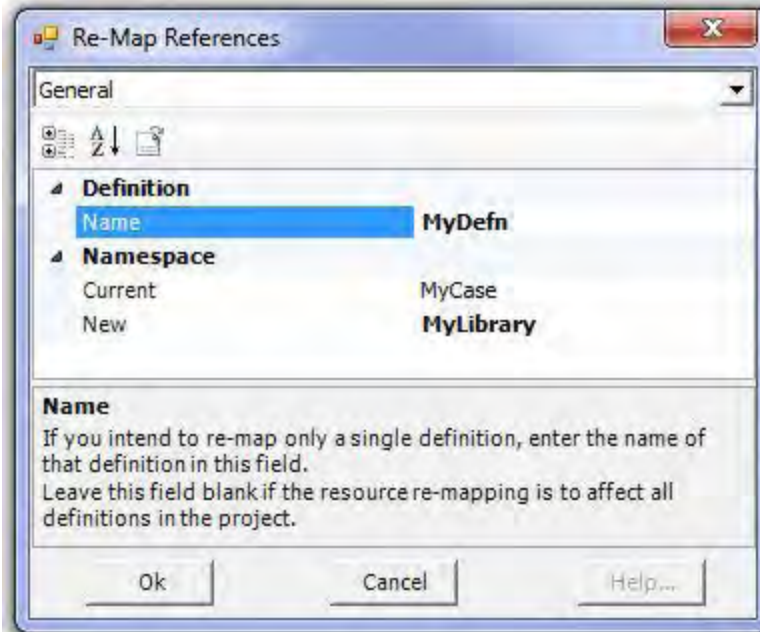


The following list describes the functions of this dialog:

- **Name:** Enter the definition name. Adding a name to this field will ensure that reference re-mapping will be specific to only that definition and no other.
- **Current:** Enter the current namespace, from which definitions are to be re-mapped.
- **New:** Enter the new namespace, to which definitions are to be re-mapped.

EXAMPLE:

A project contains 100 instances of a component, whose definition name is *MyDefn*, residing in a case project called *MyCase*. Another definition, again called *MyDefn* has been copied to a library project with namespace *MyLibrary*. It is desired to re-reference all 100 instances of the component to the *MyDefn* definition in the library project. The Re-Map References should then look like this:



## Viewing Component Properties

The complexity of a component may range from simple to elaborate. Some components may possess properties such as parameters, port connections, etc., numbered in the hundreds. When designing and debugging a new component, or verifying the settings of an existing component, it becomes important to be able to view this property data in a convenient format. The *Component Properties Viewer* provides an intuitive and convenient means of categorizing and displaying component attributes so that they may be examined efficiently.

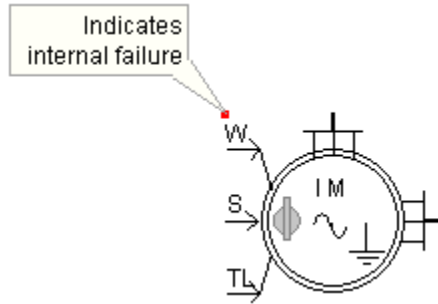
Name	Caption	Type	Unit	Minimum	Maximum	Data	Value	F
Name	Machine name	String				Sync1	Sync1	*
Nqaxw	No. of Q-axis Damper Win...	Choice				1	1	*
Cnfg	Data Entry Format:	Choice				0	0	*
MM	Multimass interface:[Enabl...	Choice				0	0	*
CfR	Armature Resistance ar...	Choice				0	0	*

Component Properties Viewer

The component properties are categorized into tabbed categories, where each section lists the individual attributes in a list format. The categories are described as follows (see Chapter 9 in this manual for details on these topics):

- **Layers:** Provides information on graphical layers and related conditional statements set in the component Graphics section.

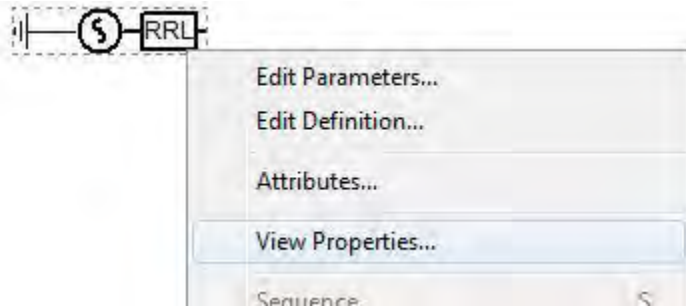
- **Ports:** Provides information on external connections (nodes) and related conditional statements set in the component *Graphics* section.
- **Parameters:** Provides information on parameter variables, such as *Symbol* name, type, unit, etc., set in the *Parameters* section.
- **Computations:** Provides information on defined computed variables, such as type and computed value set in the *Computations* segment of the *Script* section
- **Errors:** Provides a list of pre-compilation errors, for example, *Computations* segment errors.



**NOTE:** All values displayed in this viewer are shown in the same precision as will be used in a substitution. For EMTDC, this is 12 significant figures. All evaluation errors are indicated with a **#NaN** in the **Value** column.

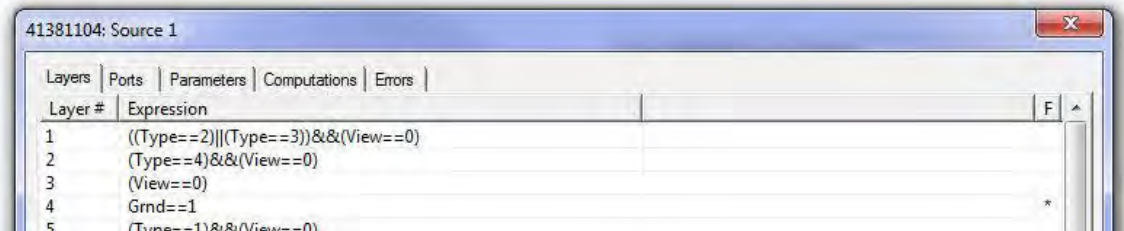
## Invoking the Component Properties Viewer

Right-click on any component and select **View Properties...** from the pop-up menu to invoke the *Component Properties Viewer*.



## Category Descriptions

The properties listed in each tabbed section are organized into columns.



## Layers Tab

Layers:

- **Layer #:** Simply a numbered list.
- **Expression:** The conditional statement defining the graphical layer.
- **F:** An asterisk "\*" in this column indicates that the *Expression* conditional statement is currently true (otherwise false).

Name	Expression	M	Type	Dim	F
VIn	(VCtrl)&&(View==0)&&(Spec==0)	<	Real		
FIn	(FCtrl)&&(View==0)&&(Spec==0)	<	Real		
Fin	(FCtrl)&&(View==1)&&(Spec==0)	<	Real		
NA	(View==0)	o			
NB	(View==0)	o			
NC	(View==0)	o			
N	(View==1)	+		2	*

## Ports Tab

Ports:

- **Name:** The *Symbol* name of the corresponding port connection.
- **Expression:** This is the conditional statement defining whether or not this port connection is enabled.
- **M:** Mode. The symbols displayed here indicate the port connection type.
  - o: Electrical node
  - <: Input data node
  - >: Output data node
  - +: Shorted node
- **Type:** The declared type (i.e. REAL, INTEGER, LOGICAL) if the port connection is non-electrical.
- **Dim:** The signal dimension if a vector. This column is left blank if the signal is a scalar (i.e. Dim = 1).
- **F:** An asterisk "\*" in this column indicates that the *Expression* conditional statement is currently true (i.e. enabled).

Name	Caption	Type	Unit	Minimum	Maximum	Data	Value	F
Name	Source Name	String				Source 1	Source 1	*
Type	Source Impedance Type	Choice				2	2	*
Grnd	Is the star point grounded?	Choice				1	1	*
View	Graphics Display	Choice				1	1	*
Spec	Specified Parameters	Choice				0	0	*
VCtrl	External Control of Voltage?	Choice				0	0	*
FCtrl	External Control of Freque	Choice				0	0	*

Parameters Tab

Parameters:

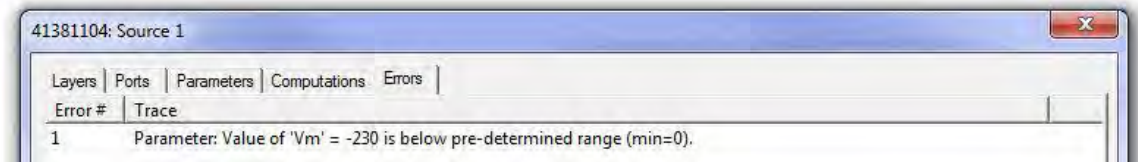
- **Name:** The *Symbol* name of the parameter.
- **Caption:** The *Description* of the parameter.
- **Type:** The type of parameter (i.e. input, text, choice, etc.).
- **Unit:** The *Default Unit* for the parameter (or Target Unit).
- **Minimum:** The *Minimum Value* of the corresponding parameter.
- **Maximum:** The *Maximum Value* of the corresponding parameter.
- **Data:** The actual value as it is entered in the parameters dialog.
- **Value:** The actual value of the parameter following evaluation by PSCAD (considering both entered units and the *Target Unit*).
- **F:** An asterisk \*\* in this column indicates that the parameter is enabled (otherwise disabled).

Name	Expression	Type	Value
Mode	Ctrl==3 ? (PCtrl ? (VCtrl ? 1 : 3) : (VCtrl ? 2 : 0) ) : 0	Integer	0

Computations Tab

Computations:

- **Name:** The name of the declared constant.
- **Expression:** The script expression in the Computations segment, defining the value of the declared constant.
- **Type:** The declared type of the constant (i.e. REAL, INTEGER, LOGICAL)
- **Value:** The calculated value of the constant following evaluation.



Errors Tab

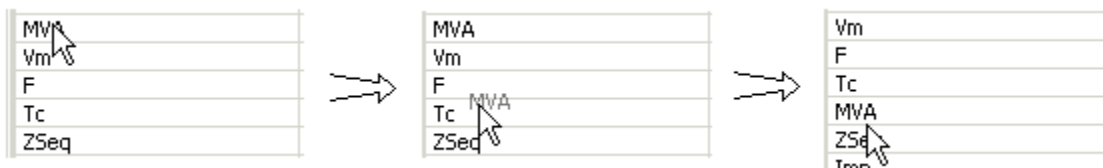
Errors:

- **Error #:** The sequential error number (i.e. order in list).
- **Trace:** A description of the actual error. This is normally a message from the internal compiler.

## Formatting Viewed Properties

Once the *Component Properties Viewer* has been opened, the data is open to some simple formatting. Although property attributes cannot be altered, users may adjust the order in which properties appear – this is useful if the data is to be saved to a file for example.

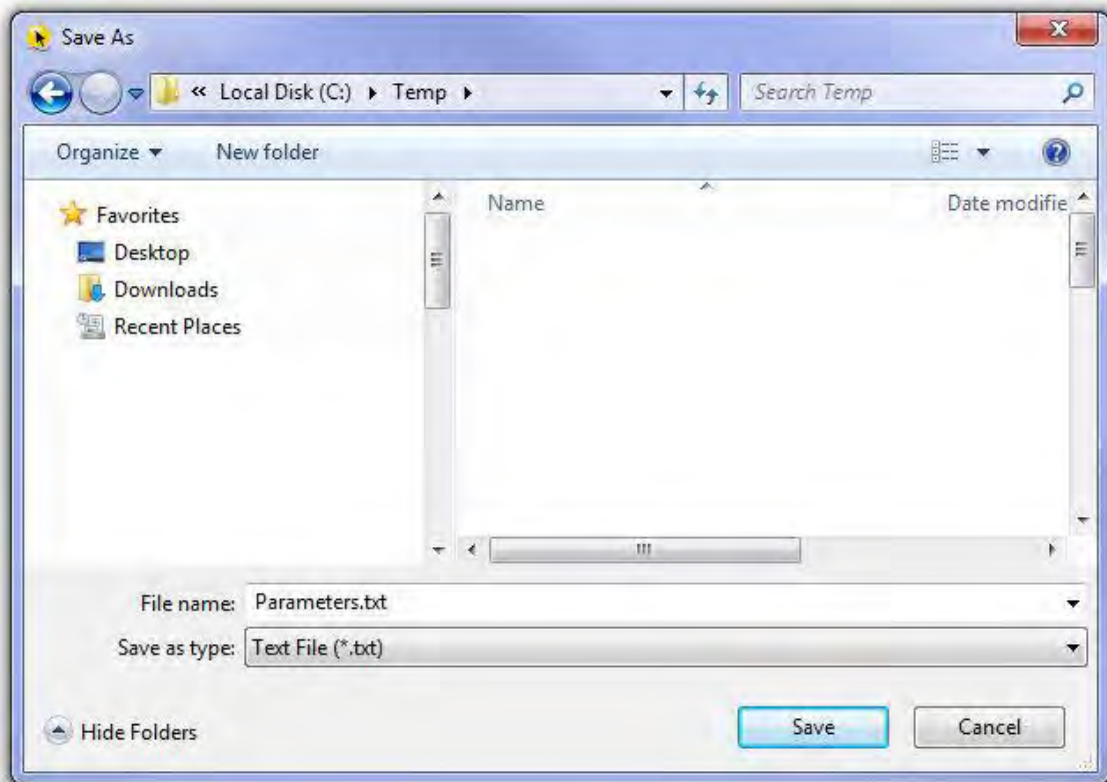
**Left-click and hold** a listed property in the display window. **Drag** the mouse pointer to where you want to place it and **release** the mouse button.



## Saving Component Properties to a File

Component data can be saved to a text file at any time. Select the desired tab and press the **Save As...** button to bring up the *Save As* dialog window.





Select a folder, enter a filename, and press the **Save** button.

## Copying Data to Clipboard

Specific rows of property data can be saved to the clipboard at any time as follows:

- To copy a single row of data, right-click within the row, and select **Copy Data**.
- To copy multiple consecutive rows, left-click on the first row, hold down the **Shift** button, left-click on the last row, and select **Copy Data**.
- To copy multiple rows that are not consecutive, hold down the **Ctrl** button, left-click in each row to be selected, and select **Copy Data**.

## Changing to Single-Line View

If the component being edited is a three-phase electrical component, then it is likely there will be a choice parameter entitled **Graphics Display** on the main properties page.



To toggle the component between single-line and 3-phase view, open the component parameter dialog, select the desired view from this field, then press **OK**.

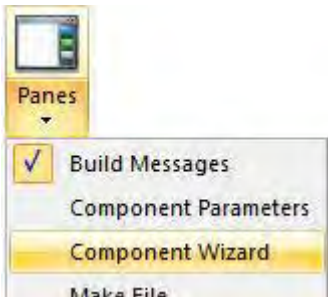
## Creating a New Component, Module or Transmission Wire

New components (including modules and transmission segments) are created by using a utility called the *Component Wizard*. The Component Wizard generates a new definition, and then creates a single instance of it (by default) for placement on the Schematic canvas. The user may then continue with the design, given this basic new component.

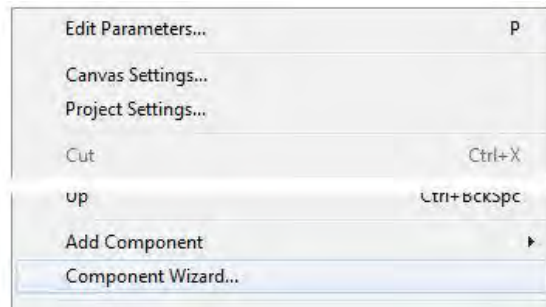
The following topics describe the steps involved when navigating the Component Wizard. For more on component design, see the chapter entitled Component Design.

### The Component Wizard

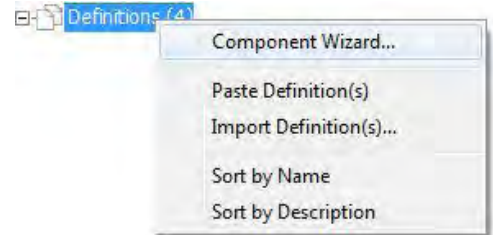
Open the Component Wizard pane by selecting either **Component Wizard** from the **Panes** drop list in the **View** tab of the ribbon control bar; or, move the mouse pointer over a blank area of any page, right-click and select **Component Wizard**; or, right-click on the definitions branch under the project node in the primary workspace window.



Ribbon



Schematic Canvas



Definitions Branch

In either case, the *Component Wizard* pane will appear:

## General

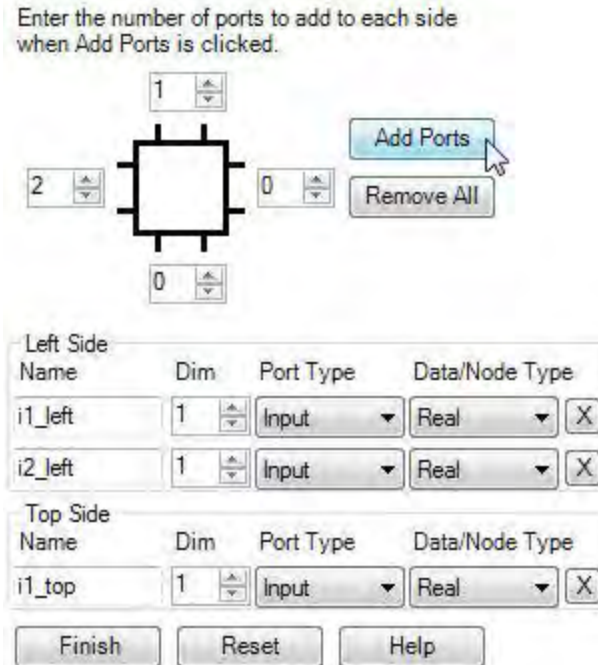
Enter all parameters required according to the descriptions below.

- **Name:** Enter a name for the new component definition. This name must conform to Fortran standards (i.e. it cannot begin with a number or contain any spaces or other illegal characters).
- **Title (Optional):** If text is detected in any of the three fields provided in this section, a text label containing the text will be added to the component graphic.
- **Module:** Select this check box if you want the new component to be a module (also referred to as a page component or a sub-page).
- **Create Definition Only:** Select this option to create the definition only, without attaching an instance to the mouse pointer.

## Ports

Enter the required parameters required according to the descriptions below.

Click the **Add Ports** button after adding how many ports you will require at each side of the component graphic. Depending on what you selected, additional input fields will appear.



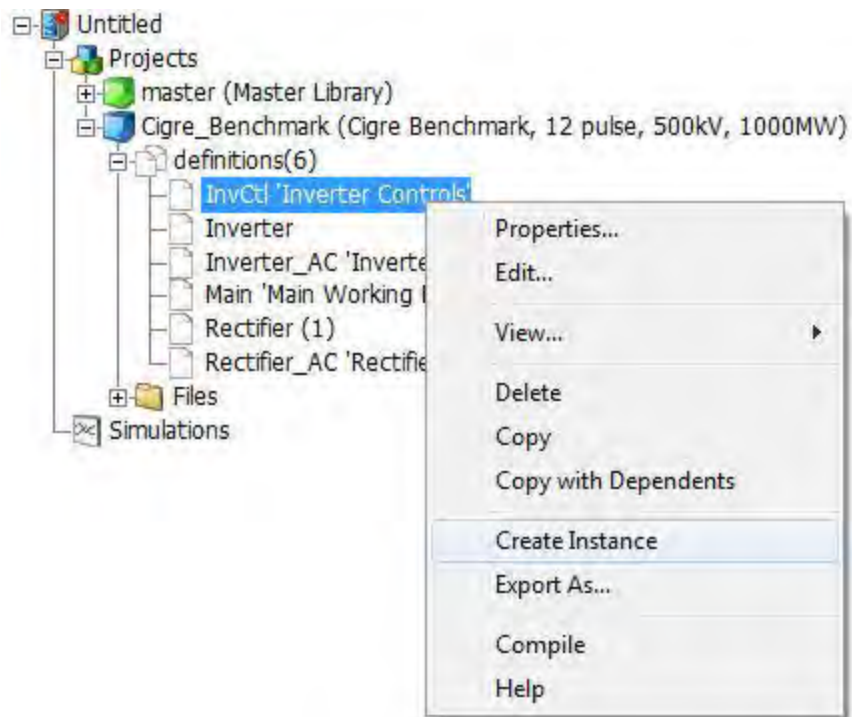
- **Name:** Enter a *Symbol* name for the port connection. This name must conform to Fortran standards (i.e. it cannot begin with a number or contain any spaces or other illegal characters).
- **Dim:** Enter a dimension for the port connection. Entering 0 indicates that this port is to assume the dimension of whatever it is connected to. Entering a 1 defines a scalar.
- **Port Type:** Select the port connection type here. Note that for *Input Data* type, the *Component Wizard* will automatically draw an input arrow graphic for the port connection.
- **Node/Data Type:** Select the type of electrical or data port connection here.

For more information on port connection types (i.e. electrical, data, etc.) see Port Connections.

Press the **Finish** button if you are satisfied that everything is correct. Press **Reset** to clear to defaults.

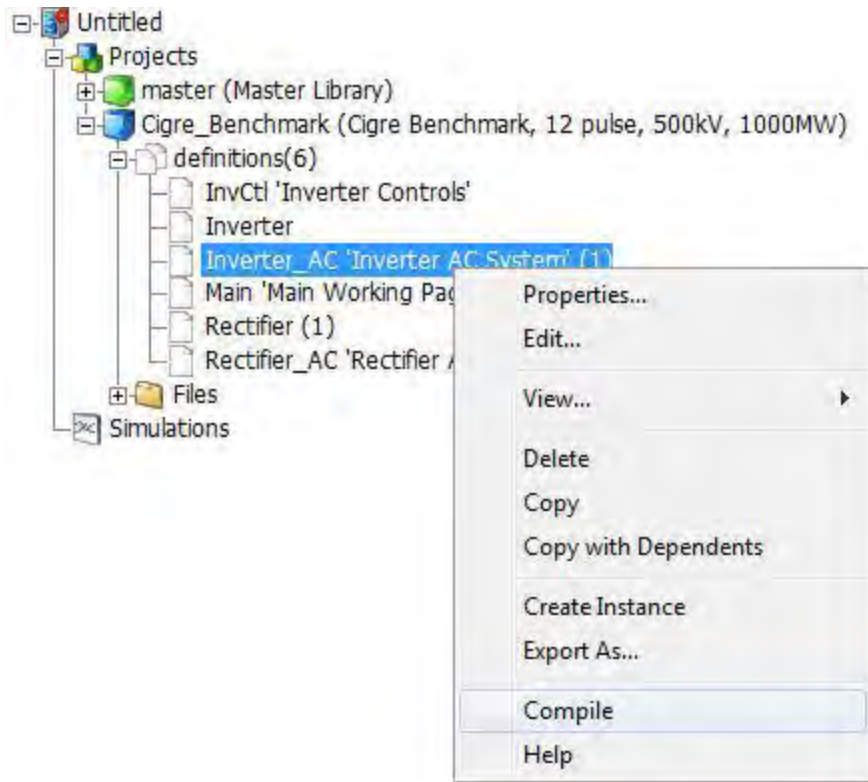
## Creating the First Instance of a Definition

In the workspace primary window, navigate to the *definitions* branch. Right-click the desired definition from the list and select **Create Instance**. Then, paste the newly created instance on the Schematic canvas (right-click on the canvas and select **Paste**).



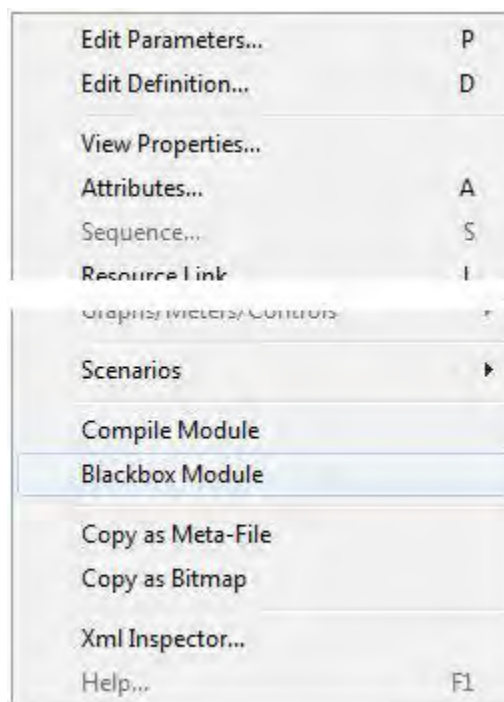
## Compiling an Individual Module Definition

In the workspace primary window, navigate to the definitions branch. Select the desired module definition from the list, right-click and select **Compile**.



## Blackbox a Module

While on the schematic canvas, right-click on the module component and select **Blackbox Module**.



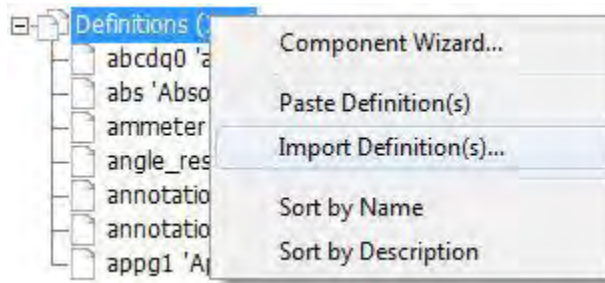
For more information on blackboxing, see Blackboxing Modules.

## Importing/Exporting Definitions

Definitions may be imported or exported to and from projects, or exchanged between users, by saving the definition as a *Definition (\*.psdx)* file. A definition file contains the XML data elements defining the component or module as it would appear in a case or library project (\*.pscx or \*.pslx) file. For more on the definitions branch and definition files, see The Secondary Window in Chapter 4.

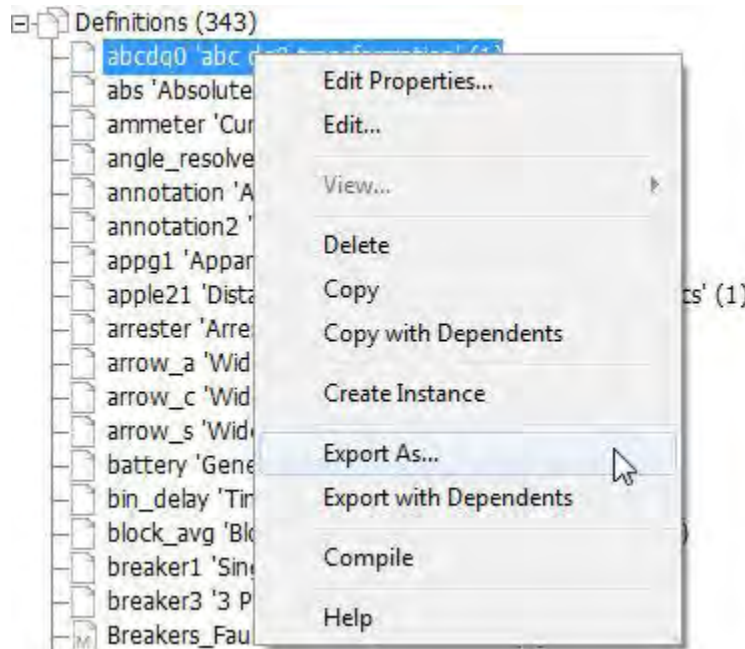
### Import a Definition

In the workspace primary window, left-click on the [+] box beside the *definitions* branch to expand the definitions list. Right-click on the *definitions* branch itself and select **Import Definition(s)...** When prompted, browse to and select the definition to be imported, then press **Open**.



### Export a Definition

To export a single definition (i.e. save the definition to a Definition (\*.psdx) file), left-click on the [+] box beside the definitions branch. Right-click on the desired definition from the list and select **Export As...**



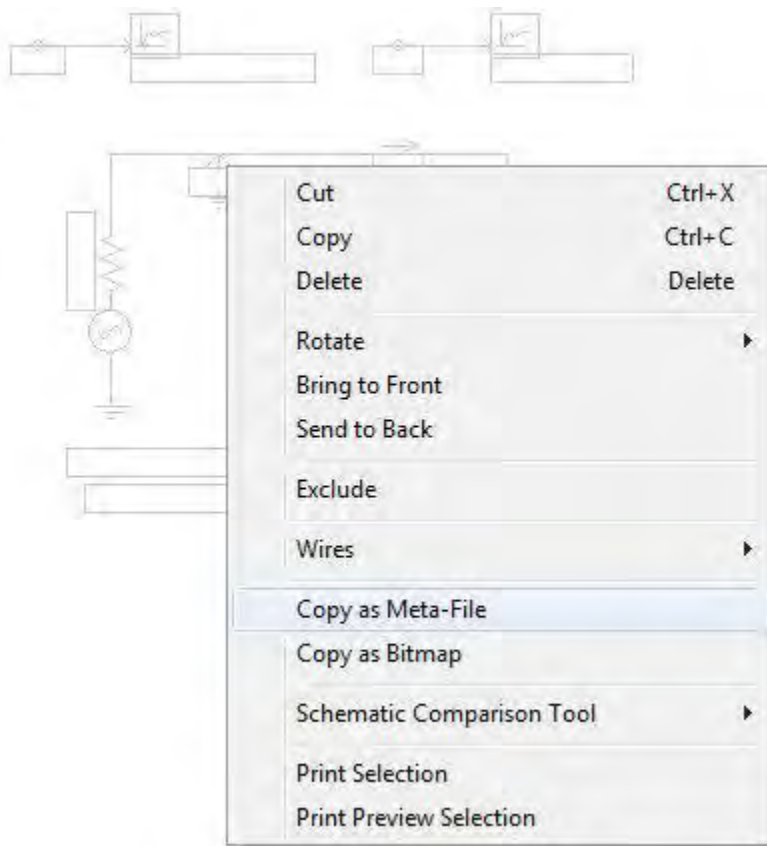
## Export a Definition with Dependents

To export a module definition hierarchy (i.e. save all the dependent definitions to a Definition (\*.psdx) file), left-click on the [+] box beside the definitions branch. Right-click on the desired definition from the list and select **Export with Dependents**.

**NOTE:** If the definition you are exporting is a module, and the module contains dependent (child) modules, then the dependent module definitions will also be included in the export file. When later imported, all definitions will be included.

## Copy as Meta-File or Bitmap

Bring up the desired area to be copied in the Schematic canvas. Select the objects to be printed (see Selecting Objects in this chapter), right-click and select **Copy as Meta-File** or **Copy as Bitmap**.



Then, simply paste the copied selection into a report program of your choice.

## Printing a Module Page

Bring up the desired module canvas in Schematic view. Right-click on a blank portion of the canvas and select **Print Page** – a print dialog window will appear. Adjust the printer properties and click **OK**.

To preview before printing, bring up the desired module canvas in Schematic view and right-click on a blank portion of the canvas. Select **Print Preview Page**. Press **Close** to return to the Schematic view.



## Tutorial: Creating a New Project

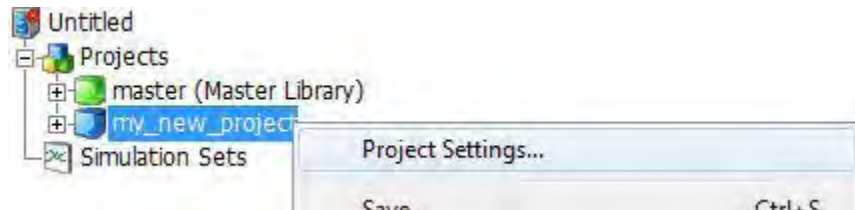
# Creating a New Case Project

Press the **New** button in the PSCAD tab of the ribbon control bar.

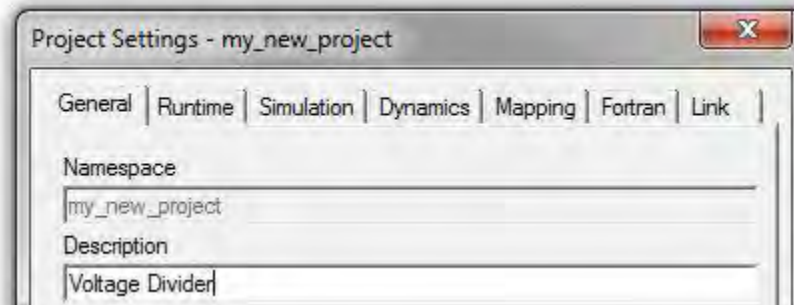


Enter a name for the project, as discussed in the section entitled Creating a New Project.

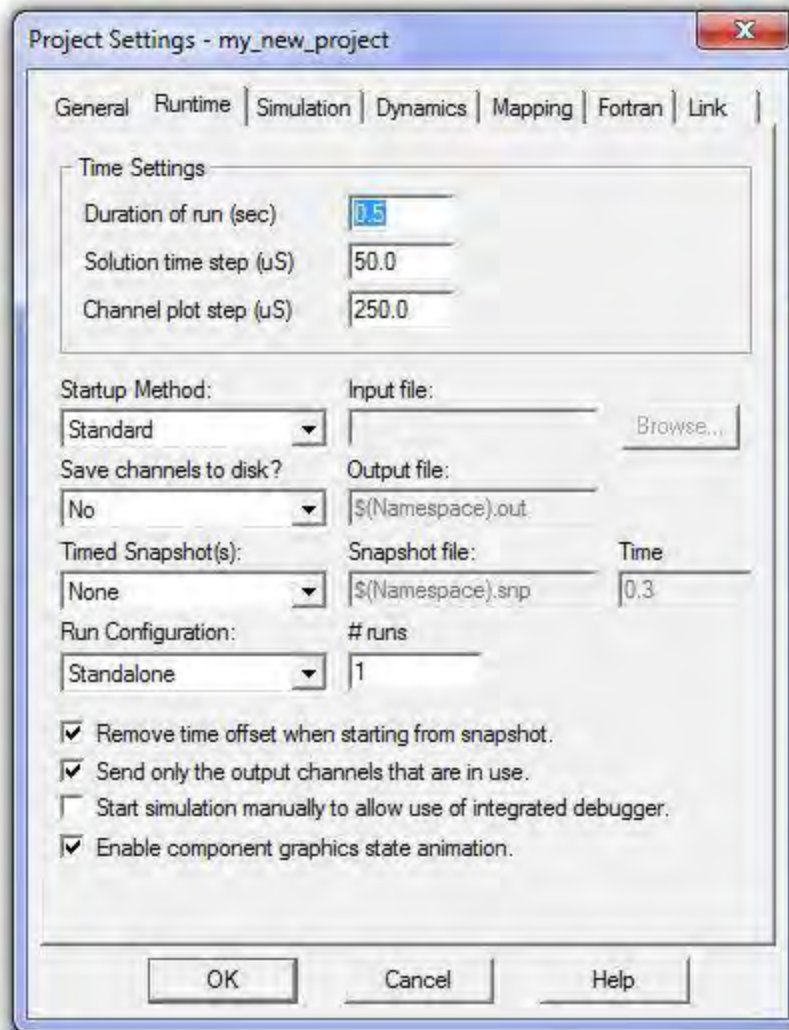
A new project will appear in the workspace window. Right-click on the project name and select **Project Settings...**



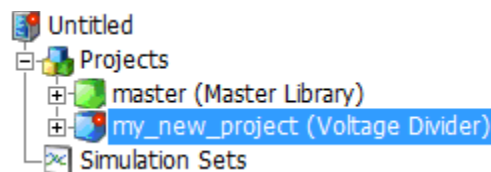
This action will open the *Project Settings* dialog. Click the **General** tab at the top of the dialog, left-click inside the **Description** field and type a description for the case, say, 'Voltage Divider'.



Click the **Runtime** tab and get familiar with the inputs displayed therein (see Runtime for details on parameters).



The project will be automatically set up with a run duration of 0.5 seconds with a 50  $\mu\text{s}$  time step by default (these settings are sufficient for now). Click the **OK** button. The description you entered should now appear in the workspace primary window as part of the project name display.



## Saving the Project

To save your newly created project, right-click on the project name in the workspace and select **Save** from the pop-up menu.

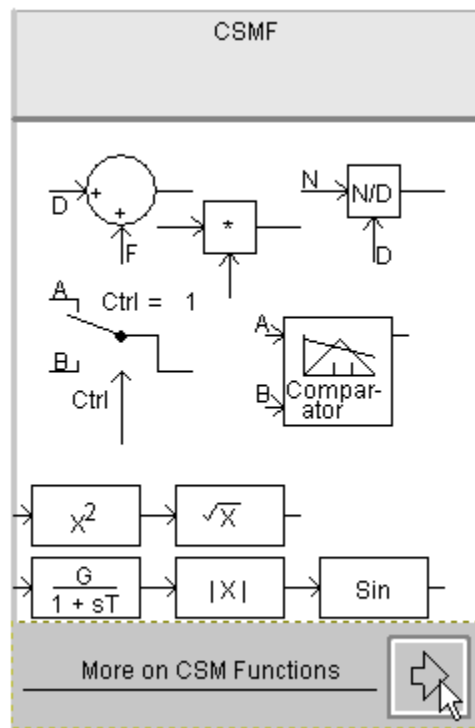


The red modified indicator on the project icon in the workspace will disappear.

## Opening the Master Library

The first project listed in the workspace primary window is always the *Master Library* (master.pslx). It contains most of the components you will ever need to build any circuit. All of the components that will be used to create this new case project example are available there.

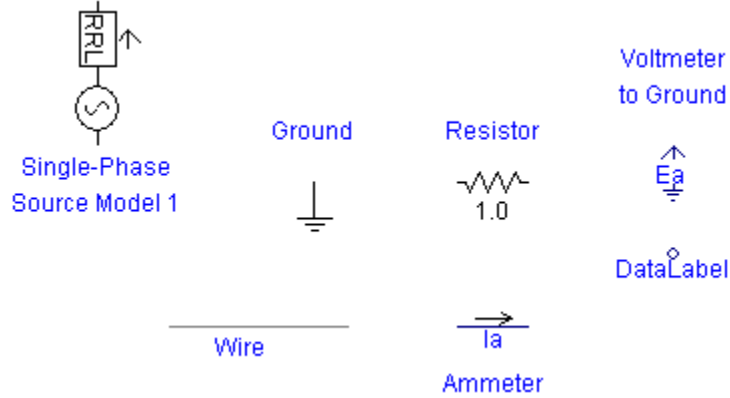
To open the master library, simply click on it in the workspace. The master library is organized as a group of modules; each opens into another canvas window, giving access to all of the components belonging to that group. To open a particular group, **left double-click** the arrow in the bottom-right corner of the module (note that this action is dependent on the Drill Down setting in the Application Options):



You may also access master library components using the *Components* tab or the *Models* tabs in the ribbon control bar. See *Adding Components to a Project* for more details.

# Assembling a Voltage Divider Circuit

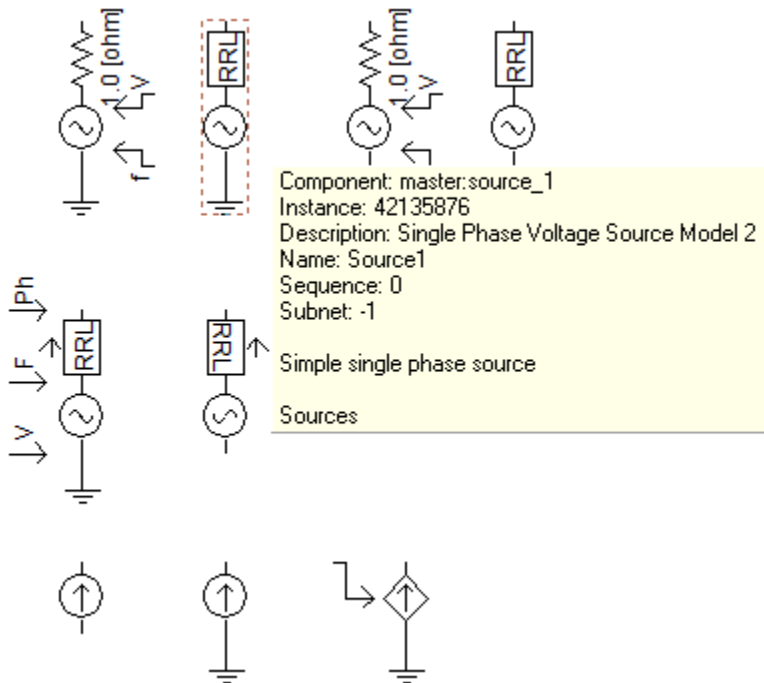
The voltage divider circuit in this example will use eight different components as shown below.



**NOTE:** Initially, it may be a challenge to navigate through the master library and find these components one by one.

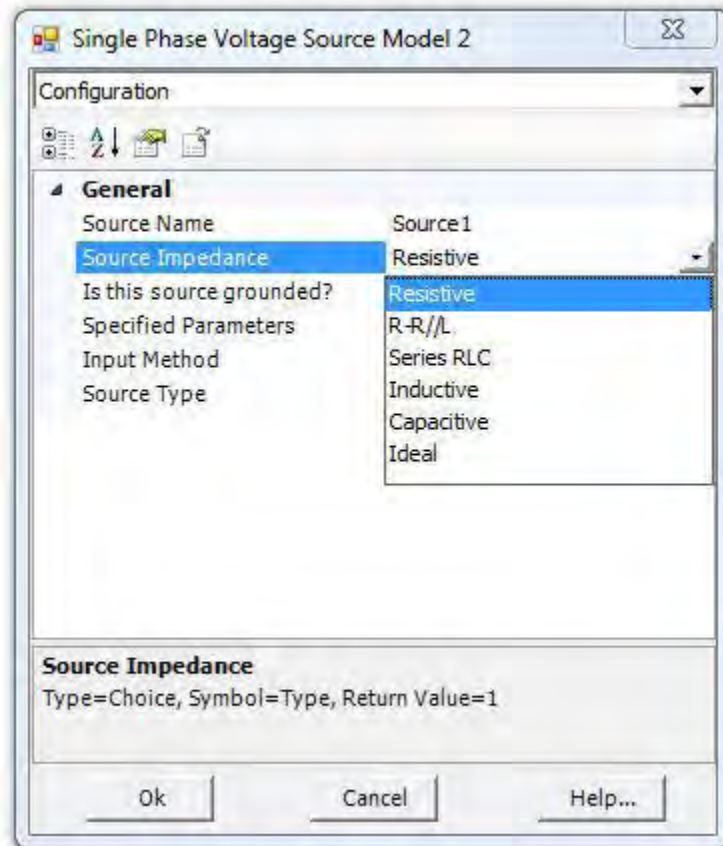
## Locating the Single-Phase Source

The first step is to locate the single-phase source model to be used in your circuit (in the master library *Sources* module). There are three different source models available – we will be using the Single-Phase Voltage Source Model 2 component.



Source 1 Model Within Sources Module in Master Library

Once you find it, add it to your new project main page using one of the techniques outlined in Adding Components to a Project in this chapter: Move the source component to an appropriate place on the page. Left double-click on the component to bring up the component parameters dialog (see Editing Component Parameters in this chapter). On the **Configuration** page, change the **Source Impedance Type** Choice List to 'Resistive'.

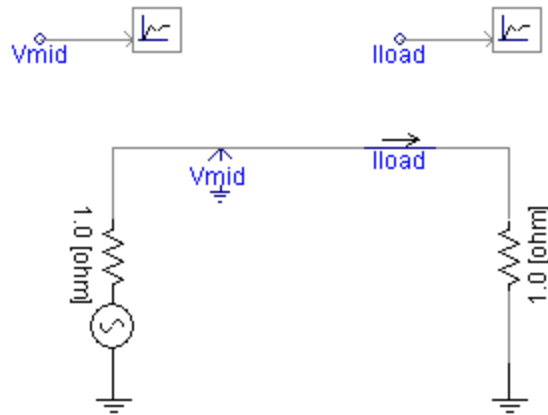


On the **Signal Parameters** category page, change the **Mag. (AC:L-G,RMS DC:Pk)** input field to *70.71 [kV]*. This will give an internal source voltage magnitude of *100 kV* peak. Press **OK**.

Save the project.

## Add and Assemble

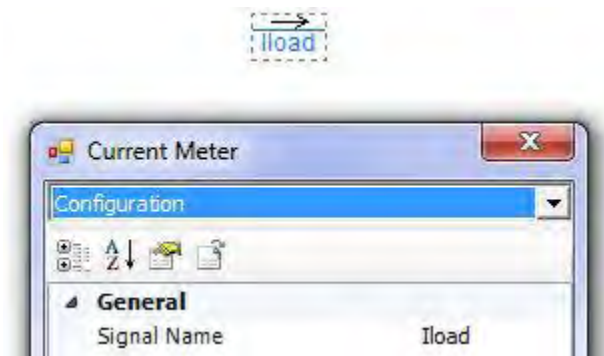
The next step is to add the remaining components (i.e. Wire, Resistor, Current Meter, Data Label, Output Channel, and Ground components). These components are all available on the Components tab of the ribbon control bar. Arrange all components to form the simple voltage divider shown in the following diagram:

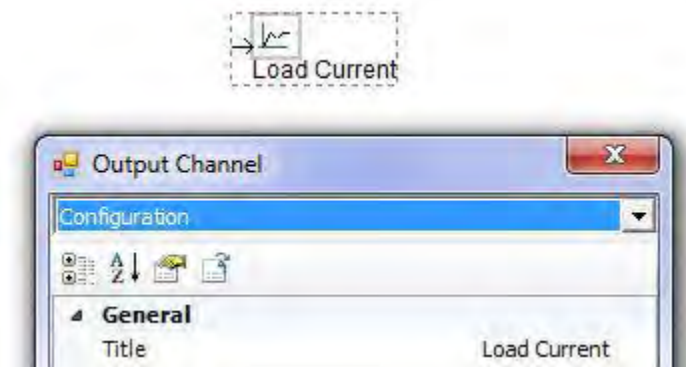
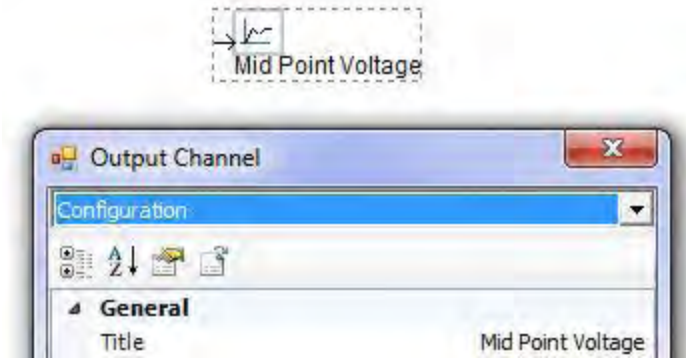


Save the project. See Adding Components to a Project, Selecting Objects and Moving or Dragging an Object in this chapter for more details.

## Editing the Remaining Component Parameters

Use the properties shown below for the remaining components (see Editing Component Parameters in this chapter). Only the properties to be changed from their default values are mentioned. Use the resistor with its default properties.





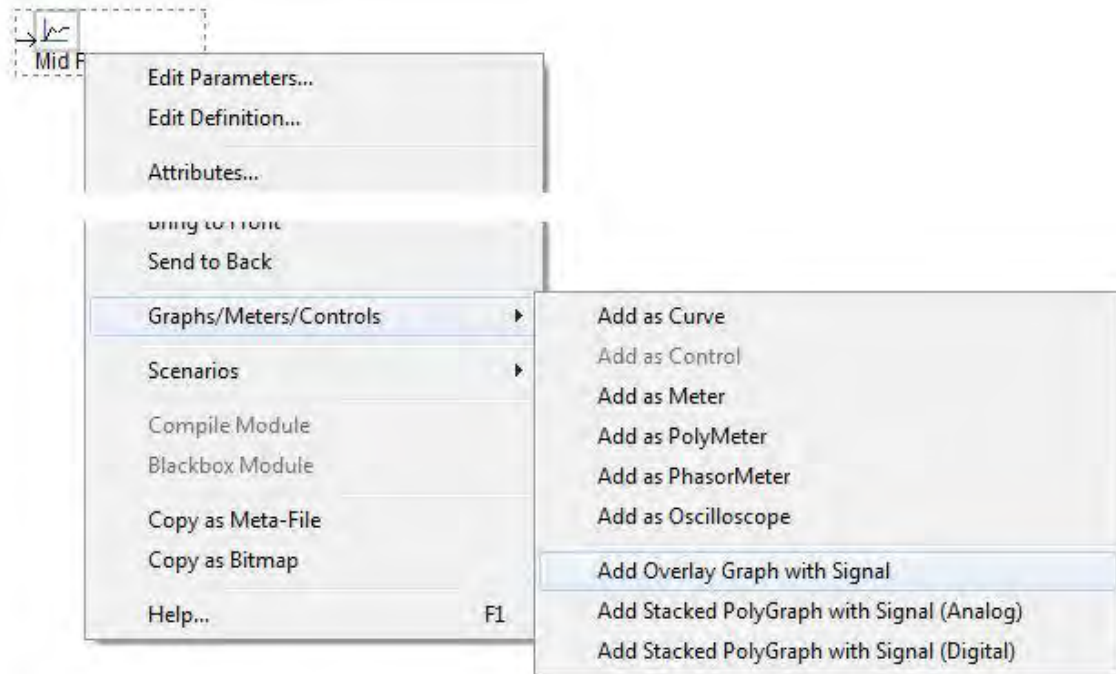
Save the project.

## Plotting Results

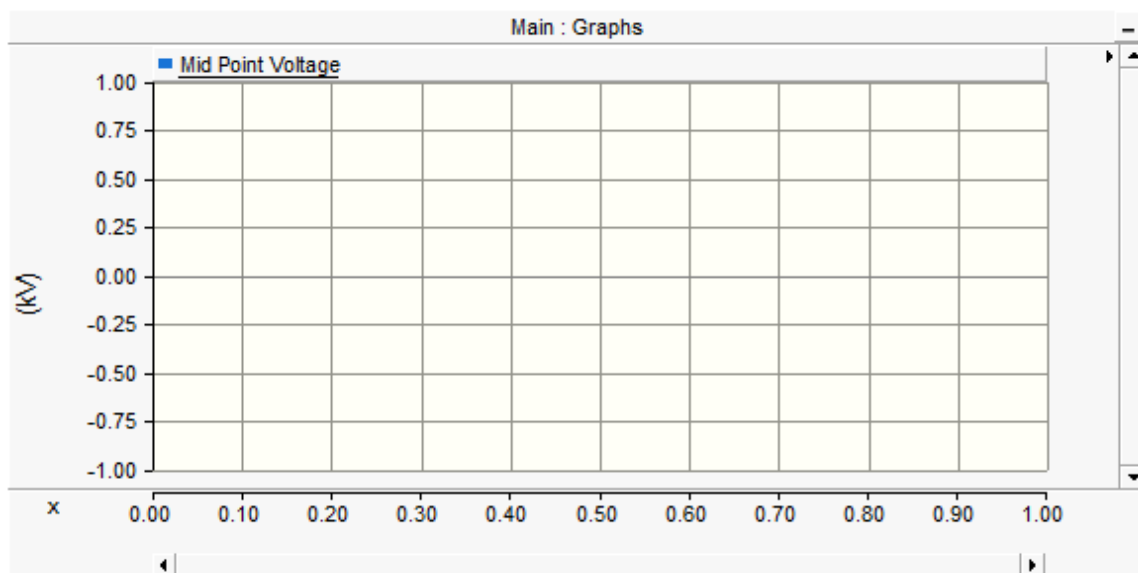
In order to view any results from our voltage divider circuit, we must add a Graph Frame and set it up to display the waveforms. The following sections describe only those aspects of plotting needed for this exercise. See Preparing Data for Control or Display in chapter 6 of this manual for further reading.

### Adding a Graph Frame

Right-click on the Output Channel component called 'Mid Point Voltage' to bring up the pop-up menu. Select **Graphs/Meters/Controls | Add Overlay Graph with Signal**.



This will create a new Graph Frame, Overlay Graph and a Curve simultaneously as shown below:

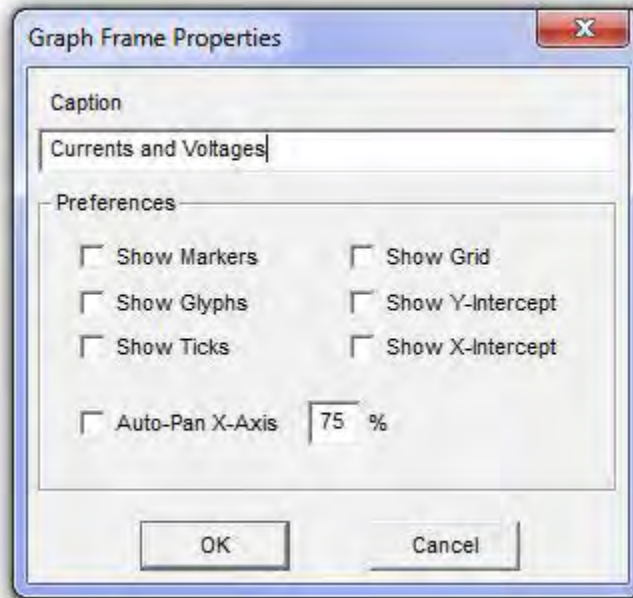


Right-click on the graph frame title bar (top bar on the plot labelled 'Main : Graphs') and select **Edit Properties...** from the pop-up menu.





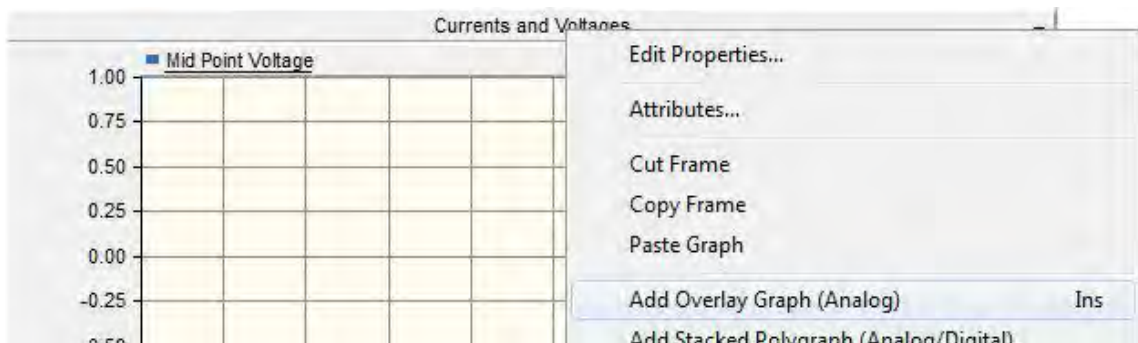
This will bring-up the *Graph Frame Properties* dialog window. In the **Caption** field, change the title to 'Currents and Voltages'. See Graph Frames in chapter 6 for more details on the options in this dialog.



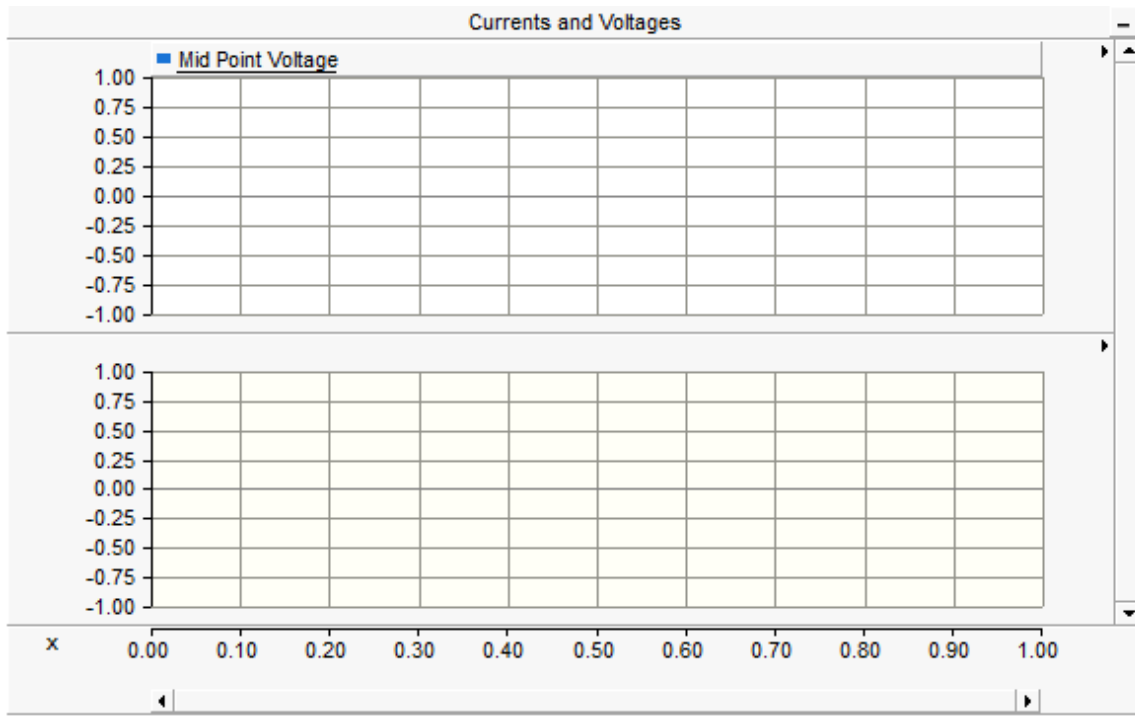
To resize the graph frame at any time, left click on the frame title bar so that grips appear. Left-click and hold on a corner grip and drag mouse outwards. Resize the graph frame to approximately 8 x 8 centimetres.

## Additional Overlay Graph and Curve

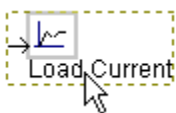
Right-click on the graph frame title bar and select **Add Overlay Graph (Analog)** (or press the **Insert** key while the mouse pointer is over the graph frame).



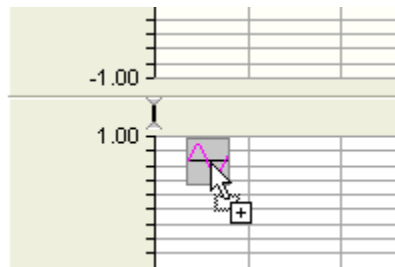
A new *Overlay Graph* will appear within the frame directly below the existing graph (see Graphs in chapter 6 for more).



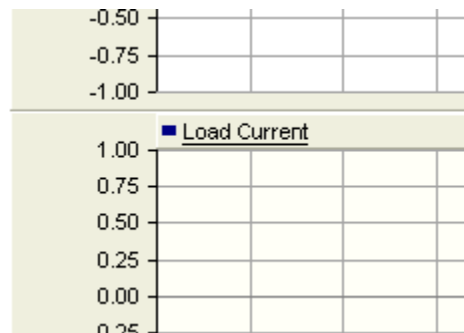
Add a curve to this graph to monitor load current: Hold down the **Ctrl** key and click and hold the **left mouse button** while over the Output Channel with caption *Load Current*. Drag the mouse pointer over the new graph and release the mouse button. A curve should appear.



Select the Output Channel (**Ctrl + left mouse hold**)



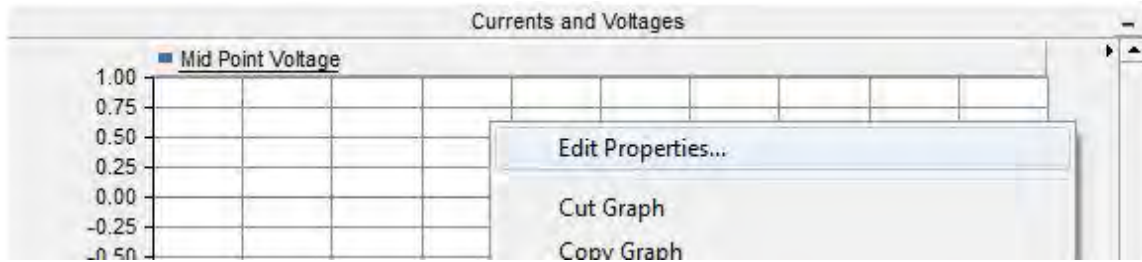
Drag the Mouse Pointer Over the New Graph



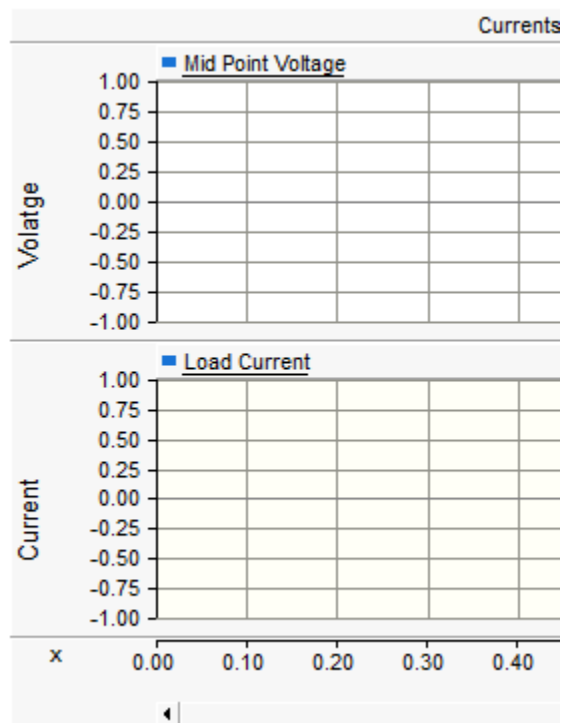
Release the Mouse Button

## Editing the Graph Properties

To customize an individual graph title and/or vertical axis label, right click over top the graph and select **Edit Properties...** Edit the graph properties as you see fit (see Graphs in chapter 6 for more).



For example, change the **Y-Axis Title** input field to show 'Voltage' on the voltage graph, and 'Current' on the load current graph. You may also want to turn on/off **Show Grid** and adjust the scaling. The graph frame should appear similar to that shown below once completed.



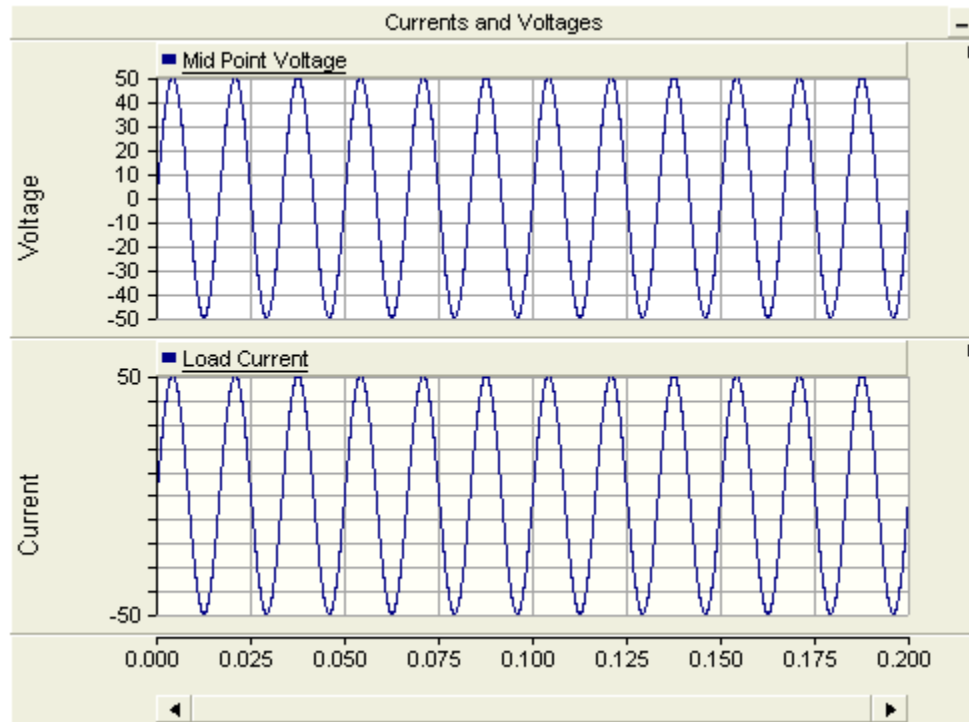
Save the project.

## Running the Project

Press the **Run** button in the **Home** tab of the ribbon control bar to run the simulation (see Running a Simulation in this chapter).



This is the last step, assuming there are no errors. If there are any, they will be logged in the Build Messages pane. See Error and Warning Messages for more details. Your simulated results should look similar to the following once the simulation has completed.



**NOTE:** It may be necessary to readjust your x and y-axes, as well as zoom and time frame. See Online Plotting and Control for more details.

## Multiple Instance Modules (MIM)

Since its introduction in PSCAD V3, the module (also known as a sub-page or page module) has been a useful tool in enhancing the organization and navigation of projects. Modules are themselves components, but possess a canvas of their own, and thereby add a third dimension to an otherwise flat environment. Their inception was in response to the growing size of simulation drawings in PSCAD V2, where a two-dimensional drawing canvas proved cumbersome when dealing with larger projects.

The original modules however lacked an important attribute of their standard component counterparts: The ability to be instantiated (i.e. two or more instances based on the same definition). This was primarily due to complications in the mapping of data signals, such as plotted curves and meter outputs. In addition to this, a module definition could potentially be part of other module definitions, and there was again a difficulty in tracking multiply-instanced modules of this kind. Essentially, the architecture of the software was, at the time, not optimized for this sort of thing and as a result, the ability to instantiate a module was restricted to a single instance for simplicity, thereby making the definition and single instance one in the same.

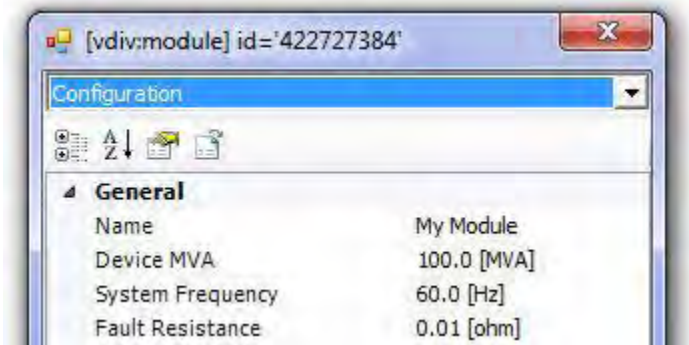
The absence of the ability to multiply-instance modules remained throughout the life of both the V3 and V4 releases, but was finally remedied in PSCAD X4. This was made possible by a complete rework of the PSCAD program architecture towards a more data-centralized model. This new architecture greatly simplified the mapping and bookkeeping involved in allowing effective use of multiple instance modules. A module may now be copied and pasted easily just like any other component, even if it contains many module components within it!

The capabilities of this feature may not be immediately obvious, but if not properly utilized it can be dangerously powerful. For example, if a module containing say 100 Output Channel components is instantiated, 100 new signals would be added for every instance. This can quickly overwhelm the memory capacity of your workstation, and/or greatly affect simulation speed. This becomes even more apparent if the module being copied contains other modules that contain more Output Channels. This section is designed to ensure that projects with multiple instance modules are designed properly from the very beginning, thereby avoiding problems like that described above.

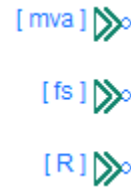
## Parameters

With the establishment of multiple instance module capabilities in PSCAD, it was also a prudent time to introduce parameter functionality as well. As veteran users know, module components did not support parameters in the past, and relied purely on port connections to transfer data signals into and out of the canvas. Parameter dialogs can now be designed for module components in the exact same manner as they are for standard components, and every instance of a module component may possess unique parameter values.

When it comes to porting parameter values onto or from the canvas, they are treated in exactly the same way as port connections – that is, each parameter requires a matching Import or Export component on the canvas. The name of the *Import* or Export component must match the Symbol name of the corresponding parameter.



Module Input Parameters



Import Connection Components on Canvas

For more details on designing component parameter dialogs, see The Parameters Section.

## Similarities and Differences

Modules and standard components are now more alike than ever: They both may possess input parameters, they both have graphics; in fact they are indiscernible from each other unless their definition is edited. The only major difference between them is that module components possess a canvas, whereas the standard components contain a section for scripting.

	Schematic	Graphic	Parameters	Script
Standard Component	x	✓	✓	✓
Module Component	✓	✓	✓	x

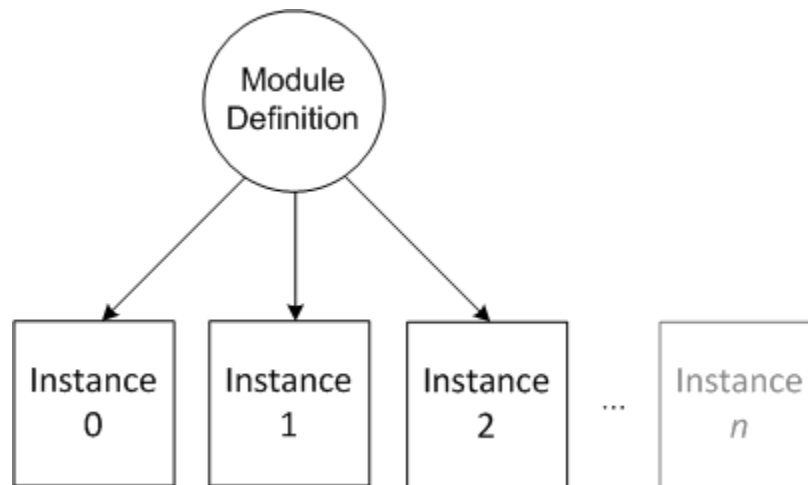
Similarities between Modules and Components

The table above hints that module components simply offer an alternative way in which to build a model: With a module component, the design is accomplished graphically via a circuit schematic, as opposed to using script and code. In fact, modules must utilize standard or user-defined components in their design – components that are pieced together to form the circuit model.

## Module Definitions and Instances

Since the concept of modules and project hierarchy was introduced, the root or top-level canvas has always been what is referred to as the *Main* page. Anyone who has used PSCAD in the past has constructed a circuit starting from the Main canvas. *Main* is itself a module component and its canvas is part of its definition. Therefore, any modifications made to *Main* modify its definition, which in turn affects any instances of it. Of course, *Main* only ever has one instance, as it is the top-level module, and it, along with its definition, are included for you automatically whenever a new project is created.

The Schematic canvas is an integral part of a module component definition, and so changes made to this canvas are changes made to the definition. If a component is added to the module canvas (or deleted from it), its location moved, or even if its input parameter values are modified, the module definition is affected. This is because the components, wires and other objects combine to define what the module is meant to represent or model. By extension then, all module component instances, based on this definition, will be affected by changes to the definition. For more seasoned users, this concept may be a bit difficult to grasp at first, as in the past, module components were never allowed to possess more than one instance. Because of this fact, modules were always unique (one definition, one instance) —not so any more.

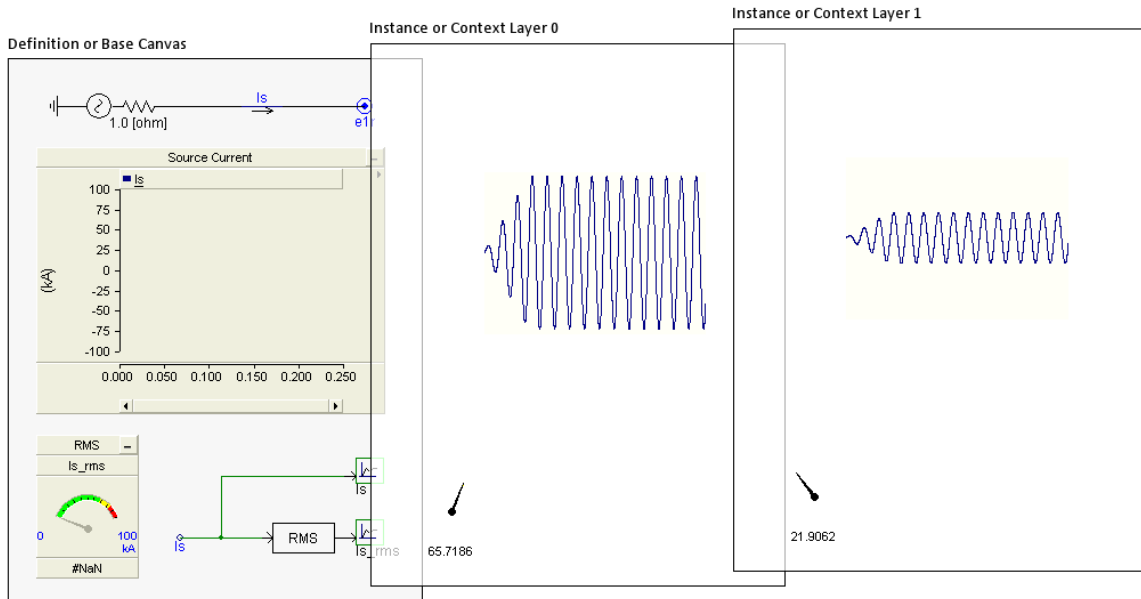


Multiple Instances Based on a Single Module Definition

With standard, non-module components, it is easy to differentiate between the components definition and instance environments; its definition is composed purely of script and code, parameters and graphics, and of course does not include a canvas. In other words, it is obvious that you are inside the definition environment when you edit the definition of a standard component. This is not true when dealing with module components. A module component utilizes a canvas to represent its definition, but at the same time, that canvas is used as part of the hierarchy of the greater circuit, meaning that one or more instances of that module component may appear in the project, all of which are used in a unique *context*.

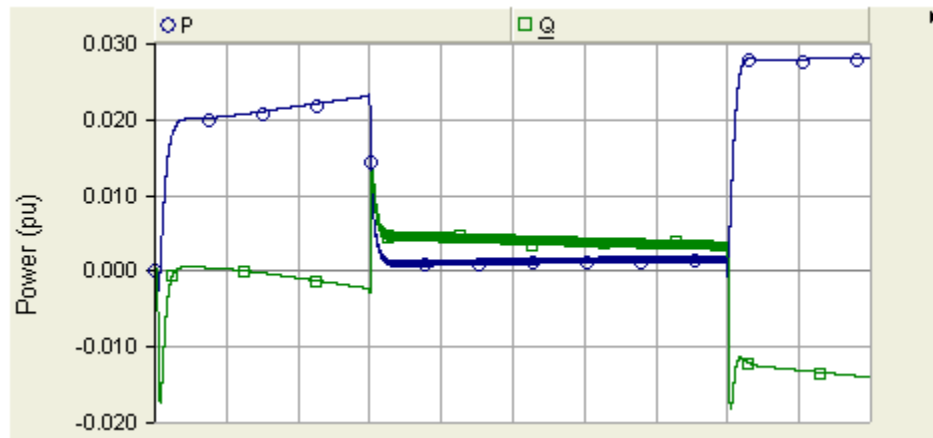
Every component, graph, control panel, etc. that is placed on a module canvas, becomes part of that module's definition. If a plot panel is resized, or a new graph is added to the panel, it modifies the definition. Changing graph settings, curve colours, zoom settings all affect the definition. The same goes for control panels, meters and online controls. If a Slider component interface for example is adjusted (say from *0.12* to *0.23*), it affects the definition and therefore all of the module instances based on it. This is because the *Slider* component interface is *defined* or sourced directly on the module canvas.

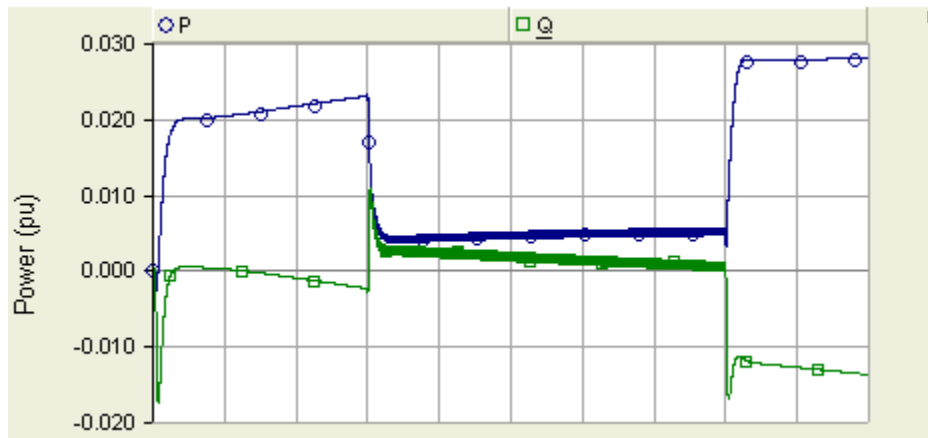
It is important to distinguish between the settings of plotting and control tools, and the actual signals themselves. The signals being displayed on graphs or meters may not necessarily be defined purely as part of the module definition. If a signal is sourced from within a module, say for example a breaker control signal, and is not influenced in any way by signal values external to the module, then the signal value will be identical in all module instances based on that definition. However, if the signal value is based or influenced by an external signal, such as an imported parameter, then the signal value may change depending on the module instance (or context), and in such cases may be considered an *instance-based signal*. This means that although the curve colour or thickness of a curve is a definition-based setting — and will be identical in every instance— the actual data signal *values* are unique from one module instance to another. This allows us to visualize the concept of a definition canvas and instance context, as a base layer canvas, with a transparent overlay representing the context of each module instance.



Module Definition Canvas with Data Values in Two Unique Instance Contexts

The image below shows the subtle differences between data signal values in two, separate instances of a module – both are based on the same definition. Notice that the graph and curve settings are identical (i.e. colour, glyphs, etc.), but their values are slightly different. This is because the data signals plotted are sourced from outside the module canvas, and are ported in through an input parameter or port connection.

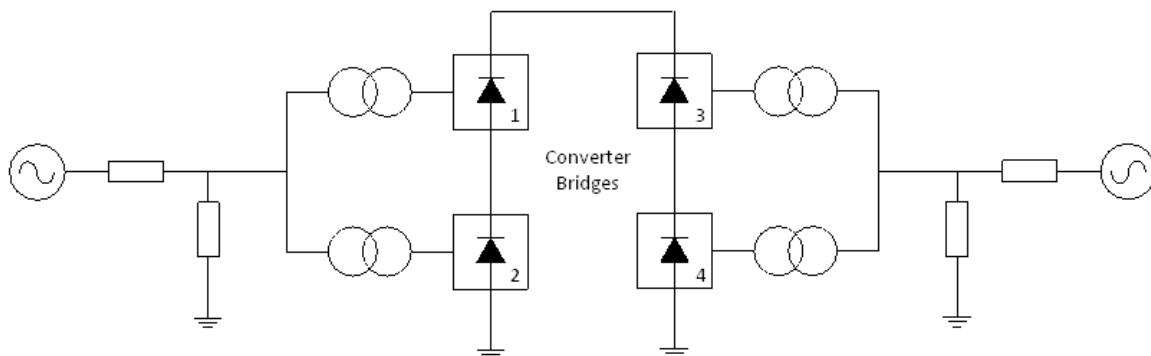




Same Graph from Two Separate Module Instances Based on the Same Definition

## A Practical Example of Multiple Module Instances in Use

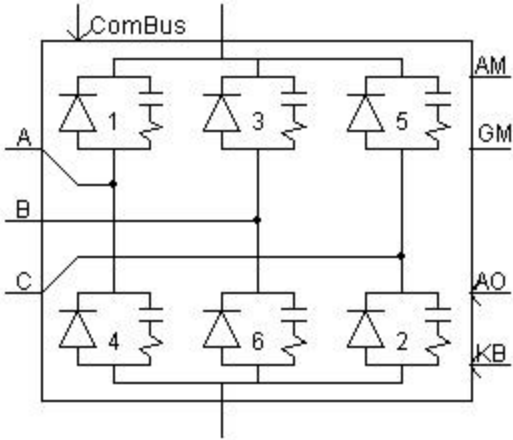
Before continuing on, it would be prudent to pause and look at a practical situation where multiple module instances could be put to good use. Consider a three-phase bridge unit that is to be used in the construction of a monopolar HVDC system. The system will utilize four bridge units in total, where each unit is identical and comes from the same manufacturer.



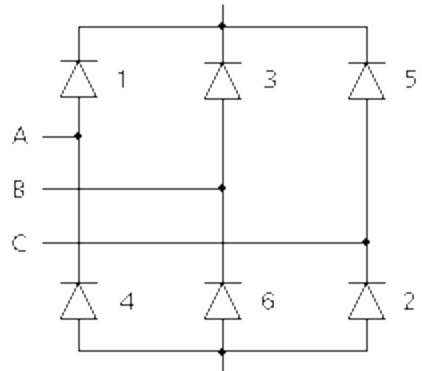
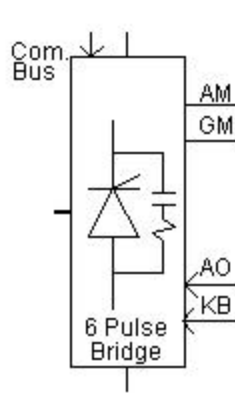
Monopolar HVDC System with Four Converter Bridge Units

Since the converter bridges are all identical, it makes sense that a bridge unit only be defined once, and used multiple times. This helps to alleviate maintenance headaches and to ensure that changes made to one unit are applied to all. Veteran PSCAD users will understand that this concept has been utilized since the beginnings of the application; up until recently however, this has been accomplished only through the use of standard (non-module) components. In fact, the master library already includes a bridge component called 6 Pulse Bridge. This component may be instantiated many times when constructing systems like that shown above.





6 Pulse Bridge Component Graphic



6 Pulse Bridge Equivalent Circuit

Standard components however are limited in flexibility, in that if they are to be combined with other components to form another single device, they must be reprogrammed into a single unit (i.e. combining two bridge units in series to form a 12-pulse converter for example). Reprogramming these models can prove cumbersome depending on the complexity of the model. With module components however, the reprogramming is performed for you by the application; the user is left to simply construct a circuit graphically.

Let us conceptually build a three-phase bridge from scratch, using only module components: The most elemental (i.e. indivisible) component of the converter is the individual thyristor valves (ignoring the snubber circuits). This is where we should start the design. In practice, thyristor valves are combined in series to form a valve group. The valve group will be our first module component, where the canvas of the valve group definition consists of a string of series-connected, non-module thyristor components. The module component will also require a graphic to represent the string of thyristors when the component is placed on the canvas of its parent module.

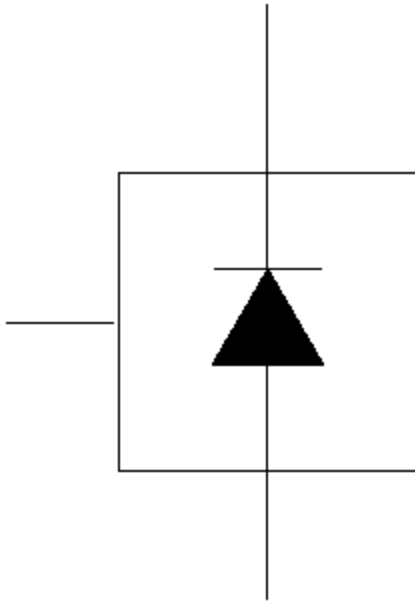


Valve Group Module Component Graphic

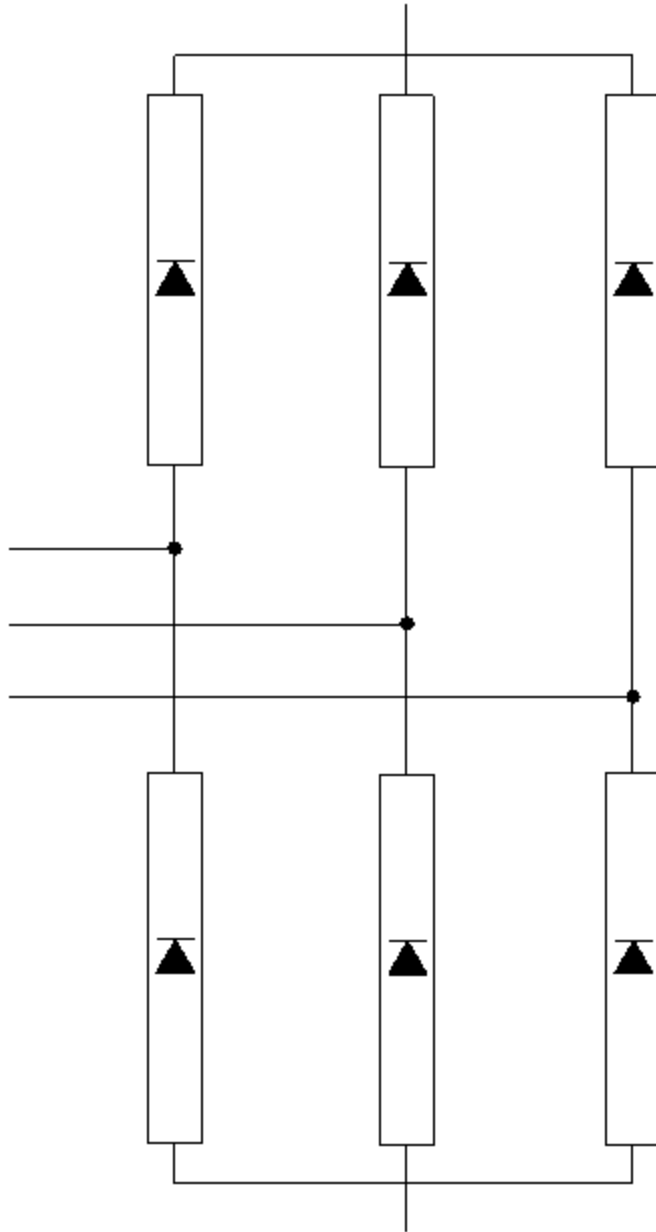


Valve Group Module Component Canvas

A 6-pulse bridge will consist of a total of six valve groups, each group representing a switching device: This is our second module component. The 6-pulse module component canvas will include six instances of the valve group module component. These are of course connected to form the bridge.



6-Pulse Bridge Module Component Graphic



6-Pulse Bridge Module Component Canvas

So far, we have constructed a 6-pulse bridge module definition, which contains instances of another module definition (called *Valve Group*) within it:

Module Definitions List:

1. Valve Group
2. 6-Pulse Bridge

Module Instance Hierarchy:

1. 6-Pulse Bridge

1. Valve Group
2. Valve Group
3. Valve Group
4. Valve Group
5. Valve Group
6. Valve Group

In the instance hierarchy above, the numbers indicate the instance number of the module (where the first instance is 0).

Our monopolar HVDC system is to contain four instances of the 6-pulse bridge we just constructed. If we assume that this system is constructed on the Main canvas, and that there are no other module components other than those above, then the project module hierarchy will be:

Module Definitions List:

1. Valve Group
2. 6-Pulse Bridge
3. Main

Module Instance Hierarchy:

1. Main
  1. 6-Pulse Bridge
    1. Valve Group
    2. Valve Group
    3. Valve Group
    4. Valve Group
    5. Valve Group
    6. Valve Group
  2. 6-Pulse Bridge
    1. Valve Group
    2. Valve Group
    3. Valve Group
    4. Valve Group
    5. Valve Group
    6. Valve Group
  3. 6-Pulse Bridge
    1. Valve Group
    2. Valve Group
    3. Valve Group
    4. Valve Group

5. Valve Group
6. Valve Group
4. 6-Pulse Bridge
  1. Valve Group
  2. Valve Group
  3. Valve Group
  4. Valve Group
  5. Valve Group
  6. Valve Group

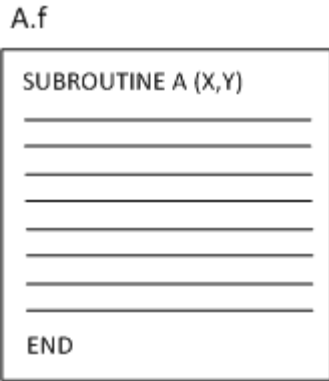
## Coding Analogies

The ability to instantiate module definitions fits perfectly into how a typical structured program is created. PSCAD is a structured code generator (presently Fortran that may include C), which combines components and modules in a project to construct and build an executable program. The project itself can be compared to a coded program in its entirety, where regular components represent snippets of inline code and modules represent subroutines.

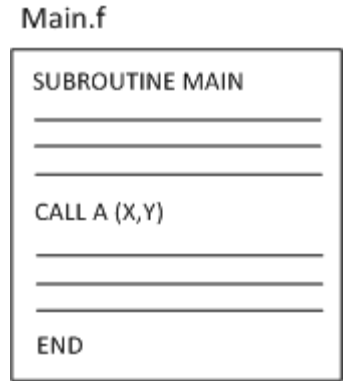
PSCAD Object	Code Equivalent
Project	Program
Module Definition	Subroutine
Module Instance	Subroutine Call
Non-Module Component	Inline Code
Port Connections/Input Parameters	Subroutine Arguments

### Analogies Between PSCAD Project Structure and Programming Structure

When a project is built, a separate Fortran (\*.f) file containing a subroutine definition is created for every unique module definition used in the project. Any instance of said module definition will appear as a call within the subroutine of its parent module (i.e. the canvas on which the module instance is placed). For example, say a module definition called *A* is created and an instance of it is placed on the Main canvas.

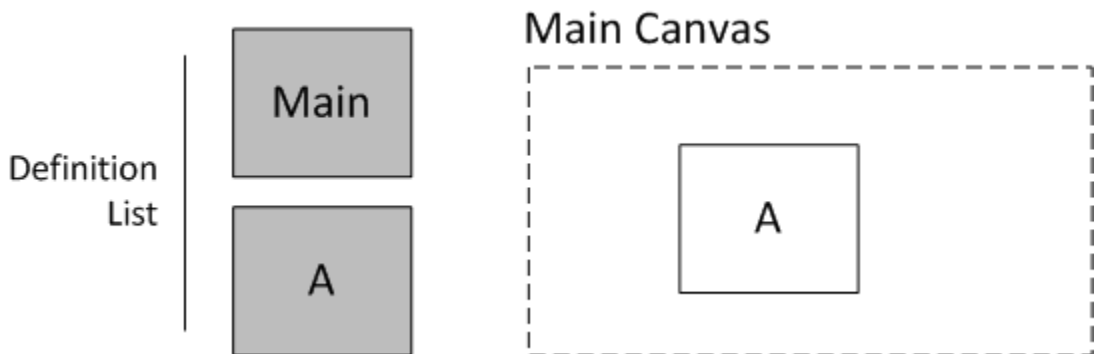


Module A Definition Fortran File



Call to Subroutine from Main Module

When the project is built, a separate Fortran file called *A.f* is created to represent the definition of *A*. The code within it (i.e. the inline code) is constructed using the non-module components placed on the canvas of *A*. An instance of *A* exists within the canvas of *Main*, so therefore it contributes to defining the subroutine for *Main* – in other words, the instance of *A* is part of the definition of *Main*. Conceptually then, this project contains two module definitions, where an instance of module *A* is set within the canvas of *Main*.



In the above example, there are two module definitions (one for *Main* and one for *A*). What would happen then if *A* was instantiated again so that there are two instances of it on the *Main* canvas? Well a new definition is not required; only an additional call statement to *A* within the *Main subroutine* is needed.

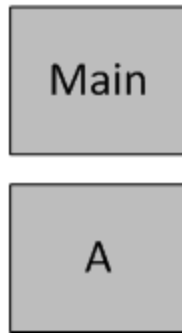
Main.f

```

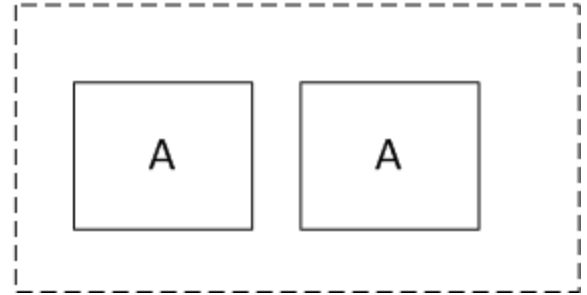
SUBROUTINE MAIN
_____
_____
_____
CALL A (X,Y)
CALL A (X',Y')
_____
_____
END
    
```

Main Definition Fortran File

Definition List



Main Canvas



Two Instances of A on the Main Canvas

Now let's complicate things a bit: Say there are two instances of another module called *B* placed on the canvas of module *A*. Since the instances of *B* are situated within *A*, then *B* becomes part of the definition of *A* – that is, calls to *B* will appear in the subroutine of *A*.

B.f

```

SUBROUTINE B (Z)
_____
_____
_____
_____
_____
_____
_____
_____
_____
END
    
```

Module B Definition Fortran File

A.f

```

SUBROUTINE A (X,Y)
_____
CALL B (Z)
CALL B (Z')
_____
_____
_____
_____
_____
_____
END
    
```

Module A Definition Fortran File with Calls to B

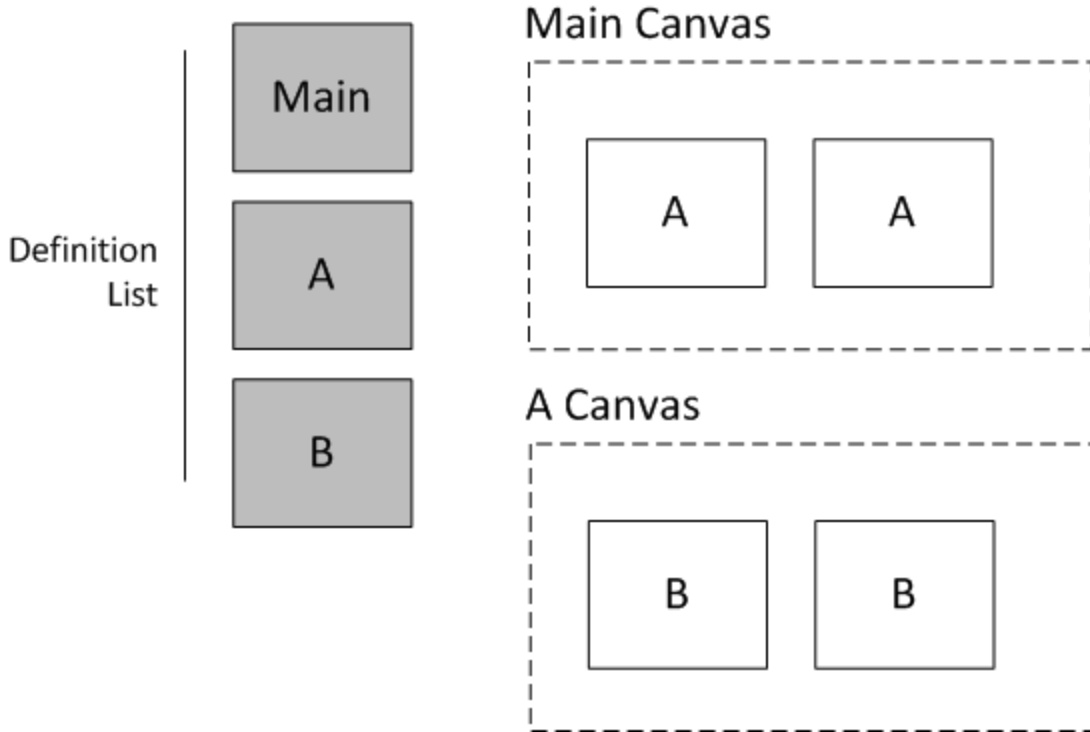
Main.f

```

SUBROUTINE MAIN
_____
_____
_____
CALL A (X,Y)
CALL A (X',Y')
_____
_____
_____
_____
END
    
```

Module Main Definition Fortran File with Calls to A

Notice above that the addition of module *B* to the project does not affect the definition of *Main*. This is because *B* is part of the definition of *A*, which appears only as an instance (or call statement) in *Main*. Conceptually:



### Instances vs. Calls

You may have noticed above that module *B* is called four times in the above project, although there are only two instances of *B*. This is because the two instances of *B* help to define *A*, and *A* is called twice from *Main*. This does not mean that there are four instances of *B*, but each of the two instances of *B* is called twice. This concept is aptly referred to as a *Call*.

	Instances	Calls
Main	1	1
A	2	2
B	2	4

Instance and Call Numbers for the Given Example

It is important to understand this concept in order to understand fully how multiple instance modules work, and how signal values are mapped and accessed within PSCAD and EMTDC.

### Subroutine Arguments

Up until now, we have not mentioned the arguments shown in both the SUBROUTINE and CALL statements above. Subroutine arguments exist whenever the module definition possesses either input parameters or port connections. In fact, this is yet another coding analogy: Input parameters and port connections are a means to transfer data defined external to the module, for use inside the module. This is exactly what subroutine arguments are used for when coding.

Say for example that our module *A* above has two connection ports *X* and *Y*. The connection ports are defined as part of the module definition, and are therefore also part of the subroutine file definition. Although the ports are named *X* and *Y* in the definition, the actual signal connected to the port can be any other pre-defined signal; be it a literal value or another variable signal.

**A.f**

```

SUBROUTINE A (X,Y)
_____
CALL B (Z)
CALL B (Z')
_____
_____
_____
_____
END

```

Module A Definition Fortran File

**Main.f**

```

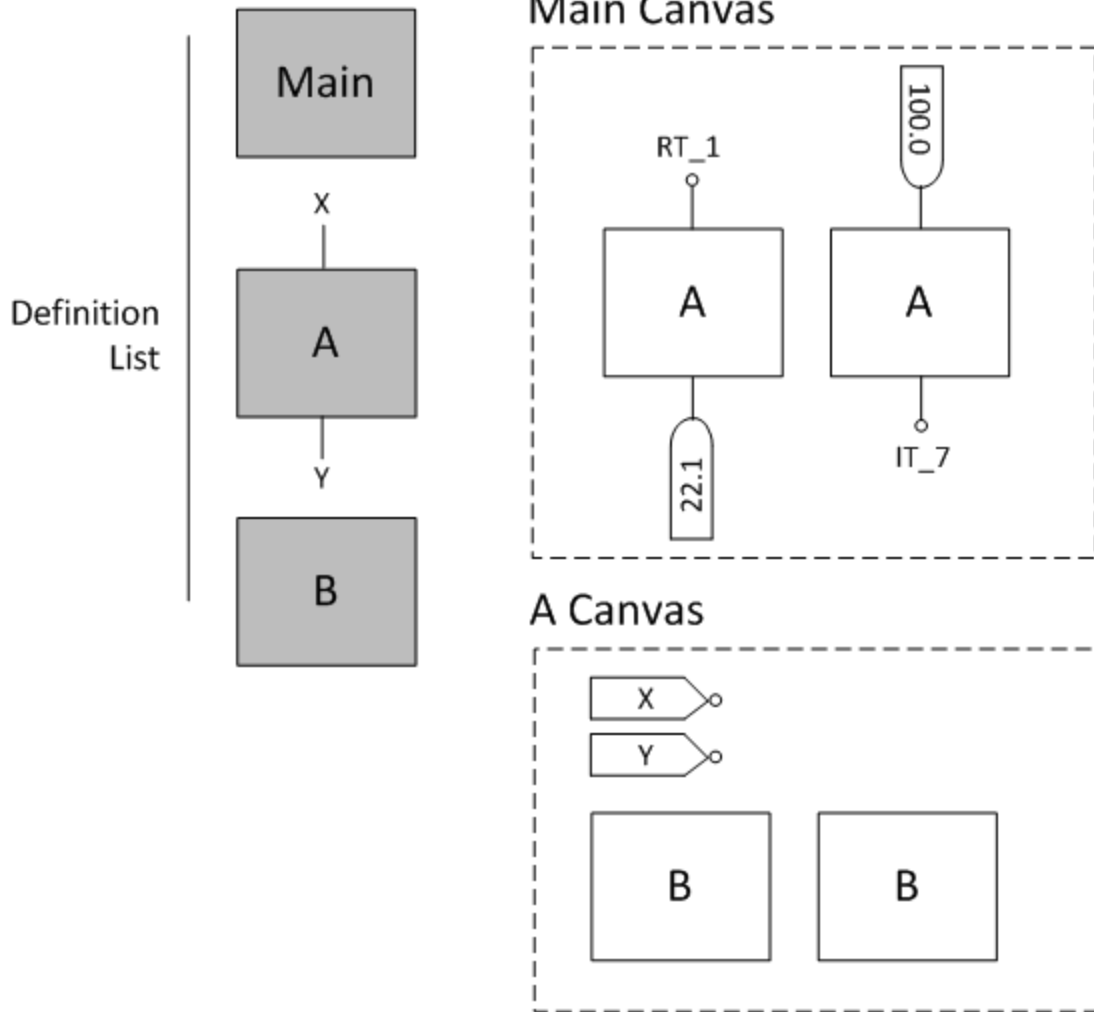
SUBROUTINE MAIN
_____
_____
_____
CALL A (RT_1,100.0)
CALL A (22.1,IT_7)
_____
_____
_____
END

```

Two Instances of A Called From the Main Canvas

Notice above that input arguments in each call statement for *A* can be literal numbers or signal values. Each argument defined in the SUBROUTINE statement is represented by an Import or Export component on the module canvas.





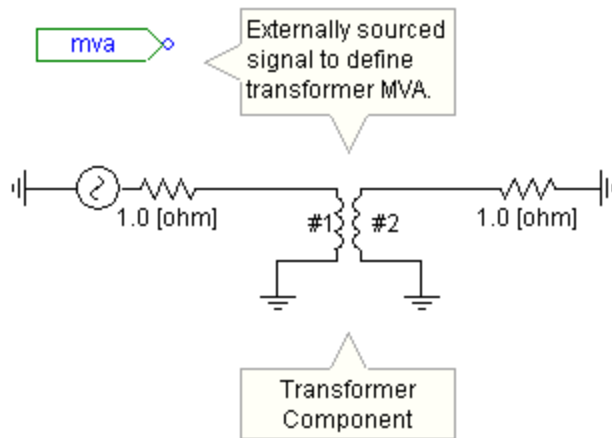
For more details on defining input parameters and port connections, see the chapter entitled Component Design in this manual.

## Important Design Considerations

Up until now, we have discussed multiple instance modules in a conceptual manner. In practice however, there are certain important things to consider when designing your system, so as to ensure maximum compile and simulation efficiency. It is essential to consider where signals are sourced. That is, whether a signal should be definition or instance-based. Also, Output Channel placement is a concern.

### Definition and Instance Variables

When designing and utilizing a system containing multiple instance modules, it is very important to study and determine how to deal with system parameters that may differ between instances. For example, if a module definition were to contain a transformer component and the MVA of this transformer is to vary from instance to instance of the module, then the source of the MVA value must be defined external to the module definition; otherwise all instances of the module will be forced to have the same transformer MVA value. Let's have a look at this example in PSCAD:



General	
Transformer MVA	mva
Base operation frequency	60.0
Leakage reactance	0.10
Max load losses	0.0

Module Definition Canvas with Transformer Component

Transformer Component Parameter Dialog

Here, a module definition contains a simple circuit with a transformer component. The transformer possesses an input parameter called *Transformer MVA*, which will accept a signal constant for its value. This signal is defined externally and imported as a signal called *mva*, which is entered directly as the transformer parameter. Although the import tag component, the transformer component and the signal itself are all part of the module definition, the *value* of the signal may be different for each module instance.

In this case, the signal *mva* is defined as an input parameter of the module itself. So if the module is instantiated many times, the *mva* parameter can possess a different value for each instance.

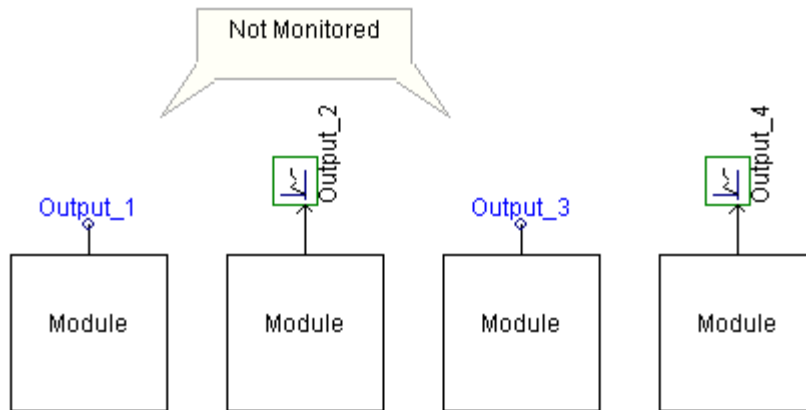
See The Parameters Section in the chapter called *Component Design* in this manual for more details on the proper design of module parameters, specifically the definitions of Literal, Constant and Variable type parameters, and how they pertain to multiple instance modules.

## Output Channel and Online Control Placement

Output channels and controls can be dangerous when using multiple instance modules. This is because an output channel component creates a signal for monitoring by EMTDC wherever it is placed, thereby reserving memory used and affecting simulation speed.

If an output channel is placed within a low-level module, say for example the *Valve Group* module discussed above, the volume of signals that single output channel creates can quickly balloon. Notice that when the *Valve Group* modules parent module (*6 Pulse Bridge*) is copied, the *Valve Group* module is called six additional times for each *6 Pulse Bridge* instance. This means that if the *6 Pulse Bridge* is copied multiple times, then the single output channel is multiplied six-fold.

It is best to avoid placing an output channel component at this level. A better option would be to export the signal to be monitored as far up the module hierarchy as possible. This way the signal is available at an upper level, where the user can decide to monitor it if the need arises. The image below illustrates this concept: Instead of attaching an Output Channel to the signal inside the module canvas, the signal is exported to the parent canvas, where it can be monitored as desired.



Multiple-Instanced Module Component with Signal Exported Through a Port Connection

## Runtime Configuration in EMTDC

In order to ensure full support of multiple-instance modules, EMTDC also had to go through some modifications. The major change to EMTDC involves the initialization of component parameters when they appear on a multiply-instanced module canvas. As a result of this, all relevant master library components have been modified to support this new structure. The same process must be applied to user-defined components, before they can fully support use on a multiple-instance canvas.

Please see the following sections in the PSCAD and EMTDC manuals for more details on this:

- **Changes to EMTDC Program Structure:** EMTDC Manual | Program Structure | The EMTDC Solution Process and System Dynamics.
- **New Input Parameter Constant Type:** PSCAD Manual | Component Design | Parameters Section | Input Fields | Input Field Data Types.
- **The #BEGIN Directive:** PSCAD Manual | Definition Script | Script Directives | #BEGIN/#ENDBEGIN.

## Copying and Transferring Module Hierarchies

There are two somewhat related methods available for copying and transferring entire module hierarchies. Of the two, Copy Transfer is the recommended and most convenient method.

### Copying Module Definitions with Dependents

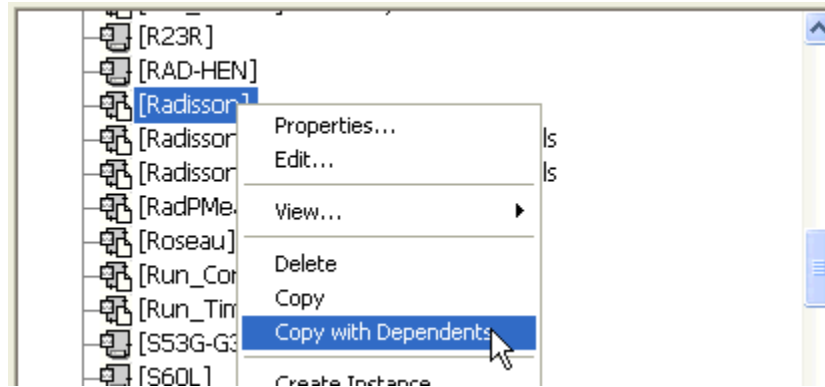
*Copy with Dependents* enables a module definition to be copied, including all of its dependent definitions. In other words, if a module contains other modules as part of its definition, and perhaps these modules have other modules inside, and so on, all module definitions will be included in the definition copy. When the definition is pasted, either in the same project or another project, copies of all the dependent definitions will also be included, along with all of the hierarchical linking information. When any of these pasted definitions are instantiated on the canvas, all child modules (i.e. modules defining its definition) will be instantiated as well, in the proper order.

This feature is provided mainly for inter-project transfer of modules, alleviating the need to manually reconstruct the module and its hierarchy. *Copy with Dependents* may also be used for copying and pasting within the same project, provided that no definitions exist in the target project that have the same name as any of the definitions being copied.

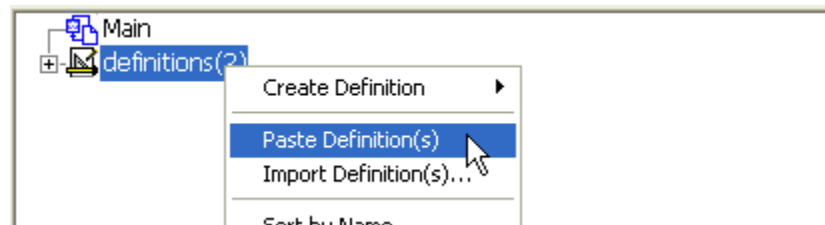
*Copy with Dependents* will also include custom component definitions that happen to reside on any of the dependent modules.

## Inter-Project

To copy a module definition with its dependent from one project to another: Select the source project in the workspace primary window and expand the definitions list. Right-click on the definition and select **Copy with Dependents** from the popup menu.



Next, select the destination project, right-click on the definition list and select **Paste**.



Lastly, instantiate the definition and paste it on the canvas (see Creating the First Instance of a Definition in this chapter).

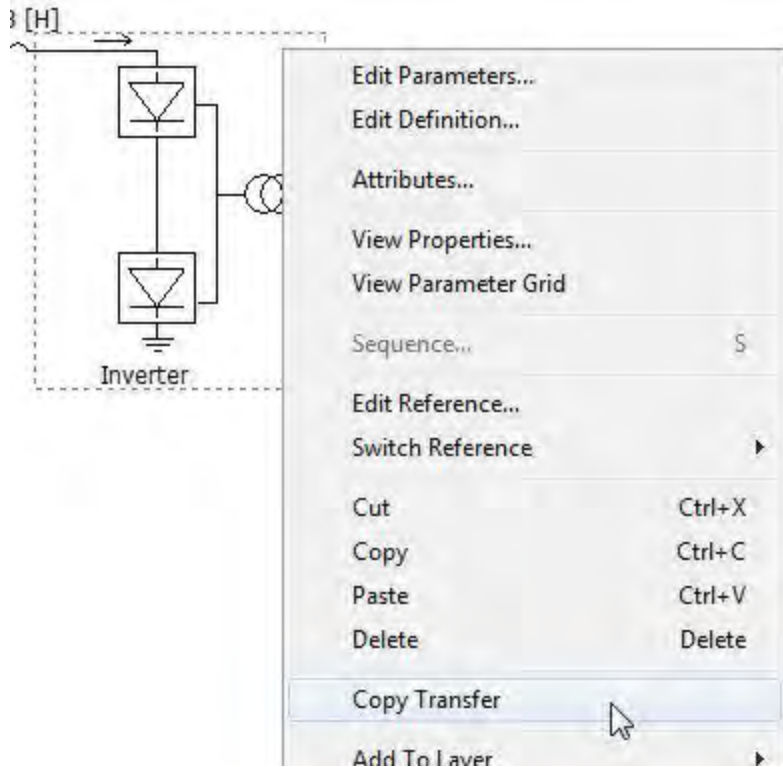
## Intra-Project

The same process as described above can also be used for copying and pasting a module definition within the same project, except the source and destination projects are the same.

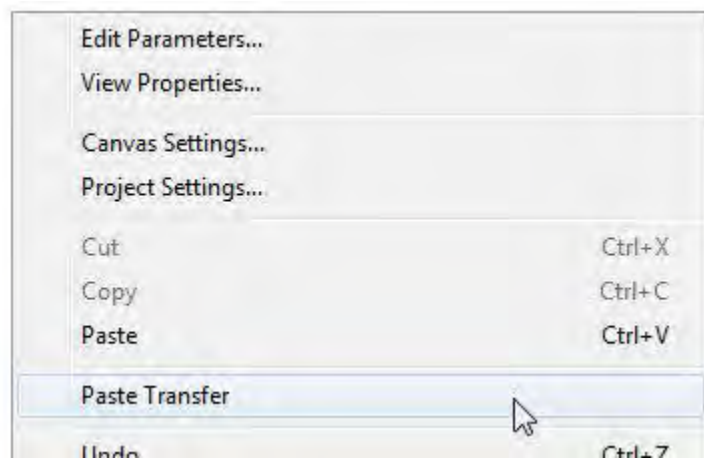
## Copy Transfer

This feature builds on the foundation provided by the Copy with Dependents feature. *Copy with Dependents* is a simple definition copy of an entire module hierarchy, including all dependent module definitions. *Copy Transfer* addresses the two major shortcomings of its predecessor: Instance information (i.e. parameter values) of the top-most module is included in the copy; and the module hierarchy is automatically re-linked together on paste, even if some or all of the definition need to be renamed. In addition to these, *Copy Transfer* also provides the convenience of copying directly from the component instance, as opposed to having to use the definition tree.

To copy transfer a module definition with its dependents, simply right-click on the top-level module instance and select **Copy Transfer**.



This action copies all module definitions that are part of this module hierarchy, along with the instance information (i.e. entered parameter data) of the top-level module. To paste the new hierarchy, right-click on a schematic canvas in any loaded project and select **Paste Transfer**.



**NOTE:** A module hierarchy that has been paste transferred is completely unique from and exclusive of the copied original.

## Parallel and High Performance Computing

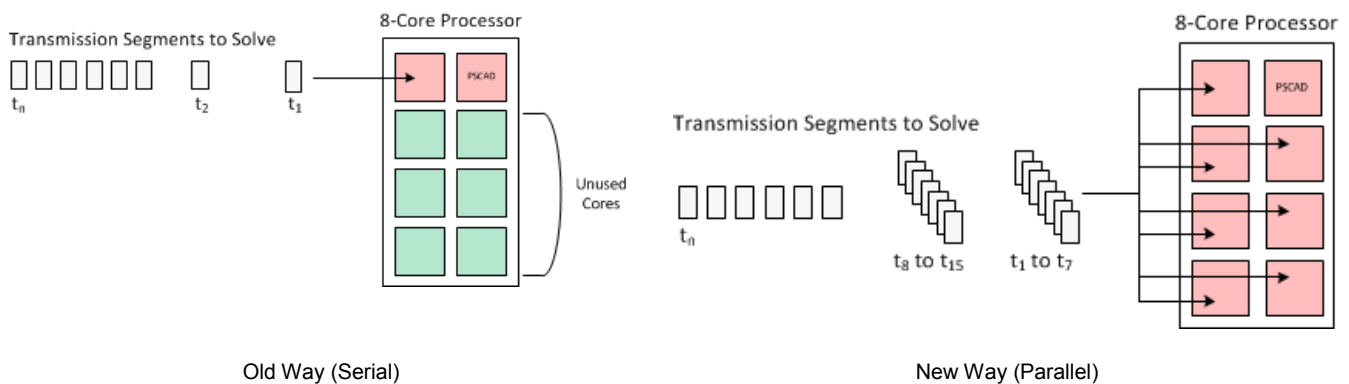
By the early to mid 2000s, the physical limitations of semi-conductor based, microelectronics had begun to alter the course of processor design. Clock speeds, which had increased by several orders of magnitude in the latter part of the 20th century, could no longer be significantly enhanced. As a result, manufacturers began to focus more on parallel computing via multi-core processors in order to increase computing power. Since then, eight, 16 and even higher-core processors have since become more prominent in newer personal computers.

Beginning in the late 2000s, requests for PSCAD and EMTDC to take advantage of parallel computing techniques began to rise. In response to this increasing feedback, the development direction began to move in part towards the exploitation of multiple-core processors; this in order to increase simulation efficiency and reduce the time needed to extract results. Versions prior to v4.5 (released in 2012) did not utilize more than two processor cores. PSCAD and EMTDC ran on separate cores, but there was always only one EMTDC process running at any given time.

With the release of v4.5, two unique parallel computing functions were incorporated that both utilize multiple processor cores: A parallel solution of transmission lines and cables, and a rudimentary ability to launch multiple EMTDC simulation runs simultaneously, where each EMTDC runtime process is based on a unique case project. In v4.6, the ability to launch multiple EMTDC processes from a single project was introduced.

## Transmission Lines and Cables

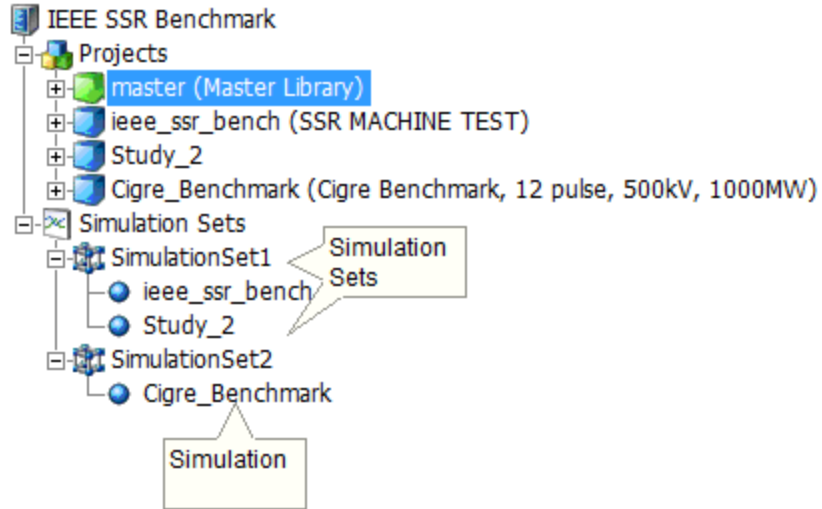
In older versions, transmission lines and cables were solved sequentially (i.e. one at a time). Some projects we have come across contain hundreds of transmission segments; the serial processing of which was cumbersome and time consuming. Given the fact that transmission lines and cables are solved independently of each other, and that there can be many of them in a single project, it makes perfect sense to exploit parallel computing techniques to solve them faster.



When a project is compiled, one of the final steps in the compilation process is to solve each and every transmission segment. Once this is complete, the EMTDC runtime executable can be built and the simulation launched. Transmission segments are now solved in parallel, based on the number of cores available on the local machine. For example, if the local processor has 8 cores, 7 of the 8 (one is being used by PSCAD) will be utilized until all segments are solved. Each segment will be assigned a single core directly: If the number of segments exceeds the number of cores available, then the segments will be solved in sequential sets until they all are solved. In the example above, note that only 7 segments are solved at a time, as there are only 7 cores available for processing.

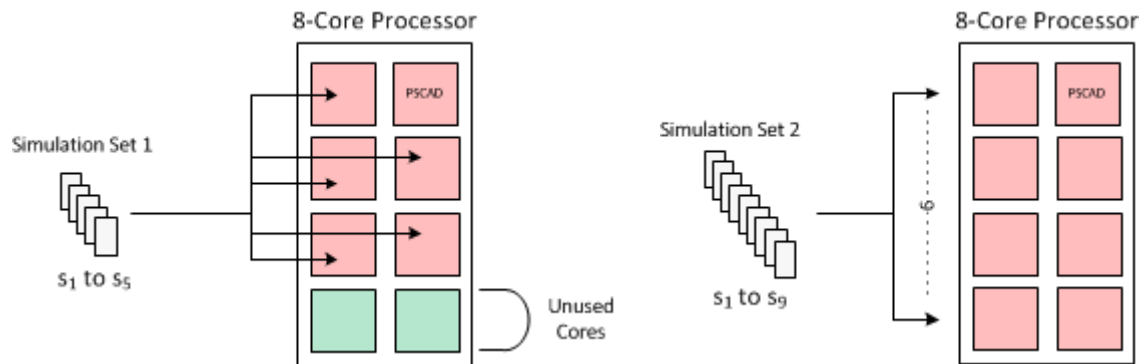
## Simulation Sets

Simulation sets are an inherent part of the workspace, and can be viewed and modified from within the workspace primary window.



The concept of a *simulation set* provides the foundation for parallel computing in PSCAD. A simulation set is a container of sorts, used to compartmentalize and configure groups of *simulations*. All simulations placed within a particular set are launched simultaneously (in parallel), utilizing all processing resources available. If multiple simulation sets have been defined, then each set is run sequentially as they appear in the list of sets: In the image above for example, *SimulationSet1* will launch and run the *ieee\_ssr\_bench* and *Study\_2* projects simultaneously. Once finished, *SimulationSet2* will launch and run the *Cigre\_Benchmark* project. The sequential launching of all sets is automatic, if the user selects to run all sets.

The image below illustrates a scenario on how resources are utilized when running simulations in sets. In this example, there are two sets, one containing 5 simulations, and the other 9. As the local processor is 8-core, then there are 7 cores available (one used by PSCAD) to run the processes. The second set contains more simulations than there are cores available (9 simulations, 7 cores). In these situations, the operating system will force all 9 processes to be shared amongst the available resources, resulting in decreased efficiency. It is important to note then to avoid exceeding the resources available to you when launching sets. Obviously, the more cores available for parallel processing, the larger the sets of simulations can be without decreasing efficiency.



Set Containing 5 Simulations Uses One Core For Each Directly

Set Containing 9 Simulations Must Share the 7 Cores Available (Decreased Efficiency)

Only projects loaded under the *Projects* branch in the workspace primary window may be added as a *Simulation* in a *Simulation Set*. Simulation sets and corresponding settings are all stored as part of the workspace.

For more information on manipulating simulation sets, see the following topics:

Adding a Simulation Set

Adding a Project to a Simulation Set

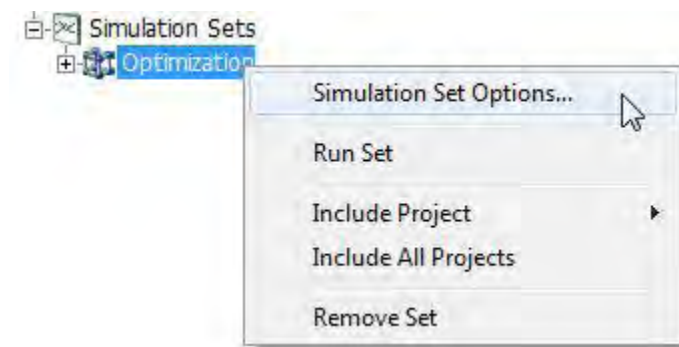
Running Simulations Sets

Pausing a Simulation Set

Stopping a Simulation Set

## Simulation Set Options

To invoke the *Simulation Set Options* dialog, right-click on an existing simulation set and select **Simulation Set Options...**



The following describes the options available for simulation sets:

### Simulation Set

- **Name:** The name of the simulation set. Name length must be 30 characters or less, and must only contain valid characters.

### Command Line

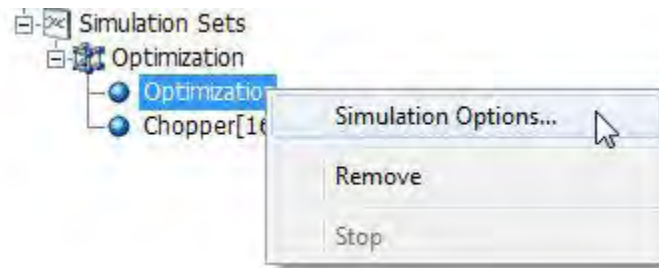
Pre and post-run processes may be performed between simulation set runs. For example, a batch file can be used to copy or move EMTDC output files to another folder before the next simulation set is started.

- **Post-Run Process:** Specify an executable (\*.exe) or batch (\*.bat) file to run, after the simulation set has completed.
- **Wait (Post-Run):** Select *Wait* or *Do Not Wait*. If wait is selected, PSCAD will wait for the *Post-Run Process* to complete before continuing with the simulation set run.
- **Pre-Run Process:** Specify an executable (\*.exe) or batch (\*.bat) file to run, before the simulation set has completed.
- **Wait (Pre-Run):** Select *Wait* or *Do Not Wait*. If wait is selected, PSCAD will wait for the *Pre-Run Process* to complete before continuing with the simulation set run.



## Simulation Options

To invoke the *Simulation Options* dialog, right-click on an existing simulation and select **Simulation Options...**



The following describes the options available for simulations:

### General

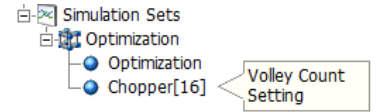
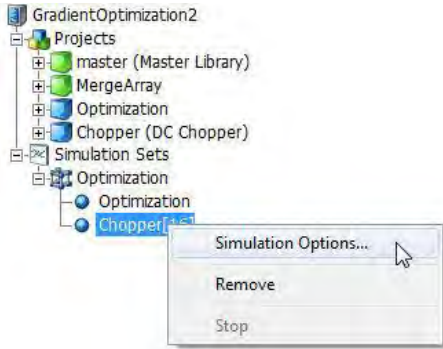
- **Namespace:** Display only. This is the associated project namespace name.
- **Volley Count:** Each project can launch multiple instances of itself for batch processing, each assigned a rank number used to isolate control behaviour. See *Volley Launch (SPMD)* for more details.
- **Trace Affinity:** Specify which project instance run (rank number) that will provide trace, or plotting information back to PSCAD. For example, if the **Volley Count** is set to 10, then there are 10 possible simulations that can be used as a 'tracer'. Select a single number (i.e. the rank number) to specify. See *Volley Launch (SPMD)* for more details.

## Volley Launch (SPMD)

In computing, the acronym SPMD (or 'spim-D') stands for Single Program, Multiple Data. It is a technique used to achieve parallelism, where multiple instances of a simulation are run simultaneously on multiple processor cores, given different input, in order to obtain results quicker and more efficiently than running them sequentially. In PSCAD, the SPMD concept is referred to more affectionately as a *Volley Launch*, analogous to the military tactic of having a line of soldiers fire all their weapons simultaneously. SPMD is also referred to as *Data Parallel Processing*.

Volley launch provides the ability to launch multiple simulation runs in parallel (up to a maximum of 64), based on a single case project. To set up a volley, a simulation must first be added to a simulation set. Once added, simply invoke the *Simulation Options* dialog and adjust the **Volley Count** option. For example, if you want to launch 7 simultaneous runs of a single project, then set the **Volley Count** to 7. When you next launch the simulation set, 7 instances of that simulation will be launched in parallel, utilizing all available processor cores.

In the example below, a project called *Chopper* is set to run as a volley of 16.



A Simulation Called Chopper Set to a Volley of 16

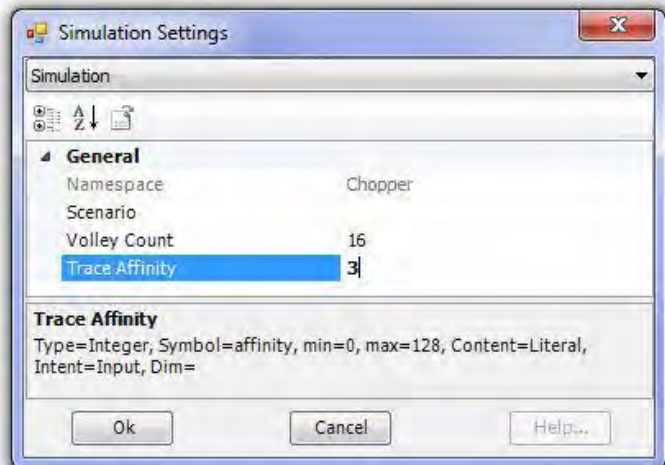
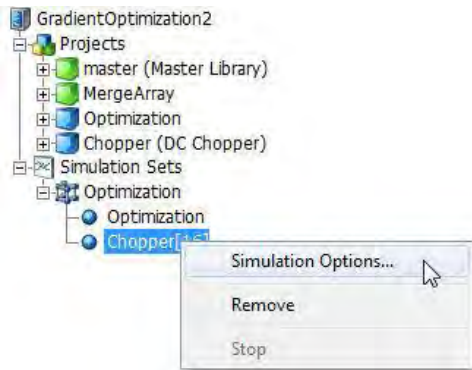
Simulation Options Dialog

Volley Count Displayed on Simulation

## Rank Number

The *Rank Number*, or simply *Rank*, is an identification number for a single simulation instance that is part of a volley launch. If there are 16 simulation instances in a volley for example, rank number 5 identifies the 5th simulation instance.

When a simulation is launched as a volley, only a single simulation can be selected to pass plotting data back to PSCAD for display. The rank number is used to identify the simulation instance to be used as the *tracer*. *Tracer* is also a military term referring to tracer bullets used in automatic weapons — projectiles that are visible to the naked eye. The tracer simulation is set via the Simulation Options dialog, using the **Trace Affinity** field.



A Simulation Called Chopper Set to a Volley of 16

Simulation Options Dialog

In the example above, the **Trace Affinity** is set to 3, meaning that the 3rd simulation instance in the volley will return the plotting waveforms. The purpose of this is to provide user feedback during development and debugging of the simulation. Once this part is complete, the affinity can be set to 0. Zero indicates that no simulation will return waveforms, resulting in better performance.

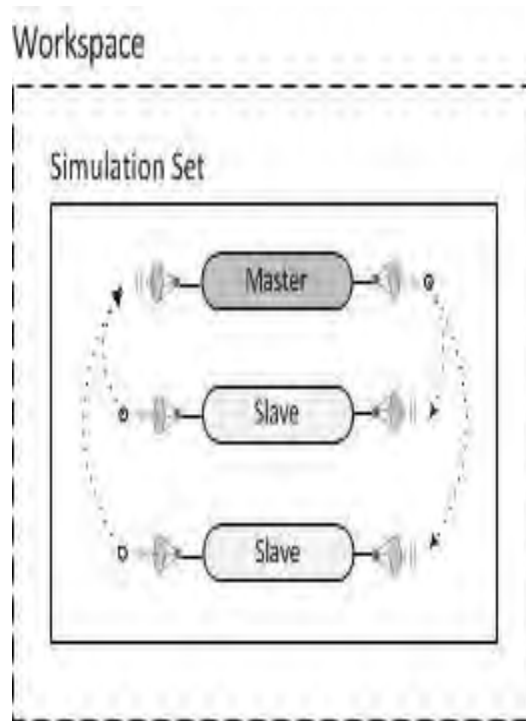
The rank number is also used to ensure unique input data for each simulation instance in the volley. Rank number can be used in combination with the Rank Number component in the master library.



## Root Control Interface (RCI)

The *Root Control Interface (RCI)* was first released as part of the v4.5 minor upgrade. Root control allows for one root, or *master* project, to control multiple *slave* projects, where both *master* and *slaves* must be part of the same Simulation Set. The idea behind the development of the RCI was to support both parameter sweep, as well as optimization-based, multiple-run studies.

Like the simulation sets, the RCI is an inherent part of the workspace, which enables inter-project communication within a single simulation set. This is accomplished using the already well defined Radio Link transmitter and receiver components, which were extended in v4.5 to include a provision field for a foreign namespace (that is, another project within the workspace). This instructs the link to collect its value from a foreign source, and thus allows for a more sophisticated means of multiple run control.



A Simulation Set with Three Projects: One Master and Two Slaves

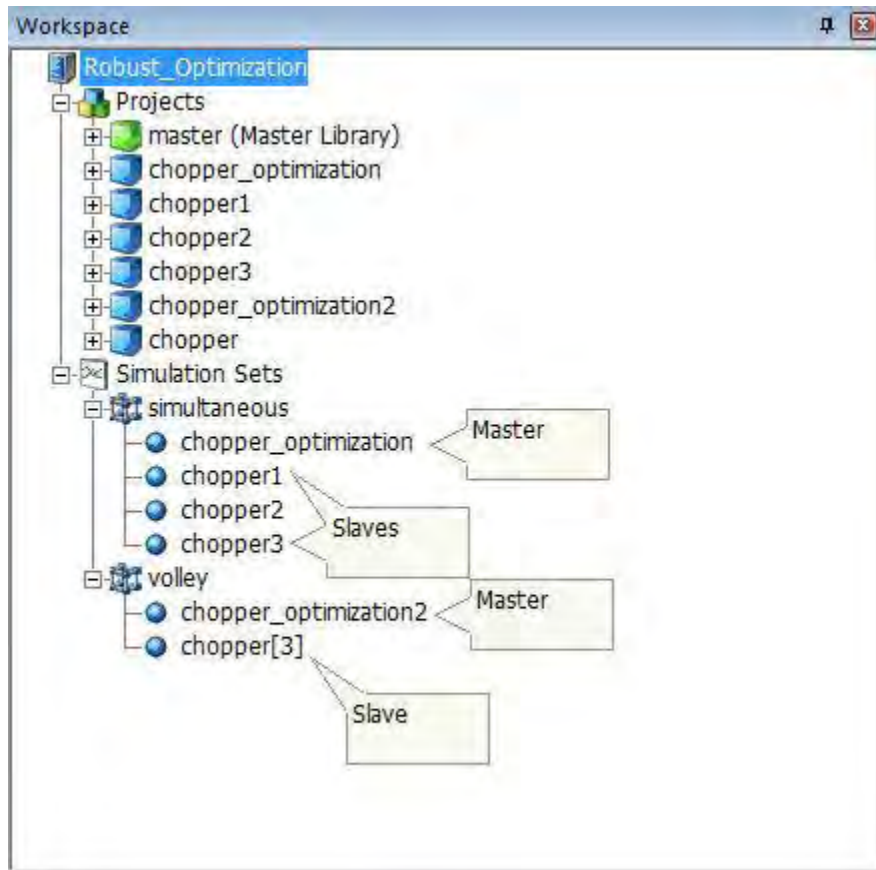
The image above illustrates an example simulation set containing three projects: One configured as a *master* and two configured as *slaves*. Each slave project communicates with the master via radio link transmitters and receivers. The master also communicates with the slaves in the same way. Communication between projects is performed only between runs; that is, following the end of one run and before the start of the next. In this way, the master project distributes the control parameters to the slaves, and the slaves send result data back to the master (via radio links). The master uses the results data from the slaves to generate input data before the next run starts.

To configure a project as master or slave, change the **Run Configuration** field under the Runtime tab in the project settings.

**NOTE:** Radio links possess an additional facility for the purpose of plotting its value for each run, within the master project.

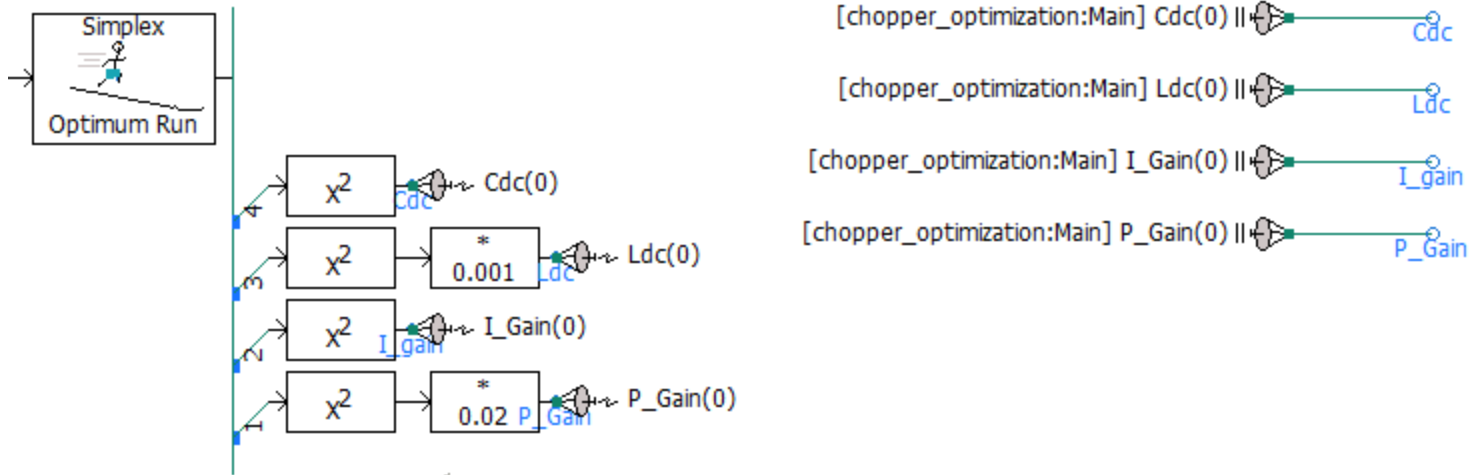
## EXAMPLE A (Non-Volley):

Consider the *Robust Optimization* example workspace in the PSCAD examples folder under ...\Examples\Root Control (start PSCAD and load this workspace if desired to better follow along with this example explanation). This workspace is composed of four projects, one configured as a master (*chopper\_optimization*) and the other three as slaves.



The *Robust Optimization* Workspace

In this example, we focus on the *simultaneous* simulation set (non-volley), where three slave projects containing a chopper circuit are run in parallel for each multiple run. Before the start of each run, the master project (called *chopper\_optimization*), which contains the Optimum Run component, provides each slave project with unique, initial data. The data provided to the slave projects is the DC parallel capacitor value  $C_{dc}$ , the DC series inductor value  $L_{dc}$ , and the proportional and integral gain values ( $I_{gain}$  and  $P_{gain}$ ) for the PI controller component. All four of these values are transmitted from the master to each slave project using radio links.

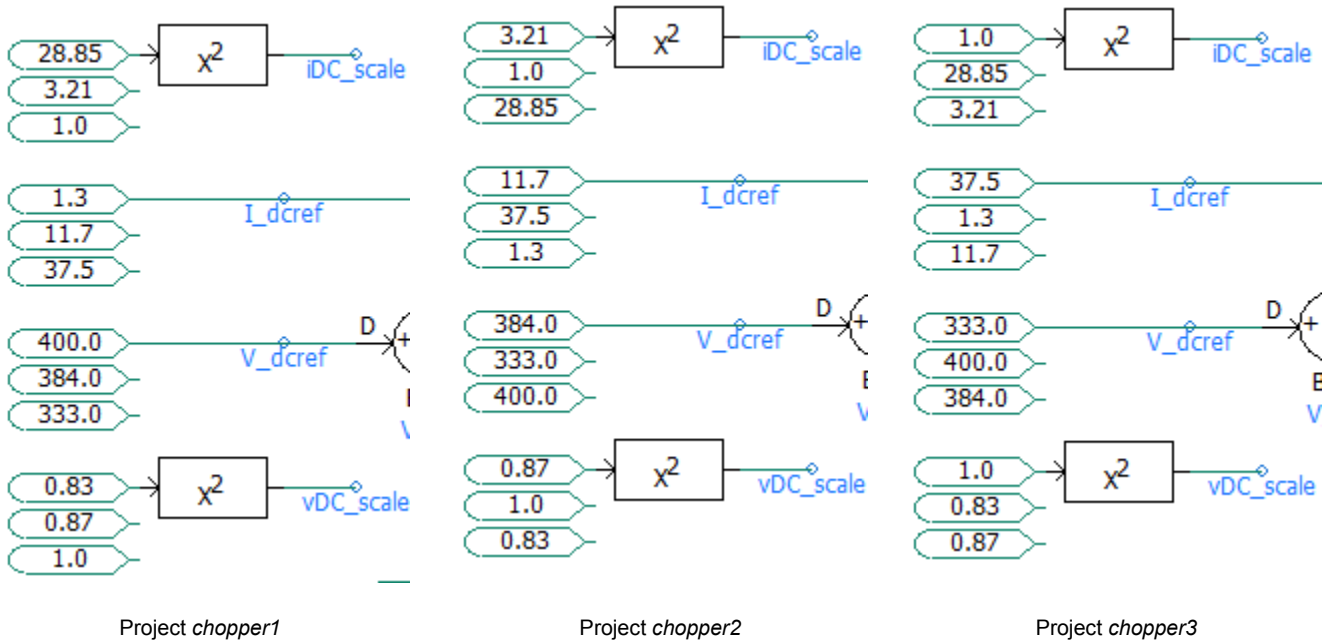


Data Transmitted from the Master Project

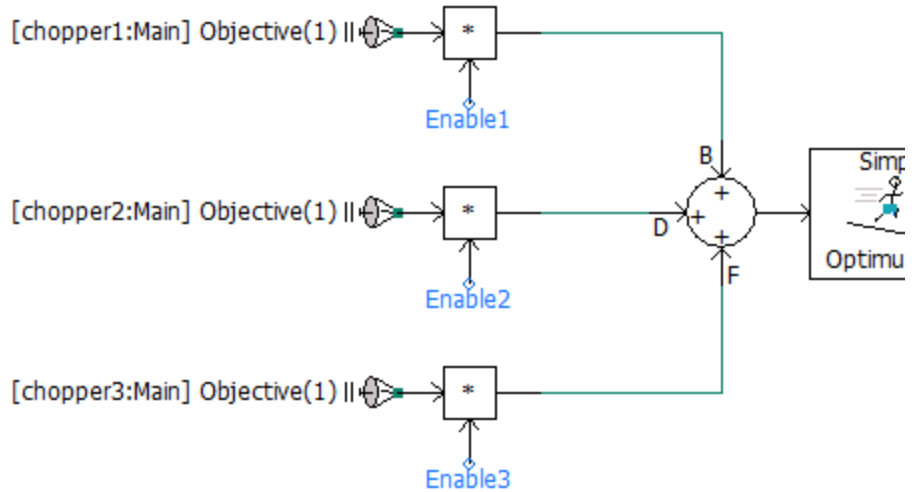
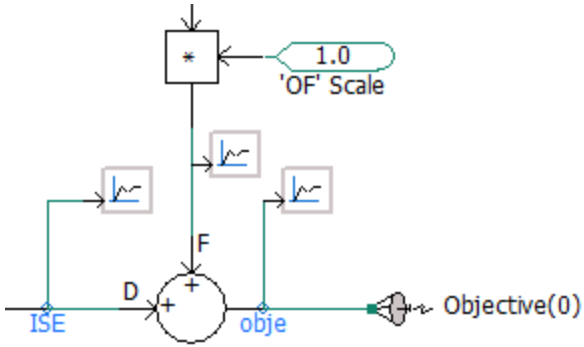
Data Received by the Slave Projects

**NOTE:** The rank number specified in each of the radio links is 0. This signifies that the signal is being transmitted to all simulation instances. In this example, there is only one simulation instance for each project.

Each chopper project is unique, in that the input constant tags in each control circuit are slightly different.



The differing inputs to the control circuit result in a unique output quantity from each slave. A single output signal, being the objective function (*Objective*) is transmitted back to the master via radio links.

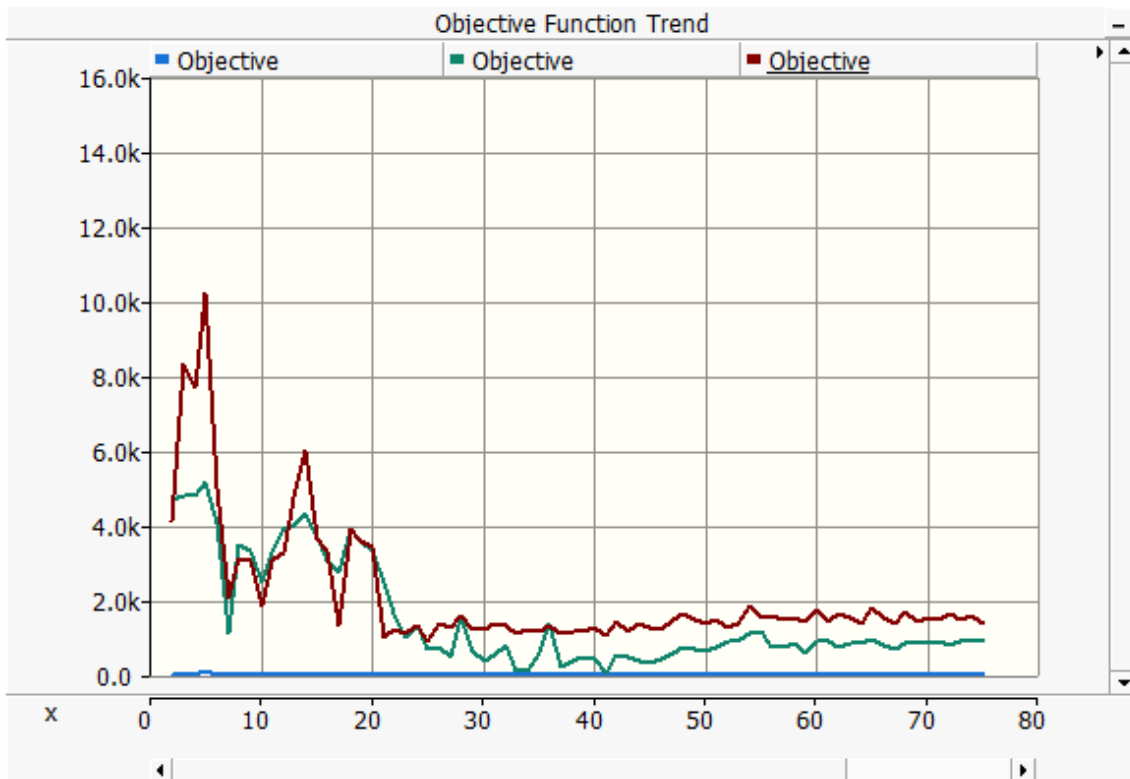


Data Transmitted from the Slave Projects

Data Received by the Master Project

**NOTE:** The rank number specified in each of the radio links in the master is 1. This is because this is a non-volley simulation and so there is only one unique instance of each simulation.

The master project in this example contains a plot of the three, unique objective functions coming back from each slave case, where the x-axis is the multiple run number. See Running Simulations Sets for more.



Objective Function Trend Graph (Unique Objective Signal from Each Slave Project)

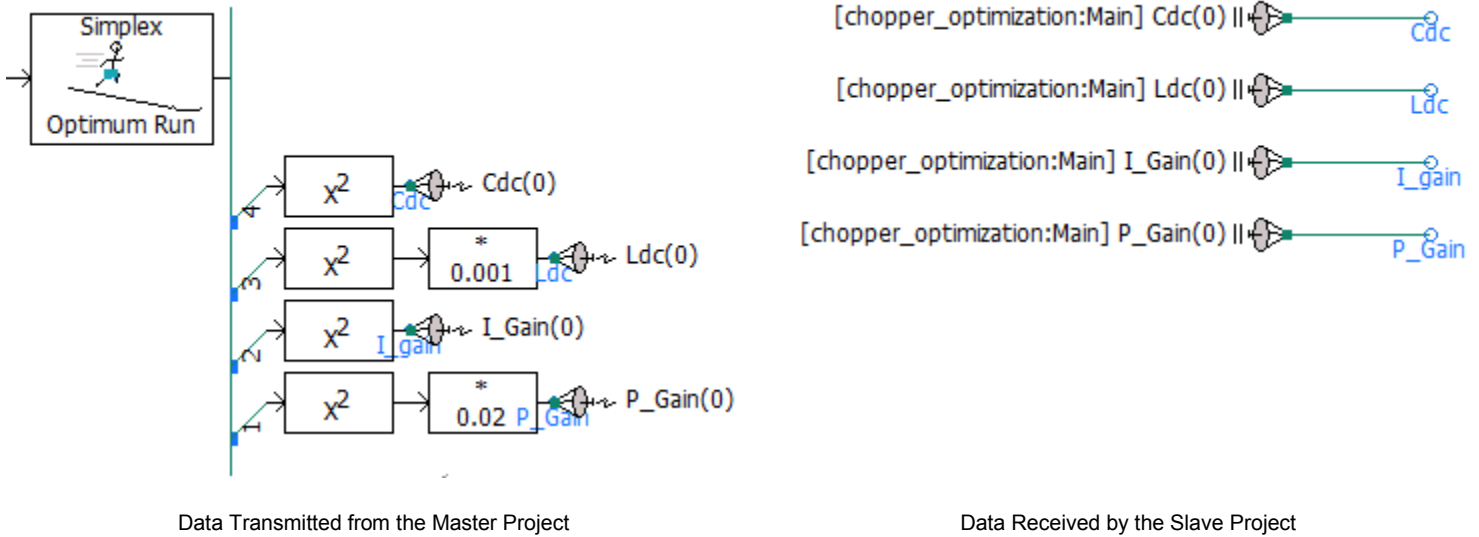
## Root Control with Volley Launch

Root control can also be used in combination with volley launch. Making use of volley launch will reduce the number of slave projects you need to maintain, down to just one. The volley rank number is then utilized to provide differentiation between simulation instances.

### EXAMPLE B (Volley):

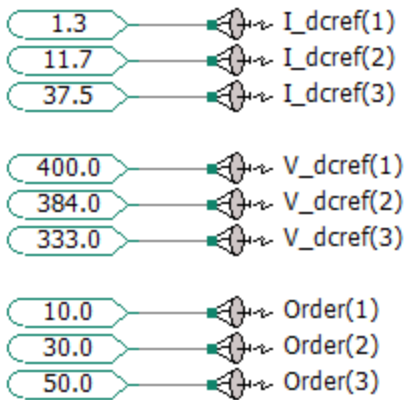
Consider again the *Robust Optimization* example workspace described above.

In this example, we focus on the *volley* simulation set, where a single slave project containing a chopper circuit is launched as a volley of three for each multiple run. As in EXAMPLE A, before the start of each run the master project (called *chopper\_optimization2*), which contains the Optimum Run component, provides the slave project initial data. The data provided is the DC parallel capacitor value *Cdc*, the DC series inductor value *Ldc*, and the proportional and integral gain values (*I\_gain* and *P\_Gain*) for the PI controller component. All four of these values are transmitted from the master to the slave project using radio links.

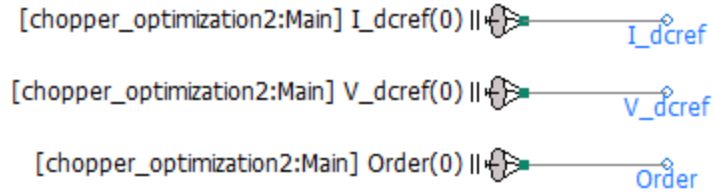


**NOTE:** The rank number specified in each of the radio links is 0. This signifies that the same signal is being transmitted to all simulation instances in the volley. In this case, all three simulation instances will receive the same data values from the master.

The *chopper* slave project is being launched as a volley and so all simulation instances will be identical. As such, we must rely on the simulation rank number in order to provide each simulation instance with unique values. The input constant tags that were made unique in each slave project in EXAMPLE A, have been moved to the master project in this example, and then the data transmitted to each simulation instance via the rank number.



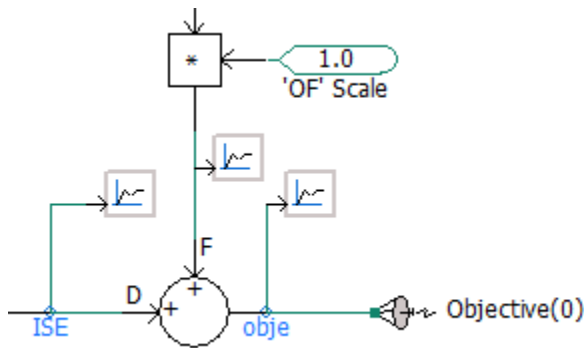
Data Transmitted from the Master Project



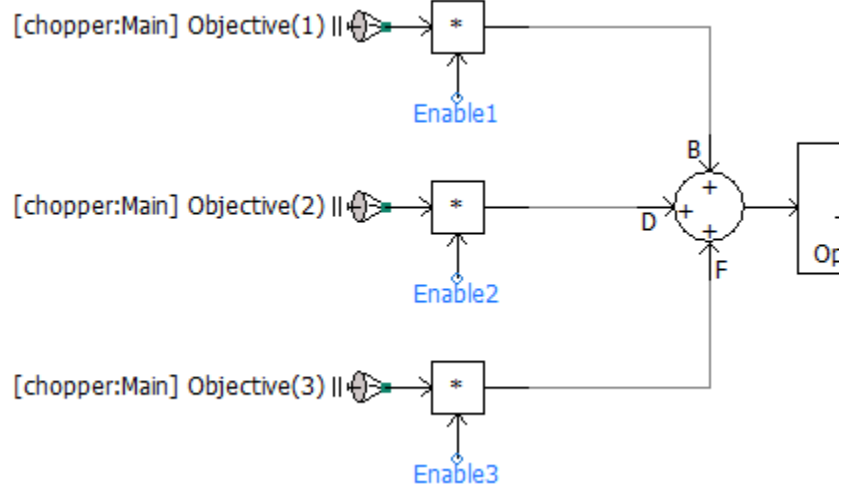
Data Received by the Slave Project

**NOTE:** A unique set of three values needed by each simulation instance are transmitted from the master, given a specific rank number. A rank number of 0 is specified in the slave to indicate that it should only receive transmitted data matching it's rank number.

A single output signal, being the objective function (*Objective*) is transmitted back to the master via a radio link.



Data Transmitted from the Slave Projects



Data Received by the Master Project

**NOTE:** The rank number is specified in each of the radio links in the master.

## IncrediBuild-XGE Grid Computing Engine

In terms of parallel computing, the user is limited only by his or her computing resources. So far, discussions on parallel computing in PSCAD and EMTDC have focused on the availability of cores on the local processor. Local processors are fine for small sets of parallel simulations, but can impose serious limitations in larger ones. Enter the *IncrediBuild-XGE*: This third-party tool overcomes reliance on just the local processor, by interfacing with and controlling other processors on the local network, using PSCAD as supervisory control software.



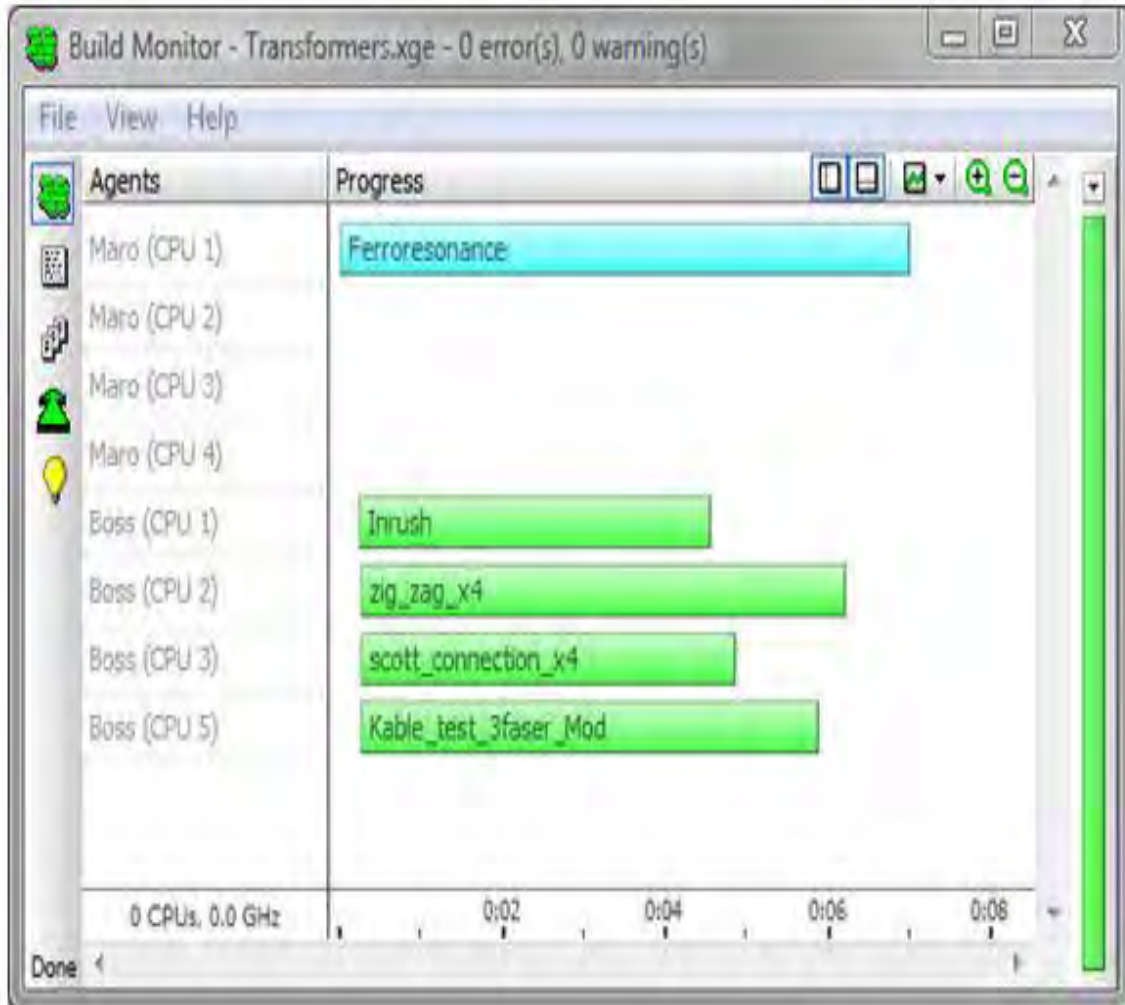
First available with PSCAD v4.6, the *IncrediBuild-XGE* (by IncrediBuild), effectively combines a designated collection of processors available on the local area network (LAN) to act as a single processor with a much greater number of cores. This greatly increases the number of parallel EMTDC processes that can be launched from PSCAD, drastically reducing the time required to perform highly parallel simulations. The *IncrediBuild-XGE* is very easy to setup, involving the installation of an agent utility on each machine whose processor is to be designated for simulation. An additional *coordinator* program is installed on single machine to coordinate and manage the processes spawned.



PSCAD Using the *IncrediBuild-XGE* for High Performance Computing

A server farm can thence be constructed, comprised of many multi-core processors that are dedicated to running simulations. In fact, most organizations will have enough idle processors on their LAN to create a virtual supercomputer!

All process commands originate from a single PSCAD machine (which must have an *agent* utility installed) and are sent to the *coordinator* machine, along with an XML file containing a task list. PSCAD itself runs normally, with all process control being managed by the *coordinator*. When using the *IncrediBuild-XGE*, a monitor window provides feedback on how the processes are being remotely deployed and coordinated.



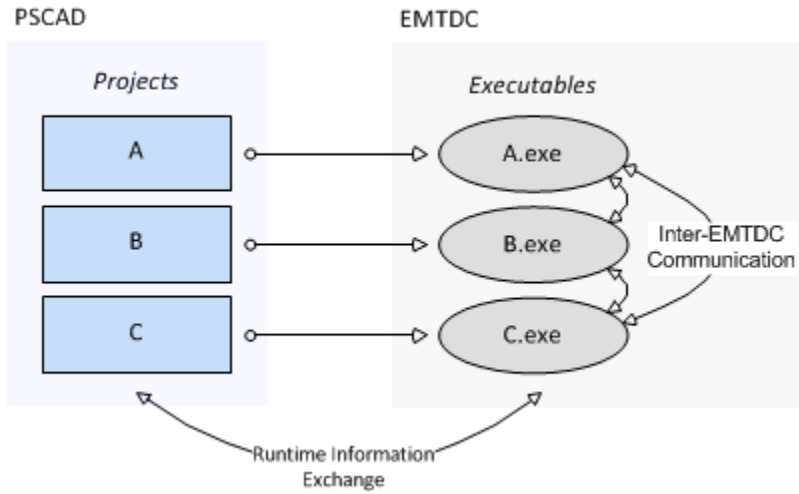
IncrediBuild-XGE Process Monitoring Window

## Procurement and Setup

If you are interested in setting up the *IncrediBuild-XGE* in your local environment, please contact our PSCAD Sales Desk for more information.

## Electric Network Interface (ENI)

Beginning with PSCAD v4.6, multiple case projects representing multiple parts of a complete electric network, may be run simultaneously as a single simulation. A single electric network may be split so that each electric subsystem is represented by a separate case project, and thereby run using a separate EMTDC process. Each EMTDC process is linked together via an *Electric Network Interface (ENI)* to form a cohesive simulation that is run from within a single workspace.



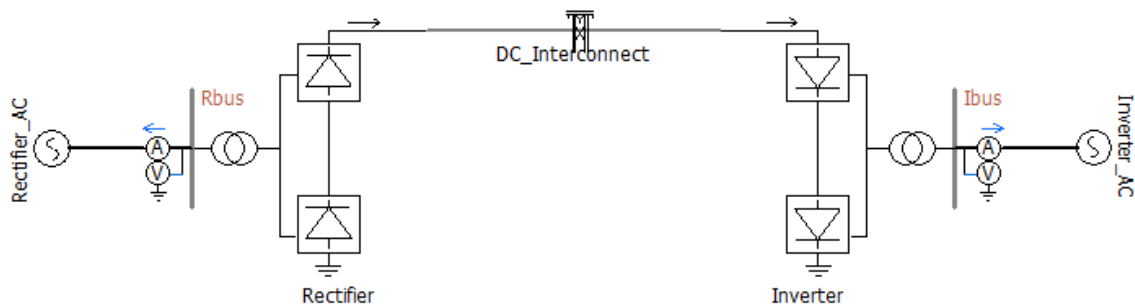
Multi-Project Simulation Using the ENI in PSCAD

In terms of runtime information for plotting and online data signal control, PSCAD and EMTDC communicate normally as they would for a single process. There is additional communication between the multiple EMTDC processes in multi-project simulations however. Due to the fact that each EMTDC process is an executable file, each can be launched on its own processor core, enabling parallel simulation of the complete system.

For larger networks containing multiple subsystems, using the ENI can greatly enhance the total simulation speed, especially when the subsystems contain high speed switching devices (i.e. FACTS devices).

## Setting Up the ENI

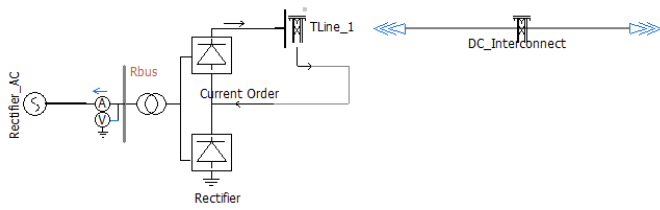
The boundaries of electric subsystems in EMTDC are defined via distributed-parameter transmission lines and cables, and so it makes sense that the transmission segments themselves be the linking points in a multi-project simulation. As a very simple example, let us take a slightly modified version of the familiar *Cigre Benchmark Model* example project. A transmission line has been added to represent the DC interconnection between the two converters (the original was simply an RLC passive network).



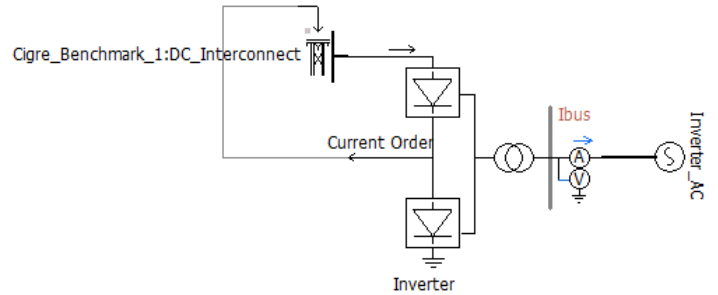
Cigre Benchmark Model Example Project with Transmission Line Interconnect

The addition of the transmission line to this system results in a division of the electric network into two, separate subsystems (approximately 25 nodes each). These two sub-networks can be compiled and launched as separate, but dependent processes, due to the natural propagation delay generated by the distributed transmission line. Before this can be accomplished however, the project must be manually split in to two separate project files.

Splitting the project up can be achieved in a number of ways, but the most efficient is to first duplicate the project and then manually modify each one. Doing this for this example results in the following two separate project circuits:



Rectifier Side Project

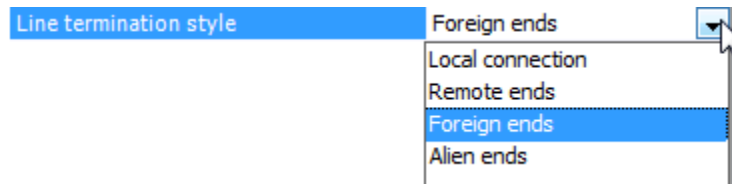


Inverter Side Project

In order to accommodate this new electric network interface between projects, some changes have been made to the transmission line components. These changes are explained in the following sections.

## Transmission Line and Cable Parameters

A few parameters and functionalities exist in both the Overhead Line and Cable configuration, as well as the Overhead Line and Cable interface components in order to accommodate the ENI interface. In the configuration components, two new termination styles have been provided.



Of the four options provided, only **Foreign** and **Alien ends** are used when setting up the ENI interface:

- **Foreign ends:** Foreign ends are used to interface between two separate case projects within the same workspace (i.e. instance of PSCAD).
- **Alien ends:** Alien ends are used to interface between two separate case projects within two separate instances of PSCAD.

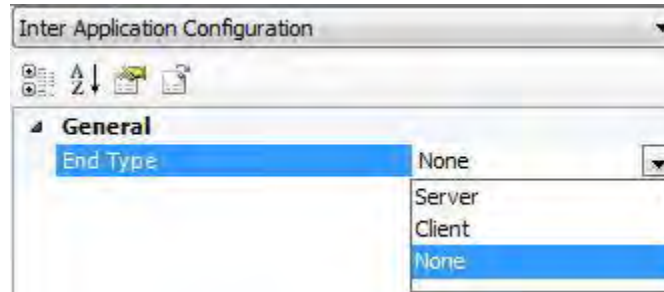
The interface components contain a few parameters involving the ENI:

<b>Non-electrical Signal Transfer</b>	
Sending Signal Dimension	0
Receiving Signal Dimension	1

These parameters provide information for the Control Signal Carrier function (described below).

- **Sending/Receiving Signal Dimension:** Enter the dimension of both incoming control or outgoing control signal. See Control Signal Carrier for more details.

There is also a parameter provided for using the ENI between separate PSCAD applications:

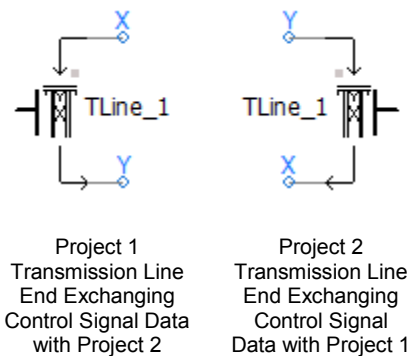


- **End Type:** This parameter specifies the end type of this side of the transmission line: **None**, **Client** or **Server**. When simulating using two separate PSCAD applications, one instance must act as the server, and the other the client: The case project containing the Overhead Line Configuration component is always designated as the server.

## Control Signal Carrier

Inevitably when a project is decomposed into separate electrical subsystems, there will be situations where control signals generated in one project are needed in another. The ENI utilizes a concept referred to *Control Signal Carrier*, which allows control signals to be piped through the transmission line interface, along with the electrical data. Of course, the control signal data arrives at the other end with no time delay.

In the example given above, the *Current Order* control signal is generated in the inverter case project, and then passed via control signal carrier to the rectifier project.



Non-electrical Signal Transfer	
Sending Signal Dimension	1
Receiving Signal Dimension	1

Transmission Line Interface Component Parameters

## Module Comparison Tool

The *Module Comparison Tool* provides the ability to compare two module definitions and identify differences through graphical feedback.

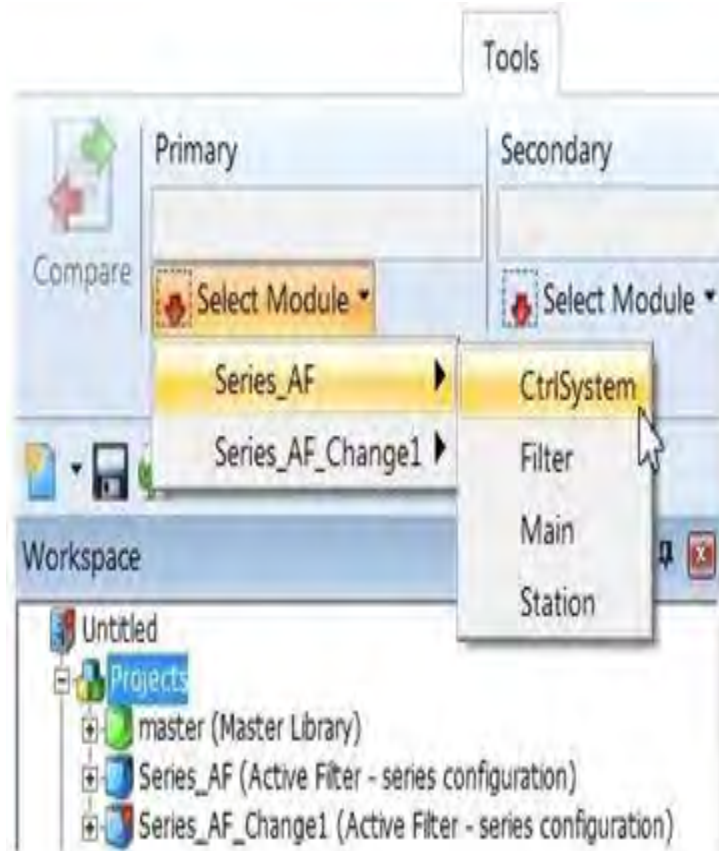
The comparison is performed directly on the XML data representing each module definition, and so the differences detected encompass just about everything from parameters, script and graphics of components on the Schematic canvas, to Canvas Settings and other properties. The tool will also detect whether or not components exist in both the primary and secondary module definition being compared.

**NOTE:** The comparison tool does not identify differences in the module definition graphic or module parameters sections.

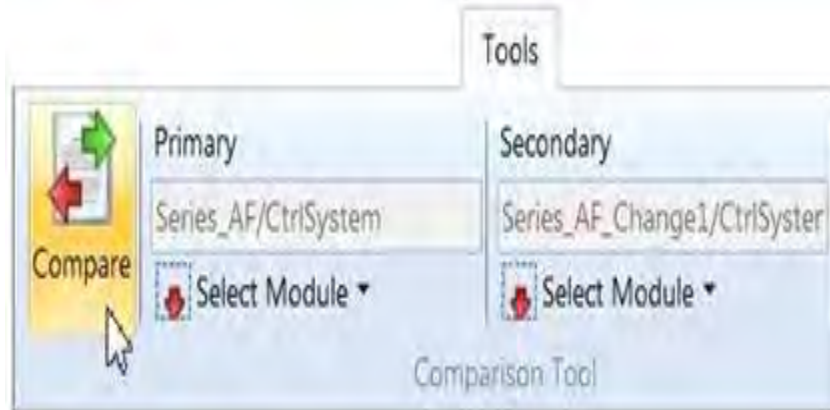
## Initiating a Comparison

The controls for the module comparison tool may be found in the *Tools* tab of the ribbon control bar. At least one project must be loaded in the workspace before proceeding. To compare two module definitions:

- Select the primary and secondary module definitions from the drop lists provided. The drop lists will include access to all module definitions in all projects currently loaded in the workspace. For example, the *CtrlSystem* module in the project *Series\_AF* and the *CtrlSystem* module in the project *Series\_AF\_Change1* are selected for comparison below.

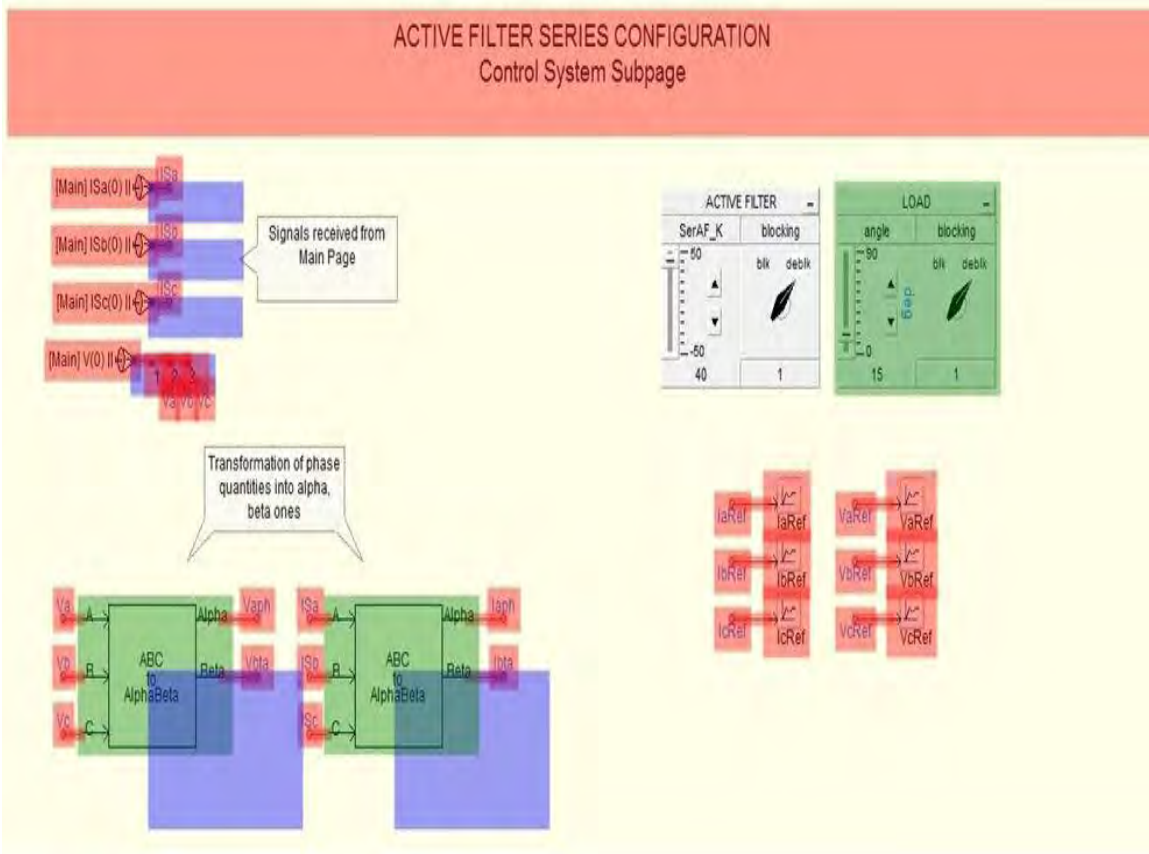


- Select the **Compare** button to perform the comparison.



## Reviewing the Differences

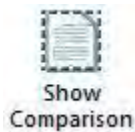
When differences are detected, they are displayed in both the *Schematic* view and in the *Comparison Tool Results* pane. In the *Schematic* view, color-coded shading is applied to areas where differences are detected. For example:



If the mouse pointer is positioned over one of the graphical results on the transparency, a popup dialog will appear, listing the description of the difference.



The comparison results are displayed on a transparency, overlaid atop the actual *Schematic* canvas. While the transparency is enabled, no changes may be made to the canvas. The comparison transparency may be toggled on and off by clicking the **Show/Hide Comparison** button in the ribbon control bar (this button may be found in either the *Tools* or the *View* tab).



Differences may also be reviewed in table format via the *Comparison Tool Results* pane. Each row in the table represents a unique difference between the two module definitions, including a description of the difference, as well as the actual values in each module as applicable.

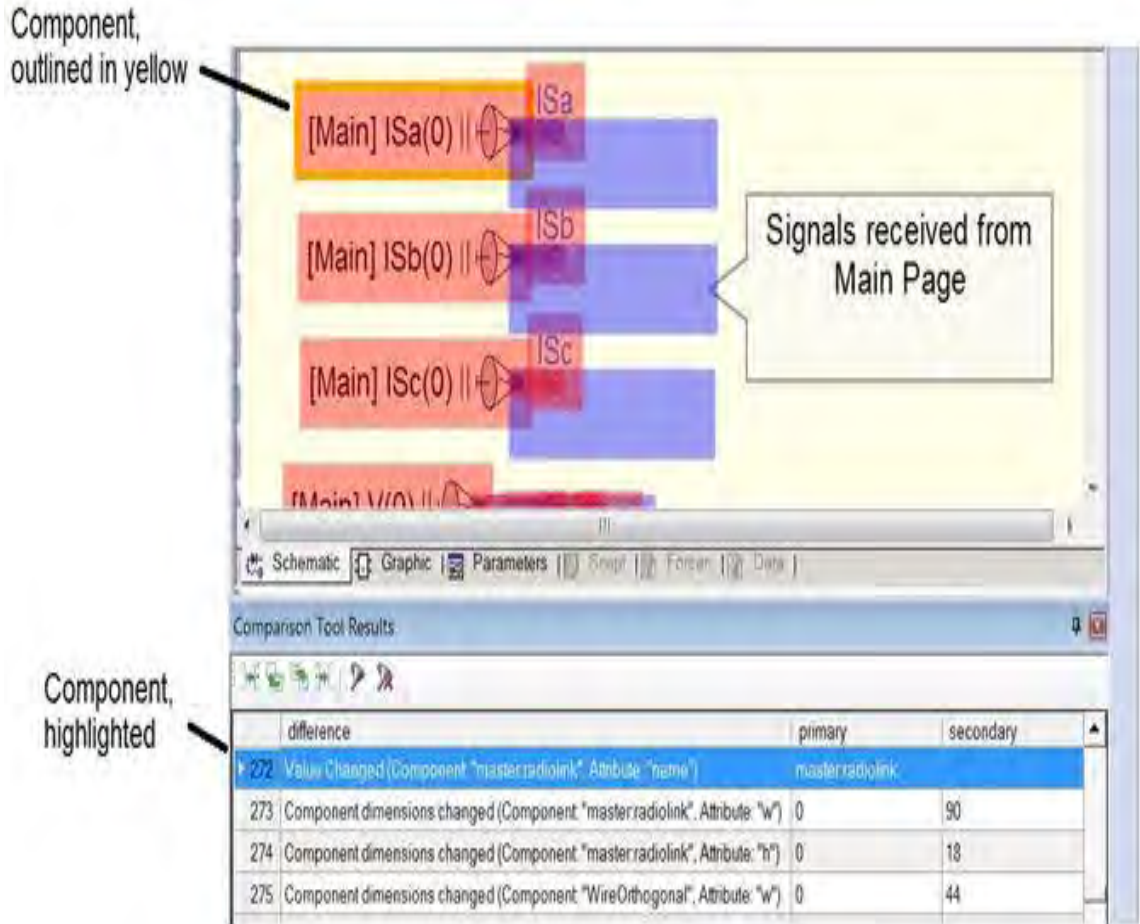
difference	primary	secondary
value changed (Component: "Canvas", Attribute: "show_grid")	1	0
Component master:consti is missing from primary schematic		master:consti

Note that like the other panes, the *Comparison Tool Results* may be hidden, tabbed or docked in the PSCAD environment, or undocked outside of PSCAD as a separate floating pane.

## Navigating Differences

The results shown in the *Schematic* view are synchronized with those given in the *Comparison Tool Results* pane. Simply click a row in the results table to highlight the result in the *Schematic* view. Alternatively, select a difference in *Schematic* view to highlight the corresponding row in the *Comparison Tool Results* pane.

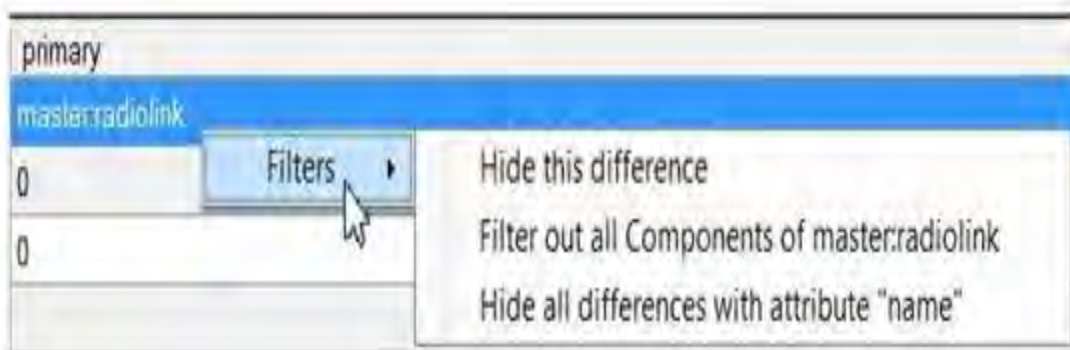




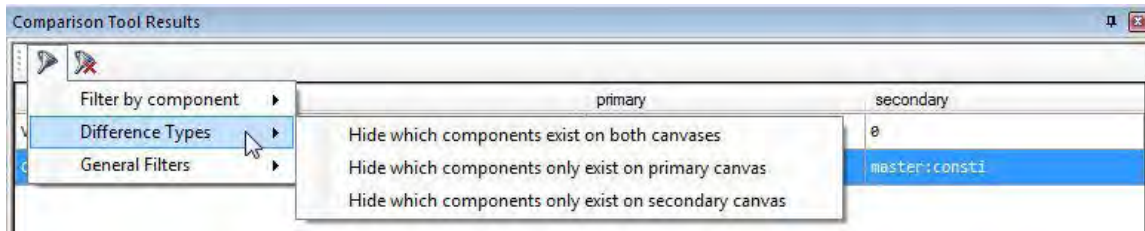
## Filtering the Results

The data in the *Comparison Tool Results* pane may be filtered in a number of different ways in order to keep the information specific to what you are looking for. To filter the results:

- Right-click on a specific row in the results pane and select a filter:



- Select a filter from the **Filter** button:



## Blackboxing Modules

With a simple click of the mouse, this feature will convert any page module into an equivalent, non-module component, complete with generated source files and optional compiled binary files. Referred to as 'blackboxing', it allows control systems to be designed and maintained graphically, only to be quickly collapsed and compiled at will; thereby protecting intellectual property when models are distributed to clients.

### Background

In essence, PSCAD is (and always has been) a source code generator for the EMTDC simulation program. Both modules and standard (non-module) components are combined to form the source defining a structured, sequential program, which includes a unique set of subroutines to represent each module in the project. All components may possess a combination of parameters and ports. When module components are coded in Fortran, these ports and parameters are used to define the arguments for the corresponding subroutine. Non-module components combine to represent the body of the subroutine.

Beginning in and around 2011, requests for instructions on how to convert module components to equivalent, independent external source have noticeably increased. This trend involves using PSCAD module hierarchies to generate Fortran source (ex. a device controller). The generated Fortran is then manually manipulated to remove the internally dependent code required by PSCAD and EMTDC, thereby creating external source that may be called from within a standard component. This effectively conceals the graphical design of the circuit, hiding the contents within the external source. The source is normally compiled into an object (\*.obj) file or static library (\*.lib) before it is supplied to the client, providing protection for the provider's intellectual property. The custom component may also be further customized to include a *Branch* segment to represent the electric network, plus ports and other required attributes – the end result being a fully 'blackboxed' version of the source module component.

### The Algorithm

Blackbox performs all the necessary steps required to convert a page module in PSCAD into a user-defined component, including compiled source code. The following functions are executed automatically:

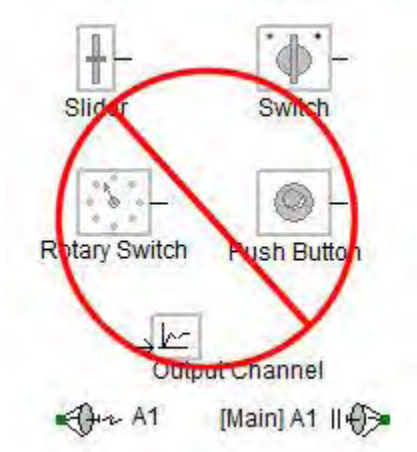
- **Fortran Source Generation:** PSCAD already generates Fortran source, however this code is written specifically to interact with EMTDC as part of the simulation project, and is not formatted to be used as independent, external source. Blackbox will generate Fortran source code specifically formatted to be used as an external file.
- **Automatic Object/Library file Creation:** The option to compile the generated source file into an object file is provided. Note that if the module being blackboxed contains other modules within it, all object files created will be bound together into a static library.
- **Automatic Component Creation:** A component definition and instance is created, based on the contents of the module hierarchy. This includes ports, parameters, graphics and script segments.

### Restrictions

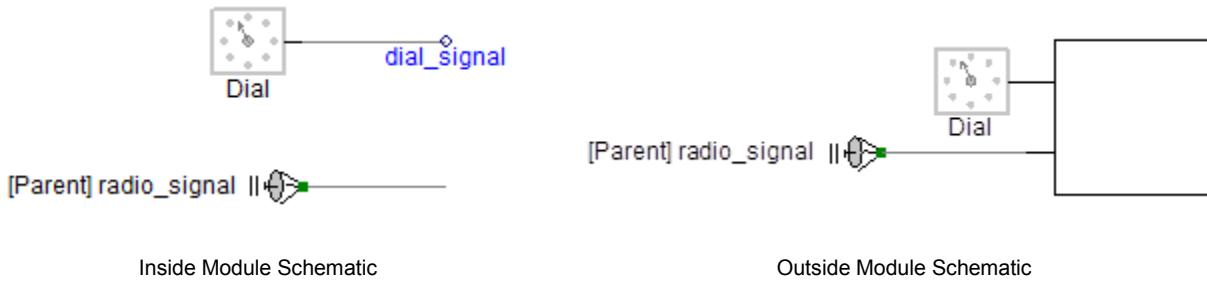
With any automated process, there are restrictions that must be adhered to before using the feature. These are explained below.

## Runtime Objects, Output Channels and Radio Links

Runtime objects, output channels and radio links are not supported by the blackbox feature, and so therefore the module schematic cannot contain any of these components.



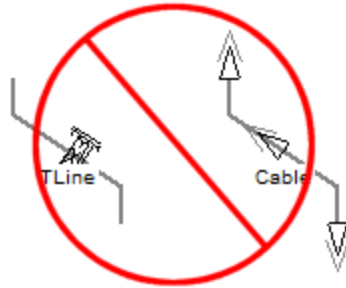
If the module you want to blackbox contains some or all of these components, then they must be removed and/or substituted by supported components. Input-based components, such as the slider, switch and dial, can be substituted in one of two ways: The preferable method is to move them out of the module so that their signal is passed via a port or parameter:



Note also that output-based components, such as radio link transmitters and output channels, must be removed completely before commencing with black boxing.

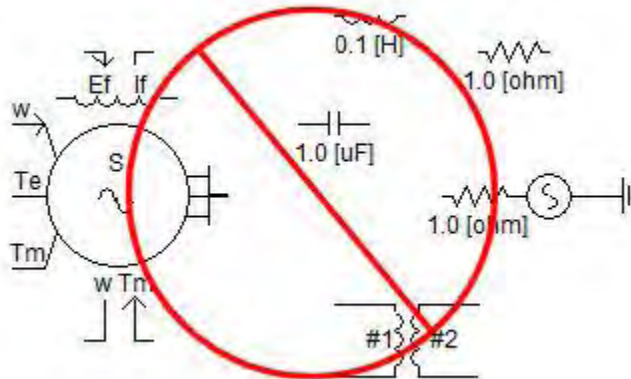
## Transmission Line Components

The module component you are blackboxing cannot contain any transmission lines or cables.



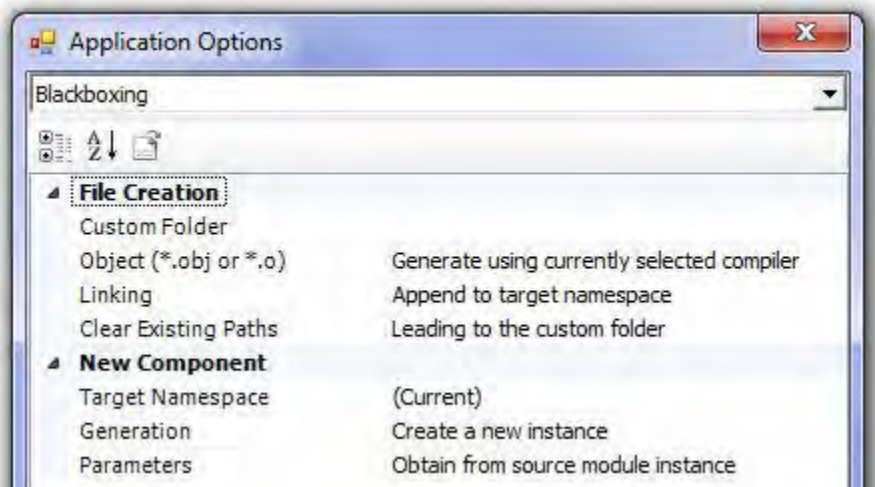
## Electrical Components

Presently, the blackboxing feature does not support any electrical components on the schematic (purely controls only). Future versions of blackboxing are expected to support electrical systems however.



## User Controlled Options

There are a few application level options that may be adjusted to help customize how and where your blackbox is created. These can all be found within the *Application Options* dialog.



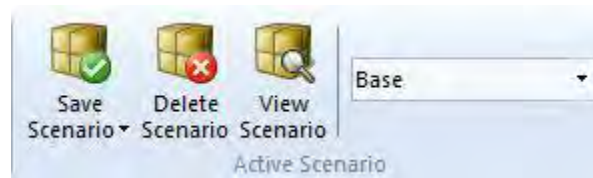
For details on the blackbox control options, see Blackboxing.

## Scenarios (Control Templates)

The *Scenarios* feature (formerly known as *Control Templates*) provides users with the ability to save unique sets of dynamic control settings (i.e. Dial, Switch and Slider components) into scenario templates. In a sense, saving a scenario is similar to taking a snapshot of all dynamic control settings in a project, which may thereafter be referenced at any time.

Scenarios are saved as part of the project file, so when a project with scenarios is loaded, any saved set of settings can be immediately reinstated by selecting the desired scenario. It is possible to store multiple scenarios so that the user can easily switch between them, without having to manually reset all the control components. In fact, scenarios can be changed during a simulation!

Scenarios can be accessed via the **Home** tab of the ribbon control bar.



### Saving the Active Scenario

Before saving your dynamic control settings as a scenario, first ensure that all Dial, Switch and Slider control components have been set, and that the current project is the active project. Press the **Scenarios** button to bring up the scenarios pop-up menu as shown below:



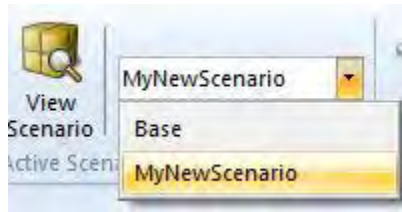
Select either **Save** or **Save As** depending on what you want to do. If this is the first scenario created for this project, select the latter.

**NOTE:** You may store all settings to the default scenario if you wish. Simply ensure that **Default** appears in the control setting window and select **Save**. Also, ensure that all controls possess a distinct name; otherwise an error message will be posted.

If creating a new scenario (i.e. **Save As**), a dialog window will pop-up – enter a name for the scenario.



When finished, press the OK button. Your dynamic control settings will now be stored in this new scenario. Whenever you want to revert back to this scenario, simply left-click the down arrow on the scenario drop list and select it.



## Delete Active Scenario

You can delete any scenario by first making the scenario to be deleted the active scenario. Then select the **Delete Scenario** button in the **Home** tab of the ribbon control bar:



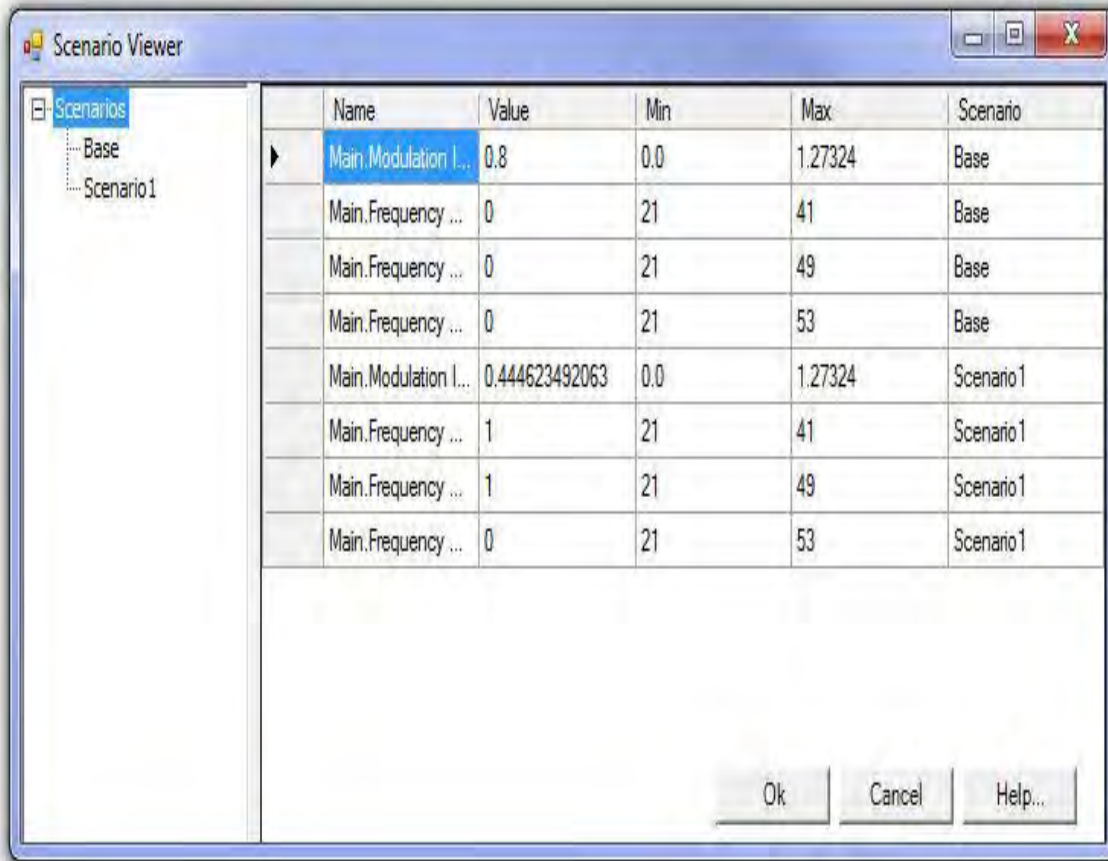
## Scenario Viewer

The application provides a scenario viewer so that the dynamic control settings in each scenario can be easily viewed and verified. To bring-up the scenario viewer, select the **View Scenario** button in the **Home** tab of the ribbon control bar:



## Features and Functions

The scenario viewer window contains a few simple features to help view and organize scenario data. The viewer consists of two sections: A scenarios tree on the left, which displays all scenarios saved in the project and a data sheet view to the right. The data sheet view lists the settings (specific to whatever scenario is selected in the scenario tree) of all control interfaces in the project.



To view the settings pertaining to a single scenario, simply select that scenario on the scenario tree.

Values in the **Value**, **Min** and **Max** columns may be modified and saved directly from the *Scenario Viewer*. Simply double-click on the field and enter a number.

	Main.Button		0	1	Default
	Main.Slider	0.567	0.0	1.0	Default
	Main.Switch	0	0	1	Scenario1

## Global Substitutions

Global substitutions provide a mechanism to use (i.e. substitute) pre-defined, constant values globally throughout a project. A global value can be substituted within any module at any level in the project hierarchy, and is normally done so via component input parameters.

Global substitutions are similar to substitutions used within component definitions, in that they are simply a text string, which represents a literal value (or another string). Once defined, the text string (or key) can be inserted into any component input parameter; its value will be substituted by the PSCAD compiler when the project is built.

The syntax for using a defined global substitution is as follows:

\$ (<Key>)

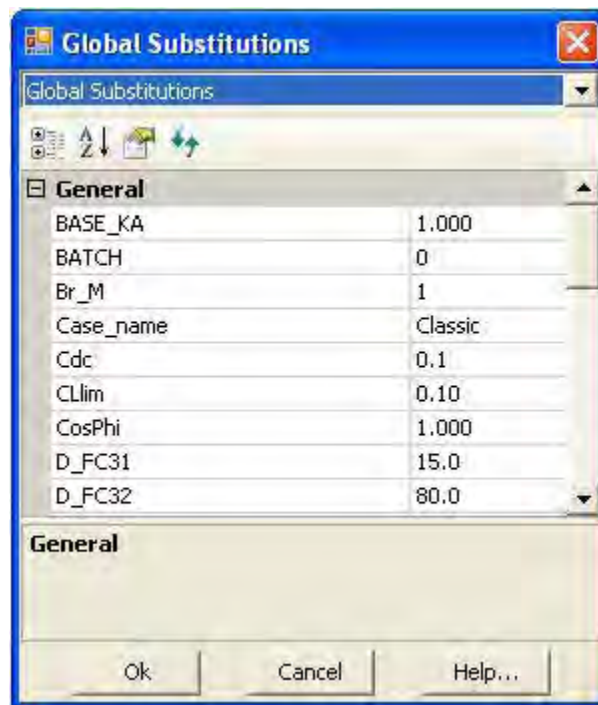
The substitution contains an item <Key>, where:

- **<Key>**: Item key (defined global parameter) that is to be placed at the substitution point.

Component input parameters containing global substitutions will be pre-processed before their value is used within the component code (or canvas if a module). To the component, the input parameter appears exactly as if the user entered the data directly.

## Viewing Defined Global Substitutions

In the workspace window, right-click on a project and select **Global Substitutions | Edit Values...** (or simply press **Ctrl + g** on your keyboard), to bring up the *Global Substitutions* dialog window.



Existing global substitution values may be modified directly from the dialog. For example, say two substitutions to represent system frequency and transmission line length (called *f* and *length*) already exist. A user wants to set the system frequency globally as *50 Hz*, and the transmission line length globally as *100 km* in a project. This can be modified as follows in the global substitutions dialog:





## Adding/Editing Global Substitutions

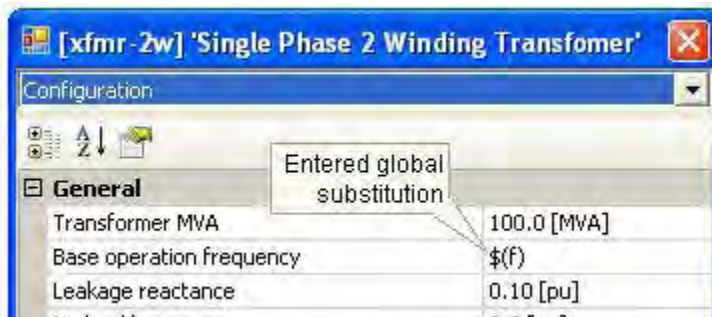
Global substitutions are handled in a very similar manner as component input parameters, in that they are defined as 'parameters' of the *Main* page. As such, the user can add new, and edit existing global substitutions by using the parameter editor for the *Main* module definition (i.e. the *Parameters* section).

To add or edit a global substitution, right-click on a project and select **Global Substitutions | Add or Modify Items...** This will simply open the *Main* module parameters section.

For more details on adding and editing input parameters, see The Parameters Section in Chapter 9 – Component Design.

## Utilizing Global Substitutions

To utilize these substitutions, enter either of the keys in the appropriate component input parameter field, according to the syntax given above:



Entering *f* in Single-Phase Transformer Parameters Dialog

Entering *length* in Transmission Line Configuration Dialog

## Unit System

Component parameter units perform limited conversion and scaling, depending on the unit entered and the defined default (or *Target*) unit for that particular input parameter. The unit system includes base units for time, length, weight and speed (both translational and rotational), as well as electrical units, such as voltage, current and power.

## Enabling the Unit System

To enable the unit system, click the associated check box in the Dynamics tab of the Project Settings dialog. This option is normally enabled by default.

## Unit Format

Units are entered exclusively into component input parameter fields and may only be associated with literal input data. That is, units are invalid when using input variables or global substitutions. All entered units must be separated at least one space from the entered value. For example:

General	
Generator Rated MVA	1.62 [MVA]
Machine rated angular mech. speed	376.99 [rad/s]
Rotor Radius	31.5 [m]
Rotor Area	3177 [m*m]
Air density	1.27 [kg/m^3]
Gear box efficiency	
Gear Ratio - Machine/turbine	
Equation for power coefficient	MOD ?

Examples of Units Entered in Component Input Fields

## Base Units

The foundation of the unit system is the base unit. Base units represent all units that are recognized by the unit system. Use of units in an input field is optional, and if the input does not have a unit specification included with it, then the unit will assume that of the **Target Unit** for the input parameter. If units are specified, then the unit conversion is strict and will evaluate an input parameter field as indeterminate (**#NaN**) if it does not recognize the unit or fails to process a compound unit. All the base units, except a few noted exceptions, conform to the International System of Units (SI).

The following lists the valid (standard) base units recognized by the Unit System. All symbols are case sensitive and must appear exactly as shown to maintain validity:

	Name	Symbol	Conversion Factor	Description
Electrical	Volt	V		Electrical voltage
	Ampere	A		Electrical current
	Ohm	ohm		Electrical resistance
	Siemens	S		Electrical conductance 1
	Siemens	mho		Electrical conductance 2
	Siemens	mhos		Electrical conductance 3
	Watt	W		Electrical power 1
	Volt-Amps	VA	1 VA = 1.0 W	Electrical power 2

	Volt-Amps	VAR	1 VAR = 1.0 W	Electrical power 3
	Horsepower	hp	1 hp = 746.0 W	Electrical power 4
	Farad	F		Electrical capacitance
	Henry	H		Electrical inductance
	Tesla	T		Magnetic flux density
Time	Second	s		Time in seconds 1
	Second	sec		Time in seconds 2
	Second	Sec		Time in seconds 3
	Minute	min	1 min = 60 s	Time in minutes
	Hour	hr	1 hr = 3600 s	Time in hours
	Day	day	1 day = 86400 s	Time in days
	Frequency	Cycles per Second	Hz	
Length	Metre	m		Length in metres
	Inch	in	1 in = 0.0254 m	Length in inches
	Feet	ft	1 ft = 0.3048 m	Length in feet
	Yard	yd	1 yd = 0.9144 m	Length in yards
	Mile	mi	1 mi = 1609.344 m	Length in miles
Weight	Gram	g		Weight in grams
	Pound	lb	1 lb = 453.59237 g	Weight in pounds
Rotational	Revolutions	rev		Revolutions ( 1 [rev] = one complete revolution)
	Radian	rad	1 rad = $1/2\pi$ rev	Angle in radians
	Degree	deg	1 deg = $1/360$ rev	Angle in degrees
Other	Revolutions per Minute	rpm	1 rpm = $1/60$ rev/s	Rotational speed in revolutions per minute
	Revolutions	RPM	1 RPM = $1/60$ rev/s	Rotational speed in revolutions per minute 2

per Minute

Per-Unit	pu		Per-unit quantity
Per-Unit	p.u.		Per-unit quantity 2
Percent	%	1 % = 0.01 pu	Percent quantity

## Prefixes

The unit system utilizes a limited list of SI prefixes in order to allow for scaling of base units. Prefixes must precede a valid base unit, and may be inserted anywhere within compound units.

The following table lists all valid prefixes:

Name	Symbol	Scale Factor
tera	T	$10^{12}$
giga	G	$10^9$
mega	M	$10^6$
kilo	k	$10^3$
milli	m	$10^{-3}$
micro	u	$10^{-6}$
nano	n	$10^{-9}$
pico	p	$10^{-12}$

## Target Units

The unit system will determine the final conversion or scaling factor to apply, based on the target unit of the input parameter field. The target unit is the symbol entered in the **Units** field (i.e. the default unit) in the Parameters section of the component definition.

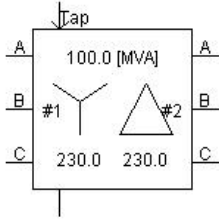
**NOTE:** You can view the target units of any component by invoking the View Properties dialog: Right-click on the component and select **View Properties**.

Target units are not limited to the base units alone, and may include prefixes by default (i.e. *kA*): In instances such as these, any prefixes in the target unit will be considered if further scaling is performed later on. In fact, this is quite common in the master library, where many target units are specified in [kA], [kV] or [uF].

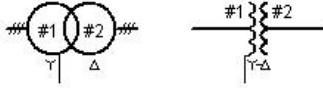
---

### EXAMPLE 5-5:

The master library component 3-Phase 2-Winding Transformer contains an input parameter called *Winding 1 Line to Line Voltage (RMS)*, whose target unit is specified as *kV* in the *Units* field.



3-Phase 2-Winding Transformer



Winding 1 Line to Line voltage (RMS)	<input type="checkbox"/> Real
Description	Winding 1 Line to Line voltage (RMS)
Symbol	V1
Group name	
Default units	kV
Minimum value	0.001
Maximum value	
Data type	Constant

Input Field Property Settings (V1)

A user enters data into the *Winding 1 Line to Line Voltage (RMS)* field as *0.153 [MV]*. Given that the target unit contains the prefix *k*, the application will understand that any quantity entered in this parameter field must be converted back to kilovolts (not the base unit of volts [V]). Therefore in this case, the quantity will be multiplied by a scale factor of 1000 to convert it from *0.153 [MV]* back to *153.0 [kV]*.

Whether or not target units include prefixes is normally of no concern, unless of course a new component is being designed. Provided that the base of the entered unit matches that of the target, all scaling and conversion is performed automatically.

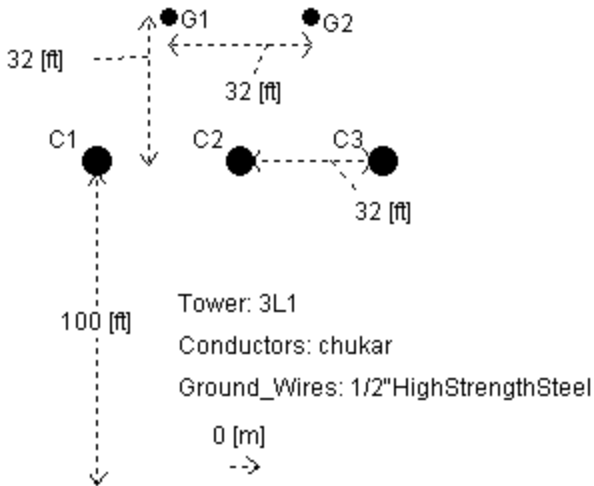
## Unit Conversions

The most useful aspect of the unit system is the ability to convert one unit to another, be it an imperial/metric conversion or simply converting from one form to another, such as radians to degrees.

Converting units from one form to another is relatively straightforward, the only rule is that the conversion must take place within the same base unit class; for example, [m] to [ft] (both units measure length) or [sec] to [hr] (both units measure time). Valid prefixes may be included in the conversion as well, for instance, [km] to [mi].

### EXAMPLE 5-6:

A user is designing a transmission line tower. The default units for the tower dimensions are in metres, but the user's specification sheets give the dimensions in feet. The unit system will allow the user to enter this data directly in feet, without the need to convert to metres.



Transmission Line Tower Component

General	
Tower Name	3L1
Height of All Conductors (Measured at Tower)	100 [ft]
Horiz. Spacing Between Phases	32 [ft]
Relative X Position of Tower Centre on Right of Way	0 [ft]
Shunt Conductance	1.0E-11 [mho/m]
Show Graphics of Cond. Sag?	No

Entered in feet

Transmission Line Tower Parameter Dialog

PSCAD will automatically convert all units entered in feet, back to metres before the Line Constants Program is called to solve the line.

## Compound Units

The unit system will recognize three types of arithmetic operator within the unit brackets, in order to allow for combining (or compounding) units together. These are:

Arithmetic Operator	Description
*	Multiply
/	Divide
^	Exponent

When dealing with compound units, it important to note some simple rules. Failure to follow these rules may result in invalid unit conversion:

- The sequence in which the arithmetic operators occur in the entered unit must match that of the target unit.

	Entered	Target
<b>Correct:</b>	[hp*min/MVA]	[MW*s/MVA]
<b>Incorrect:</b>	[hp/MVA*min]	

- The total number of arithmetic operators in the entered unit must match that of the target unit. That is, you cannot substitute an exponent for multiple 'divisions' or 'multiplications'.

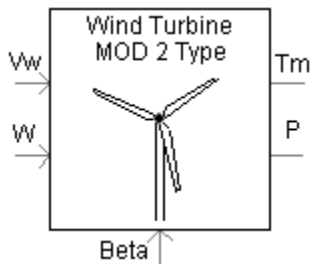
	Entered	Target
<b>Correct:</b>	[ft*ft]	[m*m]
<b>Incorrect:</b>	[ft^2]	

- Multiple 'divide' symbols are not allowed - only one 'divide' per compound unit may exist.

	Entered	Target
<b>Correct:</b>	[lb*ft/s]	[kg*m/s]
<b>Incorrect:</b>	[lb/s/ft]	

EXAMPLE 5-7:

The master library component Wind Turbine contains an input parameter called *Machine rated angular speed*, whose target unit is specified as *rad/s* in the *Units* field.



Wind Turbine

<input type="checkbox"/> Machine rated angular mech. speed	<input type="text" value="IR"/> Real
Description	Machine rated angular mech. speed
Symbol	Wrat
Group name	
Default units	rad/s
Minimum value	n

Input Field Properties Dialog (*Wrat*)

A user enters data into this parameter as *60.0 [Hz]*. This is a valid unit in this case as both *Hz* and *rad/s* are essentially the same type of measure, where  $2\pi [rad/s] = 1 [rev/s] = 1 Hz$  (see *Base Units* above).

EXAMPLE 5-8:

The master library component Wind Turbine (described above) also contains an input parameter called *Air Density*, whose target unit is specified as *kg/m^3* in the *Units* field.

[-] Air density	IR Real
Description	Air density
Symbol	Airden
Group name	
Default units	kg/m <sup>3</sup>
Minimum value	0
Maximum value	
Data type	Constant

Input Field Properties Dialog (Airden)

A user enters data into this parameter as the equivalent in imperial units  $0.07647 [lb/ft^3]$ . The units converter will apply the appropriate scale factors to this number so that the quantity will appear to EMTDC as it is still in  $kg/m^3$ .

## Verifying Unit Conversions

It is generally recommended to ensure that any unit conversions performed in a component are verified before proceeding with the simulation. This can be accomplished easily by using the Properties Viewer. This dialog displays the target unit, the entered data, and the final value following the conversion.

The following diagram shows these three parameters following the changes made the 'Air Density' input, as outlined in *Example 5-4*.

Name	Caption	Type	Unit	Minimum	Maximum	Data	Value	F
Gmva	Generator Rated MVA	Real	MVA	1e-307		1.62 [MVA]	1.62	*
Wrat	Machine Rated Angular M...	Real	rad/s	1e-307		376.99 [rad/s]	376.99	*
Rad	Rotor Radius	Real	m	0		31.5 [m]	31.5	*
Area	Rotor Area	Real	m*m	0		3177 [m*m]	3177.0	*
Airden	Air Density	Real	kg/m...	0		1.225 [kg/m^3]	1.225	*

Wind Turbine Air Density Parameter

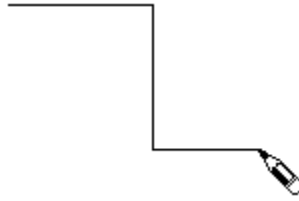
## Wire Mode

PSCAD includes a special wire drawing feature called *Wire Mode*, which enables the user to quickly draw interconnecting Wires between components.

To invoke wire mode, click the **Wire Mode** button in the **Home** tab of the ribbon control bar or press **Ctrl + w** on your keyboard. With the project open in Schematic view, move the mouse pointer onto the project canvas. The mouse pointer will have turned into a pencil.



To draw a wire, move the cursor to the node where you want the line to start and left-click. Move the cursor to where you want the line to end and left-click again to complete the wire. *Multi-segment Wires* may be built by continuing to left click at different points, then right-clicking at the last point.





To turn-off *Wire Mode*, press the **Esc** key, or press **Ctrl + w**, or click the **Wire Mode** button again.

## Drag and Drop

Drag and drop functionality greatly improves the efficiency of project design – especially the construction of online plots and controls. Drag and drop may be performed directly on the Schematic canvas, or definitions may be instantiated by drag and drop from the workspace window.

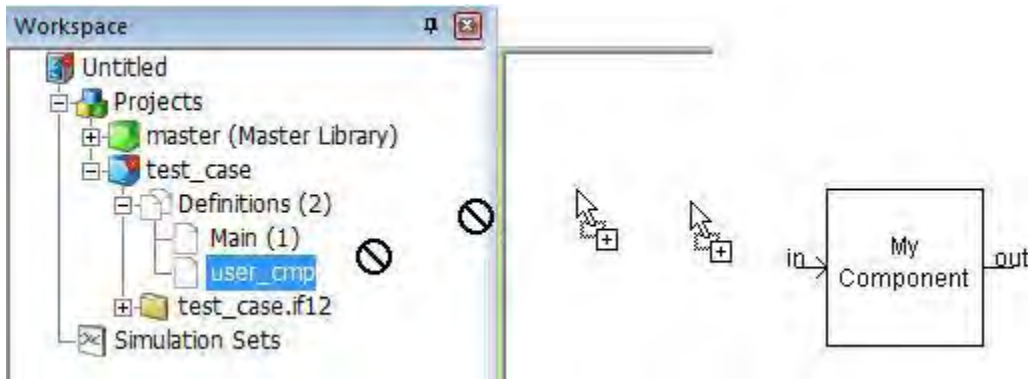
The drag and drop feature uses mouse pointer icons to indicate whether or not a dragged object may be placed under the current mouse position:

-  Drop position is valid
-  Drop position is invalid

## Create a Component Instance

Component instances can be created directly from the corresponding definition in the workspace.

1. Navigate to the *definitions* list in the workspace secondary window.
2. Depress the **Ctrl** key and **select and hold** the desired component definition with the left mouse button.
3. **Drag** the mouse pointer to a blank area on the Schematic canvas and release the button to **drop**.



Drag and Drop onto Schematic Canvas from Workspace

## Copy a Component Instance

All component instances appearing on the Schematic canvas may be copied and pasted using drag and drop.

1. Hold down the **Ctrl** key.
2. Move the mouse pointer over the component instance to be copied.
3. Select and hold with your **left mouse button**.
4. Drag the mouse pointer to a blank area of the Schematic canvas and release the mouse button to paste.

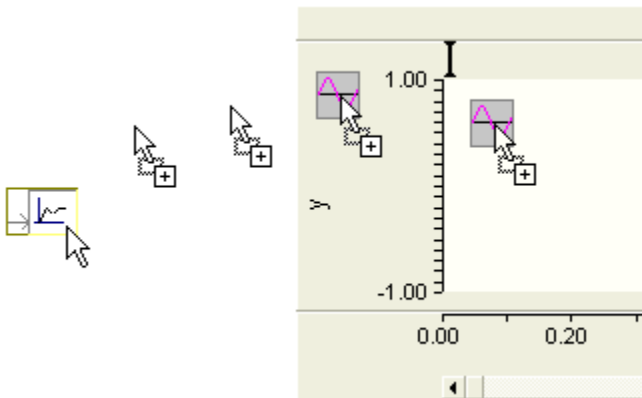


Copying a Component Instance

## Adding a Curve to a Graph

Curves can be added directly to graphs using drag and drop.

1. Hold down the **Ctrl** key (or use either the **Ctrl** or **Shift** key if the *Controls and Curve Creation* option is set to *Use shift key to create* in the Application Options dialog).
2. Move the mouse pointer over the desired Output Channel component instance.
3. Select and hold with your **left mouse button**.
4. **Drag** the mouse pointer to the desired graph and release the mouse button to paste.

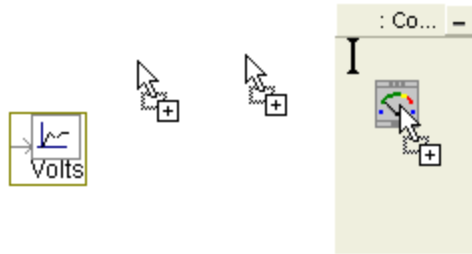


Adding a Curve to a Graph from a Output Channel

## Adding a Meter to a Control Panel

Meters can be added directly to control panels using drag and drop.

1. Hold down the **Ctrl** key (use the **Shift** key if the *Controls and Curve Creation* option is set to *Use shift key to create* in the Application Options dialog).
2. Move the mouse pointer over the desired Output Channel component instance.
3. Select and hold with your **left mouse button**.
4. **Drag** the mouse pointer over the desired control panel title bar and release the mouse button to paste.

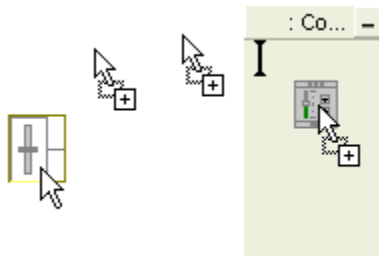


Adding a Meter to a Control Panel from a Output Channel

## Adding a Control Interface to a Control Panel

Control interfaces can be added directly to control panels using drag and drop.

1. Hold down the **Ctrl** key (use the **Shift** key if the *Controls and Curve Creation* option is set to *Use shift key to create* in the Application Options dialog).
2. Move the mouse pointer over the desired control component instance.
3. Select and hold with your **left mouse button**.
4. **Drag** the mouse pointer over the desired control panel title bar and release the mouse button to paste.



Adding a Control Interface to a Control Panel from a Output Channel

## Move/Copy Meters and Curves Between Graphs/Panels

Once a curve or meter interface has been placed in a graph or control panel, it may then be copied or moved as follows:

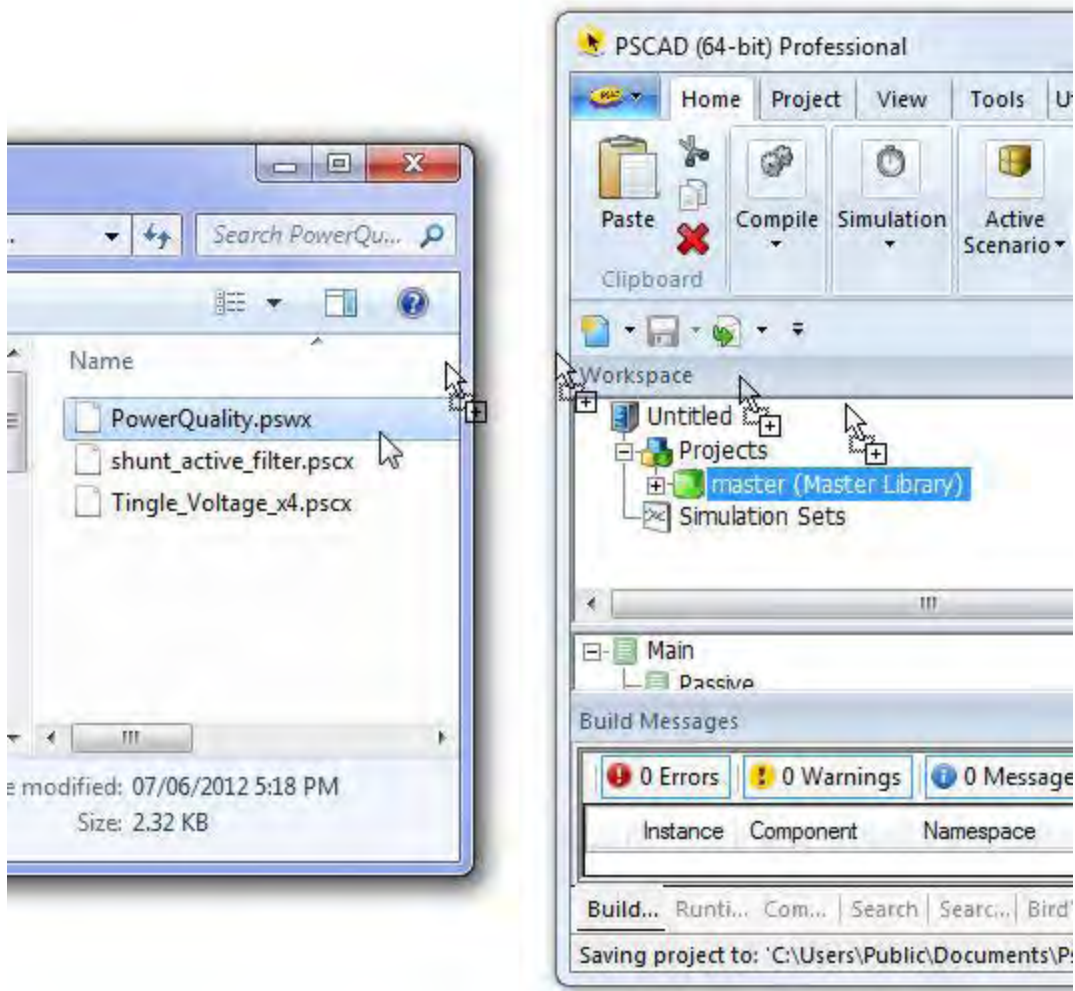
- A curve may be copied within the same graph, to another graph, or to a panel.
- A meter interface may be copied within the same panel, to another panel, or to a graph.
- A curve may be moved within the same graph or to another graph.
- A meter interface may be moved within the same panel or to another panel.

To *copy* a curve/meter interface, hold down the **Ctrl** key and then left-click and hold the object, drag and drop as described above.

To *move* a curve/meter interface, left-click and hold the object, drag and drop as described above.

## Loading Projects and/or Workspaces

Both library and case projects, or entire workspaces, may be loaded directly into PSCAD via drag and drop. For example, open PSCAD and File Explorer:

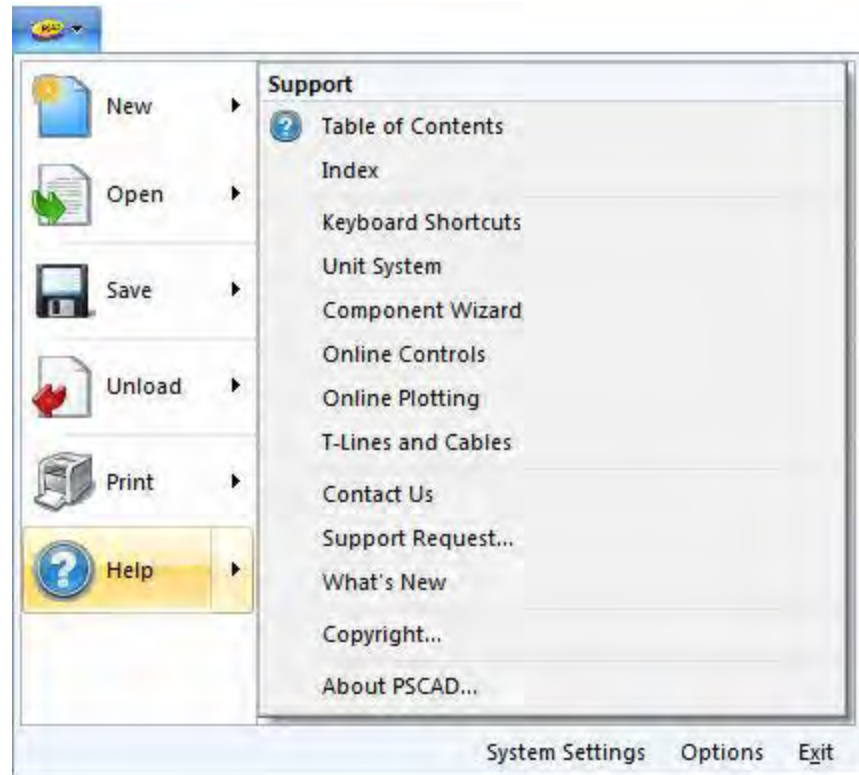


Loading a Workspace from File Explorer

To *load* a project or workspace, left-click and hold, drag and drop as described above.

## Accessing the Online Help System

The online help system table of contents (TOC) or index can be accessed directly through the **Help** menu in the **PSCAD** tab of the ribbon control bar, as shown below:



You may also bring up the online help by simply pressing the **F1** key.

Master library component specific help can be accessed by one of the following methods:

- Select the component and then press **F1**.
- Right-click on the component and select **Help...** from the pop-up menu.
- Edit the component parameters and click the **Help...** button.

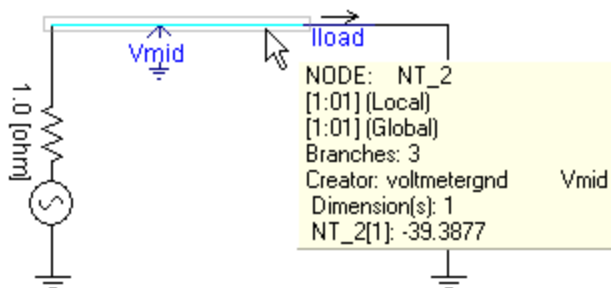
In addition to the above, other dialog windows in PSCAD will possess a **Help** button (usually in the bottom right corner). Click this button to bring up the online help topic specific to that dialog. See PSCAD On-Line Help System for more details on the help system in general.

## Tool Tips

Tool tip windows (otherwise known as *flybys*) are especially designed for obtaining dynamic circuit information 'on the fly'. Tool tips are available both for providing pop-up help on component instances, and for monitoring electrical or control signal quantities during a simulation.

To monitor an electrical or data signal during a run:

1. **Left-click** on a blank part of the Schematic canvas (this is to ensure that the Schematic canvas is the active window).
2. Move and hold the mouse pointer over the signal wire that you want to monitor. In a second or two, a window should pop up as shown below:



If the mouse pointer is held over a wire carrying an electrical signal, the flyby will indicate the voltage in kilovolts at that electrical node. The flyby shown above indicates that this is EMTDC node *NT\_2*, which is a scalar quantity (i.e. single-phase node). The value of the node voltage at the instant this snapshot was taken was *-39.3877 kV*.

If the mouse pointer is held over a data (control) signal, the signal value will be shown in the flyby.

**NOTE:** Flyby information for data signals cannot be viewed unless the Store Feed Forward Signals for Viewing parameter in the *Dynamics* section of the *Project Settings* dialog is enabled.

## Searching

If it is necessary to search a project for a signal name, port connection name, node number, etc., the Search feature may be used. To bring up the search dialog window, either:

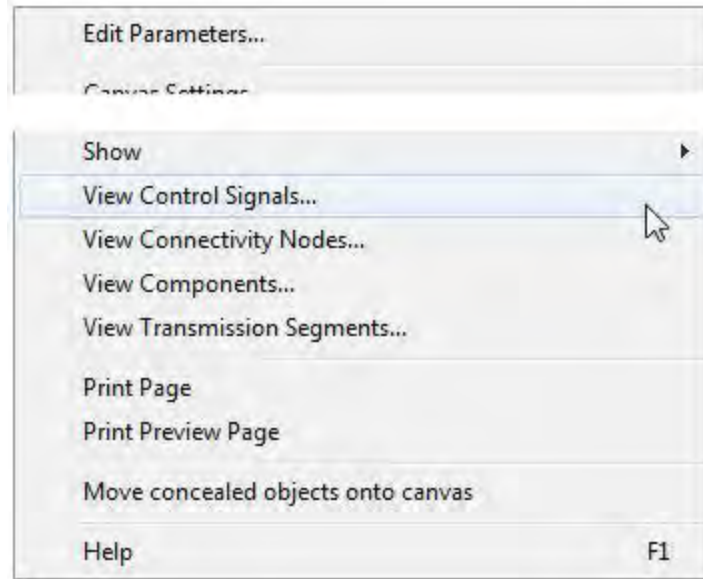
- Press **Ctrl + Shift + F** on your keyboard, while the Schematic window is open
- Press the **Search** button in the **Home** tab of the ribbon control bar.



For more details on searching, see [Searching](#).

## Component, Transmission Segment and Control Signal Tables

The component and signal table viewers provide a module-specific list of either object type (depending on which viewer you are looking at), organized in spreadsheet form. To invoke the corresponding viewer, right-click on a blank part of the canvas in Schematic view, and select **View Control Signals...**, **View Connectivity Nodes...**, **View Components...** or **View Transmission Segments...**



Note that these selections are disabled (i.e. not available) until the project has been compiled.

## Component Table

The component table lists all component instances existing in a particular module. Along with the component name and type, some other important information is also summarized to help users when debugging or understanding their project.

	Instance	Order	Name	Description	Name Parameter	Sequence	Location
1	<a href="#">225520775</a>	1	mult	Multiplier		390	Dsdyn
2	<a href="#">636384835</a>	2	div	Divider		240	Dsdyn
3	<a href="#">2128714866</a>	3	consti	Integer Constant		270	Dsdyn
4	<a href="#">1839050357</a>	4	div	Divider		180	Dsdyn

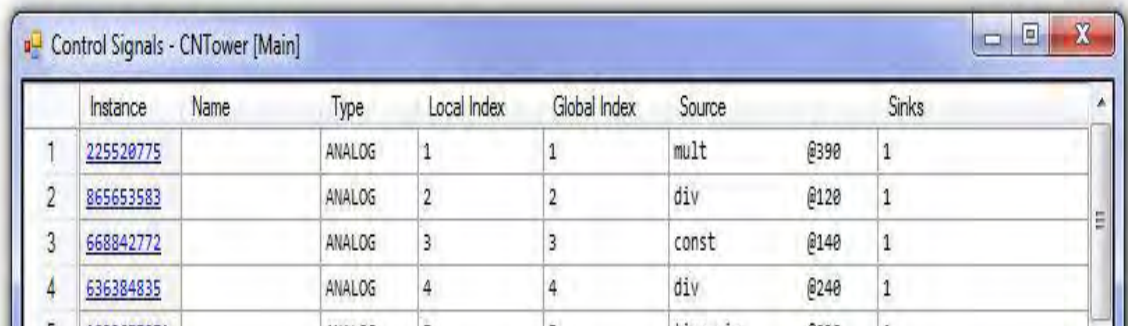
The columns displayed in this viewer are described below:

- **Instance:** Navigate directly to any component instance in the list. Simply left-click on the hyperlink.
- **Order:** Simple order number for display.
- **Name:** The component definition name.
- **Description:** The description of the component definition. See Editing Definition Settings.
- **Name Parameter:** If the component has an input parameter entitled *Name*, then its contents will be displayed here. For example, a voltmeter component will display the name of the measured signal created.

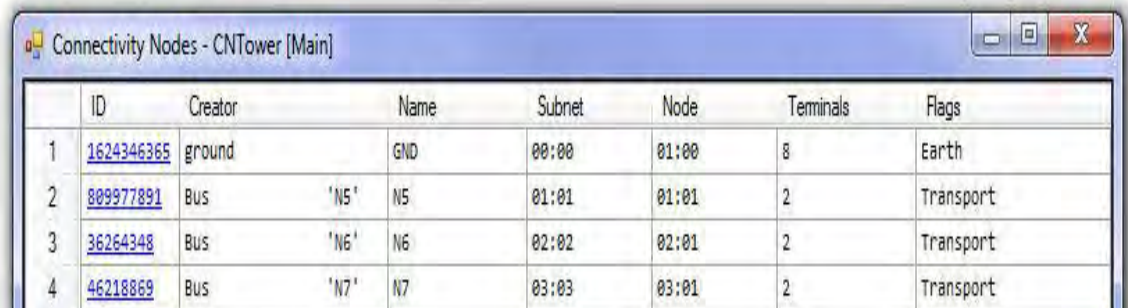
- **Sequence:** The system dynamics sequence number of the component instance. See Component Ordering in the chapter *Project Debug and Refinement* for more information.
- **Location:** The location within the system dynamics where the instance resides. This can be either DSDYN or DSOUT.

## Control Signal and Connectivity Node Tables

The signal table viewers summarize all control and electrical signals in a particular module. Information is given, such as the EMTDC control signal variable name and from where the signal is sourced. For electrical signals, node name and index are given.



	Instance	Name	Type	Local Index	Global Index	Source	Sinks
1	<a href="#">225520775</a>		ANALOG	1	1	mult @390	1
2	<a href="#">865653583</a>		ANALOG	2	2	div @120	1
3	<a href="#">668842772</a>		ANALOG	3	3	const @140	1
4	<a href="#">636384835</a>		ANALOG	4	4	div @240	1



	ID	Creator	Name	Subnet	Node	Terminals	Flags
1	<a href="#">1624346365</a>	ground	GND	00:00	01:00	8	Earth
2	<a href="#">809977891</a>	Bus	'N5'	N5	01:01	2	Transport
3	<a href="#">36264348</a>	Bus	'N6'	N6	02:02	2	Transport
4	<a href="#">46218869</a>	Bus	'N7'	N7	03:03	2	Transport

The columns displayed in these viewers are described below:

Control Signals:

- **Instance:** Navigate directly to any signal in the list. Simply left-click on the hyperlink.
- **Name:** EMTDC signal variable name. That is, the name given to the signal node by the PSCAD compiler for use by EMTDC.
- **Type:** The signal type. Can be ANALOG (REAL), DIGITAL (INTEGER) or LOGICAL.
- **Local Index:** A local (i.e. module based) index number given to the signal.
- **Global Index:** A global (i.e. project based) index number given to the signal.
- **Source:** The source or creator of the signal. Normally a component instance.
- **Sinks:** The number of points at which the signal is received.



Electrical Signals:

- **ID:** Navigate directly to any node in the list. Simply left-click on the hyperlink
- **Creator:** The creating object for the signal.
- **Name:** EMTDC electrical node name. That is, the name given to the electrical node by the PSCAD compiler for use by EMTDC.
- **Subnet:** Global and local subsystem number.
- **Node:** Global and local node number.
- **Terminals:** The total number of electrical branches attached to the node.
- **Flags:** Electrical node type.

## Transmission Segment Table

The transmission segment table lists all transmission line and cable instances existing in a particular module.

	Instance	Line Name	Steady State Frequency	Length	Dimension	Connection Mode	Couple Enabled
1	<a href="#">310323642</a>	CNT_3	1000000	0.00869	1	Local connection	Disabled
2	<a href="#">1194138762</a>	CNT_4	1000000	0.079	1	Local connection	Disabled
3	<a href="#">926351712</a>	CNT_5	1000000	0.03	1	Local connection	Disabled
4	<a href="#">1346041279</a>	CNT_6	1000000	0.335	1	Local connection	Disabled

The columns displayed in this viewer are described below:

- **Instance:** Navigate directly to any component instance in the list. Simply left-click on the hyperlink.
- **Line Name:** The segment name of this line or cable.
- **Steady-State Frequency:** The value of the steady-state frequency input parameter of this line or cable.
- **Length:** The value of the length input parameter of this line or cable.
- **Dimension:** The value of the dimension input parameter of this line or cable.
- **Connection Mode:** The value of the connection mode input parameter of this line or cable.
- **Couple Enabled:** The value of the couple enabled input parameter of this line or cable.

**NOTE:** This table is for viewing only. For an interactive viewer where you can actually modify cell values, see Parameter Grid Pane.

## EMTDC Output Files

EMTDC output files are formatted text files, which organize all data into columnar format. Each column, except the first, which is always time, represents the recorded data from a corresponding Output Channel. For example, if two output channel components exist in the project, then three columns of data will appear in the EMTDC output file. The following is a segment of text from a typical EMTDC output file:

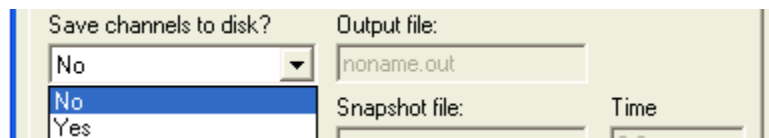
Test Case		
0.0000000000000000	0.0000000000000000	0.0000000000000000
0.1000000000000000E-02	0.0000000000000000	0.0000000000000000
0.2000000000000000E-02	0.86727047422974	0.86727047422974
0.3000000000000000E-02	1.6650619394029	1.7674567004163
0.4000000000000000E-02	1.9545665157651	2.2542364437667
0.5000000000000000E-02	2.0221282586499	2.8373003607589
0.6000000000000000E-02	1.9264422562260	4.0202957514613
0.7000000000000000E-02	2.3912531836698	5.5547061594874
0.8000000000000000E-02	2.8769239640036	6.5791178737352
0.9000000000000000E-02	2.8473253982397	6.9456914675731
0.1000000000000000E-01	2.3232656122503	7.1369144410646
0.1100000000000000E-01	1.2446128466462	7.1669414384869
0.1200000000000000E-01	1.6208317681211	7.0023750435365
0.1300000000000000E-01	1.6458908605563	7.1529925186834
0.1400000000000000E-01	4.2422849293514	9.3453281849245

The project description is written as the first row of text at the top of the file. The first column of data is always the EMTDC simulation time. The subsequent columns are not labelled –see *Column Identification and the Information File* below for more details on this.

EMTDC output files may be used for waveform analysis by a selected post-processing software package. As they are formatted in a delimited columnar format, they can be easily imported into most graphing or data analysis programs.

## Creating Output Files

EMTDC output files are created by choosing **Yes** in the **Save channels to disk?** drop list, in the Runtime section of the *Project Settings* dialog.



## Output File Naming Convention

The name of generated EMTDC output files is dependent on the project namespace, the rank number, the multiple run number and the multiple output file number. The syntax used to name output files is as follows:

```
<namespace>_r##_m#####_##(#).out
```

Where,

- <namespace> The namespace of the generating project.
- r## Two-digit rank number. If the generating project was launched as part of a volley, this number identifies its rank.
- m##### Five-digit multiple run number. If the generating project contains a multiple run control, then this number specifies the run to which the file was generated by.
- ##(##) Two or three-digit file number. If the file number is less than 100, then it will appear as two-digit. If greater than 99, three-digit. See Multiple Output Files below for more.

EMTDC output files are given the extension '\*.out' and are stored in the project temporary folder.

## Multiple Output Files

The maximum amount of columns per output file is 11 (including the time column). Therefore, if more than 10 output channel components exist in a project, multiple output files will be created. For example, if your project contains 23 output channels, a total of three output files will be created.

The naming convention for multiple output files is to simply append a sequential number as a suffix. For example, if the output file is named *abc\_r01\_m00001.out*, and there are three files as described above, the output files would be named *abc\_r01\_m00001\_01.out*, *abc\_r01\_m00001\_02.out* and *abc\_r01\_m00001\_03.out*. This sequential numbering is important when identifying data columns (see Column Identification and the Information File).

## Column Identification and the Information File

As mentioned above, EMTDC output file columns are not labelled. In order to determine which column is what, an information file (\*.inf) is also created along with the output file(s), that contains cross-referencing information. The information file will be named the same as the output file primary file name. For example, if the output file name is 'abc.out', the information file will be named 'abc.inf'. Only one information file is created, even for multiple output files.

A typical information file is shown below for a project containing three output channels.

PGB(1)	Output	Desc="Fund - mag"	Group="Main"	Max=25.0	Min=0.0	Units=""
PGB(2)	Output	Desc="2nd harm - mag"	Group="Main"	Max=25.0	Min=0.0	Units=""
PGB(3)	Output	Desc="3rd harm - mag"	Group="Main"	Max=25.0	Min=0.0	Units=""

At the extreme left is the Output Channel number (i.e. PGB(1), PGB(2), etc.). This number indicates the sequence in which the Output Channel data is written to an output file. In other words, this number corresponds to the output file column number. Remember however, that the first column in the output file is time and is not counted. Therefore, Output Channel 2 (PGB(2)) will actually be the third column from the left in the output file. This column can then be identified using the corresponding Output Channel name (i.e. 'Desc'). In this case, Output Channel 2 happens to be '2nd harm - mag'.

The Output Channel numbers in the information file will continue in sequence with the number of Output Channel components in the project. That is, if there are 50 Output Channels in the project, there will be 50 rows in the information file, numbered up to 50. We already know however, that a single output file will only hold up to 11 columns of data (including time). For 50 Output Channels then, PSCAD will create five output files, where the columns are numbered ignoring the time columns in each file. For example, column five in the 3rd output file would be Output Channel 24.

Here is a simple formula to help identify an output file column:

Single Output File:

$$\text{Output Channel \#} = \text{Output File Column \#} - 1$$

Multiple Output Files:

$$\text{Output Channel \#} = \text{Output File Column \#} - 1 + (10 \times (\text{Output File \#} - 1))$$

---

## Chapter 6

# Online Plotting and Control

The ability to plot and control data during a simulation adds tremendous value to the study environment. PSCAD comes complete a number of special runtime objects, specifically designed to allow for the modification of data signals during the simulation run. Users can interact with models and see the results of that interaction immediately on graphs and curves, all while the simulation is running happily.

Although the majority of plotting in PSCAD is done so on the basis of time, there are special plotting tools available to help with other types of data analysis. The XY Plot for example, is specifically designed to plot one data signal verses another. There are also a variety of other devices available, such as the PhasorMeter and the Oscilloscope; each lends a unique perspective to the data being displayed.

It is also helpful to ensure displayed data is transferable to other analysis tools. Portions of curve data, whole graphs, or even entire graph frames may be copied as a picture, meta file or comma separated variable (\*.csv) files for placement in documents and reports.

## Preparing Data for Control or Display

The PSCAD environment is a graphical user's interface to the EMTDC simulation engine. Therefore in order to control input variables or view simulation data, the user must provide EMTDC with instruction on which variables to make available for viewing or control. This process is performed graphically in PSCAD through a number of special components, sometimes referred to as *runtime objects*.

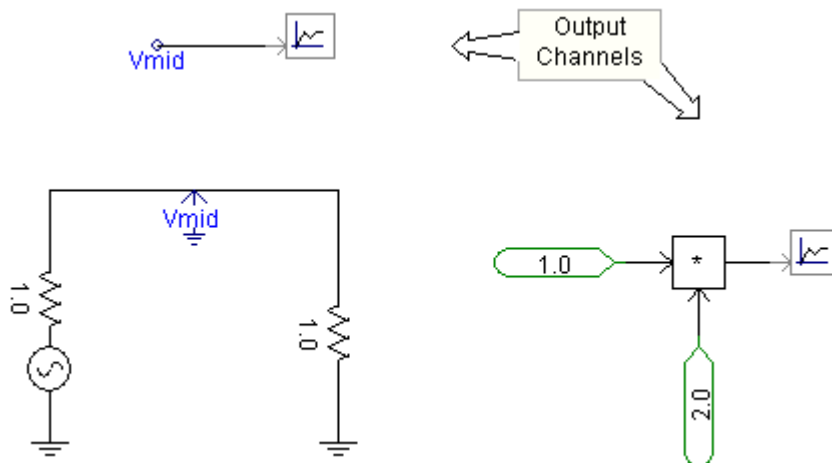
In order to record, display or control any data signals within the environment, the data signal must first be linked to a runtime object. Runtime objects are organized into three main groups:

- **Controls:** Slider, Dial, Switch and Push Button
- **Recorders:** Output Channel and RTP/COMTRADE Recorder
- **Display Devices:** Control Panel, Graph, XY Plot, PolyMeter and PhasorMeter, Oscilloscope

Each runtime object performs a specific task, and may be used in combination with others to control and/or display data.

## Channeling Output Data

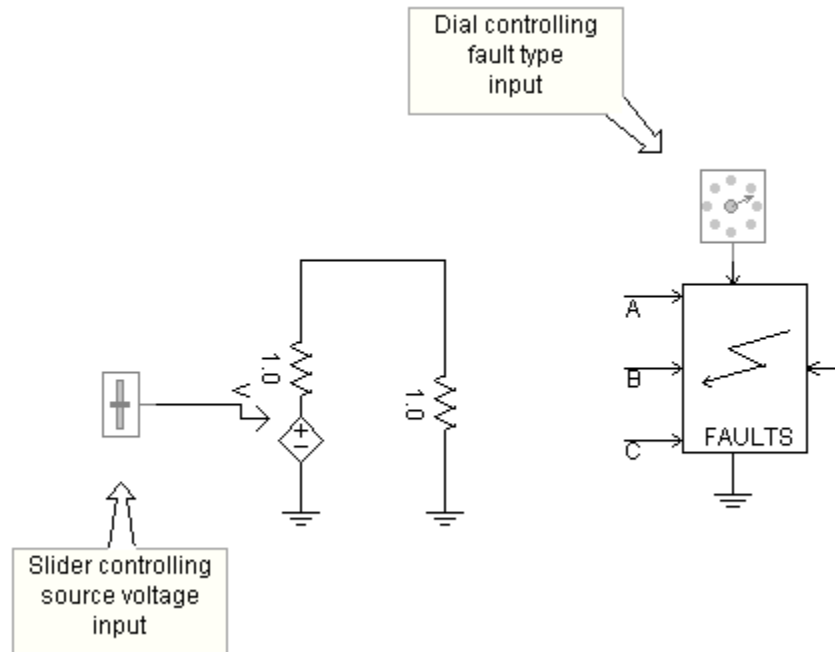
Channelling output refers to the directing of data signals for either online display in a graph or meter, or for output to file. This is accomplished by *channelling* the desired signal through an Output Channel component. For example, the image below shows how to channel a signal from a Voltmeter component (named *Vmid*), as well as how to channel an unnamed signal directly within the *Schematic* canvas:



**NOTE:** Output channel components cannot be directly connected to an electrical signal.

## Controlling Input Data

Controlling input data refers to using one of the control type runtime objects (i.e. Slider, Dial, Switch or Push Button) as a source to control a data signal. This is accomplished by simply adding a control object to the *Schematic* canvas. This signal can be input at any valid location in the schematic as shown below:

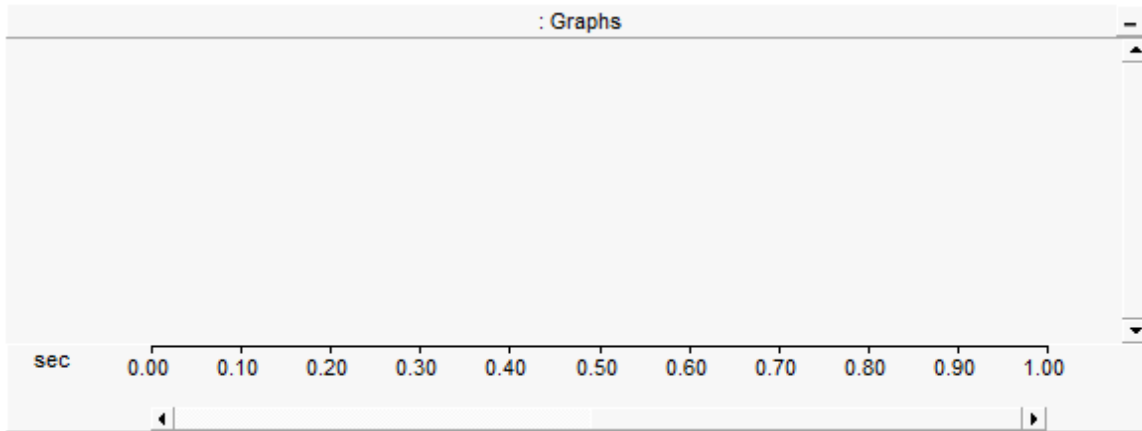


**NOTE:** The control objects cannot be manually adjusted (as they appear above) until they are linked to a control interface. See *Online Controls and Meters* for details.

## Graph Frames

A *Graph Frame* is a special runtime object container used for encapsulating Overlay or PolyGraphs, and can be placed anywhere on the *Schematic* canvas. Once a graph frame has been added, you may then proceed to add as many *Graphs* to it as you wish.

Graph frames are used exclusively for plotting curves versus time. That is, the graph frame horizontal axis is always the simulation time. If you need to a plot a curve as a function of another variable, see XY Plots.



## Adding a Graph Frame

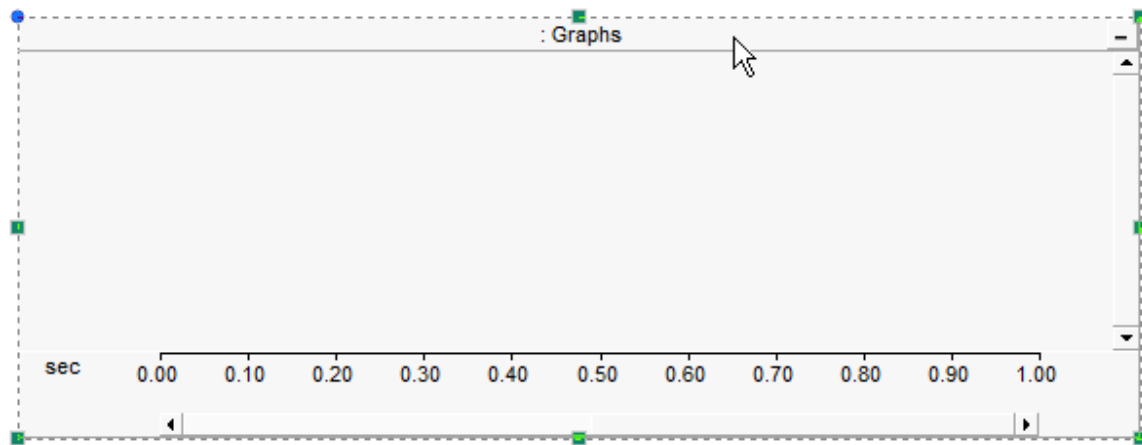
Open the project in *Schematic* view. Right-click on a blank portion of the page and select **Add Component | Graph Frame**, or press the **Graph Frame** button in **Components** tab of the ribbon control bar.



## Moving and Re-Sizing a Graph Frame

To move a graph frame, move the mouse pointer over the title bar and then **left-click and hold**. Drag the frame to where it is to be placed and release the mouse button.

To re-size, move the mouse pointer over the title bar and **left-click** to select the graph frame. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. **Left-click and hold** and then drag then move the pointer to re-size.

## Cut/Copy Frames

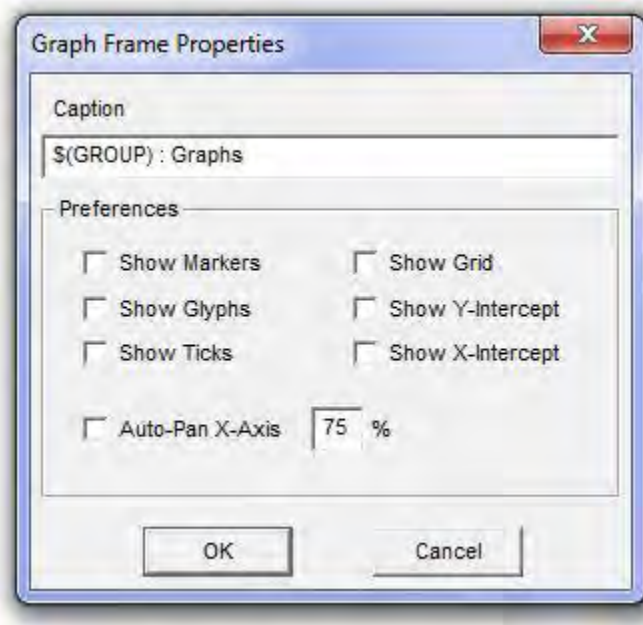
Right-click over the graph frame title bar and select **Cut Frame** or **Copy Frame** respectively. Once a graph frame has been cut or copied it may then be pasted into another location in the project (along with its contents).

## Paste Frame

Cut or copy a graph frame as described above. Right-click over a blank area of the project page in *Schematic* view and select **Paste**. A graph frame may be pasted multiple times.

## Adjusting Frame Properties

To access the *Graph Frame Properties* dialog, **left double-click** the frame title bar, or **right-click** over the title bar and select **Edit Properties...**



The properties available in this dialog are as described below:

- **Caption:** Enter a title for the graph frame (this text will appear in the graph frame title bar). The default text may appear a bit cryptic: The  $\$(GROUP)$  syntax is used as a naming convention for grouping objects in the workspace. For more information on this syntax, see *Grouping of Runtime Objects*. The  $\$(INSTANCE)$  syntax may be used to display the module name and instance number for ease in transferring to reports.

Preferences:

- **Show Markers:** Select this option to show the *X* and *O* markers on all graphs.
- **Show Glyphs:** Select this option to show glyph symbols on all curves in the frame.
- **Show Ticks:** Select this option to show ticks along the y-intercept on all graphs.
- **Show Grid:** Select this option to show the grid on all graphs.
- **Show Y-Intercept:** Select this option to show the y-intercept in all graphs.



- **Show X-Intercept:** Select this option to show the x-intercept in all graphs.
- **Auto-Pan X-Axis:** This allows the user to adjust the panning action. The input field directly beside this check box accepts an input representing the percentage of the currently viewed graph window (or aperture). For example, if the total x-axis view is 0.1 seconds, a 10% auto-pan setting will pan the viewing window every 0.01 seconds.

## Adjusting Horizontal Axis Properties

To access the *Horizontal Axis Properties* dialog, double-click over the graph frame horizontal axis, or right-click over the horizontal axis and select **Axis Properties....**



**NOTE:** Adjusting the horizontal axis properties will affect all graphs in the frame.

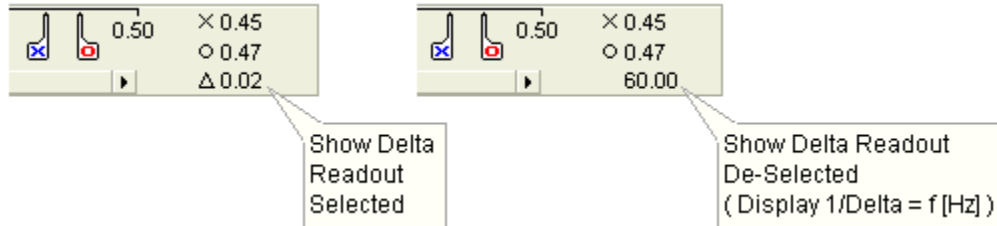
Axis:

- **Title:** Enter a title for the x-axis. This text will appear in the bottom-left corner of the frame, directly beside the x-axis.
- **Snap Aperture to Grid:** Select this feature so that when using dynamic aperture adjustment, the aperture view will snap to the major grid while scrolling.
- **Dynamic Aperture Adjustment:** Select this option to enable dynamic aperture adjustment (i.e. horizontal scroll).
- **Enable Minor Grids:** When selected, minor grid ticks will appear on the frame horizontal axis. Minor grids will always show the halfway point between major grid points, and are not labelled.
- **Max:** Sets the maximum time of the viewed range.
- **Min:** Sets the minimum time of the viewed range.

- **Grid:** Sets the time between the axis major grid points. Major grid points are labelled on the graph frame horizontal axis.

Markers:

- **Show Markers:** Select this option to show the X and O markers.
- **Show Delta Readout:** Select this option to display the time difference (i.e.  $\Delta t$ ) between the X and O markers. When this option is disabled, the equivalent  $1/\Delta t$  value (i.e. frequency in Hz) will be displayed.

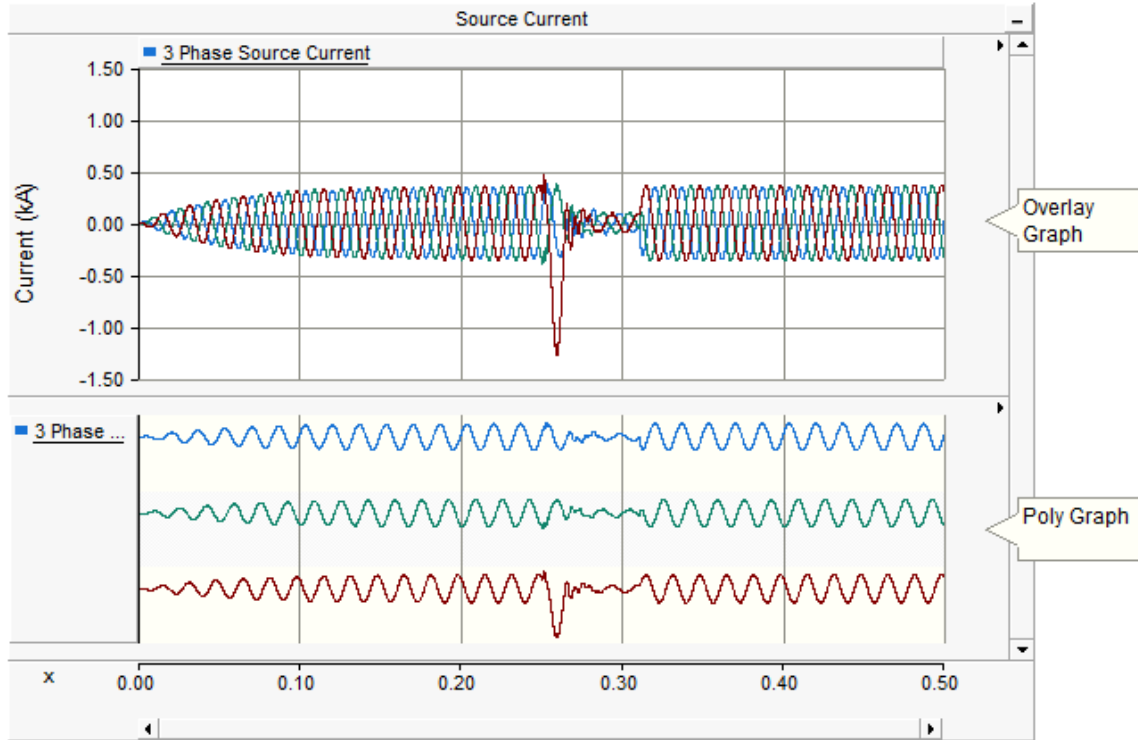


- **X Marker:** Enter the position (in seconds) to place X marker.
- **O Marker:** Enter the position (in seconds) to place O marker.
- **1/Delta:** Select this option to enter the frequency (i.e.  $1/\Delta t$ ) between markers.

## Graphs

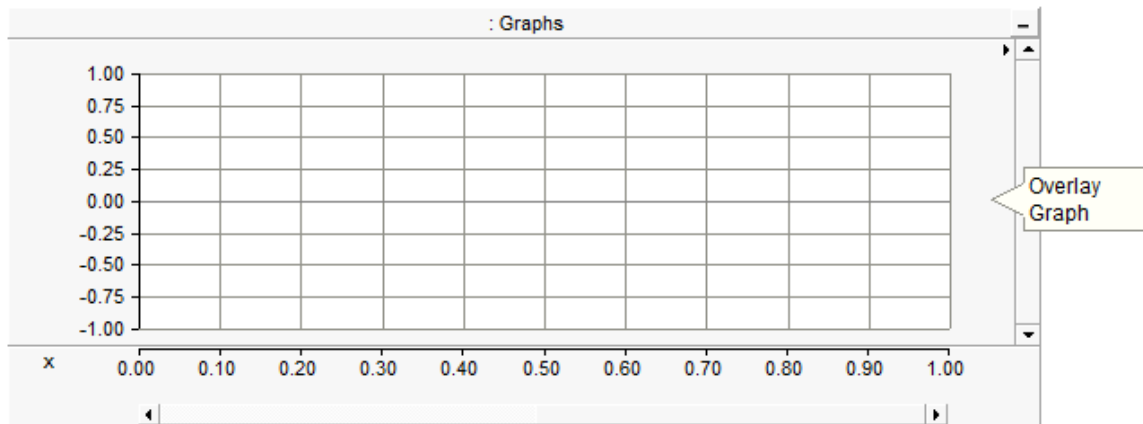
A *Graph* is a special runtime object, which can reside only inside a graph frame. There are two types of graphs available: Overlay and PolyGraphs. A single graph may hold and display multiple curves, where all curves in a graph are based on the same y-axis scale.

The following illustrates a graph frame with an overlay graph at the top and a polygraph below it.



## Adding Graphs to a Graph Frame

Graph frames may accommodate single or multiple graphs. To add one or more graphs, right-click on the frame title bar and select **Add Overlay Graph (Analog)** or **Add PolyGraph (Analog/Digital)**. You can also add an overlay graph directly by pressing the **Insert** key on your keyboard with your mouse pointer over the graph frame.



## Graph Order

Once multiple graphs have been added to a frame, you may change the order in which they appear. Right-click over the graph to be moved and select one of the following:

- **Move Graph Up**
- **Move Graph Down**

- **Move Graph to Top**
- **Move Graph to Bottom**

## Cut/Copy Graphs

Right-click over the graph to be cut (removed)/copied and select **Cut Graph/Copy Graph** respectively. Once a graph has been cut or copied it may then be pasted into the same or another graph frame.

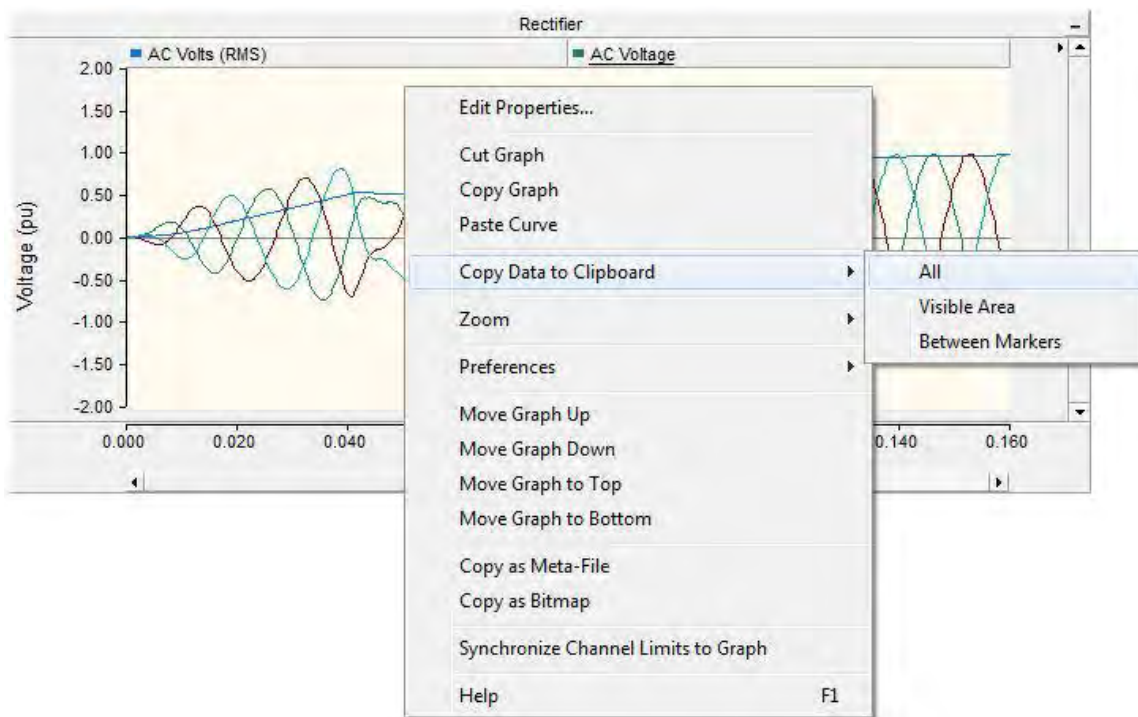
## Paste Graph

Cut or copy a graph as described above. Right-click on the graph frame title bar and select **Paste Graph**. A graph may be pasted multiple times, where each paste will replicate the entire graph.

## Copy Data to Clipboard

If a simulation has been run and your graph contains curve data, you have the option of copying all or a portion of all curve data to the clipboard. Right-click over the corresponding graph and select **Copy Data to Clipboard** and then select one of the following from the pop-up menu:

- **All:** Copies all curve data available.
- **Visible Area:** Copies all curve data visible in the graph window.
- **Between Markers:** Copies only curve data situated between markers. Note that **Show Markers** must be selected in the Axis Properties dialog.



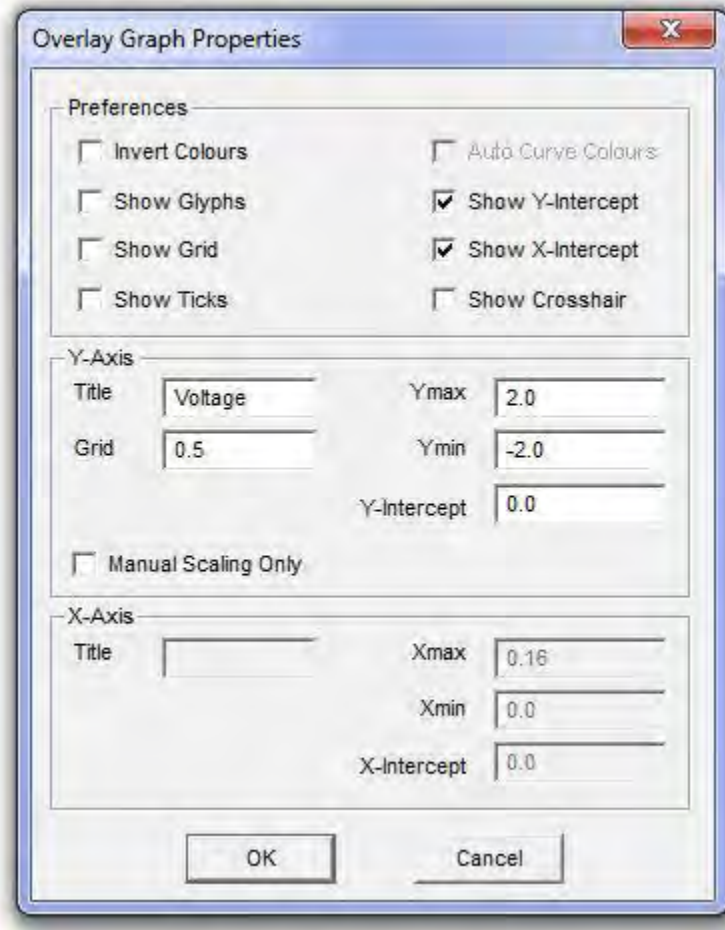
The data is copied as *Comma Separated Variables (\*.csv)* format for easy migration into common data analysis software.

## Overlay Graphs

Overlay graphs are the most common and familiar type of online plotting tool in PSCAD. These graphs display measured data as a function of time, where multiple curves may be added (or *overlaid* on top of each other) onto a single graph.

### Adjusting Overlay Graph Properties

Left double-click on the desired overlay graph, or right-click on the graph and select **Edit Properties...** This will bring up the *Overlay Graph Properties* dialog window as shown below:



There are various parameters that may be edited through this window, each of which are described below.

Preferences:

- **Invert Colours:** Select this option to give the graph a black background (instead of white or yellow).
- **Show Glyphs:** Select this option to show glyph symbols on all curves in the graph.
- **Show Grid:** Select this option to display grid lines for the x-axis and y-axis major grids.
- **Show Ticks:** Select this option to show major grid tick marks along the y-axis intercept line.

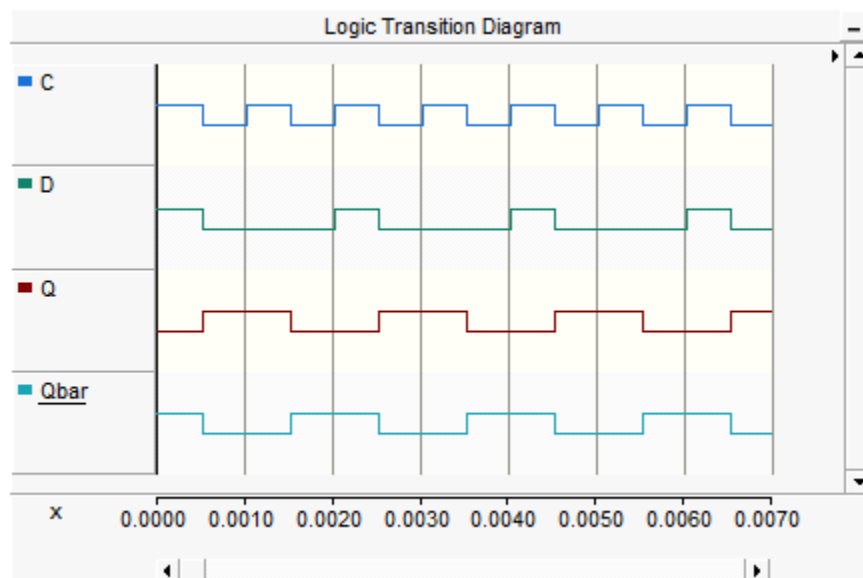
- **Auto Curve Colours:** Select this option to use automatic colouring of curves in the graph. You cannot change curve colour manually when this option is selected.
- **Show Y-Intercept:** Select this option to display the y-intercept (horizontal) intercept line. The y-intercept line can be adjusted using the *Y-Intercept* field described below.
- **Show X-Intercept:** Select this option to display the x-intercept (vertical) intercept line. The x-intercept is always at time zero, and cannot be adjusted in overlay graphs.
- **Show Cross Hair:** Select this option to invoke the cross hairs mode.

Y-Axis:

- **Title:** Enter text for display as the graph title (located on the left side of the graph).
- **Grid:** Specifies the y-axis grid interval. To view the y-axis grid lines, select the option **Show Grid** described above.
- **Ymin:** Specifies the minimum y-axis viewing limit on the graph.
- **Ymax:** the maximum y-axis viewing limit on the graph.
- **Y-Intercept:** Specifies the y-axis location of the y-intercept line. This line is only visible if **Show Y-Intercept** is selected (see above).
- **Manual Scaling Only:** Select this feature to lock the y-axis limits set in **Ymin** and **Ymax** above. The y-axis will remain locked during any subsequent zoom operations.

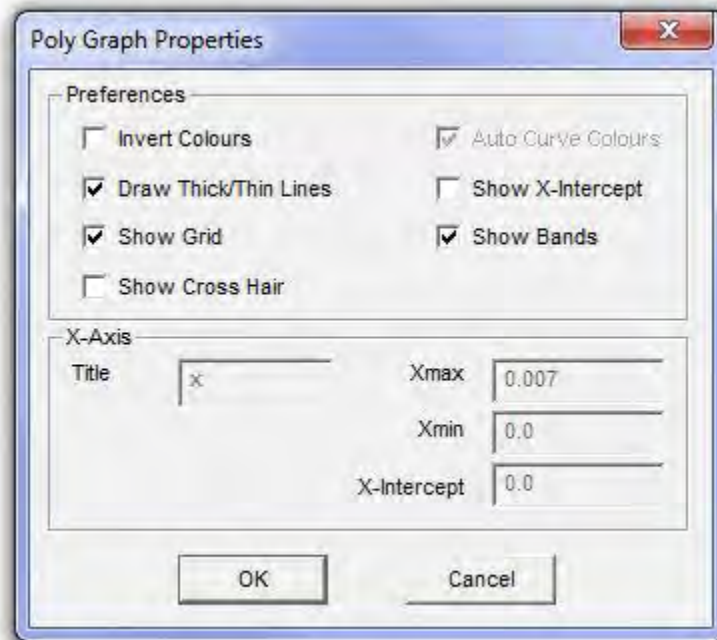
## PolyGraphs

*Polygraphs* are used specifically to display plotted curves in a 'stacked' format. That is, each curve is contained within its own viewing space, and is stacked one atop the other. A polygraph may be chosen over a standard overlay graph if the user needs to either view many single-curve plots in a compact space, or to make use of the curve digital style functions to create a logic transition diagram:



## Adjusting PolyGraph Properties

Left double-click on the desired polygraph, or right-click on the graph and select **Edit Properties....** This should bring up the *Poly Graph Properties* dialog:



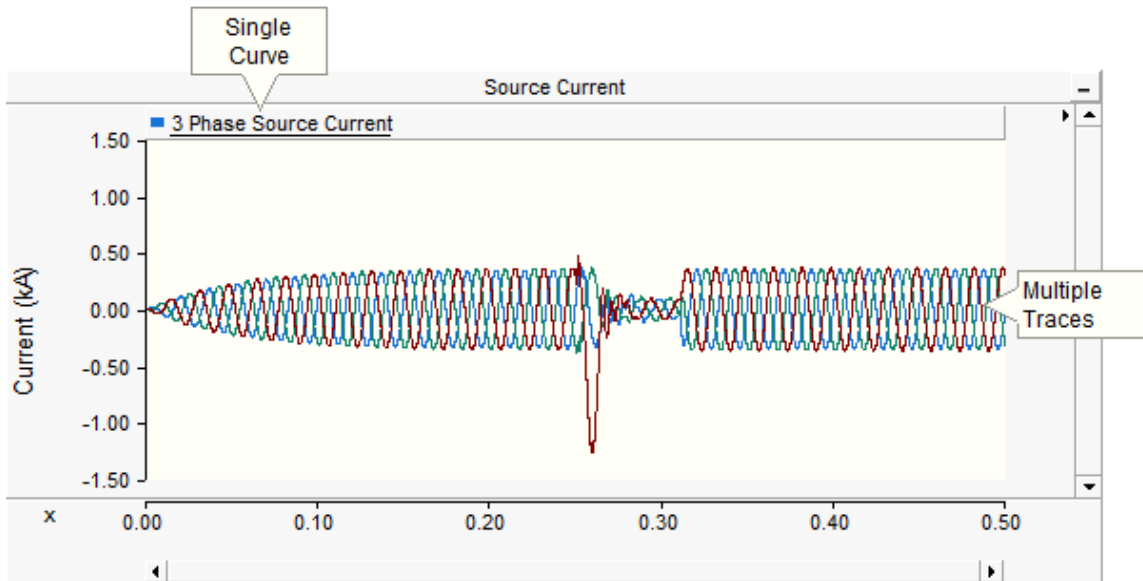
There are various parameters that may be edited through this window, each of which are described below.

Preferences:

- **Invert Colours:** Select this option to give the graph a black background (instead of white or yellow).
- **Show Grid:** Select this option to display grid lines for the x-axis and y-axis major grids.
- **Show Cross Hair:** Select this option to invoke the cross hairs mode.
- **Auto Curve Colours:** Select this option to use automatic colouring of curves in the graph. You cannot change curve colour manually when this option is selected.
- **Show X-Intercept:** Select this option to display the x-intercept (vertical) intercept line. The x-intercept is always at time zero, and cannot be adjusted in polygraphs.
- **Show Bands:** Selecting this option will give a different background colour between multiple curves in one graph, for easy visual differentiation.

## Curves

A *curve* is special runtime object best described as a graphical representation of a string of data points, where each point is associated with a simulation plot step. Curves are created by linking to an Output Channel component, to which a scalar or array set of data signals have been input. As such, curves can be multi-dimensional; that is a single curve may possess many sub-curves or Traces, where each trace corresponds to a single array value.

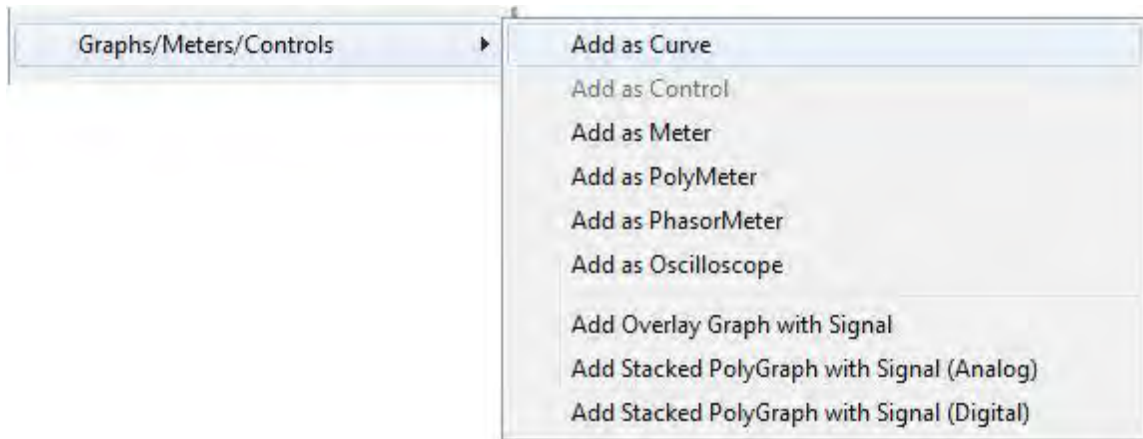


If the *Output Channel* input signal is a scalar (i.e. single dimension), then the curve will consist of a single trace. For more information on accessing and adjusting individual trace properties, see [Traces](#).

## Adding a New Curve to a Graph

Adding a curve to a graph can be accomplished a couple of different ways:

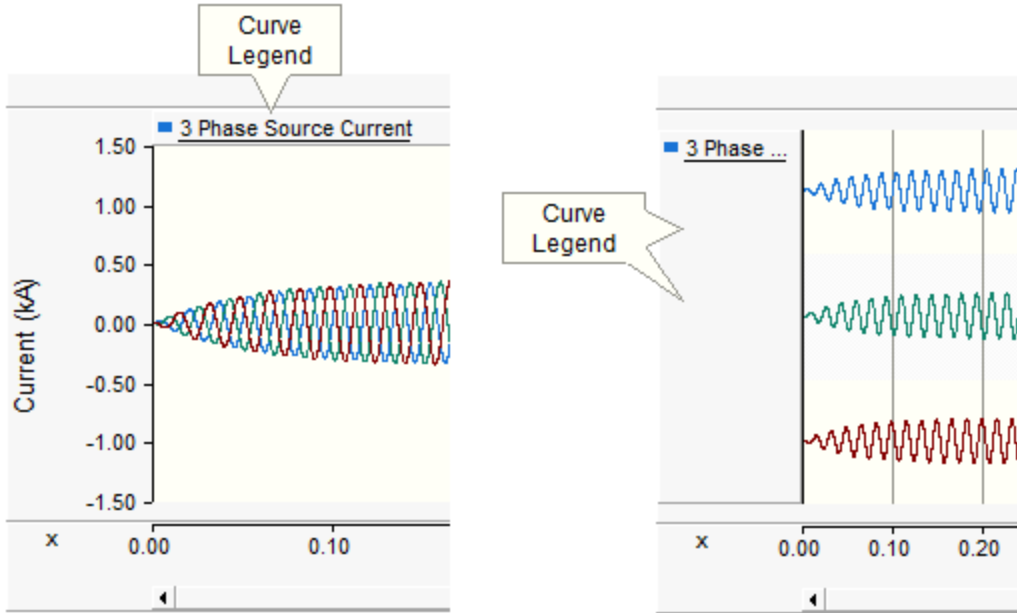
1. **Drag and Drop Method:** Hold down the **Ctrl** key. **Left-click and hold** over the output channel component from which you would like to extract the curve. **Drag** the mouse pointer over a graph and release the mouse button. See [Drag and Drop](#) for more details on this.
2. **Graphs/Meters/Controls Method:** **Right-click** on the output channel component from which you would like to extract the curve. Select **Graphs/Meters/Controls | Add as Curve**. Select the desired graph with a left-click, then right-click and select **Paste Curve**.



## Curve Legends

Once a curve has been added to a graph, the curve title will appear in the curve legend.

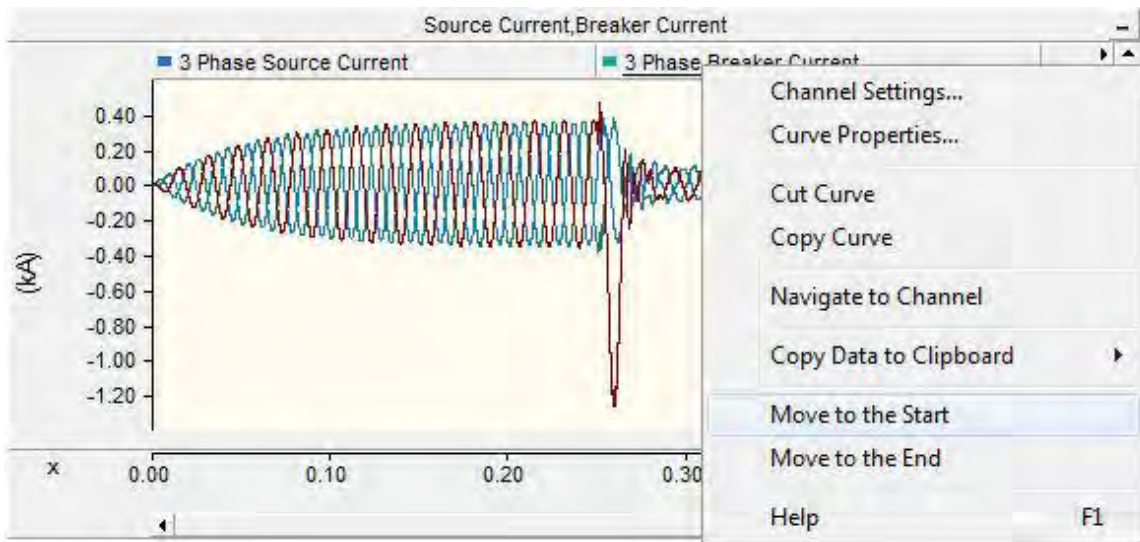




## Curve Order

Once multiple curves have been added to a graph, you may change the order in which they appear. Ordering curves can be accomplished in one of two ways:

1. **Drag and Drop:** **Left-click and hold** over the curve in the curve legend. Drag the mouse pointer to a new position in the curve legend and release the mouse button. See Drag and Drop for more details on this.
2. **Right-Click Menu:** **Right-click** over the corresponding curve legend and select one of the following from the pop-up menu: **Move to the Start** or **Move to the End**.



## Cut/Copy/Paste an Existing Curve

**Right-click** over the curve title and select either **Cut Curve** or **Copy Curve**, depending on what you want to do. **Right-click** over any graph and then select **Paste Curve** to paste the curve. The curve should then appear in the corresponding curve legend.

## Copy Data to Clipboard

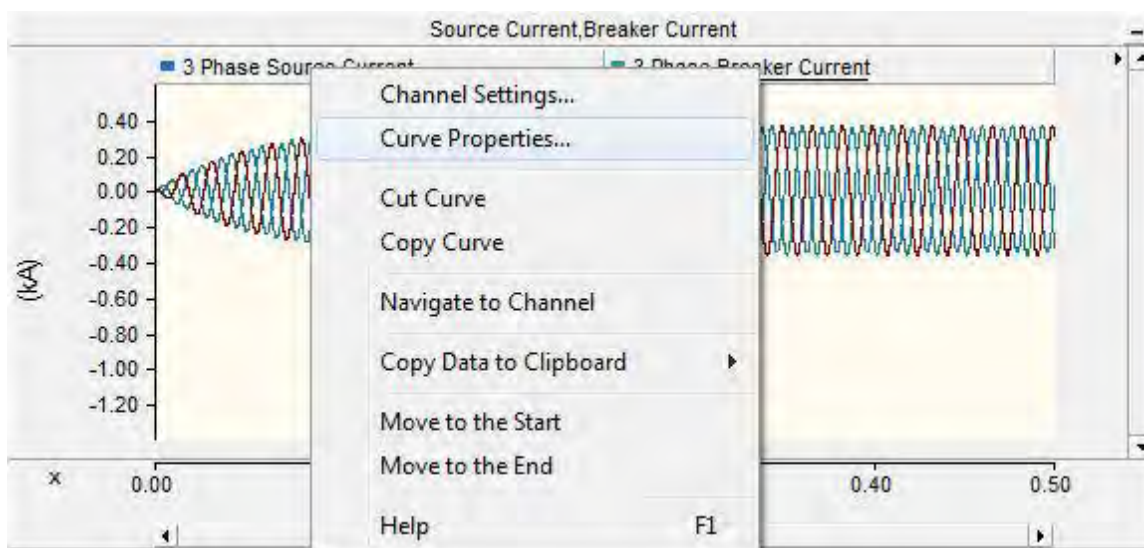
If a simulation has been run and a particular curve contains data, you have the option of copying all or a portion of this single curve data set to the clipboard. **Right-click** over the corresponding curve name, select **Copy Data to Clipboard** and then select one of the following from the pop-up menu:

- **All:** Copies all curve data available
- **Visible Data:** Copies only curve data that is visible in the graph window.
- **Between Markers:** Copies only curve data situated between markers. Note that **Show Markers** must be selected in the Axis Properties dialog.

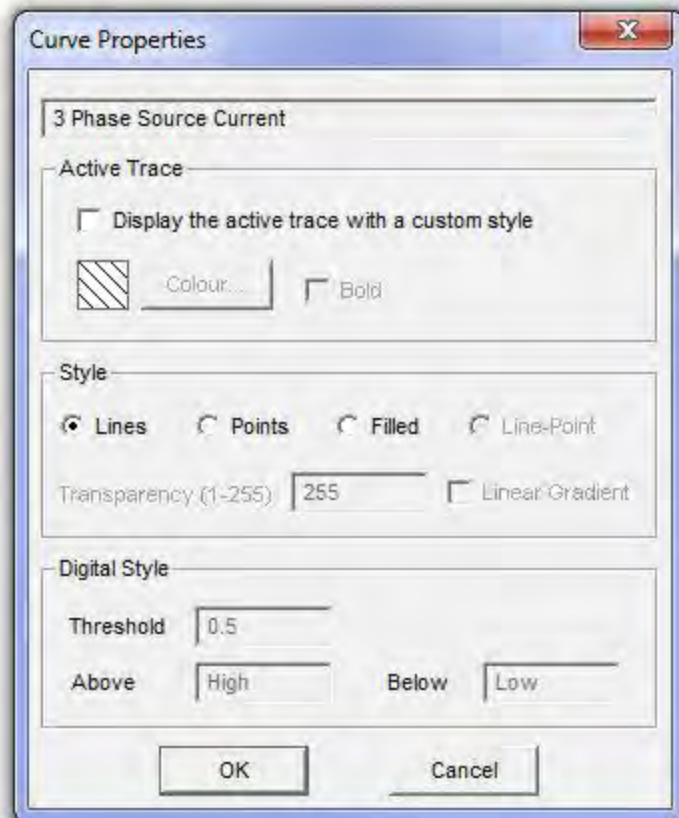
The data is copied as *Comma Separated Variables (\*.csv)* format for easy migration into common data analysis software.

## Adjusting Curve Properties

**Left double-click** on the desired curve in the curve legend, or right-click the curve and select **Curve Properties....**



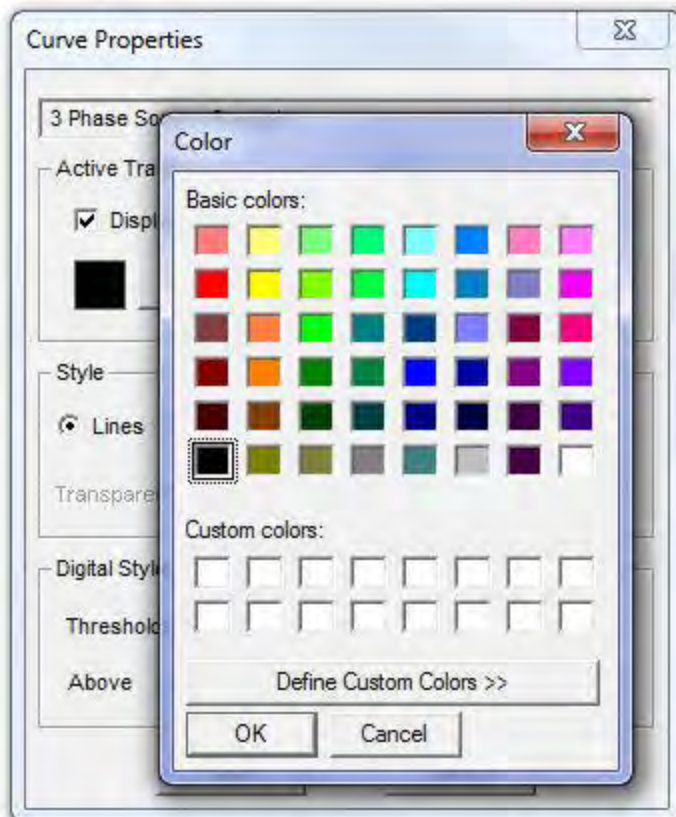
This will bring up the *Curve Properties* dialog window.



## Active Trace

The following list describes the parameters in this section:

- **Display the active trace with a custom style:** Select this option if you wish to change the colour or width of the active trace.
- **Colour:** Press the **Colour...** button to select a display colour for the trace. Press the OK button in the colour dialog. This option is enabled only if **Display the active trace with a custom style** is selected.



- **Bold:** Select this option if you wish the trace to appear bold. This option is enabled only if **Display active trace with a custom style** is selected.

## Style

The following list describes the parameters in this section:

- **Lines:** Displays the curve as a standard line.
- **Points:** Displays the curve as a series of points according to the set plot step.
- **Filled:** Fills the area under the curve (between the curve and the 0.0 line) with the curve colour.
- **Transparency (1-255):** Sets the transparency level of the filled portion under a curve. This is adjustable only when **Filled** is enabled.
- **Linear Gradient:** Select this option to create a linear gradient effect on the filled part under the curve. This is adjustable only when **Filled** is enabled.

## Digital Style

These options are only considered if the curve is part of a polygraph. Digital style controls the properties the curve traces when they are in digital mode. The following list describes the parameters in this section:

- **Threshold:** The threshold value at which to change the display state of the curve.
- **Above / Below:** Enter the state for the curve when value is above and below the set threshold respectively.

## Adjusting Channel Settings

The source Output Channel parameters for a particular curve can be accessed directly from the curve legend pop-up menu. Right-click the curve and select **Channel Settings...**

## Synchronizing Output Channel Limits with Those of the Graph

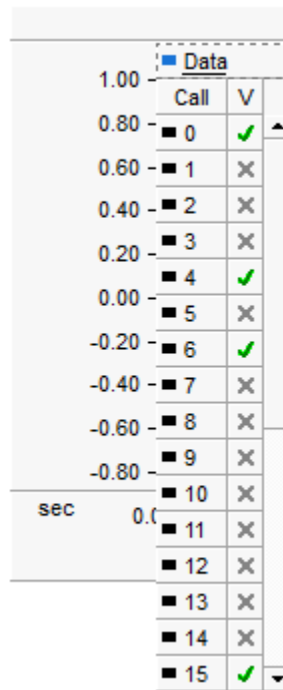
Once one or more curves have been added to an overlay graph, all corresponding Output Channel component **Min / Max** limits can be synchronized to the graph y-axis minimum and maximum limits.

To synchronize output channel limits with the graph, right-click over the overlay graph and select **Synchronize Channel Limits to Graph**.

## Curves Sourced from Multiple Instance Modules

Module components can possess more than one instance. This means that if an Output Channel exists on the canvas of a module (i.e. part of the module definition), then it will produce a unique curve for each instance of the module, even though there is technically only a single *Output Channel* component. If the curve associated with the *Output Channel* is placed in a graph that exists outside the module, which unique curve instance is displayed?

Each curve instance is referred to as a *call*. If a curve exists in a graph that is sourced from an output channel inside a multiple instance module, then you can access and control the view of any number of the unique calls. Simply hold down the **Ctrl** key and **left-click** on the curve legend. A display list will appear showing all source curves (calls) available.



In the above image, it is apparent that the curve *Data* is sourced from 16 different module instances that are based on the same definition. In this particular example, call 0 (base 0) and call 4, 6 and 15 are being displayed. You may choose to display all *calls* in a single graph, or you can place the *Data* curve in multiple graphs, and select a unique *call* to display in each graph.

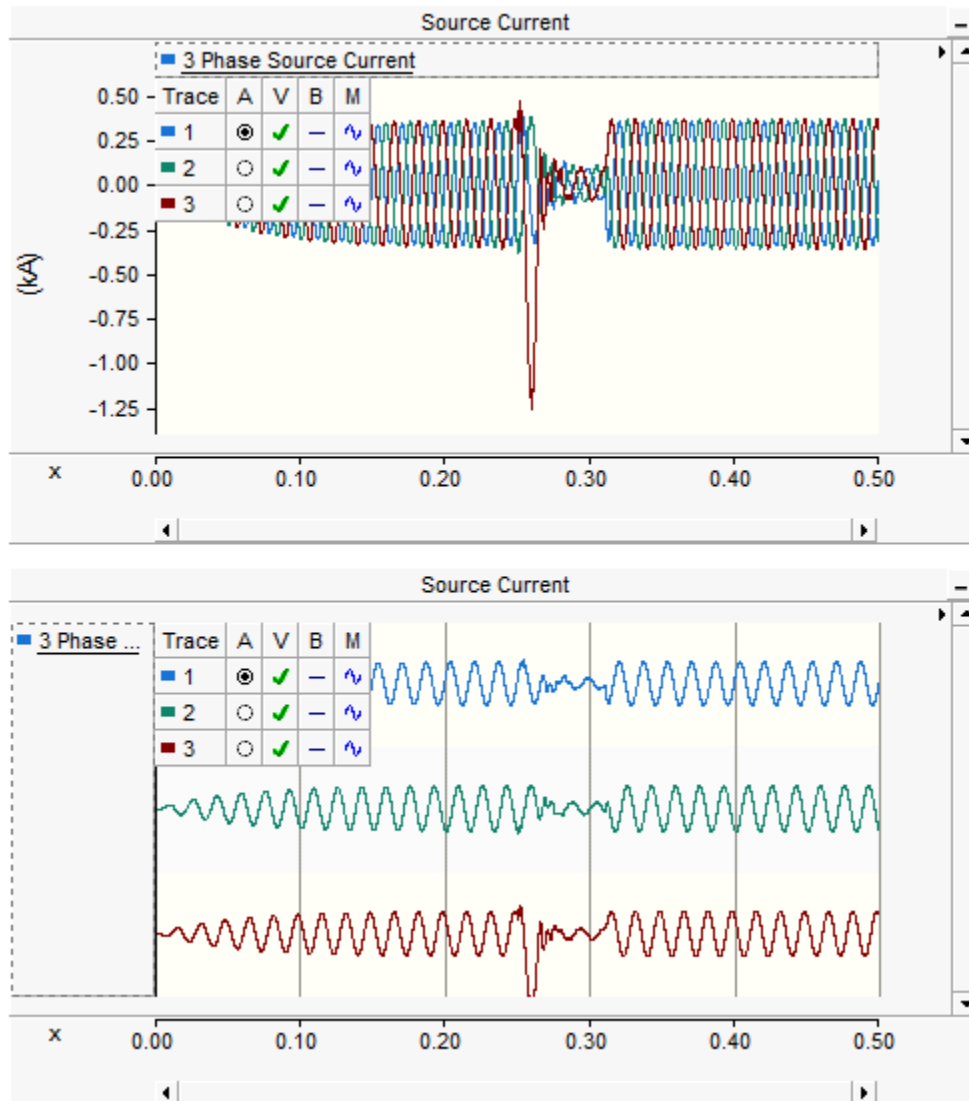
For more information on multiple instance modules, see Multiple Instance Modules (MIM).

# Traces

Array signal curves can be plotted online as a single entity, where each array element or 'sub-curve' is referred to as a *Trace*. Each trace may be enabled or disabled separately (i.e. shown or hidden).

## Trace Drop Down Menu

Trace properties and control can be accessed through a special drop down menu with a single left-click on the curve title in the curve legend.



## Adjusting Trace Properties

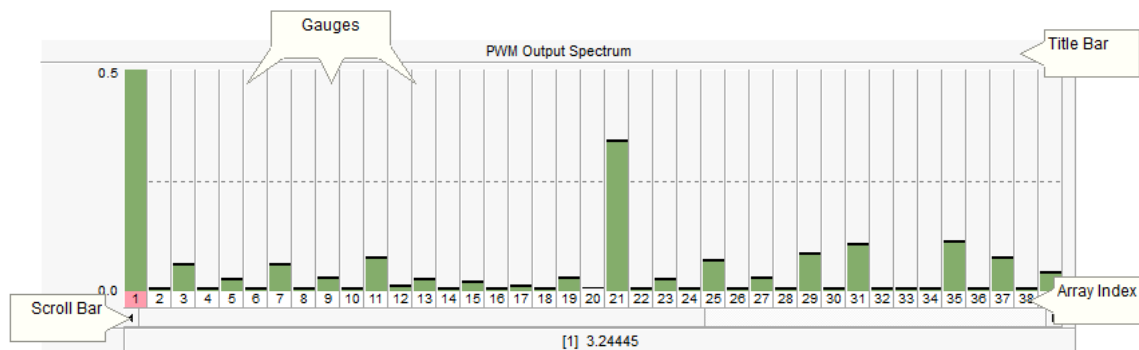
Before adjusting any trace properties, you must first invoke the trace drop-down menu as described above. The drop menu consists of four separate columns, each allow for easy access to certain trace properties. These columns are described below:

Trace	A	V	B	M
1	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	<input type="radio"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	<input type="radio"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

- **Trace:** Indicates the trace number and corresponding color setting. The number corresponds to the array index number of the multi-signal curve. To change individual trace color, see Adjusting Curve Properties.
- **A:** Stands for *Active*. Select the radio button in this column to select the active trace. The active trace will be the default focus when switching to cross-hair mode. Also, only the properties of the active trace may be adjusted: See Adjusting Curve Properties.
- **V:** Stands for *View*. Left-click the individual check boxes in this column to hide/view individual traces. You can hide/view all traces by left-clicking on the V itself.
- **B:** Stands for *Bold*. Left-click the individual check boxes in this column to bold/un-bold individual traces. You can bold/un-bold all traces by left-clicking on the B itself.
- **M:** Stands for *Mode*. This function is only valid when the curve is displayed in a polygraph. Left-click the individual boxes in this column to change the trace mode from digital to analog. When in digital mode, the trace will be displayed in a special two-state format, where the state depends on whether it is above or below a preset **Threshold** value. For more details on setting threshold and other digital properties see PolyGraphs.

## PolyMeters

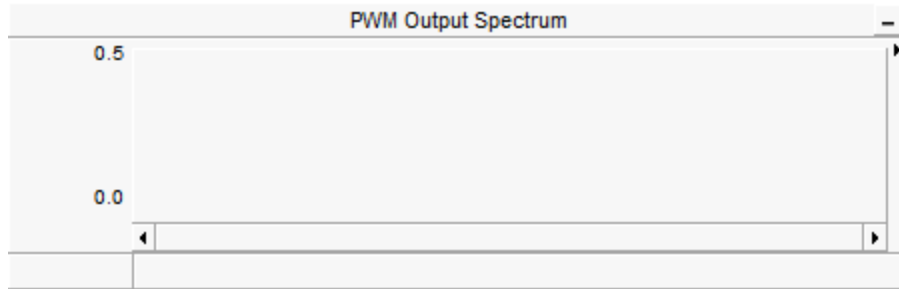
A *polymer* is a special runtime object used specifically for monitoring a single, multiple-trace curve. The polymer dynamically displays the magnitude of each trace in bar type format (called gauges), which results in an overall appearance similar to a spectrum analyzer. The power of this device lies in its ability to compress a large amount of data into a small viewing area, which is particularly helpful when viewing harmonic spectrums such as data output from the Fast Fourier Transform (FFT) component.



The relative gauge width is fixed, and so if the poly meter is not wide enough to display all data, a simple horizontal scroll bar is provided. An array Index display is included directly under the gauges for easy identification.

Polymeters are special objects that cannot be added directly from the tool bar. Each polymer is linked with a single curve from a single Output Channel component.

**NOTE:** Polymeters will appear as a blank container (as shown below) until the project is compiled and run.



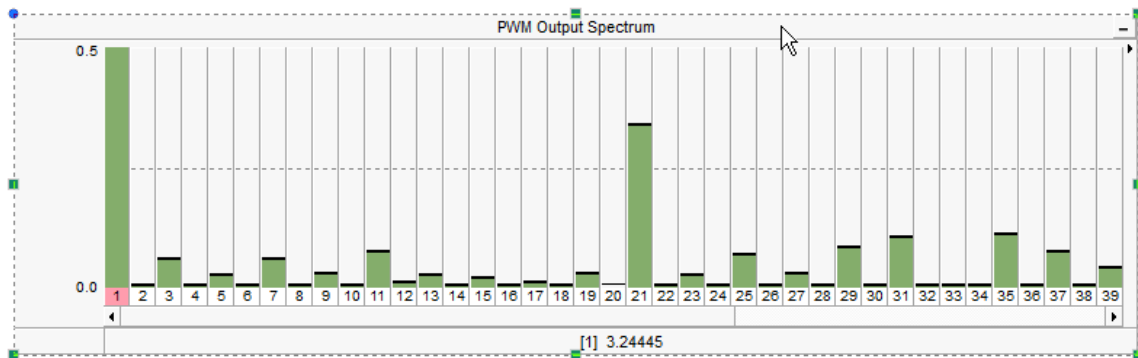
## Adding a PolyMeter

To add a polymer, **right-click** on an Output Channel component within the *Schematic* canvas and select **Graphs/Meters/Controls | Add as PolyMeter**. The polymer will appear attached to the mouse pointer. Move the pointer to wherever you wish the new meter to reside and then **left-click** to place it on the *Schematic* canvas.

## Moving and Re-sizing a PolyMeter

To move a polymer, move the mouse pointer over the title bar and then **left-click and hold**. Drag the meter to where it is to be placed and release the mouse button.

To re-size, move the mouse pointer over the title bar and **left-click** to select the polymer. Grips should then appear around the outer edge as shown below.

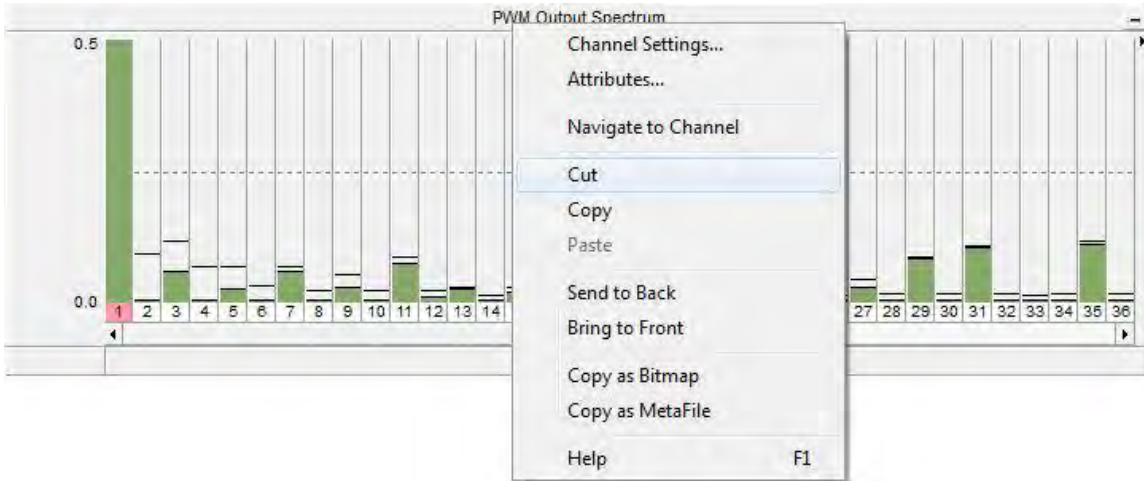


The polymer will appear attached to the mouse pointer. Move the pointer to wherever you wish the new meter to reside and then **left-click** to place it on the *Schematic* canvas.

## Cut/Copy PolyMeter

Right-click over the polymer title bar and select **Cut** or **Copy** respectively.





Once a polymer has been cut or copied it may then be pasted to another any *Schematic* canvas in the project.

## Paste PolyMeter

Cut or copy a polymer as described above. Right-click over a blank area in *Schematic* view and select **Paste**. A polymer may be pasted multiple times.

## Copy PolyMeter as Meta-File/Bitmap

The entire polymer display can be copied to the Windows clipboard in either *meta-file* (\*.wmf) or *bitmap* (\*.bmp) format. **Right-click** over the polymer title bar and select **Copy as Bitmap** or **Copy as Meta-File**. Go to your report document and paste the image.

## Navigate to Channel

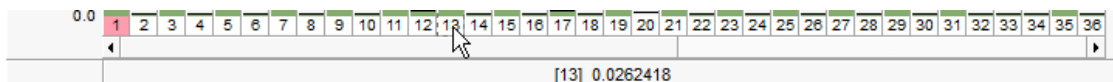
You can navigate directly to the associated Output Channel component by selecting this option. Right-click over the polymer and select **Navigate to Channel**. PSCAD will automatically find the output channel and highlight it.

## Adjusting Channel Settings

The y-axis properties of the polymer are set in the corresponding Output Channel properties dialog. This dialog may be accessed directly from the polymer by right-clicking over the title bar and selecting **Channel properties...**

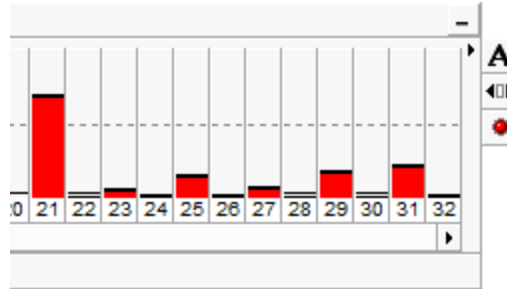
## Displaying Specific Data

The magnitude of individual array elements can be displayed in the status bar at the bottom of the polymer. To view the magnitude of that element, left-click on a particular index number in the array index.



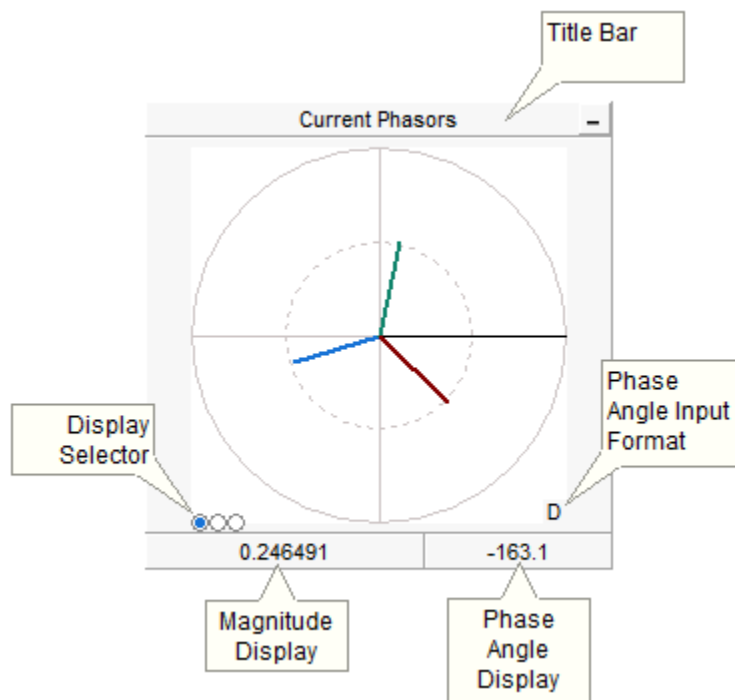
## Change Gauge Colour

You can modify the gauge colour of any polymer by simply clicking the panel legend and selecting **Colour**:



## PhasorMeters

A *PhasorMeter* is a special runtime object that can be used to display up to six, separate phasor quantities. The phasormeter displays phasors in a polar graph, where the magnitude and phase of each phasor responds dynamically during a simulation run. This device is perfect for visually representing phasor quantities, such as output from the Fast Fourier Transform (FFT) component.



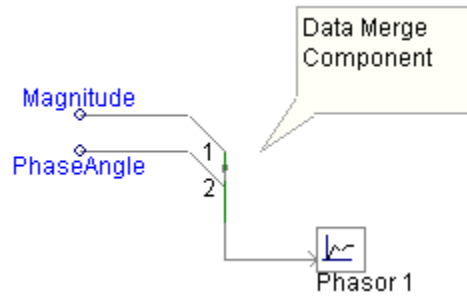
PhasorMeters are special objects that cannot be added directly from the tool bar. Each phasormeter is directly linked with a single curve from a single Output Channel component. In the case of the phasormeter, a curve with at least two traces (magnitude and phase angle) is the minimum requirement for the device to work properly. The following section describes how to prepare data signals for display in the phasormeter.

### Preparing Data for Display

A single phasor quantity, as it pertains to the phasormeter, is composed of a separate magnitude signal, along with an associated phase angle signal, which combined represents a single phasor quantity in polar format. That is:

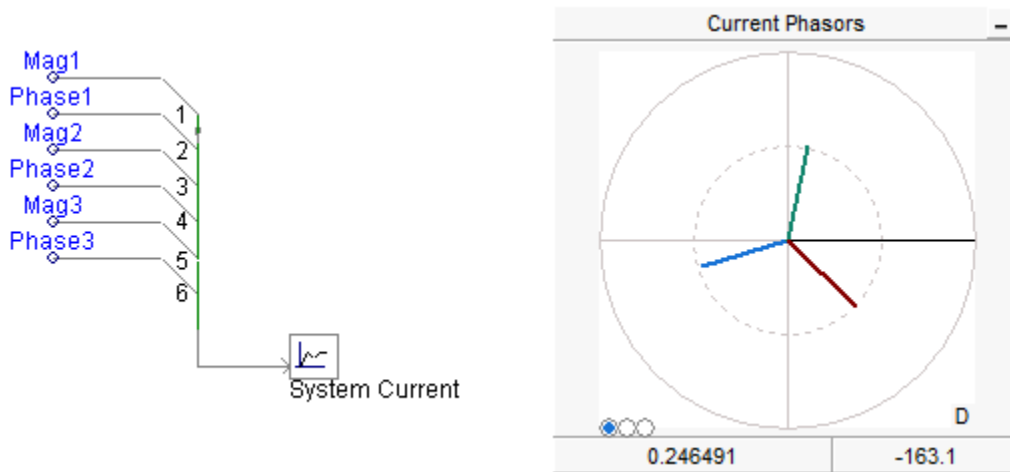
$X \angle \phi$

To display a single phasor quantity, the user must construct an array data signal of dimension two, using the Data Merge component. The two data signals will represent the magnitude and phase angle respectively. To this array signal, an Output Channel is attached as shown below:

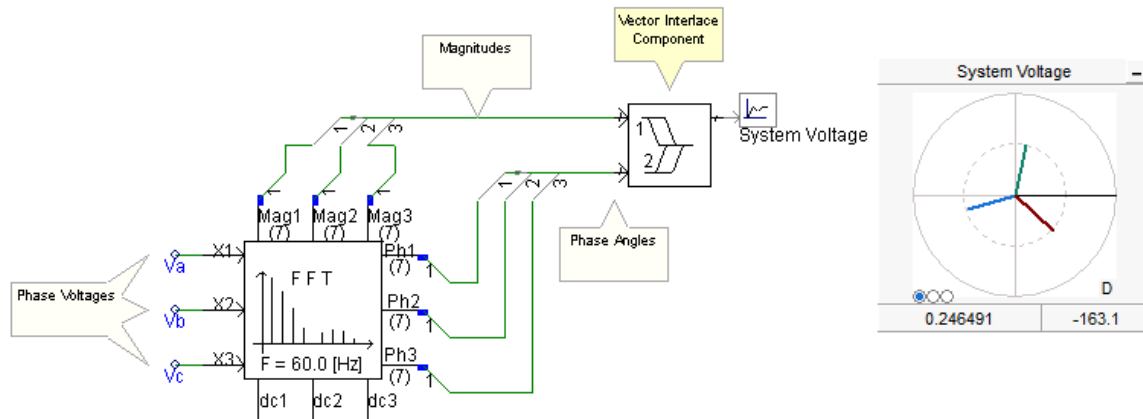


The phasor meter always assumes that the elements of the array are in the above displayed order. That is element 1 is the magnitude and element 2 is the phase angle.

The phasometer allows up to a maximum of six phasors per display, which corresponds to an input array signal of dimension 12. In cases of multiple phasors, the order of each magnitude/phase angle group must appear in the same order as the first (i.e. magnitude, phase angle, magnitude, phase angle, etc.) as shown below for a three phasor display:



More often than not, the Fast Fourier Transform (FFT) component will be used to derive the polar quantities for display in the phasometer. If so, a special component, called the Vector Interlace component was developed to make it easier to prepare data for display. The image below illustrates a situation where it could be used:



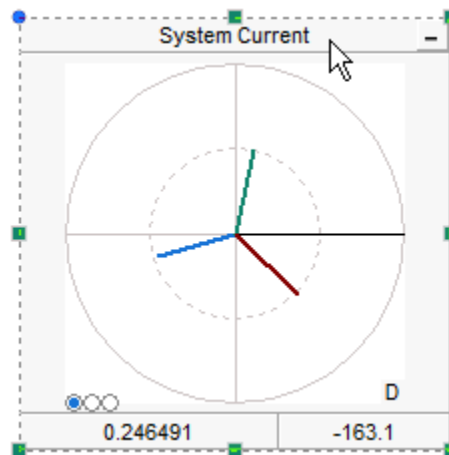
## Adding a PhasorMeter

To add a phasometer, right-click on an Output Channel component within the *Schematic* canvas and select **Graphs/Meters/Controls | Add as PhasorMeter**. The phasometer will appear attached to the mouse pointer. Move the pointer to wherever you wish the new meter to reside and then left-click to place it on the *Schematic* canvas.

## Moving and Re-sizing a PhasorMeter

To move a phasometer, move the mouse pointer over the title bar and then **left-click and hold**. Drag the meter to where it is to be placed and release the mouse button.

To re-size, move the mouse pointer over the title bar and **left-click** to select the meter. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. **Left-click, hold** and then drag the pointer to re-size.

## Cut/Copy PhasorMeter

Right-click over the phasometer and select **Cut** or **Copy** respectively. Once a phasometer has been cut or copied it may then be pasted to another location in the project.

## Paste PhasorMeter

Cut or copy a phasormeter as described above. Right-click over a blank area of the project page in *Schematic* view and select **Paste**. A phasormeter may be pasted multiple times.

## Copy PhasorMeter as Meta-File/Bitmap

The entire phasormeter display can be copied to the Windows clipboard in either *meta-file (\*.wmf)* or *bitmap (\*.bmp)* format. Right-click over the phasormeter title bar and select **Copy as Bitmap** or **Copy as Meta-File**. Go to your report document and paste the image.

## Navigate to Channel

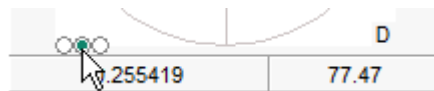
You can navigate directly to the associated Output Channel component by selecting this option. Right-click over the phasormeter and select **Navigate to Channel**: PSCAD will automatically find the output channel and highlight it.

## Adjusting Channel Settings

The polar axis limits of the phasormeter are set in the corresponding Output Channel properties dialog. This dialog may be accessed directly from the phasor meter by right-clicking over the device title bar and selecting **Channel properties....**

## Displaying Specific Data

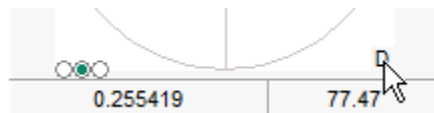
The magnitude and phase angle of individual array elements can be displayed in the status bar at the bottom of the phasormeter: Left-click the corresponding button on the display selector, where the buttons represent phasors 1 through 6 from left to right.



## Adjusting Phase Angle Input Format

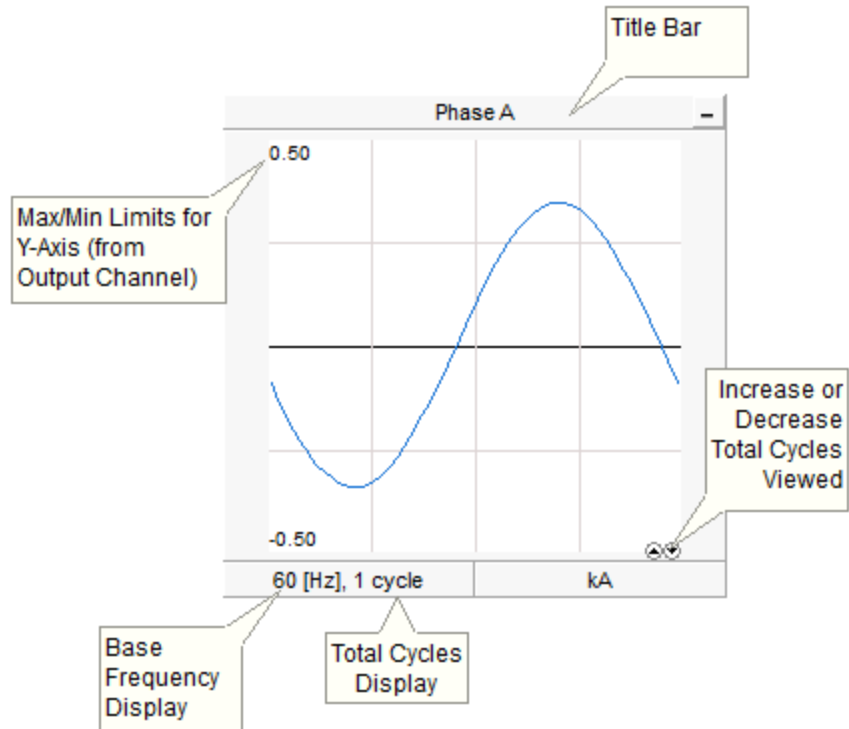
The format that the incoming phase angle data is in must be specified. This is easily accomplished by toggling the D or R display in the bottom-right corner of the graph, where:

- **D**: Degrees
- **R**: Radians



## Oscilloscopes

An *Oscilloscope* is a special runtime object that is used to mimic the triggering effects of a real-world oscilloscope on a time varying, cyclical signal like an AC voltage or current. Given a base frequency, the oscilloscope will follow the signal during a simulation (like a moving window), refreshing its display at the rate given by the base frequency. This gives the illusion that the oscilloscope is transfixed on the signals being displayed, resulting in a triggering effect.



Oscilloscopes are special objects that cannot be added directly from the tool bar. Each object is directly linked with a curve from a single Output Channel component. The oscilloscope supports array signals – that is curves containing more than a single trace. The following section describes how to prepare data signals for display in the oscilloscope.

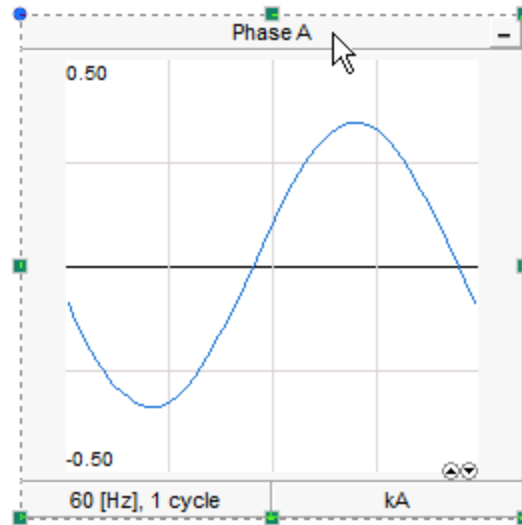
## Adding an Oscilloscope

To add an oscilloscope, right-click on an Output Channel component within the *Schematic* canvas and select **Graphs/Meters/Controls | Add as Oscilloscope**. The oscilloscope will appear attached to the mouse pointer. Move the pointer to wherever you wish the new meter to reside and then left-click to place it on the *Schematic* canvas.

## Moving and Re-sizing an Oscilloscope

To move an oscilloscope, move the mouse pointer over the title bar and then **left-click and hold**. Drag the meter to where it is to be placed and release the mouse button.

To re-size, move the mouse pointer over the title bar and left-click to select the meter. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. **Left-click, hold** and drag the pointer to re-size.

## Cut/Copy Oscilloscope

Right-click over the oscilloscope and select **Cut** or **Copy** respectively. Once an oscilloscope has been cut or copied it may then be pasted to another location in the project.

## Paste Oscilloscope

Cut or copy an oscilloscope as described above. Right-click over a blank area of the project page in Circuit view and select **Paste**. A scope may be pasted multiple times.

## Copy Oscilloscope as Meta-File or Bitmap

The entire oscilloscope display can be copied to the Windows clipboard in either *meta-file (\*.wmf)* or *bitmap (\*.bmp)* format. Right-click over the scope title bar and select **Copy as Bitmap** or **Copy as Meta-File**. Go to your report document and paste the image.

## Navigate to Channel

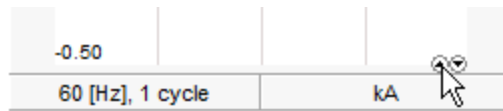
You can navigate directly to the associated Output Channel component by selecting this option. Right-click over the oscilloscope and select **Navigate to Channel**: PSCAD will automatically find the output channel and highlight it.

## Adjusting Channel Settings

The y-axis limits of the oscilloscope are set in the corresponding Output Channel Properties dialog. This dialog may be accessed directly from the scope by right-clicking over the device title bar and selecting **Channel Settings...**

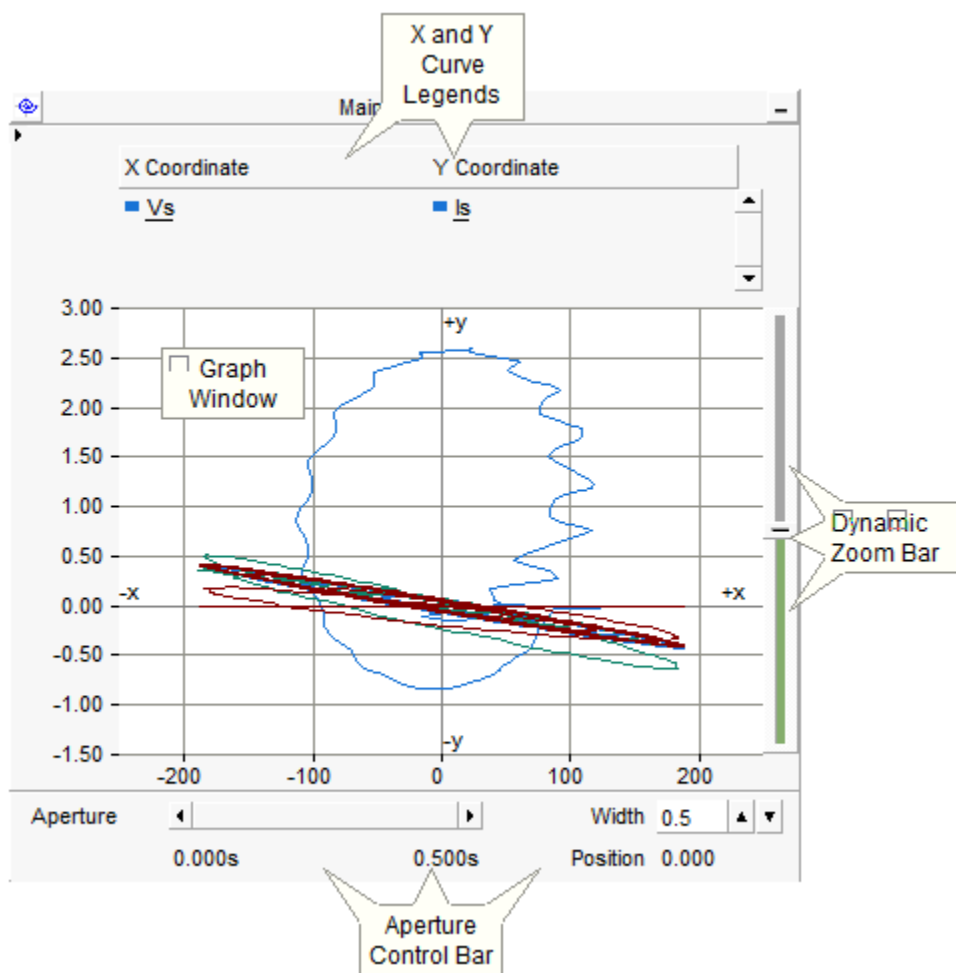
## Increase/Decrease the Total Displayed Cycles

The total displayed cycles can be adjusted in the status bar near the bottom of the oscilloscope: Left-click the corresponding button on the display selector, where the buttons represent increase and decrease respectively. This same operation can be performed by right-clicking over the device title bar and selecting either **Increase One Cycle** or **Decrease One Cycle**.



## XY Plots

The XY Plot is comprised of both a graph frame and a single, specialized graph window for the purpose of plotting one curve versus another. An xy plot can accommodate multiple curves on each of the x and y-axes, and includes dynamic zoom and polar grid features.

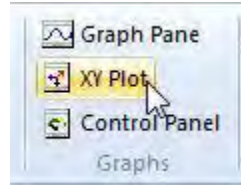


Although the xy plot is used to plot one signal versus another, each of these signals is based on the same time scale. It is therefore possible to scroll through the data in the time domain: The xy plot includes a time domain aperture control bar, located at the bottom of the plot frame.



## Adding an XY Plot

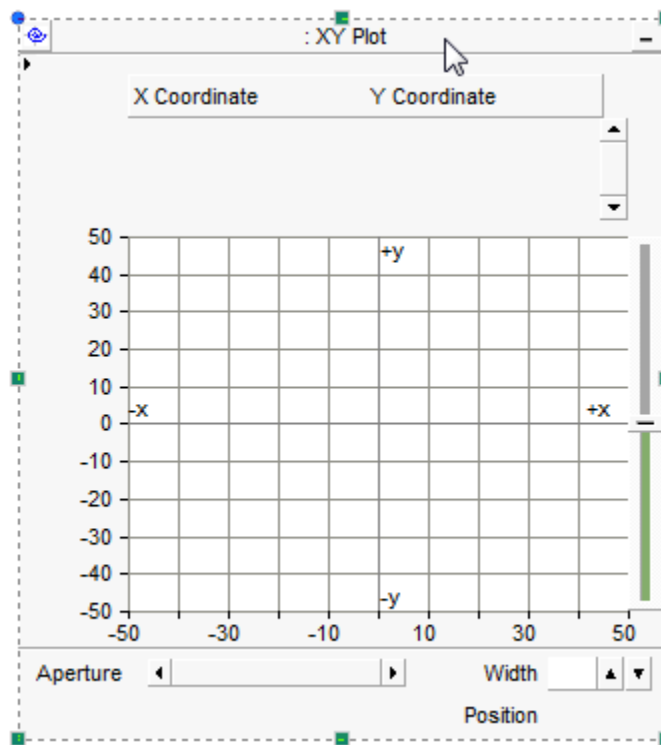
Open the project in the Circuit view. Right-click on a blank portion of the page and select **Add Component | XY Plot**, or press the **XY Plot** button in the **Components** tab of the ribbon control bar.



## Moving and Re-sizing an XY Plot Frame

To move an xy plot, place the mouse pointer over the title bar and then **left-click and hold**. Drag the plot frame to where it is to be placed and release the mouse button.

To re-size the frame, move the mouse pointer over the title bar and left-click to select it. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. **Left-click, hold** and drag the pointer to re-size.

## Cut/Copy XY Plot Frame

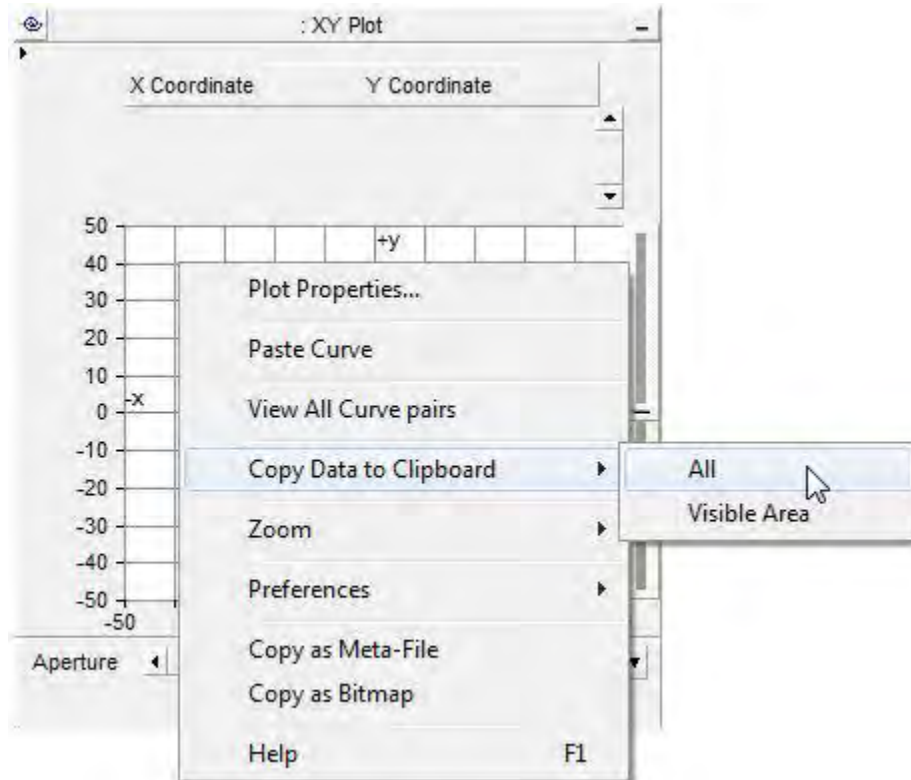
Right-click over the xy plot title bar and select **Cut** or **Copy** respectively.

## Paste XY Plot Frame

Cut or copy an xy plot as described above. Right-click over a blank area of the Circuit canvas and select **Paste**. An xy plot may be pasted multiple times.

## Copy Data to Clipboard

If a simulation has been run and your xy plot contains curve data, you have the option of copying all or a portion of this data to the clipboard.



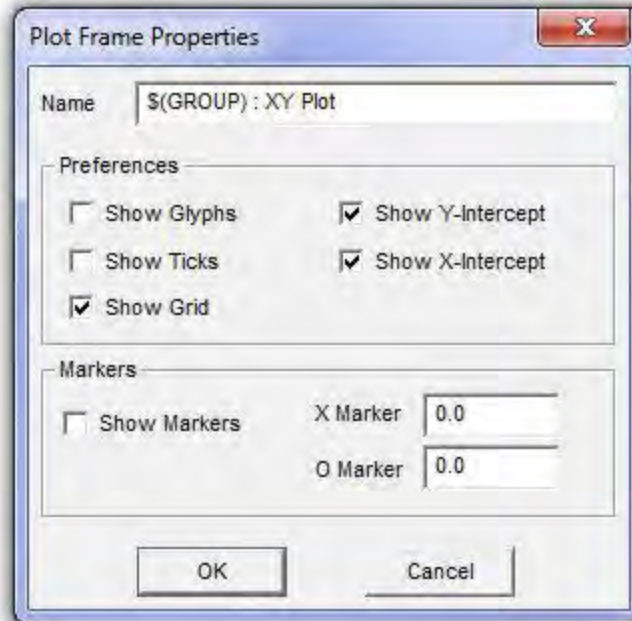
The data is copied as *Comma Separated Variables (\*.csv)* format for easy migration into common data analysis software.

Two choices are given:

- **All Data:** Copies all curve data available.
- **Visible Data:** Copies all curve data visible in the plot window.

## Adjusting XY Plot Frame Properties

To access the *Plot Frame Properties* dialog, left double-click the plot title bar, or right-click over the title bar and select **Plot Frame Properties....** This should bring up the Plot Frame Properties dialog window.



There are various parameters that may be edited through this window, each of which are described below.

- **Name:** Enter a title for the xy plot (this text will appear in the frame title bar). The default text may appear a bit cryptic: This syntax is used as a naming convention for grouping objects in the workspace. For more information on this syntax, see Grouping of Runtime Objects.

Preferences:

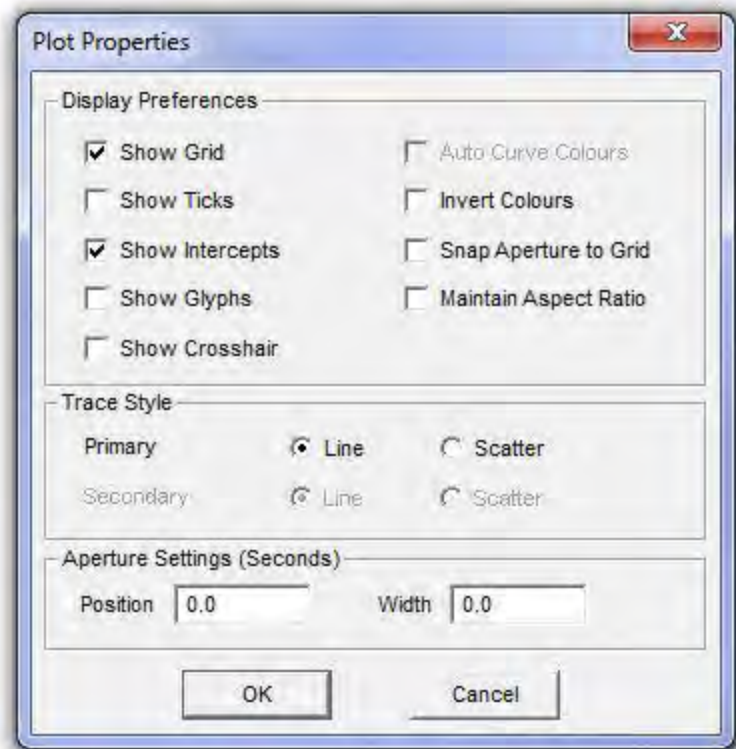
- **Show Glyphs:** Select this option to show glyph symbols on all curves.
- **Show Ticks:** Select this option to show major grid tick marks along the x and y-axis intercept lines.
- **Show Grid:** Select this option to display grid lines for the x-axis and y-axis major grids.
- **Show Y-Intercept:** Select this option to display the y-intercept (horizontal) intercept line. The y-intercept is always at zero, and cannot be adjusted.
- **Show X-Intercept:** Select this option to display the x-intercept (vertical) intercept line. The x-intercept is always at zero, and cannot be adjusted.

Markers:

- **Show Markers:** Select this option to show the X and O markers.
- **X Marker:** Enter the position (in seconds) to place X marker.
- **O Marker:** Enter the position (in seconds) to place O marker.

## Adjusting Plot Properties

Left double-click over the plot area (white part), or right-click over the plot area and select **Plot Properties....** This should bring up the Plot Properties dialog window.



There are various parameters that may be edited through this window, each of which are described below.

Display Preferences:

- **Show Grid:** Select this option to display grid lines for the x-axis and y-axis major grids.
- **Show Ticks:** Select this option to show major grid tick marks along the x and y-axis intercept lines.
- **Show Intercepts:** Select this option to display both intercept lines (horizontal and vertical). The intercepts are always at zero, and cannot be adjusted.
- **Show Glyphs:** Select this option to show glyph symbols on all curves in the graph.
- **Show Cross Hair:** Select this option to invoke the cross hairs mode.
- **Auto Curve Colours:** Select this option to use automatic colouring of curves in the graph. You cannot change curve colour manually when this option is selected.
- **Invert Colours:** Select this option to give the graph a black background (instead of white or yellow).
- **Snap Aperture to Grid:** Select this feature so that when using dynamic aperture adjustment, the aperture view will snap to the major grid while zooming.
- **Maintain Aspect Ratio:** Select this option in order to maintain the aspect ratio of the plotted curve (in both the x and y directions) whenever the plot frame is resized. If this option is disabled, the plotted curve will stretch or compress according to the actual shape of the plot frame.

Trace Style:

- **Primary:** Select whether to draw traces as **Line** or **Scatter** view. Scatter view simply adds a single dot for each X-Y coordinate.

Aperture Settings (seconds):

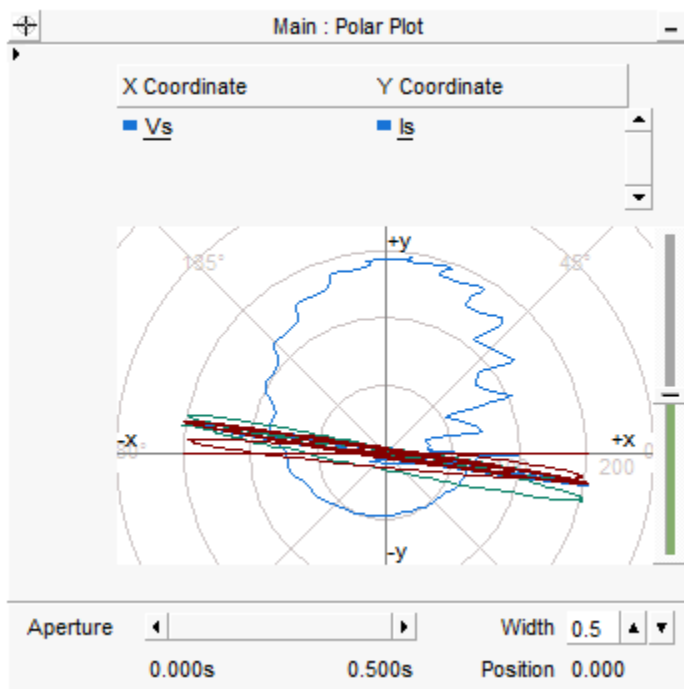
- **Position:** Enter the starting position in seconds of the aperture window.
- **Width:** Enter the width in seconds of the aperture window.

## Polar Grid

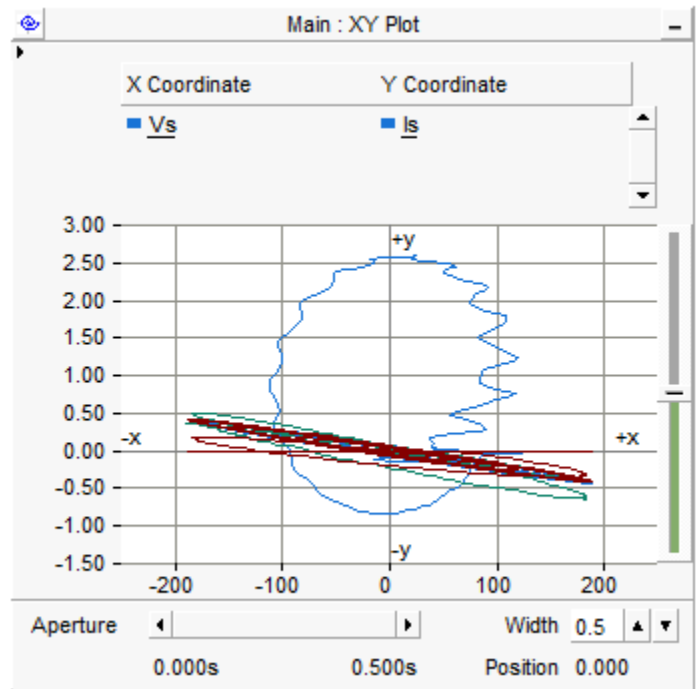
The default Cartesian (XY) grid display can be switched to a polar grid by simply toggling the xy/polar display button at the top-left of the plot frame.



The polar and xy grids are simply overlaid on top of the plotted data:



Polar Grid



Cartesian Grid

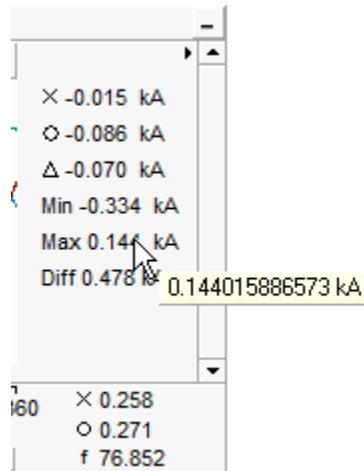
## Dynamic Zoom

See Dynamic Zoom in XY Plots for more details.

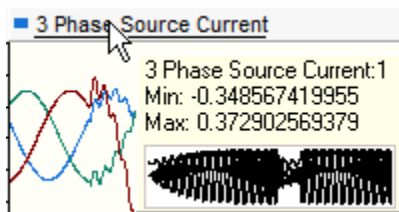
## Tool Tips in Plots

In order for plotting tools to be useful in the analysis of data, it is important that quantities be displayed with an adequate amount of precision. This presents a problem however, as the more precision that is displayed graphically on the plot the more space (or graphical real estate) is used, leaving less space in the plotting environment. This problem is overcome through the use of pop-up tool tips.

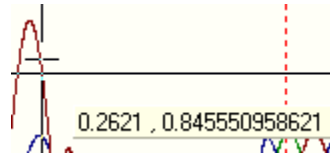
By default, the most important quantities involved in plotting are displayed directly on the graph frame. This is especially apparent when viewing marker data, which is displayed to four significant digits. For most studies, four significant digits are not enough to provide the user with accurate assessments of the data. Tool tips however will provide the true quantities up to 12 significant digits. Simply move the mouse pointer over the data displayed.



Existing tool tips, such as cross-hair and min/max curve are also standardized to 12 significant digits. The following images show the locations of other tool tips in the plotting environment:



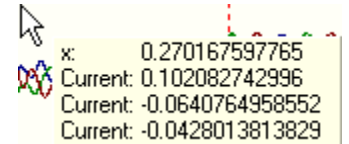
Min/Max Tool Tip



Cross-Hair Tool Tip



Marker Tool Tip

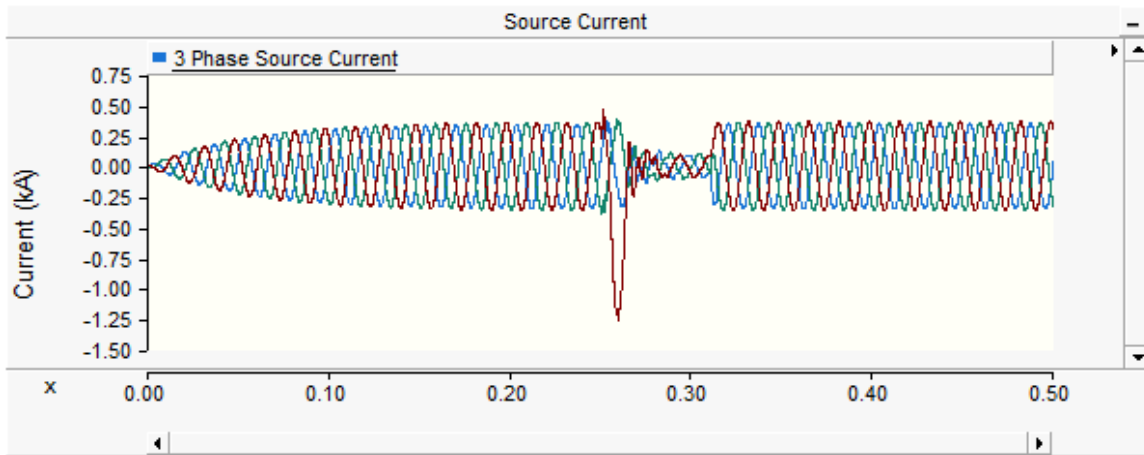


Data Tool Tip

## Dynamic Aperture Adjustment

The *Dynamic Aperture Adjustment* feature is available on both the graph frame and the xy plot objects and allows the user to define a fixed time based display window (or aperture), and then dynamically slide this aperture through the entire time scale. The aperture size itself can be re-adjusted at any time.

Although the following example uses a graph frame, it may be easily applied to the xy plot as well. Run the simulation so that your graph displays the curve data. The following image shows a simulation that has been run for 0.5 s.



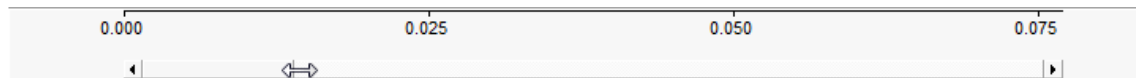
As can be seen from the graph, the fault occurs around 0.25 s and lasts for about 0.07 s. Dynamic aperture adjustment can be used to close the viewing window to a smaller time width, so that the fault waveforms can be more easily studied. To do this, move your mouse pointer over the horizontal scroll bar at the far right so that the mouse pointer turns into a double-headed arrow.



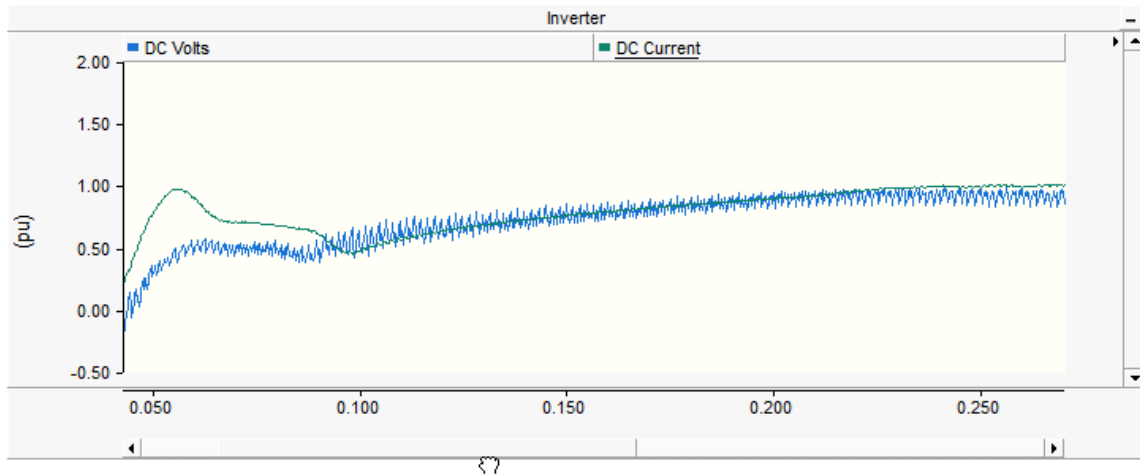
Click and hold the left mouse button and drag the scroll bar aperture slowly to the left.



You should see your graph display dynamically adjusting as you change the size of the scroll bar aperture. Keeping an eye on the horizontal axis display, shrink the aperture to an appropriate size (for the graph above, about 0.05 s or so).



Release the left mouse button and move the pointer over the scroll bar aperture, click and hold the left mouse button again – the mouse pointer should become a 'hand'. Drag the mouse so as to scroll across the time frame of the graph.



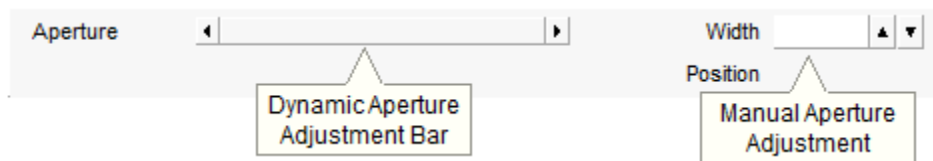
If desired, you can shrink the aperture further for even more detail. An aperture will also be created automatically when you zoom into a certain data range.

You may also use either the left and right arrow buttons at each end of the scroll bar, or the arrow keys on your keyboard to scroll through data: To scroll in small increments, simply left-click the either arrow, for larger increments hold down the **Ctrl** key and then left-click. When using the arrow keys, make sure that the focus is on the graph frame being viewed, as these keys can also function as a scroll mechanism for the Circuit canvas.

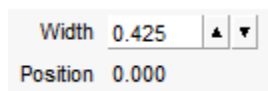
**NOTE:** This exercise is for the purpose of describing dynamic aperture adjustment. A more efficient way to zoom into this aperture window is to use horizontal zoom.

## Adjusting and Controlling the Aperture in XY Plots

As mentioned previously, the user is able to set and control the time viewing window for the xy plot. At the bottom of the xy plot frame, there is an area devoted solely for the adjustment and control of this viewing window - otherwise known as the aperture.



This area contains two sections: The dynamic aperture adjustment scroll bar on the left, and the manual aperture adjustment field on the right. You can increment the aperture width manually by pressing the up/down arrows to the right of the manual aperture adjustment field.

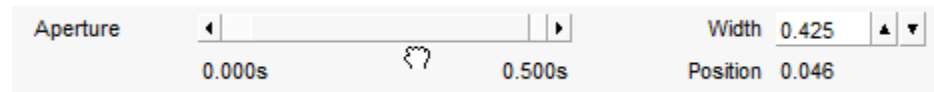


If the aperture **Width** is a fraction of the total time scale for the plot, then the aperture indicator scroll bar will reflect this by showing a smaller aperture window:





You may now move the aperture window along the time axis, maintaining its set width. To do this, move your mouse pointer over the aperture indicator in the scroll bar and **left-click and hold**. Your mouse pointer should become a 'hand' symbol. Move the mouse left or right along the axis.



The aperture **Position** indicator at the bottom-right corner will indicate the starting time of the aperture.

## Markers

Markers are a special feature included in both graph frames and xy plot frames to help users with the analysis of their online data. Specifically, they are used to delineate the data so as to focus analysis to that specific range. Depending on marker positions, legend displays will indicate the difference between the two markers in both the x and y directions.

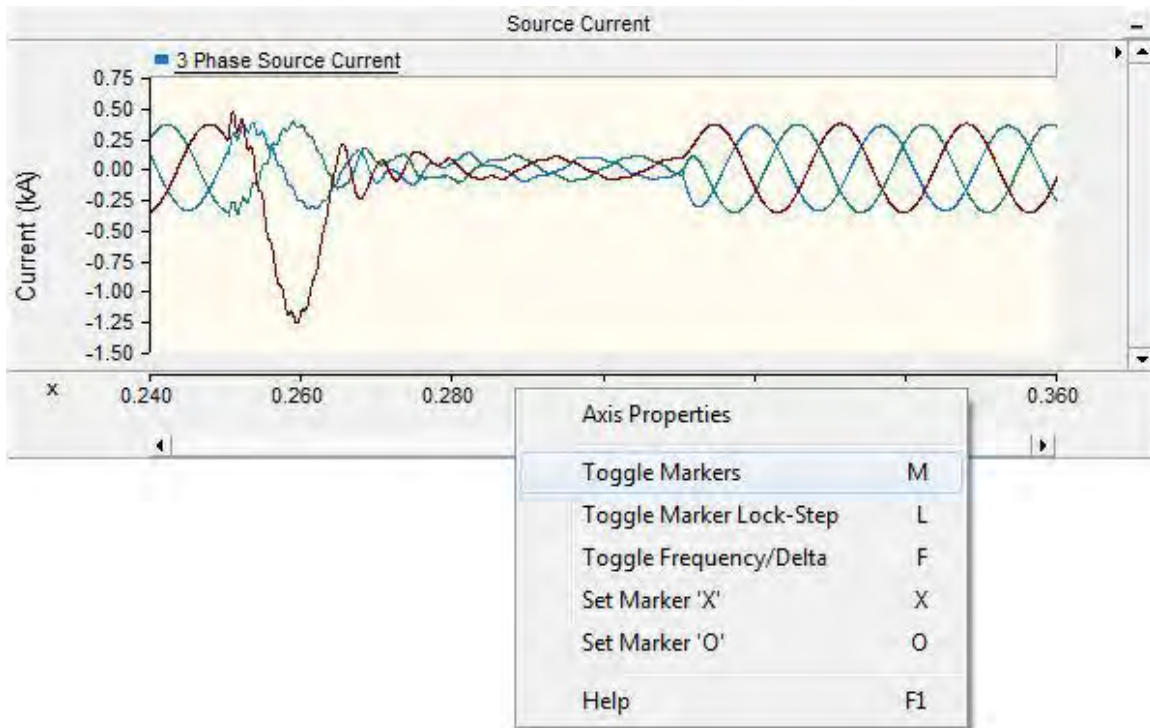
Markers are used only on the x-axis (time axis) and will appear as two adjustable tabs. The marker tabs are labelled as **X** and **O** and the combination of the two set the specified boundaries. Once markers are set, analysis can be performed on the data contained within them.

Markers are used in a slightly different manner between graph frames and xy plots. Any differences are noted in the sections below.

### Show/Hide Markers

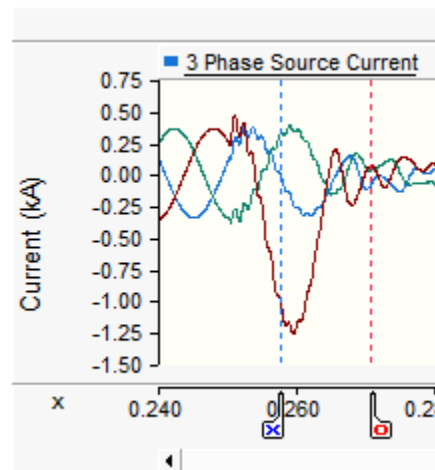
There are a few ways to show or hide Markers:

- Select the desired graph frame or xy plot with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select **Preferences | Show Markers**.
- Left double-click the graph frame horizontal axis, or right-click over the horizontal axis and select **Axis Properties...** to bring up the Axis Properties dialog. Select the **Show Markers** selection box.

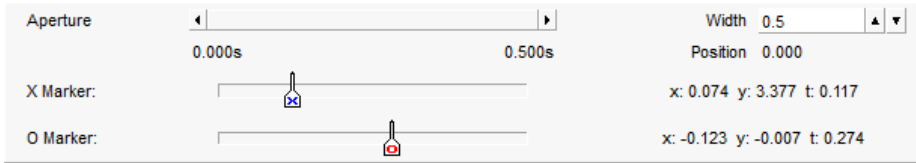


- Left double-click an overlay or polygraph, or right-click over the graph and select **Graph Properties...** to bring up the Graph Properties dialog. Select the **Show Markers** selection box.

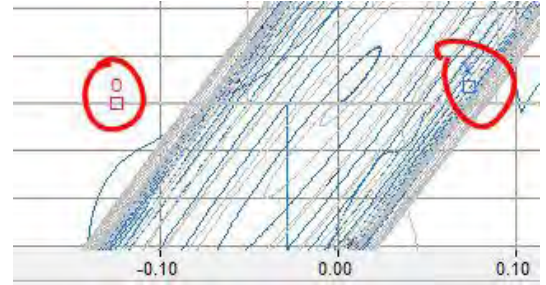
If you are working with a graph frame, two tabs should appear along the horizontal axis. Each marker tab, labelled **X** and **O**, correspond to an x-axis (time) position.



If you are working with an xy plot, an extra display bar will appear at the bottom of the plot frame as shown below:



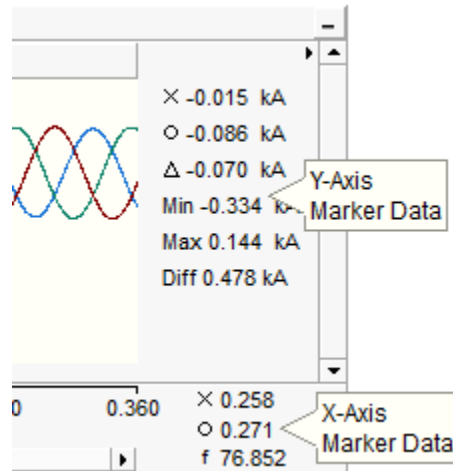
Aperture Control Bar



Marker Display in Graph

## Graph Frame Marker Legends

Once markers are enabled (shown) in a graph frame, legends will appear on the right hand side of the frame itself.



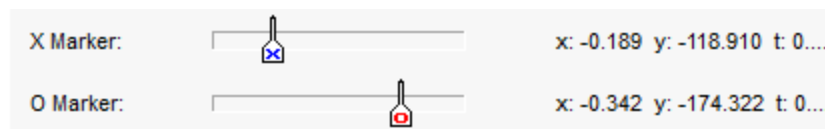
Each graph in the graph frame will have a legend directly to the right of the graph. The values displayed are specific to each graph and are described as follows:

- **X**: Displays the y-axis value at the X marker position for the active trace.
- **O**: Displays the y-axis value at the O marker position for the active trace.
- **Δ**: Displays the difference between the above two values
- **Min**: Displays the minimum value of the active trace within the marker boundaries
- **Max**: Displays the maximum value of the active trace within the marker boundaries

The x-axis will also have its own legend where the values displayed are similar to those described above, but for the x-axis.

## XY Plot Marker Legends

Once markers are enabled (shown) in an xy plot, a marker bar will appear at the bottom of the plot frame. As soon as either of these markers is moved, marker legends will appear directly to the right of each:



The values displayed are described as follows, specific to each marker:

- **x**: Displays the marker x-axis value for the active trace.
- **y**: Displays the marker y-axis value for the active trace.
- **T**: Displays the marker time-axis value for the active trace.

## Changing the Active Curve

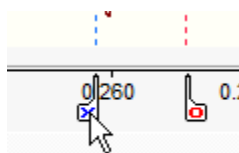
The markers will only monitor the active trace of one curve at a time. If multiple curves exist in an overlay or polygraph, you can scroll through them by simply pressing the **Space Bar** on your keyboard. You may also conveniently select the desired curve in the graph legend.

When using xy plots, you must select the desired curve in the graph legend, as the **Space Bar** function is not available.

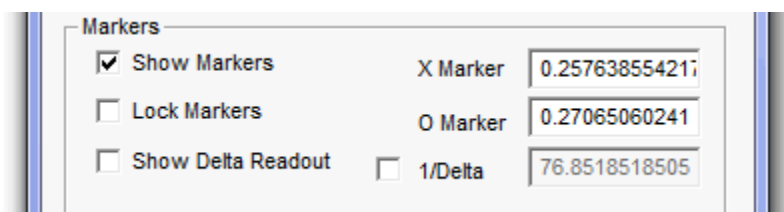
## Adjusting Marker Positions

Once markers are enabled, there are a few ways to adjust their positions:

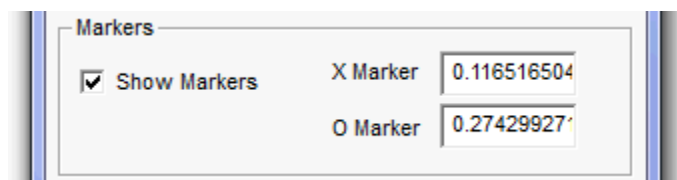
- To manually adjust, place the mouse pointer over the marker tab in either the graph frame or xy plot, left-click and hold and drag it left or right. Release the left mouse button when the marker is in the desired position.



- When using graph frames, left double-click the frame horizontal axis, or right-click over the horizontal axis and select **Axis Properties...** to bring up the Axis Properties dialog. Select the **Show Markers** selection box and then set the x-axis values of each marker directly in the input fields provided. Note that it is also possible to directly set the frequency (i.e. 1/Delta) between the two markers.

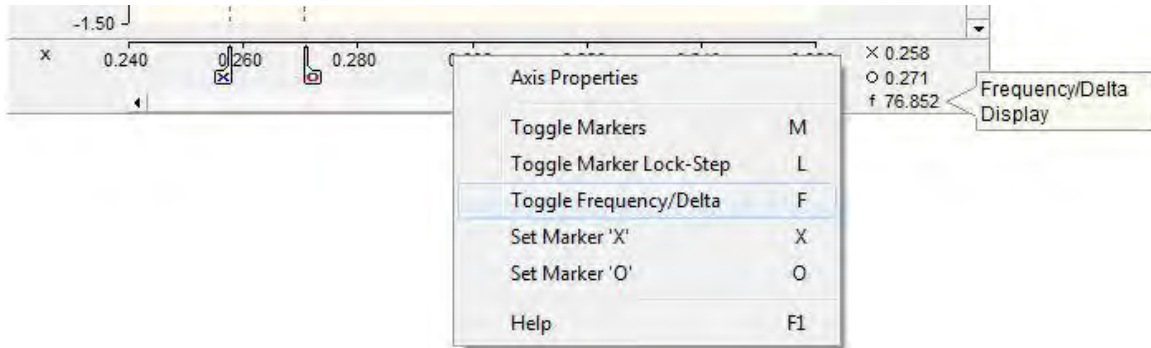


- When using xy plots, right-click over the plot title bar and select **Plot Frame Properties...** to bring up the Plot Frame Properties dialog. Select the **Show Markers** selection box and then set the time-axis values of each marker directly in the input fields provided.



## Toggle Time Difference Frequency/Delta

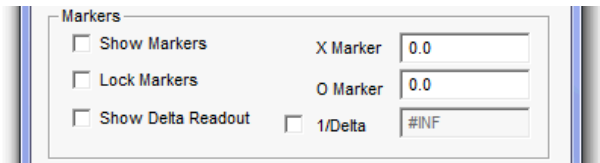
Once the markers are enabled, you can conveniently invert the time axis difference between them to display corresponding frequency. **Left double-click** the graph frame horizontal axis, or right-click over the horizontal axis and select **Toggle Frequency/Delta** (or press the **F** key on your keyboard):



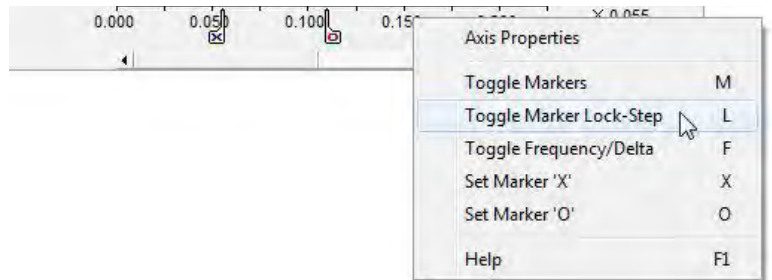
This feature is not available in xy plots.

## Locking/Unlocking Markers

Locking will force the two markers to maintain a constant distance between each other when they are moved along the x-axis. Left double-click the graph frame horizontal axis to bring up the Axis Properties dialog and select **Lock Markers**, or right-click over the horizontal axis and select **Toggle Marker Lock-Step** (or press the **L** key on your keyboard) to lock the markers. Perform the same operation again to unlock.



Axis Properties Dialog

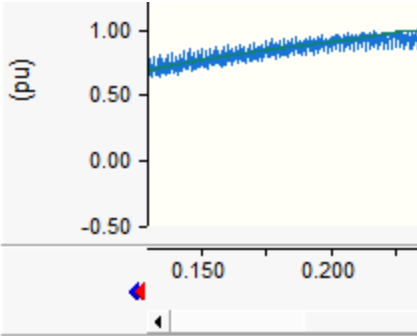


Right-Click Menu

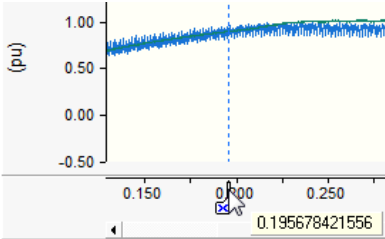
## Setting Markers

Markers can be set to a certain position on the graph frame time axis as follows:

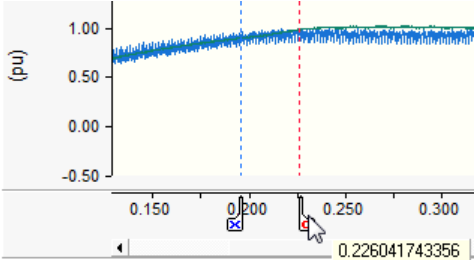
1. Show the markers by pressing the **M** key.
2. Left-click on the axis at the approximate position where you wish to place the **X** marker. Press the **X** key on your keyboard.
3. Repeat this procedure to set the **O** marker, except press the **O** key.



Press the **M** Key

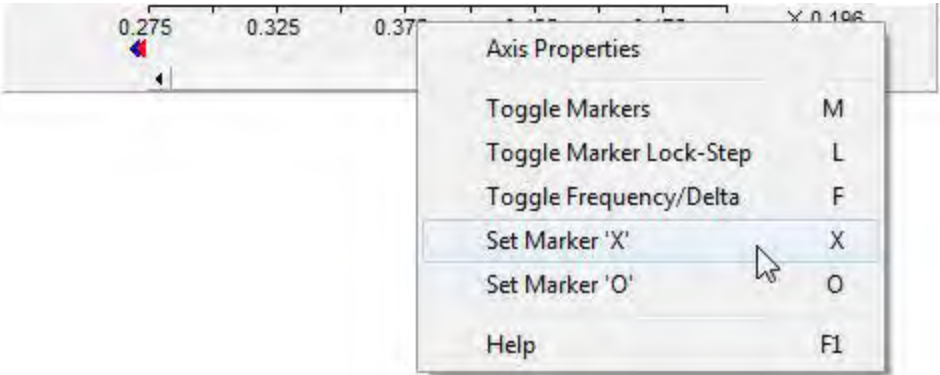


Press the **X** Key



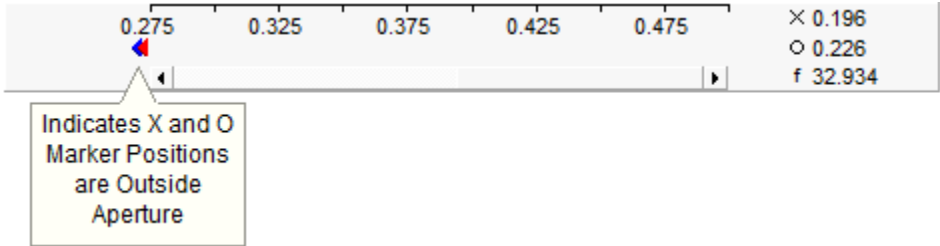
Press the **O** Key

Instead of pressing the above keys, you can also use the time axis pop-up menu. Left-click on the graph frame time axis at the approximate position where you wish to place a marker, then select the appropriate menu function.



### Using Markers as Bookmarks

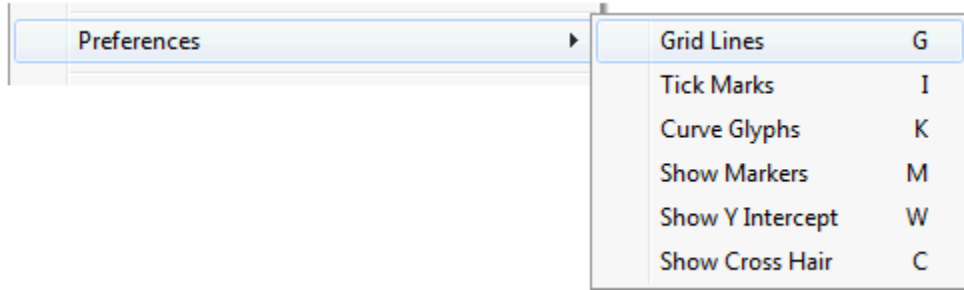
If only a fraction of the total graph frame time scale is currently being viewed in a graph, the markers may appear outside the viewing frame when they are first enabled. If this happens, blue and red arrow symbols will appear on the time axis indicating the markers are set off the screen.



The current viewing window can be moved directly to the position of either marker: Simply left-click either the blue or the red arrow (blue for the X marker and red for the O marker). This functionality, along with setting the markers, can be used to 'bookmark' viewing positions on the graph.

## Preferences

Plotting preferences can be adjusted from either the graph or xy plot pop-up menus (depending on which one you are using). Simply right click over a graph or xy plot to bring up the corresponding pop-up menu. Then select **Preferences**.



The functions available in this menu are described below:

- **Grid Lines:** Select this preference to toggle grid line display on the selected graph. You can also use the **G** keyboard shortcut.
- **Tick Marks:** Select this preference to show major grid tick marks along the y-intercept. You can also use the **I** keyboard shortcut.
- **Curve Glyphs:** Select this preference to show glyphs on all curves in the graph. You can also use the **K** keyboard shortcut.
- **Show Markers:** Select this preference to show the markers. You can also use the **M** keyboard shortcut.
- **Show Y Intercept:** Select this preference to display the y-intercept (horizontal) intercept line. The y-intercept line can be adjusted using the **Y-Intercept** field in the Graph Properties dialog. On xy plots, the y-intercept is always along the x-axis. You can also use the **W** keyboard shortcut.
- **Show Cross Hair:** Select this preference to invoke the cross hairs mode.

Many of the above preferences can be set within the various dialog windows involved with graphs and xy plots.

## Zoom Features

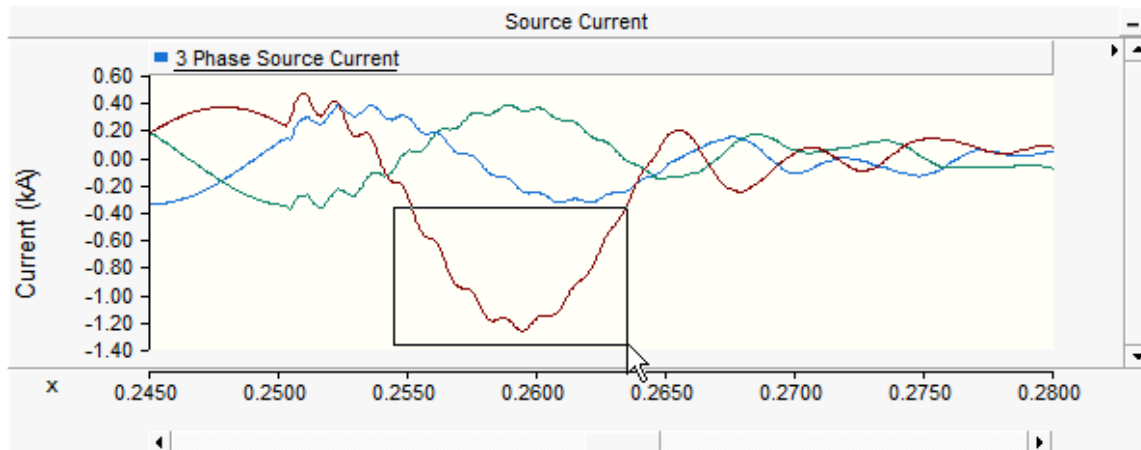
Once a simulation has been run and output data has been collected, there are several ways to zoom in and out of the data displayed. The following are some of the more common methods.

### General Zoom In and Zoom Out

Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select **Zoom | Zoom In** or **Zoom | Zoom Out**. As an alternative, you can also use the **+** or **-** keyboard shortcuts for **Zoom In** or **Zoom Out** respectively (once the desired graph has been selected).

## Box Zoom

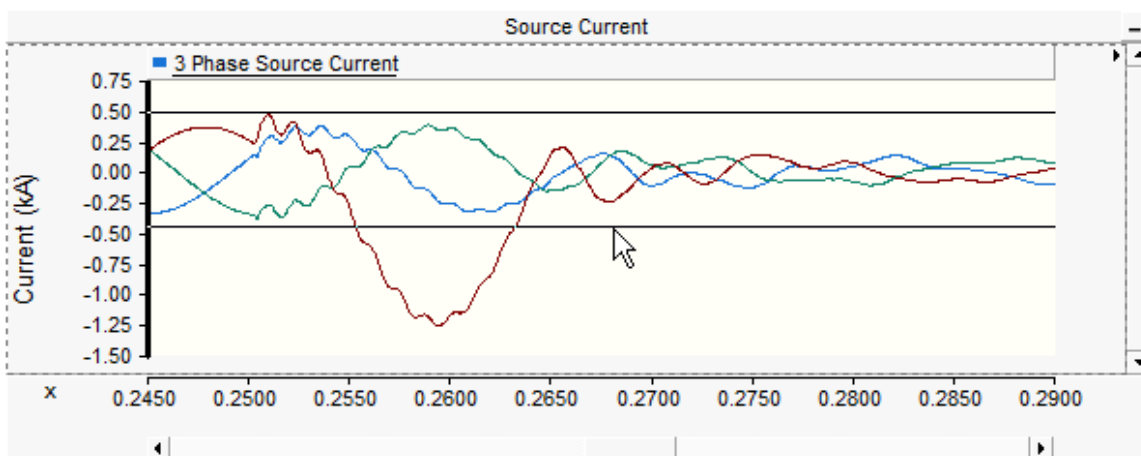
Select the desired graph with a left-click on the graph display area. **Click and hold the left mouse button** and drag the mouse pointer to create a boxed region. Release the left mouse button to zoom to that region.



**NOTE:** You can also zoom along a single axis by drawing out a narrow box. The zoom will ignore the narrow side of the box.

## Vertical Zoom

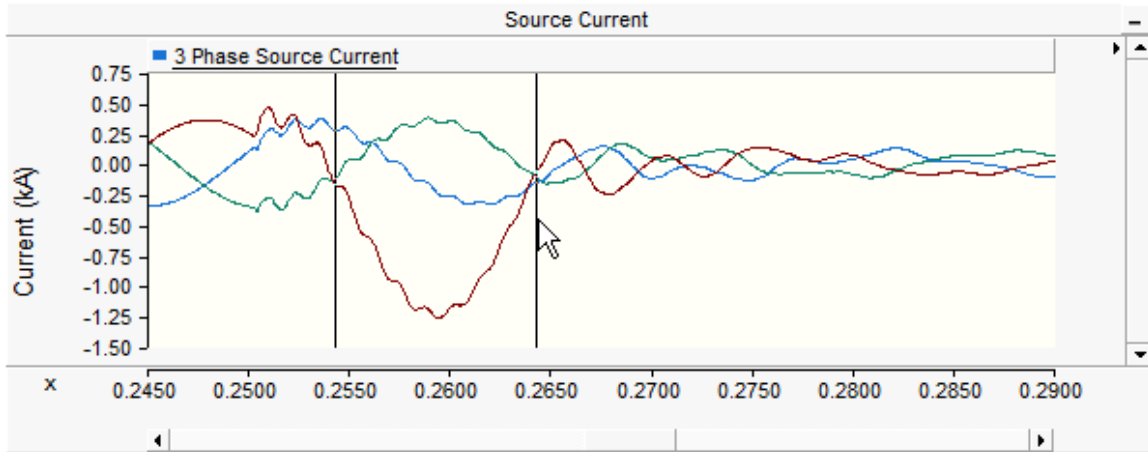
Select the desired graph with a left-click on the graph display area. Press **Shift + left mouse hold** and drag the mouse pointer in a vertical direction (i.e. up or down along the y-axis) to create a vertical zoom region. Release the left mouse button to zoom to that region.



## Horizontal Zoom

Select the desired graph with a left-click on the graph display area. Press **Ctrl + left mouse hold** and drag the mouse pointer in a horizontal direction (i.e. left or right along the x-axis) to create a horizontal zoom region. Release the left mouse button to zoom to that region.





## Zoom Previous/Next

Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select **Zoom | Previous** or **Zoom | Next**. As an alternative, you can also use the **P** or **N** keyboard shortcuts for **Zoom Previous** or **Zoom Next** respectively (once the desired graph has been selected).

## Zoom Extents

The Zoom Extents feature allows the user to zoom to the extents of the plotted data. The data 'extents' refers to the absolute maximum and minimum values that exist during the entire simulation run, in either the x or y direction.

Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select either **Zoom | X Extents** or **Zoom | Y Extents**.

As an alternative, you can also use the **X** or **Y** keyboard shortcuts for **Zoom X Extents** or **Zoom Y Extents** respectively (once the desired graph has been selected).

## Zoom Limits

The Zoom Limits feature allows the user to zoom to predefined limits in either the x or y direction. The y-axis limits are set based on the **Default Display Limits** parameters in the corresponding Output Channels for the curves. In the event of multiple curves, the limits are based on the largest and smallest *Default Display Limits* among all relevant output channels. The x-axis limits are based on the desired duration of the simulation.

Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select either **Zoom | X Limits** or **Zoom | Y Limits**.

As an alternative, you can also use the **E** or **U** keyboard shortcuts for **Zoom X Limits** or **Zoom Y Limits** respectively (once the desired graph has been selected).

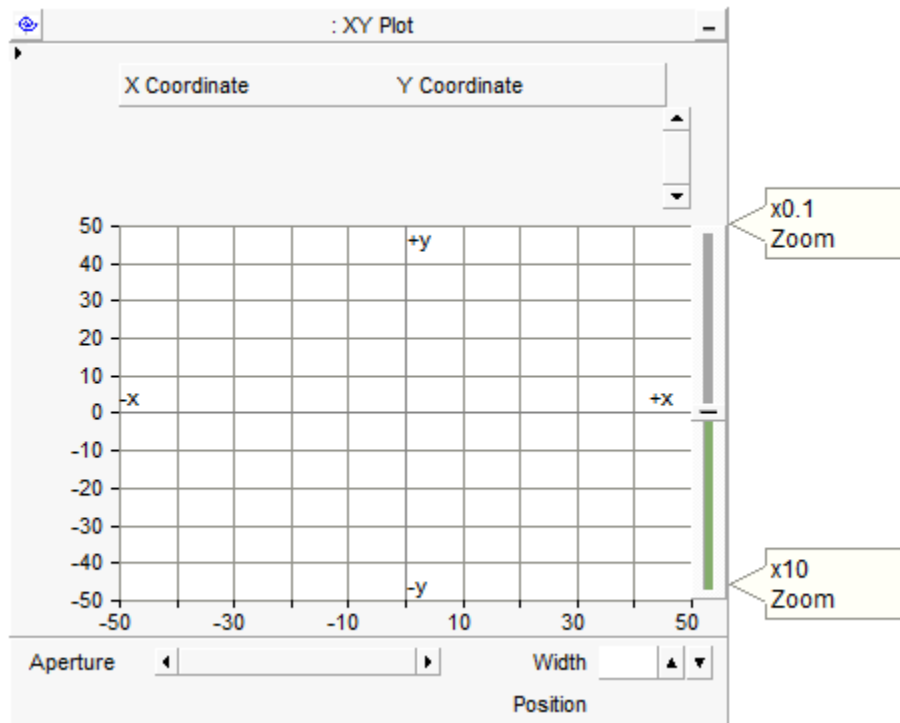
## Resetting All Extents and Limits

Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select either **Zoom | Reset All Extents** or **Zoom | Reset All Limits**.

As an alternative, you can also use the **R** or **B** keyboard shortcuts for **Reset All Extents** or **Reset All Limits** respectively (once the desired graph has been selected).

## Dynamic Zoom in XY Plots

Dynamic zoom is a feature specific to the xy plots. Left click and hold with your mouse pointer over the dynamic zoom control bar as shown below:



With a left click, your mouse pointer should become an open 'hand'. Hold down the left mouse button and move the hand down to zoom in, or up to zoom out.

## Cross Hair Mode

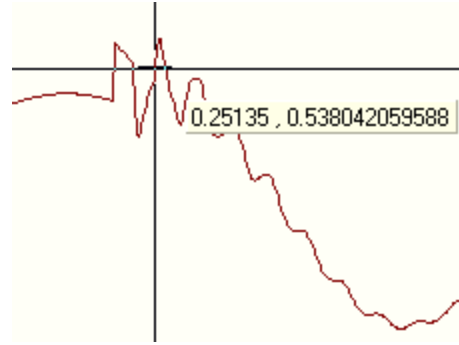
Once a simulation has been run and output data has been plotted, you can navigate through curve values using the cross hairs. Right-click over the graph to generate a pop-up menu and then select **Preferences | Show Cross Hair**. You can also select the graph and press the **C** key on your keyboard.

Once invoked, cross hair mode will remain on until it is turned off. While in cross-hair mode, the mouse pointer becomes a 'cross-hair' while over top the graph:



Cross-Hair Mode Indicator

Once cross hair mode is enabled, move the mouse pointer over the graph and left-click and hold. Drag the mouse along the graph. If multiple curves exist in the graph, then you can move the cross hair from curve to curve by simply pressing the **Space Bar** on your keyboard.

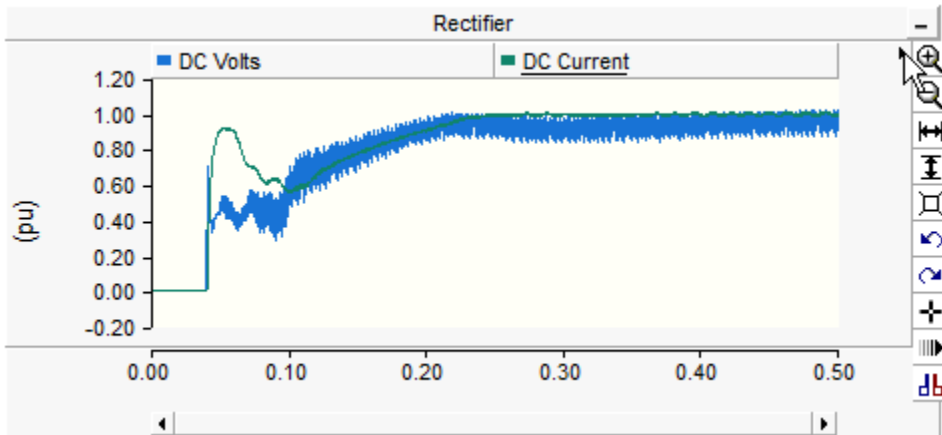


As shown above, the curve xy data will be displayed beside the cross hair. When your left mouse button is released, the cross hair will disappear, but cross hair mode will remain invoked until you press the **C** key again (or select **Preferences | Show Cross Hair**).

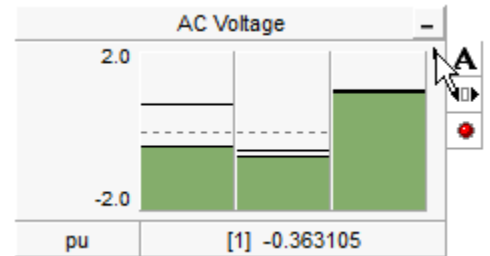
In xy plots, the cross hair will follow the mouse pointer only.

## Pop-Up Tool Bars

Pop-up tool bars are provided with graphs and polymeters in order to provide a quick and easy means for accessing the most common functions. Pop-up tool bars may be invoked by left-clicking on the small arrow in the top-right corner of the graph or poly meter windows:




Graph












PolyMeter




The pop-up tool bar functions are described as follows:

Graphs:

Button	Description
	Zoom In

	Zoom Out
	Zoom horizontal extents
	Zoom vertical extents
	Reset all extents
	Previous zoom
	Next zoom
	Toggle cross hair mode
	Toggle auto-pan of x-axis
	Toggle markers

PolyMeters:

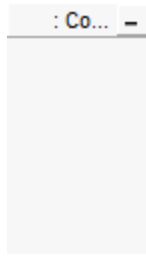
Button	Description
	Toggle index labels
	Toggle scroll view mode
	Modify gauge color

## Online Controls and Meters

Special objects in PSCAD allow the user online access to input data signals, so that these signals (and hence the simulation results) can be altered during the course of the simulation run. These objects and how to use them are described in the following sections.

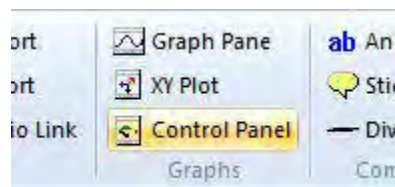
### Control Panels

A *Control Panel* is a special component used for accommodating control or meter interfaces and can be placed anywhere on a *Schematic* canvas. Once a control panel has been added, you may then proceed to add as many control or meter interfaces to it as you wish.



## Adding a Control Panel

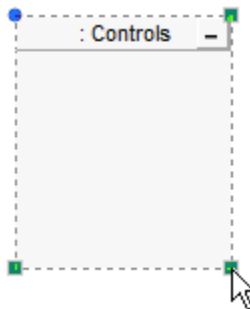
Open the project in the Circuit view. Right-click on a blank portion of the page and select **Add Component | Control Panel**, or press the **Control Panel** button in **Components** tab of the ribbon control bar.



## Moving and Re-sizing a Control Panel

To move a control panel, move the mouse pointer over the title bar and then **left-click and hold**. Drag the frame to where it is to be placed and release the mouse button.

To re-size, move the mouse pointer over the title bar and left-click to select the panel. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. **Left-click and hold** and then drag then move the pointer to re-size.

## Cut/Copy Panel

Right-click over the control panel title bar and select **Cut** or **Copy** respectively. You can also use the standard **Ctrl + X** or **Ctrl + C** shortcuts. Once a control panel has been cut or copied it may then be pasted into another location in the project (along with its contents).

## Paste Panel

Cut or copy a control panel as described above. Right-click over a blank area of the *Schematic* canvas and select **Paste**. A control panel may be pasted multiple times.

## Adjusting Panel Settings

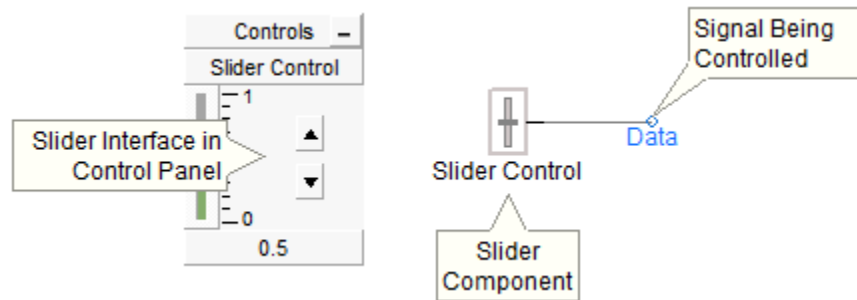
To access the Control Panel Settings dialog, left double-click the control panel title bar, or right-click over the title bar and select **Control Panel Settings...**

Presently the only adjustable panel property is the caption:

- **Caption:** Enter a title for the control panel (this text will appear in the panel title bar). The default text may appear a bit cryptic: This syntax is used as a naming convention for grouping objects in the workspace.

## Control Interfaces

A *Control Interface* is a user interface object, which allows manual adjustment of input data signals dynamically. A control interface must first be linked with one of the runtime control objects available in the master library (i.e. Slider, Two State Switch, Rotary Switch, and Push Button). The control interface will then control the output of the linked control component. For example, the following image shows a slider component linked to a control interface in a control panel.

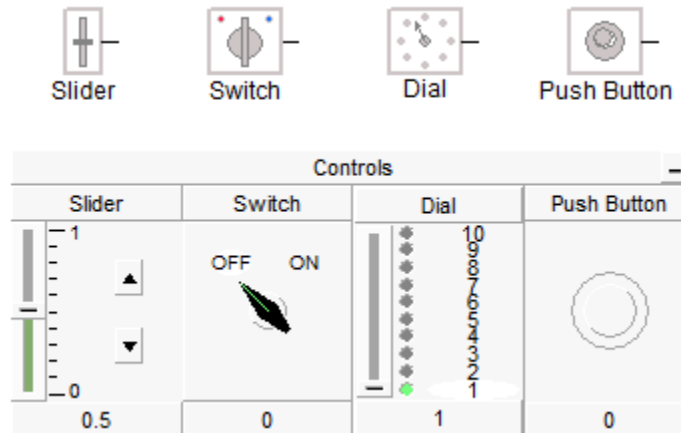


## Adding a Control Interface to a Control Panel

There are two methods to accomplish this:

- **Drag and Drop:** Hold down the **Ctrl** key, left-click and hold the associated control component and drag and release over the control panel. See Drag and Drop for more details.
- **Right-click** on the associated control component and select **Graphs/Meters/Controls | Add as Control**. Right-click over the desired control panel title bar and select **Paste**.

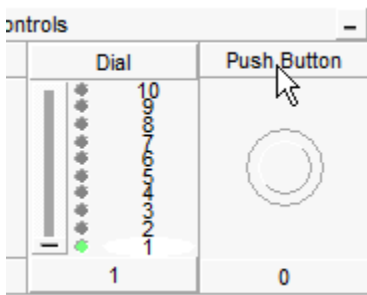
Of course, each type of control component will have a different control interface when added to a control panel. The following figure shows the available control components and their corresponding control interfaces:



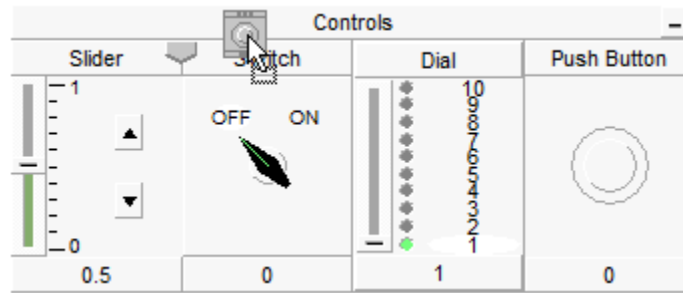
## Control Interface Order

Once multiple control interfaces have been added to a particular panel, you may change the order in which they appear. This is accomplished by either using drag and drop, or the right-click menu:

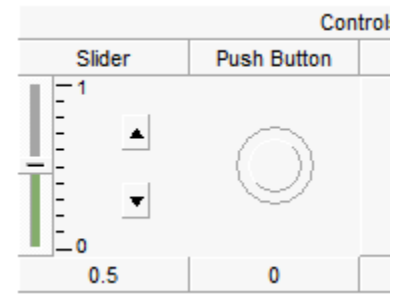
- **Drag and Drop:** Press **Ctrl** and then **left-click and hold** over the title bar of a particular control interface and then drag the mouse pointer over the control panel title bar. A downward arrow will appear between existing interfaces depending on the position of the mouse pointer. Release the left button to drop the interface.



Left-Click and Hold



Drag



Release Button

- Right-click over the control interface and select **Set Control Order**. Choose one of the options given.

## Cut/Copy Control Interface

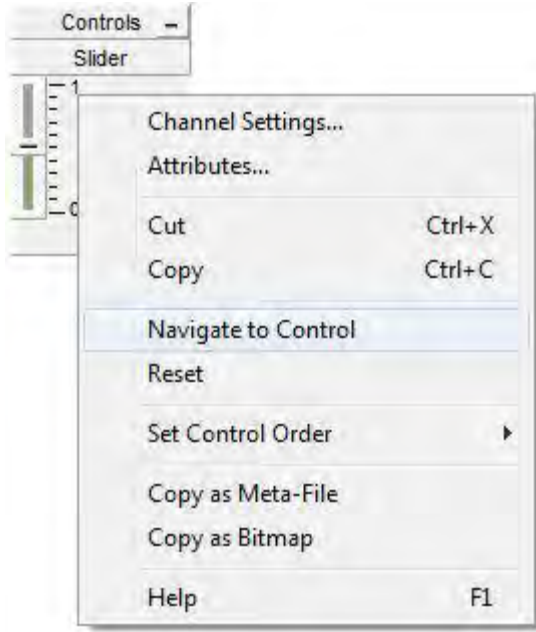
Right-click over the control interface title bar and select **Cut** or **Copy** respectively. Once a control interface has been cut or copied it may then be pasted into control panel in the project.

## Paste Control Interface

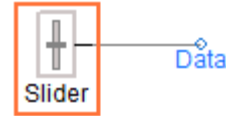
Cut or copy a control interface as described above. Right-click over a control panel title bar and select **Paste**. A control interface may be pasted multiple times.

## Navigate to Control

You can navigate directly to the control component associated with a particular interface, by selecting this option. Right-click over the interface and select **Navigate to Control**. PSCAD will automatically find the associated output channel and highlight it.



Select Navigate to Control



PSCAD Points to Object

## Adjusting Control Interface Properties

Control interface properties and the corresponding control component properties are one and the same. Therefore, you can either adjust these properties through the control component directly, or through the corresponding interface as follows: Left double-click the interface title bar, or right-click over the interface title bar and select **Channel Settings...**

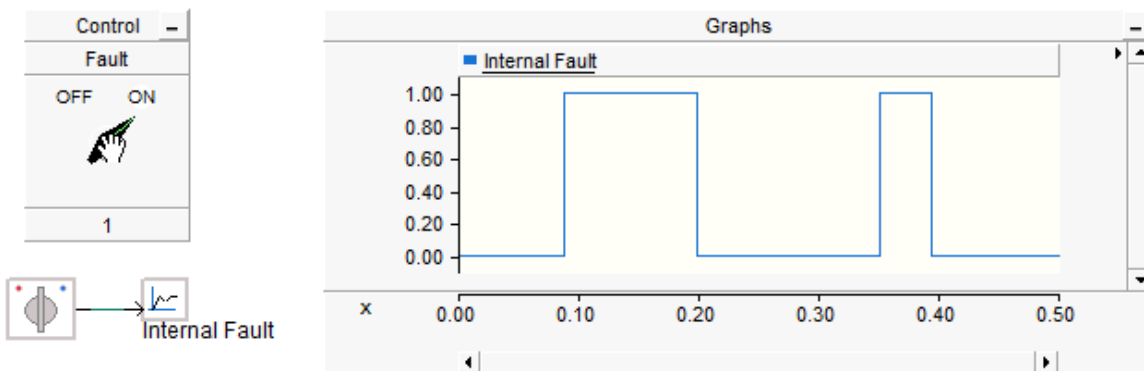
## Using the Control Interfaces

Once a particular control interface has been added to a panel, the user may operate it manually, either before or during a simulation run. Of course, each type of interface will perform a different operation on its corresponding output signal.

### Switch and Push Button Components:

To operate either the Switch or Push Button components, simply left-click over top on the control interface itself. For example, each time the switch interface is clicked it will toggle the corresponding switch component between its two output states.

The image below shows a plot of a switch component output, where the user has changed its state (i.e. turned it on and off) during a 1.0 second simulation run.

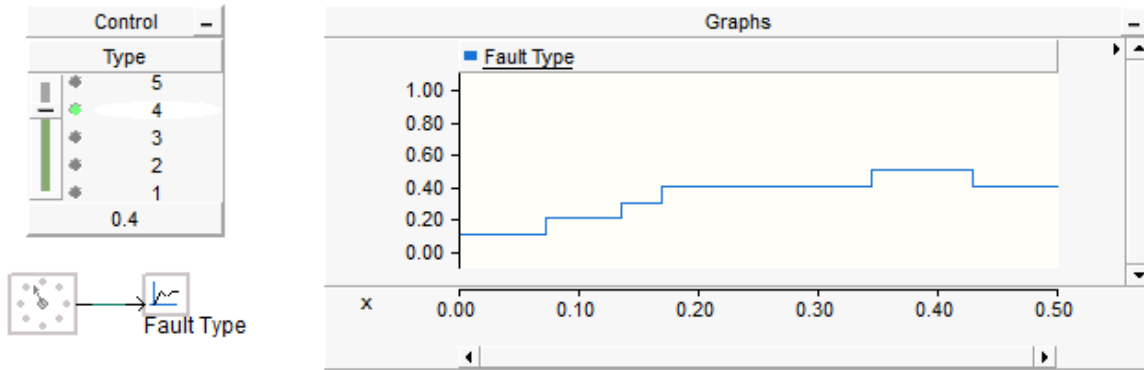




Dial Component:

To operate the Dial component, left-click and hold on the slider control knob and move the knob up or down. As the control knob is moved, the dial interface will indicate graphically, which of the multiple states is currently being output from the corresponding dial component.

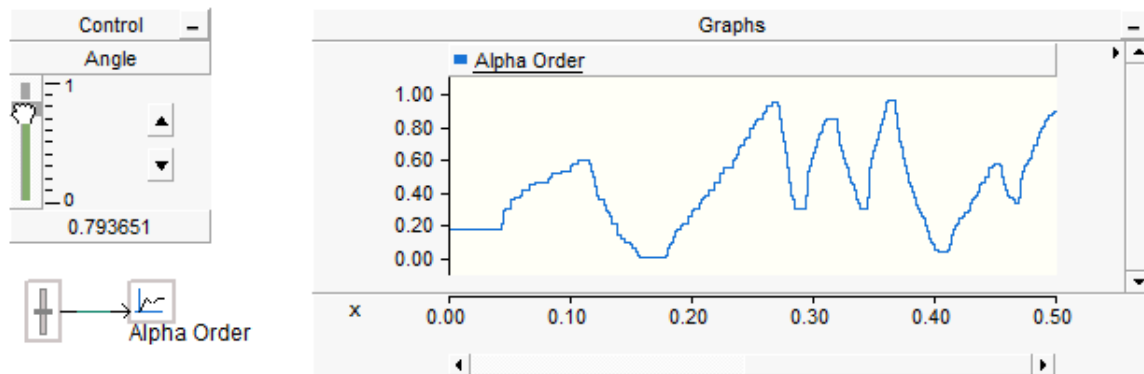
The image below shows a plot of a dial component output, where the user has changed its state during a 1.0 second simulation run.



Slider Component:

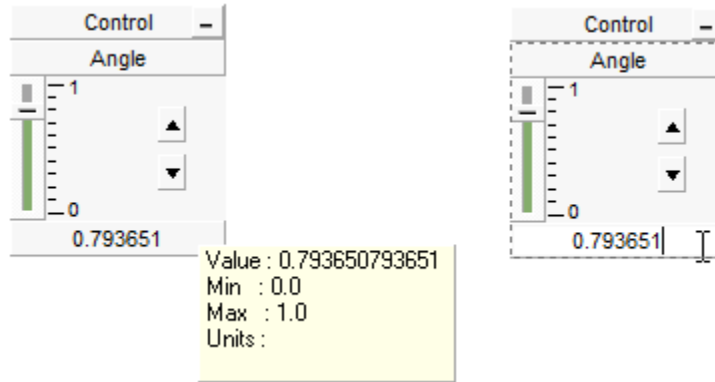
The Slider component is operated in the same manner as the dial interface described above. The only difference is that the slider output does not operate in discrete states, but provides a continuously variable output.

The image below shows a plot of a slider component output, where the user has varied its output during a 1.0 second simulation run.



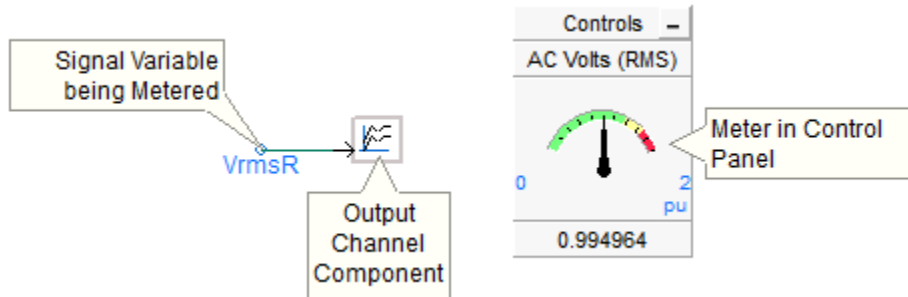
It is important to note that the Slider component will output exactly what is shown on the slider interface. The slider interface has a maximum precision of 6 significant digits. If more precision is required, you can add successive slider outputs together, each handling different data ranges. Once you have fine-tuned the slider output, you can replace it with a Real Constant component, which is capable of higher precision.

The slider interface also allows you to enter an exact value (up to 6 significant digits) directly into its display window. Simple left double-click on the display window and then enter the data. Press the **Enter** button to exit and save the entered data, or press **Esc** to exit without saving changes.



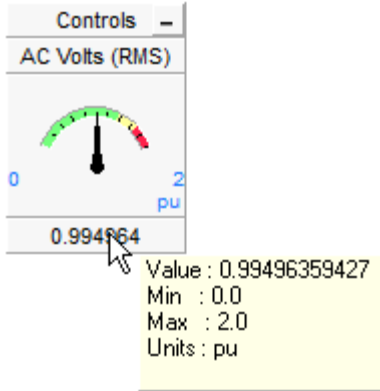
## Meters

A meter is similar to a graph, in that it is used for displaying an output signal and is linked to a corresponding Output Channel. Instead of displaying the signal as a curve (as a graph would), the signal is used to operate a realistic meter display, with a pointer position proportional to the signal magnitude. For example, the following image shows an output channel component linked to a meter in a control panel.



Meters can exist only within a control panel.

Like a control interface, meter interfaces display a maximum of 6 significant digits. However, twelve significant digits can be viewed via the tool tip.



## Adding a Meter to a Control Panel

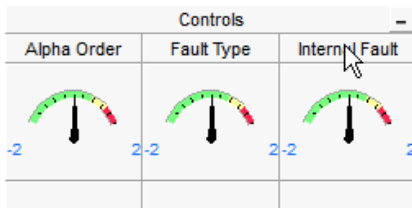
There are a couple of methods to accomplish this:

- **Drag and Drop:** Hold down the **Ctrl** key, **left-click and hold** the associated Output Channel component and drag and release over the control panel. See Drag and Drop for more details.
- **Right-click** on the associated output channel component and select **Graphs/Meters/Controls | Add as Meter**. Right-click on the desired control panel title bar and select **Paste**.

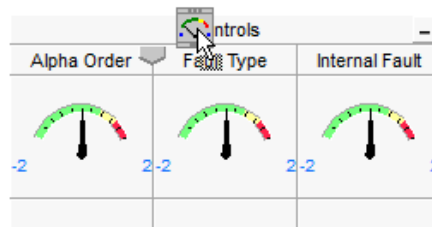
## Meter Order

Once multiple meter interfaces have been added to a particular control panel, you may change the order in which they appear.

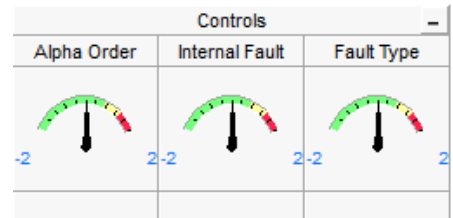
- **Drag and Drop:** Press **Ctrl** and then **left-click and hold** over the title bar of a particular meter interface and then drag the mouse pointer over the control panel title bar. A downward arrow will appear between existing interfaces depending on the position of the mouse pointer. Release the left button to drop the interface.



Left-Click and Hold



Drag



Release Button

- Right-click over the meter interface and select **Set Control Order**. Choose one of the options given.

## Cut/Copy Meter

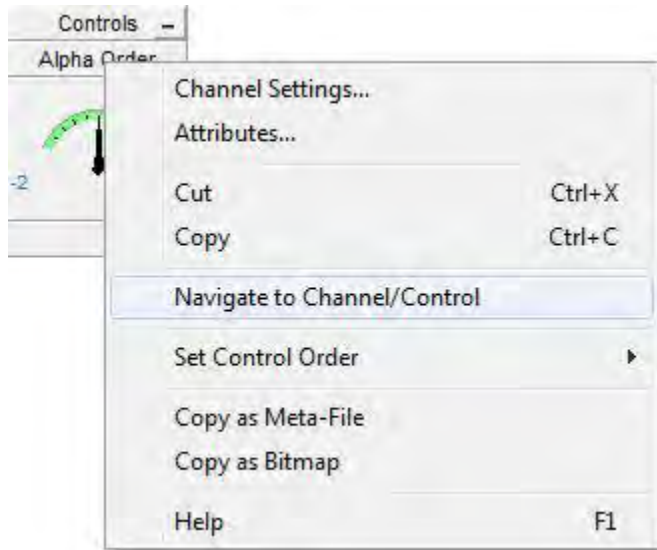
Right-click over the meter title bar and select **Cut** or **Copy** respectively. Once a meter has been cut or copied it may then be pasted into any control panel in the project.

## Paste Meter

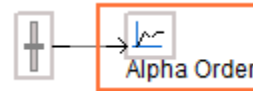
Cut or copy a meter as described above. Right-click over a control panel title bar and select **Paste**. A meter may be pasted multiple times.

## Navigate to Channel/Control

You can navigate directly to the Output Channel or I/O control component associated with a particular meter interface, by selecting this option. Right-click over the meter interface title bar and select **Navigate to Channel/Control**. PSCAD will automatically find the associated output channel or I/O control component and highlight it.



Select Navigate to Channel/Control



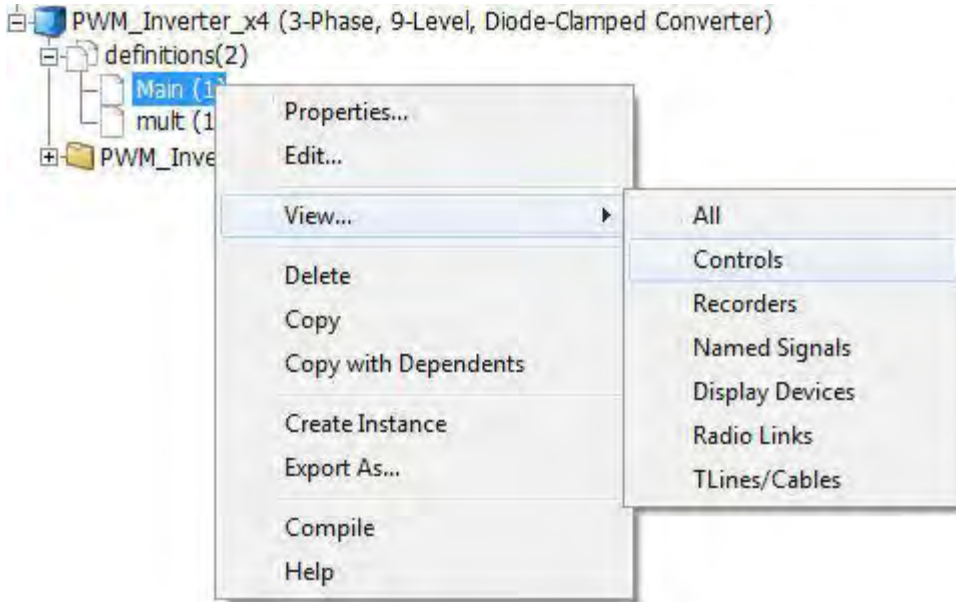
PSCAD Points to Object

## Adjusting Meter Properties

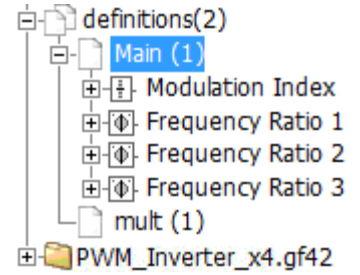
Meter properties and the corresponding Output Channel parameters are one and the same. Therefore, you can either adjust these properties through the output channel directly, or through the meter as follows: Left double-click the meter title bar, or right-click over the title bar and select **Channel Settings....**

## Grouping of Runtime Objects

It is possible to organize both control (i.e. Slider, Dial, Switch or Push Button) and Output Channel objects according to a *Group Name*. Once group names have been specified, objects may be viewed according to their group name in the primary workspace window. See Runtime Objects in the Definitions Branch for more details on accessing runtime objects.



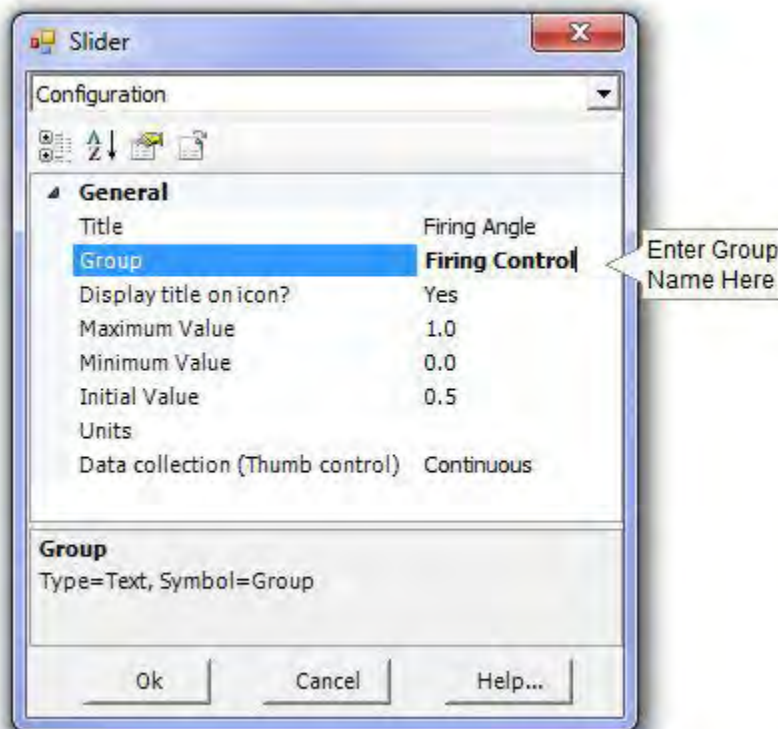
Manually Populating the List



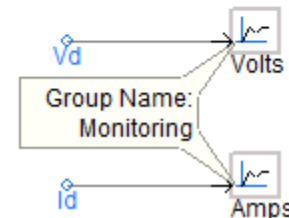
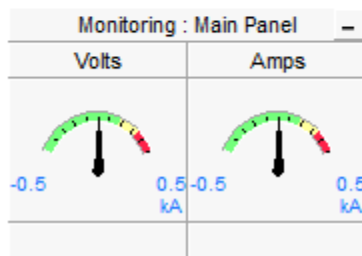
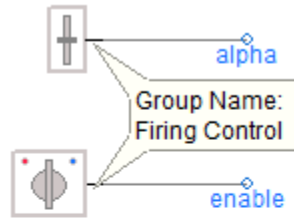
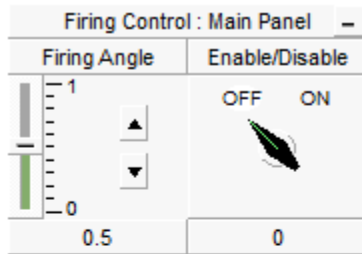
Resulting List of Controls-Type Runtime Objects in definition *Main*

## Creating a Runtime Object Group

Creating a runtime group is straightforward: All that is required is that a group name be added to the object input parameter called *Group*. The image below shows the *Group* parameter in the Slider component parameters dialog.



For example, consider the following project *Schematic* canvas, where the user wishes to group existing control objects according to function:



Both the *Firing Angle* slider and *Enable/Disable* switch controls are grouped under the name *Firing Control*, whereas the two output channels *Volts* and *Amps* are grouped under the name *Monitoring*.

## Displaying Group Name on Graph Frames and Control Panels

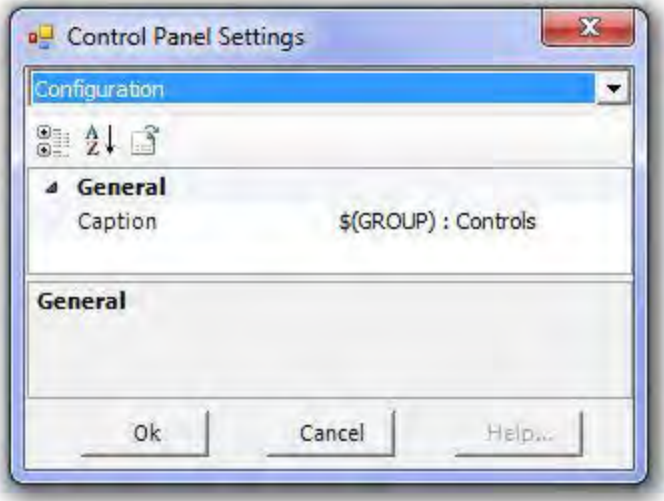
The *Group Name* of any Control or Meter Interface in a Control Panel, or any Curve in a Graph can be automatically displayed through the use of a special syntax in the title bar:

$\$(GROUP)$

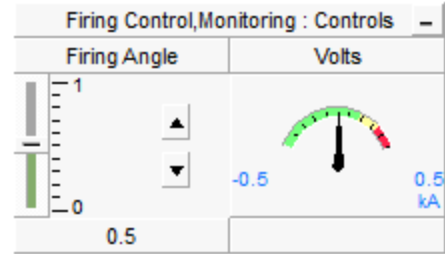
Wherever this character string is encountered within either a Graph Frame or a Control Panel **Caption** input, PSCAD will substitute the *Group Name(s)*. If the control panel or graph frame contains objects with different *Group Names*, these names will be substituted as a comma separated list.

### EXAMPLE 6-1:

A user wants to display the *Group Name* of all interfaces within a control panel. The default **Caption** parameter in the control panel settings dialog is  $\$(GROUP) : Controls$ . Two interfaces are then inserted into the control panel, one called *Firing Angle* with group name *Firing Control*, and the other named *Volts* with group name *Monitoring*. The resulting caption will appear as illustrated below:



Control Panel Settings

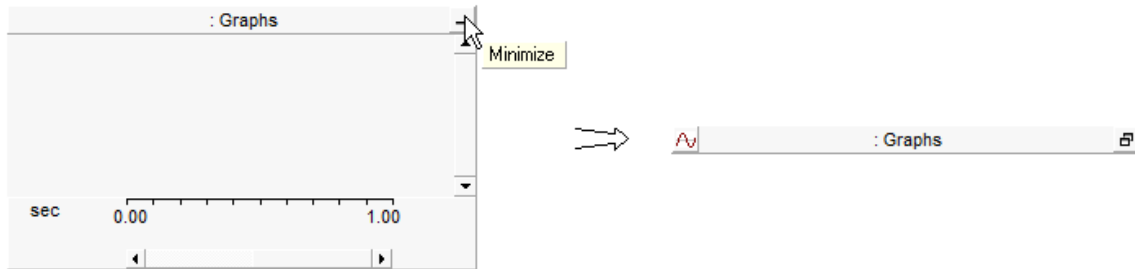


Control Panel with Objects from Two Different Groups

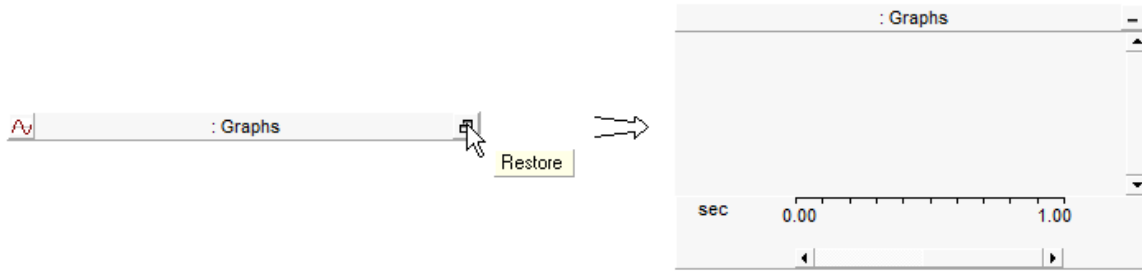
## Frame/Panel Minimization

All graph frames, control panels, polymeters and xy plot frames can be minimized in order to reduce clutter and save space when viewing data.







Minimizing a frame or panel is accomplished simply by clicking the **Minimize** button in the top right-hand corner:



The frame/panel can then be restored by clicking the **Restore** button in the right-hand corner:



Icons appear at the left of the minimized objects so that the different types of the frame/panels can be identified. A summary of these icons is given below:

-  Graph Frame
-  Control Panel
-  PolyMeter
-  PhasorMeter
-  XY Plot
-  Oscilloscope

## Moving and Organizing Minimized Frames/Panels

Once a frame/panel has been minimized, the minimized object can be moved in a similar fashion as a restored frame/panel. **Simply left-click and hold** the minimized object and drag it to wherever it needs to be placed on the canvas.

The power in using minimized frames/panels for organization is that PSCAD remembers the last position of the object before it is minimized/restored. Therefore, it is possible to 'stack' the minimized objects to optimize space, yet when any of the objects are restored, they will appear in the same position as they were before being minimized. The reverse is true as well.

## Displaying Plots and Control in Reports

Any plotting or control object (i.e. graph frames, graphs, xy plots, control panels, meters, etc.) may be copied to the Windows clipboard for inclusion in reports or other documents. Options are provided to store the objects as either a picture in *bitmap format (\*.bmp)*, or *Windows meta-file format (\*.wmf)*. Note that meta-file format is limited to the current screen size. Use bitmaps for larger graphs.

### Setting Panel Style

You can change the panel style of all graph frames, xy plots and control panels in the Workspace Options dialog, before including them in reports. See Environment for more details.

### Copy as Meta-File or Bitmap

Right-click over the plotting or control object and select **Copy <object> as Meta-File** or **Copy <object> as Bitmap** from the pop-up menu. Depending on the object selected, the **<object>** part of the command will be replaced by the object type.

You may then go directly to a report document and paste the image, while it is still in the Windows clipboard.



## Chapter 7

# Transmission Lines and Cables

Overhead transmission line and underground cable segments (or right-of-ways) are represented in two parts: The definition of the transmission segment itself, which can include the admittance/impedance data or the conductor and insulation properties, ground impedance data, and geometric position of all towers and conductors; and the interface to the rest of the electrical system, through electrical interface components (one at each end). If the transmission line is configured in *Direct Connection* mode, the interface components are not required.

Transmission segments considered to be electrically short in length (i.e. less than 15 km for a 50  $\mu$ s time step), may also be represented using an equivalent  $\pi$ -section representation. This is accomplished by either: Using the  $\pi$ -section component in the master library, where only the admittance and impedance data of the line segment is required, or by using the  $\pi$ -section equivalent component creator feature in the Line Constants Program (LCP).  $\Pi$ -sections are essentially a network of passive elements, and hence do not represent propagation delay.

**NOTE:** A 15 km line length to a 50  $\mu$ s time step is derived assuming that waves propagate through the line at the speed of light. In general however, wave propagation speed is less than the speed of light.

Using the data provided by the cross-sectional definition of the segment, the transmission lines and cables are modeled using one of two distributed (travelling wave) models:

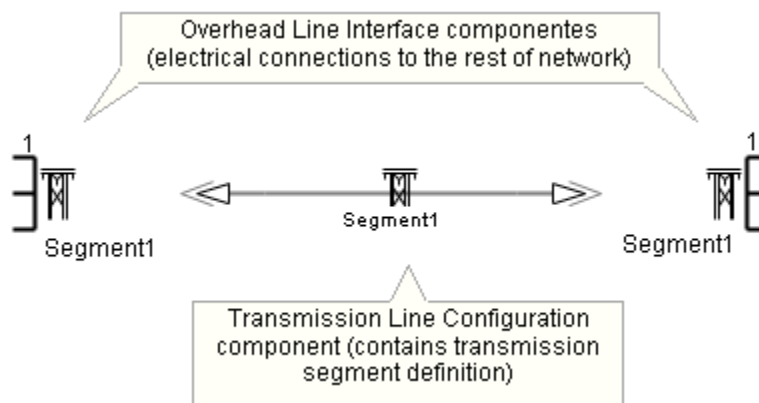
- Bergeron (Single Frequency)
- Frequency Dependent (Phase) (Multiple Frequencies)

The most accurate of these is the *Frequency Dependent (Phase)* model, which represents all frequency-dependent effects of a transmission line. When using the *Bergeron* model, impedance/admittance data can also be entered directly to define the transmission segment.

For the frequency-dependent model, detailed conductor information (i.e. segment geometry, conductor radius) must be given. For more details on the differences between models available in PSCAD, see Transmission Lines and Cables in the EMTDC section.

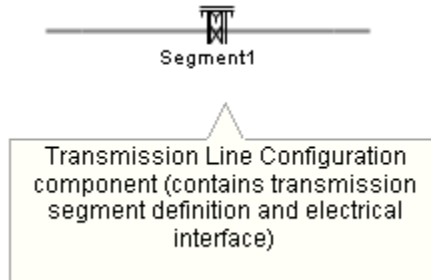
## Constructing Overhead Lines

There are two basic ways by which to construct an overhead line in PSCAD: The first is the original method (referred to as the *Remote Ends* method), which involves a Transmission Line Configuration component with two Overhead Line Interface components, representing the sending and receiving ends of the line. The purpose of the interface components is to connect the transmission line to the greater electric network. This style of transmission segment is illustrated below:



An Overhead Transmission Line (Remote Ends Method)

The second method is to use the *Direct Connection* mode, where both the interfaces and the segment properties are housed within a single component.



An Overhead Transmission Line (Direct Connection Method)

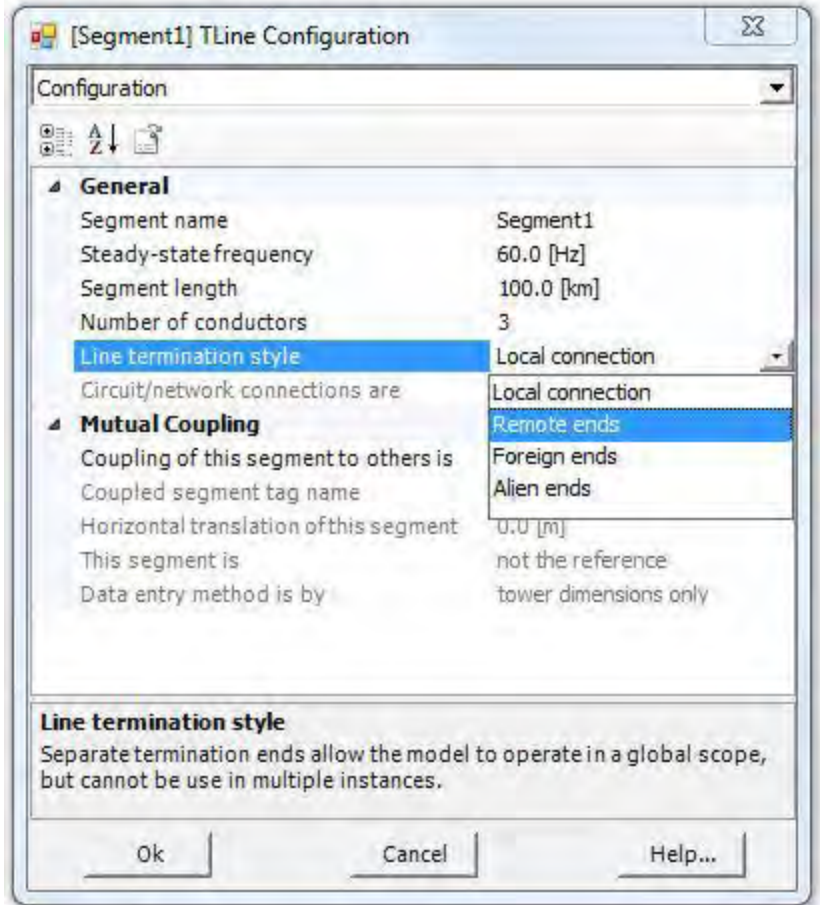
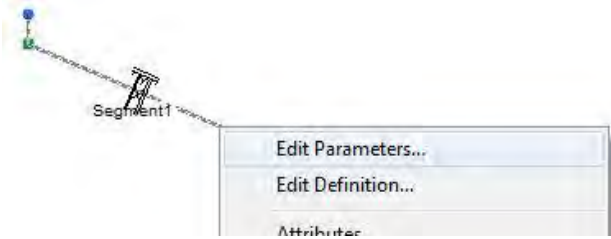
## Building an Overhead Line (/w Remote Ends)

Add one Transmission Line Configuration component and two Overhead Line Interface components to the page. The most straightforward way to add the interface components is via the *Components* tab in the ribbon control bar. See Adding Components to a Project for details.



The Transmission Line Configuration component must be added via the Component Wizard.

Next, edit the properties of the configuration component and ensure that **Line Termination Style | Remote Ends** is selected.



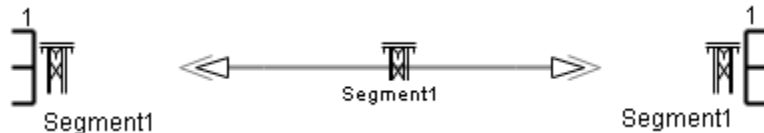
Edit the Parameters of the Transmission Line Configuration component

Set Termination Style to Remote Ends

The *Transmission Line Configuration* component and two *Overhead Line Interface* components are interlinked through input parameters. Ensure that the following steps are taken:

1. **Segment Name:** Ensure that the segment name inputs in both the *Transmission Line Configuration* component and two *Overhead Line Interface* components are identical (case sensitive).
2. **Number of Conductors:** Ensure that the number of conductors in both the *Transmission Line Configuration* component and two *Overhead Line Interface* components are identical.

When finished, you should have something similar to that shown below on your project page:



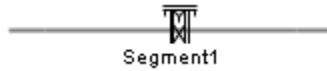
A 3-Conductor Overhead Transmission Line (Remote Ends Mode)

**NOTE:** These components do not need to be in proximity to one another. As long as they are connected by name, each component can be located anywhere in the project, including different modules.

## Building an Overhead Line (Direct Connection Style)

Add one Transmission Line Configuration component to the page. This component must be added by using the Component Wizard. Proceed as describe above.

Next, edit the parameters of the configuration component and ensure that **Line Termination Style | Direct Connection** is selected. *Direct Connection* mode is the default when the component is first created. When finished, you should have something similar to that shown below on your project page:



An Overhead Transmission Line (Direct Connection Mode)

**NOTE:** The Transmission Line Configuration component is sizable (it is in fact a type of Wire component) and may assume a variety of shapes and sizes.

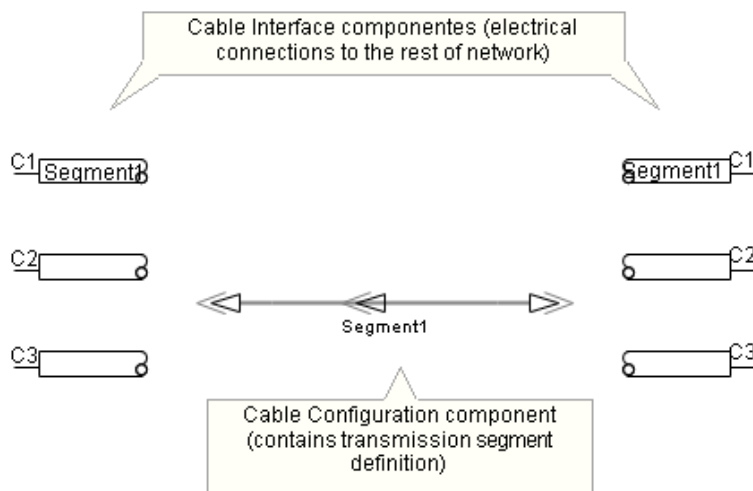
## Editing the Transmission Line Parameters

The parameters of the overhead line may be adjusted directly within the Transmission Line Configuration component. Simply right-click over the component and select **Edit Parameters...**

To edit parameters specific to towers, conductors and ground, select **Edit Definition...** instead. See Editing an Overhead Line Segment Cross-Section later in the section for details.

## Constructing Underground Cable Systems

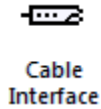
Unlike overhead lines, the *Direct Connection* method cannot be used with cable systems. This leaves just the *Remote Ends* method, which involves a Cable Configuration component with two Cable Interface components, representing the sending and receiving ends of the cable. The purpose of the interface components is to provide connection of the cable to the greater electric network. An underground cable system illustrated below:



An Underground Cable

## Building an Underground Cable System

Add one Cable Configuration component and two Cable Interface components to the page. The most straightforward way to add the interface components is via the *Components* tab in the ribbon control bar. See Adding Components to a Project for details.

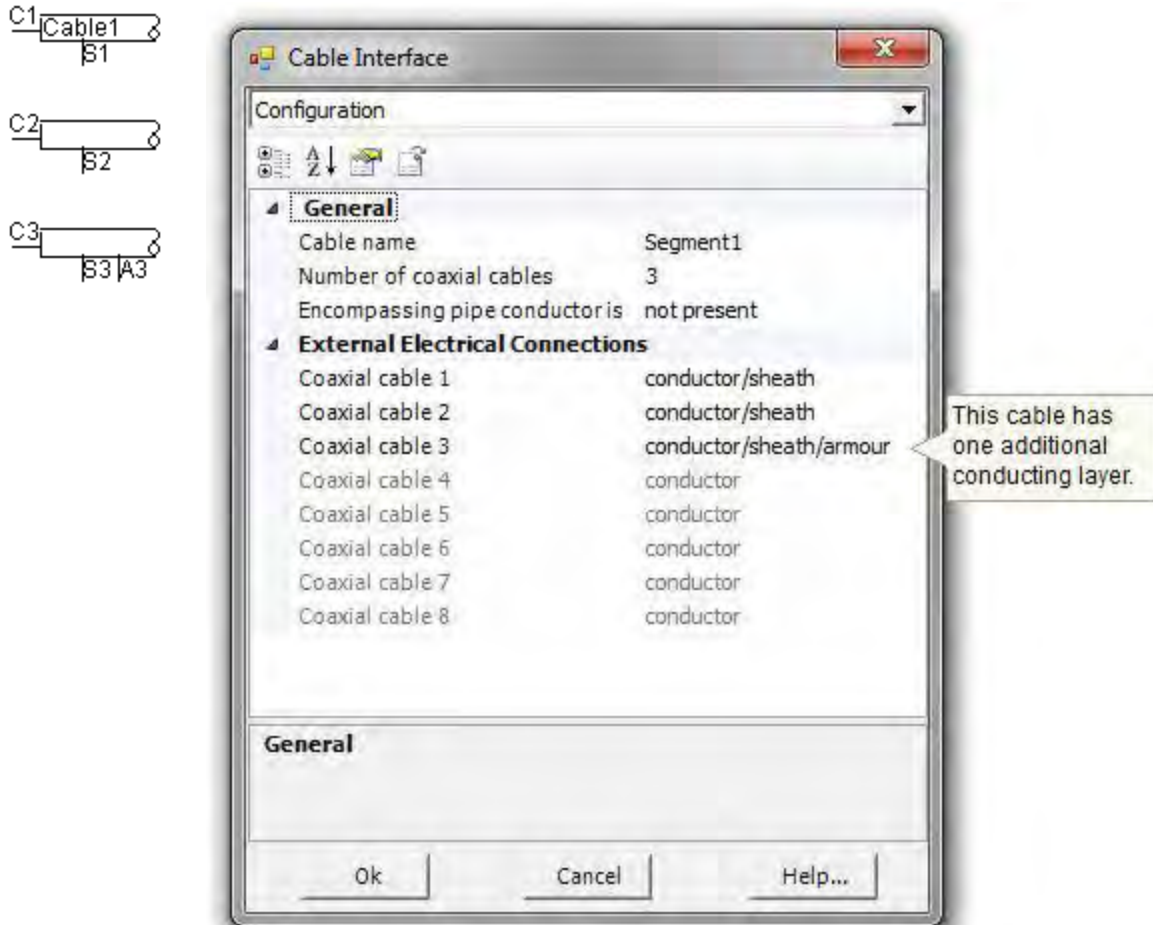


The *Cable Configuration* component must be added by using the Component Wizard.

The image shows a dialog box titled "Cable Configuration" with three tabs: "Component", "Transmission Line", and "Cable". The "Cable" tab is selected. Inside the dialog, there is a "Name:" label followed by a text input field containing "Segment 1". Below the input field is a checkbox labeled "Create Definition Only" which is currently unchecked. At the bottom of the dialog, there are three buttons: "Finish", "Reset", and "Help".

The *Cable Configuration* component and two *Cable Interface* components are interlinked through input parameters. Ensure that the following steps are taken:

1. **Segment Name:** Ensure that the segment name inputs in both the Cable Configuration component and two Cable Interface components are identical (case sensitive).
2. **Number of Cables:** The *Number of Cables* parameter in the *Cable Interface* components specifies how many individual cables there are in the system. Each of these cables however, could contain different numbers of conducting layers, and so the total number of conductors is arbitrary (this is why the *Number of Conductors* parameter is disabled in the *Cable Configuration* component). The user is required to specify the number of conducting layers for each cable in the *Cable Interface* components. This data must match what is defined within the Cable Segment Cross-Section.



Cable Interface Component

Specifying Cable Set-up in Cable Interface Parameters Dialog

**NOTE:** These components do not need to be in proximity to one another. As long as they are connected by name, each component can be located anywhere in the project, including different modules.

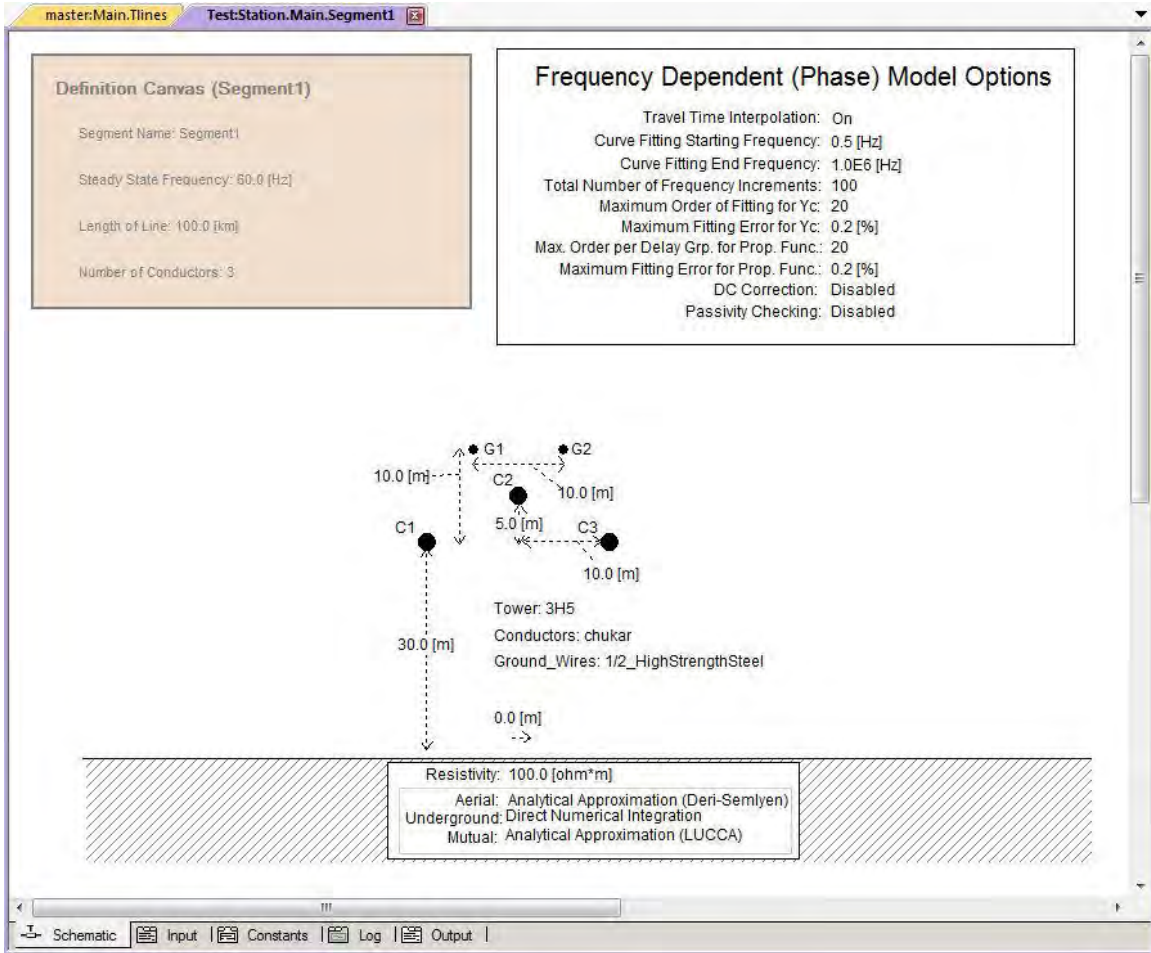
## Editing the Cable Parameters

The parameters of the cable may be adjusted directly within the Cable Configuration component. Simply right-click over the component and select **Edit Parameters...**

The properties given in the dialog are common to all elements of the cable. For information these parameters, see the Cable Configuration component help. To edit parameters specific to towers, conductors and ground, select **Edit Definition....** See Editing a Cable Segment Cross-Section later in this chapter for details.

## The Transmission Segment Definition Editor

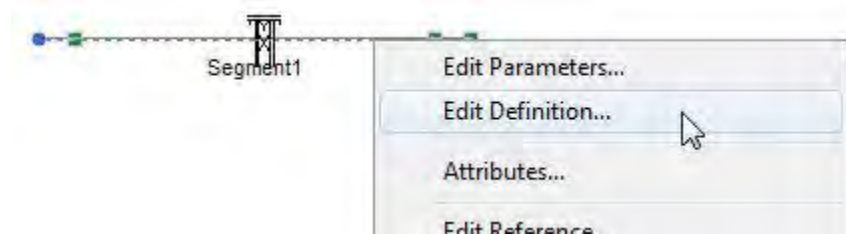
The *Transmission Segment Definition Editor* is a graphical user interface designed especially for modifying transmission line and cable system definitions. When invoked, this editor will appear within the main window, and includes its own tab sections for easy viewing of files related to segment being viewed.



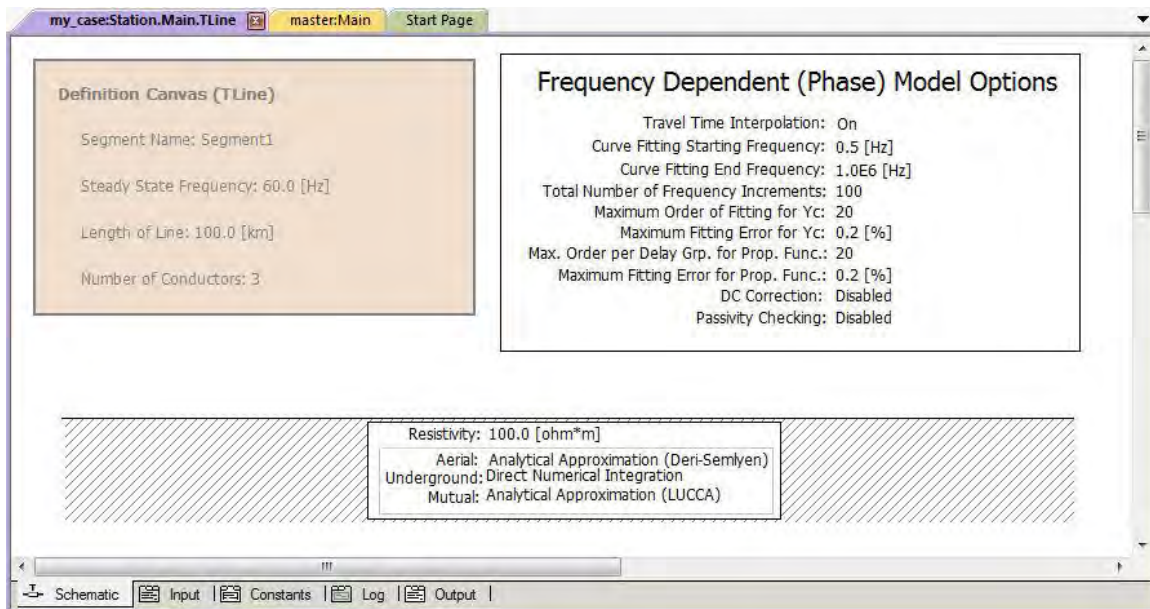
The main section tab is entitled *Schematic*, and is the default section view when the editor is invoked. The four remaining sections are for viewing files related to the segment. Note that these files will not exist unless either the segment has been solved manually, or the project has been compiled.

## Editing a Transmission Line Segment Definition

To invoke the editor, right-click over the Transmission Line Configuration component (without selecting it) and select **Edit Definition...** (or double-click) as shown below:



The editor will open within the main window. As shown below, the default view is the *Editor* section, where the transmission line is graphically defined.

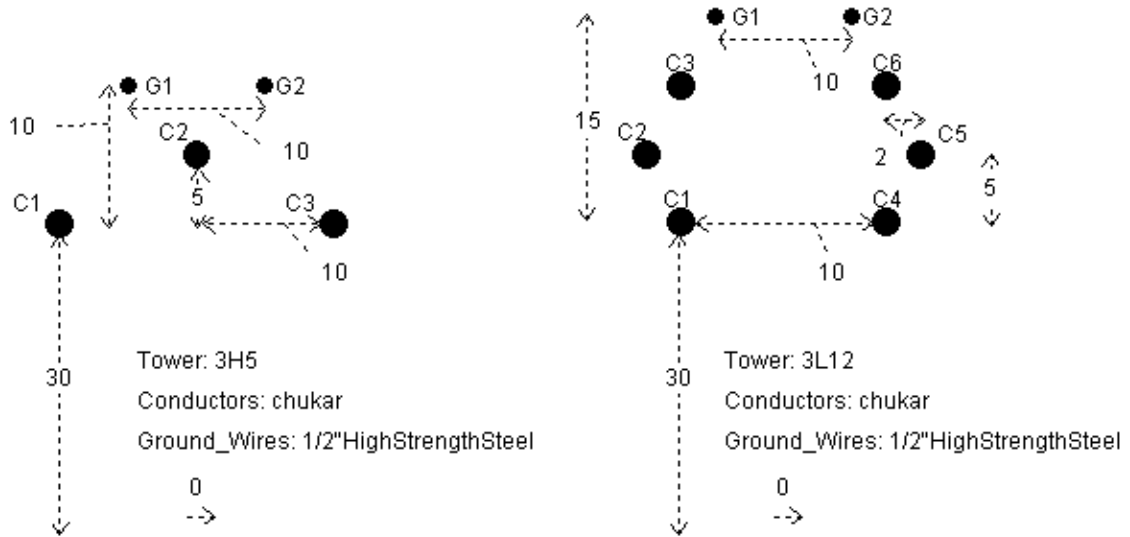


By default, the editor section will contain three graphical objects:

- **Definition Canvas (<Definition Name>):** Located in the top-left corner, this object simply displays what has been entered into the corresponding Transmission Line Configuration dialog. This object is for display only and cannot be modified from inside the editor.
- **Frequency Dependent (Phase) Model Options:** This component represents the transmission line model being used, and by default is the Frequency Dependent (Phase) model component. The parameters of this component may be edited by either a left double-click on the component (or right-click and select **Edit Parameters...**) to bring up the corresponding dialog window.
- **Entry of Ground Data:** This component (called the Ground Plane), usually located near the bottom of the editor window, represents the transmission line ground return path. The parameters of this component may be edited by either a left double-click on the component (or right-click and select **Edit Parameters...**) to bring up the corresponding dialog window.

A fourth object required is the definition of the transmission line itself. This definition can be a geometrical cross-section of the Transmission Line Tower component, or can be a Manual Entry of Admittance/Impedance Data (Bergeron model only). In either case, this is left for the user to add manually.

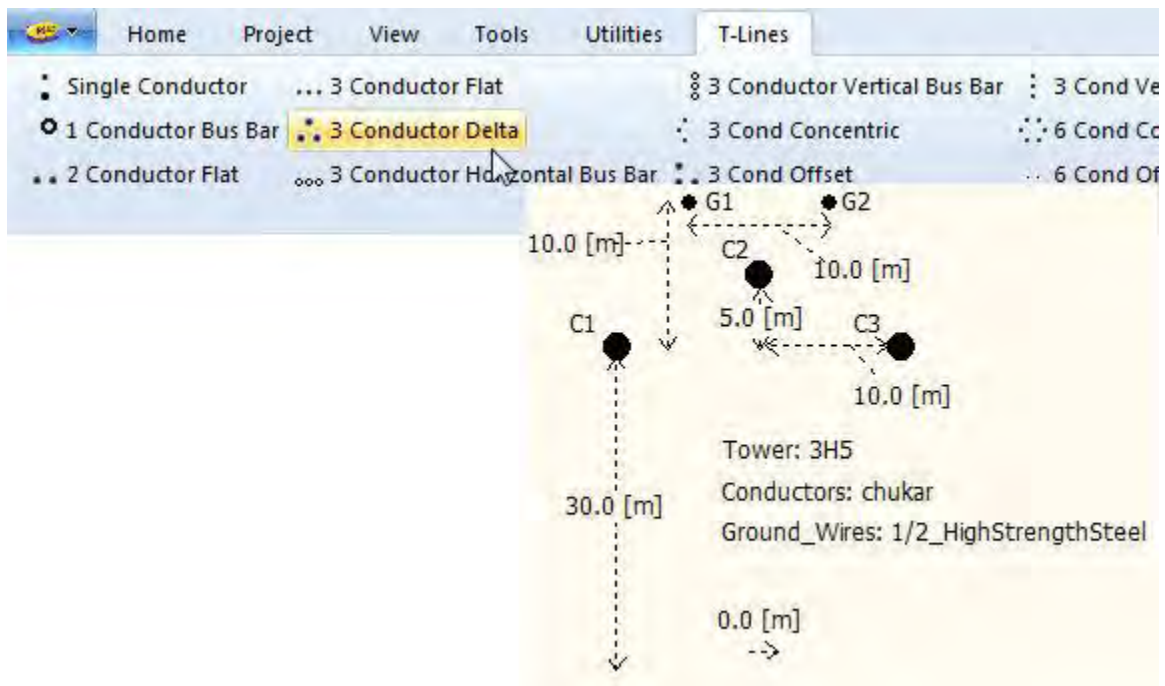




Transmission Line Tower Components

## Adding a Tower Component

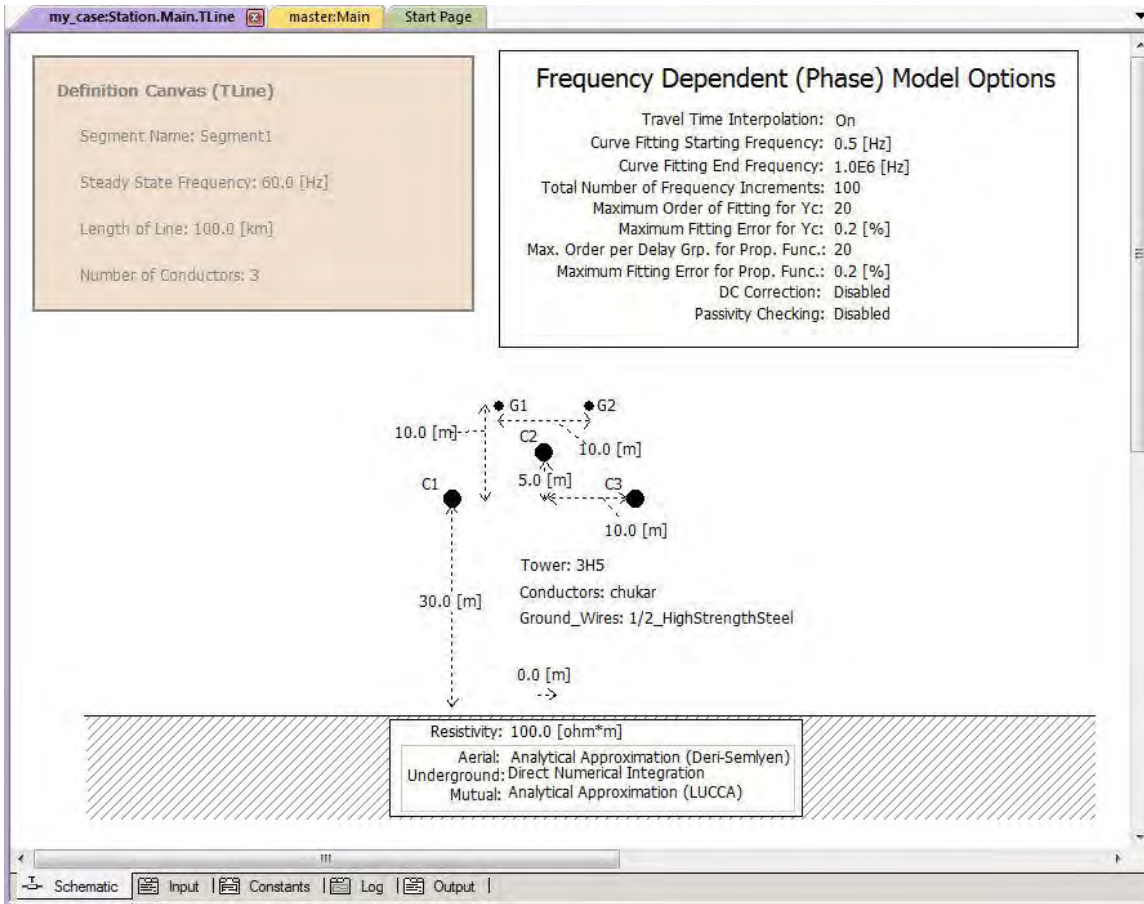
Tower components can be added to the editor in a couple different ways. The easiest and most straightforward way is to use the ribbon control bar. Simply select one of the available towers and place it on the Schematic canvas. See Adding Components to a Project for more details.



Another way is to use the context menu. On the Schematic tab, move the mouse pointer over a blank area of the window. Right-click and select **Add Tower Cross-Section**. A sub-menu will appear containing a list of all loaded library projects. Each of these will contain a list of all transmission line tower components available in that particular library. Select a tower and it will be automatically added.

You can also copy and paste tower components directly from from the master library. Open the library in Schematic view. Select a tower component, then right-click on the component and select **Copy** (or press **Ctrl + c**). Open the *Editor* tab, right-click over a blank area and select **Paste** (or press **Ctrl + v**).

When finished, you should have something similar to that shown below:



**NOTE:** The graphical location of the tower component does not affect the results. However, the tower (or towers) should be positioned to allow for ease in readability. That is, directly on top of the Ground Plane component.

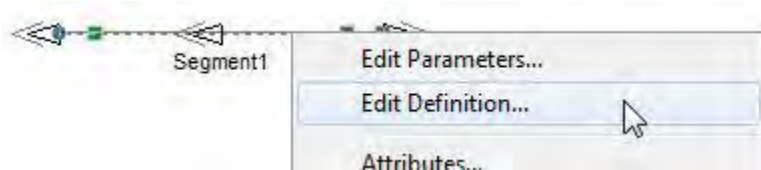
Multiple towers may be added to a single configuration. Just remember to ensure that the conductors are numbered properly within each of the tower components, and that the x-positions of the new towers in the corridor are adjusted. Also, any conductors added by the additional towers must be reflected in the corresponding Transmission Line Interface components (if in *Remote Ends* mode).

## Editing Tower Parameters

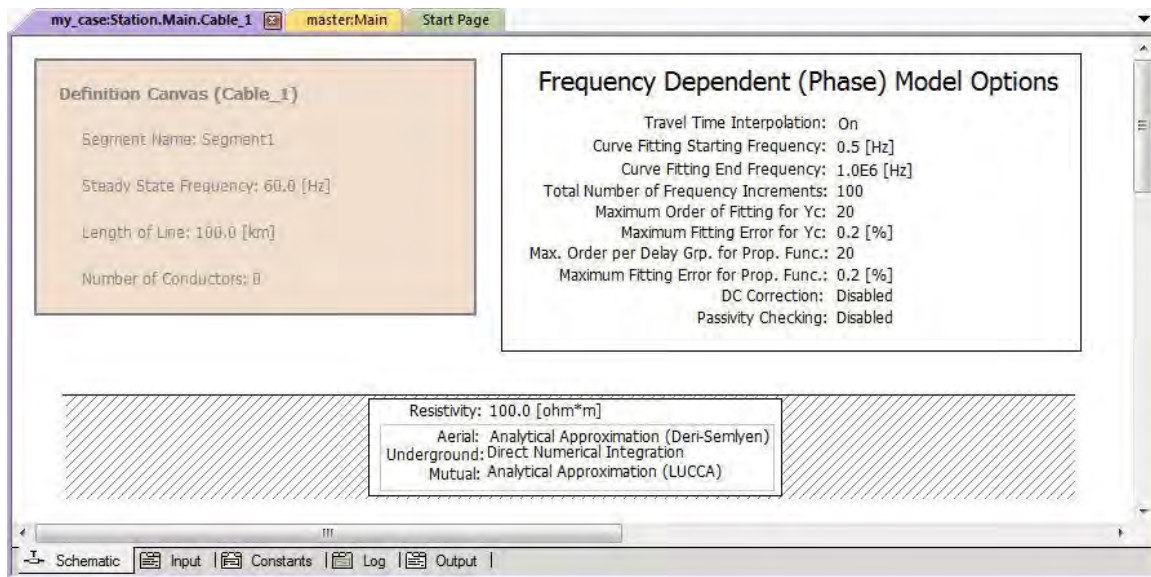
Tower parameters can be edited through the corresponding tower parameters dialog window. Right-click over the tower component (without selecting it) and select **Edit Parameters...** to access this dialog.

## Editing a Cable Segment Definition

To invoke the editor, right-click over the Cable Configuration component (without selecting it) and select **Edit Definition...** (or double-click) as shown below:



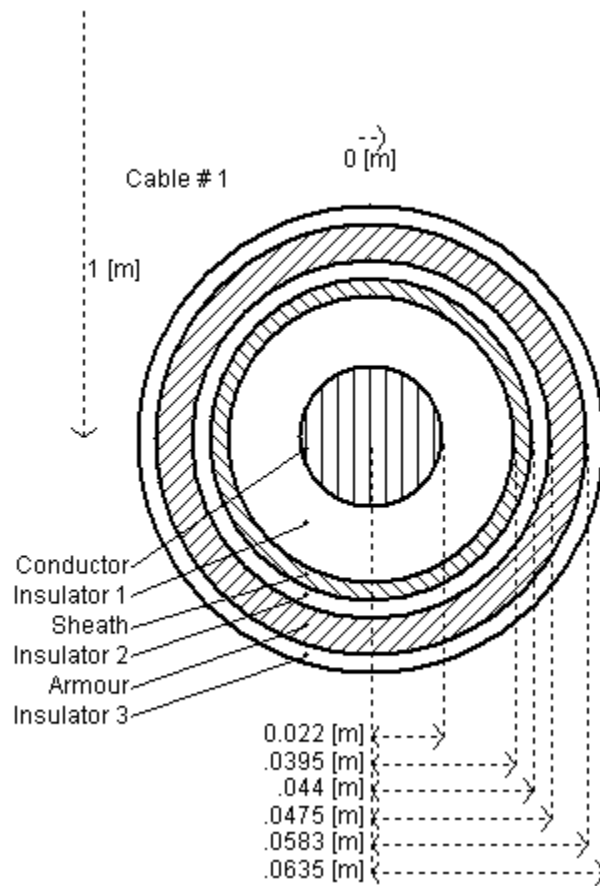
In either case, the editor will then open within the main window. As shown below, the default view is the *Editor* section, where the cable system is graphically defined.



By default, the Editor section will contain three graphical objects:

- **Definition Canvas (<Definition Name>):** Located in the top-left corner, this object simply displays what has been entered into the corresponding Cable Configuration dialog. This object is for display only and cannot be modified from inside the editor.
- **Frequency Dependent (Phase) Model Options:** This component represents the transmission line model being used, and by default is the Frequency Dependent (Phase) model component. The parameters of this component may be edited by either a left double-click on the component (or right-click and select **Edit Parameters...**) to bring up the corresponding dialog window.
- **Entry of Ground Data:** This component, usually located near the bottom of the *Editor* window, represents the transmission line ground return path. The parameters of this component may be edited by either a left double-click on the component (or right-click and select **Edit Parameters...**) to bring up the corresponding dialog window.

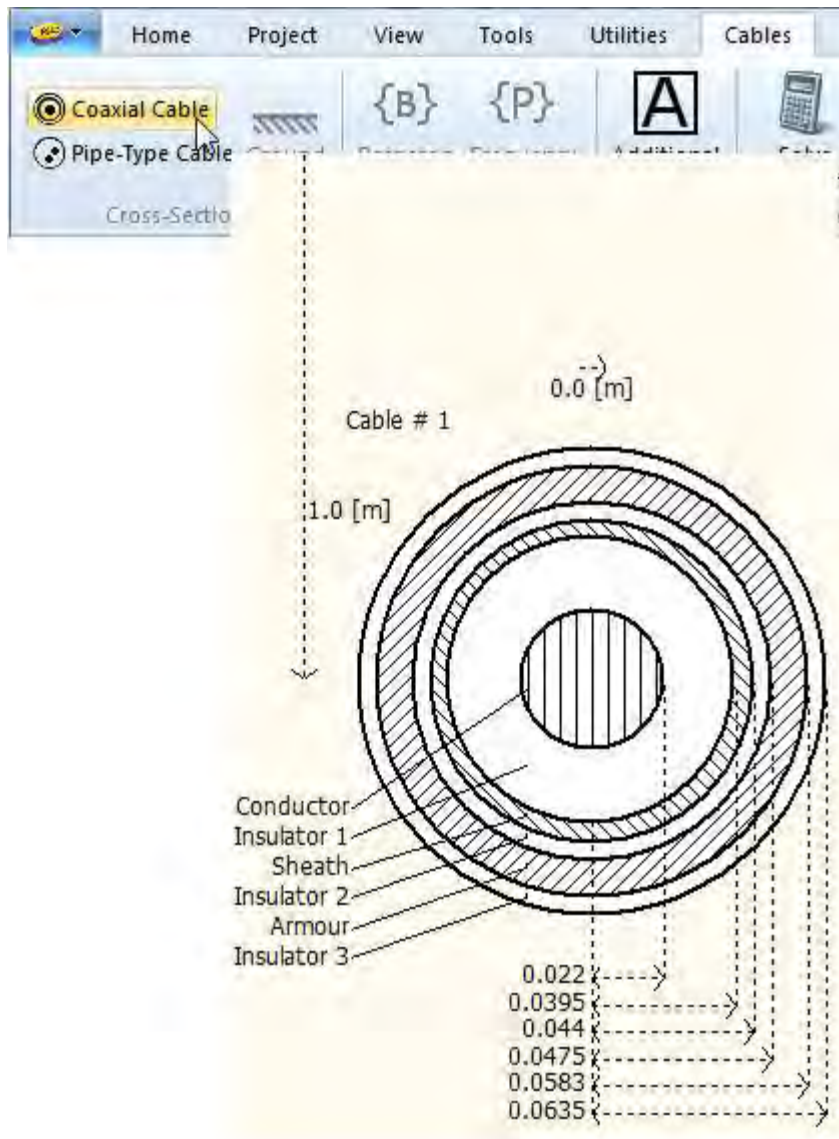
A fourth object required is the definition of the cable itself. This definition is a geometrical cross-section of the cable called a Cable Cross-Section. The user must add this manually.



Cable Cross-Section Component

## Adding a Cable Cross-Section Component

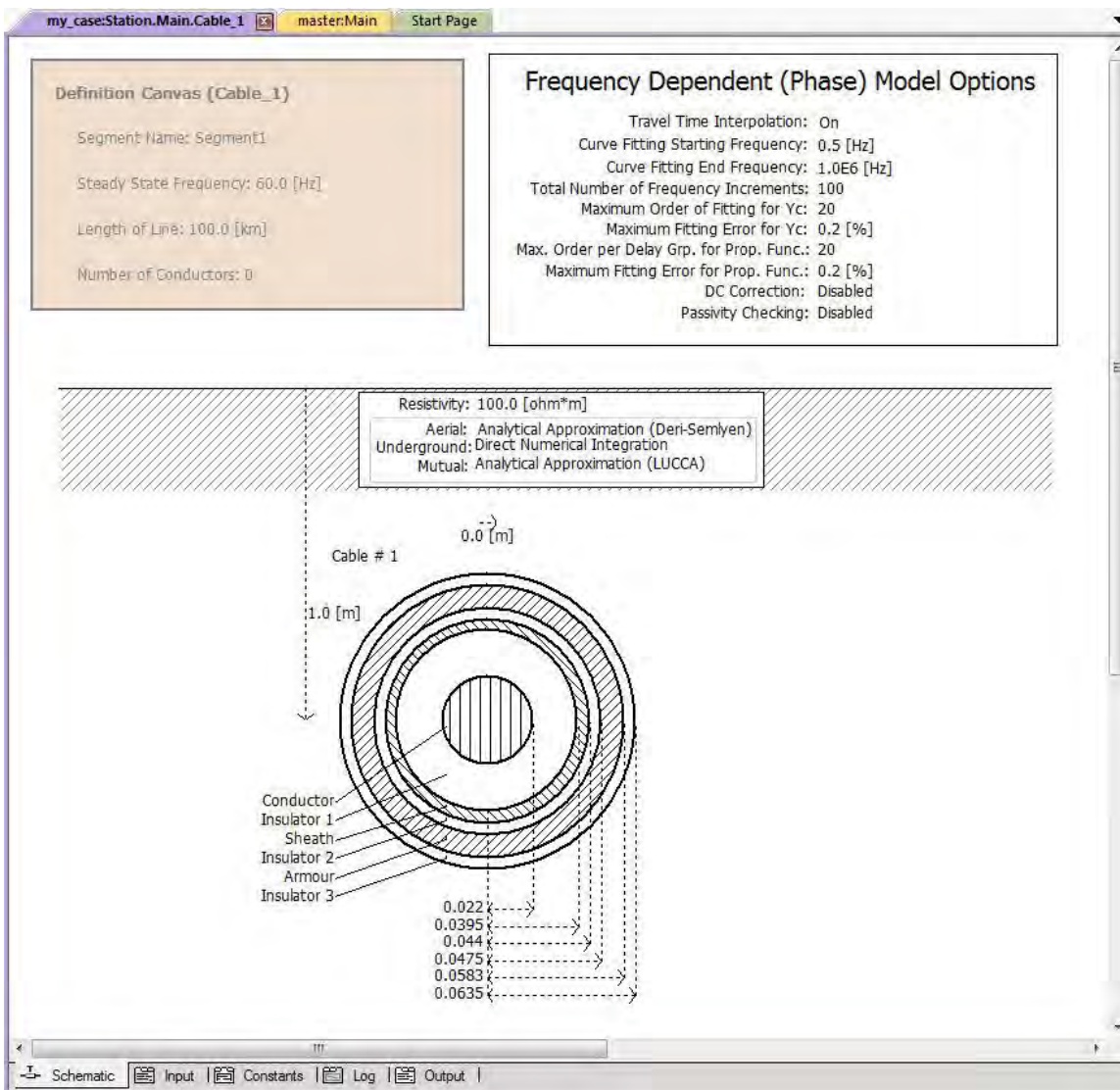
Cable Cross-Section components can be added to the editor in a couple different ways. The easiest and most straightforward way is to use the ribbon control bar. Simply select one of the available cross-sections and place it on the Schematic canvas. See Adding Components to a Project for more details.



Another way is to use the context menu. On the Schematic tab, move the mouse pointer over a blank area of the window. Right-click and select **Add Cable Cross-Section**. A sub-menu will appear containing a list of all loaded library projects. Each of these will contain a list of all cable components available in that particular library. Select a cable and it will be automatically added.

You can also copy and paste cable components directly from any library project. Open the library in Schematic view. Select a cable cross-section, then right-click on the component and select **Copy** (or press **Ctrl + c**). Open the *Editor* (tab), right-click over a blank area and select **Paste** (or press **Ctrl + v**).

When finished, you should have something similar to that shown below:



**NOTE:** The location of the Cable Cross-Section component does not affect the results. However, the cross-section (or cross-sections) should be positioned to allow for ease in readability. That is, directly below the Ground Plane component.

Multiple cables may be added to a single configuration. Just remember to ensure that the cables are numbered properly, and that the xy-positions of the new cross-sections in the corridor are adjusted. Also, any conductors added by the additional cables must be reflected in the corresponding Cable Interface components.

## Editing Cross-Section Parameters

Cable cross-section parameters can be edited through the corresponding cross-section parameters dialog window. Right-click over the cross-section component (without selecting it) and select **Edit Parameters...** to access this dialog.

## Selecting the Proper Line Model

There are three types of distributed (i.e. travelling wave) transmission models that may be selected to represent your transmission corridor: The Bergeron model, the Frequency-Dependent (Mode) model, and the Frequency-Dependent (Phase) model. These models exist as components in the master library and each include adjustable properties. The requirements for your study will determine which of the three models is suitable.

### Bergeron Model Options

Travel Time Interpolation: On  
 Reflectionless Line (ie Infinite Length): No

### Frequency Dependent (Phase) Model Options

Travel Time Interpolation: On  
 Curve Fitting Starting Frequency: 0.5 [Hz]  
 Curve Fitting End Frequency: 1.0E6 [Hz]  
 Maximum Order of Fitting for YSurge: 20  
 Maximum Order of Fitting for Prop. Func.: 20  
 Maximum Fitting Error for YSurge: 2 [%]  
 Maximum Fitting Error for Prop. Func.: 2 [%]

### Frequency Dependent (Mode) Model Options

Travel Time Interpolation: On  
 Curve Fitting Starting Frequency: 0.5 [Hz]  
 Curve Fitting End Frequency: 1.0E6 [Hz]  
 Maximum Order of Fitting for ZSurge: 20  
 Maximum Order of Fitting for Prop. Func.: 20  
 Maximum Fitting Error for ZSurge: 2 [%]  
 Maximum Fitting Error for Prop. Func.: 2 [%]

## The Bergeron Model

The Bergeron model represents the  $L$  and  $C$  elements of a  $\pi$ -section in a distributed manner (not using lumped parameters like  $\pi$ -sections). It is accurate only at the specified frequency and is suitable for studies where the specified frequency load-flow is most important (e.g. relay studies).

When using the Bergeron model, it is not always necessary to use a tower component to represent a transmission line. If you are modeling a three-phase system, then you can enter the line data, in admittance or impedance format, directly by substituting the Manual Entry of Y, Z component.

## Manual Entry of Y,Z

```
+ve Sequence R: .6861e-7 [pu/m]
+ve Sequence XL: .951e-6 [pu/m]
+ve Sequence XC: .571e6 [pu*m]
  0 Sequence R: .7175e-6 [pu/m]
  0 Sequence XL: .251e-5 [pu/m]
  0 Sequence XC: .793e6 [pu*m]
```

This component can be added just as you were adding a tower component. Keep in mind that this component substitutes for the tower component, and you still need the Bergeron model present in your transmission line configuration.

## The Frequency-Dependent (Mode) Model

The Frequency-Dependent (Mode) model represents the frequency dependence of all parameters (not just at the specified frequency as in the Bergeron model). This model uses modal techniques to solve the line constants and assumes a constant transformation. It is therefore only accurate for systems of ideally transposed conductors (or 2 conductor horizontal configurations) or single conductors.

## The Frequency-Dependent (Phase) Model

The Frequency-Dependent (Phase) model also represents the frequency dependence of all parameters as in the 'Mode' model above. However, the *Frequency Dependent (Phase)* model circumvents the constant transformation problem by direct formulation in the phase domain. It is therefore accurate for all transmission configurations, including unbalanced line geometry.

This model should always be the model of choice, unless another model is chosen for a specific reason. This model is the most advanced and accurate time domain line model in the world!

**NOTE:** For further technical detail, or more information on choosing a line model, see Chapter 9 – Transmission Lines and Cables in the EMTDC Manual.

## Adding a Line Model

Line model components can be added to the editor in a couple different ways. The easiest and most straightforward way is to use the ribbon control bar. Simply select one of the available models and place it on the Schematic canvas. See Adding Components to a Project for more details.

Another way is to use the context menu. On the Schematic tab, move the mouse pointer over a blank area of the window. Right-click and select **Choose Model**. A sub-menu will appear containing a list of all model components available in the master library. Select a model and it will automatically be added.

You can also copy and paste model components directly from any library project. Open the library in Schematic view and select a line model component, right-click on the component and select **Copy** (or press **Ctrl + c**). Open the *Editor* section (tab), right-click over a blank area and select **Paste** (or press **Ctrl + v**).

## Editing Line Model Parameters

Line model parameters can be edited through the corresponding model parameters dialog window. Right-click over the line model component (without selecting it) and select **Edit Parameters...** to access this dialog.



## Multiple Instances of Transmission Segments

From the perspective of the PSCAD application itself, transmission segments are not unlike modules: They possess a canvas, input parameters, and a definition. It would seem then that transmission segments could also be multiple instanced, like their module component counterparts – this is indeed possible. However, there are some subtle differences between the two types of component that tend to complicate matters.

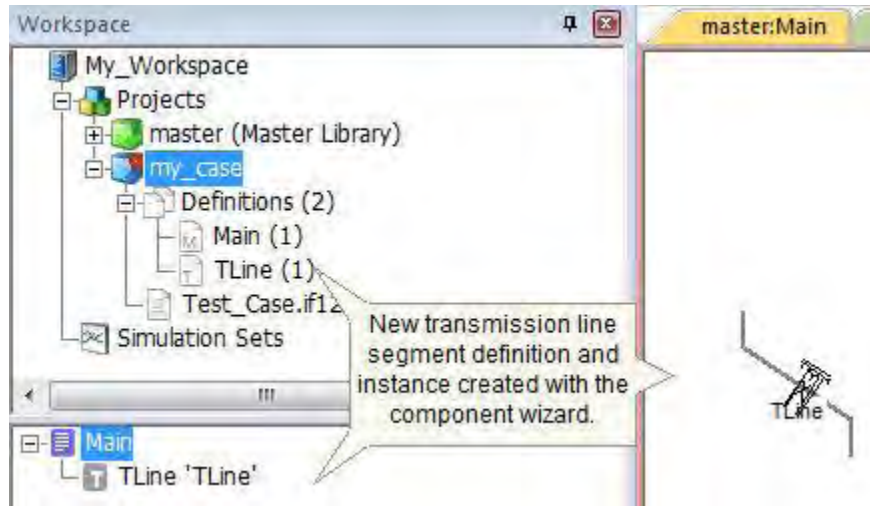
Modules are compiled only once no matter how many times the component has been instantiated in the project. This is because a module component does not depend on any of its input parameter (or connection port) values, to define its definition. A transmission segment however, is highly dependent on its length (i.e. the length affects the definition), and since this is an instance-based parameter, each transmission segment instance must be compiled separately.

Surely the segment length could easily be made part of the definition, instead of existing as an instance-based parameter, but then a new segment definition would need to be created whenever the length changed. This would mean that for the same tower configuration, a new definition would need to be created if one segment was *15.0 km* and the other was *15.001 km*. It seems to make more sense to instead allow a single definition to represent a single tower configuration, and this configuration can be multiply instanced.

The length therefore remains as an instance-based parameter, and hence each individual segment instance must be solved uniquely – even if they are based on the same definition. There is however a way in which to circumvent this: See the section called Multiple Instances without Solving Multiple Times below for details.

## Creating a Transmission Segment Definition

The manner in which to construct a transmission segment – be it an overhead line or underground cable – is explained in detail in the previous sections. The component wizard creates a segment definition and also the first instance of the definition. In fact, once you create a new segment, you will find its definition listed in the workspace window, along with its new instance as part of the project module hierarchy..



## Multiple Instances of the Same Segment Definition

Once a new transmission line or cable has been created, you may modify it in the same manner as you would a module. From this point forward, any instance of the transmission segment can be instantiated by simply copying and pasting the component on the Schematic canvas.



As discussed above, the final step in multiply instantiating a transmission segment is that it must be given a unique *Segment Name*. By default when the first instance of a segment is created, its instance name is the same as its definition name. This is okay if there is only one instance; however, if more than one instance is present based on the same definition, the segment name must be distinct.

The segment name is a component input parameter – so to modify it, simply open the parameters dialog (i.e. right-click and select **Edit Parameters...**) and change the name.

If four instances from above are given unique names, they would appear as follows on the Circuit canvas:



When these line instances (all based on the definition *TLine*) are connected as a valid circuit in the project, each line will be solved individually when the project is compiled.

## Multiple Instances without Solving Multiple Times

It is possible to instantiate a transmission segment many times, while ensuring it is only solved once. This is desirable for example, if the transmission line or cable segments possess the same cross-sectional configuration (i.e. definition) and the same length. A practical situation would be perhaps a cross-bonded cable or transposed aerial line, where the transposed segments are all equal.

The above is accomplished by 'wrapping' the segment within a module canvas. This essentially makes the transmission line or cable object part of the definition of its parent module. Since a module definition is only solved once, so shall it be for the transmission line. EMTDC will simply access the same Segment Constants (\*.tlo/\*.clo) file multiple times.

### EXAMPLE 8-1:

A user wishes to model a 3-phase, transposed transmission line, where each segment is of equal cross-sectional configuration, and the same length. The system has 6 segments in total.

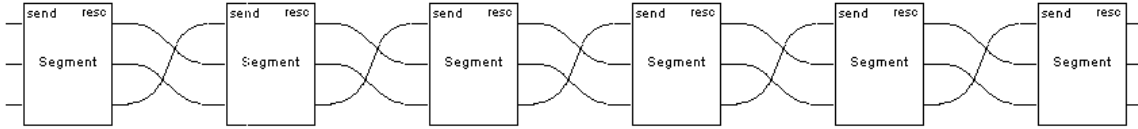
1. First create a new module component as described in *Creating a New Component or Module* in Chapter 5. Make sure it has a 3, 1-phase port connection nodes on both the left and the right side of the component graphic.
2. Inside the Circuit canvas of the newly created module, create a new directly connected transmission line segment as discussed above in the section *Constructing Overhead Transmission Lines*.



New Module Graphic

Transmission Segment Wire on Module Circuit Canvas (Inside New Module)

3. Instantiate the new module component 6 times and then place transpositions between them.



Although the above system contains six calls to the transmission line within the module, it is only actually solved once, as it is part of the module definition.

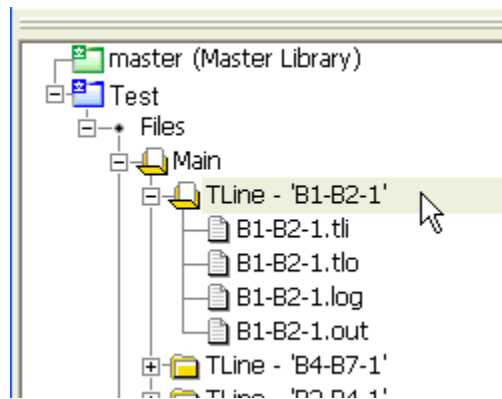
## Line Constants Files

Whenever a project containing a transmission line or a cable segment is compiled (or the line constants are solved manually), text files involved in the calculation are created. PSCAD uses information extracted from the various components involved with the transmission system, and constructs a Segment Input (\*.tli) file. This file is then used as input by the Line Constants Program (LCP) – a separate executable supplied with your PSCAD software.

Depending on the transmission segment type (i.e. overhead line or underground cable) and the actual line model used, the LCP creates three main files. These are:

- Segment Constants File (<segment name>.tlo/.clo)
- Segment Log File (<segment name>.log)
- Segment Output File (<segment name>.out)

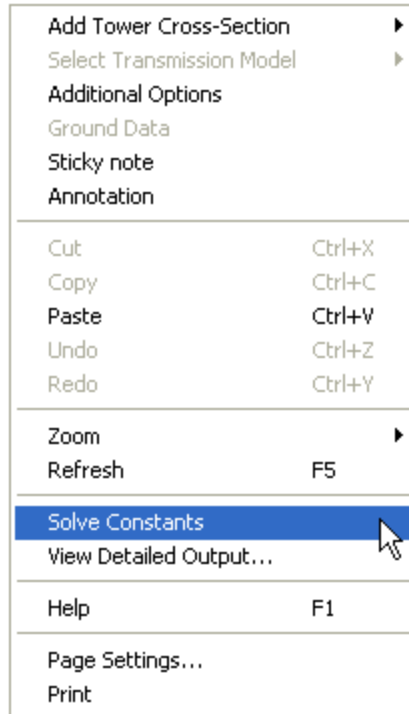
These files are located in the project temporary directory (<project name>.emt) and may be viewed either directly within the Transmission Segment Definition Editor (by clicking the corresponding tab window), or in the Files tree in the workspace.



## Solving the Line Constants Manually

To manually solve a specific transmission line or cable segment in your project, first open the Transmission Segment Definition Editor (see previous section for instructions if needed).

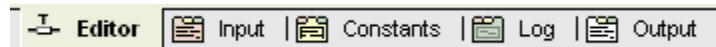
Right-click over a blank area of the canvas and select **Solve Constants**:



**NOTE:** When manually solving a transmission segment, check the Log file (by clicking the **Log** tab) to ensure that the line was properly solved. Along with other logged text, any errors issued by the LCP will be displayed in this file.

## Viewing Line Constants Files

To view any of the LCP files, first manually solve the line, as described in the previous section. Then, simply click the corresponding tab, located near the bottom of the window.



The tab window viewers specified in this bar correspond as follows:

- **Editor:** Transmission Segment Definition Editor
- **Input:** Segment Input (<segment name>.tli/.cli) file
- **Constants:** Segment Constants (<segment name>.tlo/.clo) file
- **Log:** Segment Log (<segment name>.log) file
- **Output:** Segment Output (<segment name>.out) file

## Segment Input File

The segment input file is automatically constructed when the project is compiled, or when the line constants are solved manually (see Solving the Transmission Segment Constants Manually above). This file is used as input to the LCP; it is automatically generated and cannot be modified.

The input file is pieced together using information taken from:

1. **Configuration Component:** Depending on whether you are modeling a transmission line or a cable segment, the first part of this file is composed of input parameters taken directly from the corresponding instance of the Transmission Line or Cable Configuration component, as shown below for a transmission line:

```
Line Summary:

{

Line Name = FLAT230

Line Length = 100.0

Steady State Frequency = 60.0

Number of Conductors = 3

}
```

2. **Tower or Cross Section Component:** Important parameters such as conductor data and ground data, dimensions, etc. are extracted from the Model-Data segments of whatever tower or cable cross-section components exist in the editor. These are placed in the input file as well and represent the bulk of the file:

```
Line Constants Tower:

{

Name = H-Frame-3H4

Circuit = 1

{

Transposed = 0

Conductors = 3

Conductor Phase Information = 1 2 3

Radius = 0.0203454

DCResistance = 0.03206

ShuntConductance = 1.0e-011

P1 = -10.0 30.0

P2 = 0.0 30.0

P3 = 10.0 30.0

Sag = 10.0

}
```

```

Sub-ConductorsPerBundle = 2
{
  BundleSpacing = 0.4572
}
}
GroundWires = 2
{
  Eliminate Ground Wires = 1
  Radius = 0.0055245
  DCResistance = 2.8645
  P1 = -5.0 35.0
  P2 = 5.0 35.0
  Sag = 10.0
}
}
Line Constants Ground Data:
{
  GroundResistivity = 100.0
  GroundPermeability = 1.0
  EarthImpedanceFormula = 0
}

```

3. **Model Component:** Some important information will appear in the input file from whatever model is being used. In this case, the it is the Frequency Dependent (Phase) model:

```

Frequency Dep. (Phase) Model Options:
{
  Interpolate Travel Times = 1
  Infinite Line Length = 0
  Curve Fitting Start Frequency = 0.5
  Curve Fitting End Frequency = 1000000.0
  Maximum # of Poles for Surge Admittance Fit = 20
}

```

```

Maximum # of Poles for Attenuation Constant Fit = 20

Maximum Fitting Error (%) for Surge Admittance = 0.2

Maximum Fitting Error (%) for Attenuation Constant = 2.0

Weighting Factor 1 = 100.0

Weighting Factor 2 = 1000.0

Weighting Factor 3 = 1.0

Write Detailed Output Files = 0

}

```

## Constants File (\*.tlo\\*.clo)

The constants (\*.tlo\\*.clo) file is a LCP output file, which is used as input for EMTDC to set-up the transmission line interface to the greater electric network. This file contains all the information necessary for EMTDC to represent the transmission segment in the time domain, and is for the most part, of no concern to the user.

By default, this file is automatically generated by the LCP, and will overwrite any file of the same name that already exists in the temporary folder if the corresponding segment has been modified. However, the user may also opt to provide their own custom file. See Overhead Line Configuration or Cable Configuration for more details.

## Log File

The log file is a collection of messages sequentially output by the LCP while it runs its course. This is an important file when initially debugging and tuning a transmission line or cable segment. It is recommended that user scan through this file to ensure that no problems occurred while the line constants were being solved.

If question arise regarding the output in this file, please contact the PSCAD Support Desk ([support@pscad.com](mailto:support@pscad.com)).

## Output File

Output files are created by the LCP and are used to display important transmission segment data in a convenient format for the user. This can include impedance and admittance matrix information, sequence data and travel times.

The format of the output file will change slightly depending on the model used. See the Additional Options component for details on formatting data in this file.

## Phase Data

The phase data section displays relevant line constants at a specific frequency, where all quantities are in the phase domain. The term 'phase domain' refers to quantities that have not been transformed into the modal domain. Phase data represents the 'real life' characteristics of the line.

## Series Impedance Matrix (Z)

This data represents the system series impedance matrix  $\mathbf{Z}$  per-unit length ( $\Omega/\text{m}$ ). The diagonal terms represent the self impedances of each conductor, whereas the off-diagonals are the mutual impedances between the respective conductors. The dimension of the matrix depends on the final number of equivalent conductors/ground wires in the system  $N$ :

$$Z = \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1N} \\ Z_{21} & Z_{22} & \cdots & Z_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{N1} & Z_{N2} & \cdots & Z_{NN} \end{bmatrix} \quad [\Omega/m]$$

Series Impedance Matrix

All elements in this matrix are complex and are given in Cartesian format:

$$R_{ij}, X_{ij} \rightarrow Z_{ij} = R_{ij} + j \cdot X_{ij}$$

The positions of elements in this matrix are dependent on the manner in which the conductors/ground wires have been numbered. The type of ideal transposition that has been selected will also affect this matrix.

## Shunt Admittance Matrix (Y)

This data represents the system shunt admittance matrix  $Y$  per-unit length (S/m). The diagonal terms represent the self admittances of each conductor, whereas the off-diagonals are the mutual admittances between the respective conductors. The dimension of the matrix depends on the final number of equivalent conductors/ground wires in the system  $N$ :

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1N} \\ Y_{21} & Y_{22} & \cdots & Y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N1} & Y_{N2} & \cdots & Y_{NN} \end{bmatrix} \quad [S/m]$$

Shunt Admittance Matrix

All elements in this matrix are complex and are given in Cartesian format:

$$G_{ij}, B_{ij} \rightarrow Y_{ij} = G_{ij} + j \cdot B_{ij}$$

The positions of elements in this matrix are dependent on the manner in which the conductors/ground wires have been numbered. The type of ideal transposition that has been selected will also affect this matrix.

## Long-Line Corrected Series Impedance Matrix (ZLL)

This data represents the long-line corrected, series impedance matrix  $Z$  ( $\Omega$ ). This matrix is the impedance of the entire line length, where all quantities have been passed through a correction algorithm to account for the electrical effects of long line distances.



The long-line corrected quantities have a specific use: They should be used whenever a single  $\pi$ -section equivalent is being derived to represent the entire line length at a specific frequency. This data should not be used to define time domain travelling wave models.

## Long-Line Corrected Shunt Admittance Matrix (YLL)

This data represents the long-line corrected, shunt admittance matrix  $Y(S)$ . This matrix is the admittance of the entire line length, where all quantities have been passed through a correction algorithm to account for the electrical effects of long line distances.

The long-line corrected quantities have a specific use: They should be used whenever a single  $\pi$ -section equivalent is being derived to represent the entire line length at a specific frequency. This data should not be used to define time domain travelling wave models.

## Sequence Data

The sequence data section displays relevant line parameters at a specific frequency, where all quantities are sequence quantities. The sequence data is calculated directly, through the use of a sequence transform matrix  $T$ :

$$T = \frac{1}{2\sqrt{3}} \begin{bmatrix} 2 & 2 & 2 \\ 2 & -1-j\sqrt{3} & -1+j\sqrt{3} \\ 2 & -1+j\sqrt{3} & -1-j\sqrt{3} \end{bmatrix}$$

Transformation Matrix

## Sequence Impedance Matrix ( $Z_{sq}$ )

This data represents the system sequence impedance matrix  $Z_{sq}$  per-unit length ( $\Omega/m$ ).  $Z_{sq}$  is derived directly from the series impedance matrix  $Z$  and the sequence transform matrix  $T$  (both described above) as follows:

$$Z_{sq} = T^{-t} \cdot Z \cdot T$$

If all 3-phase circuits in the  $Z$  matrix are ideally transposed, then the sequence impedance matrix  $Z_{sq}$  will be a diagonal matrix, where the diagonal terms are the equivalent zero, positive and negative sequence components.

For a single 3-phase, ideally transposed circuit, the sequence impedance matrix appears as follows:

$$Z_{sq} = \begin{bmatrix} Z_0 & 0 & 0 \\ 0 & Z_+ & 0 \\ 0 & 0 & Z_- \end{bmatrix}$$

Sequence Impedance Matrix (Single, Balanced 3-Phase Circuit)

In the case of two, ideally transposed 3-phase circuits, the sequence impedance matrix will appear as shown below:

$$Z_{sq} = \begin{bmatrix} Z_{0f} & 0 & 0 & Z_{0M} & 0 & 0 \\ 0 & Z_{+f} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{-f} & 0 & 0 & 0 \\ Z_{0M} & 0 & 0 & Z_{0z} & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_{+z} & 0 \\ 0 & 0 & 0 & 0 & 0 & Z_{-z} \end{bmatrix}$$

Sequence Impedance Matrix (Two, Balanced 3-Phase Circuits)

Where,

- $Z_{0n}$  = Zero sequence impedance of the  $n$ th circuit [ $\Omega/m$ ]
- $Z_{+n}$  = Positive sequence impedance of the  $n$ th circuit [ $\Omega/m$ ]
- $Z_{-n}$  = Negative sequence impedance of the  $n$ th circuit [ $\Omega/m$ ]
- $Z_{0M}$  = Zero sequence mutual impedance [ $\Omega/m$ ]

Sequence data of course, only makes sense when 3-phase circuits are considered. The format of this matrix depends heavily on the manner in which the individual circuits in the system are transposed (if at all). For example, if a double-circuit line is transposed so that all 6 conductors are included in the transposition, a sequence matrix will not be provided.

The positions of elements in this matrix are dependent on the manner in which the conductors/ground wires have been numbered. The type of ideal transposition that has been selected will also affect this matrix.

## Sequence Admittance Matrix ( $Y_{sq}$ )

This data represents the system sequence admittance matrix  $Y_{sq}$  per-unit length (S/m).  $Y_{sq}$  is derived directly from the shunt admittance matrix  $Y$  and the sequence transform matrix  $T$  (both described above) as follows:

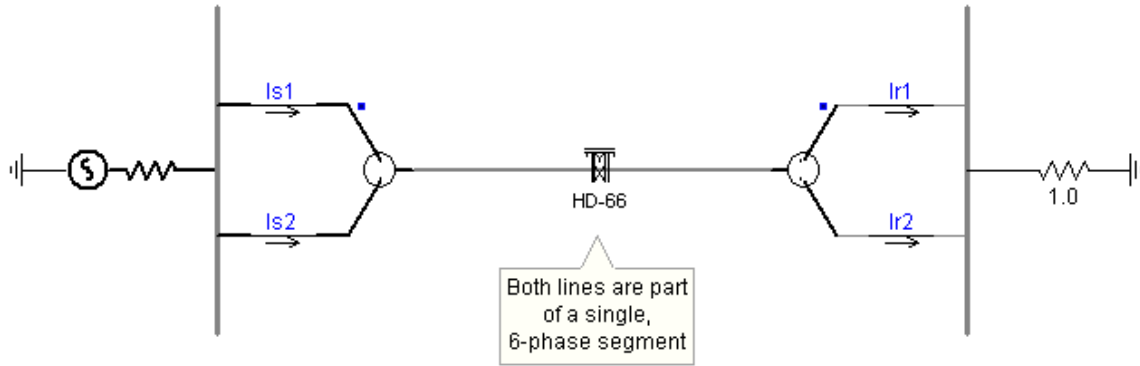
$$Y_{sq} = T^{-1} \cdot Y \cdot T$$

The same concepts described for the sequence impedance matrix  $Z_{sq}$ , apply to  $Y_{sq}$ .

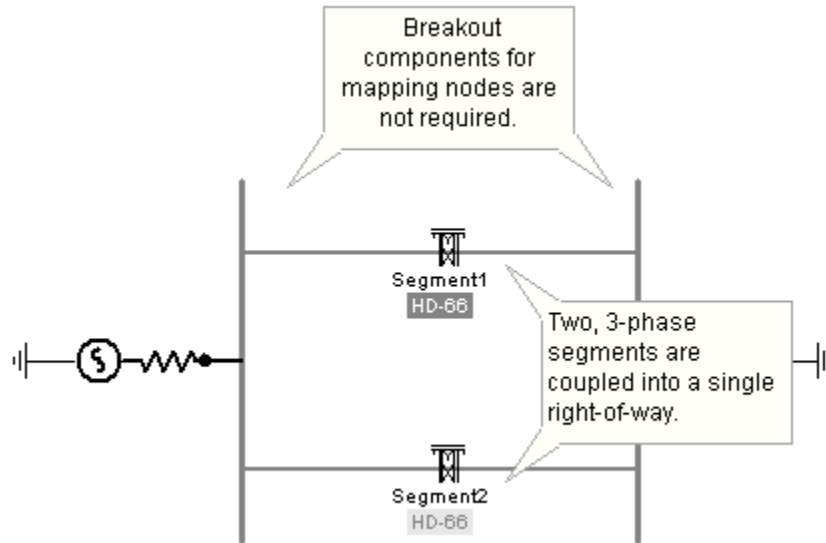
## Load Flow RXB Formatted Data

This data is simply the data given in the Sequence Data section, but reformatted for easy reading. The following diagram shows from where the data is taken:





Single, 6-Phase Segment

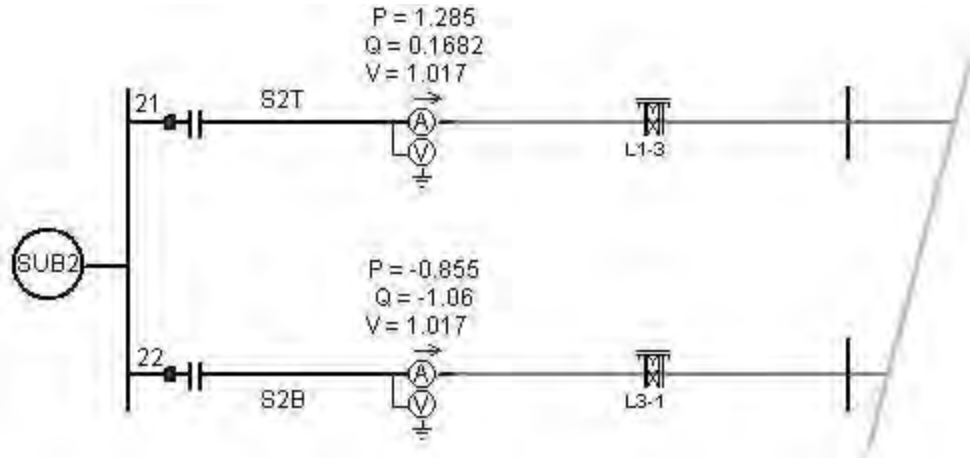


Two Mutually Coupled, 3-Phase Segments

## Mutual Coupling of Transmission Segments

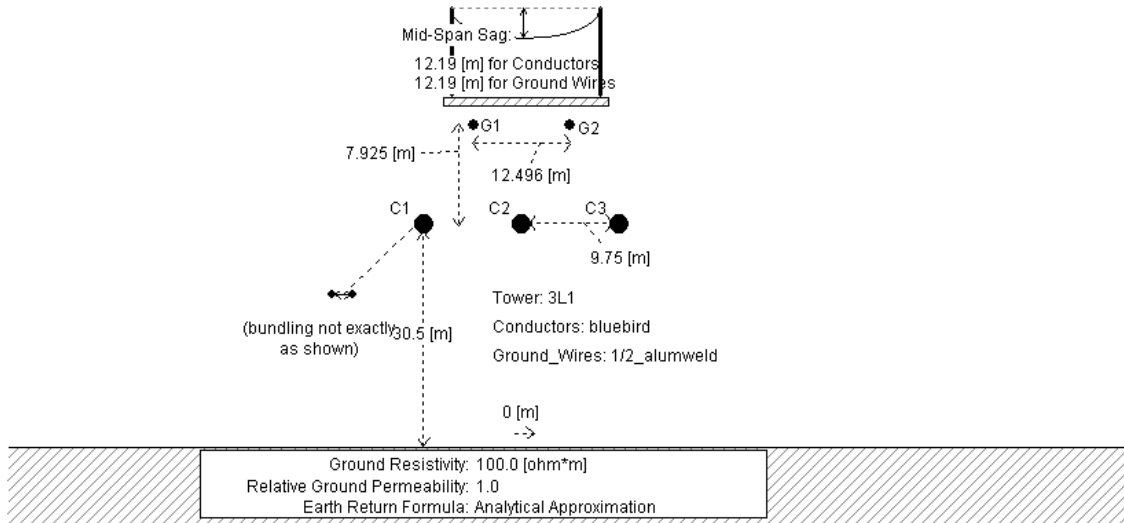
In order to couple two or more segments together, they first must possess the same length. Each unique segment in a collection of coupled segments (referred to as a *Right-Of-Way* or *ROW*) must possess a **Horizontal Translation** value, which serves to orientate segments on a global x-axis.

Also, one segment must be chosen as the **reference**, simply to indicate which segment will define the model and ground details used for the entire ROW. For example, consider the following two, unique transmission segments:

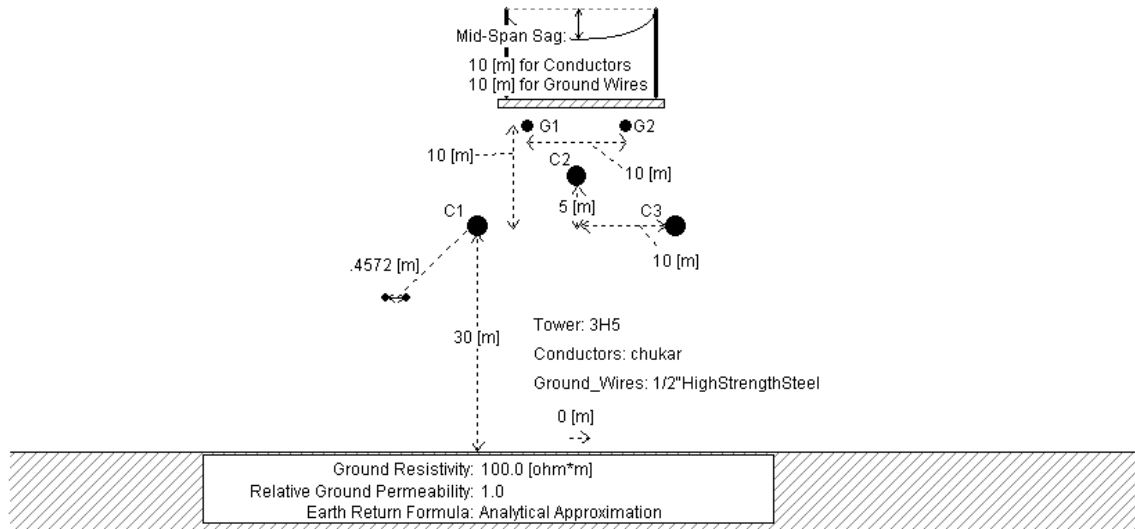


Transmission Segments on the Project Canvas

The two segments above are modeled as unique, and mutually exclusive from each other; that is to say they are not mutually coupled. Not being such is equivalent to assuming that the lines *L1-3* and *L3-1* are physically separated from each other – rendering any electromagnetic coupling effects between them insignificant.



Transmission Segment *L1-3* Editor Canvas



Transmission Segment L3-1 Editor Canvas

Let us assume however that in the real, physical system, lines L1-3 and L3-1 are indeed close enough to each other (perhaps routed through the same ROW) to allow coupling effects to play a more significant role in the solution. To provide this full representation in PSCAD, the two segments must be combined into a single ROW.

## Creating a ROW

First of all, ensure that all transmission segments possess the same length. Also keep in mind that it is not possible to combine overhead lines and underground cables within the same ROW. Next, decide which of the segments will be the **reference** segment. Right-click on the reference segment and select **Edit Parameters....** The *Configuration* dialog will appear.

Mutual Coupling	
Coupling of this segment to others is	disabled
Coupled segment tag name	row
Horizontal translation of this segment	0.0 [m]
This segment is	not the reference
Data entry method is by	tower dimensions only

Portion of the Transmission Configuration Dialog

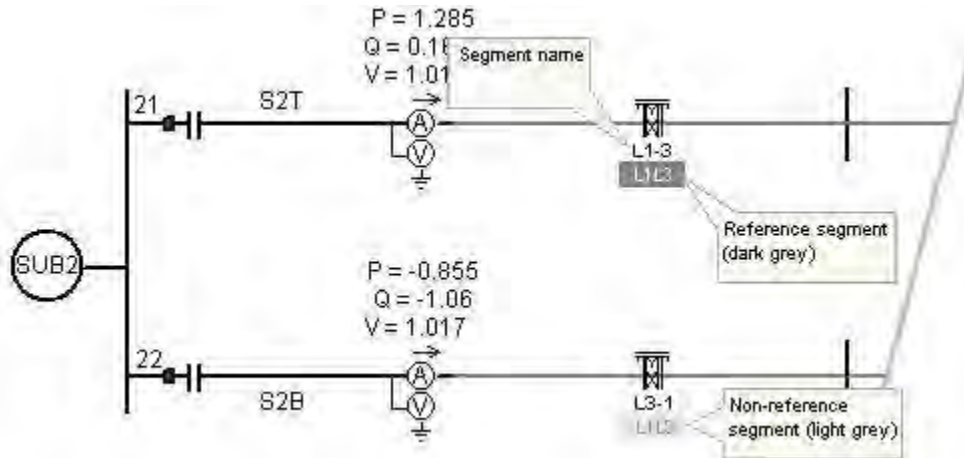
There are four parameters involved in defining a Right-Of-Way:

- Coupling of this segment to others is:** This parameter is used to enable/disable the mutual coupling feature. This toggle is convenient if the user should want to compare effects on the system with and without coupling.
- Coupled segment tag name:** This text parameter is used to specify the name of the ROW to which this line segment belongs. All segments included in a particular ROW must share the same tag name.
- Horizontal translation of this segment:** This parameter specifies a global horizontal placement of this particular segment, based on this segment's x-origin point (where a positive value signifies a translation to the right). You may utilize the unit system as well (the default unit is metres [m]): See the section entitled Unit System in Chapter 5 for more on units.

- **This segment is:** This parameter is used to specify the ROW reference segment. The selected reference segment is where PSCAD will extract the model and ground component parameters when constructing the segment input file. A ROW can contain only one reference segment.
- **Data entry method is by:** This toggle controls whether or not the data is to be entered using tower cross-sections (i.e. geometric position data), or by direct entry of sequence data. See Sequence Data Entry Method below for more details on this.

## Graphical Indicators

Additional graphics are provided in order to help distinguish between stand-alone segments, and those coupled within a ROW. For example, if lines *L1-3* and *L3-1* are coupled into a ROW called *L1L3*, where *L1-3* is the **reference** segment, then the circuit on the project canvas would appear as follows:



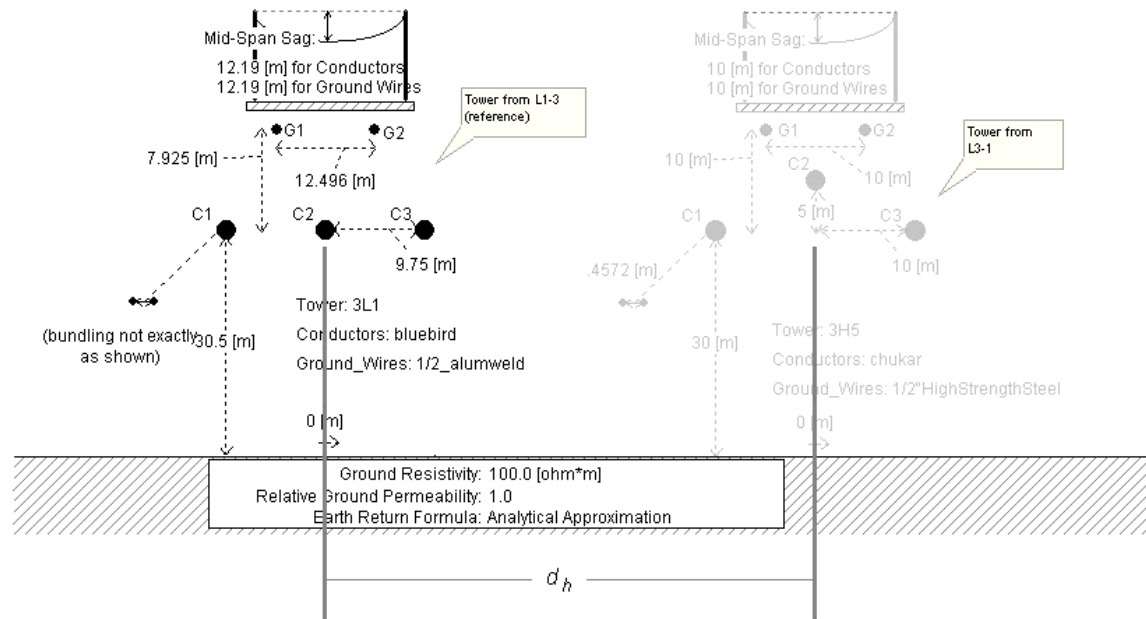
Mutually Coupled Transmission Segments on the Project Canvas

## Behind the Scenes

When a project is compiled, a unique segment input file is generated for every transmission segment in the project. The Line Constants Program (LCP) is called once for every input file, and in doing so produces a unique segment constants file for every call. The LCP constants files are used as input to EMTDC, when it needs to set-up the transmission system interface in the time domain.

Each input file contains information that defines the transmission segment; including conductor positions, resistance, circuits, ground properties, etc. The mutual coupling algorithm simply merges all towers (or cable cross-sections) from all segments in the ROW into a single segment input file, which assumes the name of the ROW (i.e. ROW tag name). Using only the horizontal translation distance between coupled segments, conductor position and numbering can be made relative to entire ROW.

For example, consider lines *L1-3* and *L3-1* as discussed in the previous section, where *L1-3* is the reference. Given the horizontal distance  $d_h$  between their respective x-origins, the two towers can be combined into a single system as follows:



Graphical Representation (Behind the Scenes) of a Mutually Coupled ROW

Two particular properties are modified during this process: *Conductor/Cable Position Data* and *Conductor Phasing/Cable Number Data*.

## Conductor/Cable Position Data

The conductor or cable position data of all segments in the ROW are modified such that the corresponding conductors/cables appear translated by the corresponding horizontal translation distance. Note that this modification would need to be performed by way of the *Relative X Position of Tower Centre on Right-Of-Way* tower input parameter, if additional towers were being merged manually – this is taken care of by the mutual coupling algorithm.

## Conductor Phasing/Cable Number Data

The line constants program requires that all conductors/ground wires/cables possess unique number indicators in order to distinguish conductor or cable order in the frequency domain solution. When more than one tower is added to a particular segment, this information is entered manually in each tower or cable.

The mutual coupling algorithm takes care of all conductor phasing and cable numbering automatically.

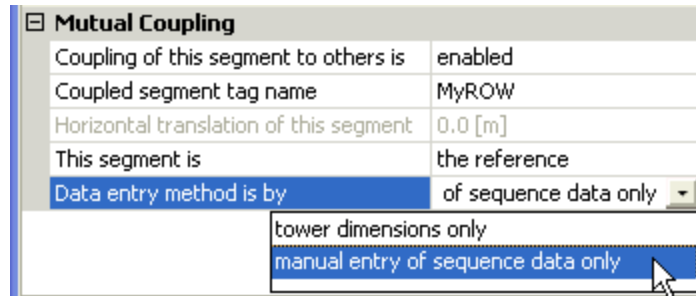
## Sequence Data Entry Method

If data is being entered directly as sequence quantities (that is, every transmission segment involved is utilizing the Manual Entry of Y,Z component) then conductor phasing and position do not play a part.

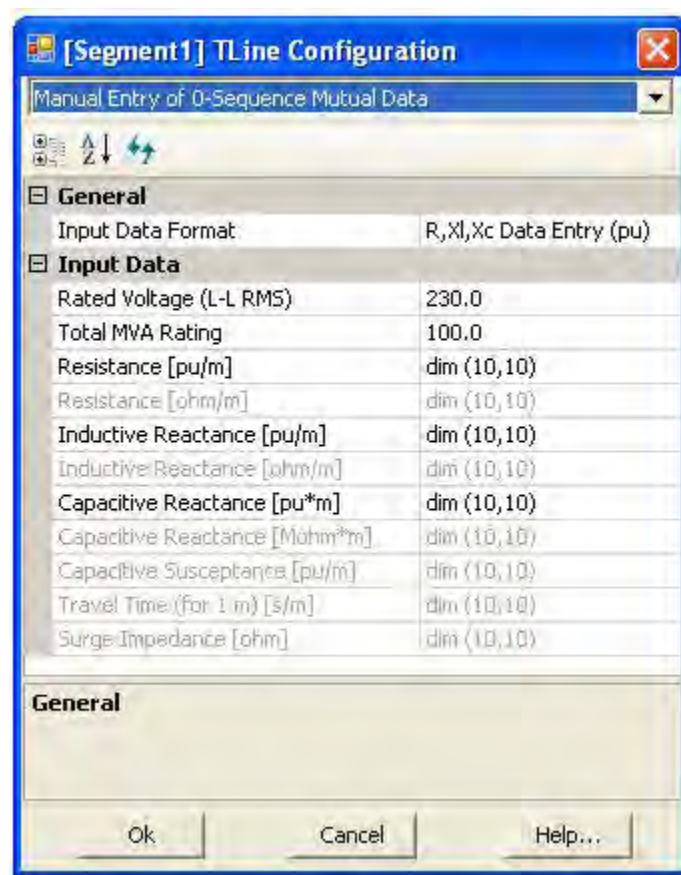
What occurs instead with sequence data entry is that PSCAD will construct Y and Z matrices for the entire coupled system; these matrices are based on information from both the individual Manual Entry of Y,Z components in each segment involved, as well as additional *0-sequence mutual data*. The *0-sequence mutual data* must be provided separately by the user, in the same data format as that data entered in the Manual Entry of Y,Z components.

The mechanism for entering the *0-sequence mutual data* is provided within the transmission segment configuration dialog of the reference segment for a particular ROW. To enter this data, first set the *Data entry method is by* input parameter to *manual entry of sequence data only*.





This will enable the second category page in the dialog called *Manual Entry of 0-Sequence Mutual Data*.



The data entered here works in a similar fashion to how data is entered within the Manual Entry of Y,Z component. First of all, ensure that the proper *Input Data Format* is chosen (this must be the same format as each individual segments in the ROW).

The *Input Data* in this dialog is entered in matrix format, where each row/column of the matrix represents a 3-phase system. Each input data parameter can hold information for up to 10, 3-phase coupled segments.

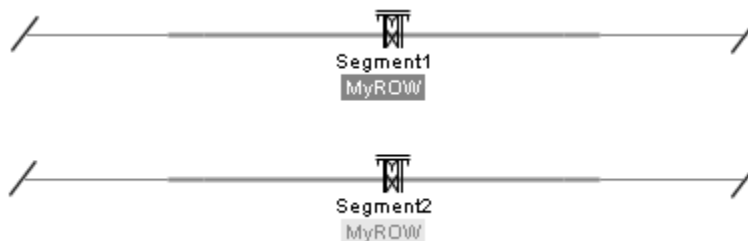
	Resistance [pu/m]									
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
▶ 1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	0	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	0	0	-1	-1	-1	-1	-1	-1	-1	-1
4	0	0	0	-1	-1	-1	-1	-1	-1	-1
5	0	0	0	0	-1	-1	-1	-1	-1	-1
6	0	0	0	0	0	-1	-1	-1	-1	-1
7	0	0	0	0	0	0	-1	-1	-1	-1
8	0	0	0	0	0	0	0	-1	-1	-1
9	0	0	0	0	0	0	0	0	-1	-1
10	0	0	0	0	0	0	0	0	0	-1

The majority of this table array contains  $-1$  values, which signifies that these positions are not used (they are the diagonals and the symmetrical duplicates in the matrix). The  $0$ -sequence mutual data is entered in the other positions as required. For example, if you are coupling 3, 3-phase lines, valid data will need to be entered in positions  $(2,1)$ ,  $(3,1)$  and  $(3,2)$ ; representing the  $0$ -sequence mutual values between segments 1 and 2, segments 1 and 3 and segments 2 and 3 respectively.

---

#### EXAMPLE 8-2:

A user wishes to couple two, 3-phase transmission segments (called *Segment1* and *Segment2*), which are each defined by directly entered sequence data (using the Manual Entry of Y,Z component).



The local data from each segment forms two, individual  $3 \times 3$  matrices each representing both  $\mathbf{Y}$  and  $\mathbf{Z}$ . If we just concentrate on  $\mathbf{Z}$  ( $\mathbf{Y}$  is treated in an identical manner), then the two segments will be represented by two unique  $\mathbf{Z}$  matrices called  $\mathbf{Z}_1$  and  $\mathbf{Z}_2$ . When two, 3-phase systems are coupled to form a single 6-phase system,  $\mathbf{Z}_1$  and  $\mathbf{Z}_2$  become the diagonals of the coupled  $6 \times 6$  matrix as follows:

$$Z_{ROW} = \begin{array}{|c|} \hline Z_1 & Z_m \\ \hline Z_m & Z_2 \\ \hline \end{array}$$

The matrix  $Z_m$  above represents the *0-sequence mutual data* that must be entered through the transmission segment configuration component. In this example, there are only two, 3-phase segments and so data is only required in the (2,1) position in each *Input Data* table parameter. The *Input Data Format* chosen for this example is *R,XI,Xc Data Entry (pu)* and so three separate values must be entered:

Resistance [pu/m]								
	R1	R2	R3	R4	R5	R6	R7	R8
1	-1	-1	-1	-1	-1	-1	-1	-1
2	6.86e-7	-1	-1	-1	-1	-1	-1	-1
3	0	-1	-1	-1	-1	-1	-1	-1

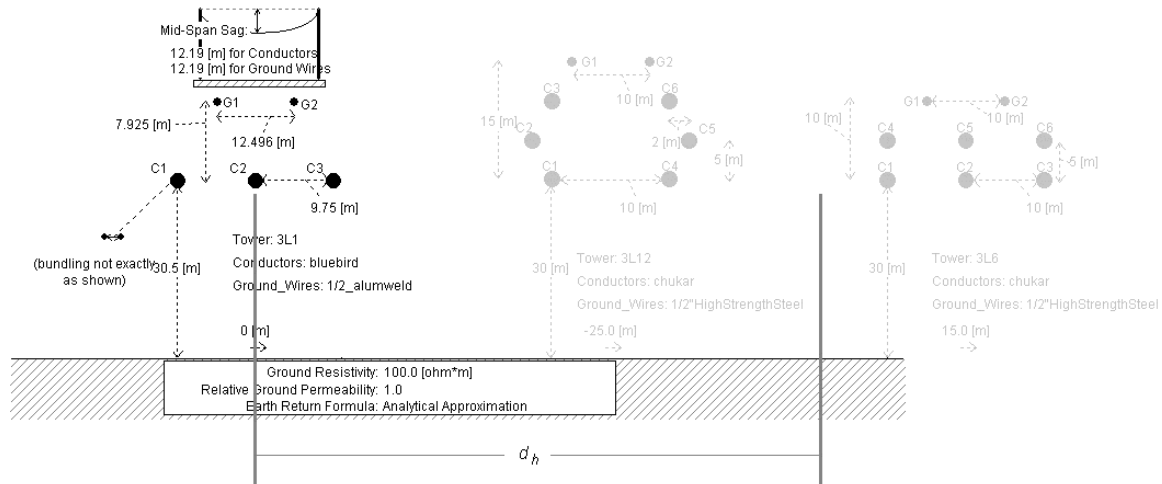
Inductive Reactance [pu/m]							
	XI1	XI2	XI3	XI4	XI5	XI6	XI7
1	-1	-1	-1	-1	-1	-1	-1
2	6.37e-7	-1	-1	-1	-1	-1	-1
3	0	-1	-1	-1	-1	-1	-1

Capacitive Reactance [pu*m]							
	Xc1	Xc2	Xc3	Xc4	Xc5	Xc6	Xc7
1	-1	-1	-1	-1	-1	-1	-1
2	5.4e7	-1	-1	-1	-1	-1	-1
3	0	0	-1	-1	-1	-1	-1

Once this data is entered, PSCAD will solve these two lines as one coupled system.

## Multiple Tower Segments

The mutual coupling algorithm fully supports combining individual segments containing multiple towers/cables. In fact, these are dealt with in exactly the same manner as single tower/cable segments:



Mutually Coupled ROW With Multi-Tower Segment

For more details on how conductor positions are calculated, see the section entitled The PSCAD Line Constants Program in Chapter 9 – Transmission Lines and Cables in the EMTDC Manual.

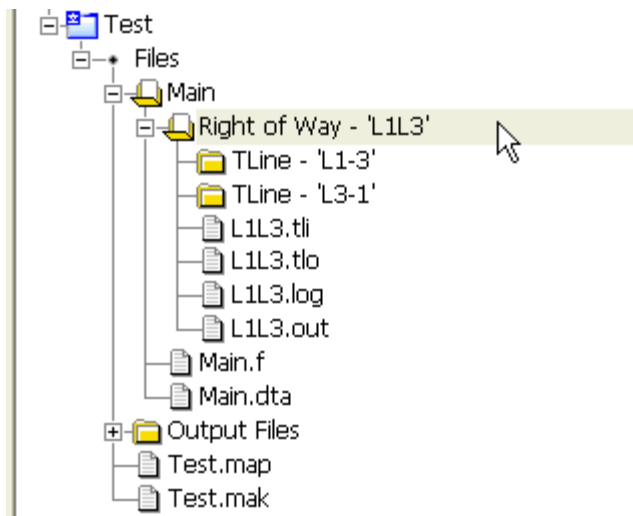
## Remote End Support

Transmission systems in *Remote Ends* mode are supported by the mutual coupling algorithm – this includes overhead lines and cables (all cables are in remote end mode). For overhead lines, it is also possible to combine segments in *Remote End* mode with those in *Direct Connection* mode.

## Viewing Mutually Coupled ROW Files

When individual transmission segments are coupled together to form a new, mutually coupled ROW, the files involved (i.e. input, constants, log and output) will assume the name of the ROW itself. In terms of PSCAD, a mutually coupled ROW is effectively a 'virtual' transmission segment that, unlike actual transmission segments, does not possess a definition. As such, its information cannot be accessed by using the Transmission Segment Definition Editor.

This means of course that there are no convenient tab windows, by which to view the files. Mutually coupled ROW files can however, be viewed by using the File tree in the Workspace window. The File tree detects both individual transmission segments and coupled ROWs in a project and intelligently organizes them into folders.



See the section entitled The File Tree for more details.

## Cautionary Measures

There are some precautions that should be taken when utilizing the mutual coupling algorithm that may not appear obvious at first.

### Tuning the ROW

It is important to remember that a mutually coupled right-of-way is itself a unique transmission segment, and therefore may require tuning just like any other individual segment. Do not assume that just because all the individual segments are properly tuned and giving good results, that the combined system will be tuned. In fact, the solution for the combined ROW may even fail at first!

### Model Type and Ground Parameters

Due to the fact that multiple, individual transmission segments are being combined into a single ROW, there may be dissimilarities between transmission line models and ground properties. To avoid this confusion, the mutual coupling algorithm assumes that the model and ground components used in the reference segment, is the same for all segments. If there are such dissimilarities between segments, PSCAD will issue build warnings in the output window to warn of any assumptions.

### Maximum Conductors/Cables

Keep in mind that the maximum number of conductors and/or cables in a right-of-way must still be adhered to. See the section entitled Scope Boundaries in Chapter 1 for details.

## LCP Detailed Output Viewer

The LCP Detailed Output Viewer utility provides a visual environment for the plotting of frequency domain, detailed output data from the Line Constants Program (LCP). Detailed output is available only when using the Frequency Dependent (Phase) or Frequency Dependent (Mode) models.

The viewer supports both matrices and vectors, in that any element of any output matrix can be plotted individually. Some examples of what data is available for plotting are:

- Series Impedance
- Shunt Admittance

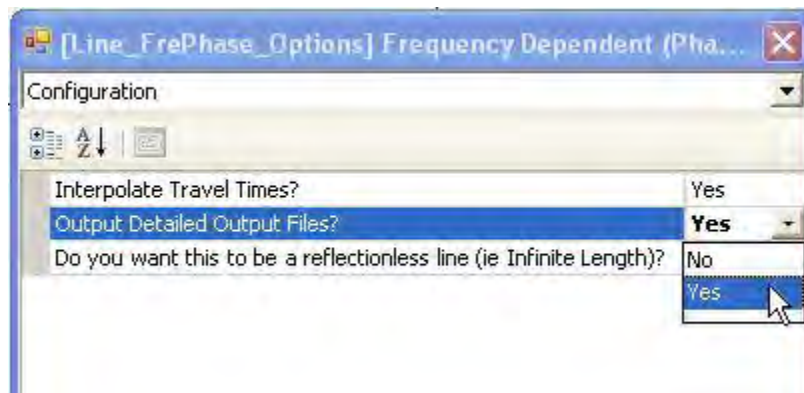
- Eigenvectors/Eigenvalues
- Curve Fitting Output

The plotting tools used in the viewer are based on those used for online plotting in PSCAD, including features such as markers, zoom and cross-hairs. Data may be written to file or copied to the clipboard just as it can be in the online plotting tools.

The detailed output data files are written as columnar formatted ASCII text files, which are placed in the project temporary directory (\*.emt) when the segment is solved.

## Creating Detailed Output

Before any detailed output can be viewed, the transmission line or cable segment must be solved by either using the Frequency Dependent (Phase) or Frequency Dependent (Mode) model. Note that in either case, the input parameter **Output Detailed Output Files?** must be selected as **Yes** before the constants are solved.



For more details on solving the constants, see the section entitled Solving the Transmission Segment Constants Manually in this chapter.

**NOTE:** Although mutually coupled ROW detailed output files can be generated, the Detailed Output Viewer does not currently support viewing of these files.

## Invoking the Detailed Output Viewer

The Detailed Output Viewer can be opened from within the Transmission Segment Definition Editor pop-up menu.



Detailed output is not available for the Bergeron model, as this model solves at only one frequency. See the section entitled Segment Output (\*.out) File in this chapter for more on viewing Bergeron model output.

## The Viewer Environment

The following sections provide a quick overview of the main parts of the Detailed Output Viewer utility.

### Spreadsheet Data Viewer

The viewer main page is an easily navigated, spreadsheet like viewer, which is opened as a separate window when the viewer is invoked.

LCP Detailed Output Viewer - [FLAT230]

log(f)	f [Hz]	(1, 1)	(1, 2)	(1, 3)
-0.30103000	0.50000000	0.99772927	0.12185723E-03	0.12185723E-03
-0.23801970	0.57806983	0.99765168	0.14642391E-03	0.14642391E-03
-0.17500940	0.66832946	0.99756832	0.17624187E-03	0.17624187E-03
-0.11199910	0.77268219	0.99747982	0.21228230E-03	0.21228230E-03
-0.48988796E-01	0.89332853	0.99738689	0.25562709E-03	0.25562709E-03
0.14021504E-01	1.0328125	0.99729034	0.30746900E-03	0.30746900E-03
0.77031804E-01	1.1940755	0.99719104	0.36911083E-03	0.36911083E-03
0.14004210	1.3805181	0.99708994	0.44196427E-03	0.44196427E-03
0.20305240	1.5960717	0.99698801	0.52755001E-03	0.52755001E-03
0.26606270	1.8452818	0.99688620	0.62750091E-03	0.62750091E-03
0.32907300	2.1334035	0.99678541	0.74357109E-03	0.74357109E-03
0.39208330	2.4665124	0.99668633	0.87765408E-03	0.87765408E-03
0.45509360	2.8516328	0.99658935	0.10318137E-02	0.10318137E-02
0.51810390	3.2968858	0.99649449	0.12083299E-02	0.12083299E-02
0.58111420	3.8116604	0.99640119	0.14097599E-02	0.14097599E-02

<FLAT230> - Detailed Output

Each column in the spreadsheet represents an individual quantity (usually a single matrix or vector element) and each row of data represents a calculation at a particular frequency.

### Column Organization

Depending on the output parameter being viewed, the data may exist as either matrix or array, or whether curve fitting results are included.

*Matrix:*

log(f)	f [Hz]	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, etc...
--------	--------	--------	--------	--------	--------	------------

Matrix parameters, such as the Series Impedance matrix (Z), are listed in order from left-to-right and top-to-bottom. The column headers are labelled in (row,column) format.

*Array:*

log(f)	f [Hz]	(1)	(2)	(3)	etc...
--------	--------	-----	-----	-----	--------

Array quantities are simply listed in order.

*Calculated/Fitted Quantities:*

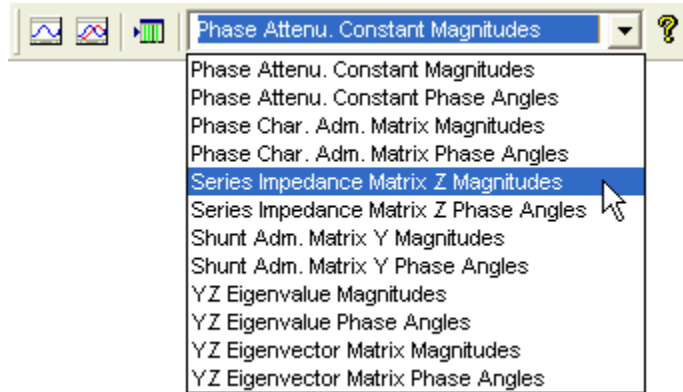
log(f)	f [Hz]	(1)C	(1)F	(2)C	(2)F	(3)C	(3)F etc...
--------	--------	------	------	------	------	------	-------------



When curve fitting results are included, the calculated and fitted quantities are staggered as indicated by the 'C' and 'F' respectively.

### Switching between Spreadsheets

Only one parameter may be viewed at a time in spreadsheet form. To switch to another available parameter, click on the parameter selection drop list in the Detailed Output Toolbar:



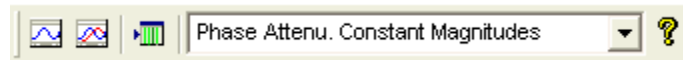
### Copying Data Rows

One or more rows of data maybe copied to the Windows clipboard. Simply click on a single row, or hold down the **Ctrl** key and select multiple rows, then right-click on the selection and select **Copy**.




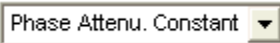

log(f)	f [Hz]	( 1, 1 )	(
-0.30103000	0.50000000	0.99772927	0.121
-0.23801970	0.57806983	0.99765168	0.146
-0.17500940	0.66832946	0.99756832	0.176
-0.11199910	0.77268219	0.99747982	0.212
-0.48988796E-01	0.89332853	0.99738689	0.255
0.14021504E-01	1.0328125	0.99729034	0.307
0.77031804E-01	1.1940755	0.99719104	0.369
0.14004210	1.3805181	0.99708994	0.441
0.20305240	1.5960717	0.99698801	0.527
0.26606270	1.8452818	0.99688620	0.627
0.32907300	2.1334035	0.99678541	0.743
0.39208330	2.4651000	0.99668633	0.877
0.45509360	2.8451000	0.99658935	0.103
0.51810390	3.2968850	0.99649449	0.120
0.58111420	3.8114600	0.99640110	0.140

### Detailed Output Toolbar

The Detailed Output Viewer is accompanied by its own toolbar:



The individual toolbar buttons are listed below with a short description:

Button	Description
	View only curves from presently viewed spreadsheet
	View all curves at once
	X-axis settings (i.e. plot vs. frequency or log(f))
	Drop list to select data viewed in spreadsheet
	Access help

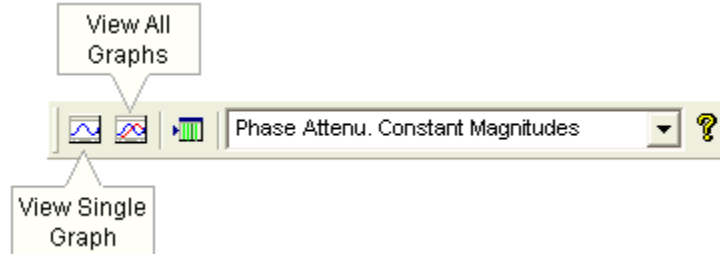
## Curve Viewer

The Curve Viewer is a simple utility used to for viewing the line constants detailed output data graphically. The utility uses the same online plotting facilities that are provided in the main PSCAD environment, tailored to this specific use. There are two Curve View utilities: **View Single Graph** and **View All Graphs**.

**NOTE:** For detailed information on using these plotting tools, see Chapter 6 – Online Plotting and Control.

### Invoking the Curve Viewer

To invoke the Curve Viewers, press either the *View Single Graph* or *View All Graphs* button in the Detailed Output toolbar:



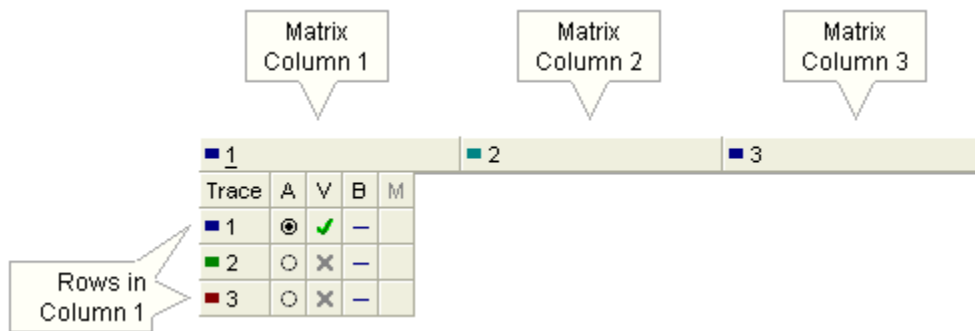
This will bring up the Curve Viewer as a floating window, as shown below for the Single Graph Viewer:



When viewing matrix quantities, such as the *Series Impedance (Z)* above, initially only the diagonal elements of the matrix are enabled; this is mainly to reduce the number of possible traces to be displayed.

### Element Identification

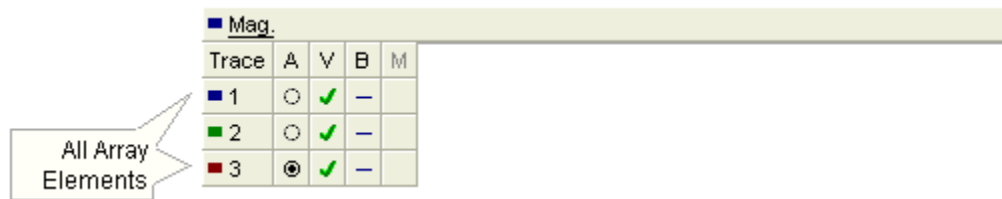
The Curve Viewer makes use of the multi-trace Curve feature available in the PSCAD online plotting tools, in order to allow plotting of 2-dimensional parameters such as matrices. For example, if plotting a 3x3 matrix, such as the Y or Z matrix magnitudes, the matrix elements are organized according to Curve and Trace, where each multi-trace Curve represents an entire matrix column, and the respective Traces represent the elements of that column.



A 3x3 Matrix

In the diagram above, the column 1 Curve has been expanded to reveal the three Traces representing the 3-elements of the column. That is, elements (1,1), (2,1) and (3,1). For more on multi-trace Curves, see the section entitled *Curves and Traces* in Chapter 6 – Online Plotting and Control.

Array items, such as the eigenvalue array are organized such that all elements are included within a single Curve. For example, the magnitudes of a 3-element array would appear as follows:



Trace	A	V	B	M
1	○	✓	—	
2	○	✓	—	
3	⊙	✓	—	

A 3-Element Array

## Chapter 8

### Component Design

# Designing Components

One of the features that make PSCAD such a powerful simulation tool is its allowance for the design of custom models. Users can develop models from the very simple, to the very complex, limited only by their skills and knowledge of the subject. Many seasoned users continue to amaze us (the developers of PSCAD) with what they have managed to accomplish with the software over the years.

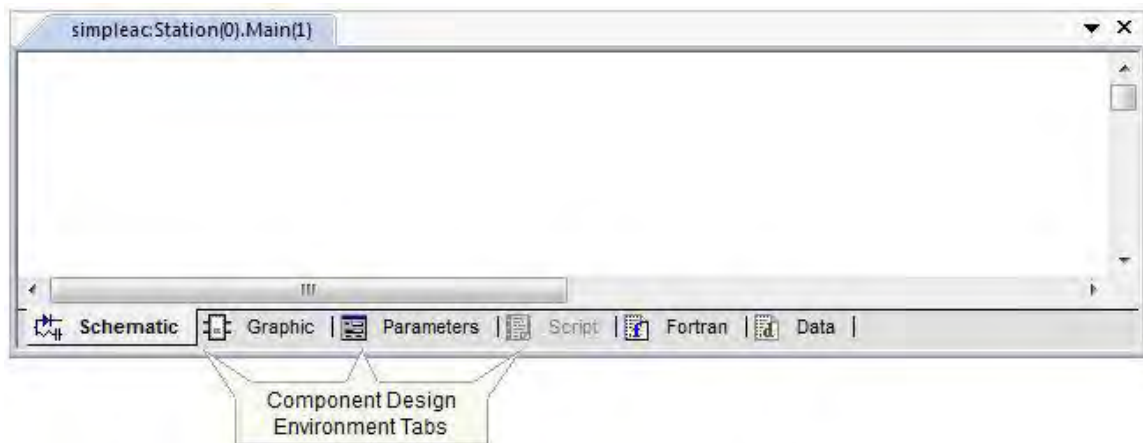
Custom components may be constructed in one of two ways: Either graphically by means of a module type component, or by direct coding. Regardless of the method used, in order for a custom model to be included in either the system dynamics or the electric network, a component must first be created to define the model. A component acts as a graphical representation of the model, where the user may supply input parameters, perform pre-calculations on input data, and change the component appearance. In the case of a module, it will possess a schematic, on which other components may be pieced together to form the model.

This chapter discusses the various features and tools available for the design of custom components. The information here is closely linked with the following chapter entitled Definition Script. It is suggested that you become familiar with both chapters before proceeding on with your component design.

The last section of this chapter includes a tutorial called Creating a New Component, which employs many of the features and concepts described here, and in the next chapter.

## The Definition Editor

In PSCAD V2, user components were designed and edited by way of a text file. In PSCAD V3, component design was accomplished by using a utility entitled the *Component Workshop* (or CWS). The Component Workshop was invoked by editing the component definition and contained different sections for the design of graphics, dialog windows and code sections. In PSCAD V4 and onwards, components are designed in a more integrated environment, which is no longer a separate utility. In fact, what is referred to as the *Definition Editor* is incorporated as part of the main tabbed window.



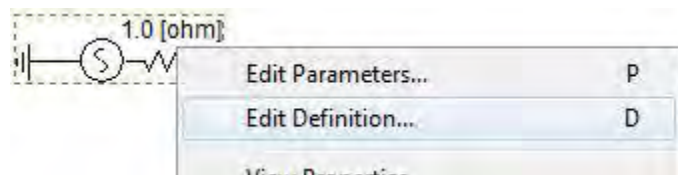
Four of the tabs represent the main sections of the definition editor, and will become enabled depending on whether the user is editing the definition of a regular component or a module. For example, modules do not possess a Script section, and so therefore this tab is disabled while editing module definitions:

- **Schematic:** The *Schematic* window displays the canvas of whatever module is being viewed (including the *Top-Level* module). At the same time, this canvas also acts as a graphical design environment for modules definitions (replacing the Script section).

- **Graphic:** The *Graphic* window is utilized specifically for the design of component graphics. A component or module graphic is the icon which represents it on the *Schematic* canvas.
- **Parameters:** This section is for the design of component or module input parameters and parameter dialogs. These represent the user-interface to the model.
- **Script:** The *Script* section harbours all of the component code (if any). The internal operability of the model and how it reacts to input is defined here. This section is not available in module definitions.

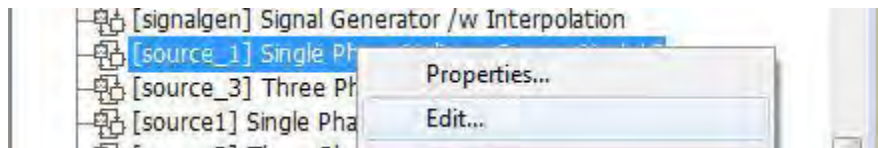
## Editing a Component or Module Definition

Access to the definition editor (i.e. editing a definition) can be performed a few different ways, depending on user preference. The most popular way is to right-click over the component and select **Edit Definition...** from the pop-up menu.



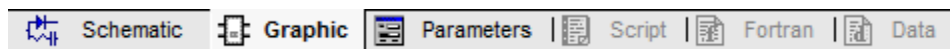
Other methods include:

- Hold down the **Ctrl** key and then left double-click on the component.
- Right-click on the definition listed in the workspace window and select **Edit...**



## The Graphic Section

The component graphic is important as it represents your model visually on the Schematic canvas. In the Graphic section, component graphic objects, text labels and port connections can be added, removed and manipulated.



## Navigation and Zoom

There are a few navigational features available to help you efficiently navigate about the Graphic canvas.

### Scroll Bars

Vertical and horizontal scroll bars are available in all environment windows. These are located at the right-most and bottom-most edges of the open window respectively.



## Arrow Keys

You can use the arrow buttons on your keyboard to scroll both horizontally and vertically.

## Panning (Dynamic Scroll) Mode

The panning or dynamic scroll feature allows you to scroll in a fluid motion. You can invoke panning mode by one of the following methods:

- On a blank portion of the page, press and hold the **Ctrl** and **Shift** keys, then click and hold the left mouse button (**Ctrl + Shift + left mouse hold**). Moving the mouse will then allow panning through the page.
- Press the **Pan** button in the ribbon control bar **Home** tab to invoke panning mode. To indicate that you are in panning mode, the mouse pointer will change into a hand shape. Press **Esc** to cancel pan mode.

## Zooming

There are a couple different methods for zooming available:

- Press the **+** or **-** keys on your keyboard to zoom in or out respectively.
- From the ribbon control bar **Home** tab, select either the **Zoom In**, **Zoom Out**, **Zoom Extents**, or **Zoom Rectangle** buttons, or select a percentage zoom directly from the **Zoom** drop down list.



## Cut, Copy, Paste and Delete Graphic Objects and Text

You can cut, copy or paste any graphic object or text label by using one of the following methods:

- Select the object with a left mouse click. Click the **Cut** or **Copy** buttons from the ribbon control bar **Home** tab. Click the **Paste** button to paste.
- Right-click the object and select **Cut** or **Copy** from the pop-up menu. Right-click over a blank area of the Graphic window and select **Paste** from the pop-up menu.
- Select the object with a left mouse click so that grips appear. Press **Ctrl + x** or **Ctrl + c** to cut or copy the label respectively. Press **Ctrl + v** to paste.

# Graphic Objects

There are several types of purely graphic objects available. These include:

- ¼ Arc
- ½ Arc
- Ellipse
- Line
- Rectangle



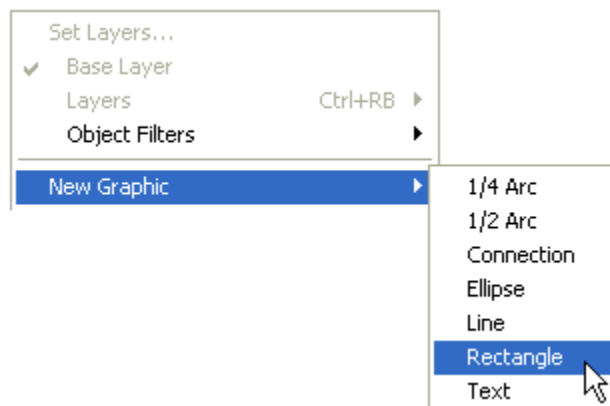
Object properties, such as colour, fill and size can be changed to suit the component graphical needs. A good graphical design can enhance user understanding of the purpose of the component and what its primary function is.

## Adding Graphic Objects

To add a graphic object to your component definition, the most straightforward method is to use the ribbon control bar **Shapes** tab:



Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic window. Right-click and select **New Graphic**.

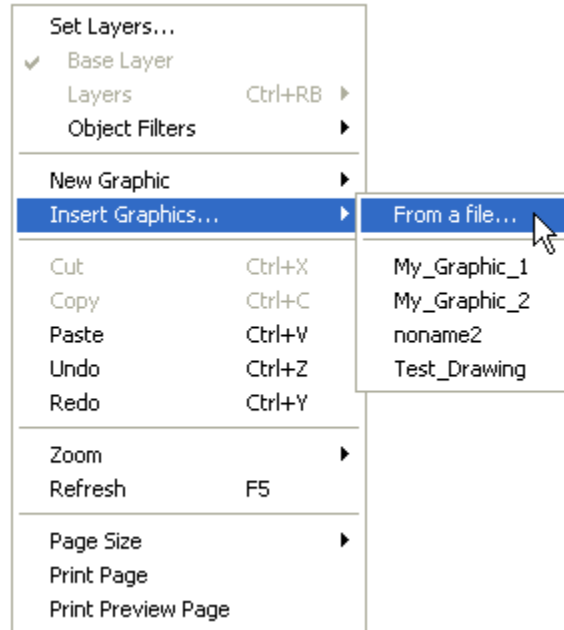




Depending on the object selected, a new object will appear attached to your mouse pointer. With your mouse, move the object to the desired location on the canvas and left-click to place the object.

## Inserting Graphics

Graphic objects may also be inserted by importing a previously defined *Scalable Vector Graphics (\*.svg)* file. To do so, move the mouse pointer over a blank area of the Graphic window. Right-click and select **Insert Graphics....**



If you would like to insert a specific graphics file, select **From a file...** and point directly to the file.

A special folder called *overlays*, under the PSCAD installation directory, has been designated as the default folder to house graphics files that have been previously saved from within PSCAD (See section entitled Saving Graphics to File below.). Any files stored in this folder will be listed in the sub-menu, as shown above.

The location of the graphics folder may be changed through the *Workspace Options* setting called *SVG script editor*. See Dependencies in chapter 3 for more details on this setting.

Note that the format of the file must be very specific in order to be imported into the Graphics section. Presently, only minimal support has been implemented (i.e. basic PSCAD graphic objects), and so this feature should be used as a way to transfer graphics from one component to another. In fact, it is best that you only use the *Save Graphics* feature described below to create the files from within PSCAD, and avoid using external programs. A partial \*.svg file, created by PSCAD, is shown below as an example:

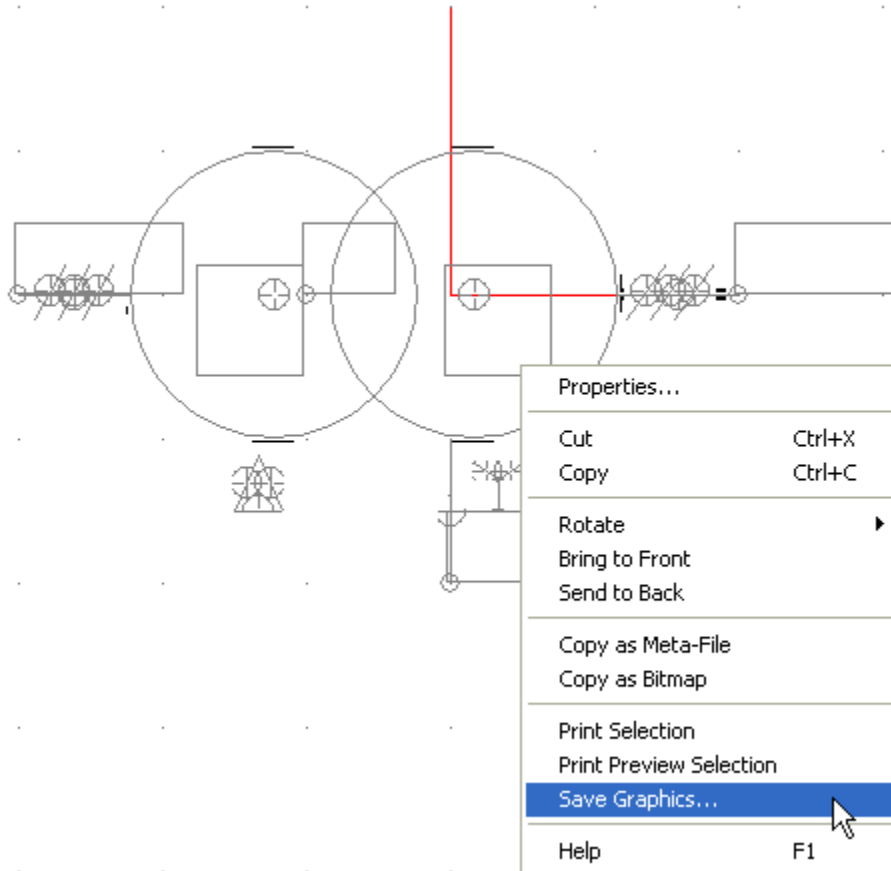
```
<svg id='noname' viewBox='-200 -200 200 200'>
  <port model="Natural" name="G1" x="-18" ... ></port>
  <line x1="6" y1="23" x2="3" y2="21" stroke="Black" ... />
  <ellipse cx="3" cy="0" rx="18" ry="18" stroke="Black" ... />
  <text x="6" y="6" stroke="Black" fill="Black" ... ></text>
</svg>
```

For more details on *Scalable Vector Graphics*, see <http://en.wikipedia.org/wiki/Svg>.

## Saving Graphics to File

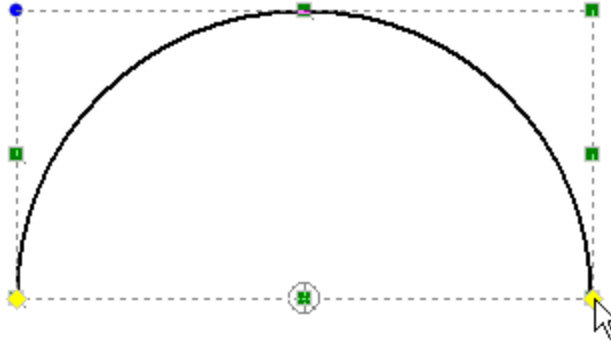
One or more graphic objects may be saved to a *Scalable Vector Graphics* (\*.svg) file format. In this way, collections of selected graphic objects may be duplicated for transfer from one component to another.

To save a collection of graphic objects to file, select the objects (using say box select), right-click and choose **Save Graphics...** from the pop-up menu. You will be presented with a dialog, from which you can save the graphics as a \*.svg file.



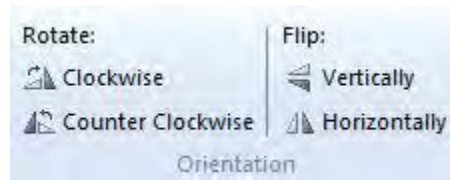
## Rotate, Flip, Mirror and Re-size Graphic Objects

Graphic objects can be rotated, flipped, mirrored or re-sized. To re-size, left-click on the object so that grips appear.

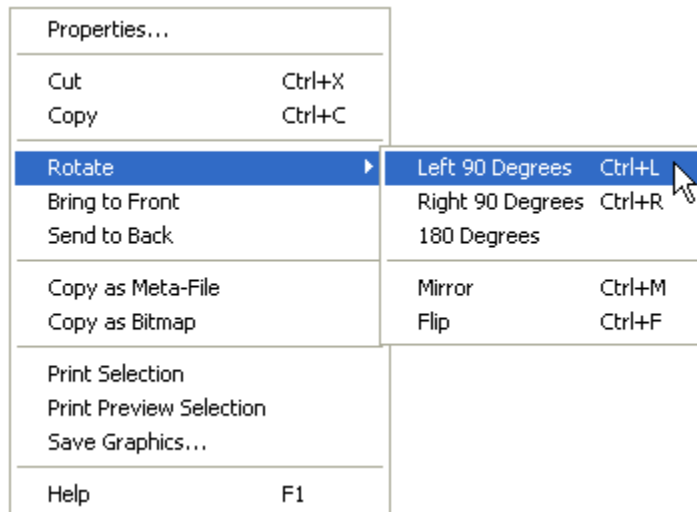


Place the mouse pointer over a selected grip, press and hold the left mouse button, and move the mouse. Note that the corner grips will allow re-sizing in both directions, while the mid-point grips will only allow movement in either the horizontal or vertical directions.

Rotate, flip or mirror can be accomplished in one of three ways. The first is to use the buttons in the ribbon control bar **Shapes** tab:



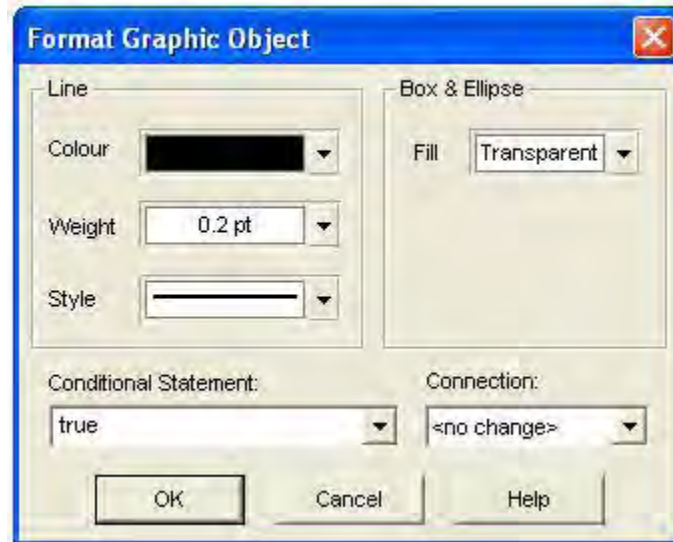
You can also use the right-click pop-up menus: Right-click over the object and select **Rotate**. Choose one of the options given.



As an alternative to the right-click menu and ribbon bar, you can also use keyboard shortcuts **Ctrl + r**, **Ctrl + f** and **Ctrl + m** (or simply **r**, **f** and **m**) for rotate, flip and mirror respectively. Make sure that the mouse pointer is over top the object before using these keys. If you select the object first, then the object will rotate, flip and mirror centred on the object itself.

## Changing Graphic Object Properties

Graphic object properties (except the *Arc*) can be adjusted through the Format Graphic Object dialog window. To change the object properties: Either left double-click or right-click over the object and select **Properties...**



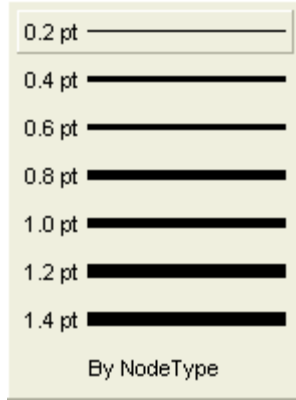
As shown above, graphical features such as styles and colours can be adjusted.

Line:

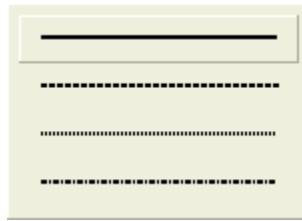
- **Colour:** Select the down arrow to bring up the colour palette shown below. Left-click on a colour button to change the colour of the object line. If the object is a *Rectangle* or *Ellipse*, this will change the border colour.



- **Weight:** Select the down arrow to bring up the line weight palette shown below. Left-click on a weight button to change the weight or thickness of the object line. If the object is a *Rectangle* or *Ellipse*, this will change the border weight. If **By Node Type** is selected, the type of node to which the object is associated with will determine the line weight. See the Connection input field description below for more details.

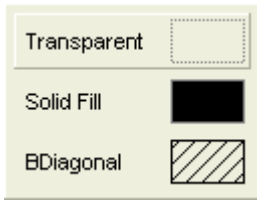


- **Style:** Select the down arrow to bring up the line style palette shown below. Left-click on a style button to change the style of the object line. If the object is a *Rectangle* or *Ellipse*, this will change the border style.



Rectangle & Ellipse:

- **Fill:** Select the down arrow to bring up the fill palette shown below. With the mouse pointer over the **Transparent**, **Solid Fill** or **Pattern** buttons, left click the down arrow to bring up the respective sub-palettes shown below. This input is disabled for *Line* objects!



Fill Palette



Solid Fill Sub-Palette

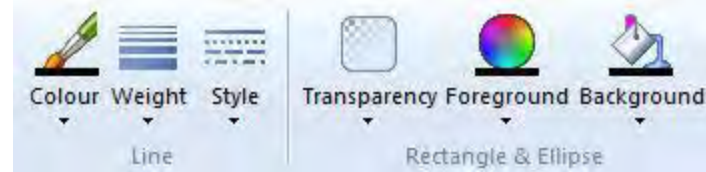


Pattern Sub-Palette

Other:

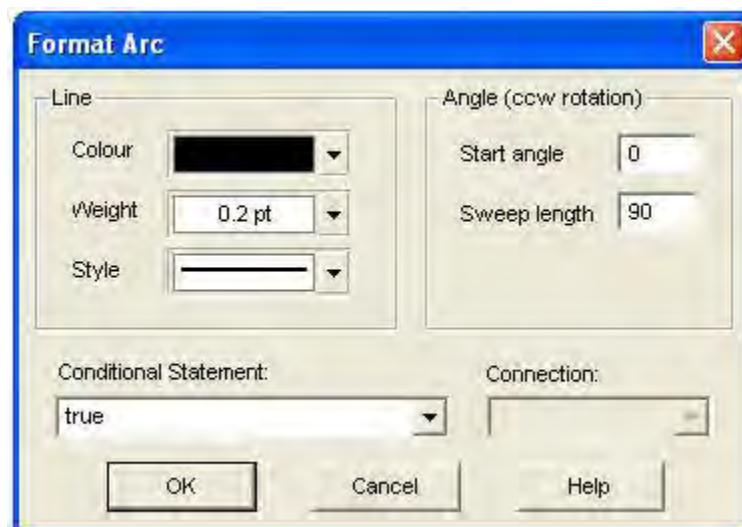
- **Conditional Statement:** Enter a conditional statement to determine under what input conditions the object is to be visible. See Conditional Statements, Layers & Filters for more details.
- **Connection:** Enter the name of a local, electrical port connection to which this object is to be associated. This input is used to associate the line weight of a graphical object with a specific local port connection. If the dimension of the electrical port is greater than 1 (i.e. an array), then its line weight is doubled. This method is used extensively in master library components for single-line diagram compatibility.

In addition to the *Format Graphic Object* dialog, you can change the line/border colour, weight and style, as well as fill properties directly from the ribbon control bar **Shapes** tab. First, select the object with a left-click (grips appear). Then adjust any of the above properties by using the appropriate buttons.



## Changing Arc Object Properties

The Arc is a special type of graphical object. Arc object properties can be adjusted through the Format Arc dialog window. To change the Arc properties: Right-click over the Arc object (without selecting it) and select **Properties...**



As shown above, graphical features such as styles and colours can be adjusted.

Line:

The properties within the Line area (i.e. **Colour**, **Weight** and **Style**) function exactly as described for graphic objects. See Changing Graphic Object Properties above.

Angle (ccw rotation):

- **Start Angle:** Enter the angle in degrees at which the arc has its starting point. This angle is based on a standard, four-quadrant system.
- **Sweep Length:** Enter the angular distance through which the arc is to sweep. Sweep distance is always in the counter-clockwise direction.

Other:

The remaining properties (i.e. **Conditional Statement** and **Connection**) function exactly as described for graphic objects. See Changing Graphic Object Properties above.

## Text Labels

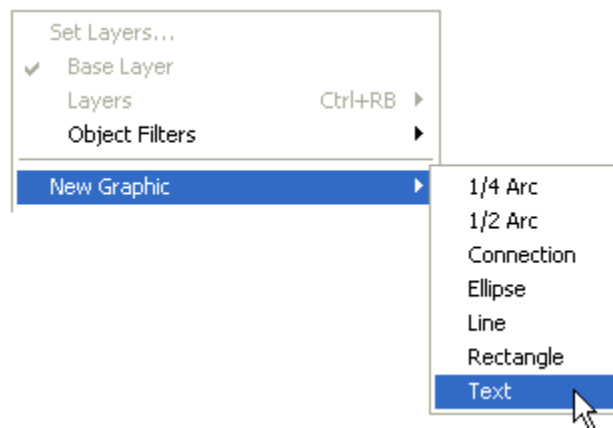
Text can be displayed on component graphics by using a *Text Label* object. Text labels can be one line of text only, and the user may select alignment and size.

## Adding Text Labels

To add a Text Label to your component definition, the most straightforward method is to use the ribbon control bar **Shapes** tab:



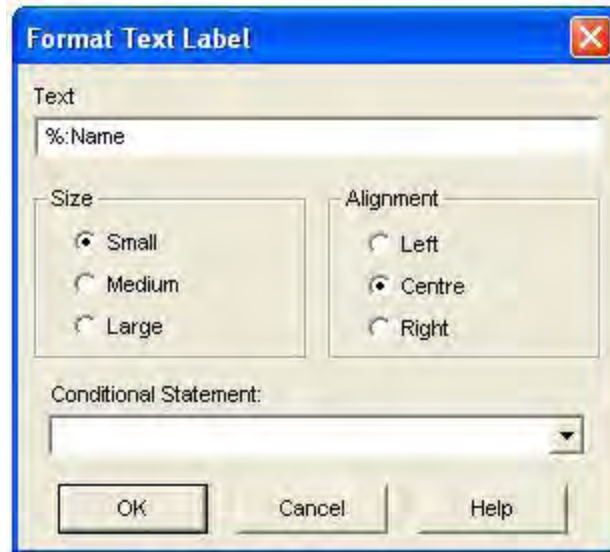
Simply left-click the **Label** button, drag the label to where you want it placed and left-click again. Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic window. Right-click and select **New Graphic | Text**.



A *Text Label* will appear attached to your mouse pointer. With your mouse, move the label (left-click and hold) to the desired location within the Graphic canvas.

## Changing Text Label Properties

Text label properties can be adjusted through the **Format Text Label** dialog window. To change the Text Label properties: Either left double-click or right-click over the *Text Label* and select **Properties....**



As shown above, text features such as size and style can be adjusted.

- **Text:** Enter the text you wish to appear on the label.
- **Size:** Select **Small**, **Medium** or **Large** sizes for your text.
- **Alignment:** Select **Left**, **Centre** or **Right** justification for your text.
- **Conditional Statement:** Enter a conditional statement to determine under what input conditions the object will be visible. See Conditional Statements, Layers & Filters for more details.

## Linking a Text Label to an Input Field

It is possible to link *Text Label* text to a specific input parameter within the same definition. Once linked, the label can be used to graphically display the parameter value (and unit). For example, you can display the MVA rating of a transformer model according to what is entered in its *Rated MVA* input field.

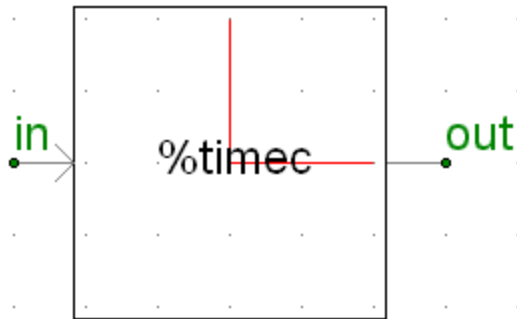
Linking is accomplished by simply entering the Symbol name of an associated input parameter into the text label, preceded by either a percent (%) or a dollar (\$) symbol. If the \$ prefix is used, just the value of the linked input field will be displayed. If the % prefix is used, both the value and the specified unit will be displayed. The following example illustrates how to link to an input field. See The Parameters Section and Substitutions in Chapter 10 for more details.

---

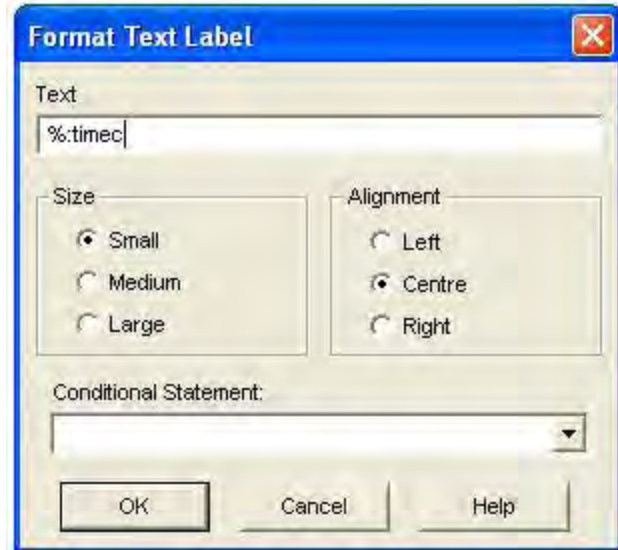
### EXAMPLE 9-1:

A user wants to display the value of an input field with Symbol name *timec*, which represents the component time constant. The user adds a text label to the component graphic and adds the following to the **Text** input field in the Format Text Label dialog:



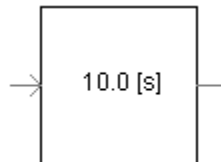


Component Graphic section



Text Label Property Settings

If the *timec* input field has a value of say *10.0 [s]*, then the resulting display on the Circuit canvas would be similar to:



**NOTE:** If the % prefix above is replaced by the \$ prefix, the units will not be displayed.

## Port Connections

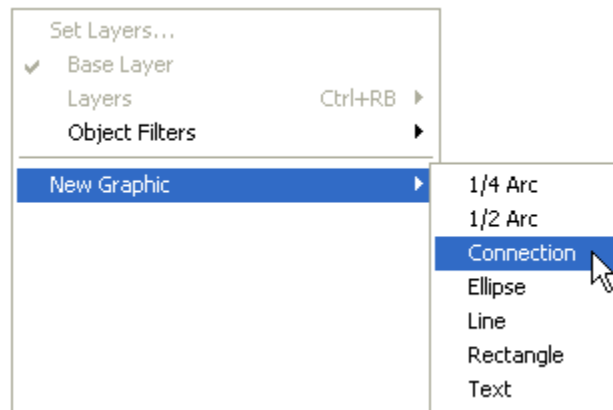
*Port Connections* are used to provide graphical access to signals defined within the Circuit canvas where the component instance resides. They provide a means to either read signals from, or output signals to the external system each simulation time step. These ports play an essential part in the construction of circuits in PSCAD, and are the graphical signal communication avenue between models.

### Adding Port Connections

To add a *Port Connection* object to your component definition, the most straightforward method is to use the ribbon control bar **Shapes** tab:



Simply left-click the **Port** button on the ribbon bar, drag the connection to where you want it placed and left-click again. Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic window. Right-click and select **New Graphic | Connection**.

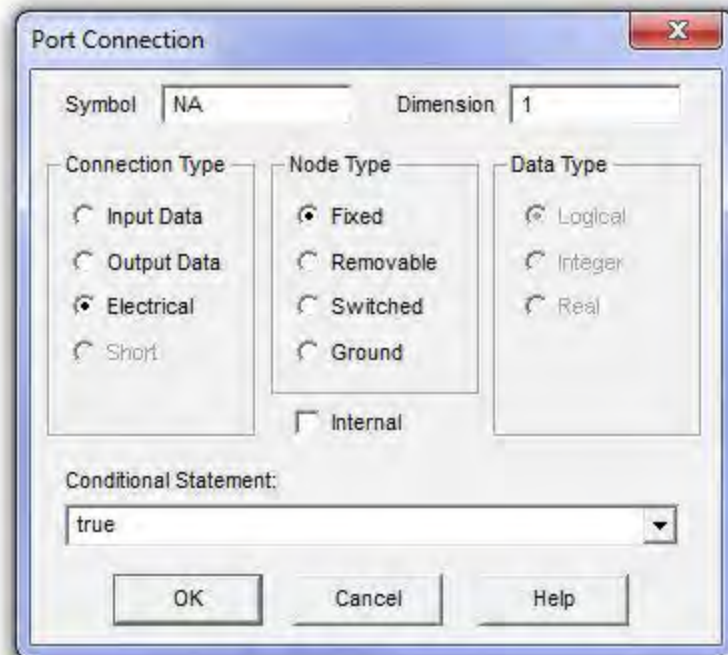


A port connection should appear attached to your mouse pointer. With your mouse, move the node to the desired location within the Graphic window and left-click to place the object.

**NOTE:** You may notice that port connections are always snapped to the Graphic canvas drawing grid. This is to ensure that when other components are connected to yours, their respective connection points will overlap.

## Changing Port Connection Properties

Port connection properties can be adjusted through the Port Connection dialog. To change the connection properties: Either left double-click or right-click over the connection and select **Properties...**



As shown above, connection features such as name and type can be adjusted.

- **Symbol:** Enter a name for the port connection. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, must not include spaces or other illegal characters, etc.).
- **Dimension:** If this connection is to carry an array signal, then this input specifies the dimension of the array. For example, if port connection *N1* is to be defined as REAL N1(3), then this field should be specified as 3. You may also substitute the Symbol of an integer input field or choice list in this field. The connection will assume a dimension equal to this value upon compilation.
- **Connection Type:** Select **Input Data**, **Output Data** or **Electrical**. If this connection is to be part of the EMTDC system dynamics (i.e. a control signal), then you must either choose **Input** or **Output Data**. Only select **Electrical** if this connection to be part of an electrical circuit.
- **Node Type:** Select **Fixed**, **Removable**, **Switched** or **Ground**. This parameter is only enabled if the **Connection Type** is **Electrical** (see the following Electrical Node Types section). If designing a module component, only Fixed-type electrical nodes may be used.
- **Data Type:** Select **Logical**, **Integer** or **Real**. This input defines the type of data signal that will be passing through this connection and is based on the standard Fortran LOGICAL, INTEGER or REAL declarations. It is only enabled if **Connection Type** is **Input** or **Output Data**.
- **Internal:** Selecting this choice box simply disables the compiler warning message indicating that an internal electrical, isolated node exists. Note that this port can still be connected to other ports (via a wire, etc.) when flagged as internal.
- **Conditional Statement:** Enter a conditional statement to determine under what input conditions the connection is to be enabled. See Conditional Statements, Layers & Filters for more details.

## Electrical Node Types

When the connection is selected as an electrical node, there are four electrical node types available to the user. These are described below:

- **Fixed:** A fixed node is the most common type of electrical node and should be the default chosen when in doubt. It represents a simple electrical node.
- **Removable:** A removable node is that which may be 'removed' by PSCAD, if it is to be part of a collapsible branch. For example, a branch with separate series RLC elements can be collapsed by PSCAD into an equivalent, single element impedance branch ( $Z$ ) (effectively removing two extra nodes). Select removable if you would like to take advantage of this feature.
- **Switched:** If your node is to be part of a frequently switching branch, that is a branch whose equivalent conductance is changing many times during a simulation (thyristor, GTO, etc.), then this option should be chosen. Switched nodes are included in the *Optimal Node Ordering* algorithm, which makes matrix decomposition more efficient, and thereby speeds up the simulation.
- **Ground:** Select this option if your node is to be a ground node.

**NOTE:** Only **Fixed**-type electrical ports are allowed in module component definitions.

## Undo and Redo

Undo/redo is presently not supported in the Graphics section.

## Adjusting Graphic Page Size

In order to accommodate very large component graphics, an option to change the Graphic canvas size is given. Move the mouse pointer over a blank area of the Graphic window. Right-click and select **Page Size**.

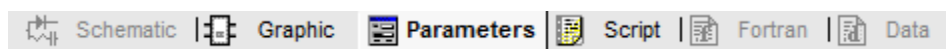


The default canvas size is **Smaller**.

## The Parameters Section

The *Parameters* section is used to design the primary interface between the component and the user. This is accomplished through the use of programmable dialog windows know as *Categories*, where users may input everything from model parameters to model conditions and appearance. Category windows act as the users interface once the component design is completed.

If the Parameters section is not enabled, click on the *Parameters* tab as shown below:



## Categories

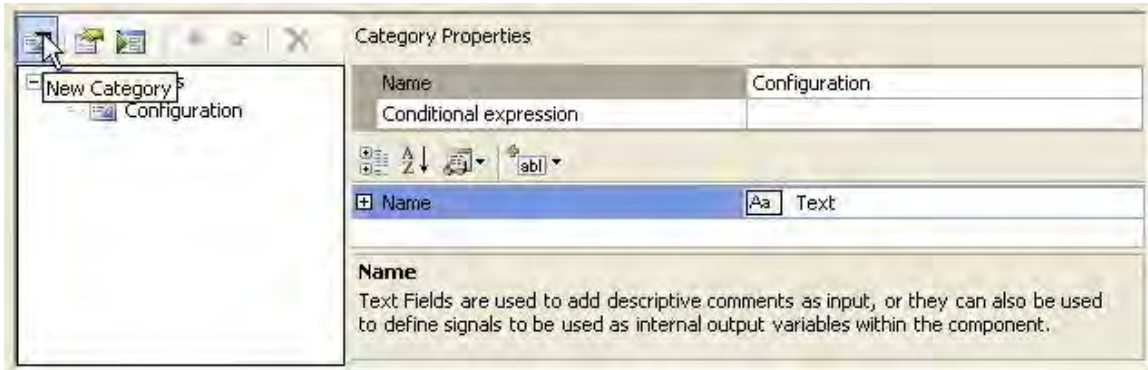
Within the Parameters section, you may include several category pages, each containing inputs pertaining to a similar function, or you may place all inputs in a single category page. There are several types of input parameter fields that may be added categories. These include:

- Choice
- Integer
- Real
- Logical
- Text
- Table
- Boolean

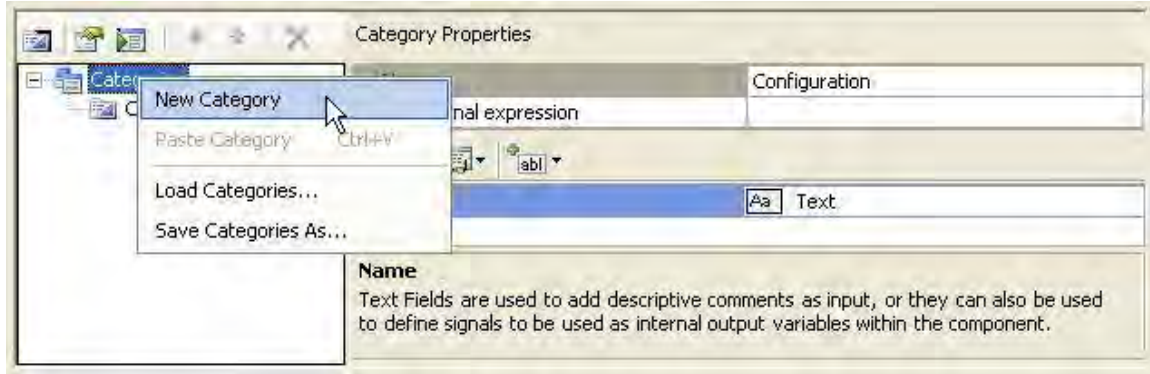
**NOTE:** When you first create a new component, a default category called *Configuration* will be created for you. This category will possess a single default parameter field called *Name*. *Name* is used for display in the workspace secondary window, if the component is a module. See The Secondary Window for details.

## Adding a New Category

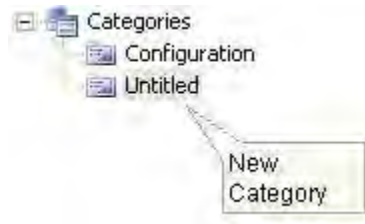
To add a new category page to your component definition, simply click the **New Category** button:



Or, right-click on the *Categories* tree and select **New Category**.

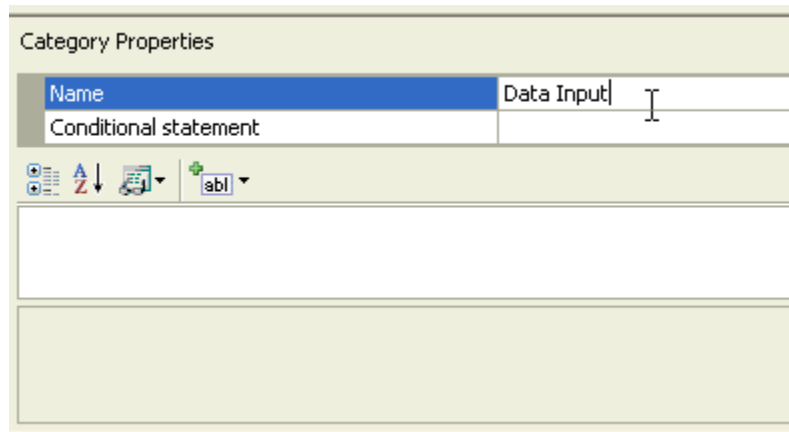


In either case, a new category page will appear with default name *Untitled* as shown below.



## Changing Category Properties

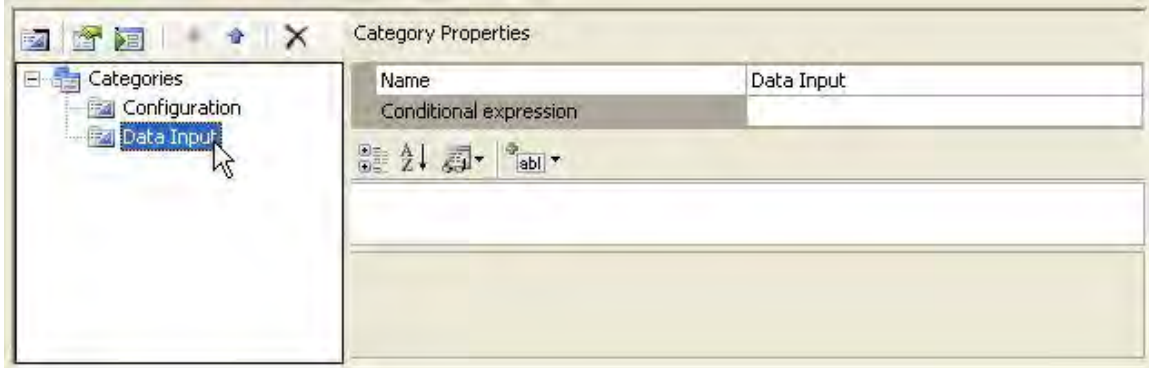
Category properties can be adjusted at any time.



- **Name:** Enter a descriptive name the category page.
- **Conditional Statement:** Enter a conditional statement (optional) to determine under what input conditions the category is to be enabled. If a category is disabled, the user will still be able to view the contents, but all input fields within it will be disabled. See Conditional Statements, Layers & Filters for more details on conditional statements.

## Navigating Through Categories

Once you have added more than one category, there will of course be a need to navigate through these pages. In the category tree, left-click on the desired category page to view it.



## Ordering Categories

Once you have added more than one category, there may be a need to re-order the sequence of the category pages. Select a category page and then click the up or down arrow on the tool bar as shown below:



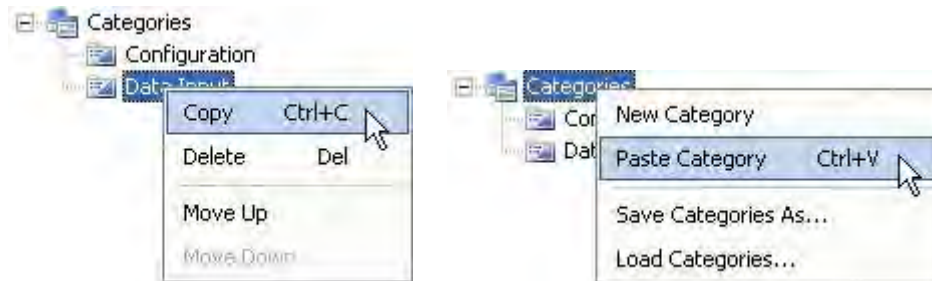
In the above diagram, the *Configuration* category page was moved below the *Data Input* page.

Another method is to drag and drop a category: Left-click and hold a category page in the tree and then drag it to the desired spot. Release the mouse button.



## Duplicating a Category

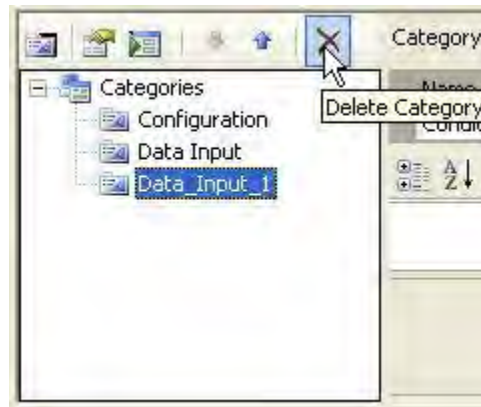
Category pages are duplicated by simply copying and pasting them in the Categories tree. Select the desired category in the tree, right-click and select **Copy** (or press **Ctrl + c**). Paste the category: Right-click on the Categories tree and select **Paste** (or press **Ctrl + v**).



A new category page will appear in the tree with a number appended to the copied category name. You may rename the category as described in Changing Category Properties above.

## Deleting a Category

To delete a particular category, select it in the Categories tree and then click the **Delete Category** button in the tool bar (or press the **Delete** key).



You may also right-click on the category page itself and select **Delete**.

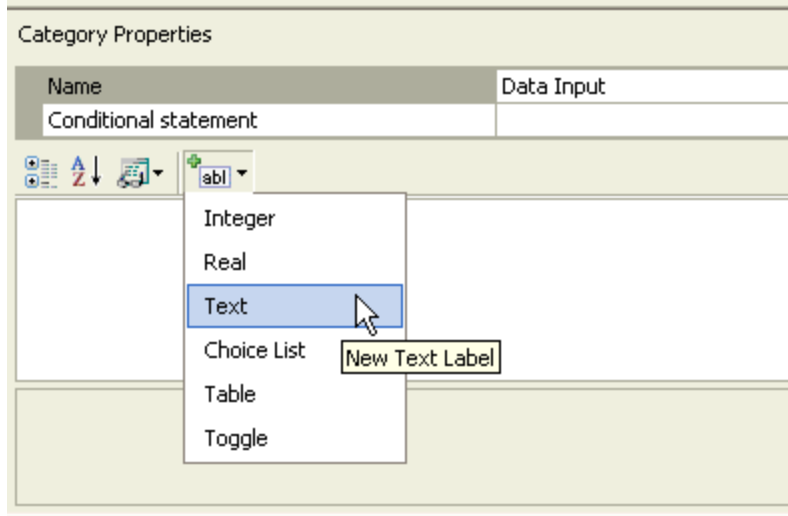
## Text Fields

Text fields are used primarily to add descriptive comments as input, or to define signals to be used as internal output variables within the component. See Internal Output Variables for more details.

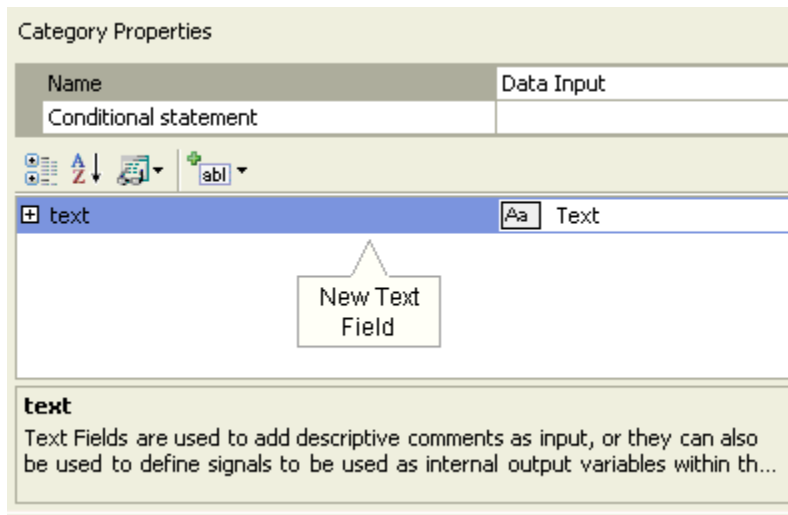
## Adding Text Fields

To add a new text field to a category page, select the category in the Categories tree and then select **Text** within the **Add Parameter Field** drop list button in the tool bar:





A new text field will appear:




## Changing Text Field Properties

Text field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the text field tree node.

Category Properties

Name	Data Input
Conditional expression	



Model Name	Aa Text
Description	Model Name
Symbol	my_model
Default value	MyModel
Group label	
Prompt text	
Help mode	Append
Regular expression	
Error text	
Conditional expression	

**Default value**  
Use this property to add a reasonable default value that will be placed in the control value when the component is first created.

The properties available are described as follows:

- **Description:** Enter a caption to act as the visible title of the text field.
- **Symbol:** Enter a symbolic name for the text field, which will be used as a variable name when addressing this field within code. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not include spaces, etc.).
- **Default Value:** Use this field to add text, which will show up as default text in the text field box.
- **Group Label:** Use this field to organize the display of the input parameters in the actual parameter dialog. All parameter fields that possess the same group name will be grouped together under the group name heading. For example, the image below shows part of an input parameter dialog, where the parameters have all been given the same group name (in this case Positive Sequence):

<b>Positive Sequence</b>	
+ve Sequence Resistance	.1781598E-4 [ohm/m]
+ve Sequence Inductive Reactance	.31388E-3 [ohm/m]
+ve Sequence Capacitive Reactance	273.5448 [Mohm*m]

- **Prompt Text:** Enter a brief statement describing the field. This text will be displayed on the actual input parameter dialog for the component.
- **Help Mode:** Select *Append* or *Overwrite*. If *Overwrite* is selected, only the help text will appear at the bottom of the dialog when the user selects this parameter. If *Append* is selected, then the prompt text will be appended to the other parameter attribute information displayed.
- **Regular Expression:** Enter an expression for the purpose of 'masking' the entered value for this field. *Regular expression* (or *Regex*) is a powerful language used for the recognition of character patterns. As the designer, you can use this language to ensure that users of your component enter data values in a specific format. For more details on regular expressions, see <http://en.wikipedia.org/wiki/Regex>.
- **Error Text:** Enter a message to be output if the expression entered in *Regular Expression* fails.

- **Conditional Expression:** Enter a conditional statement to indicate under what input conditions the text field is to be enabled. See Conditional Statements, Layers & Filters for more details.

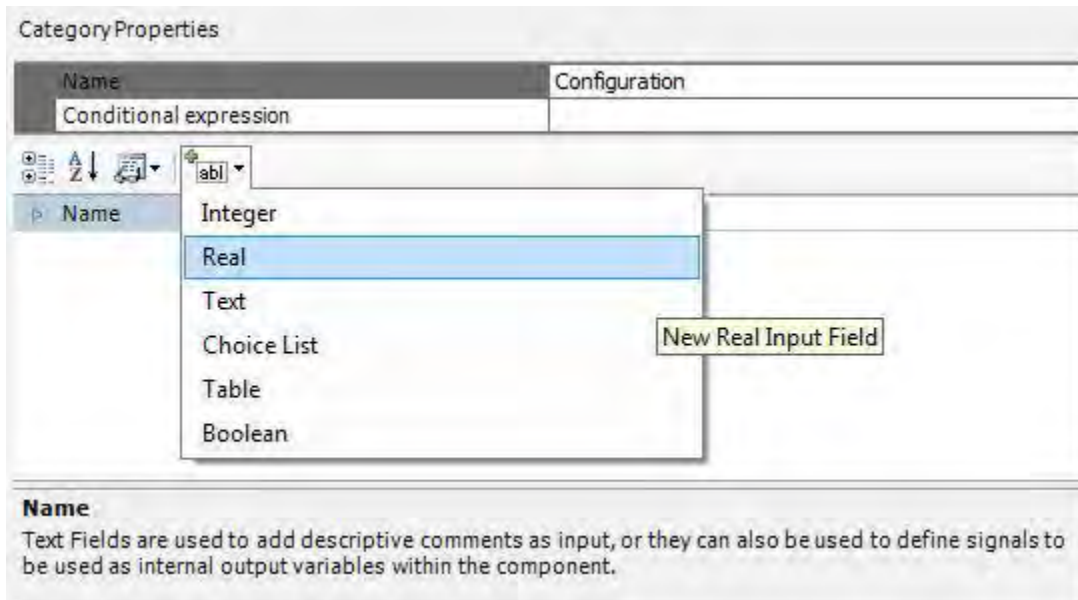
## Value Fields

Value fields allow the user to import or export numerical or signal input to and from the component. Value fields include the following numerical sub-types:

- Real
- Integer
- Logical

## Adding Value Fields

To add a new value field to a category page, select the category in the Categories tree and then select **Real**, **Integer** or **Logical** within the **Add Parameter Field** drop list button in the tool bar:



A new value field will appear:

Category Properties

Name	Configuration
Conditional expression	

Name	<input type="text" value="Aa"/> Text
untitled	<input type="text" value="R"/> Real

New Real Input Field

**untitled**  
Real Input Fields allow the user to add numerical input to the component or they can be used to import system signals into the component.

## Changing Value Field Properties

Value field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the value field tree node.

Category Properties

Name	Configuration
Conditional expression	

Name	<input type="text" value="Aa"/> Text
System Frequency	<input type="text" value="R"/> Real
Description	<b>System Frequency</b>
Symbol	<b>f</b>
Group label	
Default units	<b>Hz</b>
Minimum value	<b>+1e-308</b>
Maximum value	<b>+1e+308</b>
Dimension	1
Data type	Literal
Intent	Input
Help text	
Help mode	Append
Conditional expression	
Default value	<b>60.0</b>

**System Frequency**  
Real Input Fields allow the user to add numerical input to the component or they can be used to import system signals into the component.

The properties available are described as follows:

- **Description:** Enter a caption to act as the visible title of the value field.
- **Symbol:** Enter a symbolic name for the value field, which will be used as a variable name when addressing this field within code. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not include spaces, etc.).
- **Group Label:** Use this field to organize the display of the input parameters in the actual component parameter dialog. All parameter fields that possess the same group name will be grouped together under the group name heading.
- **Default Units:** Add a unit (if any) to represent the *Target Unit* related to this Field. See Unit System for more details.
- **Minimum / Maximum Value:** Enter the maximum and minimum value limits. If you are unsure of what these limits should be, or limits are not required, simply enter  $-1e+038$  and  $1e+038$  respectively. If value outside this range is entered, PSCAD will provide a warning in the Build or Runtime Message Panes upon compilation.
- **Dimension:** Real and Integer type parameters may be defined as single-dimension arrays. Simply enter the dimension of the array here. Note that non-scalar parameters (i.e. dimension > 1) must be set as *Data Type | Variable*.
- **Data Type:** Select **Literal**, **Constant** or **Variable**. The selection of data type is very important. Please see the following section entitled Value Field Data Types for a complete description of each.
- **Intent:** Select whether or not this parameter is to function as an input or an output parameter. The intent setting is important to ensure well formed Fortran code is generated for this component when the project is built.
- **Help Text:** Enter a brief statement describing the field. This text will be displayed on the actual input parameter dialog for the component.
- **Help Mode:** Select *Append* or *Overwrite*. If *Overwrite* is selected, only the help text will appear at the bottom of the dialog when the user selects this parameter. If *Append* is selected, then the prompt text will be appended to the other parameter attribute information displayed.
- **Conditional Expression:** Enter a conditional statement to indicate under what input conditions the input field is to be enabled. See Conditional Statements, Layers & Filters for more details.
- **Default Value:** Use this field to add a value, which will appear as the default value in the input field parameter display. Assume that a user who is not familiar with this component might simply accept your default value for lack of a better value. So, do your best to enter an appropriate value here.

## Value Field Data Types

Value fields can be subdivided into two types: Real and Integer. These data types are quite familiar in programming languages and can be readily defined. In PSCAD, Real, Integer and Logical type value fields can be further categorized into three sub-types: *Literal*, *Constant* and *Variable*.

PSCAD versions prior to X4 allowed only type *Literal* and *Variable* value fields. Seasoned users will remember the use of the *Allow Signal Names* option in the Value Field dialog: By default, all value fields were *Literal*, but could be made *Variable* by selecting this option.

As a direct by-product of the multiple instance module (MIM) feature, a third input field sub-type was created and dubbed a *Constant*. A *Constant* type input field is a hybrid of sorts in that it possesses key properties of the other two data types. For example, a *Constant* type field cannot be modified during runtime like a *Variable*, but can accept the name of a previously defined signal, or an actual number (*Literal*).

A brief definition and example is given below for each data type:

- **Literal:** *Literal* type value fields will accept only fixed, numeric values. For example: 23, 657.29, -33.8, or -1 are all valid inputs. Literals are defined at build time, and remain fixed throughout the simulation.

- **Constant:** Constant type value fields will accept only fixed input. However, unlike a *Literal* type, it will also accept a signal name as its value. The signal value must be from a source that is fixed (non-variable). For example: *freq, my\_signal, out2*, as well as all the examples given for a *Literal* type above. *Constants* are defined at build time, and remain fixed throughout the simulation.
- **Variable:** *Variable* type value fields accept both numeric and non-numeric input. Input values may remain fixed, or they can vary throughout the simulation. This value type is equivalent to having the *Allow Signal Names* option enabled in PSCAD versions previous to X4.

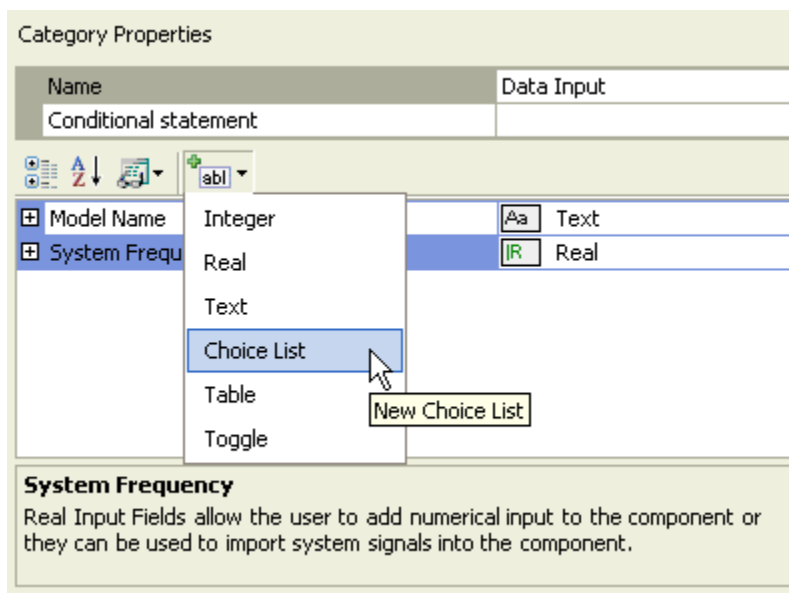
See Multiple Instance Modules for more details on using *Constant* type value fields.

## Choice Lists

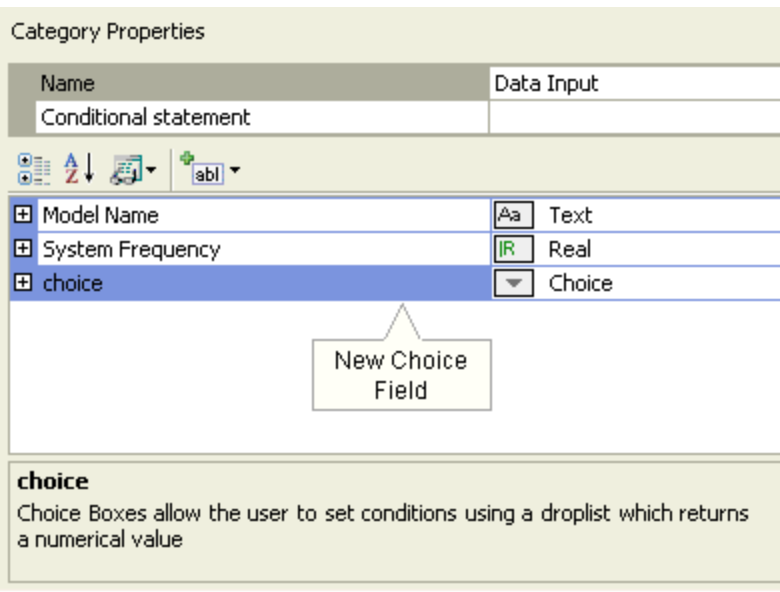
The purpose of a choice list is to allow the user to set conditions according to a selected choice, which will possess an associated integer number for use internally in conditional statements.

### Adding Choice Lists

To add a new choice to a category page, select the category in the Categories tree and then select **Choice** within the **Add Parameter Field** drop list button in the tool bar:

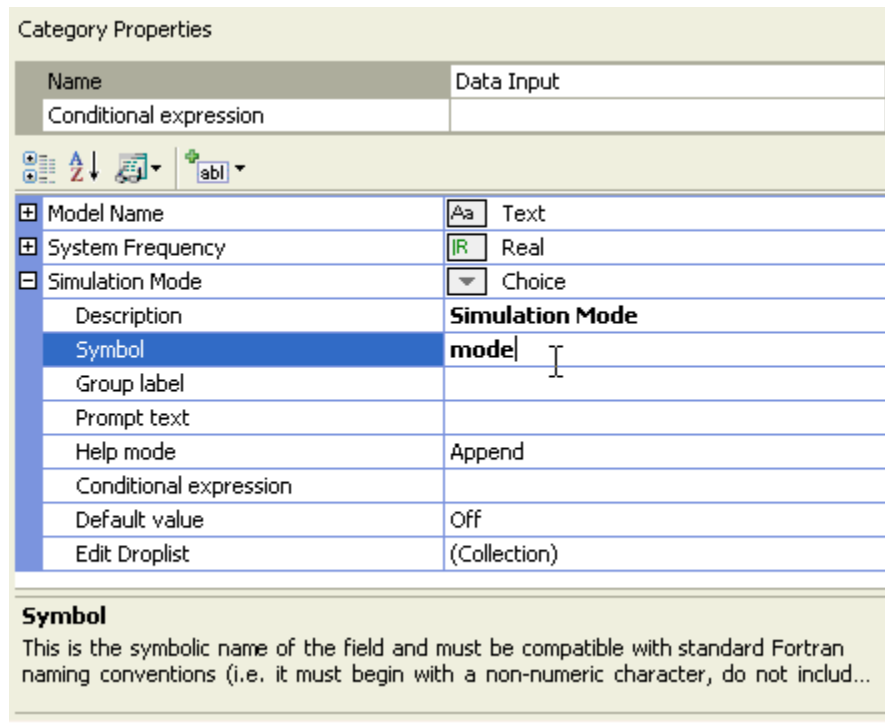


A new input field will appear:



## Changing Choice List Properties

Choice list properties can be adjusted directly on the category page. Left-click on the [+] box to expand the choice list tree node.



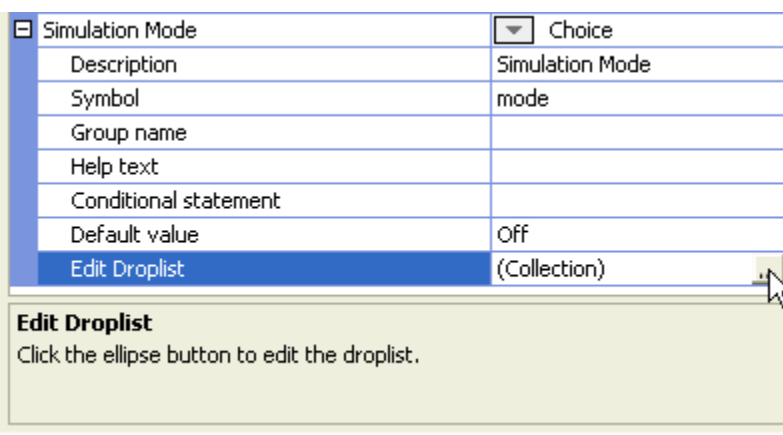
The properties available are described as follows:

- **Description:** Enter a caption to act as the visible title of the choice list field.

- **Symbol:** Enter a symbolic name for the choice list, which will be used as a variable name when addressing this field within code. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not include spaces, etc.).
- **Group Label:** Use this field to organize the display of the input parameters in the actual component parameter dialog. All parameter fields that possess the same group name will be grouped together under the group name heading.
- **Prompt Text:** Enter a brief statement describing the field. This text will be displayed on the actual input parameter dialog for the component.
- **Help Mode:** Select *Append* or *Overwrite*. If *Overwrite* is selected, only the help text will appear at the bottom of the dialog when the user selects this parameter. If *Append* is selected, then the prompt text will be appended to the other parameter attribute information displayed.
- **Conditional Expression:** Enter a conditional statement to indicate under what input conditions the choice list is to be enabled. See Conditional Statements, Layers & Filters for more details.
- **Default Value:** Use this field to add an integer (ex. 0, 1, 2, etc.), which will correspond to the default choice from the list.
- **Edit Drop List:** This field is used to invoke the *Drop List Editor*. This editor is used to construct the actual list of choices. See the following section for details.

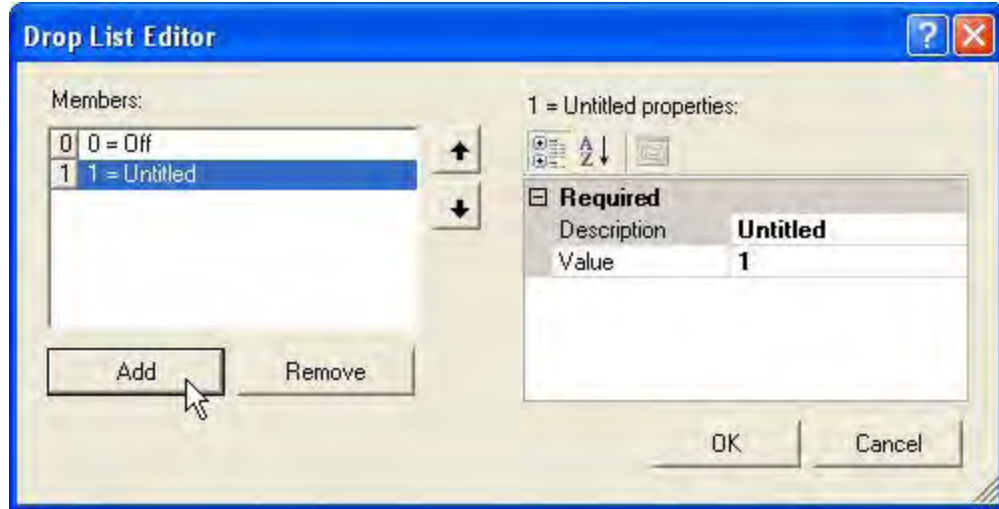
## Adding Choices to a Choice List

Choice lists are constructed using the *Drop List Editor*. To open the editor, left-click on the [+] box to expand the choice list tree node, select the **Choice List** field and then left-click the [...] button.

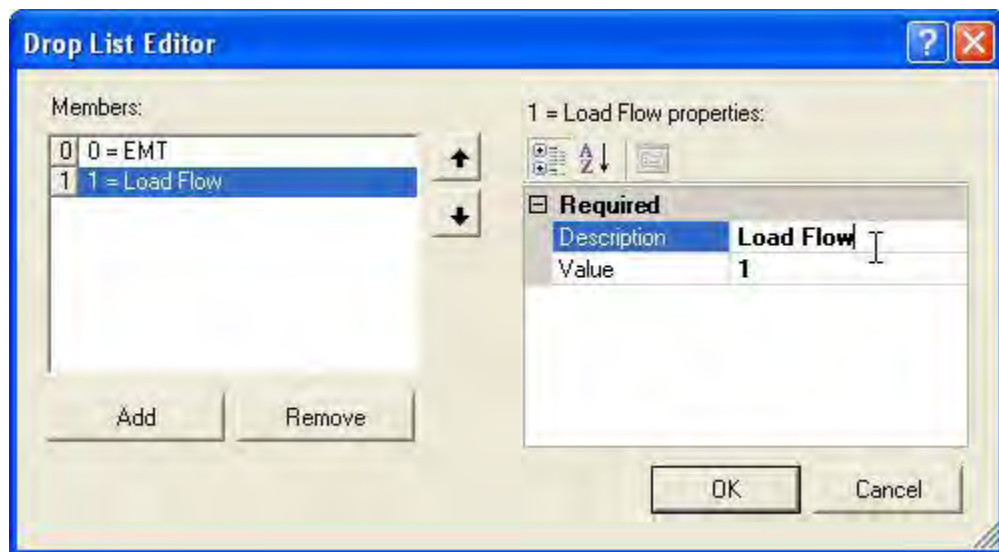


The Drop List Editor will appear. To add choices to the list, click the **Add** button.





The contents of any choice in the list can be modified in the editor window to the right: Within the **Members** list, left-click on the choice name to select it. To edit the properties of the selected choice, left-click either the **Description** and/or the **Value** field.



Note that when adding choices to the choice box, the most important thing to remember is that a *unique* integer must be assigned to each entry in the **Members** choice list. This is accomplished by modifying the **Value** field as described above. It is this assigned number that will be associated with the Symbol name of this choice list.

To delete a choice in the choice list, left-click on a choice within the **Members** list to select it. Click the **Remove** button.

## Ordering Choices in a Choice List

Once multiple choices exist in a list, they may be re-ordered as follows: Select the desired choice from the Members list. Click the up or down arrows to move it up and down the list.

## Table Fields

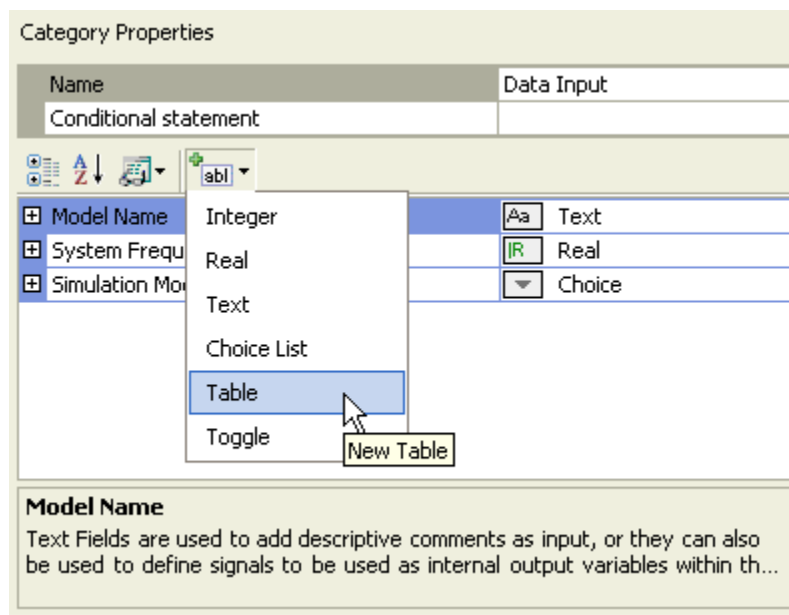
Table fields are specially designed to allow the user to enter data in vector or matrix form directly as a component parameter. Table fields include the following numerical sub-types:

- Real
- Integer

**NOTE:** Table fields can currently be used only in standard, non-module components.

## Adding Table Fields

To add a new table field to a category page, select the category in the Categories tree and then select **Table** within the **Add Parameter Field** drop list button:



A new table field will appear:

Category Properties

Name	Data Input
Conditional statement	

Model Name	Text
System Frequency	Real
Simulation Mode	Choice
table	Table

New Table Field

**table**  
 Tables allow the user to create and numerical values using a table interface.

## Changing Table Field Properties

Table field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the table field tree node.

Category Properties

Name	Data Input
Conditional expression	

Model Name	Text
System Frequency	Real
Simulation Mode	Choice
XY Points	Table
Description	<b>XY Points</b>
Symbol	<b>xy_points</b>
Group label	
Prompt text	
Help mode	Append
Conditional expression	
Data type	Real
Decimal places	-1
Edit Columns	(Collection)
Edit Rows	(Collection)

**Symbol**  
 This is the symbolic name of the field and must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not includ...

The properties available are described as follows:

- **Description:** Enter a caption to act as the visible title of the table field.
- **Symbol:** Enter a symbolic name for the table field, which will be used as a variable name when addressing this field within code. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not include spaces, etc.).
- **Group Label:** Use this field to organize the display of the input parameters in the actual component parameter dialog. All parameter fields that possess the same group name will be grouped together under the group name heading.
- **Prompt Text:** Enter a brief statement describing the field. This text will be displayed on the actual input parameter dialog for the component.
- **Help Mode:** Select *Append* or *Overwrite*. If *Overwrite* is selected, only the help text will appear at the bottom of the dialog when the user selects this parameter. If *Append* is selected, then the prompt text will be appended to the other parameter attribute information displayed.
- **Conditional Expression:** Enter a conditional statement to indicate under what input conditions the table field is to be enabled. See Conditional Statements, Layers & Filters for more details.
- **Data Type:** Select **Real** or **Integer**.
- **Decimal Places:** Enter the precision of the input data values to a maximum of 6 decimal places. If the values are of arbitrary precision, select *-1*.
- **Edit Columns:** Define the columns for the vector or matrix. This will invoke the *Column Editor* dialog, which functions in the same manner as the *Drop List Editor* described in the Choice Lists section above.
- **Edit Rows:** Define the rows for the vector or matrix. This is also where you can enter the default value of each matrix element. To add a row, press the **Add** button. Default values may be entered directly in each element field.

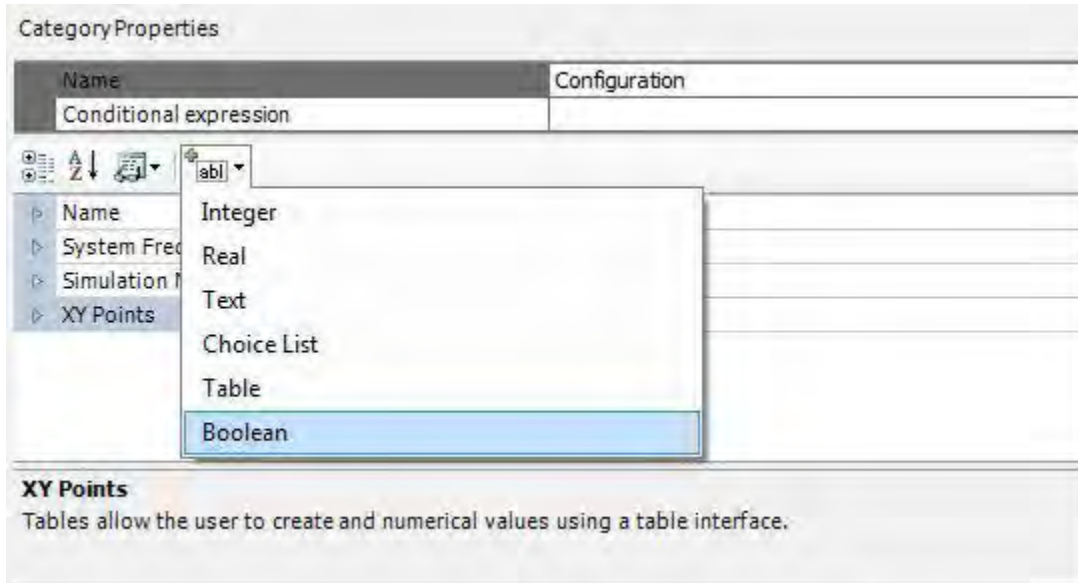
	X	Y
1	1.1	0
2	0	0

## Boolean Fields

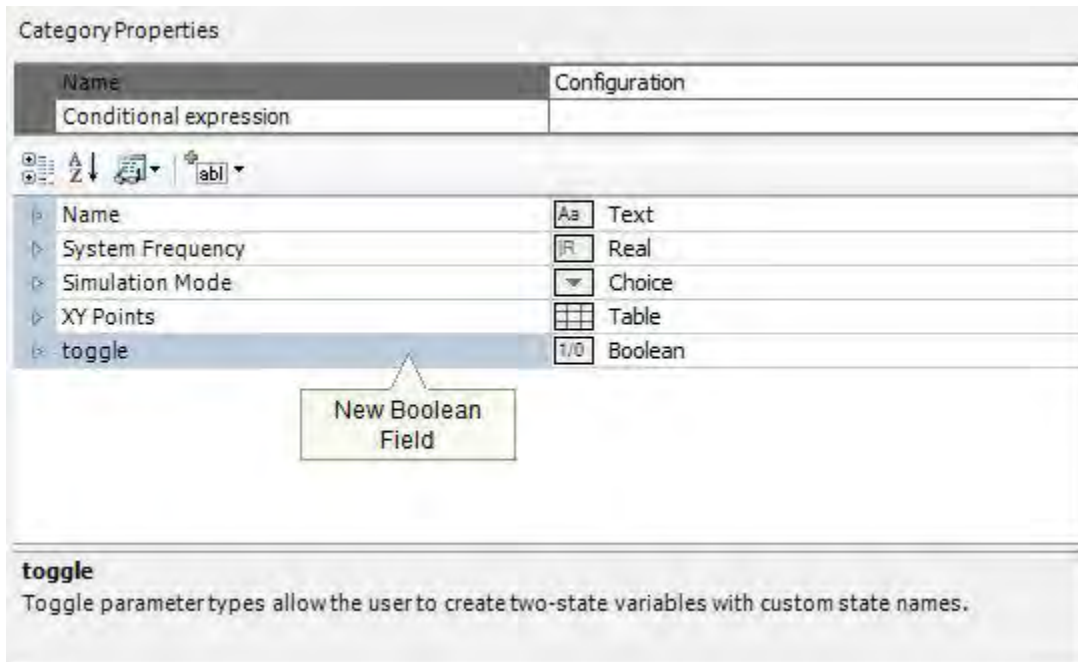
Boolean fields are designed for simple on/off or yes/no parameters.

## Adding Boolean Fields

To add a new boolean field to a category page, select the category in the Categories tree and then select **Boolean** within the **Add Parameter Field** drop list button:



A new boolean field will appear:



## Changing Boolean Field Properties

Boolean field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the boolean field tree node.

Category Properties

Name	Configuration
Conditional expression	
▶ Name	Aa Text
▶ System Frequency	FR Real
▶ Simulation Mode	▼ Choice
▶ XY Points	Table
▲ Block/Deblock	1/0 Boolean
Description	<b>Block/Deblock</b>
Symbol	<b>block</b>
Group label	
Help text	
Help mode	Append
Conditional expression	
Text value (enabled state)	<b>Block</b>
Text value (disabled state)	<b>Deblock</b>
Default value	Block

**Block/Deblock**  
Toggle parameter types allow the user to create two-state variables with custom state names.

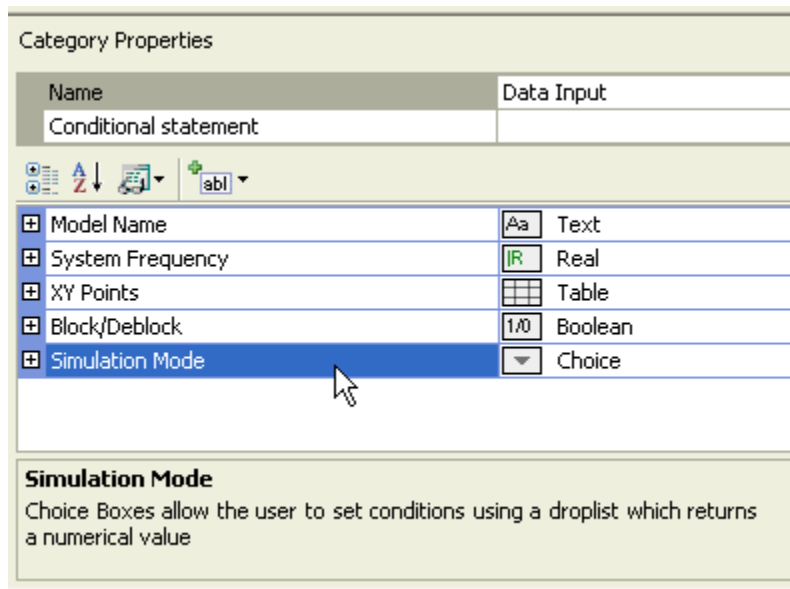
The properties available are described as follows:

- **Description:** Enter a caption to act as the visible title of the toggle field.
- **Symbol:** Enter a symbolic name for the toggle field, which will be used as a variable name when addressing this field within code. Note that this name must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not include spaces, etc.).
- **Group Label:** Use this field to organize the display of the input parameters in the actual component parameter dialog. All parameter fields that possess the same group name will be grouped together under the group name heading.
- **Help Text:** Enter a brief statement describing the field. This text will be displayed on the actual input parameter dialog for the component.
- **Help Mode:** Select *Append* or *Overwrite*. If *Overwrite* is selected, only the help text will appear at the bottom of the dialog when the user selects this parameter. If *Append* is selected, then the prompt text will be appended to the other parameter attribute information displayed.
- **Conditional Expression:** Enter a conditional statement to indicate under what input conditions the toggle field is to be enabled. See Conditional Statements, Layers & Filters for more details.
- **Text Value (Enabled State):** Enter a name for the enabled (true) state.

- **Text Value (Disabled State):** Enter a name for the disabled (false) state.
- **Default Value:** Use this field to select which of the two toggle states is to be the default.

## Ordering Input Fields within a Category Page

Input fields, text labels and choices may be arranged at will. Left-click and hold the desired field in the Input Fields list. Drag the field to where you want it placed and then release the mouse button.



## Cut, Copy, Paste and Delete of Category Fields

You can cut, copy or paste an input field by using one of two methods:

- Right-click on the field and select **Cut** or **Copy** from the pop-up menu. Right-click over a blank area of the Input Fields list and select **Paste** from the pop-up menu.
- Left-click on a field and press **Ctrl + x** to cut, **Ctrl + c** to copy and **Ctrl + v** to paste.

To delete a field, select it and press the **Delete** key, or right-click and select **Delete**.

## Undo and Redo

Undo/redo is presently not supported in the Parameters section.

## Conditional Statements, Transparencies and Filters

All objects and fields associated with either the Graphic or the Parameters sections will possess an input field labelled *Conditional Statement*. Within these fields, the user may add statements to either enable or disable the object, or make the object visible or invisible according to the logic specified in the statement.

## Conditional Statements

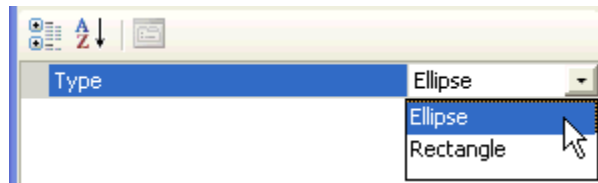
The primary purpose of conditional statements is to either enable or disable an object (such as an input field), or to make an object visible or invisible (such as a graphic object).

The user may construct conditional statements by using both *Arithmetic* and *Logical Operators*, where the condition values themselves are always based on the value of a choice list. Through conditional statements, the operation and appearance of the component becomes instance-based and is controllable by the user.

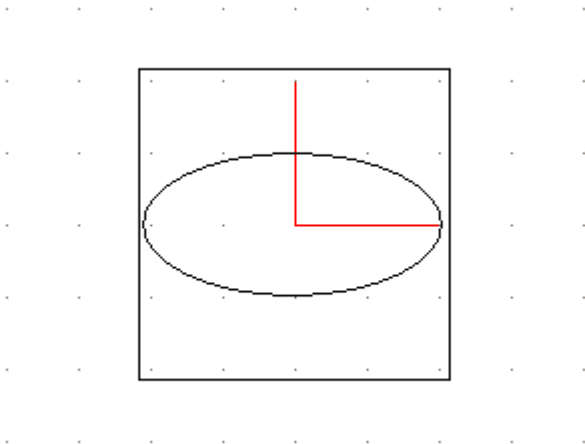
### EXAMPLE 9-2:

A component designer wants to change the appearance of a component according to user-selected input. The component graphic is to be either an ellipse or a rectangle, depending on a choice list with Symbol name Type. The choice list specifies two choices: *Ellipse*, which is given the integer value 0, and *Rectangle*, which is given the integer value 1.

In the component Parameters section, the designer adds a single choice list:

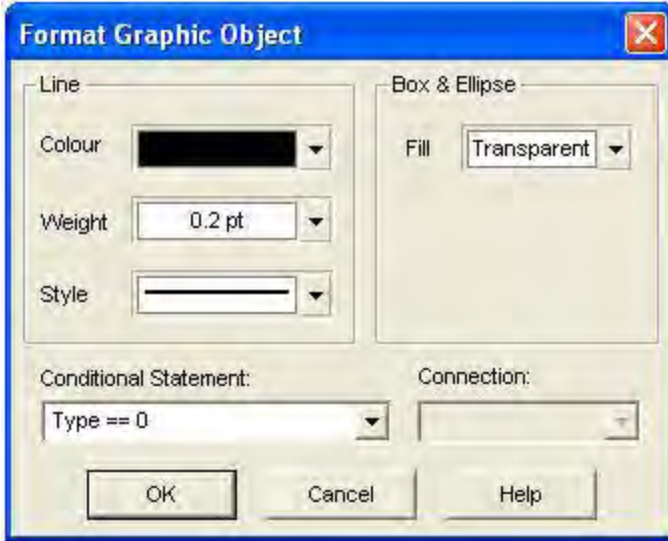


In the component Graphic section, the designer draws a simple ellipse and a simple rectangle:

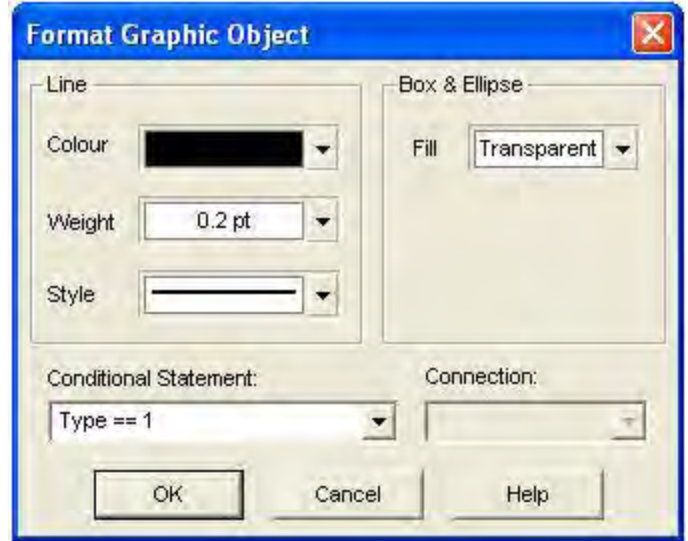


In the properties dialog for the Graphic Objects, the following text is added to the respective **Conditional Statement** input fields:





Ellipse Property Settings



Rectangle Property Settings

If a user selects *Ellipse* in the choice list, only the ellipse object will appear in the component graphic. If *Rectangle* is chosen, only the rectangle will appear.

Conditional statements are not limited to a single logical truth. It is possible to use several logical conditions in one statement. For example, say there was another choice list added to the component of Example 9-2 with Symbol name *Type2*. Now assume that in order for the ellipse object to be visible on the component graphic, it is required that *Type2* have a value of 3 (in addition to *Type* having a value of 0). Then the following conditional statement would need to appear in **Conditional Statement** field of the ellipse object:

```
(Type == 0) && (Type2 == 3)
```

The above states: If *Type* equals 0 and *Type2* equals 3, then make the ellipse visible.

Arithmetic Operators may also be included in conditional statements. For example, the following statement is also valid:

```
(Type + Type2 == 3)
```

The above states: If the sum of *Type* and *Type2* equals 3, then make the ellipse visible.

**NOTE:** Care must be exercised when using the divide (/) function, as a *divide by zero* error can occur.

## Transparencies

As component graphics become larger and more complicated, the graphical working environment can become quite unruly – especially when many conditional statements are used.

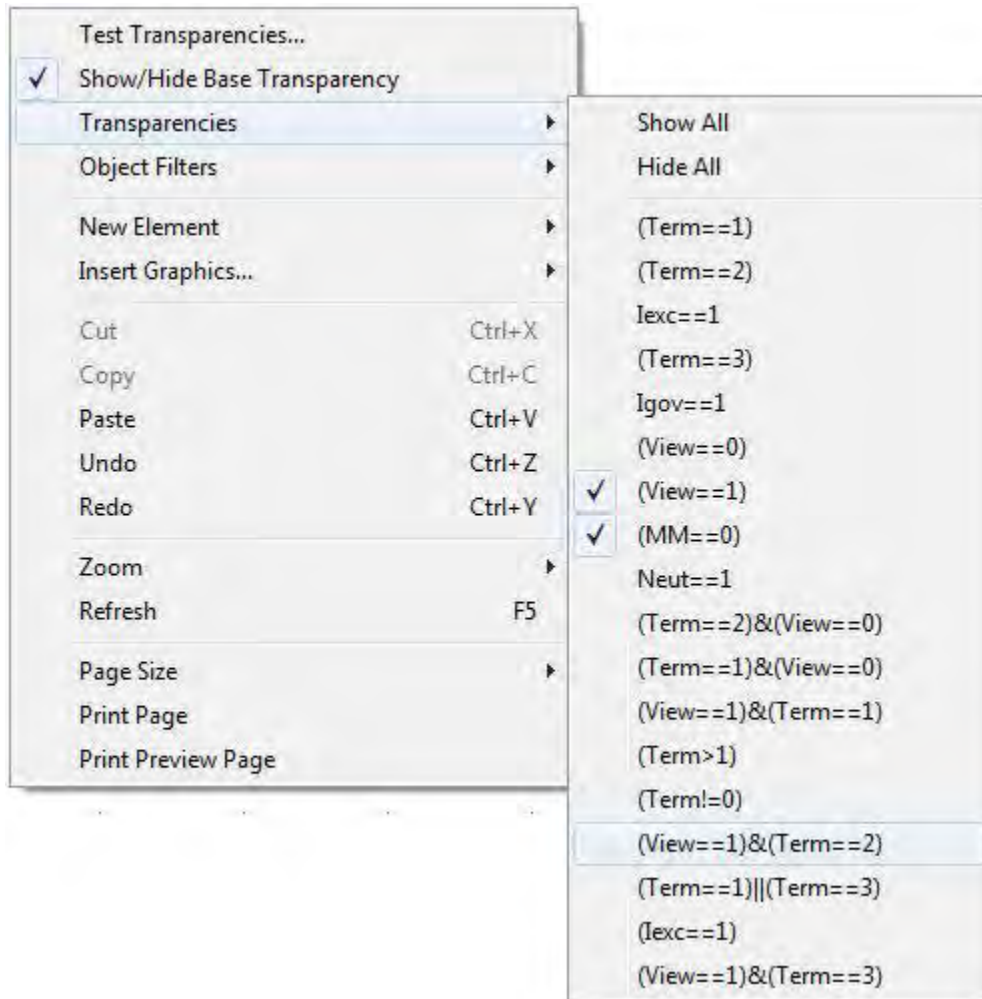
Fortunately, there is a way to avoid the potential clutter by utilizing the graphical layers available in the Graphic section. Transparencies are based solely on conditional statements, in that whenever a unique conditional statement is entered into an object or field, a new graphical transparency is created. Any other graphic object, which utilizes an identical conditional statement, will also be visible in that particular transparency.

**NOTE:** Be sure when adding conditional statements that the statements themselves are identical in format, as well as logical truth. If the statements vary slightly in format, a separate transparency will be created, even though the statements may indicate the same thing.

## Viewing Transparencies

By default, only the transparencies visible on the graphic of a particular component instance will be initially visible in the Graphic canvas when the component definition is edited. Although a component definition may have several instances, only the transparencies present on the instance used to access the definition will be visible by default. If the definition is accessed from the definition itself in the Workspace, then the transparencies will be based on the component default conditions.

To adjust the transparency visibility, move the mouse pointer over a blank area of the Graphic canvas. Right-click and select **Transparencies**. The resulting sub-menu will list all of the available transparencies (or conditional statements) that currently exist in the component graphics.

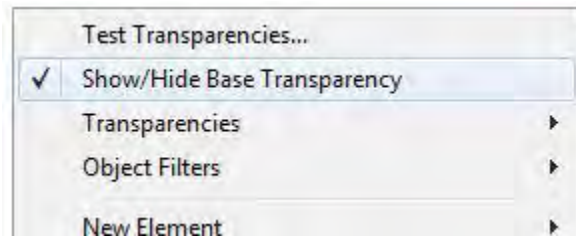


Simply select or de-select each transparency to make visible or invisible respectively. You can also select **Show All** or **Hide All** or, press the **Show All/Hide All** button on the **Filtering** tab of the ribbon control bar to toggle the visibility of all transparencies.:



## Base Transparency

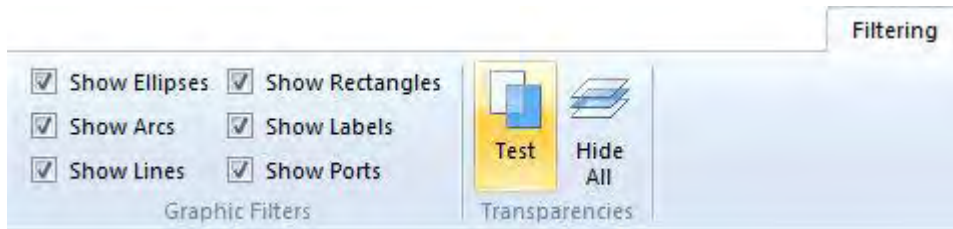
The base transparency contains any graphical objects and port connections that do not possess a defined conditional statement (i.e. are not part of a defined transparency). The base transparency may be toggled on and off using the pop-up menu: Move the mouse pointer over a blank area of the Graphic canvas. Right-click and select **Show/Hide Base Transparency**.



## Testing Transparencies

Instead of turning transparencies on and off through the right-click menu (which can become cumbersome as the number of transparencies increases), you can set and test the visible transparencies directly using the component parameter dialog.

Right-click on a blank part of the canvas and select **Test Transparencies...** from the pop-up menu. Or, press the **Test** button on the **Filtering** tab of the ribbon control bar:

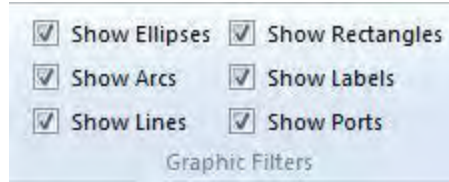


This feature provides a fully functional preview of the parameter dialogs as they would be seen by a user. Simply set all the choice lists to the desired settings and click the **OK** button – the graphical transparencies pertaining to that setting will become visible in the Graphic canvas.

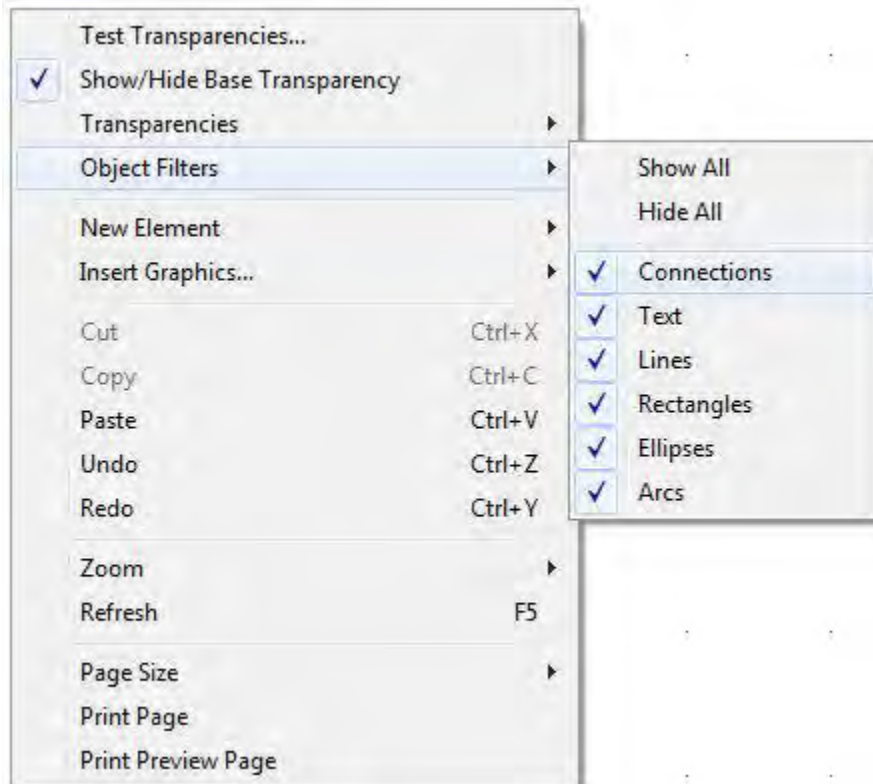
## Graphic Filters

In addition to graphical transparencies, you may also use *Graphic Filters* to help alleviate graphical clutter. Graphic filters allow the user to view objects based on the graphic object type (i.e. port connections, text labels, etc.).

To adjust graphic filters, the most straightforward method is to use the *Graphic Filters* section in the **Filtering** of the ribbon control bar.



Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic canvas. Right-click and select **Graphic Filters**.



## The Script Section

Contained within the Script section is the heart of the definition, where the fundamental purpose of the component is defined. The Script section is utilized in non-module components only.

This section of the definition is used primarily for lower level code entry, and consists of a variety of *Segments*, each performing a specific function. Not all segments need to exist in every component, but the most commonly used segments are *Fortran*, *Computations* and *Branch*, and these are included by default in every newly created definition.

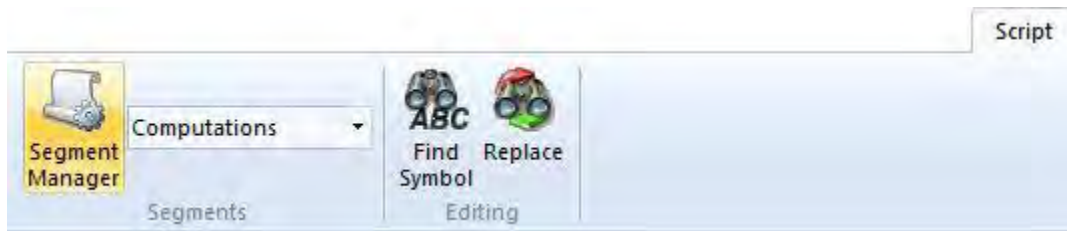
The segments that your component definition will require depend on what function your component is to perform. Are pre-calculations required? Is it an electrical component? Is there source code involved? The following sections describe each available script segment in detail.

## Segments

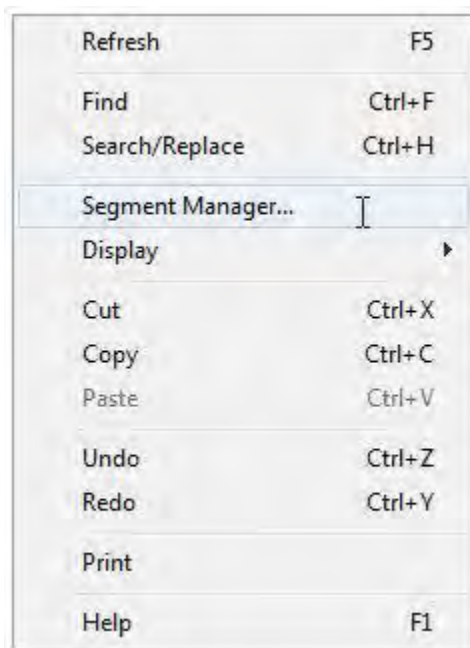
Segments are ordered similar to pages in a book. Each segment is itself a simple text editor, where data, script and other code is added to perform a specific task. Segments may be added or deleted at will, but must follow a strict naming convention.

### Managing Segments

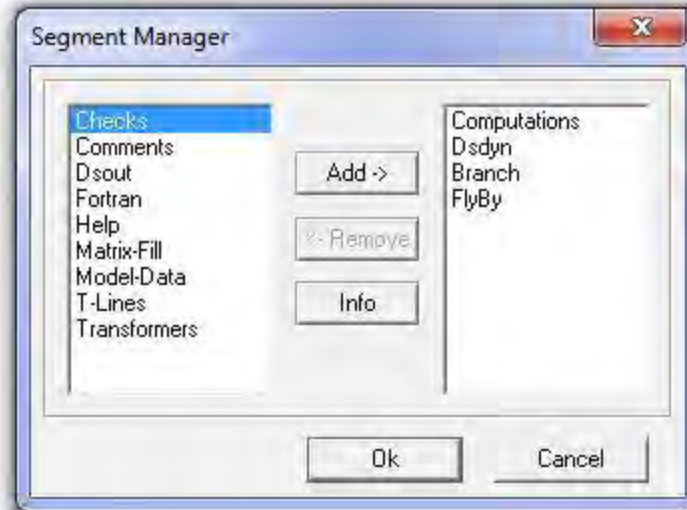
The *Segment Manager* enables users to add or remove segments from within a simple dialog. To invoke the segment manager, the most straightforward method is to use the Segment Manager button under the **Script** tab of the ribbon control bar:



Another method is to use the right-click menu: Move the mouse pointer over a blank area of the segment page. Right-click and select **Segment Manager...**



The segment manager dialog will appear as shown below:

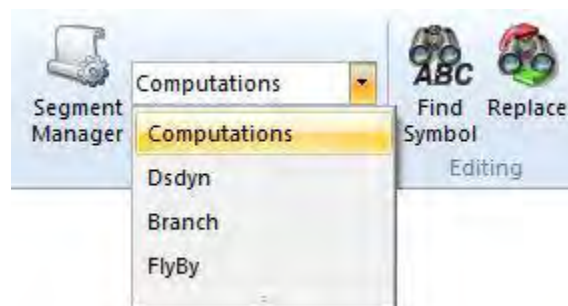


The list on the left-side of the dialog displays the segments available to add to this definition, while the right-side lists are the segments already in the definition. To add or remove segments to/from the definition, simply select the **Add->** or **<- Remove** buttons. Once satisfied with the selection, click the **OK** button. If you chose to remove any existing segments, a dialog will appear ask you to confirm the removal. Press the **Cancel** button to cancel the operation.

**NOTE:** Deleted segments will be lost if removed, as removal cannot be undone!

## Viewing Segments

Once you have added more than one segment, there may be a need to navigate through them. Use the drop list in the *Segments* section under the **Script** tab of the ribbon control bar:



## Segment Types

There are several segment types available. Each exists for a specific purpose, but are only required if that specific function is needed as part of the component design. Most of the time, a definition will only require two or three of segments.

**NOTE:** Although this section does provide an assortment of simple examples, there is a huge wealth of examples in the master library project itself. Simply edit any master library definition (right-click on the component and select **Edit Definition...**) to study its contents.

## Computations

The *Computations* segment is an environment provided for the pre-processing of component input data, as well as creating new internal variables. Although component parameter input may be in the most convenient form for the user, it may not be so convenient for the definition itself. For example, the definition may require the system frequency in radians per second, but the input data field may ask the user for the same quantity in Hertz.

The *Computations* segment is the first segment considered by the compiler when the component is compiled. Due to this fact, any quantities defined here may be substituted elsewhere in any of the other segments. This segment allows both mathematical and logical expression evaluation.

**NOTE:** All script processed within the *Computations* segment is done so before any Fortran code is constructed for EMTDC, and only considered once. Therefore, quantities defined here are static for the remainder of the simulation.

A typical *Computations* segment entry must have the following standard format:

```
<DataType> <Name> = <Expression>
```

Where,

- <DataType> may be either a REAL or INTEGER type value. If omitted, it is assumed to be REAL
- <Name> is the name of the constant
- <Expression> is a mathematical or logical expression containing only constant parameters

---

### EXAMPLE 9-3:

A designer wants to convert an input parameter with symbol name *Vset* in *kV*, to a per-unit value called *SetPU*, before it is used in component Fortran code. An additional input parameter with symbol name *Vbase* in *kV* has also been defined for the system voltage base.

▷ Name	Aa Text
▾ System Voltage	IR Real
Description	System Voltage
Symbol	Vset
Group label	
Default units	kV
Minimum value	-1e+308
Maximum value	+1e+308
Dimension	1
Data type	Literal
Intent	Input
Help text	
Help mode	Append
Conditional expression	
Default value	230.0
▾ Base Voltage	IR Real
Description	Base Voltage
Symbol	Vbase
Group label	
Default units	kV
Minimum value	-1e+308
Maximum value	+1e+308
Dimension	1
Data type	Literal
Intent	Input
Help text	
Help mode	Append
Conditional expression	
Default value	230.0

Input Field Property Settings for *Vset* & *Vbase*

The Computations segment could then contain something similar to the following:

```
REAL SetPU = Vset / Vbase
```

In the above example, the designer has defined a new REAL constant entitled *SetPU*. This constant may now be used in any other segment in the Script section, both in equations and/or in logical expressions.

## Branch

The *Branch* segment is primarily for the provision electric branch information to the EMTDC Electric Network Conductance Matrix: It is essentially an input portal for control of the EMTDC Equivalent Conductance (GEQ) Interface. *Branch* design is accomplished by specifying the type and size of passive elements (i.e. lumped R, L and/or C), and to which electrical port connections they fall between.



A single *Branch* statement represents a single electrical branch, to which is associated an impedance  $Z = R + jX$ , where  $Z$  is derived from a combination of series-connected, lumped R, L and/or C values. The *Branch* segment may also be used to define switching branches and those that contain an ideal voltage source (i.e. infinite bus).

A typical Branch Segment entry must have the following standard format:

```
[<Branchname> = ] $<TO>
$<FROM> [<Keyword>] [$]<R> [$]<L> [$]<C>
```

Where,

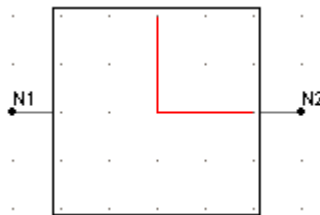
- <TO> and <FROM> are the symbol names of the electrical port connections defined in the Graphic section
- <R> is the resistance value of the branch [ $\Omega$ ]
- <L> is the inductance of the branch (optional) [H]
- <C> is the capacitance of the branch (optional) [ $\mu\text{F}$ ]
- <Branchname> = may be used to give the branch a name, which can be used to refer to the branch (instead of its nodes) in other sections. Note that many EMTDC subroutines require the branch name as an argument.
- <Keyword> can be either *SOURCE* or *BREAKER* and is explained in the following sections
- \$ is the Substitution Prefix Operator (\$)

**NOTE:** If a branch does not contain all three types of R, L and C elements, simply substitute *0.0* for the value of the element that is not present.

---

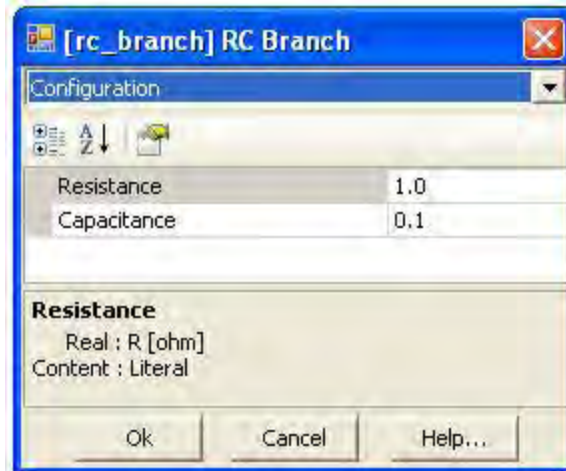
#### EXAMPLE 9-4:

A designer wants to define a component, which represents an RC series branch, connected in parallel with a purely inductive branch, between electrical port connections *N1* and *N2*. These two port connections have already been defined in the Graphic section of the definition.



Component Graphic with Electrical Connections *N1* and *N2*

The actual values of the R and C elements are to be entered though input fields (with symbol names *R* and *C* respectively) in the Parameters section of the definition. The inductance value is not accessible to the user and is given a static value of *0.001 H* directly.

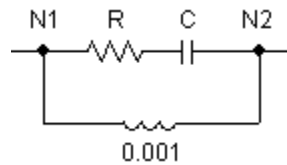


Component Parameters Dialog

The *Branch* segment would then contain the following two statements:

```
$N2 $N1 $R 0.0 $C
$N2 $N1 0.0 0.001 0.0
```

These statements are equivalent to the following schematic:



The designer then decides to refer to the branches by branch name in other sections of the definition. To achieve this, the names *BRN1* and *BRN2* are defined. The designer modifies the above branch statements as follows:

```
BRN1 = $N2 $N1 $R 0.0 $C
BRN2 = $N2 $N1 0.0 0.001 0.0
```

**NOTE:** If *BRN1* and *BRN2* are used in other segments, they must be preceded by the Substitution Prefix Operator (\$).

The branch may also contain an inherent ideal source in series with the passive elements. The *Branch* segment syntax allows you to indicate to the application that there is a voltage source in a particular branch as follows:

```
[<Branchname> = ]  $<TO> $<FROM>  SOURCE  [$]<R>  [$] [<L>]  [$] [<C>]
```

Now that PSCAD knows that there is a source in this branch, control of the source (i.e. magnitude, etc.) is then up to the user. One way this can be accomplished is by defining the EMTDC internal variable EBR every time step, directly in the model code itself.

**NOTE:** For more examples on this, edit the definition of the Voltage Source Model 2 in the master library project.

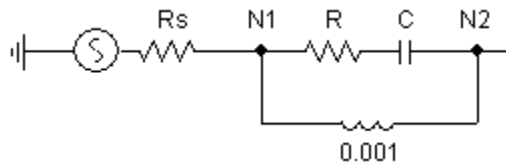
EXAMPLE 9-5:

A designer adds a resistive branch containing an ideal voltage source to the component in Example 9-4, which is to be connected between node *N1* and a ground electrical port connection *GND*. The source possesses a source resistance *RS*, which can be entered as an input parameter.

The *Branch* segment could then contain something similar to the following:

```
BRNS = $N1  $GND SOURCE  $RS  0.0  0.0
BRN1 = $N2  $N1          $R   0.0  $C
BRN2 = $N2  $N1          0.0  0.001  0.0
```

These statements are a schematic equivalent to:



The *Branch* segment syntax also allows you to indicate to the application that a particular branch is a switching branch:

```
[<Branchname> = ]  <TO> <FROM>  BREAKER  <Initial_Value>
```

Where,

- <Initial\_Value> is the initial resistance of the switch [ $\Omega$ ]. This value is used for initialization purposes and does not affect the simulation. A good default value to use is 1.0.

## Fortran

The *Fortran* segment is where any Fortran code, defining what the component is to model, is placed. Code entered in this segment should either exist in the form of standard Fortran 90, or Definition Script (or a combination of these two). It is also possible to define a function, or call an external subroutine from this segment.

When using actual Fortran code in this segment, it is important to note that all lines should be preceded by the allotted six spaces.

### EXAMPLE 9-6:

The following code is a simple example of how standard Fortran would appear in the *Fortran* segment. This code defines an instantaneous voltage source magnitude  $V_T$ , according to the value of an input parameter *Type*. If *Type* = 0, then the magnitude is calculated using the standard equation for a sinusoidal source with phase shift:

$$v(t) = V_{\text{peak}} \cdot \sin(2\pi f \cdot t + \phi)$$

If *Type* = 1, then the source magnitude is a constant given by *VDC*. *FREQ* and *PHI* are also input parameters defined in the Parameters section of the definition. *TWO\_PI* is an EMTDC internal constant.

```
! Calculation of an AC or DC voltage source magnitude
! -----
!
!23456 Remember to precede code by at least six spaces!
!
      IF ( $TYPE .EQ. 0 ) THEN
!
! AC source
!
      $VT = $VPEAK*SIN( TWO_PI*$FREQ*TIME + $PHI )
!
      ELSE
!
! DC source
!
      $VT = $VDC
```

```

!
      ENDIF
!

```

This same code could also be written as a combination of standard Fortran and Definition Script as follows:

```

! Calculation of an AC or DC voltage source magnitude
! -----
!
!23456 Remember to precede code by at least six spaces!
!
#IF $Type == 0
!
! AC source
!
      $VT = $VPEAK*SIN( TWO_PI*$FREQ*TIME + $PHI )
!
#ELSE
!
! DC source
!
      $VT = $VDC
!
#ENDIF
!

```

Note that the above two examples have been simplified slightly. Generally speaking, the value for the  $2\pi ft$  function should never be allowed to simply grow larger and larger as the simulation progresses. It should be reset to zero whenever  $2\pi ft = 2\pi$  is reached.

There are pros and cons to adding in-line Fortran code directly into the segment, as opposed to calling a subroutine or function. Here are some key points to consider:

- All EMTDC Internal Variables may be utilized, without the need to declare them or add include files, when coding in the *Fortran* segment.
- Definition Script may only be utilized when coding directly in the Fortran segment.
- All code placed in the *Fortran* segment will be added directly to the module Fortran file (<module>.f), when the project is compiled. If there is a large amount of code and/or there are many instances of this definition in the module, the module Fortran file can become huge and difficult to debug. In such cases, it would be wise to create a function or subroutine, which could be simply called from the Fortran segment. Also, if Fortran line numbering is used in your code, these same number may appear multiple times in the Fortran file (<module>.f), causing compiler errors.

The *Fortran* segment is a 'smart' segment, in that code placed here will be intelligently placed in the EMTDC System Dynamics, in order to avoid problems with time step delays. In other words, PSCAD will choose to place the code in either the DSDYN or DSOUT subroutines, according to what variables are being defined in the code, and what this component is connected to in the project.

## DSDYN

This segment is identical to the *Fortran* segment, except that all code will be forced into the DSDYN section of the EMTDC System Dynamics.

## DSOUT

This segment is identical to the *Fortran* section, except that all code will be forced into the DSOUT section of the EMTDC System Dynamics.

## Checks

The *Checks* segment is used to ensure that data entered into the component input parameters by the user is reasonable. If a specific condition is true, then a given warning or error message can be made to appear in the Build and Runtime Message Panes when the case is compiled.

A typical *Checks* segment entry would have the following standard format:

```
<MESSAGE TYPE> <Message> : <Expression>
```

- <MESSAGE TYPE> is the first word in the message line and must be chosen as either WARNING or ERROR depending on the problem severity. If chosen as a warning, the message will appear as a warning in the Build and Runtime Message Panes. If specified as an error, then the simulation will be stopped until the condition is rectified.
- <Message> is the diagnostic message that is printed if an error or warning is produced. This message should be descriptive enough for another user to determine what the problem is, and where it is coming from.
- <Expression> is the test used to determine if an error or warning message should be generated. This expression is based on negative logic, which means the message is generated only if the expression evaluates to false.

---

### EXAMPLE 9-7:

A designer's component contains an input for frequency with symbol name *F*. The designer wants to ensure that only a value greater than zero is entered.

The *Checks* segment should then contain the following:

```
ERROR System frequency must be greater than zero : F > 0.0
```

Both Logical and Arithmetic Operators may be used in the expression. For example:

```
WARNING R1 / R2 must be greater than 100 : R1 > 100*R2
```

Remember that negative logic is used in the above examples!

## Help

The *Help* segment is useful when the user wants to provide an external HTML based help file for a particular user-defined component. For example if the user has created an HTML document entitled *help.html* for example, then all that is required that the name and extension be added to the *Help* segment as shown below:

```
help.html
```

When a user right clicks on the component instance and selects **Help** from the pop-up menu, the external document indicated will open.

Please note the following important facts:

- **Help File Placement:** In order to reduce confusion when linking components to help documents, the path to the library project (\*.pslx) file, in which the definition is located, will be appended to the filename. Therefore, the external help file must always be located in the same directory as the parent library project.
- **HTML Browser:** You must choose an HTML browser program for viewing custom help files. This is accomplished through the **HTML Browser** workspace setting. See Application Options Dependencies category description for more.

## Comments

Add your comments / reminders / notes about the design of this component here. Do not use this segment as a means of providing help to users. This segment is ignored by the application and is accessible only when editing the definition.

## FlyBy

The *FlyBy* segment provides an avenue for the designer to apply fly-by (or pop-up) help to a component. Fly-by windows can help to provide users with a quick description of the model itself, or even individual port connections on the component.

The *FlyBy* segment syntax is straightforward:

```
<Descriptive text for the component>
```

```
: <Connection_Symbol_Name>
```

```
<Descriptive text for the connection>
```

## EXAMPLE 9-8:

A designer wants to add fly-by help to the component below:



In addition to a general description of the component, the designer wants to provide a fly-by for each of the input port connections *A* and *B*.

The *FlyBy* segment should then contain something similar to the following:

```
This is my User Component
```

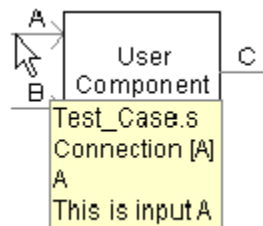
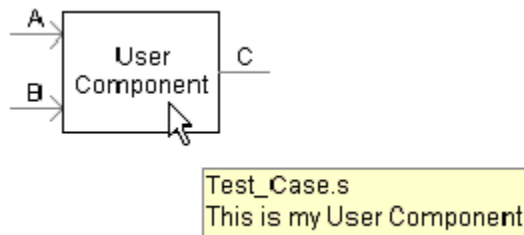
```
:A
```

```
This is input A
```

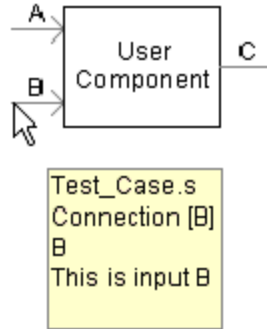
```
:B
```

```
This is input B
```

If the mouse pointer is now moved over the component while on the Circuit canvas, the following fly-by windows will appear:







## Transformers

The *Transformers* segment is used both to define data for any existing mutual impedance matrices, as well as to provide dimensioning information to EMTDC regarding transformers and windings. A single component may contain multiple mutual impedance matrices.

The user must include transformer information in the following parts:

- **Number of transformers (matrices):** This is entered via the use of a #TRANSFORMERS directive. See Definition Script for more details.
- **Maximum Number of Windings:** This is entered via the use of a #WINDINGS directive. See Definition Script for more details.
- **Number of Windings:** This number will define the dimension of each matrix. If this number is given as a positive value, then PSCAD will assume that the matrix is in non-inverted format, and therefore the matrix data must be entered directly. If the number is negative, it signifies that this transformer is 'ideal' (i.e. contains only inductance) and PSCAD will assume that the matrix is already inverted. The matrix data must therefore be entered accordingly.
- **Matrix Data:** The matrix data is entered here, depending on whether the transformer is 'ideal' or not.

According to the above description, a basic *Transformers* segment would appear as follows (for a classical type transformer) in general format:

```
#TRANSFORMERS <Number_of_Transformers>

#WINDINGS <Number_of_Windings>

<Prefix><Number_of_Windings> /

<Node_1> <Node_2> <R_11> <L_11> /

<Node_2> <Node_3> <R_12> <L_12> <R_22> <L_22> /

...
```

**NOTE:** The *Transformers* segment format is different when defining a UMEC type transformer. For more information on the UMEC format, see the UMEC transformer component definitions in the master library, or contact the PSCAD Support Desk (support@pscad.com).

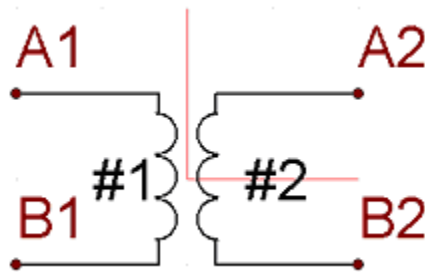
<Prefix> is not mandatory, but is required to indicate an 'ideal' transformer matrix (i.e. already inverted). Also, resistance values (i.e. <R\_xx>) are not required if the transformer is 'ideal'.

Note that only the diagonal and the data below the diagonal need to be entered. The matrix is assumed to be symmetrical.

---

EXAMPLE 9-9:

A user wants to create a non-ideal, single-phase, two-winding transformer. The component Graphics, Parameters and other Script section has already been included. The component electrical port connections have been set-up as shown below in the Graphic section.



The winding self and mutual resistance and inductance parameters have been defined and given symbol names  $R11$ ,  $L11$ ,  $R12$ ,  $L12$ ,  $R22$  and  $L22$ .

The *Transformers* segment should then appear similar to the following:

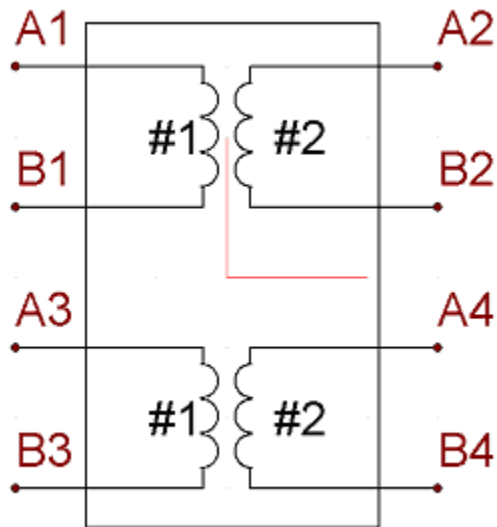
```
#TRANSFORMERS 1
#WINDINGS 2
!
2 /
$A1 $B1    $R11 $L11 /
$A2 $B2    $R12 $L12  $R22 $L22 /
!
```

---

When designing a component with more than one transformer, you can use a special syntax (called 888 syntax) so that, if all the matrix elements are identical, you can avoid entering the repetitive data. This is especially useful if the mutual impedance matrices are very large and numerous.

## EXAMPLE 9-10:

In the example above, the user wants to modify the component, so that it contains two identical single-phase transformers within it. The component electrical port connections have been set-up as shown below in the Graphic section.



The *Transformers* segment should then appear similar to the following:

```
#TRANSFORMERS 2
#WINDINGS 2
!
2 /
$A1 $B1   $R11 $L11 /
$A2 $B2   $R12 $L12  $R22 $L22 /
!
888 /
!
$A3 $B3 /
$A4 $B4 /
```

## Model-Data

The *Model-Data* segment is used as a way to bring input data into a user-defined subroutine, without having to declare an argument for each bit of data. There is no specific format for text in this segment, as this format depends on READ statements within the user code.

When the project is compiled, PSCAD will include all text in this segment within the corresponding module Data File (\*.dta), under the headings DATADSD or DATADSO. If the subroutine is called from DSDYN, the *Model-Data* contents will appear in the DATADSD section; and DATADSO otherwise.

In order to read this information, a standard Fortran READ statement must be added to the user-defined subroutine as follows:

```
!
!  Add include file to define IUNIT
!
      INCLUDE 'fnames.h'
!
!  ...
!
      READ(IUNIT,*) ...
!
```

## Matrix-Fill

When a project is initially compiled, PSCAD will construct a temporary logical matrix for the purpose of indicating how the electrical system is connected (i.e. how nodes and branches are put together). The *Optimize Node Ordering* algorithm in PSCAD then uses this information to optimize node placement in the actual system conductance matrix. However, only electrical nodes and branches, defined within the *Branch* segments of each component present in the system, are considered.

Any internal node connections defined in segments other than *Branch* will not be included in this matrix. Therefore, the logical matrix may have missing information, thereby reducing the effectiveness of the *Optimize Node Ordering* algorithm. The *Matrix-Fill* segment can be used to 'help' this algorithm by providing the omitted connection information for any internal component nodes.

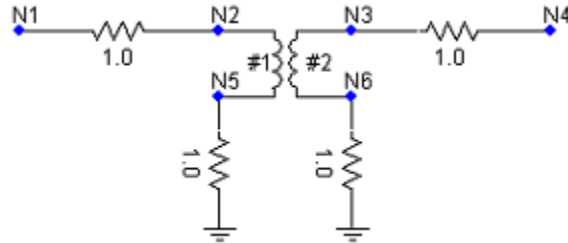
The general format for a *Matrix-Fill* Segment statement would appear as follows:

```
<Node_1> <Node_2> <Node_3> <Node_4> ...
```

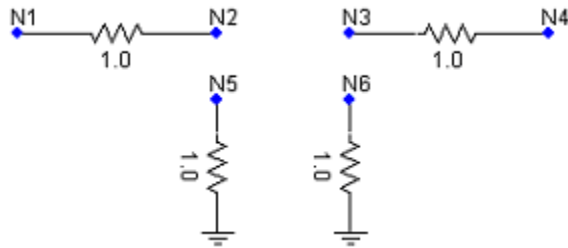
Where <Node\_#> is the Symbol name of the port connection involved. All port connections in a single statement are assumed as connected to each other.

EXAMPLE 9-11:

A user creates a single-phase electrical circuit in PSCAD, which includes a classical transformer component connected as follows:



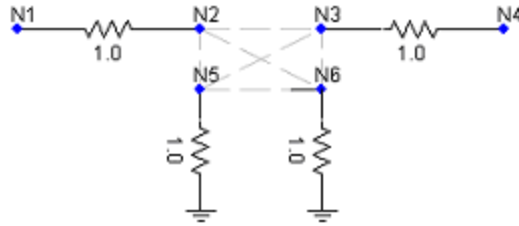
As can be seen above, there are 6 electrical nodes in this system, and therefore the system matrix will have dimension 6 x 6. Connections internal to the transformer component cannot be determined initially by PSCAD, and so when a logical matrix for the *Optimize Node Ordering* algorithm is created, the transformer appears as a 'black box' as shown below:



The logical matrix for the *Optimize Node Ordering* algorithm would appear as follows, where X indicates a known connection:

$$\begin{bmatrix} X & X & 0 & 0 & 0 & 0 \\ X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & 0 & 0 & X \end{bmatrix}$$

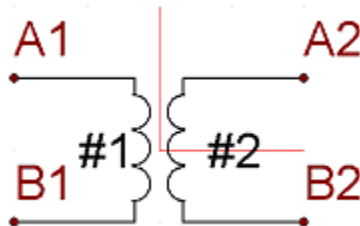
The matrix above does not contain all of the connection information required however. We know through the *Transformers* segment in the transformer component, that nodes N2, N3, N5 and N6 are all part of a mutual impedance matrix and are hence all connected together as shown below:



By adding in *Matrix-Fill* information, we can tell PSCAD to add connection information to the matrix so it will appear as follows, where '+' indicates a connection added by *Matrix-Fill*:

$$\begin{bmatrix} X & X & 0 & 0 & 0 & 0 \\ X & X & + & 0 & + & + \\ 0 & + & X & X & + & + \\ 0 & 0 & X & X & 0 & 0 \\ 0 & + & + & 0 & X & + \\ 0 & + & + & 0 & + & X \end{bmatrix}$$

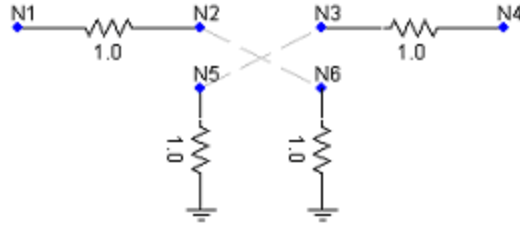
If the four electrical port connections on the transformer were given as follows in the Graphic section of the definition,



then the *Matrix-Fill* segment should appear as:

```
$A1 $B1 $A2 $B2
```

Note that in the above example, it assumed that all connections between the four nodes involved are possible. If however, only connections as shown below are needed,



then the *Matrix-Fill* segment should appear as:

```
$A1 $B2
```

```
$A2 $B1
```

## T-Lines

The T-Lines segment is used to specify any electrical port connections that are to be used as sending or receiving end port connections for a transmission line or cable. The T-Lines segment is rarely needed in component design. The general format of statements in this segment is given below:

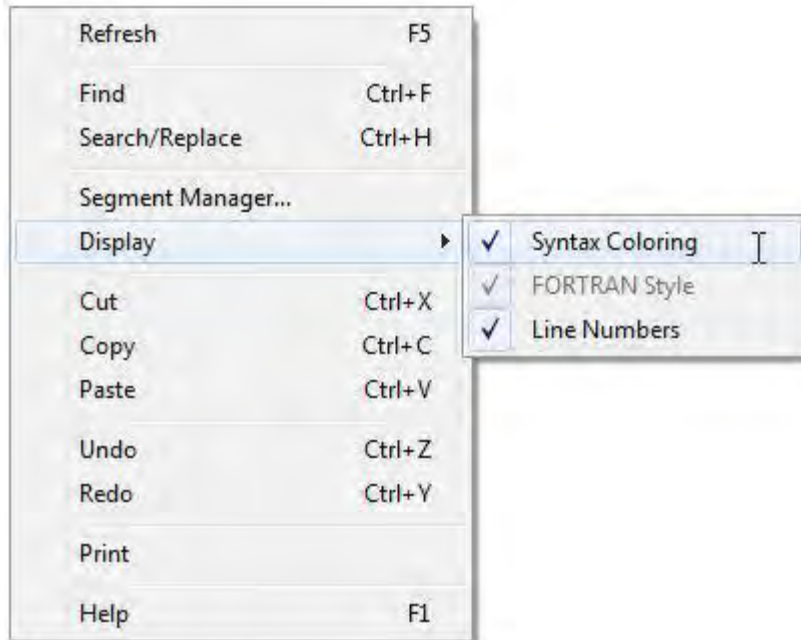
```
<Subsystem_#> <Cond_1> <Cond_2> <Cond_3> ...
```

<Cond\_#> signifies that any port connection Symbol name that is entered in this position will be the sending or receiving end port connection for that particular conductor number in the transmission line or cable properties.

This Segment is used only within the Transmission Line and Cable Interface components in the master library.

## Segment Display Settings

There are a few display options that can be adjusted when viewing script segments. These may all be found under **Display** in the script section canvas pop-up menu.



- **Syntax Coloring:** Select this option to enable code coloring. When de-selected, all Script code will appear in black.
- **FORTRAN Style:** This option provides graphical indication to illustrate the Fortran 6th column. It appears as a green, vertical bar when viewing Fortran files, or when in the Fortran, DSDYN or DSOUT segments.
- **Line Numbers:** Select this option to enable display of line numbers when viewing code in the Script section or when viewing any text file in PSCAD.

## Undo and Redo

Undo/redo is presently not supported in the Script section.

## Internal Output Variables

Outputting EMTDC Measured Voltages and Currents

An *Internal Output Variable* is internal component data, which has been made available as a signal to be plotted or monitored. Any internal quantities that need to be output must either be passed through an output parameter, or be defined within the component Fortran segment using an #OUTPUT directive.

Internal data used as output can either come from an internal component variable, from an EMTDC storage array, or as a measured branch current or voltage from EMTDC.

## Outputting EMTDC Measured Voltages and Currents

There are a couple of substitution operators that can be used with #OUTPUT directives to extract measured quantities directly from electrical system nodes and branches. These are described below.

### CBR

*CBR* is a special substitution operator that will output the current measured in a specified branch. A special syntax is used for this – see the example below.



## EXAMPLE 9-12:

A user wants to measure the current in a branch called *BRN*. A text field has been added to the component Parameters section with Symbol name *Ia*.

The following should then appear in the Fortran segment:

```
#OUTPUT REAL Ia {$CBR:BRN}
```

## VDC

*VDC* is a special substitution operator that will take the voltage difference between two different electrical nodes. A special syntax is used for this – see the example below.

## EXAMPLE 9-13:

A user wants to measure the voltage difference between two port connections defined within a component definition, called *N1* and *N2*. A text field has been added to the component *Parameters* section with symbol name *Vdiff*.

The following should then appear in the Fortran segment:

```
#OUTPUT REAL Vdiff {$VDC:N1:N2}
```

**NOTE:** It is important to ensure that neither *N1* nor *N2* are type ground nodes. The reason being is that ground nodes are given a *0* node number, which is an invalid entry to the *VDC(NA,SS)* array.

### Adding a Reference to a Source File

It is possible to call a subroutine or function defined in an external source file (i.e. \*.f, \*.f90, \*.for or \*.c file) from within the Fortran, DSDYN or DSOUT segments of a definition. In order to link to this code when the project is compiled and built, you must ensure a reference to this file is provided. This can be done in one of two ways:

1. Add a reference from the Project Settings dialog: Right-click over a blank area the Schematic canvas and select **Project Settings...** Under the Fortran tab, add a reference to the file in the **Additional Source files (\*.f, \*.for, \*.f90, \*.c, \*.cpp)** input field.
2. Add a File Reference component: Right-click over a blank area of the Schematic canvas and select **Add Component | File Reference**. Open the *File Reference* component and select the source file. Note that the **Additional Source files (\*.f, \*.for, \*.f90, \*.c, \*.cpp)** input field should be used instead whenever possible.

# Interfacing to C Language Source

It is possible to interface to C language source, so long as the source is defined as a procedure (or procedures) within an external file – either a C source file (\*.c) or precompiled object (\*.obj) file. C source cannot be inserted directly into the Fortran, DSDYN or DSOUT segments, as these are reserved for Fortran only.

When using the GFortran compiler, C procedures may be called directly from within the Fortran, DSDYN, or DSOUT segments of a component definition, although this is not recommended (see *Calling C Procedures Directly (GFortran Only)* below for reasons why). Preferably, users should instead call a special Fortran interface routine (sometimes referred to as a 'wrapper'), which will in turn call the C procedure.

Prior to compilation of the project, ensure that any source (\*.c) files are referenced in the Additional Source Files project setting (under the Fortran tab). Alternatively, object files (\*.obj) or a library file (\*.lib) should be referenced in the Additional Library (\*.lib) and Object (\*.obj) Files project setting (under the *Link* tab).

For more details on interfacing to C source code, see the example projects in the ...\Examples\CInterface directory within your installation directory.

## Calling C Procedures Directly (GFortran Only)

As mentioned above, in many instances it is perfectly all right to call C procedures directly from the Fortran, DSDYN or DSOUT segments. This, for the most part, is true for simple procedures that do not require advanced EMTDC intrinsic variables.

When compiling the Fortran code for a module, PSCAD includes only a standard set of EMTDC header files. A snippet of a typical Fortran file is shown below, which illustrates the header files that are included:

```
!-----
! Standard includes
!-----

    INCLUDE 'nd.h'

    INCLUDE 'emtconst.h'

    INCLUDE 'emtstor.h'

    INCLUDE 's0.h'

    INCLUDE 's1.h'

    INCLUDE 's2.h'

    INCLUDE 's4.h'

    INCLUDE 'branches.h'

    INCLUDE 'pscadv3.h'

    INCLUDE 'fnames.h'

    INCLUDE 'radiolinks.h'

    INCLUDE 'matlab.h'

    INCLUDE 'rtconfig.h'
```

If your C procedure requires variables from EMTDC header files other than those shown above, then you cannot call the procedure directly from the Fortran, DSDYN or DSOUT segments. You must create a Fortran interface routine that includes the required header files before the procedure is called (next section).

See Include Files for descriptions of the available header files.

---

**EXAMPLE 9-14:**

A user has written the following simple subroutine in C, and wants to call this source code directly from a user-defined component:

```
/* User C Source Code */  
void test_csub_(int* arg, int* res)  
{  
    *res = 6*(*arg);  
}
```

A standard call statement can be made from within the Fortran, DSDYN or DSOUT segments of the component definition.

```
CALL TEST_CSUB($IN,$OUT)
```

Where *\$IN* and *\$OUT* are input and output port connections on the component respectively.

---

**EXAMPLE 9-15:**

A user has written the following simple function in C, and wants to use this source code in a user-defined component:

```
/* User C Source Code */  
int test_cfun_(int *arg)  
{  
    return 2*(*arg);  
}
```

The function must be declared before it is used within the component. Then, the function is used as it normally would in Fortran in the Fortran, DSDYN or DSOUT segments of the component definition.

```
#FUNCTION INTEGER TEST_CFUN
    $OUT = TEST_CFUN($IN)
```

Where *\$IN* and *\$OUT* are input and output port connections on the component respectively.

---

## Intel Visual Fortran

If you are using one of the supported commercial Fortran compilers, you must call your C procedures through an interface subroutine. These compilers require that you construct an *INTERFACE* statement to describe your C procedure. See the following section for details.

## Calling C Procedures through a Fortran Interface Routine

It is good coding practice that an auxiliary Fortran subroutine be provided for the purpose of interfacing to C functions and procedures. It is this Fortran subroutine that is called from the Fortran, DSDYN or DSOUT segments of your user-defined component. Constructing an interface will ensure a reasonable level of portability between the Fortran compilers available with PSCAD.

The following examples illustrate how to call a C subroutine when using the Fortran 90 compilers.

---

EXAMPLE 9-16:

A user has written the following simple subroutine in C, placed it into a C file (\*.c).

```
/* User C Source Code */
void test_csub(int* arg, int* res)
{
    *res = 6*(*arg);
}
```

The user then defines an interface subroutine in a Fortran (\*.f) source file as follows:

```

SUBROUTINE AUX_CSUB(in, out)
  INTEGER in, out
!
! Fortran 90 interface to a C procedure
!
  INTERFACE
    SUBROUTINE TEST_CSUB (in, out)
      !DEC$ ATTRIBUTES C :: TEST_CSUB
      !DEC$ ATTRIBUTES REFERENCE :: in
      !DEC$ ATTRIBUTES REFERENCE :: out
!
! in, out are passed by REFERENCE
!
      INTEGER in, out
    END SUBROUTINE
  END INTERFACE
!
! Call the C procedure
!
  CALL TEST_CSUB(in, out)
END

```

A standard call statement is then added to the Fortran segment of the user component definition:

```
CALL AUX_CSUB($IN,$OUT)
```

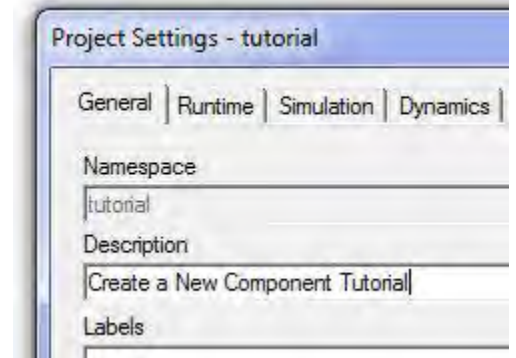
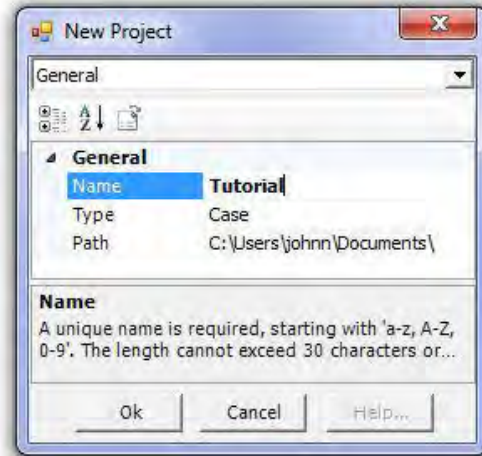
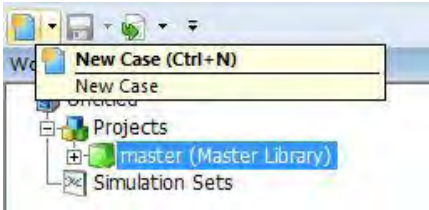
Where *\$IN* and *\$OUT* are input and output port connections on the component respectively.

---

## Tutorial: Creating a New Component

# Creating a New Component

Open PSCAD, create a new case project, save it under a unique name, edit the project settings and add a description.



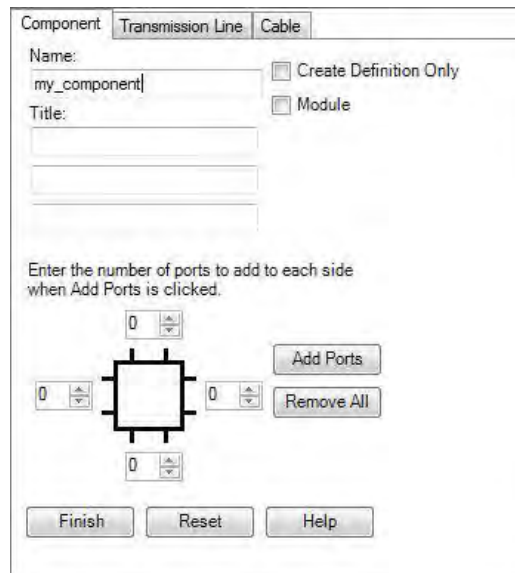
Select the 'New Case' Button

Give the Project a Unique Name (ex. 'Tutorial.pscx')

Edit the Description in the Project Settings Dialog

At this point you may want to save the workspace, although it is not necessary for this tutorial.

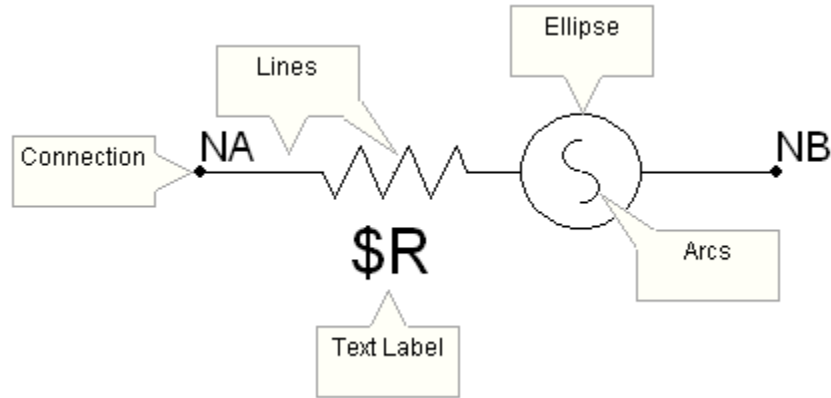
Create a new component using the *Component Wizard*.



Invoke the Component Wizard

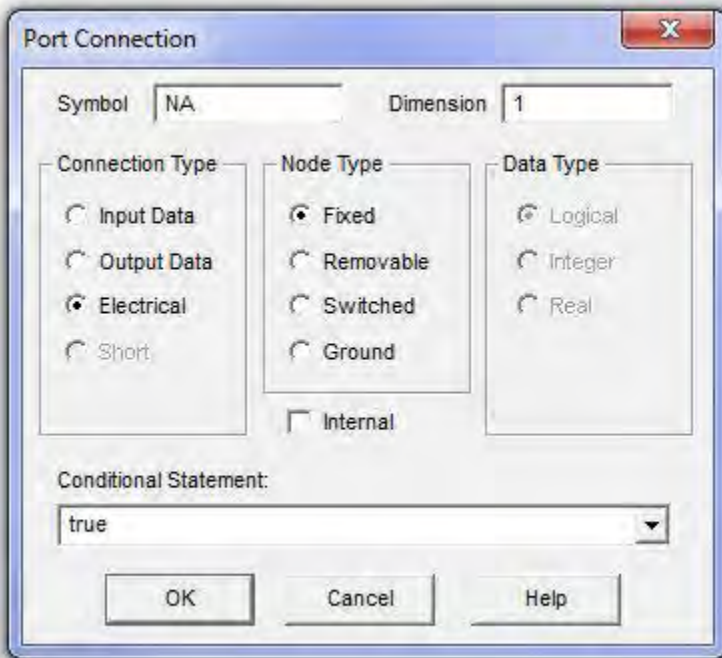
## Adding Graphics

Edit the component definition and in the Graphic section, draw the following diagram (minus the sticky notes!); where  $SR$  is a text label, and  $NA$  and  $NB$  are port connections.

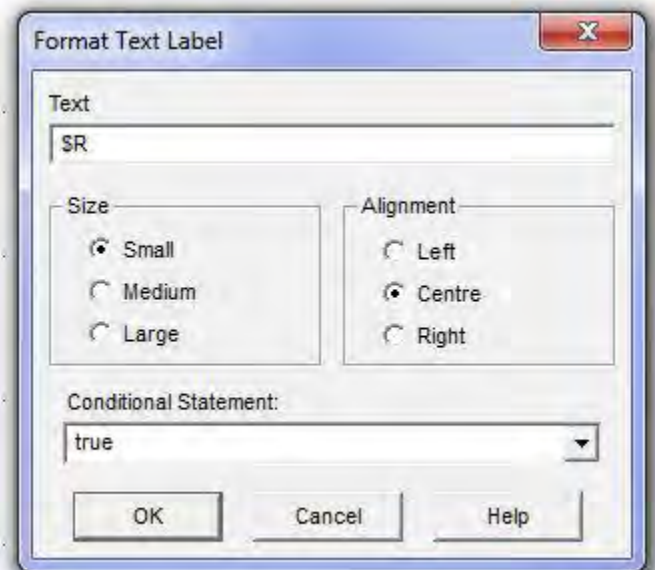


**NOTE:** Make sure the graphics are of a reasonable size in relation to other components in the master library.

Configure the properties of the port connections so that they are fixed, electrical nodes. Also, configure the text label to display  $SR$ :



Connection Property Settings ( $NA$ )



Text Label Property Settings ( $SR$ )

Save the Project.

## Adding a User Interface

Edit the component definition and go to the Parameters section. Notice that a default category called *Configuration*, including a text field called *Name*, has been created for you.

Add value fields to the category with the following **Descriptions**: *Voltage Magnitude (RMS)*, *Frequency*, *Ramp Up Time* and *Resistance*. These are to have **Default Units** set to *kV*, *Hz*, *s* and *ohm* respectively. Give each field a short, but descriptive symbol name (say *Vrms*, *f*, *tr* and *R* respectively) and provide a reasonable **Default value** (see the image below), as well as **Minimum** and **Maximum** limits.

The **Data Type** of each of these parameters should be declared as **Constant**. This is to ensure that the component is *Runtime Configurable* – in other words it can accept unique, constant input when used within a module that is multiply-instanced. This may become clearer as we add script code to the component in the section Adding Code to Define the Source in this tutorial.

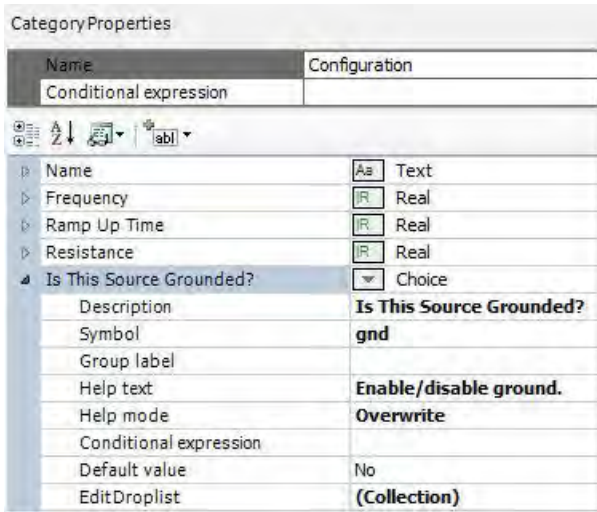
See Runtime Configuration and Multiple Instance Modules for more details on using *Constant*-type parameters.

Category Properties	
Name	Configuration
Conditional expression	
<div style="display: flex; align-items: center; gap: 5px;"> <span>⊕</span> <span>⊖</span> <span>A</span> <span>Z</span> <span>↓</span> <span>↻</span> <span>labl</span> </div>	
▶ Name	<input type="text" value="Aa"/> Text
▶ Frequency	<input type="text" value="R"/> Real
Description	<b>Frequency</b>
Symbol	<b>f</b>
Group label	
Default units	<b>Hz</b>
Minimum value	-1e+308
Maximum value	+1e+308
Dimension	1
Data type	<b>Constant</b>
Intent	Input
Help text	<b>The source frequency.</b>
Help mode	<b>Overwrite</b>
Conditional expression	
Default value	<b>60.0</b>
▶ Ramp Up Time	<input type="text" value="R"/> Real
▶ Resistance	<input type="text" value="R"/> Real
Description	<b>Resistance</b>
Symbol	<b>R</b>
Group label	
Default units	<b>ohm</b>
Minimum value	-1e+308
Maximum value	+1e+308
Dimension	1
Data type	<b>Constant</b>
Intent	Input
Help text	<b>Source resistance.</b>
Help mode	<b>Overwrite</b>
Conditional expression	
Default value	<b>1.0</b>

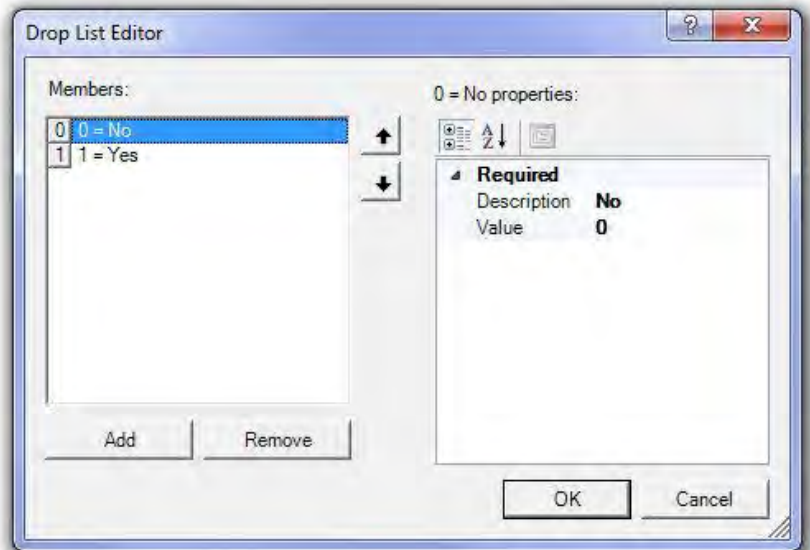
Value Field Properties (ex. *Frequency* & *Resistance*)



Now add a choice list to the category with the **Description** *Is This Source Grounded?*. Give it an appropriate **Symbol** name (say, *gnd*) and then add two choices: 1=Yes and 0=No.



Choice List Properties (*Is This Source Grounded?*)



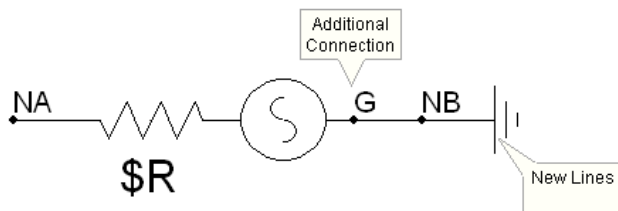
Drop List Editor

Save the Project.

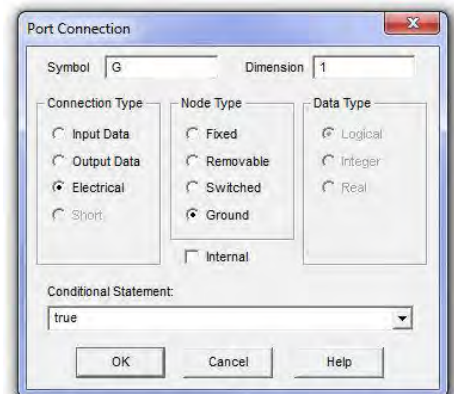
## Adding Conditional Statements to the Graphic Objects

In order to make our component more convenient to use, we should give the user the option to ground the source at one end. This will be the primary purpose of the *Is This Source Grounded?* choice list. Using the state of this field (i.e. Yes or No), we can alter both the appearance and the electrical properties of the component.

Edit the component definition and in the Graphic section, modify the component graphics as shown in the diagram below, where G is a port connection configured as an electrical ground node.

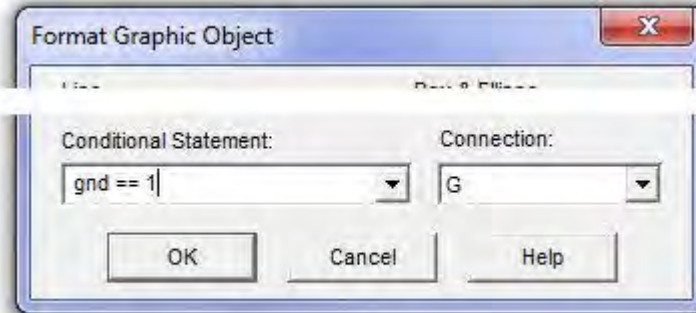


Modified Component Graphics



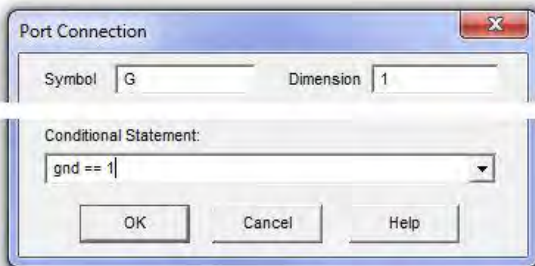
Connection Property Settings (G)

In the **Conditional Statement** field of all the newly added line objects, add conditions so that they will become visible/invisible according to the setting specified in the *Is This Source Grounded?* choice list. That is, when *Is This Source Grounded?* is set to Yes, the ground symbol graphics above should be visible.

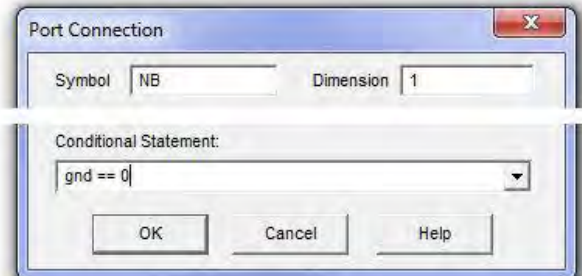


Editing Conditional Statement in Line Objects

In the **Conditional Statement** field of port connections *G* and *NB*, add conditions so that they become enabled or disabled according to the setting specified in the *Is This Source Grounded?* choice list. That is, when *Is This Source Grounded?* is set to Yes, port connection *G* should be enabled and port connection *NB* should be disabled (and vice-versa):



Editing Conditional Statement in Port Connection *G*



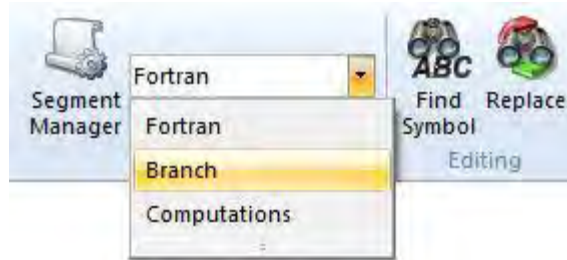
Editing Conditional Statement in Port Connection *NB*

You may be wondering at this point about the placement of the port connections *G* and *NB* in the Graphic section diagram above: When placing your port connections within your component graphic, it is important to consider where the most intuitive position for the port connection is. That is, it would be best to place *NB* at the end of the line object, as this would be an obvious connection point.

Save the Project.

## Defining the Electrical Branch

Edit the component definition and in the Script section, navigate to the Branch segment.



Add statements that define the electrical properties of the component for use by EMTDC.

This branch is to be purely resistive, just to keep things simple. Due to the fact however that the *Resistance* parameter is of Constant-type, its actual value will be initialized when coding the component (see the next section). For now, we simply need to add a non-zero value (i.e. *1.0*) in the branch statement to indicate the presence of a resistor. If it was decided earlier to declare the *Resistance* parameter as Literal-type, then its value could be substituted directly into these branch statements (i.e. instead of *1.0*, it would be *\$R*).

```
#IF gnd == 1
    BRN = $NA $G SOURCE 1.0 0.0 0.0
#ELSE
    BRN = $NA $NB SOURCE 1.0 0.0 0.0
#ENDIF
```

Conditional statements (*#IF*, *#ELSE*, etc.) are added as well, so that the branch will be defined between *NA* and *NB* or *NA* and *G*, depending on the condition of the *Is this source grounded?* choice list. For example, if *Is this source grounded?* is selected as *Yes* (i.e. *gnd = 1*), then the branch will be defined between nodes *NA* and *G*.

In addition to indicating that a resistor defines part of the branch between *NA* and *NB* or *NA* and *G*, we have also indicated that an ideal voltage source is present. This is accomplished through the *SOURCE* statement above. We have also given the branch a variable name *BRN*. This enables EMTDC to map the location of this branch in the electric network, so that the source can be controlled (the usage of *BRN* will become clear in the next section).

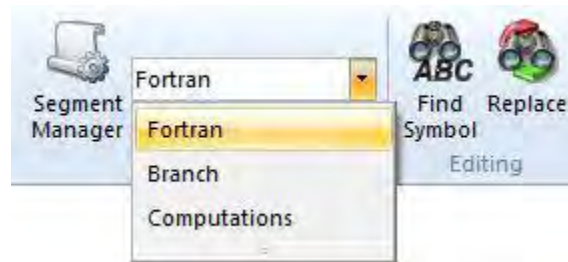
For more information on branch naming and the *SOURCE* syntax, see Branch segment.

## Adding Code to Define the Source

Instead of writing our own code to define the voltage source, we can use EMTDC internal subroutines to do it for us. We simply need to ensure that all of the subroutine arguments are defined in the component before we call it. The subroutines we will use are as follows:

- **E\_BRANCH\_CFG**: This routine is used to initialize the branch properties that we have already outlined in the previous section.
- **E\_1PVSRC\_CFG**: This is the runtime configuration routine for one of the EMTDC single-phase voltage sources. It is used in the Single-Phase Voltage Source Model 2 component in the master library.
- **EMTDC\_1PVSRC**: This is the actual EMTDC single-phase voltage source routine. It is used in the Single-Phase Voltage Source Model 2 component in the master library.

Edit the component definition and in the Script section, navigate to the Fortran segment.



Add the following script to the segment (you may copy directly from this document):

```
#BEGIN

    CALL E_BRANCH_CFG($BRN,$SS,1,0,0,$R,0.0,0.0)

    CALL E_1PVSRG_CFG(1,0,1,$Vrms,$f,0.0,$R,0.0,0.0,0.0,0.0,$tr)

#ENDBEGIN

#STORAGE LOGICAL:1 INTEGER:6 REAL:8 RTCF:4

#LOCAL REAL RVD1_1
#LOCAL REAL RVD1_2
#LOCAL REAL RVD1_3
#LOCAL REAL RVD1_4

! Single Phase AC source: Type: R

    RVD1_1 = RTCF(NRTCF)
    RVD1_2 = RTCF(NRTCF+1)
    RVD1_3 = RTCF(NRTCF+2)
    RVD1_4 = RTCF(NRTCF+3)
    NRTCF = NRTCF + 4

    CALL EMTDC_1PVSRG($SS,$BRN,RVD1_4,.TRUE.,RVD1_1,RVD1_2,RVD1_3)
```

## The #BEGIN Block

The block of script between the #BEGIN and the #ENDBEGIN directives is runtime configurable code that will be inserted into the BEGIN section of the system dynamics in EMTDC. Here, we are calling two EMTDC subroutines, which combined will initialize both the branch parameters and the source control. Descriptions of the argument lists are given as follows:

E\_BRANCH\_CFG (NBR, M, ER, EL, EC, RO, HL, FC)

Argument List:

- **NBR** : Number of the branch being configured.
- **M**: Subsystem number where the branch resides.
- **ER**: Include/exclude resistance (1 or 0).
- **EL**: Include/exclude inductance (1 or 0).
- **EC**: Include/exclude inductance (1 or 0).
- **R0**: Resistance value [ $\Omega$ ]
- **HL**: Inductance value [H]
- **FC**: Capacitance value [ $\mu$ F]

`E_1PVSRC_CFG (AC, SPEC, TYP, VM, F, PH, R, L, RP_C, P, Q, TC)`

#### Argument List:

- **AC** : AC or DC source (1 or 0 respectively).
- **SPEC**: Internal values or at source terminal (0 or 1 respectively).
- **TYP**: Impedance type ( $R = 1$ ).
- **VM**: RMS voltage [kV].
- **F**: Frequency [Hz].
- **PH**: Phase angle [deg]
- **R**: Resistance value [ $\Omega$ ]
- **L**: Inductance value [H]
- **RP\_C**: Capacitance value [ $\mu$ F]
- **P**: Active Power output
- **Q**: Reactive power output
- **TC**: Ramp-up time [s].

## Storage and Dynamics Script

The storage allocation (#STORAGE) is necessary as it provides memory usage information for dynamic memory dimensioning in EMTDC. The LOGICAL, INTEGER and REAL allocations are the storage usage for the *EMTDC\_1PVSRC* subroutine. The RTCF allocation is used to dimension the runtime configurable storage.

Directly preceding the call to the *EMTDC\_1PVSRC* subroutine is the extraction of initialized values from the RTCF storage array. The values stored in this array are defined in the BEGIN section at the beginning of the simulation (within the *E\_1PVSRC\_CFG* subroutine) and remain constant. Extracting this data from storage ensures that the initialized values are instance-dependent – that is they are specific to wherever this component code is placed in the system dynamics.

The last bit is the actual call to the subroutine that controls the source throughout the simulation: *EMTDC\_1PVSRC*.

`EMTDC_1PVSRC (M, NBR, RT, AC, VP, W, A)`

Argument List:

- **M** : Subsystem number where the branch resides.
- **NBR**: Number of the branch being configured.
- **RT**: Ramp-up time [s].
- **AC**: True if an AC source.
- **VP**: Peak voltage [kV].
- **W**: Frequency [rad/s].
- **A**: Phase angle [rad].

Save the Project.

## Sanity Checks

It is always a good idea to apply sanity checks to the users input, so as to avoid erroneous results. This is accomplished by using the Checks segment.

Edit the component definition and in the Script section, add a new Checks segment by using the segment manager. Add a statement, which will give an error if the entered source resistance (i.e.  $R$ ) is less than or equal to  $0.0$ . Note that you must use negative logic in this section, and so this statement should appear as follows:

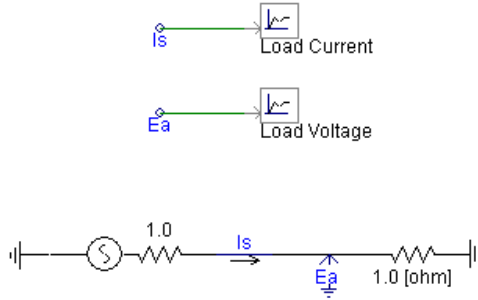
```
ERROR Source resistance must be greater than zero : R > 0.0
```

Save the Project.

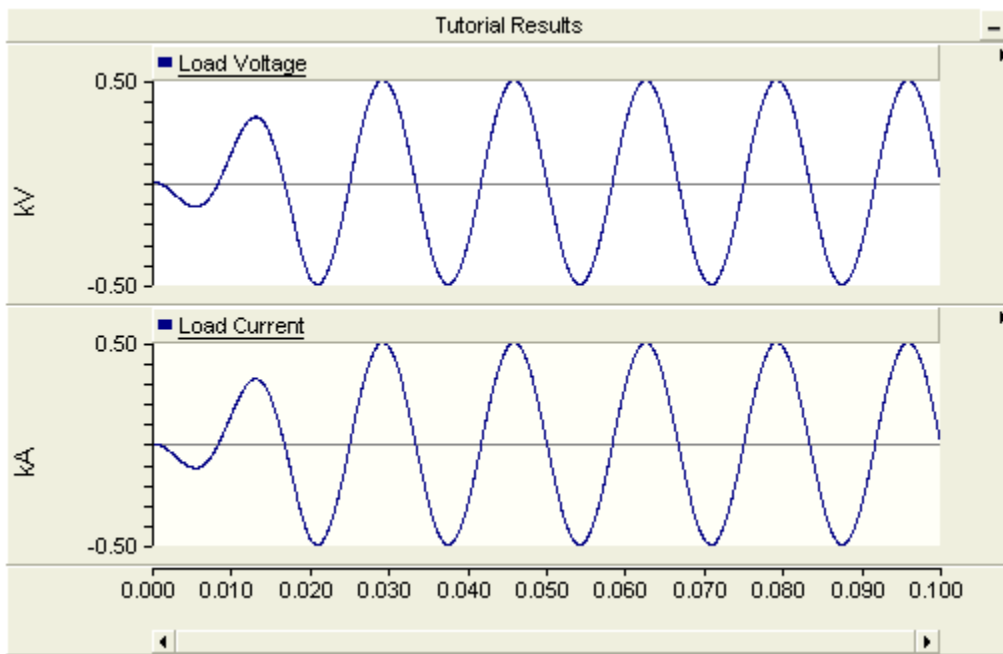
## Testing Your New Component

At this point it would be a good idea to check if your component will actually run, and if it does, whether the results are correct. Below are what the results should be if the source model was connected to the indicated circuit, and if the source is set to:

- Voltage Magnitude (RMS) = 0.7071 kV
- Frequency = 60 Hz
- Ramp-Up Time = 0.02 s
- Source Resistance = 1.0  $\Omega$



The results are as shown below:



Save the Project.

## Adding Internal Output Variables

It is possible to monitor quantities internal to the component, for display on plots and meters. This may be accomplished by defining an internal output variable. In this tutorial, we will set-up such variables to monitor source voltage and branch current.

There are two methods to do this. The older method is to use the #OUTPUT directive. The newer way (first introduced in v4.5) is to use a value field of intent output directly, without the need for an #OUTPUT directive. Both methods are presented below.

Edit the component definition and in the Parameters section, add a text field and a value field: The value field with **Symbol** name *Isource* and **Description** *Source Current* (ensure that **Intent** is set to *Output*); the text field with **Symbol** name *Vsource* and **Description** *Source Terminal Voltage*.

Category Properties	
Name	Configuration
Conditional expression	
▸ Name	<input type="text" value="Aa"/> Text
▸ Frequency	<input type="text" value="IR"/> Real
▸ Ramp Up Time	<input type="text" value="IR"/> Real
▸ Resistance	<input type="text" value="IR"/> Real
▸ Is This Source Grounded?	<input type="text" value="v"/> Choice
▾ Source Current	<input type="text" value="IR"/> Real
Description	<b>Source Current</b>
Symbol	<b>Isource</b>
Group label	
Default units	<b>kA</b>
Minimum value	-1e+308
Maximum value	+1e+308
Dimension	1
Data type	<b>Variable</b>
Intent	<b>Output</b>
Help text	<b>This is the branch current [kA].</b>
Help mode	<b>Overwrite</b>
Conditional expression	
Default value	<b>Ibranch</b>
▾ Source Terminal Voltage	<input type="text" value="Aa"/> Text
Description	<b>Source Terminal Voltage</b>
Symbol	<b>Vsource</b>
Default value	
Group label	
Help text	<b>This is the source terminal voltage [kV].</b>
Help mode	<b>Overwrite</b>
Regular expression	
Error text	
Conditional expression	

Parameter Field Properties (*Isource* & *Vsource*)

Save the Project.

Edit the component definition and navigate to the Fortran segment. Extract the current in this source branch (i.e. from branch *BRN*), using the CBR internal variable as follows:

```
$Isource = CBR($BRN, $SS)
```

The value given by *CBR(\$BRN,\$SS)* will be used to define the value of the output parameter *Isource*. Note that *SS* substitutes the number of the EMTDC subsystem that this branch is part of.



Extract the node voltage from port connection *NA* to *G* or *NB* with an #OUTPUT directive statement and using the VDC internal variable. Remember that we have added conditional statements that allow users to directly connect the source branch to ground. So, the branch voltage measurement will depend on which port connections are enabled. We can then choose a branch statement according to the *Is this source grounded?* choice list.

Your Fortran segment should then look similar to the following when completed:

```
#BEGIN

    CALL E_BRANCH_CFG ($BRN, $SS, 1, 0, 0, $R, 0.0, 0.0)

    CALL E_1PVSRG_CFG (1, 0, 1, $Vrms, $f, 0.0, $R, 0.0, 0.0, 0.0, 0.0, $tr)

#ENDBEGIN

#STORAGE LOGICAL:1 INTEGER:6 REAL:8 RTCF:4

#LOCAL REAL RVD1_1
#LOCAL REAL RVD1_2
#LOCAL REAL RVD1_3
#LOCAL REAL RVD1_4

! Single Phase AC source: Type: R

    RVD1_1 = RTCF (NRTCF)

    RVD1_2 = RTCF (NRTCF+1)

    RVD1_3 = RTCF (NRTCF+2)

    RVD1_4 = RTCF (NRTCF+3)

    NRTCF = NRTCF + 4

    CALL EMTDC_1PVSRG ($SS, $BRN, RVD1_4, .TRUE., RVD1_1, RVD1_2, RVD1_3)

!

    $Isource = CBR ($BRN, $SS)

!

#IF gnd == 1

    #OUTPUT REAL Vsource {$VDC:NA:G}

#ELSE

    #OUTPUT REAL Vsource {$VDC:NA:NB}

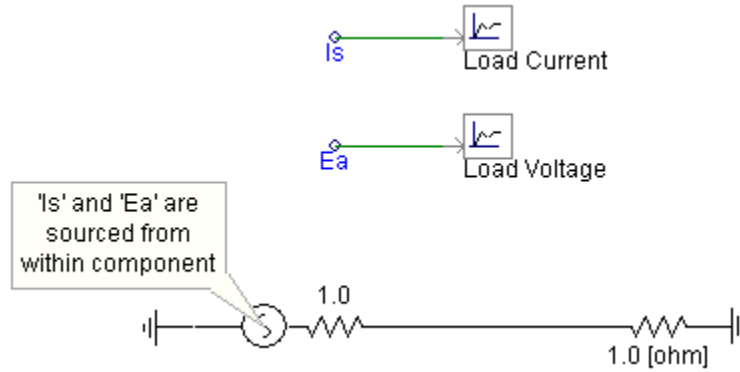
#ENDIF
```

Any text field with a Symbol name identical to those defined above (i.e. *Isource* and *Vsource*), will act as internal output variable ports for the component. Any variable name entered into either the *Source Current* or *Source Terminal Voltage* text fields will create signals which, when connected to an Output Channel, can be plotted.

Save the Project.

## Running With Your Component

Keeping the test circuit used in the Testing Your New Component section, remove the ammeter and voltmeter from the circuit. Replace these two signals by entering  $V_s$  and  $I_s$  into the internal output variable fields that were created.



Run the case again. You should see identical results as before in Testing Your New Component.

## Chapter 9

# Definition Script

The PSCAD definition script language is used extensively in the design of component definitions. It serves mainly as a communication interface between the designer and the application, providing instruction on how to incorporate a component into the greater project. This communication occurs at compile time, when each definition segment is considered sequentially, and its script parsed into information meaningful to the compiler.

Although there are some segments that allow for the direct inclusion of source code (i.e. code that does not require parsing), such as Fortran, DSDYN and DSOUT, exploiting definition script ensures that code source remain current, as well as compiler and language independent. Definition script is given the highest priority during the compilation process; it is considered first before any direct source code, and is consequently used in expression evaluation, substitutions, and for the provision of compiler directives.

This chapter is closely linked with Component Design. It is recommended that you familiarize yourself with both chapters before attempting to design your own component. The following sections describe what scripting tools are available, and provide examples on how each can be used. Component Design contains a tutorial on designing your own component.

## Substitutions

\$ Value Substitution Prefix Operator

% Data Substitution Prefix Operator

{ } Enfolding Braces

! Comment Operator

Substitutions are fundamental script operators that can appear in all parts of a definition. They provide a means to communicate information between definition sections and segments, bring a dynamic aspect to component graphics, and to help in formatting code and comments.

The substitution operators available are as listed below:

- Value Substitution Prefix: \$
  - Contextual Substitution of Text
  - \$#DIM: Substituting Port Dimension
  - \$#Component: Substituting Component Instance ID
- Data Substitution Prefix: %
- Enfolding Operators (Braces): { }
- Comment Operator: !

### \$ Value Substitution Prefix Operator

The \$ Value Substitution Prefix Operator (or simply '\$') is the most important of the substitution operators, and is used quite heavily in component definition design. The \$ operator is normally pre-pended to a variable, where the variable may represent a numerical value (or even other text) that has been defined at some previous point in the evaluation process. Regardless of what the variable represents, its value will be substituted whenever a \$ operator is prefixed to it.

The \$ operator may only precede variable names defined as one of the following:

- A port connection *Symbol* name

- An input parameter field, text field, or choice list *Symbol* name
- Any variable defined in the Computations segment
- A key name

There are numerous ways in which the \$ operator may be utilized. One method is the substitution of literal values, in place of a component variable name: The literal value of component variable – be it a port connection, input parameter or Computations variable – will be directly substituted upon compilation. Literal value substitution may also be used to format comments.

---

EXAMPLE 10-1:

The literal value entered into an input parameter field can be substituted directly into source code by prefixing \$ to the Symbol name of the field.

Frequency	abl	Real
Description	Frequency	
Symbol	freq	Symbol Name
Default units	Hz	
Minimum value	0.0001	
Maximum value		
Data type	Constant	
Default value	60.0	
Conditional statement		

Input Field Properties Dialog

```

1 !
2
3 !

```

OUT = 2.0\*PI\*\$freq

Value of 'freq' will be substituted

Substituting Field Value into a Fortran Statement

```

!
REAL    RT_1                ! EMTDC variable to represent 'freq'
REAL    OUT

RT_1 = STOF(ISTOF + 2)    ! EMTDC extracts parameter value
                        ! from stored data array

OUT = 2.0*PI*RT_1

!

```

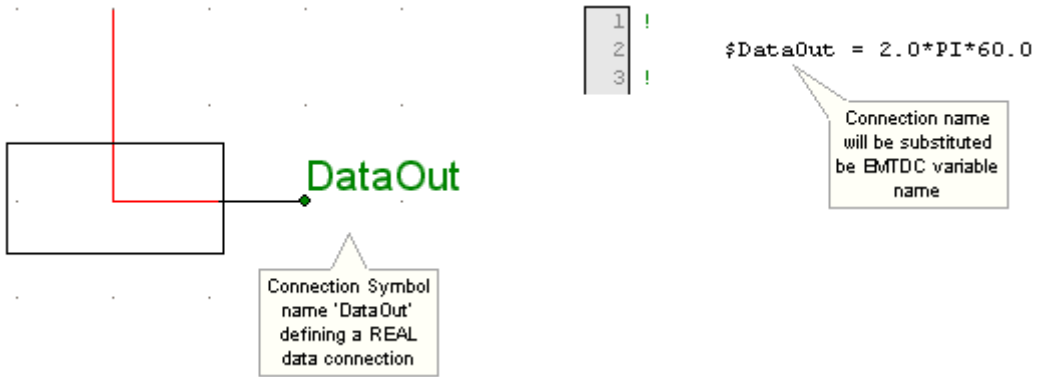
Actual Fortran Code Following Compile

**NOTE:** The input parameter value substituted for *freq* is the resulting value *following* unit processing. See the section entitled Unit System in *Operations and Feature Overview* for more details on units.

---

EXAMPLE 10-2:

The value of an output port connection entitled *DataOut* is defined by a Fortran statement. Upon compile of the project, the internal compiler will declare a variable to represent the *DataOut* port connection in the associated Fortran file: The \$ operator will ensure that this new variable is substituted in the proper places.



Component Graphic Containing a Connection

Substituting a Port Connection Name into a Fortran Statement

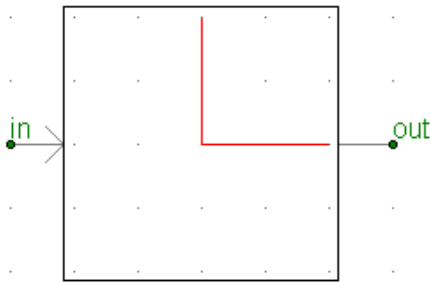
```

!
      REAL    RT_1      ! EMTDC variable to represent 'DataOut'
!
      RT_1 = 2.0*PI*60.0
!
    
```

Actual Fortran Code Following Compile

EXAMPLE 10-3:

A user component utilizes a function, which requires an angle input argument in degrees to convert to radians. The data input in this case is a port connection, defined in the Graphic section of the component definition, and has a Symbol name *in*. The function output *out* is defined as an output port connection.



Component Graphic Containing Port Connections

```

1 #FUNCTION REAL PH_CON Phase Angle Converter
2 !
3     $out = PH_CON($in)
4 !
5 ! When this Script is compiled by PSCAD, $in will be substituted
6 ! by whatever EMTDC decides to name the data node to which 'in' is
7 ! connected (ex. RT_1). $out will substituted by whatever EMTDC
8 ! decides to name the data node to which 'out' is connected
9 ! (ex. RT_2).
10 !

```

Fortran Segment Script

```

!
REAL    PH_CON          ! Phase Angle Converter
REAL    RT_1, RT_2
!
RT_2 = STOF(ISTOF + 2) ! EMTDC extracts port connection value
                        ! from stored data array
RT_1 = PH_CON(RT_2)
!

```

Actual Fortran Code Following Compile

**NOTE:** See the #FUNCTION Directive for more details on declaring Fortran functions.

---



---

#### EXAMPLE 10-4:

A user-defined component definition called *my\_new\_comp* has been programmed to display the contents of certain key input parameter fields, along with the definition name, within comments provided in the component Fortran segment, namely: *Frequency* with Symbol name *freq*, value *60.0* and units in *Hz*, and *Voltage* with Symbol name *volts*, value *120.0* and units in *kV*. Note that both *freq* and *volts* are local variables and so the local context short form is used.

Frequency		Real
Description	Frequency	
Symbol	freq	
Default units	Hz	
Minimum value	0.01	
Maximum value	1e+008	
Data type	Literal	
Default value	60.0	
Conditional statement		

Input Field Property Settings (*freq*)

Voltage		Real
Description	Voltage	
Symbol	volts	
Default units	kV	
Minimum value	-1e+008	
Maximum value	1e+008	
Data type	Literal	
Default value	120.0	
Conditional statement		

Input Field Property Settings (*volts*)

When the component is evaluated, the Symbol names (*freq* and *volts*) are given the value entered in their respective fields. In this case, *freq* = 60.0 and *volts* = 120.0. The user is free to insert these variables elsewhere within the component script by using the \$ operator.

Something similar to the following comments could appear in either the Fortran, DSDYN, or DSOUT segments:

```

1 ! User-Defined Component: $(Defn:Name)
2 !
3 ! Frequency: $freq Hz, Voltage: $volts kV
4 !

```

Fortran Segment Code

```

1 ! User-Defined Component: my_new_comp
2 !
3 ! Frequency: 60.0 Hz, Voltage: 120.0 kV
4 !

```

PSCAD Fortran File Code following Compilation

This feature is excellent for formatting source code comments.

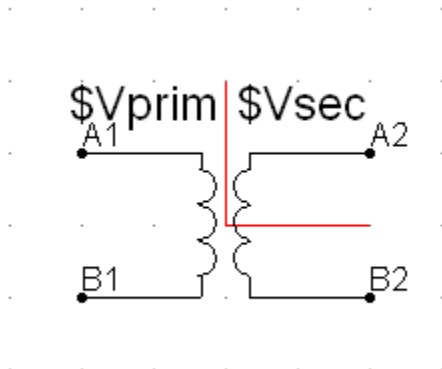
Literal values can also be substituted directly into text labels in the Graphic section, by using the \$ operator. This is a convenient feature, as it allows users to change text appearance on the component graphic according to say, for example, input parameters.

EXAMPLE 10-5:

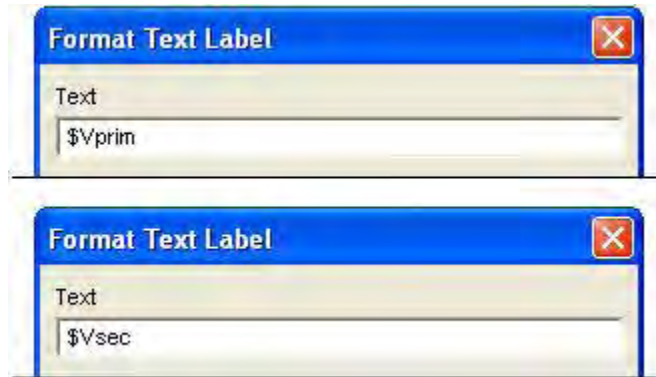
A user has created a single-phase, two-winding transformer component and wishes to display both the primary and secondary winding voltage ratings directly on the component graphic. The primary voltage rating input parameter *Vprim* is entered as 25.0 [kV] and the secondary input *Vsec* is set at 4.0 [kV] as shown below:

Winding #1 voltage (RMS)	230.0 [kV]
Winding #2 voltage (RMS)	230.0 [kV]

In the component Graphic section, the user adds two new text labels (positioned accordingly) and enters *\$Vprim* and *\$Vsec* as shown below:

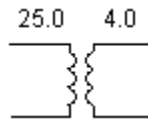


Component Graphic Containing Text Labels



Text Label Property Settings

When the component instance is viewed on the Circuit canvas, input parameter values for  $V_{prim}$  and  $V_{sec}$  will appear directly on the component graphic. For this example,



## Contextual Substitution of Text

Contextual substitution is a formal way of handling text substitutions from other contexts outside the definition. Contextual substitution is enabled by inserting delimiting symbols following the \$ operator. The standard syntax is as follows:

$$\$(\langle \text{Context} \rangle : \langle \text{Key} \rangle)$$

The substitution contains both a  $\langle \text{Context} \rangle$  and item  $\langle \text{Key} \rangle$  separated by a scope operator and delimited by parentheses, where:

- **$\langle \text{Context} \rangle$** : The context that is to be used to scope the item  $\langle \text{Key} \rangle$ . See below for a list of valid  $\langle \text{Context} \rangle$  names.
- **$\langle \text{Key} \rangle$** : Item key (actual substitution) that is to be placed at the substitution point. See below for a list of valid  $\langle \text{Key} \rangle$  names.

Contextual substitution may be used in text labels within the Graphic section, as well as comments in the Fortran, DSDYN, DSOUT, Branch, Model-Data, Matrix-Fill, Transformers, or T-Lines segments. In addition, it may also be used to substitute global substitutions and various keys under their respective contexts.

The following table lists all valid context names that may be used in place of  $\langle \text{Context} \rangle$  and  $\langle \text{Key} \rangle$  as described above (both *Context* and *Key* names are case sensitive):



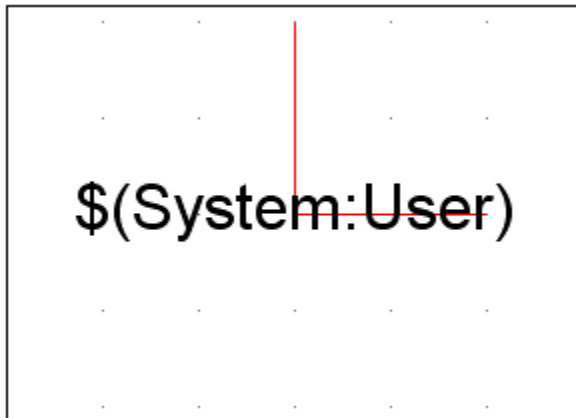
Context Name	Description	Available Keys
<none>	An empty context key is automatically context of the Component Instance.	
Defn	Context is that of the Definition of the Component Instance.	Name, Desc, Path
Project	Context is that of the Project that contains the Component Instance.	Name, Desc, Version, Author, Path
Session	Context is that of the current running session of the application (PSCAD).	Name, Version
System	Context is that of the operating system.	Name, Version, User

When using substitutions within the context of the component instance, there is no need to include the <Context> identifier and it hence may be excluded along with the scope operator. In such cases the syntax is:

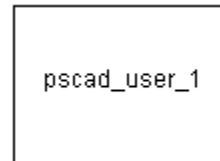
\$ (<Key>) or simply \$<Key>.

EXAMPLE 10-6:

A user has created a definition and wants to display the current user on the component Graphic. A text label is added the Graphic section, which contains a \$ operator extracting the required information.



Component Graphic Containing Text Label



Component Instance on Circuit Canvas

### ##DIM: Substituting Port Dimension

The dimension of any connection port may be substituted as an integer directly into the component script using the *##DIM* operator. The *##DIM* operator may appear in Fortran, DSDYN, DSOUT and the Checks segments. This operator must appear in the following format:

`$#DIM(<Node_Name>)`

Where:

- **<Node\_Name>**: The connection port Symbol name.

An example usage of this operator is presented below:

```
!
#IF $#DIM(NT)==3 || $#DIM(NF)==3 ! Used to define #IF logic based on the
dimension of port NT.
!
ERROR Dimesion Exceed - Maximum 3 phase : $#DIM(NT)<4 && $#DIM(NF)<4 ! Used
in the Checks segment.
!
IF (($si .GT. 0).AND.($si .LE. $#DIM(in))) THEN ! Used directly as a
substitution in Fortran code.
!
```

## **\$#Component: Substituting Component Instance ID**

The instance ID number of any component instance may be substituted using the  *\$#Component* operator. The  *\$#Component* operator may appear in Fortran, DSDYN and DSOUT segments. This operator must appear in the following format:

`$#Component`

The primary purpose of this operator is to provide the instance id of the calling component to EMTDC for the purpose of defining a messaging source. In this way, PSCAD can provide a navigable link in the build messages pane if a message is generated by a specific component. An example usage of this operator is presented below:

```
!
CALL COMPONENT_ID(ICALL_NO,$#Component) ! Used to assign Call number and
Instance number of a component. These values are
! passed to EMTDC via the
COMPONENT_ID intrinsic subroutine.
!
```

## **% Data Substitution Prefix Operator**

The % Data Substitution Prefix Operator (or simply '%') can be used to make a direct text substitution. Data substitution is very similar to the \$ Value Substitution Prefix Operator, except that it substitutes the exact contents of an input parameter as text, and may only be used to substitute input parameters.

The % operator may be used in text labels within the Graphic section, as well as within commented code in the Fortran, DSDYN, DSOUT, Branch, Model-Data, Matrix-Fill, Transformers, or T-Lines segments.

## EXAMPLE 10-7:

Consider Example 10-4 above, but replace the \$ operator with the % operator. The following comments could then appear in either the Fortran, DSDYN or DSOUT segments:

```

1 ! User-Defined Component: $(Defn:Name)
2 !
3 ! Frequency: %freq, Voltage: %volts
4 !

```

Fortran Segment Code

```

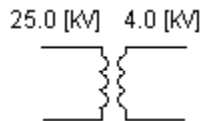
1 ! User-Defined Component: my_new_comp
2 !
3 ! Frequency: 60.0 [Hz], Voltage: 120.0 [kV]
4 !

```

PSCAD Fortran File Code following Compilation

## EXAMPLE 10-8:

Consider Example 10-5 above, but replace the \$ operator with the % operator in the new text labels. When the component instance is now viewed on the Circuit canvas, values of the input parameters  $V_{prim}$  and  $V_{sec}$  will appear on the component graphic as follows:



## { } Braces

Braces can be used to perform mathematical, logical and formatting operations before all other definition script is considered by the compiler. Braces can appear all over the component definition. There are two main forms in which braces may be used: Expression substitution and block processing. The brace syntax is as follows:

```
[ $ ] { <Expression> }
```

Where:

- **\$**: Optional. Optional. The \$ Value Substitution Prefix Operator is not required in some instances, such as in most block processing operations.
- **<Expression>**: Can be a mathematical expression, or simply a block of text.

It is important to note that mathematical and logical operations within braces are performed prior to the insertion of code into the Fortran or Data files. Subsequently, these operations cannot involve variables – only literals and constants (such as constant input parameters, local constants, or constants defined in the Computations segment).

Although simple examples are provided here, you can find more examples by studying the definitions of master library components.

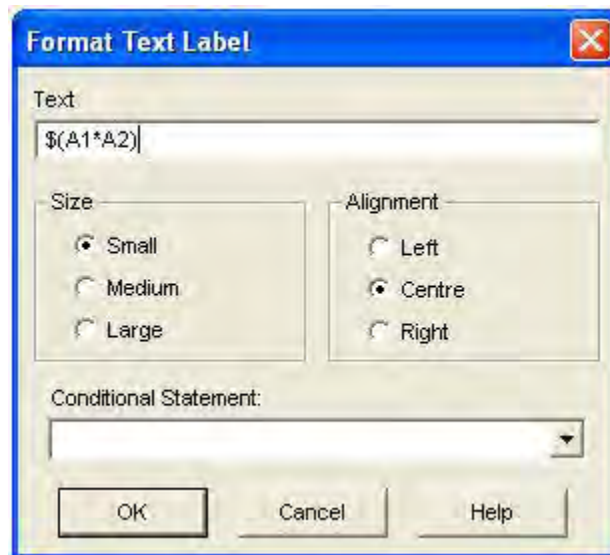
---

**EXAMPLE 10-9:**

The following example shows how expression braces can be used within text labels: A mathematical operation is performed on substituted values. Say that two input parameters are entered as Symbol names *A1* and *A2* with values 2.0 and 3.0 respectively. The user wants to display another value, based on these values, directly on the component graphic as follows:

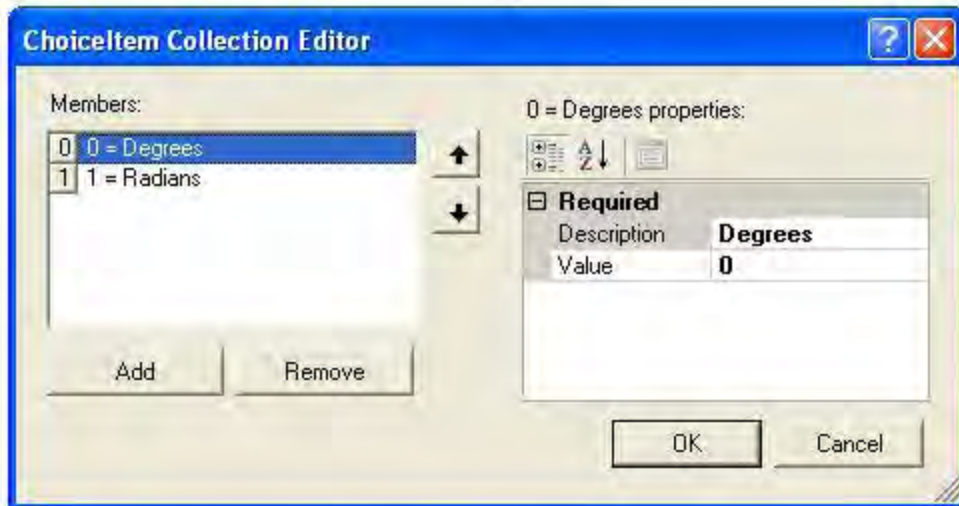
$$\text{Text Label} = A1 * A2 = 6.0$$

The following is entered into the text label properties dialog:

**EXAMPLE 10-10:**

A user has modified the function *PH\_CON* in Example 10-3 to include phase angle conversion in both directions (i.e. degrees to radians and radians to degrees). In the component Parameters section, a choice list with Symbol name *Ptype* is designed with two options as shown below:

Input In?	Choice
Description	Input In?
Symbol	PType
Default value	0
Conditional statement	
Choice List	(Collection)



Choice Item Collection Editor (*Ptype*)

The function *PH\_CON* is modified to include a second argument indicating the conversion type required, based on the state of *Ptype*. The additional argument is a character string, where *D2R* indicates degrees to radians conversion and *R2D* radians to degrees. Braces are used, along with a *#CASE* directive and *~* Line Continuation Operator:

```

1 #FUNCTION REAL PH_CON Frequency Converter
2 !
3     $out = PH_CON($in,~
4 !
5 #CASE FType {'D2R'}) {'R2D'})
6 !

```

Fortran Segment Script

```

!
RT_2 = F_CON(RT_1,'D2R')
!

```

Actual Fortran Code Following Compile

*RT\_1* and *RT\_2* are variable substitutions given by EMTDC: Their names may vary.

## ! Comment Indicator

The exclamation point is used throughout the component definition segments to indicate that a particular line of code is a comment. All lines of script preceded by this operator will be considered source comments for inclusion in the project Fortran or Data files.

**NOTE:** Commented code that is written to the data file, requires that the Comment Indicator be in the first column from the left.

### EXAMPLE 10-11:

Comments should be used to describe the user's code, as well as to provide spacers and section breaks in order to enhance readability. The comment indicator may be used everywhere except the Branch segment.

The following example illustrates the use of comments in the Fortran or DSDYN segments in the a user component definition:

```
!
! MY FIRST PSCAD SCRIPT
! -----
!
! Storage:
!
#STORAGE REAL:1
!
! Main body of script:
!
```

**NOTE:** See the #STORAGE Directive for more details.

## Script Directives

#IF, #ELSEIF, #ELSE, #ENDIF

#CASE

#STORAGE

#LOCAL

`#BEGIN/#ENDBEGIN``#FUNCTION``#SUBROUTINE``#OUTPUT``#TRANSFORMERS``#WINDINGS``#VERBATIM`

Script directives are used to facilitate the creation of source code, which will ultimately be used in the building of an executable program representing the project. Each directive type has a specific purpose, as explained in the following sections. Script directives always begin with a pound # symbol prefix, where spaces preceding and following the # symbol are allowed.

Directives are an important tool when scripting definitions, as they instruct the internal compiler what source code to build and how to build it. It is highly recommended that directives be used wherever and whenever possible, as they ensure that user defined script will remain compatible with coding standards as they change and evolve over time.

## **#IF, #ELSEIF, #ELSE, #ENDIF**

#IF, #ELSEIF, #ELSE, #ENDIF directives are used specifically as logical structures, in the same manner as their equivalent IF, ELSE, etc. source language intrinsic functions. Within script however, they are used to include or exclude blocks of script.

#IF, #ELSEIF, #ELSE, #ENDIF directives can be used in all segments and should appear as follows:

```
#IF <Logic>
    ...Application_Code...
#elseif <Logic>
    #IF <Logic>
        ...Application_Code...
    #ELSE
        ...Application_Code...
    #ENDIF
#else <Logic>
    ...Application_Code...
#endif
```

If just a simple IF-THEN condition is required, the following shortcut using expression braces may also be used:

```
#IF <Logic> {<Expression>}
```

<Logic> is a logical expression using logical operators. <Expression> can be a variable definition.

---

#### EXAMPLE 10-12:

A user needs to change the output of a signal generator model, according to whether a sine or cosine output is required. The component definition provides an input parameter choice list in the Parameters section, called *Type*. If this parameter is *1*, then the output is sinusoidal (if *0* then co-sinusoidal).

The following code should appear in the Fortran, DSDYN or DSOUT segments of the component:

```
!
! Signal Generator
!
#IF Type == 1
    $OUT = SIN(TWO_PI*$F)
#ELSE
    $OUT = COS(TWO_PI*$F)
#ENDIF
!
```

Where *F* is a pre-defined variable (perhaps an input parameter or Computations segment variable) and *OUT* is an output port connection in the Graphic section.

**NOTE:** *Type == 1* in the above code is a Logical Expression. See Expression Evaluation for more details.

Using Enfolding Operators, the above example could also be written as:

```
!
! Signal Generator
!
#IF Type == 1 {      $OUT = SIN(TWO_PI*$F) }
#IF Type != 1 {     $OUT = COS(TWO_PI*$F) }
!
```

---



## EXAMPLE 10-13:

A user has created an electrical component that can either be a simple resistor, inductor or capacitor. The component definition provides an input parameter choice list in the Parameters section, called *Type*. This parameter can either be 1, 2 or 3 to represent a resistor, inductor or capacitor respectively. Three other input text boxes also exist (called *R*, *L* and *C*) in order to specify the respective values of these elements.

The following code should appear in the Branch segment:

```
#IF Type == 1
    $N1 $N2 $R 0.0 0.0
#ELSEIF Type==2
    $N1 $N2 0.0 $L 0.0
#ELSE
    $N1 $N2 0.0 0.0 $C
#ENDIF
```

*N1* and *N2* are electrical port connections defined in the Graphic section. Using Enfolding Operators, the above example could also be written as:

```
!
#IF Type == 1 { $N1 $N2 $R 0.0 0.0 }
#IF Type == 2 { $N1 $N2 0.0 $L 0.0 }
#IF Type == 3 { $N1 $N2 0.0 0.0 $C }
!
```

## #CASE

The #CASE directive is a short-form notation method, which can be used in place of #IF, #ELSEIF, #ELSE, #ENDIF directives. It is normally used in conjunction with the ~ Line Continuation Operator to provide a space saving alternative, especially when a great number of #IF, #ELSEIF, #ELSE, #ENDIF directives are required.

The #CASE directive can be used in all segments, and should appear as follows:

```
#CASE <Expression> {<Clause_0>} {<Clause_1>} ...
```

<Expression> must return an integer from 0 to *n*, and can either be a mathematical expression or simply a defined variable. <Clause\_*n*> represents what is to occur given the value of the expression, and where in the sequence of clauses it resides. For example, the results of <Clause\_0> will occur if <Expression> is equal to 0. If <Expression> is equal to 1, <Clause\_1> will occur, etc.

---

**EXAMPLE 10-14:**

Consider the signal generator output discussed in Example 10-12:

```
!  
! Signal Generator  
!  
#IF Type == 1  
    $OUT = SIN(TWO_PI*$F)  
#ELSE  
    $OUT = COS(TWO_PI*$F)  
#ENDIF  
!
```

The following is a simple illustration of an equivalent script using the #CASE directive:

```
!  
! Signal Generator  
!  
    $OUT = ~  
#CASE Type {~COS~} {~SIN~}  
~(TWO_PI*$F)  
!
```

Note that *Type* should be either 0 or 1.

---

**EXAMPLE 10-15:**

The following illustrates how the #CASE directive is used in the Transformers segment of the 3-Phase, 2-Winding classical transformer component, located in the master library.

Here, *YD1* and *Lead* are both component input parameters. *YD1* can equal either 0 or 1 and *Lead* either 1 or 2. Multiplying these two integers can result in 0, 1 or 2, and this information is used to determine which clause to select:

```
!
#CASE YD1*Lead {$A1 $G1~} {$A1 $B1~} {$A1 $C1~}
!
```

## #STORAGE

This directive is required only if a definition is utilizing EMTDC Storage Arrays. Its purpose is to define the manner in which memory is used – specifically the type and amount of EMTDC storage to be allocated. The #STORAGE directive acts as a direct interface to the storage arrays; which provide time step to time step data transfer capability, as well as data transfer between BEGIN and corresponding DSDYN or DSOUT sections. EMTDC array usage must be specified with a #STORAGE directive to ensure that memory is properly dimensioned by PSCAD at compile time.

The #STORAGE directive is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#STORAGE <TYPE>:<Number>
```

The actual #STORAGE statement can appear anywhere in the segment, but should be placed near the top. <TYPE> refers to the storage array type (see table below). <Number> is the amount of elements to be stored, and may be a substituted constant.

The following storage examples are all valid when using the \$ Substitution Prefix Operator with this directive:

```
#STORAGE REAL:$ (X) INTEGER:$ (Y) // X and Y are literal parameters
#STORAGE REAL:$#DIM(N1) INTEGER:$#DIM(N2) // N1 & N2 are electrical ports
#STORAGE INTEGER:$#DIM(N1) LOGICAL:7
#STORAGE REAL:$ {X+Y}
```

	Type	EMTDC Storage Array	Description
Inter Time Step Data Transfer	STOR	STOR (NEXC)	This array is left over from PSCAD V2 and is deprecated. Do not use this array if possible.
	REAL	STORF (NSTORF)	For floating point (REAL) storage only.
	INTEGER	STORI (NSTORI)	For INTEGER storage only.
	LOGICAL	STORL (NSTORL)	For LOGICAL storage only.

	COMPLEX	STORC (NSTORC)	For COMPLEX storage only.
BEGIN to DSDYN/DSOUT Data Transfer	RTCF	RTCF (NRTCF)	For floating point (REAL) storage only.
	RTCI	RTCI (NRTCI)	For INTEGER storage only.
	RTCL	RTCL (NRTCL)	For LOGICAL storage only.
	RTCC	RTCC (NRTCC)	For COMPLEX storage only.

## EXAMPLE 10-16:

Consider the following definition script, taken from the Fortran segment of the XYZ Transfer Function component in the master library:

```
#IF NL==0
#STORAGE INTEGER:2 REAL:110
#ELSEIF NL==1
#STORAGE INTEGER:2 REAL:1100
#ELSEIF NL==2
#STORAGE INTEGER:2 REAL:11000
#ENDIF
#FUNCTION REAL XYZFUNC XYZ-Transfer function
! File name: $FILE
      SZ = XYZFUNC ($X, $Y, $MODE, $Xoff, $Yoff, $Zoff, $Kx, $Ky, $Kz)
```

This script uses the #IF, #ELSEIF, #ELSE, #ENDIF directives, along with #STORAGE to determine the EMTDC storage array requirements for this component. Depending on the variable *NL*, which in this component represents the *Number of Lines in the File* input parameter, the number of REAL elements to be allocated ranges from 110 to 11000. The number of INTEGER elements is always 2.

Note that the above is how variable storage was dealt with prior to the introduction of substitution in storage directives. A much simpler and more efficient equivalent to the above code is as follows, where the variable *NREAL* is derived from *NL*, the *Number of Lines in the File* input parameter, and substituted as the REAL storage dimension:

```
#STORAGE INTEGER:2 REAL:$ (NREAL)

#FUNCTION REAL XYZFUNC XYZ-Transfer function

! File name: $FILE

      $Z = XYZFUNC ($X, $Y, $MODE, $Xoff, $Yoff, $Zoff, $Kx, $Ky, $Kz)
```

This allocation represents the type and number of EMTDC storage array elements that are used internally by the function *XYZFUNC*. That is to say that the *XYZFUNC* function code utilizes both the *STORI* and the *STORF* arrays to store and transfer data from time step to time step during a simulation.

## #LOCAL

This directive is used to declare local variables directly within definition script. Quite often, it is necessary to declare local variables to say, define a dummy variable for an unused subroutine argument, or perhaps as an intermediary value in a set of equations.

The #LOCAL directive instructs the PSCAD compiler to place a local variable declaration directly in the parent module Fortran file when the project is built. #LOCAL variables do not require the \$ Value Substitution Prefix Operator.

The #LOCAL directive is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#LOCAL <TYPE> <Name> <Array_Size_1> <Array_Size_2>
```

<TYPE> can be either REAL, INTEGER, LOGICAL, or COMPLEX. <Name> is the given name for the local variable. <Array\_Size\_1> is an optional integer or substituted constant that defines the size of the array. If declaring a matrix, then <Array\_Size\_1> defines the number of rows and <Array\_Size\_2> defines the columns. If the variable is a scalar, then leave <Array\_Size\_1> and <Array\_Size\_2> blank.

The following storage examples are all valid when using the \$ Substitution Prefix Operator with this directive:

```
#LOCAL REAL X $#DIM(N1) 2           // N1 is an electrical port
#LOCAL REAL Y $#DIM(N1) $#DIM(N2) // N1 & N2 are electrical ports
#LOCAL INTEGER Z $(XXX)            // XXX is a literal parameter
#LOCAL INTEGER ZZZ ${X-Y}          // X and Y are literal parameters
#LOCAL COMPLEX D 2                  // D is a locally declared, complex array
```

## EXAMPLE 10-17:

A user's custom component requires two local variables for use as subroutine arguments. According to a conditional statement based on a pre-defined variable *A*, a local INTEGER variable *MY\_X* and a local REAL array *Error* are defined before the subroutine is called.

The #LOCAL declarations should appear as follows:

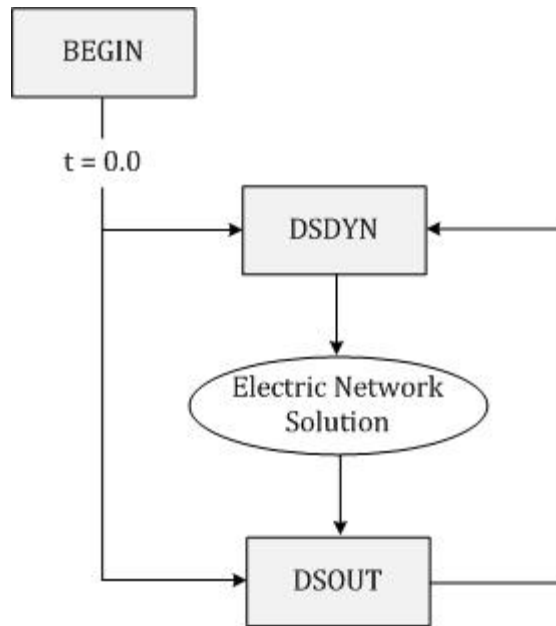
```
#LOCAL INTEGER MY_X
#LOCAL REAL Error 2
!
#IF A > 1
    MY_X = 1
    Error(1) = 0.2
#ELSE
    MY_X = 0
    Error(2) = 0.8
#ENDIF
!
    CALL SUB1(MY_X, Error)
!
```

**NOTE:**  $A > 1$  in the above code is a Logical Expression. See Expression Evaluation for more details.

---

## #BEGIN/#ENDBEGIN

This directive block provides a portal to the BEGIN outer process layer in EMTDC, which is essentially a module based, subroutine that is called prior to both DSDYN and DSOUT at time zero.



This process layer provides *Runtime Configuration*, for the support of components existing in modules with multiple instances. For more information on these topics, see Multiple Instance Modules in Chapter 5 of this manual and/or Runtime Configuration in Chapter 2 of the EMTDC manual.

The #BEGIN/#ENDBEGIN directive block is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#BEGIN
...
#ENDBEGIN
```

The contents of the directive block is arbitrary and dependent on the component being designed – but will of course require a minimum of standard script in order to be of any use. Using the #BEGIN block will ensure that time zero code is considered only once at compile time, avoiding a *TIMEZERO* logic check every time step; thereby ensuring efficient simulation speed. The #BEGIN block will require the following general parts:

- **Time Zero Initialization:** All custom component time zero initialization code placed within the #BEGIN directive block, will be inserted as Fortran code into either the DSDYN or DSOUT BEGIN subroutine (corresponding to the parent module) when the project is compiled.
- **Component Instance and Call Numbers:** If there is a potential for any warning or error messages to be issued within the #BEGIN block, then both the instance and call numbers must be provided to EMTDC. If a message is indeed issued, its source can be mapped from within PSCAD by using these numbers. The instance and call numbers are provided through the use of the *COMPONENT\_ID* subroutine. See Chapter 5 – Custom Model Design in the EMTDC manual for more details.

---

#### EXAMPLE 10-18:

One of the more simple components in the master library to include BEGIN code is the Exponential Functions component. This component models an exponent function, which can possess both a base and an exponential coefficient:

$$y = A \cdot e^{B \cdot x} \text{ or } y = A \cdot 10^{B \cdot x}$$

The coefficients  $A$  and  $B$  are declared Constant, meaning that these parameter values could be defined by variables that may possess different numeric values, depending on the instance of its parent module (i.e. the canvas on which this component resides). Therefore, these input quantities must be stored in sequence within the BEGIN storage arrays, in order to be extracted in the proper sequence according to module instance.

The following script is a simplified version of what appears in the Fortran segment of the Exponential Functions component definition (it is assumed base  $e$  and a scalar input signal):

```
#BEGIN

    RTCF (NRTCF)    = $A

    RTCF (NRTCF+1) = $B

    NRTCF = NRTCF + 2

#ENDBEGIN

#STORAGE RTCF:2

!

    $OUT = RTCF (NRTCF) * EXP (RTCF (NRTCF+1) * $IN)

    NRTCF = NRTCF + 2

!
```

The #BEGIN/#ENDBEGIN directive block at the top of this script, ensures that code is placed by the compiler in the BEGIN section of the system dynamics: The coefficients  $A$  and  $B$  are stored in the proper sequence, in accordance with the present storage pointer value  $NRTCF$ .

The rest of the script outside of the #BEGIN/#ENDBEGIN directive block is inserted in either DSDYN or DSOUT sections of the system dynamics. Notice that the quantities required to represent the coefficients  $A$  and  $B$  (i.e.  $RTCF(NRTCF)$  and  $RTCF(NRTCF+1)$  respectively) are extracted from the same storage locations as they were stored at time zero. The  $NRTCF$  pointer is incremented accordingly.

See EMTDC Storage Arrays in Chapter 5 of the EMTDC Manual for more details on this functionality.

## #FUNCTION

This directive is used to declare the existence of a function, and the argument type it returns. #FUNCTION is mandatory if a function is used within a component definition: It will ensure that a function declaration statement is placed within any source subroutine where the component code is placed.

The #FUNCTION directive is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#FUNCTION <TYPE> <Name> <Description>
```

<TYPE> can be either REAL, INTEGER, or LOGICAL. <Name> is the given name of the function. <Description> will be included as a comment line near the beginning of the corresponding module source subroutine.



## EXAMPLE 10-19:

The Hard Limiter component in the master library utilizes a REAL function called *LIMIT* to determine the value linked to an external port connection *O*, according to input arguments *LL*, *UL*, and *I*. *LL* and *UL* represent the component input parameters *Lower Limit* and *Upper Limit* respectively, whereas the *I* variable is pre-defined by an input port connection in the Graphic section.

The following code appears in the Fortran segment of the Hard Limiter component definition:

```
#FUNCTION REAL LIMIT Hard Limiter
!
      $O = LIMIT($LL, $UL, $I)
!
```

## #SUBROUTINE

This directive is used to provide a description for a subroutine that is being called from the component. #SUBROUTINE is used simply for cosmetic purposes, and is not mandatory (although its use is recommended anyway).

The #SUBROUTINE directive is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#SUBROUTINE <Name> <Description>
```

<Name> is the given name for the subroutine. <Description> will be included as a comment line near the beginning of the corresponding module source subroutine.

## EXAMPLE 10-20:

A user includes a call to a subroutine, named SUB1, within the component definition script. It is desired that a description be added for clarity.

The following code should appear in the Fortran, DSDYN or DSOUT segment of the component:

```
#SUBROUTINE SUB1 User Subroutine
!
      CALL SUB1($X, $Y, $Z)
!
```

X, Y and Z are pre-defined variables that could be port connections, Computations variables or input parameters.

## #OUTPUT

This directive extracts specified data values so that they may be monitored, plotted, or used externally by other components. #OUTPUT performs two tasks: It defines a new variable according to a specified name, and then assigns it a value according to an expression.

The #OUTPUT directive is used only in the Fortran, DSDYN or DSOUT segments, and should appear as follows:

```
#OUTPUT <TYPE> <Name> <Array_Size> {<Expression>}
```

<TYPE> can be either REAL, INTEGER, or LOGICAL. <Name> is the given name for the variable. <Array\_Size> is an optional integer, which defines the size of the array. If the variable has only a single dimension, then leave <Array\_Size> blank. <Expression> can be a mathematical expression, a storage location, or simply a defined variable.

### EXAMPLE 10-21:

A new variable declared by the #OUTPUT directive may have its value defined in many different ways. The following list gives examples of some of the possible methods. Note that the master library also contains a multitude of examples on the use of #OUTPUT within its component definitions.

```
! Defines a REAL variable 'freq' and substitutes
! the value of a pre-defined variable 'Fout'.
!
#OUTPUT REAL freq {$Fout}
!
! Defines an INTEGER variable 'Xon' and assigns
! it the value of a storage location.
!
#OUTPUT INTEGER Xon {STORI(NSTORI+1)}
!
! Defines an REAL variable 'POut' and assigns
! it the value of a given mathematical
! expression.
!
#OUTPUT REAL POut {$V*$I}
!
```

## #TRANSFORMERS

This directive should be included whenever mutually coupled windings are to be represented in a component definition. It is normally used in conjunction with the #WINDINGS directive (described below). Examples of master library components that utilize this directive are (of course) the transformers and the  $\pi$ -section components.

#TRANSFORMERS has two primary purposes:

1. Provides a method to sequentially number all components in a project that contain mutually coupled windings, for the purpose of dimensioning EMTDC matrices and arrays.
2. Provides addressing information for the monitoring of mutually coupled winding currents. Transformer winding currents are measured using the  $CDCTR(M,N)$  internal EMTDC matrix.  $CDCTR(M,N)$  is the electrical current through the Mth winding of the Nth transformer.

The #TRANSFORMERS directive is used only in the Transformers segment, and should appear as follows:

```
#TRANSFORMERS <Number>
```

<Number> indicates the total number of transformers within the component.

### EXAMPLE 10-22:

The 3-Phase, 2-Winding classical transformer component, located in the *Transformers* section of the master library, consists of three, single-phase transformers.

The directive would appear as follows in the Transformers segment:

```
#TRANSFORMERS 3
#WINDINGS 2
```

## #WINDINGS

This directive is used to assign the number of coupled windings in a transformer, and is normally used in conjunction with the #TRANSFORMERS directive. PSCAD will look for the highest number associated with all existing #WINDINGS directives and then assigns that number (as the maximum number of windings) to the Map file.

The #WINDINGS directive is used only in the Transformers segment, and should appear as follows:

```
#WINDINGS <Number>
```

<Number> indicates the maximum number of coupled windings within that specific component.

See Example 10-22 above.

## #VERBATIM

This directive is used to pass a line of script directly into the Fortran file from your component, unmodified and unprocessed.

The #VERBATIM directive should appear as follows:

```
#VERBATIM {<Text>}
```

<Text> can be any line of text, such as a comment, compiler directive or source code. <Text> will appear in the Fortran file generated by PSCAD exactly as is (ie. verbatim).

```
! Caution should be exercised as ANY line of code will be written to
! the Fortran file, be it Fortran compatible or not!
!
! PSCAD Script:
!
#VERBATIM {! This is a comment line.}
#VERBATIM { X = 1.0 ! This is a line of Fortran code.}
#VERBATIM {@#$%^&*!& This is a line of rubbish.}
!
! Fortran File:
!
! This is a comment line.
    X = 1.0 ! This is a line of Fortran code.
@#$%^&*!& This is a line of rubbish.
```

## ~ Line Continuation Operator

Occasionally, it may be desirable to break a single script statement into multiple lines. This is particularly useful when a statement is very long, or if part of the statement is variable in accordance with conditional statements.

Line continuations are used in definition script mostly for clarity. That is, PSCAD will automatically break segment lines that exceed the Fortran standard of 72 columns anyway, and so the user does not normally need to worry about this. However, to ensure that code is easily readable to everyone, line continuations should be used.

The line continuation operator is used as follows: A line ending in the ~ symbol will be joined with the next line if it starts with a ~ symbol as well. This processing is performed after conditional statements have been interpreted and is used in formatting the text for output into the Fortran or Data files.

**NOTE:** The line continuation operator can be used anywhere, except it cannot be used on lines containing a directive statement. The only exception to this is the #CASE directive, which supports the use of the continuation operator.

---

EXAMPLE 10-23:

In Example 10-12, #IF, #ELSEIF, #ELSE, #ENDIF directives were used to define the output shape of a signal generator model. The script from that example is shown again below:

```

!
! Signal Generator
!
#IF Type == 1
    $OUT = SIN(TWO_PI*$F)
#ELSE
    $OUT = COS(TWO_PI*$F)
#ENDIF
!

```

As a simple illustration of how a ~ line continuation operator can be exploited, the above script is re-written using an equivalent method:

```

!
! Signal Generator
!
    $OUT = ~
#IF Type==1
~SIN~
#ELSE
~COS~
#ENDIF
~(TWO_PI*$F)
!

```

If *Type* = 2, then after the #IF, #ELSEIF, #ELSE, #ENDIF directives and line continuation operators are processed, the definition script would appear as follows:

```

!
! Signal Generator
!
    $OUT = COS(TWO_PI*$F)
!

```

# Expression Evaluation

All intrinsic Fortran functions are available through direct coding in the Fortran, DSDYN or DSOUT segments. However in segments such as Computations, some mathematical and logical functionality is also required: PSCAD possesses a limited set of its own intrinsic mathematical functions and logical operators for use within segments where direct access Fortran functions is not available.

## Mathematical Functions

Mathematical computations can be performed on input parameters, input signals, or on results of other computations.

The table below lists all of the available intrinsic mathematical functions. These functions are used only in the Computations segment:

Function	Description
CEIL (x)	Rounds fraction to next upper integer
FLOOR (x)	Rounds fraction to next lower integer
ROUND (x)	Adds 0.5 to a REAL value and then performs an INT function (i.e. ROUND(X) = INT(X+0.5)). Do not use with negative numbers
TRUNC (x)	Rounds x toward zero, returning the nearest integral value that is not larger in magnitude than x.
FRAC (x)	Removes integer part of REAL value (left side of decimal)
REAL (x)	Real part of complex number
IMAG (x)	Imaginary part of complex number
ABS (x)	Absolute value
ARG (x)	Returns the phase angle (or angular component) of the complex number x, expressed in radians.
NORM (x)	Norm of complex number ( $x^2 + y^2$ )
SIN (x)	Sine function
COS (x)	Cosine function
TAN (x)	Tangent function
ASIN (x)	Inverse sine function
ACOS (x)	Inverse cosine function
ATAN (x)	Inverse tangent function
SINH (x)	Hyperbolic sine function

COSH (x)	Hyperbolic cosine function
TANH (x)	Hyperbolic tangent function
SQRT (x)	Square root
LOG (x)	Natural logarithm
LOG10 (x)	Base 10 logarithm
EXP (x)	Exponential
RAND (x)	Random value between 0 and x
P2RX (m, $\theta$ )	Polar to rectangular conversion ( $\theta$ in degrees)
P2RY (m, $\theta$ )	Polar to rectangular conversion ( $\theta$ in degrees)
R2PM (x, y)	Rectangular to polar conversion
R2PA (x, y)	Rectangular to polar conversion
INT (x)	Removes fractional part of REAL value (right side of decimal)
NINT (x)	Returns the nearest integer to the argument

## Arithmetic Operators

Listed below are the arithmetic operators available in PSCAD. These operators are used to perform mathematical calculations primarily in the Computations, Fortran, DSDYN and DSOUT segments.

Function	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Remainder
**	Raise to power
\	Parallel (xy) / (x +y)

## Logical Operators

Listed below are the logical operators available in PSCAD. Logical expressions are primarily used in conjunction with #IF, #ELSEIF, #ELSE, #ENDIF directives and the Ternary Operator. They are also used in the Checks segment.

**NOTE:** Logical expressions will return a value of 1 if true, and a value of 0 if false.

Function	Description
==	Equal to
!=	Not equal to
!	Not
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
	OR
&&	AND

#### EXAMPLE 10-24:

Logical operators can appear in many different areas. The following examples illustrate the various ways these operators can be used:

```

!
! ...with #IF, #ELSEIF, #ELSE, #ENDIF Directives
!
#IF F >= 60.0
    Fout = 60.0
#ENDIF
!
! ...with a Ternary Operator in the Computations
!   segment
!
REAL L = X == 0.0 ? Y*2.0 : Y/3
!
! ...in the Checks segment

```



```
!
ERROR Value too small : R > 0.001
!
```

---

## Ternary Operator

The ternary operator is yet another short form method offered in definition script for representing an IF-ELSE-ENDIF type expression. It allows the user to define a variable, according to certain conditions, in a single line.

The ternary operator is used only in the Computations segment, and should appear as follows:

```
<Logic> ? <Value_if_True> : <Value_if_False>
```

<Logic> is a logical expression using logical operators. <Value\_if\_True> and <Value\_if\_False> can either be a single constant, or a mathematical expression.

**NOTE:** Care must be taken when designing components with ternary operators: Ensure that variables used within the ternary expression will not be disabled under otherwise valid conditions. In other words, unrelated logic may disable one or more of the variables used within the ternary expression, which may render the ternary result invalid. Input variables are enabled/disabled by way of Conditional Statements, Layers and Filters.

---

### EXAMPLE 10-25:

A user wants to define a REAL variable *X* in the Computations segment of a component definition. The value of *X* is to be 1.0 if an input parameter *N* is 2 or 3, and defined by a mathematical expression otherwise.

The following code should appear in the Computations segment:

```
REAL X = (N==2 || N==3) ? 1.0 : SQRT(2)*V
```

Where *V* is a pre-defined constant.

---

### EXAMPLE 10-26:

A user wants to define a REAL variable *Torq* in the Computations segment of a component definition. The value of *Torq* is defined by a mathematical expression, where one element of this expression varies according to a condition. This can be accomplished by using the ternary operator as follows:

```
REAL Torq = (X > 1 ? 0.0 : Tm) + Te*100
```

Where *X*, *Tm* and *Te* are pre-defined constants.

---



## Chapter 10

# Project Debug and Refinement

Ironing out the bugs in a project can be the most challenging part of the design process. No matter how carefully a system is constructed, there are almost always a few errors or warnings issued when the project is compiled and run. This chapter describes the reasons for some of the most common errors and warnings, along with how to deal with them.

Debugging a project can become even more difficult when user-written code is present. PSCAD does not include an integrated tool for debugging user system dynamics code (i.e. source code appended to the system dynamics through user-defined components), and so an external debugging tool must be used: Some possible methods for linking to an integrated debugger are described in this chapter.

In finalizing a project, the user may want to protect (i.e. pre-compile or encrypt) source code when supplying projects and components to other users. Topics are included here that describe both how to pre-build source code into a *library (\*.lib)* or *object (\*.obj)* files, as well as the ciphering tool for encrypting user-defined module code. Both of these methods will enable you to distribute projects to clients, without running the risk of exposing trade secrets.

## Error and Warning Messages

PSCAD does not include a real time error manager, so in order to generate and view any warning or error feedback you must first compile and build your project. Error messaging is displayed in the Build Messages pane. This classification can be very helpful in determining just what the problem is, and how to fix it.

As discussed in the Build and Runtime Message Panes topic, this table includes messages sourced from the building or compilation of the project (*build*). The build messages pane will also display messages involving transmission line and cable solving, as well as other, general informational messages.

## Build or Compilation Messaging

The build or compilation process as you can imagine, is a complicated one. However, it can be simplified a bit by further sub-dividing it into steps:

1. **Building Source and Data Files for Simulation:** In this first step of the build process, PSCAD collects all of the module definitions in the project and compiles them. The result of this compilation is the creation of source (i.e. Fortran (\*.f) files) and Data (\*.dta) files. If one or more problems exist in any module, an error or warning message will be issued. PSCAD will complete this step by compiling all modules flagged for compilation, regardless if errors are detected in any single module: Compilation will not continue to the next step however.
2. **Creating Map File:** Once all of the modules have been built, their respective local nodes and subsystems must be linked together globally – this is performed during the creation of the project Map (\*.map) file. If there are illegal issues with connectivity between modules, or something of the like, pertinent error and warning messages will appear here.
3. **Creating Make File:** The *Make (\*.mak)* file is written as an instructional file for the Fortran compiler. Any problems during this process will be displayed here.
4. **Solving Transmission Segments:** The final process before the simulation executable file is produced is to solve all transmission line and cables in the project. For each transmission segment, PSCAD produces either a Transmission Line Input (\*.tli) or a Cable Input (\*.cli) file and then calls the Line Constants Program (LCP) to solve the segment. If a problem occurs during the process of constructing the input file (ex. Checks segment logic failure), or the LCP solve fails, an error or warning message will be displayed here.

The following image illustrates the messages for the steps described above in the Output Window:

Icon	Component	Instance	Description	Project
!			0 Errors	
!			0 Warnings	
!			13 Messages	
			simpleac	
i			Generating network and source code C:\Users\Public\Documents\PscadBeta\Examples\tutorial\simpleac.gf42\Main.f	simpleac
i			Generating C:\Users\Public\Documents\PscadBeta\Examples\tutorial\simpleac.gf42\simpleac.map	simpleac
i			Time for Compile: 63ms Make: 15ms	simpleac
i	FLAT230	1935965525	Solving right-of-way coupling using C:\Users\Public\Documents\PscadBeta\Examples\tutorial\simpleac.gf42\FLAT230.ttl	simpleac
i			Solve Time = 577ms	simpleac
i			Will execute: call C:\Program Files (x86)\GFortran\4.2.1\bin\gf42vars.bat	simpleac
i			Will execute: make f simpleac.mak	simpleac
i			Will execute: C:\Users\Public\DOCUME~1\PSCADB~1\Examples\tutorial\SIMPLE~1.GF4\simpleac.bat	simpleac
i			Creating EMTDC executable...	simpleac
i			C:\Users\Public\Documents\PscadBeta\Examples\tutorial\simpleac.gf42>call C:\PROGRA~2\GFortran\428484~1.1\bin\gf42vars.bat	simpleac
i			Compiling generated source 'Station.f' into object code.	simpleac
i			Compiling generated source 'Main.f' into object code.	simpleac
i			Linking object code and libraries into binary 'simpleac.exe'	simpleac

Obviously there was no problem building the *simpleac* project above. Any error messages that do occur will appear under their respective categories.

## Runtime Messaging

Upon the successful completion of the build process, the runtime process will begin. Runtime messages are streamed directly from EMTDC to PSCAD into a separate pane called the Runtime Messages pane. These messages are related directly to the runtime process and contain information such as: Messages related to the running of the executable file, EMTDC software copyright information and time summary information.

## EMTDC Non-Standard Messages

These messages are very important, and include; file read errors, matrix manipulation, and any other type of problem arising from the initialization of the run. When initially debugging and refining a project, users should continually consult the messages displayed here.

## Common Output Window Messages

There are a large number of possible error and warning messages that can be generated and displayed in the Output Window. A message source can originate directly from the PSCAD or EMTDC applications, or may be generated by individual components. Most of these messages however, will never occur if a project is judiciously constructed in the first place.

The following descriptions outline the most common occurring messages.

### Warning: Suspicious isolated node detected

This warning is issued if PSCAD detects an open electrical circuit connection on a component - usually caused by an electrical node that is not connected to anything.

If an open circuit connection is indeed what you wish to accomplish, the suggested method to deal with this warning is to connect a large resistance (approximately  $1\text{ M}\Omega$ ) to ground at the node. This will ensure numerical stability and will have negligible affect on the simulation results.

### Unresolved substitution of key '<name>'

A key has not been properly defined or does not exist in the component data.

<name> is the name of the text input field.

## Signal '<name>' type conversion may lose accuracy

This warning is issued if PSCAD detects a REAL signal being sent to an input expecting an INTEGER. PSCAD will automatically convert the REAL signal value to the nearest integer, hence the warning.

<name> is the name of the signal.

## Signal '<name>' source contention at component [<defn\_name>] '<instance\_name>'

This error is issued if PSCAD detects a signal of the same name being generated from more than one source. This error is most commonly caused when component instances with defined internal output variables are copied, hence duplicating the internal output variable.

<name> is the name of the signal. <instance\_name> is the name of the component.

## Signal '<name>' dimension mismatch -> <dim\_1> != <dim\_1>

This error is issued if PSCAD detects that a signal of dimension <dim\_1> is being sent to an input expecting a signal of dimension <dim\_2>. This error commonly occurs with the power electronic switch components, which expect a 2-dimensional input gate signal when set for interpolation.

<name> is the name of the signal.

## Array '<Name>' cannot be typecasted

This error is issued if PSCAD detects an array signal type mismatch. For example, if a data signal array was defined as type INTEGER and the user attempted to tap off a single element with the Data Signal Array Tap component set to type REAL (or vice-versa); or, if an array of type REAL is input into a component, where the external input Connection in question is defined as an INTEGER array.

<name> is the name of the signal.

## Invalid breakout connection to ground at '<Node>'. Node array elements cannot be individually grounded.

This error is related to the use of the Breakout component: Ground components cannot be directly connected to *Breakout* terminals. The *Breakout* is designed specifically for mapping multiple connections on the scalar side to a single array. Since *Ground* nodes cannot be mapped, the compiler will issue this warning. The suggested work around is to use a Current Meter as a series element between the *Breakout* terminal and ground. See the section entitled Valid Connections in the *Breakout* component online help for more details.

<Node> is the name of the *Breakout* reference node connected to ground.

## Short in breakout at '<Node>'. Node array elements must be uniquely defined.

This error is related to the use of the Breakout component: The nodes on the 3-phase side of this component are not actual electrical nodes, but references that will assume the node number to which they are connected. This error is posted if these reference nodes are shorted (i.e. electrically connected together). Each node on the 3-phase side of the *Breakout* component must be unique. See the section entitled Valid Connections in the *Breakout* component online help for more details.

<Node> is the name of the *Breakout* connection which is shorted.

## Branch imbalance between breakouts at '<Node>'. Node array elements cannot be shared between signals.

This error is related to the use of the Breakout component: The nodes on the 3-phase side of this component are not actual electrical nodes, but references that will assume the node number to which they are connected. A special condition that cannot be referenced is referred to as an 'unbalanced' condition, where the imbalance refers to electrical nodes, not actual impedance. The basic rule to remember here is that all branches on the 3-phase side must include at least one series impedance. See the section entitled Valid Connections in the *Breakout* component online help for more details.

<Node> is the name of the *Breakout* connection which is shorted.

## Viewing Build and Data Files

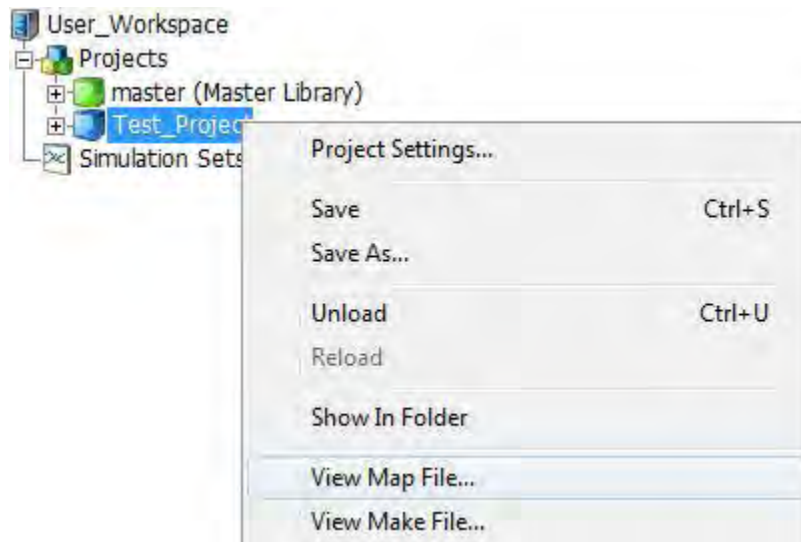
Whenever a project is compiled and built, several files are created and stored in the associated project temporary folder. Some of these files, such as Fortran, Data and Map files, can be useful in debugging. All of these files can be viewed directly from within PSCAD.

### Fortran and Data Files

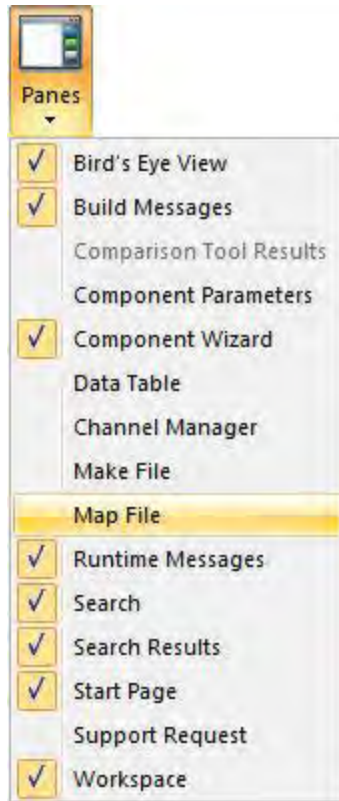
Fortran and Data files can be viewed by simply clicking the Fortran or Data tabs at the bottom of the main canvas (following compilation of course). These files are module specific and are automatically generated by PSCAD. These files may not be edited.

### Map and Make Files

Map and Make files involve the entire project, and are thus not included in the tab bar. To view either of these files, right-click on the project name in the Workspace window and select either **View Map File...** or **View Make File...**



Selecting either view will invoke the associated viewer pane. The *Map* and *Make* file panes can also be opened via the *Panes* button in the ribbon control bar under the *Views* tab.



## Component Ordering

PSCAD includes a smart algorithm, which will automatically sequence (or order) all components involved in the EMTDC System Dynamics. This is carried out automatically in order to ensure that variables are calculated in their proper sequence, and that time step delays are minimized: The algorithm iteratively scans the entire project and then assigns sequence numbers to all existing components. In general, input constants are moved to the top of the sequence, whereas outputs are moved to the bottom.

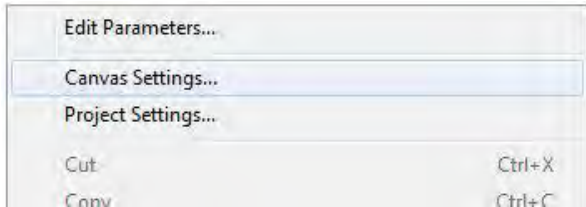
This algorithm should be left on by default at all times; however there may be instances where it is desirable to turn it off and manually order the components during the debugging process. This may be the case if you wish to manually control the feedback points. Alternatively, a feedback can be introduced by inserting a Feedback Loop Selector component in the signal path.

**NOTE:** Component ordering functions are performed by PSCAD on a per module basis. By default, the **Sequence Ordering** option in the *Canvas Settings* dialog is set to **Automatic Assignment** in all new and existing modules. This option may be disabled by the user in selected modules, and yet still maintain automatic ordering in others.

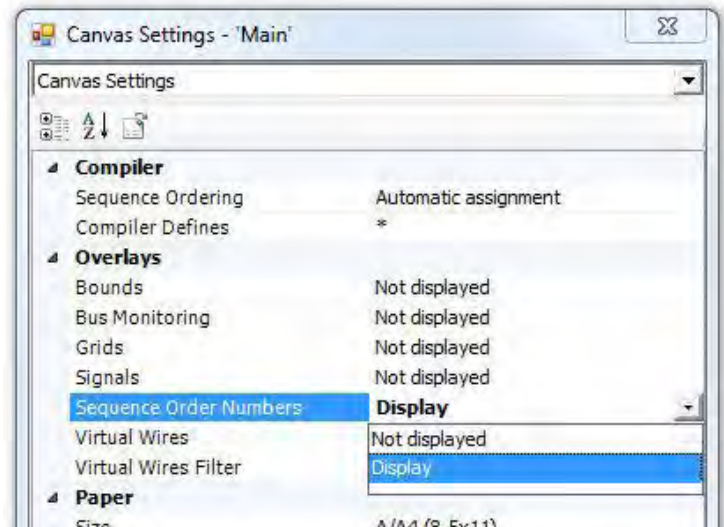
Component ordering features may be accessed through the *Canvas Settings* dialog. See *Editing Module Canvas Settings* for more details.

## Showing Sequence Numbers

Before manually ordering any components, you must first compile the case and then ensure that the **Sequence Order Numbers** setting is enabled in the *Canvas Settings* dialog: To bring-up the *Canvas Settings* dialog, right-click on a blank part of the *Schematic* canvas and select **Canvas Settings...**

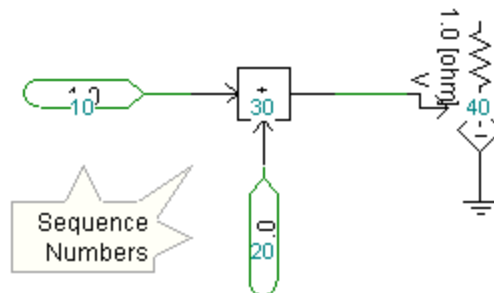


Schematic Canvas Pop-Up Menu



Canvas Settings Dialog

Once this option is enabled, each component instance in your project should have numbers overlaid on top of their graphic, similar to that shown below:



## Colours

There are two possible locations within the system dynamics (i.e. DSDYN or DSOUT) where the code for a particular component can reside. As a result, the sequence numbers are colour coded so that the user can graphically determine where code is. These colours are listed below:

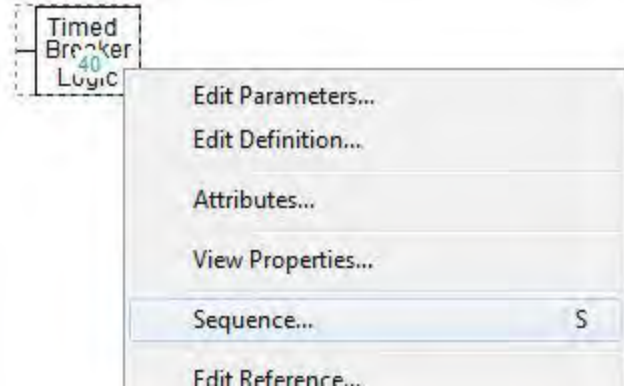
Colour Legend:

- **Aqua:** The component code resides within DSDYN for the current module.
- **Olive:** The component code resides within DSOUT for the current module.

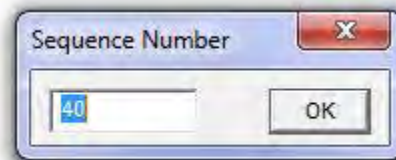
## Manually Setting Sequence Numbers

To manually set sequence numbers, first ensure that the **Sequence Order Numbers** option is set to **Display** (as described above), and that the **Sequence Ordering** option is set to **Manual Assignment** in the *Canvas Settings* dialog. Right-click on a component and select the **Sequence...** option from the pop-up menu.





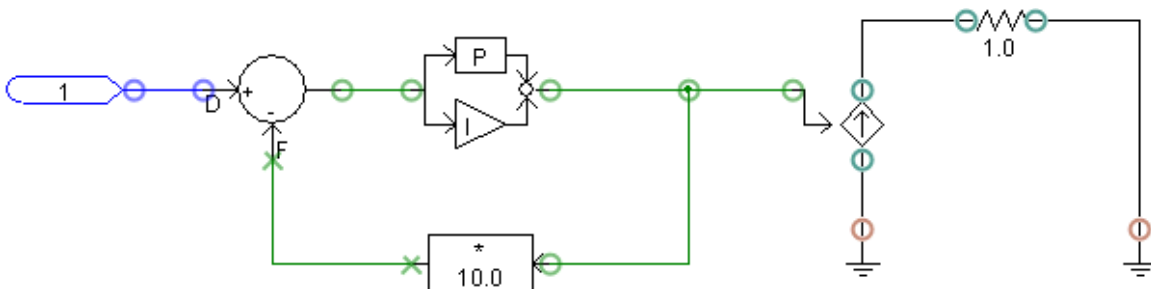
This will bring up the *Sequence Number* dialog.



Enter the desired sequence number and click the **OK** button. Repeat this process for the remaining components and modules.

## Show Signal Locations

Another feature that is helpful when ordering components is the **Signals** option in the Canvas Settings dialog. When this option is enabled, PSCAD will use icons placed on connections and wire termination points to allow for graphical determination of where time step delays are present. In addition, icon colours are used to represent the signal type.



The icons used are listed below with explanations:

- Feed-Forward Connection/Electrical Signal:** This symbol indicates that the signal passing through the connection point is classified as a feed-forward signal, if a control signal. This means that the value of the control signal is always defined within the present time step. This simple is also used to indicate an electrical signal type.

- **✕ Feedback Signal:** This symbol indicates that the signal passing through the connection point is classified as a feedback signal. This means that the value of the signal was defined in the previous time step ( $t - \Delta t$ ). Due to this fact, feedback signals must be written to, and then extracted from storage each time step.
- **☐ Feed Fixed:** This symbol indicates that the signal passing through the connection point is classified as a feed fixed signal. Feed fixed signals are similar to feedback signals, in that their value is always extracted from storage. The difference is that their values are usually defined by an online control, such as a slider or switch.

## Colours

Control Signal Legend:

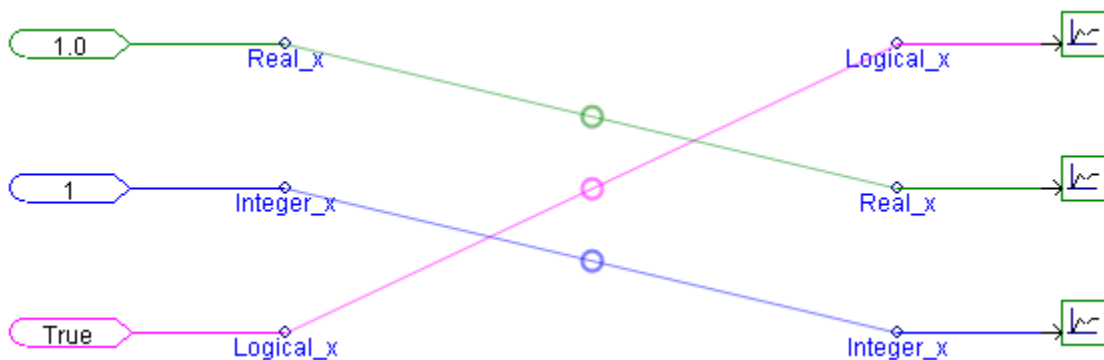
- **Green:** Indicates a signal of type REAL
- **Blue:** Indicates a signal of type INTEGER
- **Magenta:** Indicates a signal of type LOGICAL

Electrical Signal Legend:

- **Green:** Indicates an active node
- **Brown:** Indicates a ground node
- **Grey:** Indicates a removed node
- **Red:** Indicates an isolated node

## Virtual Control Wires

Virtual control wires can be used to provide a visual 'virtual connection' between two or more Data Label components of the same name within a specific module. Virtual control wires appear as a dashed line, which runs directly between corresponding Data Labels as shown below:

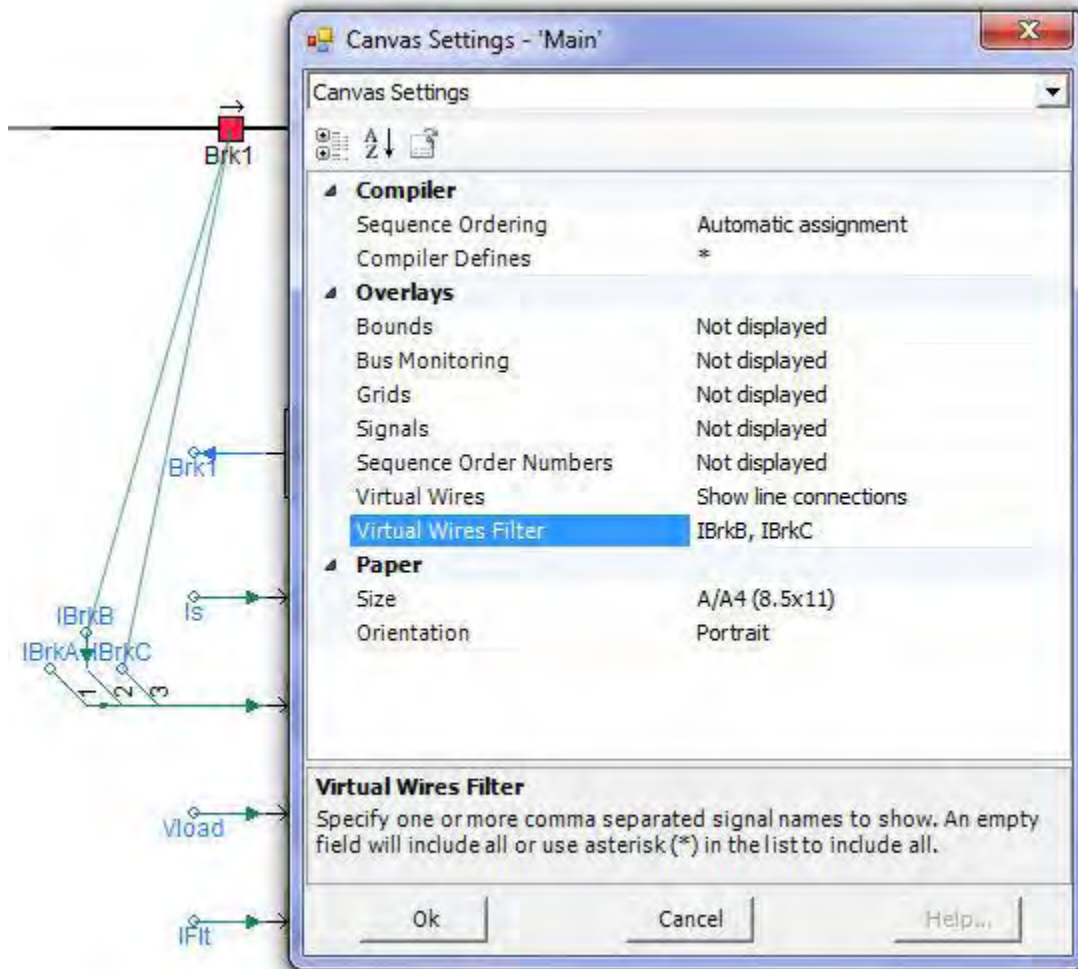


To enable this feature, right-click on a blank part of the module canvas and select **Canvas Settings...** to bring up the Canvas Settings dialog. Select the **Virtual Wires** option.

**NOTE:** Virtual wires are purely visual, that is you cannot use them as physical connections for data sources or sinks. You must compile the case first before you can view the virtual control wires.

## Virtual Wires Filter

You can filter the display of virtual wires based on signal name. This is useful in cases where there is an unruly amount of data connections, making it difficult to sort and find the signals you are interested in. Filter the virtual wires display by simply adding the names of the signals you want to see, comma separated, into the **Virtual Wires Filter** option in the *Canvas Settings* dialog. For example in the image below, the filter is being used to display only the signals **IBrkB** and **IBrkC**.



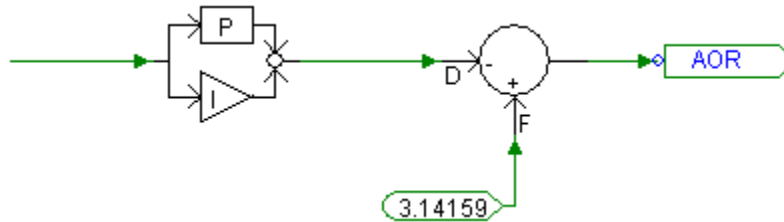
## Colours

The colour of the virtual control wire indicates the data type of the Data Labels it is connecting. The colour legend is provided below:

- **Green:** Indicates a signal of type REAL
- **Blue:** Indicates a signal of type INTEGER
- **Magenta:** Indicates a signal of type LOGICAL

## Control Signal Pathways

Control signal pathways can be used to help visualize the flow of control signals (i.e. from source to sink) following compilation of the project. The indicators will appear on Wire components that are part of a control signal path. The signal flow indicators appear as arrowheads directly on the Wire.



To enable this feature, bring up the *Project Settings* dialog (right-click on the project in the Workspace and select **Project Settings...**). Click the **Dynamics** tab and select **Compute and Display Signal Pathways on Control Wires**.

**NOTE:** The indicators are orientated by default according to the Wire component direction (i.e. towards the end point of the Wire), not the actual control signal flow. If a flow indicator appears reversed, simply reverse the vertexes of the Wire. You must compile the case first before you can view the flow indicators.

## Colours

The indicator colour will appear according to the legend below.

Colour Legend:

- **Green:** Indicates a signal of type REAL
- **Blue:** Indicates a signal of type INTEGER
- **Magenta:** Indicates a signal of type LOGICAL

## Deployment of an Integrated Debugger

Project Options to Preset

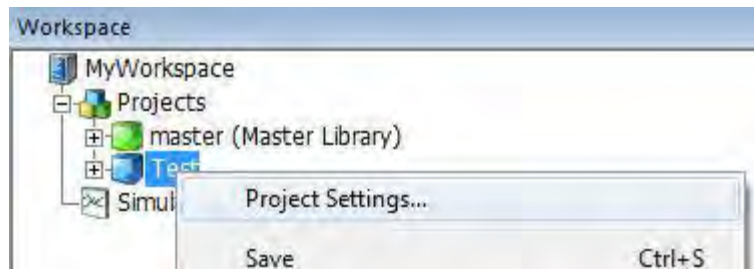
Linking to the Debugger

If you are using one of the supported, commercial Fortran compilers, then it is possible to utilize the integrated debugger application included with the respective compiler. The following sections describe how to preset relevant project settings, as well as step-by-step instructions on how to link EMTDC with your debugger.

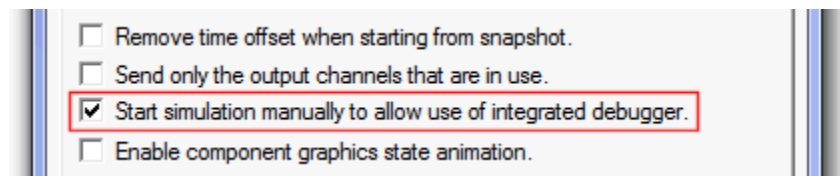
If you are using the free *GFortran* compiler supplied with PSCAD, there are debugging tools available. See <http://sources.redhat.com/insight/index.php>.

## Project Options to Preset

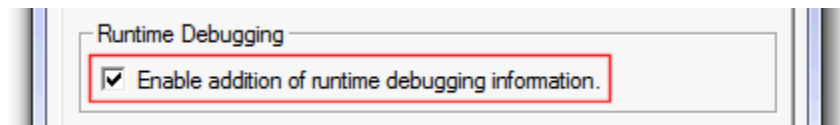
There are a few project settings that must be preset before attempting to proceed – all of which are set in the *Project Settings* dialog. See *Editing Project Settings* for details.



In the Runtime section of the dialog, enable the **Start simulation manually to allow use of integrated debugger** option. Enabling this option allows EMTDC to be started manually.



Next, navigate to the Fortran section of the dialog and enable the **Enable addition of runtime debugging information** option in the *Runtime Debugging* area.

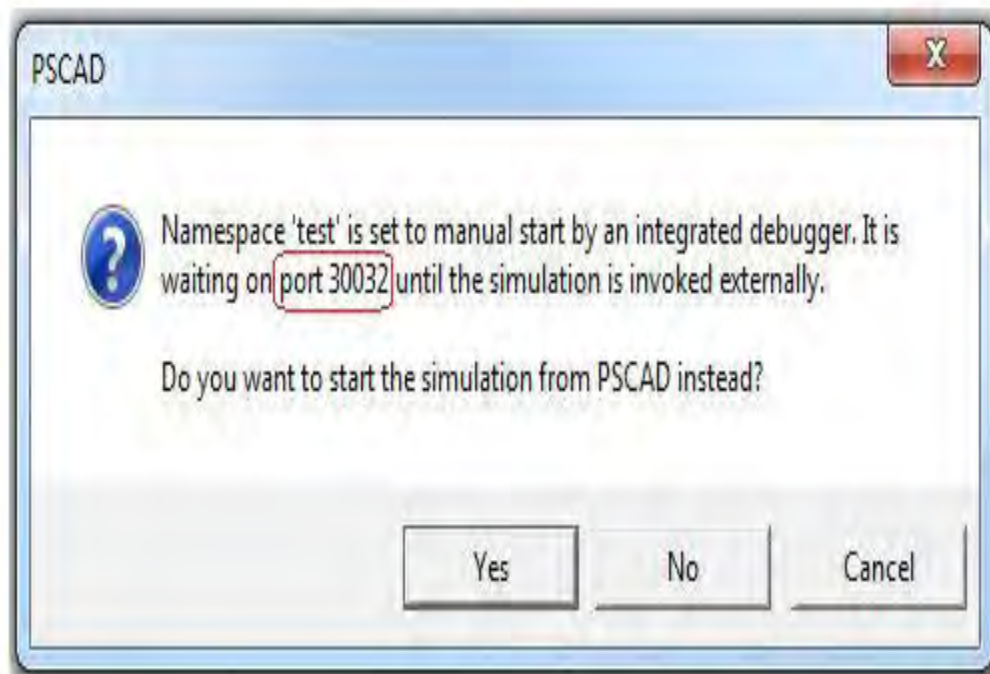


Ensure that any required user source files are referenced in the Additional Source files (\*.f, \*.for, \*.f90, \*.c, \*.cpp) field as well. If you fail to do this, your source files will not be included when the project is compiled, and you will not be able to debug your code.

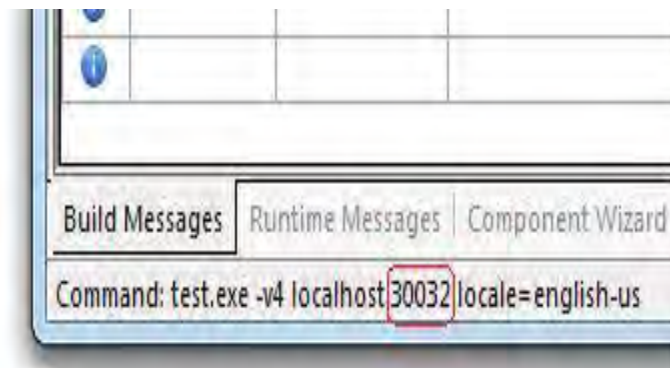
## Linking to the Debugger

The following procedure outlines how to set-up an integrated debugging session for your case project. Note that the following steps are described using the *Intel Fortran 10* compiler (steps for the *Compaq Visual Fortran 6* compiler environment are virtually identical).

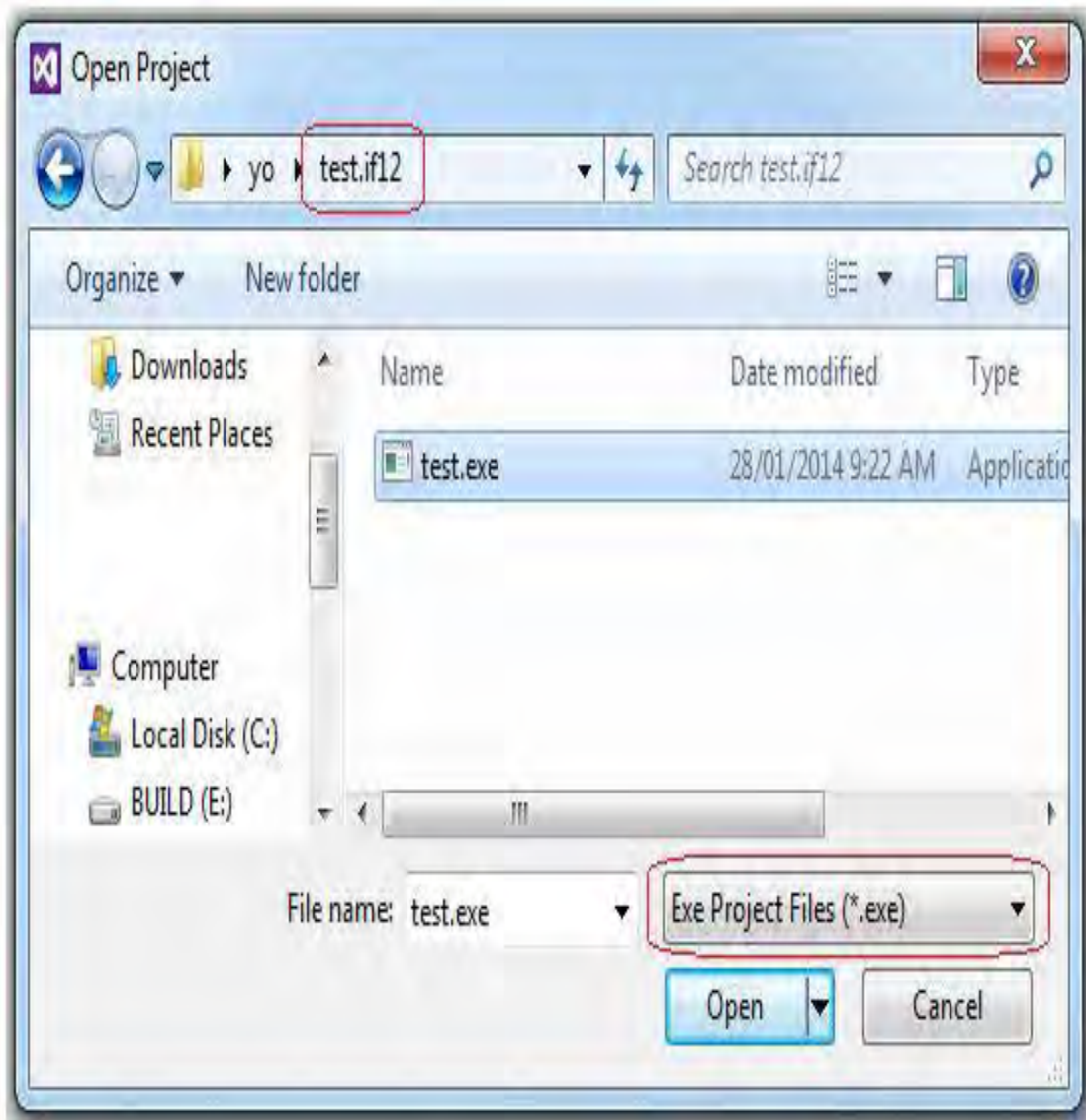
1. Ensure that the project settings described above in Project Options to Preset are enabled. Run the simulation. A pop-up message should appear as follows:



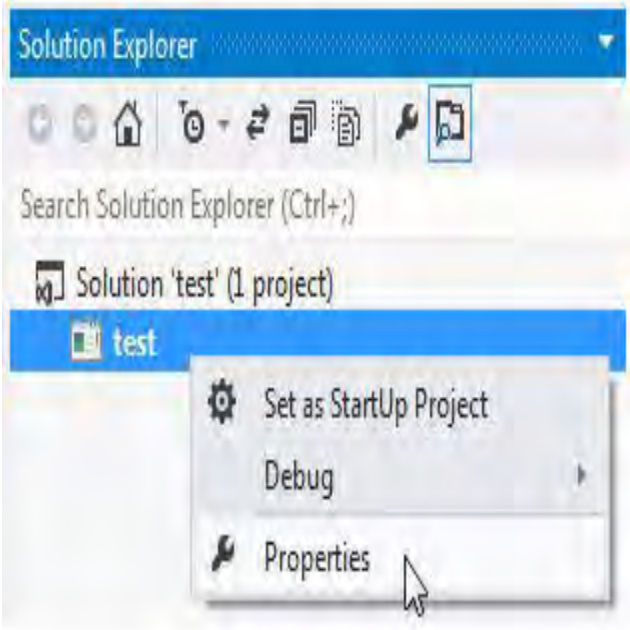
2. Write down the port number on a piece of paper (you will need it in step 4). If you forget to write down the port number, it is also visible at the bottom left corner of the application window.



3. Select the **No** button to continue with debugging.
4. Open *Visual Studio* and select **File | Open | Project/Solution** (for CVF: **File | Open**).
  - a. Change the **Objects of Type** drop list to **Executable Files (\*.exe)**.
  - b. Navigate to the PSCAD Temporary Folder associated with your project file, select the executable file (\*.exe) for your project and then click the **Open** button. For example, if your project is entitled 'test.pscx', then you would select 'test.exe' from the temporary folder as shown below:



5. While still in *Visual Studio*, select **Project | Properties...** (CVF: **Project | Settings...**) to bring up the *Properties* dialog.



Click the **Debugging** branch (CVF: **Debug** tab) and in the **Arguments** field, enter the following:

```
-v4 localhost #####
```

Note that the number (represented by ##### above) must be the same as that in Step 1. Click the **OK** button.

<b>Application</b>	
Executable	C:\Users\georgew\Desktop\yo\test.if12\test.exe
<b>Parameters</b>	
Arguments	-v4 localhost 30032
Attach	No
Connection	Local Debugger

6. While still in *Visual Studio*, open the appropriate Fortran source file (\*.f) by selecting **File | Open | File...** Navigate to and then open a Fortran file generated by PSCAD (ex. *Main.f*).
7. Insert a breakpoint at the point where your subroutine is called in the source code and press the **Start Debugging** button. Manually step in to your subroutine. It is important to note that a breakpoint should not be placed directly within your Fortran source file. This is because PSCAD makes a copy of your source file when the executable is built, and so your original Fortran source is not accessed!

Please note that it is assumed that the user is familiar with the debugging software and can continue from here.

Once debugging is complete and the code is clean, make sure that all debugging options outlined in the Project Options to Preset section are disabled. Failure to do so may affect simulation speed.



## Creating Library (\*.lib) and Object (\*.obj) Files

There are occasions where user-written source code may harbour trade secrets, or simply represent a large corporate investment in development time. In instances such as this, it may be in your best interest to secure this code before it is distributed, especially when models are to be sold or used by clients or partners in joint ventures. Source code can be easily protected by providing it to clients in a pre-compiled (i.e. binary) format. Any source files linked to a PSCAD project are by default compiled into separate object files, by whichever Fortran compiler is being used. PSCAD also provides a utility to efficiently incorporate multiple object files into a single, compiled library file.

Source files are linked to a project by one of two methods:

1. Using the File Reference component
2. The Additional Source (.f) Files field in the *Project Settings* dialog

### Object (\*.obj) Files

Normally, when a source file is linked to a project, the Fortran compiler used to compile the project will automatically create a compiled object (\*.obj) file for each linked source file. This file is placed in the project temporary folder. The user may choose to supply clients with these compiled object files, rather than the source code. This is fine if only one or two source files are involved. Large projects however, can contain many source file links, and supplying an object file for each source file can quickly become cumbersome. One way to circumvent this problem is to merge all source routines into a single file. This process can also be tedious, and may create problems in the on-going development of the source code.

### Static Library (\*.lib) Files

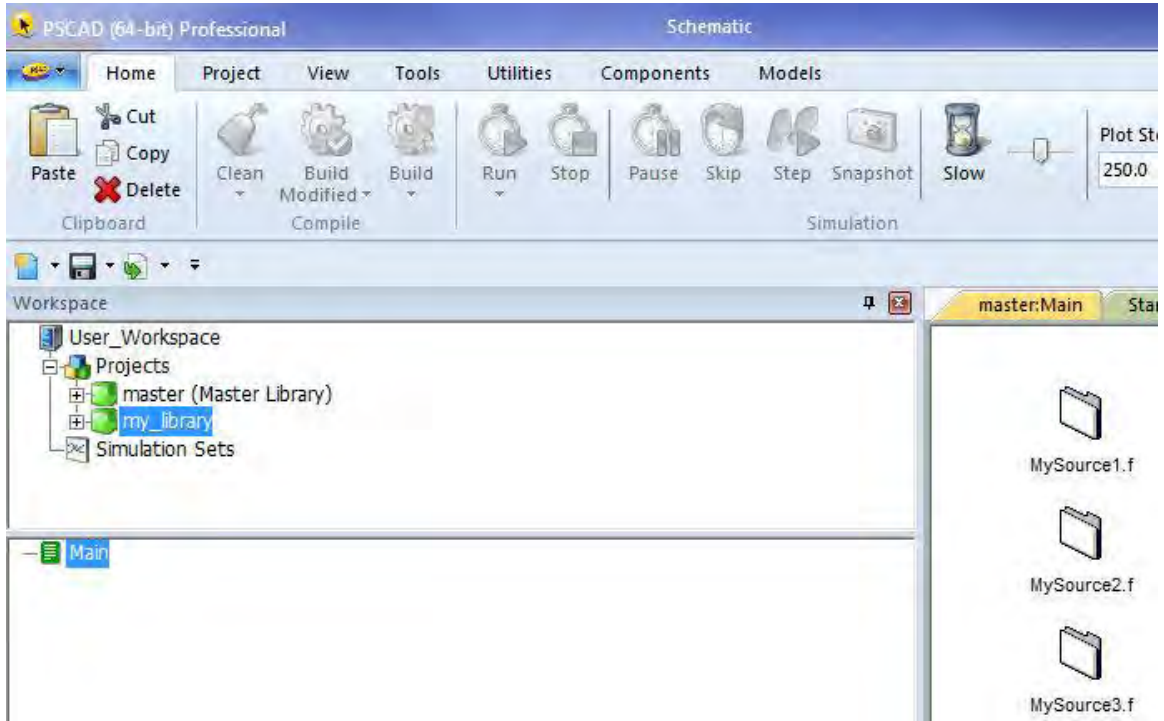
A more efficient means for merging many source files together is to combine all the individual object files into a single compiled library (\*.lib) file. PSCAD provides an easy avenue to create a compiled library file for any library project (\*.pslx) file, provided that links to the source files are set within the library project. In library projects, this may be accomplished through the use of File Reference components.

### Creating a Library (\*.lib) File

The first thing to consider before creating a library file is the Fortran compiler. Each Fortran compiler will create compiled files, which may or may not be compatible for use with other compilers. In other words, it is important to know what type of Fortran compiler the client is using, so that the files provided are compatible. Most PSCAD users will create an equivalent file for each supported Fortran compiler. These files can then be placed in the appropriate directories as described in Additional Library (\*.lib) and Object (\*.obj) Files.

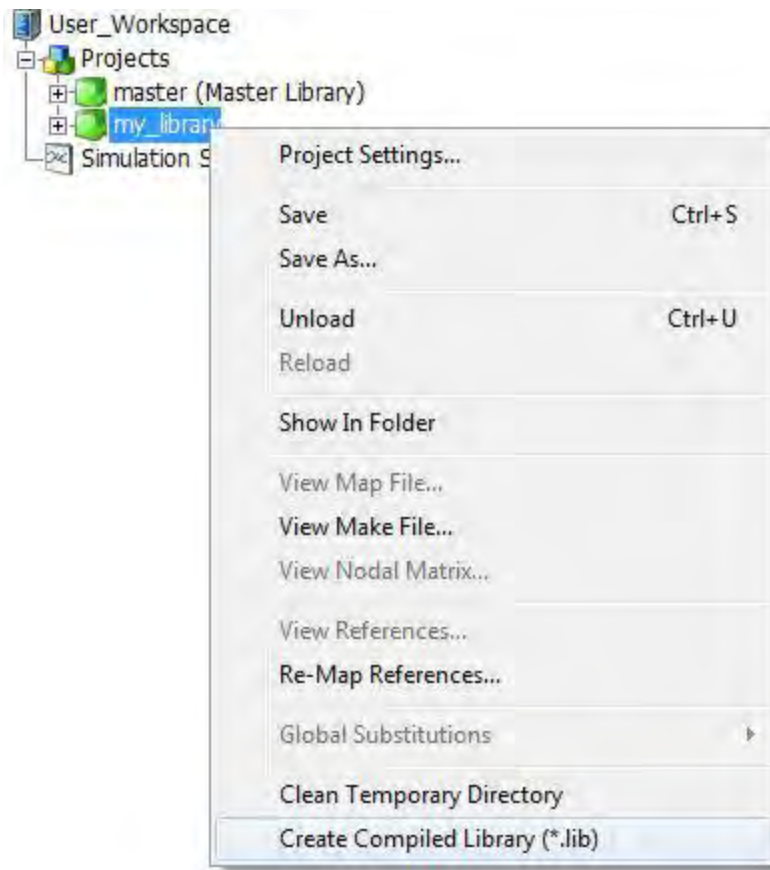
#### Step 1:

Create a new library project as described in Creating a New Project (or edit an existing library), and then link each source file to be included in the compiled library (\*.lib) file using File Reference components.



Step 2:

Right-click on the Library name in the Workspace and select **Create Compiled Library (\*.lib)**:



When the **Create Compiled Library (\*.lib)** function is invoked, PSCAD will create a temporary folder for the library project located in the same directory as the project (\*.pslx) file. In it will be placed the compiled library file (\*.lib), plus an individual object file (\*.obj) for each linked source file.

## Including Dynamic Link Library (\*.dll) Files

It is possible to include *Dynamic Link Library (\*.dll) files* when building projects, although links to these files must be provided by way of an *Import Library (\*.lib) file*. In other words, it is the import library file, which must be directly linked to in the exact manner as is described for linking static library files in *Additional Library (\*.lib) and Object (\*.obj) Files*.

There are however, a couple of extra considerations when linking dynamic link libraries:

- **Location of the \*.dll File:** The dynamic link library file must be placed either in the same folder as the project executable, or preferably in a directory pointed to by a PATH variable. For example, create a directory called `C:\temp\my_dlls` and place your \*.dll files within it. Then add a PATH environment variable with variable value `C:\temp\my_dlls` (i.e. points to this directory).
- **Missing Import Library (\*.lib) File:** If an import library file is not available for the \*.dll file, then you must create one. Instructions detailing the creation of an import library, given a \*.dll file, can be found in Microsoft knowledge base article 131313 (<http://support.microsoft.com/kb/131313>).



## Chapter 11

# MATLAB®/Simulink® Interface

PSCAD provides users with the ability to interface and utilize the functionality of MATLAB commands and toolboxes (including all graphical commands) through a special interface. This is achieved by calling a special subroutine from within a standard component in PSCAD.

Components that interface to MATLAB/Simulink are not offered as part of the Master Library, and must be specially developed for this purpose. In other words, if a specific MATLAB/Simulink component is required, the user must design his or her own to do the job. Once designed however, this component will be treated as a normal component in PSCAD, and may be used interactively with other components in a particular project.

There are a few important items to remember before attempting to interface with MATLAB from PSCAD:

1. **The MATLAB interface is not compatible with the GFortran compiler. One of the supported commercial compilers must be used.**
2. MATLAB must be installed on your computer in order to use the MATLAB interface!
3. PSCAD can be interfaced with library files from MATLAB version 5 or greater.

## MATLAB Interface Subroutine

PSCAD interfaces to MATLAB through a single Fortran subroutine called MLAB\_INT. This routine is included in the main EMTDC library and may therefore be called from any user-defined component. This routine performs the following functions:

- Launches MATLAB through engine using MATLAB Fortran API 'engOpen' commands.
- Changes the working directory to where MATLAB '\*.m' files are located.
- Accesses EMTDC variables from the PSCAD STORF and STORI arrays.
- Converts Fortran variables to C-style pointers and allocates/de-allocates memory locations.
- Uses the MATLAB Fortran API to pass the variables/pointers to the MATLAB engine so they can subsequently be accessed from '\*.m' files.
- Gets MATLAB output variables using MATLAB Fortran API and places them into the STORF and STORI arrays.

## Arguments

```
SUBROUTINE MLAB_INT(MPATH, MFILE, INPUTS, OUTPUTS)
```

### Inputs

Argument	Type	Description
MPATH	CHARACTER	Character string of MATLAB '*.m' file path
MFILE	CHARACTER	Name of module within '*.m' file (the .m extension should not be added)
INPUTS	CHARACTER	Format string for all input variables.

## Outputs

Argument	Type	Description
OUTPUTS	CHARACTER	Format string for all output variables

The formatting for the INPUTS and OUTPUTS variables should be as follows:

- R for REAL type
- I for INTEGER type
- R(dimension) or I(dimension) for array variables
- Ensure that a space is placed between variables

In PSCAD, INPUTS and OUTPUTS variables can be empty strings, in which case the '\*.m' file will run without arguments. This is helpful in initializing the MATLAB environment and designing a component that runs MATLAB '\*.m' files and Simulink '\*.mdl' files simultaneously.

### EXAMPLE 12-1:

A MATLAB module is called by [D] = TEST(A,B,C), where TEST is a module in a MATLAB file 'TEST.m', that is located in C:\TEMP\MLAB\_FILES. The input 'A' is a REAL variable, 'B' is a REAL array of dimension 31 and 'C' is an INTEGER. The output 'D' is a REAL array of dimension 10.

The MATLAB interface subroutine call would then appear as follows:

```
CALL MLAB_INT("C:\TEMP\MLAB_FILES", "TEST", "R R(31) I", "R(10)")
```

### EXAMPLE 12-2:

A MATLAB file entitled 'TEST.m' is located in C:\TEMP\MLAB\_FILES. It consists of MATLAB commands that may take a snapshot of MATLAB results, or initialize the environment (such as setting global variables or changing directory, etc.).

The MATLAB interface subroutine call would then appear as follows:

```
CALL MLAB_INT("C:\TEMP\MLAB_FILES", "TEST", "", "")
```

## Simulink Interface Subroutine

PSCAD interfaces to Simulink through a single Fortran subroutine called 'SIMULINK\_INT'. This routine is included in the main EMTDC library and may therefore be called from any user-defined component. This routine performs the following functions:

- Launches MATLAB through Fortran API functions the same as the MATLAB Interface Subroutine.
- Changes the working directory to where the Simulink '\*.mdl' files are located.
- Accesses EMTDC variables from the PSCAD STORF and STORI arrays.
- Uses the MATLAB Fortran API to pass the variables/pointers to the MATLAB engine so they can subsequently be accessed from '\*.mdl' files.
- Sets the simulation data, specified by the Workspace I/O pane of the Simulation Parameters dialog box, and runs the Simulink module using the MATLAB command 'set\_param'.
- Synchronizes Simulink to the PSCAD. That is, PSCAD proceeds only after the Simulink module simulation is completed each time step, which ensures that the correct results from Simulink are passed to PSCAD.
- Gets Simulink output variables and places them into the EMTDC STORF arrays. This is done in two steps, first the MATLAB command 'get\_param' is employed to get the variable name of Simulink outputs from the Workspace I/O pane in the Simulation Parameters dialog box, then MATLAB Fortran API is utilized to exact and place them into EMTDC STORF arrays.

## Arguments

```
SUBROUTINE SIMULINK_INT(MPATH, MFILE INPUTS)
```

### Inputs

Argument	Type	Description
MPATH	CHARACTER	Character string of MATLAB 'r;*.mdl' file path
MFILE	CHARACTER	Name of module within 'r;*.mdl' file (the *.mdl extension should not be added)
INPUTS	CHARACTER	Format string for all input variables.

The formatting for the INPUTS variable is the same as that in the MATLAB interface subroutine. Note that the OUTPUT of the SIMULINK interface is automatically handled inside the subroutine and is always put into the corresponding EMTDC STORF array.

#### EXAMPLE 12-3:

A Simulink module called 'TEST.mdl', has external inputs A, B and C and is located in C:\TEMP\SIMULINK\_FILES. The input 'A' is a REAL variable, 'B' is a REAL array of dimension 31 and 'C' is an INTEGER.

The Simulink interface subroutine call would then appear as follows:

```
CALL SIMULINK_INT("C:\TEMP\SIMULINK_FILES", "TEST", "R R(31) I")
```

## Designing a MATLAB Component

Designing a MATLAB component involves two simple steps:

1. Create a new component
2. Write a MATLAB file (.m) to perform the required modeling.

### Component Design

Any number of signals or parameters can be passed to or from a MATLAB component. The Fortran code inserted into the Fortran segment of the component definition should perform four tasks:

1. Input variables for the MATLAB function should be transferred to STORF and/or STORI arrays.
2. The MLAB\_INT subroutine must be called with arguments for the MATLAB module and path name, input format string and output format string (more information to follow).
3. Output variables should be transferred from STORF and/or STORI arrays into the PSCAD component output connection nodes.
4. Increment the NSTORF and/or NSTORI index pointers by the total number of variables used.

#### EXAMPLE 12-4:

Consider a simple example for a PSCAD component, which has 2 REAL input connection nodes (A and B), and a single REAL output connection node (C).



The following code should then appear in the Fortran segment of the component definition:

```
#STORAGE REAL:3

    STORF(NSTORF) = $A

    STORF(NSTORF+1) = $B

!

    CALL MLAB_INT("$Path", "$Name", "R R", "R")
```



```

!
      $C = STORF (NSTORF+2)

      NSTORF = NSTORF + 3
!

```

The component definition will need to define at least two input fields in a Parameters section category page. In this case for example, **\$Path** is a text field symbol name expecting the pathname to where the \*.m files are located. **\$Name** is also a text field symbol name expecting the name of the MATLAB module. For example, if the MATLAB function is called 'TEST1', contained within a file called 'TEST1.m', then the **\$Name** parameter should then be entered as 'TEST1'.

General	
Matlab Module Name	TEST1
Relative path of .m Files	C:\matlab\mfiles

More complex input and output arguments can also be used. If an array signal is used, the input or output format string arguments should contain the type and dimension. For example, a REAL array of dimension 31 would appear as R(31), or an INTEGER array of dimension 10 as I(10). Each variable should be separated by one or more spaces, and the order of variables should be identical to the order expected in the MATLAB function.

A good mechanism in Fortran to transfer array variables into or out of the EMTDC STORF or STORI arrays is the DO/ENDDO loop.

#### EXAMPLE 12-5:

Here is an example illustrating a MATLAB component, which has a single REAL input connection array of dimension 31, and a single output connection of dimension 2.

The following code should then appear in the Fortran segment of the component definition:

```

#STORAGE REAL:33
#LOCAL INTEGER I_CNT
!
! First Input Array (REAL(31))
!
      DO I_CNT = 1, 31, 1
          STORF (NSTORF+I_CNT-1) = $INPUT (I_CNT)
      ENDDO

```

```

!
    CALL MLAB_INT("$Path", "$Name", "R(31)", "R(2)")
!
! First Output Array (REAL(2))
!
    DO I_CNT=1, 2, 1
        $OUTPUT(I_CNT) = STORF(NSTORF+31+I_CNT-1)
    ENDDO
!
! Increment STORF pointer
!
    NSTORF = NSTORF + 33
!

```

Note that when the output variables are extracted from the STORF array, the offset 31 must be added, as 31 input variables were already used to put the INPUT variables into the STORF array. In the entire routine, 33 STORF locations were used (31 inputs, 2 outputs).

---

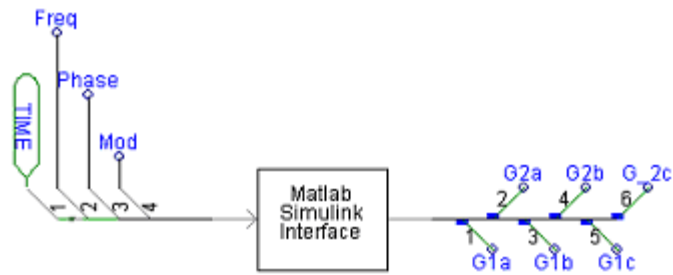
## Designing a MATLAB/Simulink Component

The design steps and principles used in creating a MATLAB/Simulink component, are similar to that explained in the preceding section. Here, an example is provided to illustrate how to design a component that utilizes both the MATLAB and the Simulink interface.

---

### EXAMPLE 12-6:

The figure below illustrates a MATLAB/Simulink component in PSCAD. The input to the component is an array of four variables, named 'TIME', 'Freq', 'Phase', and 'Mag'. The output from the component is an array of six variables as labelled.



The following code appears in the Fortran segment of the component definition:

```

#STORAGE REAL:12
#LOCAL INTEGER I_CNT, M_CNT
!
! PSCAD MATLAB INTERFACE
! MODULE: Matlab Interface with Simulink
!
      I_CNT = 1
      M_CNT = 0
!
! Call *.m files in order to initialize the
! environment...
!
      IF (TIMEZERO) THEN
#IF "$mName1" != ""
          CALL MLAB_INT("$Path", "$mName1", "", "")
#ENDIF
#IF "$mName2" != ""
          CALL MLAB_INT("$Path", "$mName2", "", "")
#ENDIF
      ENDIF
!

```

```

! TIME info to run either *.m or *.mdl module
!
#IF ($OPTSEC == 0 )
    STORF(NSTORF) = TIME
    STORF(NSTORF + 1) = DELT
    M_CNT = 2
#ELSE
    STORF(NSTORF) = TIME
    M_CNT = 1
#ENDIF
!
! Transfer inputs to EMTDC STORF array
!
    DO I_CNT = 1, $#DIM(sig_in)
        STORF(NSTORF + M_CNT + I_CNT-1) = $sig_in(I_CNT)
    END DO
    M_CNT = M_CNT + I_CNT - 1
!
! CALL PSCAD MATLAB INTERFACE
!
#IF $OPTSEC == 0
    CALL MLAB_INT("$Path","$mfile","R(6)","R($#DIM(sig_out))")
#ELSE
    CALL SIMULINK_INT("$Path","$simfile","R(5)")
#ENDIF
!
! Transfer MATLAB output variables
!
    I_CNT = 1
    DO WHILE (I_CNT .LE. $#DIM(sig_out))

```

```

    $sig_out(I_CNT) = STORF(NSTORF + M_CNT + I_CNT - 1)

    I_CNT = I_CNT + 1

    END DO

!
! Update storage array
!

    NSTORF = NSTORF + M_CNT + I_CNT - 1

!
```

Various parameters category pages within the component definition define the variables used in the above code. This category page would appear similar to that shown below:



Each of the input fields above defines a variable (preceded by a \$ symbol) in the component code.

## Interfacing Notes

The MATLAB engine performs operations very slowly, compared with the same equivalent operation hard-coded directly into a PSCAD component. The MATLAB source code is interpreted each time it is called, allowing users to dynamically edit the \*.m file in the middle of a run and see its effect immediately. This inter-activity is also possible in PSCAD through the use of on-line sliders, switches, dials and buttons. Any combination of the two methods can be used simultaneously.

## Alternative Simulink Interface

There is an alternative way to invoke a Simulink module. Instead of calling the 'SIMULINK\_INT' subroutine, users may call 'MLAB\_INT' subroutine to invoke an '\*.m' file, which uses the MATLAB command 'sim' to handle the Simulink module. However, the use of 'SIMULINK\_INT' is highly recommended due to the synchronization mechanism between MATLAB and PSCAD implemented within this subroutine. This is especially true for Simulink modules that run longer than the time step defined inside EMTDC.

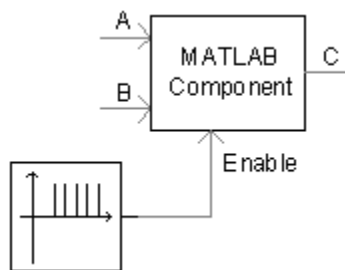
## Simulation Speed

To try and speed up the MATLAB solution, it is often a good idea to try and use a larger time step when invoking MATLAB components (wherever possible or practical). An enable/disable switch can also be implemented, so as to allow PSCAD to operate at close to full speed.

---

### EXAMPLE 12-7:

The figure below illustrates one way to speed up your PSCAD/MATLAB simulation. Here, an impulse train is applied to an additional enable/disable input to control how often the MATLAB solution engine is invoked.



The Impulse Train frequency can be varied for optimal speed/accuracy considerations.

---

## Conversion to C

It is possible to convert '\*.m' source code directly to 'C' code using the MATLAB Coder, and then directly compile and link the 'C' source code into the EMTDC executable. With this 'hard-compiled' approach, you lose the ability to edit the MATLAB '\*.m' interpreted file during simulations. The 'hard-compiled' approach also may not work with any MATLAB graphical functions.

A good compromise can be reached by first using MATLAB components to develop and test algorithms, but in the end to optimize the speed of the final design by hard-coding the algorithms in Fortran or C (either manually or using the MATLAB 'C' compiler). The final hard-coded algorithms are linked directly with the EMTDC solution engine and are very fast (as they can be optimized using modern compilers).

## Plotting Enhancements

The MATLAB graphics functions are a very powerful addition to the PSCAD plots and graphical interface. Three-dimensional plots, active graphics and rotating images are possible and integrate seamlessly with the PSCAD graphical libraries.

## Enabling and Using the Interface

In order to make use of the MATLAB®/Simulink® interface in PSCAD, you must first set-up the MATLAB® version you are using (as well as the path to the installation library directory if you are using MATLAB® v5). To do this, go to the *Application Options* dialog under the Dependencies section.

Once the MATLAB® interface particulars are set-up, you must then enable the interface in your case Project Settings under the Link tab.

**NOTE:** You may find discontinuations when starting from a snapshot file if you use complex models in MATLAB®/Simulink®. If this is the case, it can be prevented by adding extra code in the \*.m or \*.mdl file and the component definition calling the interface.





## Chapter 12

# Migrating from Older Versions

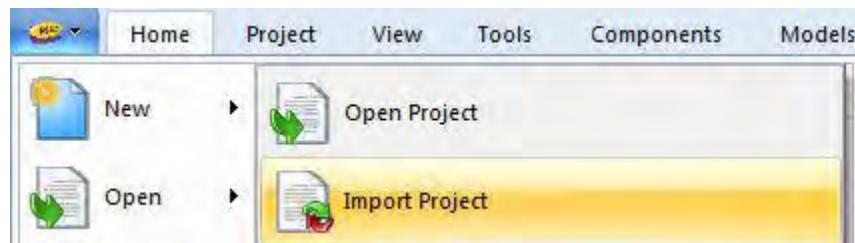
The PSCAD X4 release represents a vast transformation in program architecture. It is also a prudent point at which to continue unfettered by old operating systems, Fortran compilers, and older PSCAD versions. This latest version allows you to import \*.psl and \*.psc format project files that were last saved in PSCAD v4.2.1 only, and it is not backwards compatible: In other words, once your projects have been migrated into X4, they cannot be exported back to the older formats.

**NOTE:** If you are a PSCAD V3 user, and you want to import your projects into X4, you must first import, test and save your projects in PSCAD v4.2.1. PSCAD V2 users will of course need to do the same.

This chapter discusses topics regarding the migration of projects from older versions into the latest PSCAD version. For more information on migrating user-defined code and other EMTDC related issues, see the section entitled Converting V2 Fortran Files in Chapter 10 of the EMTDC Manual.

## Importing PSCAD V4 Projects to X4

Importing a V4 project (v4.2.1 only!) into PSCAD X4 is for the most part, as simple as a couple of mouse clicks. Simply select the PSCAD tab in the ribbon control bar and select **Open | Import Project**, to import a v4.2.1 project.



Upon import, PSCAD will create a new file and convert your original project to the new file format. Your original project will be left untouched. Project files are converted as follows:

Old File Format	New File Format
<project_name>.psc	<project_name>.pscx
<project_name>.psl	<project_name>.pslx

As mentioned above, it is not possible to save back to the older format, so any changes made to the new project files cannot be migrated back. If you need to maintain files in both formats, then you must make equivalent changes in the old format and the new separately.

If all goes well, the above procedure is all that is needed to upgrade your PSCAD V4 projects to X4. During the import process however, there may have been compatibility issues in migrating your project over to the new format. These issues may not arise until you attempt to compile and run the new project.

The following sections attempt to describe most of the issues that you may experience with a newly imported project, as well as other important topics you should be aware of.

## Duplicate Definition Linking Priority

Every project, be it a case or a library, contains component or module instances. The definitions, on which these instances are based, may or may not be stored in the project. They may in fact be stored in previously loaded, user library projects, the master library, or a combination of both. There may be situations where duplicate definitions exist in the workspace (i.e. definition sharing the exact same name).

So, how do we choose a definition if there are duplicates? To avoid chaos on import, there must be certain rules imposed. In previous versions of PSCAD, the definition linking mechanism was given a set of priorities: Any definition residing in the master library was given first priority; then previously loaded user-defined library projects (i.e. \*.psl files); then the local project itself. For example, say a user has a custom definition called *resistor*, which is stored in a case project. When the case is loaded into the workspace, PSCAD v4.2.1 would link any instances of the *resistor* in the case project to the master library definition of the same name, not the local *resistor* definition.

A fundamental change was made in PSCAD X4 effectively reversing the older priority set. Now the local definition is given priority, then user library definitions, then the master library, upon load or import of a project.

In v4.2.1	New in X4
Master Library	Local Project
Other Library Projects	Master Library
Local Project	Other Library Projects

Definition Linking Priority on Import/Load Project

## Namespace Linking

Note that the above linking priority is imposed only when projects are first imported. This is because component instances were linked to their definition by name only; no information was given on where the definition resides. As such, PSCAD needed to rely on the linking priority in order to 'best guess' which definition to link to.

Old Definition Name Format	New Definition Name Format
<definition>	<namespace:definition>

A new concept called *Namespace Linking* was introduced in PSCAD X4, which provides project information (or scope) to each definition. Once your project has been saved in X4 format, all definition links will possess both a definition name and an namespace name. In this manner, PSCAD no longer needs to guess where the definition is: If the corresponding project is not loaded, then the component looking for a definition in that project simply appears as a placard.

With the namespace concept comes a variety of other convenient features that allow users to link and re-link their components from one definition to another. See Definition Referencing for more details on this.

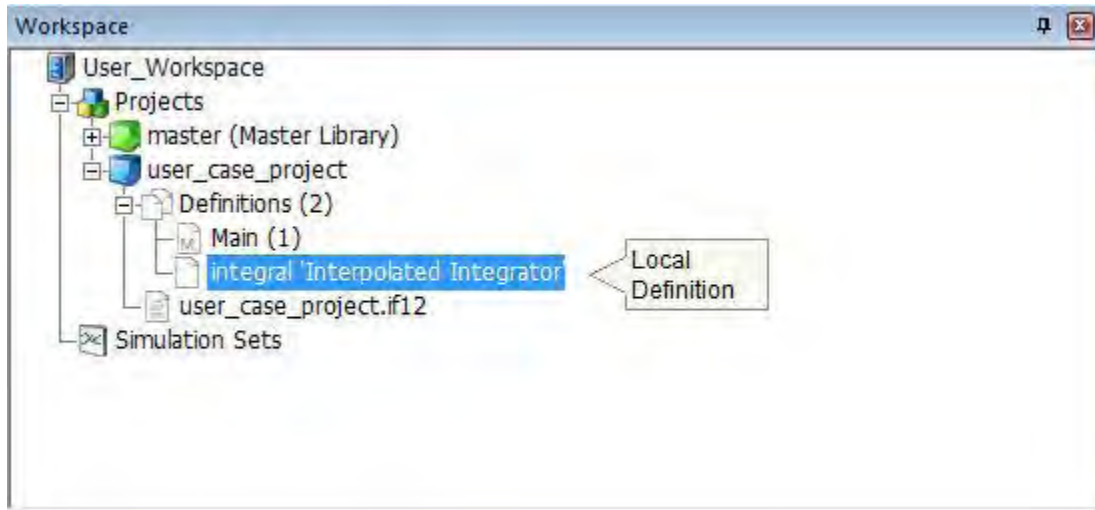
## Problems to be Aware of

If you have duplicate definitions in your workspace, you may experience problems when compiling imported project files. These problems may manifest themselves in many ways, but the most common problem arises when your project contains a local definition (or many) that shares the same name as a definition residing externally in either a user library project, or the master library.

## EXAMPLE 13-1:

A user possesses a case project that contains an unused definition (i.e. it does not have any instances) called *integral*.

In PSCAD v4.2.1, this definition would always be ignored, as this is also the name of a master library definition, and the linking priority was master library first; so the master library *integral* definition would always be linked to the instances in the case. If this case project is imported into PSCAD X4 however, the local *integral* definition will by default be linked to all instances of *integral* in your project.



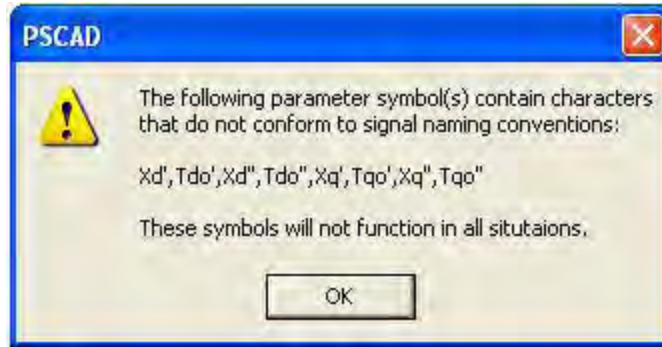
This will most likely result in compilation errors and other problems. Or, it may even go undetected!

These problems are best dealt with before a project is imported into PSCAD X4. The following is a checklist of tasks that should be performed in PSCAD v4.2.1 prior to importing your project:

- **Unused Definitions:** Ensure that all unused definitions stored in your project are either deleted, or have unique names before import. This will ensure that the PSCAD importer will initially link your instances to the proper definition (i.e. external), and not to any local definitions that share that name.
- **Duplicate Definitions:** To avoid linking priority problems, all duplicate definitions stored in either user library projects, or in any case projects should be given unique names, or redundant duplicates deleted. If this task is performed prior to import, the PSCAD X4 importer will not initially link instances to the wrong definition.

## Illegal Character Issues

There is an assortment of characters that are not allowed as part of any XML content, as they are used by the language as delimiting characters. These characters were initially deprecated in PSCAD v4.0 (2003), but have been functioning in 'compatibility mode' until now. In previous versions, editing a component definition with input parameter *Symbol* names containing any of these characters would result in a similar dialog to the following:



Upon import of existing projects into X4 (i.e. \*.psc and \*.psl files), these characters will be replaced automatically (when used in certain situations), as their presence is detrimental to the structure of the project file under the new XML standard.

Illegal characters and their replacements are listed below:

Character	Name	Replaced By
&	Ampersand	–
< >	Inequality Signs	–
"	Quotation Mark	–
'	Apostrophe	–
°	Degree	deg

The following is a list of areas where these characters will be replaced if found by the importer:

- Project Description

Component Definitions:

- Category Page Names

Component Input Parameters:

- Descriptions
- Symbol Names
- Default Values
- Default Units

Component Instance Input Parameters:

- Parameter Values

## Modification of Script

Modification of component script has been provided as an adjustable application option.

## EXAMPLE 13-2:

A PSCAD v4.2 project is to be imported into X4, and contains a user-defined component definition. The definition contains a parameter with *Symbol* name *Xd''*. Upon import of the project file, the X4 importer detects the illegal characters in the *Xd''* parameter and replaces them with underscore characters (as described above). As a result, the parameter *Symbol* name will be *Xd\_\_* following import of the project file.

Original Symbol Name	New Symbol Name
<i>Xd''</i>	<i>Xd__</i>

The importer does not however, modify the component definition script sections. As a result, if this parameter *Symbol* name is used in any substitutions in the script, compile errors will occur. The Checks section of this component is defined as follows:

```
ERROR The quantity Xs cannot be greater than Xd : (Xs<Xd'')
```

Since the *Xd''* symbol name was not replaced, if left as is PSCAD will issue a build error message similar to the following upon compile of the project:

```
Checks Script Error: Evaluate: (Xs<Xd'')[Error]; Evaluate: 'Xd'' is not a parameter, computation or constant.
```

The user must therefore modify the script section (in this case the Checks segment) as follows:

```
ERROR The quantity Xs cannot be greater than Xd : (Xs<Xd__)
```

## Helpful Legacy Import Tools

There are a few convenient tools available to you when importing or loading your legacy (older) projects:

- Obsolete 'datatap2' components
- Parameter Synchronization
- Obsolete #DEFINE directives
- Orphaned Global Substitutions

All of these tools are application options. See Workspace for a description of each.

## Adding Multiple Instance Module Support to User-Defined Components

With the inclusion of *Multiple Instance Modules (or MIM)* in PSCAD X4, changes to EMTDC system dynamics functionality were unavoidable. This means that some user-defined components may need to be modified to support their usage within a module with multiple instances. Components that support usage within multiple instance modules are referred to as *Runtime Configurable*.

PSCAD Manual:

- Operations and Feature Overview: Multiple Instance Modules (MIM)
- Component Design: The Parameters Section | Input Fields

EMTDC Manual:

- Program Structure: System Dynamics (various sections)
- Custom Model Design (various sections)

## Other New Features

For a complete list of all that is new in PSCAD X4, please see the What's New? topic in the online help (In the online help, go to Opening Screen and click the *What's New?* link).

# Converting PSCAD V3 Projects to V4

Conversion Issues

Loading PSCAD V3 projects into PSCAD V4 is fairly straightforward, and for the most part will be accomplished without incident. Simply load the V3 case or library project exactly as you would a V4 project.

## Conversion Issues

As is normal when software is updated, there are changes made to existing features, which may or may not affect your case. There are a few important issues to be aware of before starting to use your V3 projects in PSCAD V4. These are described in the following sections.

### System Dynamics Component Ordering

Components used in the EMTDC System Dynamics (i.e. CSMF components, modules, etc.) are automatically ordered in V4 with a sophisticated new sequencing algorithm. In PSCAD V3, these types of components were ordered using a simpler method.

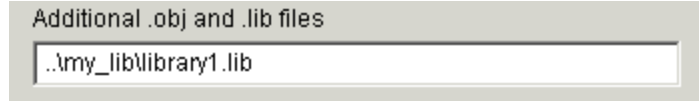
This important feature should be considered when initially verifying your V3 project results in PSCAD V4. The new sequencing algorithm may adjust the order in which components appear in the Fortran code. Depending on where the components within the EMTDC System Dynamics (i.e. DSDYN or DSOUT), a single time step delay may be added or removed in comparison with V3 results.

If this does indeed occur in your case, PSCAD V4 allows you to manually adjust the component sequence. Please see the section entitled Component Ordering in Chapter 11 of this manual for more details.

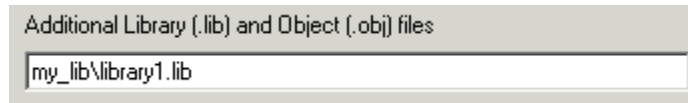
### Additional Libraries and Object Files

In PSCAD V3, additional library and object file paths entered in the *Additional .obj and .lib Files* field in the Project Settings dialog, were relative to the respective temporary (\*.emt) directory for the project. In PSCAD V4, these paths are now relative to the *User Library Path* (set in the Workspace Settings dialog window), or can be entered directly as absolute paths.

These paths must be changed accordingly in your V4 projects. Open the respective Project Settings dialog for the project (right-click on the project filename in the Workspace and select *Project Settings...*) and select the *Link* tab. Modify the paths in the Additional Fortran Library (\*.lib) and Object (\*.obj) Files input field. For example, a V3 reference is shown below:



would appear as follows in PSCAD V4:



## Flyby Windows

A new feature was added to PSCAD V4, which optimizes variable storage between time steps during a simulation. Unfortunately, when this optimization algorithm is enabled, it also disables Flyby window functionality. To enable Flyby windows while debugging your project, the optimize storage feature must be turned off.

Right-click on the project filename in the Workspace window select *Project Settings...* Click the Dynamics tab and select the option called *Store Feed-Forward Signals for Viewing*.

## Node Loop Component Output Format

The output format of the Node Loop component was altered to reflect changes made to the subsystem splitting algorithm in PSCAD V4. This becomes important if your PSCAD V3 user components utilize the Node Loop for input. If this is the case, then you must alter your components before running any V4 projects.

In PSCAD V3, the subsystem global variable (SS) always retained the same value within the same circuit module. In PSCAD V4, the components on a single circuit can reside in different subsystems. As such, a new *Loop* prefix has been added to tell the compiler to use the subsystem number of the Node Loop component when processing:

V3 format (single subsystem):

```
$SS, $Loop:NA, $Loop:NB, $Loop:NC
```

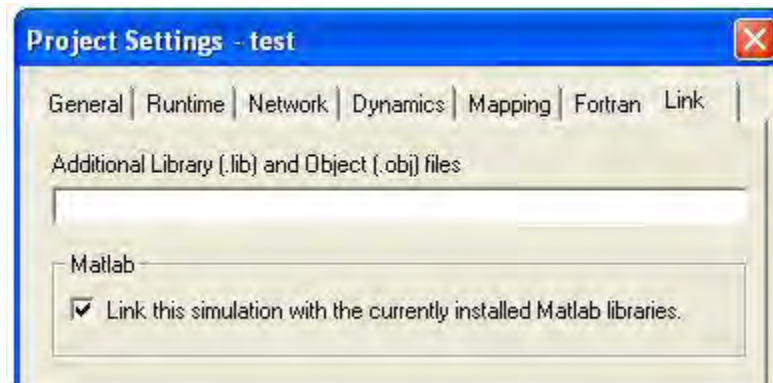
V4 format (multi-subsystem):

```
$Loop:SS, $Loop:NA, $Loop:NB, $Loop:NC
```

## MATLAB Libraries and Interface

Project and Workspace settings should be adjusted slightly when migrated V3 projects that used the MATLAB interface. Please perform the following adjustments:

1. Open the respective *Project Settings* dialog for the project (right-click on the project filename in the Workspace window and select *Project Settings...*) and select the *Link* tab. Delete all of the MATLAB installation library paths in the *Additional Library (\*.lib) and Object (\*.obj) Files* input field. Ensure that the *Link this Simulation with the Currently Installed Matlab Libraries* check box is selected.



2. Open the Workspace Settings dialog by selecting *Edit | Workspace Settings...* and click the *Matlab* tab. In the Interface Settings area, select the *Installed Version* and *Library Path* to the MATLAB installation libraries.



## Converting PSCAD V2 Projects to V4

PSCAD V2 projects can be directly migrated into V4 without too much difficulty. This of course mainly depends on the complexity of the V2 project, the components used and the condition of any user-written code involved. Unlike V2, PSCAD V4 stores all information pertaining to a project into a single portable project file (either a case project (\*.psc) or a library project (\*.psl) file).

Some V2 files cannot be directly imported into PSCAD V4. The following is a list of which file types can, and which file types cannot be migrated:

Can be migrated:

- Draft circuit files (\*.dft and \*.dfx)
- Runtime batch files (\*.rtb)
- Draft component definitions (for example in xdraft\_lib)



- EMTDC Fortran source code
- Line constants solved data (\*.tlb or tlines files)

**NOTE:** If you have EMTDC source code written in C, then you must upgrade this code manually.

Cannot be migrated:

- Cable files (\*.cbl)
- Multiplot or Uniplot batch files

The following topics describe the procedures involved when importing PSCAD V2 circuits, components and associated files into PSCAD V4.

## User-Written EMTDC Source Code

Fortran source code, written specifically for use with EMTDC in PSCAD V2, must be filtered before it can be used in PSCAD V4. For more information on importing user-written Fortran source code, see the section entitled Converting V2 Fortran Files in Chapter 10 of the EMTDC Manual.

## Conversion Issues

V2 Control Type Components

V2 Electrical Interface Components

V2 Component Libraries

Component Definitions

The following sections describe a general overview and background of some important conversion issues. Please review these sections first before attempting to upgrade to PSCAD V4.

### V2 Control Type Components

All control type components (i.e. components whose code appears in the EMTDC System Dynamics only) should be upgradable. One important fact to consider however is that using the EMTDC internal variable for time step (i.e. DELT) in the Computations portion of a component definition is no longer allowed in PSCAD V4.

### V2 Electrical Interface Components

There have been many changes made to EMTDC since V2, mostly associated with electrical signals and their interface to user-written components. It is possible that these types of components may cause some compatibility problems when upgrading to V4. For instance, electrical branches in EMTDC were originally referenced using a branch TO and FROM node convention. This tended to cause problems with parallel branches (as they have identical connection nodes), and resulted in numerous workarounds to avoid the problem. One example was the output of the calculated current in faults, breakers, thyristors, diodes, GTOs, arresters, etc. - all of these had a single time step delay. This was because all parallel switching branches were combined into a single branch for solution in the main program. One side effect was that the current had to be output as an argument of the DSDYN subroutine call, and could not be placed in DSOUT.

In PSCAD V4, the above problem is circumvented, as each branch is given a unique branch number and the current in each branch is referenced directly in DSOUT. This system of referencing electrical branches directly (by a branch number instead of by node numbers) results in some obsolete function calls. These include all switching routines, as well as any electrical interface array names.

## V2 Component Libraries

Since V2, PSCAD has allowed users to create their own components to represent custom models. In V2, this was accomplished by editing a text file, where graphical information, parameter form data, and Data/Fortran output code were entered. Each component was contained within its own file, which resided in either the user's library (~\PSCAD\xdraft\_lib), or in a Group Library (only 2 libraries were allowed).

Those who received a project from other users always experienced difficulties in maintaining these components and great confusion (as to what the most recent version of the component was), generally resulted. If two or more components by the same name were kept, then the PSCAD V2 Draft program would search first in the User's xdraft\_lib directory, then in the Group directory (if one has been specified), and then finally in the Master library. This allowed user's to over-write the functionality of Master Library components, often with confusing or inconsistent results.

In PSCAD V4, the Design Editor is used to graphically manipulate custom components. Component definitions are stored within a single library project file (\*.psl), where any number of library projects can be loaded in the Workspace simultaneously. Users who wish to transfer custom components to other users need only to supply the library project containing the components.

When PSCAD V4 loads a project, it knows not only the name of the component, but that it came from a library project. This allows users to copy a component definition from the any other library, customize it, and then place it in their own library. Whenever either of these components is used, PSCAD will keep track of which library each is from. PSCAD V4 also allows components to be kept directly in case projects (\*.psc), so that a temporary component can be developed for a specific case, without having to clog up a library with the test code.

## Component Definitions

If a required component definition cannot be found when a V2 draft project is loaded into PSCAD V4 (i.e. the definition is not included in any loaded V4 library project or the specific V2 draft project), the draft project will still load successfully. However, any component instances based on the missing definition will be displayed with a 'placeholder' component. To rectify this situation, ensure that a V4 library project already containing the required component definition is loaded before the draft project, or that draft project itself was saved in V2 as a transfer file (\*.dfx).

The PSCAD V4 Master library is automatically loaded when PSCAD V4 is started. It contains many new components, in addition to all PSCAD V2 components from the following libraries:

- PSCAD V2 Master library
- PSCAD V2 PEMISC library
- PSCAD V2 Machines library

Note that it is important to always use unique names for user-defined component definitions. This will avoid unwanted interactions between user and Master Library component definitions.

## Importing V2 Draft and Runtime Batch Files

Most PSCAD V2 draft projects can be directly imported into PSCAD V4. Projects containing more complex circuits however, may require some manual adjustments. This section describes procedures for importing PSCAD V2 draft files (\*.dft or .dfx) and runtime batch files (\*.rtb) into PSCAD V4.

Before attempting to import your PSCAD V2 projects, please ensure the following:

- All obsolete PSCAD V2 components are replaced with the latest versions from the most current V2 Master Library.
- All PSCAD V2 files associated with the draft project itself (i.e. \*.rtb, \*.tlb, and tline\_out) must be located in the same directory as the draft file before the draft file is imported into PSCAD V4.

**NOTE:** PSCAD V4 will initially allow the import of only a single runtime batch (\*.rtb) file, associated with a specific V2 draft (\*.dft or \*.dfx) file. This \*.rtb file must have the identical filename as the draft file being imported. For more information on importing any remaining Runtime batch files, see the section entitled Importing Additional Runtime Batch Files below.

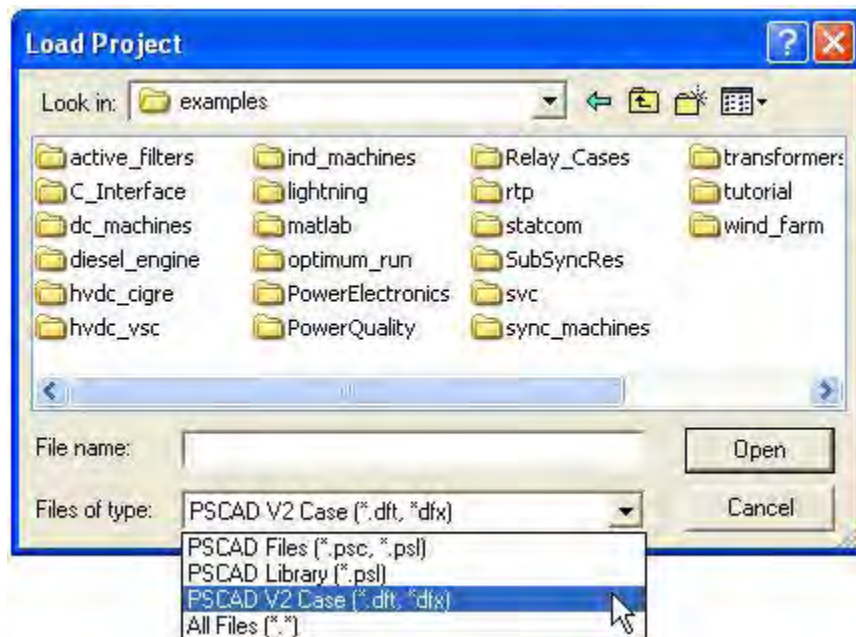
- If the draft project contains custom written components (i.e. from user or group libraries), then ensure that the draft file has been saved as transfer (i.e. \*.dfx file) in PSCAD V2. Otherwise, see Importing V2 User Libraries or Importing Individual V2 Components below for more details.

**NOTE:** If a V2 draft project containing undefined components is loaded into PSCAD V4, all undefined components will be substituted with a temporary 'placeholder' component.

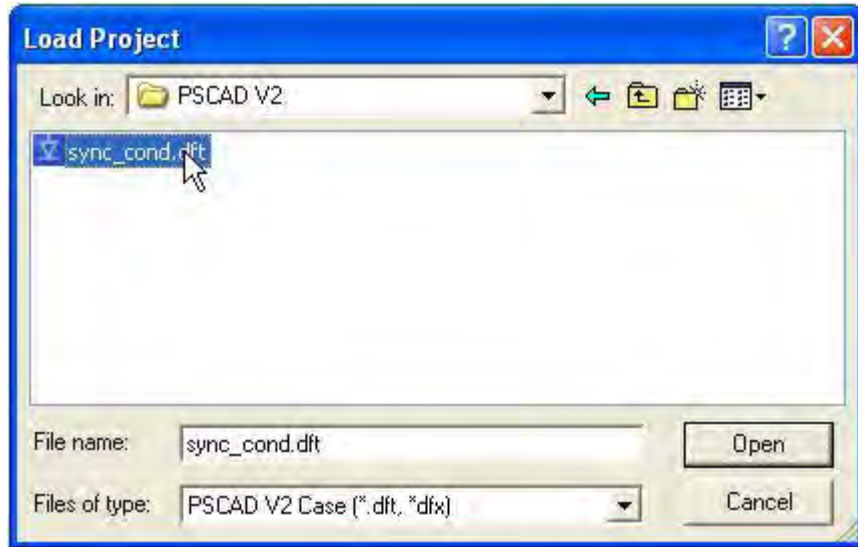
Please follow these steps to avoid problems:

1. To load a V2 draft file, start PSCAD V4 and select *File | Load Project...* from the Main Menu bar. The Load Project dialog window should appear.
2. Near the bottom of the Load Project dialog, change the *Files of Type* drop list to *PSCAD V2 Case (\*.dft, \*.dfx)* so as to view PSCAD draft files only. If you do not have a direct network connection from your PC to your UNIX system, then you must first copy (or ftp) the files to your PC from UNIX.

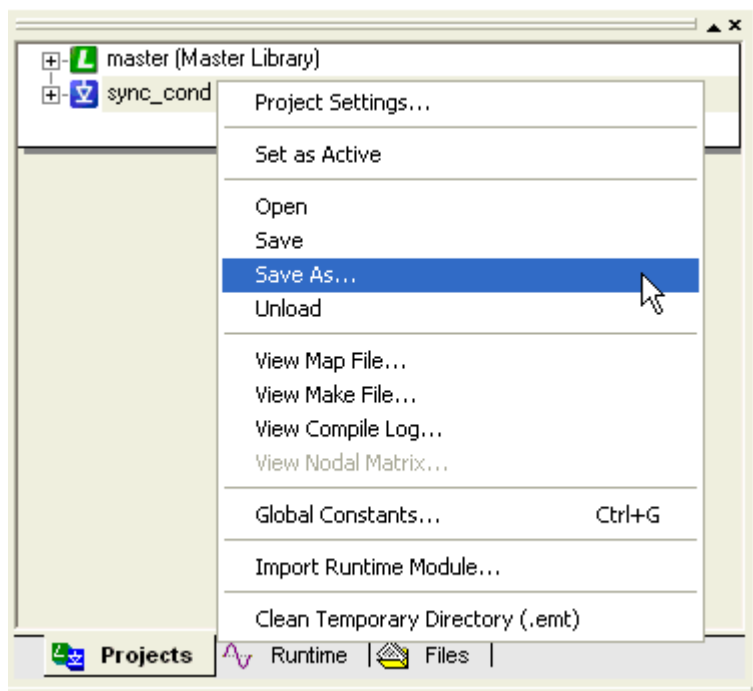
**NOTE:** Note your V2 files may still be in UNIX format. In order to successfully import the file to PSCAD V4, you must ensure that all files are in DOS format. This can be accomplished by using the 'unix2dos' UNIX utility, or FTP in ASCII mode.



3. Navigate to the directory containing your V2 project files, select the V2 draft file to be loaded, and then press the *Open* button.



4. Check for warning or error messages in the Output window - see Common Warning Messages below for details. Rename and save the new PSCAD V4 case project in V4 format (right-click on the project name in the Workspace window and select Save As...).



## Common Warning and Error Messages

There are some common warnings and error messages that may appear in the Output window following the migration of a V2 draft file. Some of these are described below:

Warnings:

- **Unresolved keyword 'xxxxx':** This warning indicates that data in a PSCAD V2 component was stored, but is not required in the PSCAD V4 component. This often occurs in components, which can be used in both EMTDC and RTDS, where some of the data is not required for EMTDC runs.

- **Component aliased from 'xxx' to 'yyy':** This warning indicates that a component has been renamed in PSCAD V4, but the data from the old V2 component has been converted.

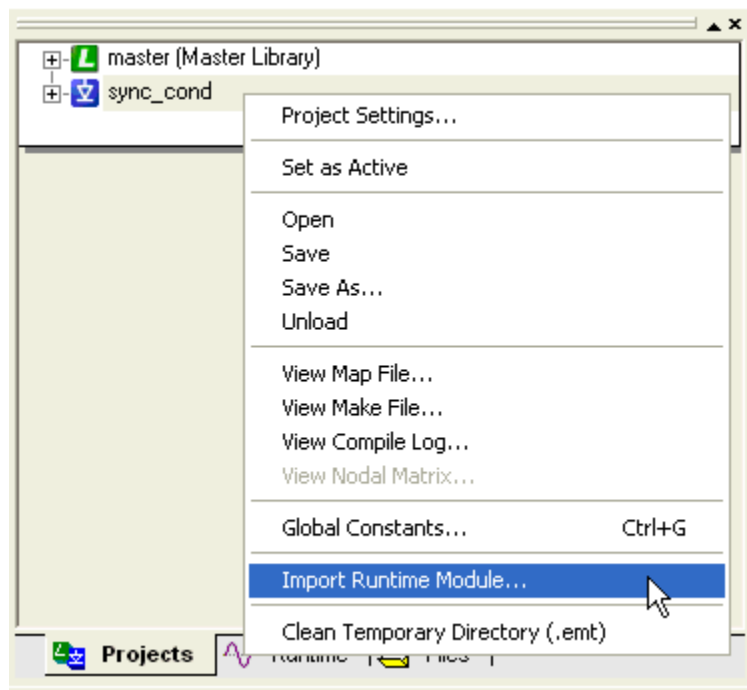
Errors:

- **Component Definition 'xxx' not resolved:** This means that a custom component was used in PSCAD V2 but was not available in any of the library projects loaded in the PSCAD V4 Workspace. If you have previously converted your PSCAD V2 libraries into PSCAD V4 library projects, then simply ensure that the libraries are loaded in PSCAD V4 first before converting the PSCAD V2 draft project. If you have not yet converted PSCAD V2 libraries into PSCAD V4 library projects, then follow the procedure outlined in Importing V2 User Libraries below. You can also go back to PSCAD V2 and save your draft project as transfer (i.e. \*.dfx file) so as to incorporate any custom components into it before importing to V4.

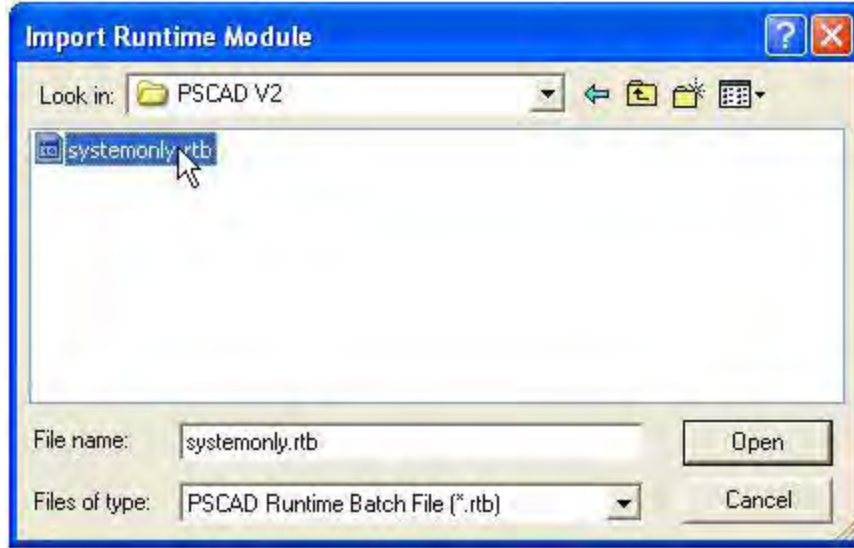
## Importing Additional Runtime Batch Files

As mentioned above, only a single runtime batch (\*.rtb) file will be initially imported along with a PSCAD V2 draft (\*.dft or \*.dfx) file. If more than one runtime batch file exists in a specific V2 draft project, then the remaining \*.rtb files can be imported separately. This is accomplished as follows:

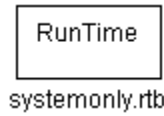
Once steps 1 to 4 above have been completed (i.e. the draft file has been successfully imported into PSCAD V4), right-click on the case filename in the Workspace and select *Import Runtime Module...*



This will bring up the *Import Runtime Module* dialog window. Select the \*.rtb file you wish to import and click the *Open* button.



A new Module containing the contents of the Runtime batch file should appear near the top-left corner of your case project main page, similar to that shown below. Simply left double-click the Module to enter.

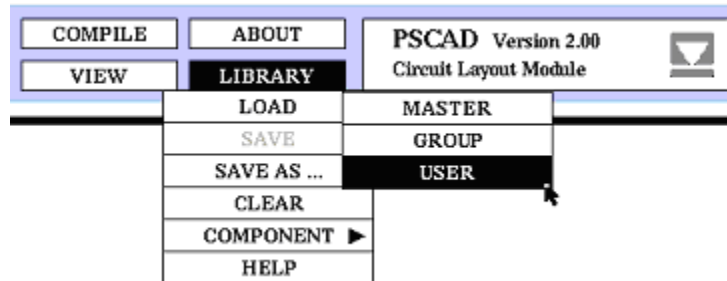


## Importing V2 User Libraries

The PSCAD V2 Draft program used a library (\*.lib) file as a palette (appears on the right-hand side of the V2 Draft program canvas) for the actual component library.

To transfer this palette into PSCAD V4:

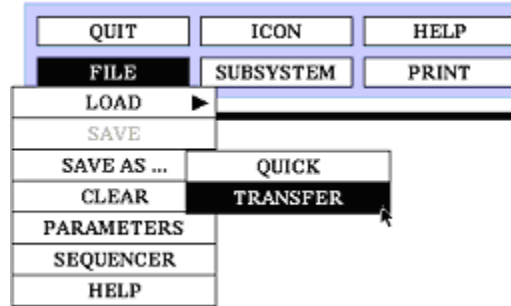
1. Open PSCAD V2 on your UNIX terminal and run the Draft program. Press the *LIBRARY* button in the main menu and select *LOAD | USER*.



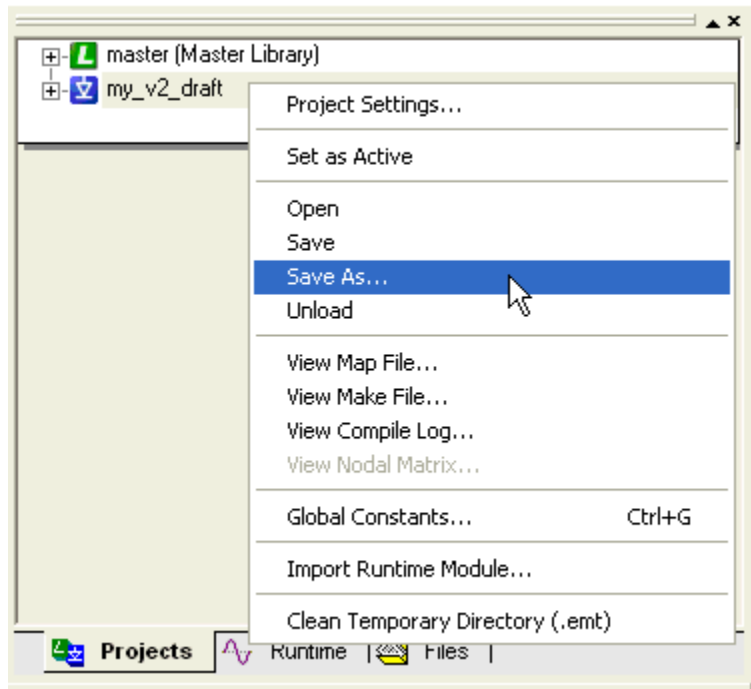
**NOTE:** Make sure that any of the components included in your PSCAD User Library file have a unique component definition name. Otherwise these components may be overwritten by other component definitions.

2. Select the desired user library file from the Load User Library dialog and then press the *PROCEED* button.

- Copy the desired component instances from the user library palette over to the Draft canvas. From the *FILE* menu, select *SAVE AS... | TRANSFER* to save the Draft project as a transfer file (i.e. as a \*.dfx file). This will ensure that the component definitions are included within the V2 draft file.



- Transfer this V2 project file to a location where it can be accessed from your Windows PC. If you do not have a direct network connection from your PC to your UNIX system, then you must copy (or ftp) the files to your PC from UNIX.
- Follow Steps 1 to 4 in Importing V2 Draft and Runtime Batch Files to load the V2 draft project into PSCAD V4.
- In the PSCAD V4 Workspace, right-click on the project filename and select *Save As...* from the pop-up menu.



- Rename the case project to a library project in the *File Name* field of the Save Project As dialog window. Press the *Save* button.

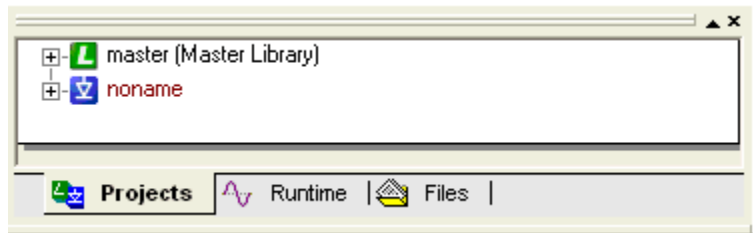
You now have converted your V2 user library to a PSCAD V4 library project!

## Importing Individual V2 Components

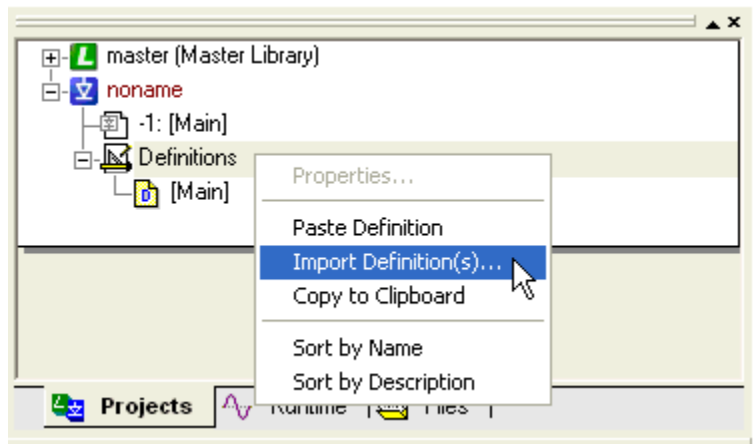
It is possible to import individual PSCAD V2 component definition files, directly into a PSCAD V4 library project. Once this is accomplished, an instance of the definition may be created and displayed within the library.

Please follow these steps to avoid problems:

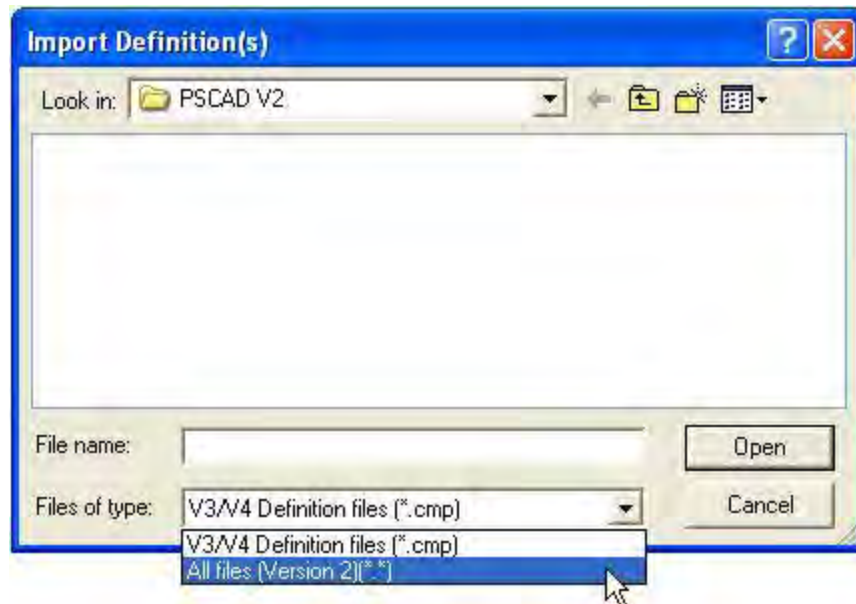
1. Create a new library project in PSCAD V4 (or open an existing library). To create a new library, select *File | New | Library*. A new library called 'noname' should appear in the Workspace window.



2. Expand the new library project tree by clicking on the '+' symbol beside its name. Right-click on the Definitions branch, and select *Import Definition(s)...*

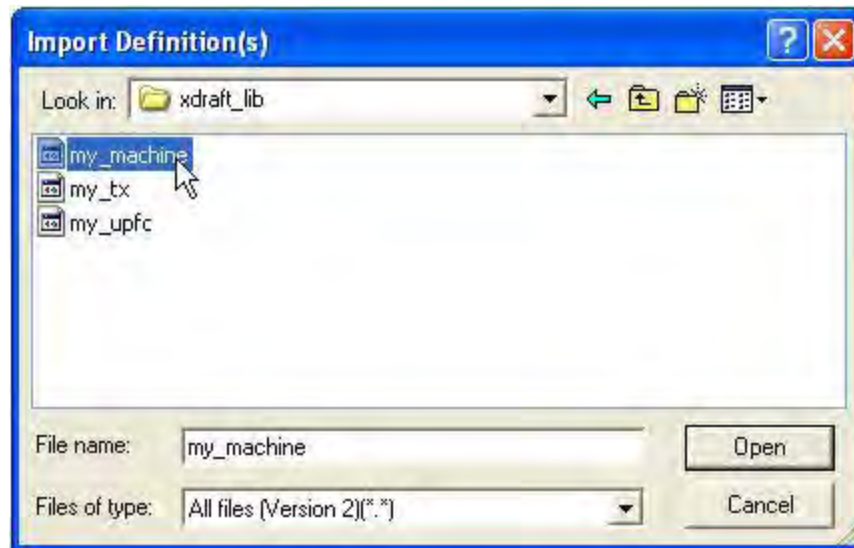


3. In the Import Definition(s) dialog window, change the *Files of Type* drop list to 'All Files'.



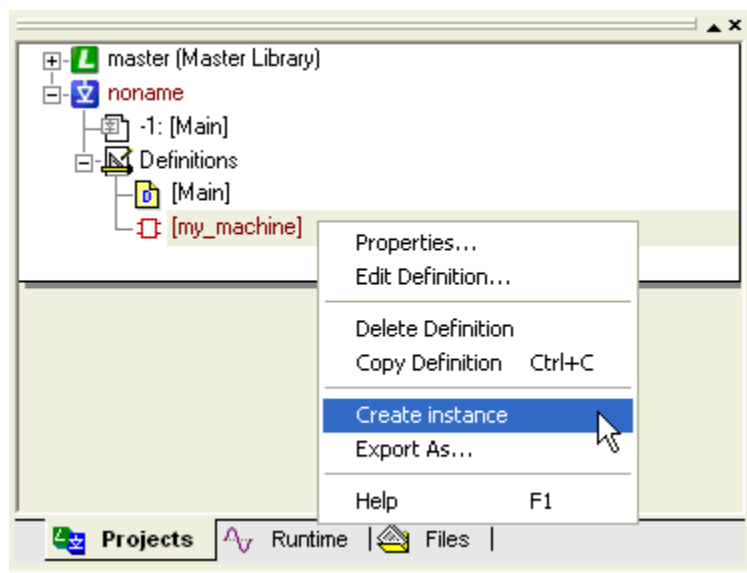
4. Navigate to the directory where you have stored your PSCAD V2 component definition files ('~/PSCAD/xdraft\_lib' for example). If you do not have a direct network connection from your PC to your UNIX system, then you must first copy (or ftp) the files to your PC from UNIX.



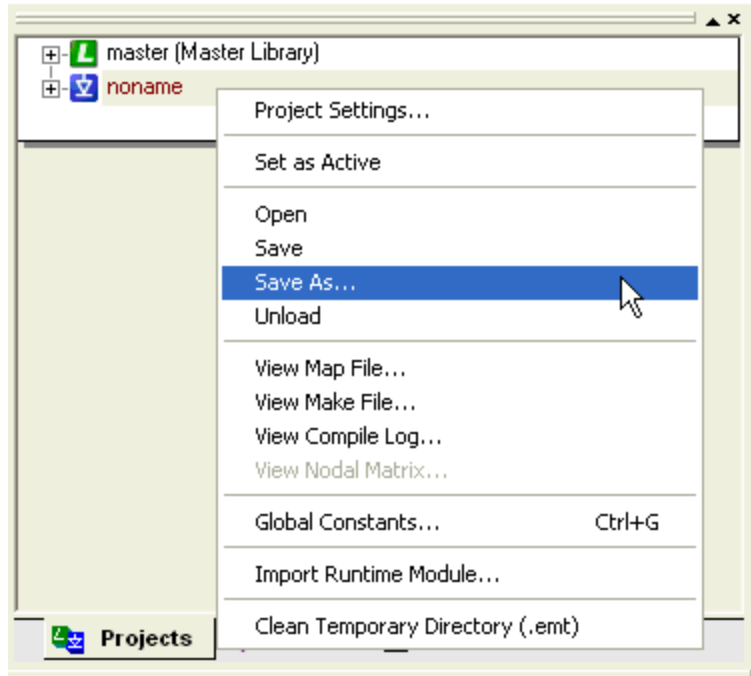


**NOTE:** Do not select any PSCAD V2 macro files (\*.g), library files (\*.lib), or any file beginning with a 'period'.

5. Select one or more files, and then click on the *Open* button. The new definition(s) should appear within the Definitions branch in the Workspace window.
6. Check for warning or error messages in the Output window. If you receive an error message similar to "Macro file 'xxx.g' not found", you need to move all associated V2 macro files (\*.g) from the '~/PSCAD/script' directory on your UNIX system to the same directory from where you are importing your V2 definition files. Unload the new library and start over from Step #1.
7. Expand the Definitions branch in the Workspace window by clicking on the '+' symbol beside its name. Select a definition, right-click and select *Create Instance* from the pop-up menu.



8. Open the library project main page in Circuit view. Right-click on the page and select *Paste* from the pop-up menu. A new graphical instance of the component should appear. You can also use Drag and Drop to perform steps 7 and 8.
9. Right click on the 'noname' library in the Workspace window and select *Save As...* to rename and save the library project.



## Manual Revisions to the New PSCAD V4 Project

Open the new PSCAD V4 project in Circuit view (left double-click on the project name in the Workspace window) and you should see a parent (top-level) module that did not exist in your PSCAD V2 project. This module will contain other modules representing each page that existed in your original V2 draft project. A separate module will also be included, which contains all information from the associated runtime batch (\*.rtb) file.

Depending on whether or not there were any associated import or export connections in the original V2 draft project, each module may now possess external input and output connections on their respective graphics. Each of these connections corresponds to an import or export connection in PSCAD V2. PSCAD V4 uses this input/output information to order modules in such a way as to minimize any feedback paths in signals.

### Too Many External Connections

If the parent module is visually unreadable (due to too many external connection graphics on the module) then this is probably because the original V2 Draft page had an excessive number of import or exports (i.e. > 100). The auto-routing of these connections in PSCAD V4 becomes too complex in such situations, but the connections are still functionally correct. To simplify matters, you can go back to the PSCAD V2 Draft circuit, and manually collapse related import and export signals into arrays. Then go through the import procedure again. This will reduce the number of connections in the parent module and help to keep things organized.

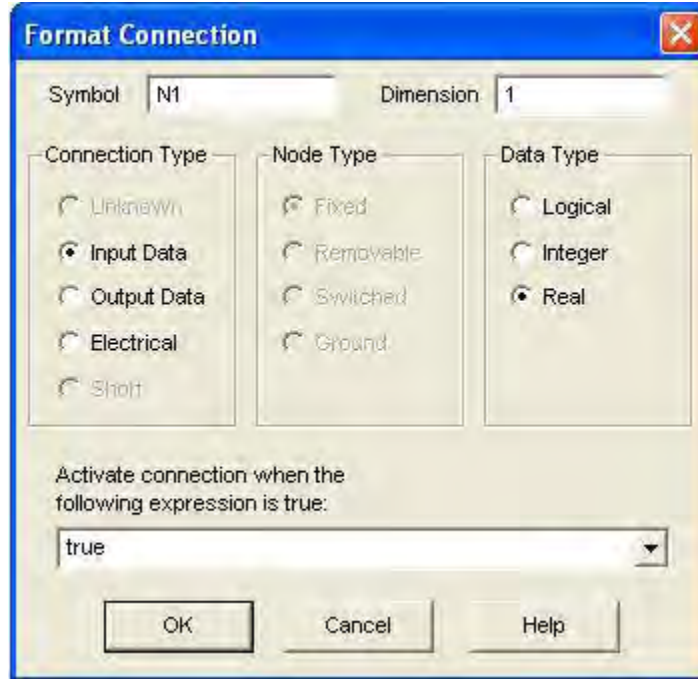
### Import/Export Tags with Array Variables

Each module may also require editing if you used any import or export tags with array variables in the original PSCAD V2 draft project. The old import and export components did not contain any information regarding the dimension or type of variable (all import/exports were performed with REAL numbers).

PSCAD V4 now allows import/exports to be performed with the original variable type, thus avoiding needless (and potentially incorrect) data conversions. The user must manually enter the import/export variable type and dimension information into the modules (there will be at least two modules that must be edited - one for the import and one for the export).

If you already know which signals are affected (i.e. imported or exported signals using arrays, or of a type other than REAL), then you must edit the definition of the module in PSCAD V4 to identify the signal types.

1. Open the definition of the module: Right-click over the module and select *Edit Definition....*
2. In Graphic view (appears by default), double-click on the Connection, having the incorrect type or dimension, to bring up the Format Connection dialog.



3. Modify the *Data Type* and *Dimension*. Press the *OK* button to save changes and then go back to Circuit view. Note that the *Connection Type* will be 'Input Data' if the signal in the original PSCAD V2 project was being imported, and will be 'Output Data' if the original signal was being exported.
4. Repeat steps 1- 3 for any other existing connections for this signal.

To find out if there are any more connection errors for a particular module, compile the module manually (i.e. right-click on the module and select *Compile Module*). This will force PSCAD to generate the data and Fortran files for this module alone, and any error or warning messages should appear in the Output window.

Examples of such errors are:

- **Signal 'ABC' dimension mismatch at signal 'XYZ':** This indicates a dimension mismatch between two connected signals.
- **Signal 'DEF' type mismatch at signal 'UVW':** This indicates a data type mismatch between two connected signals.

## Runtime Module

The Runtime module represents the contents of the V2 runtime batch file. This module may require some 'house cleaning' and re-organization (i.e. re-size graph frames, etc.). Control panels are used to contain sliders, dials, switches and meters. Grouped components in the PSCAD V2 Runtime program are now added merely as additional modules.

## Migrating V2 Cable Systems

PSCAD V2 cable migration into V4 is not supported. Any cable systems in your V2 project (i.e. defined in \*.clb files) must be reconstructed from scratch once the project is migrated into PSCAD V4.

## Migrating V2 Transmission Line Systems

There are fundamental differences in how transmission line systems are represented between PSCAD versions 2 and 4. In V4, transmission line systems are interfaced to the rest of the electric circuit through special interface components. The properties of the actual transmission corridor are defined within a special properties component (i.e. tower geometry, conductor properties, etc.).

When a PSCAD V2 draft project containing transmission line systems is migrated into V4, the above described transmission system is not automatically constructed. Instead, the transmission line system is inserted into the V4 project using special V2 alias components called V2 Style T-Line Connection. The transmission system properties remain based on the transmission line batch (\*.tlb) file created by V2.

Although this substitution will provide correct results when the simulation is run, it may prove cumbersome if the transmission system properties need to be changed (must edit the \*.tlb file), or if the project file is transferred to other users (must move the \*.tlb file with it). It is therefore recommended that V2 style transmission systems be eventually converted to V4 format.

Although this process must be performed manually, it is straightforward. Simply replace the V2 Style T-Line Connection components with an equivalent V4 transmission line interface component. The transmission line properties can then be read directly from the V2 transmission line batch file, and inserted into a properly constructed transmission line corridor. See the section entitled Constructing Overhead Lines in Chapter 8 of this manual for more details on constructing a V4 transmission line.