USER'S GUIDE

on the use of PSCAD

PSCAD

Power Systems Computer Aided Design

MANITOBA-HVDC
RESEARCH
CENTRE

211 Commerce Drive, Winnipeg, Manitoba, Canada R3P 1A3

Print history:

- February 3, 2003 - Version 4.0.0, first printing.
- March 21, 2003 - Version 4.0.1
- July 18, 2003 - Version 4.0.2, second printing.
- December 23, 2003 - Version 4.0.3
- April 30, 2004 - Version 4.1.0, third printing.
- April, 2005 - Version 4.2.0, fourth printing.
- February, 2010 - Version 4.2.1, fifth printing.

Foreword

Technology lives, breathes and fades away, giving way to fresh, innovative ideas. Strength comes from the sharing of thoughts and efforts of those people who work with and create these ideas - a synergy that is greater than the sum of its parts.

Sometimes moving forward means letting go of past practices. To create something new involves a willingness to take risk on those ideas that form a belief for the future. With that risk comes long days of change and self examination that is necessary along the way. Why it was done...matters to team; what was done...matters to everyone else. Many times during the planning, that path was changed as a result of new knowledge and understanding. Openness to that change made it possible to adapt and mold the vision. Ideas that seemed at first to be the right ones, grew increasingly obtuse, forcing them to be abandoned or re-factored into a new form.

I would not be truthful if I said I expected it to evolve as it had. The time and energy needed for this fundamental change far exceeds anyone's expectations. One of the traits that keeps the faith is the belief the solution is just around the corner. The road winds and bumps, the wind pushes back, but every wayside along the way bears fruit and reinforces the promise of the "road less traveled." Of the many goals of the original vision for "neXus," most have been delivered on the promise and have been brought to the table here. The techniques and ideas have been realized into a whole. Every part has been revisited and refined to share and interact with an all new smooth backbone of information. This intimacy of form and function is a new place; a new sunny beachfront with which a refreshing start is made.

The best part is that this is only the beginning of a whole new exciting series of changes to come that could never be done in the past. Tools and methods that were not possible before are now possible and in fact, very close at hand. It will be an exciting time ahead, and we are happy to share in the experience.

Craig Muller, P.Eng.
Software Development Manager
February, 2010

This masterpiece is dedicated to all the hardworking, programmers, engineers, students, professors and support staff who made this program possible.

Table of Contents

About This Guide

The goal of this guide is to provide a comprehensive reference for using the PSCAD software. It is composed of material from the Online Help, but may not contain the most up-to-date information. When in doubt, the Online help should be assumed accurate.

Information specific to certain models is not included here, but can be found in the Online Help. The online help also includes the EMTDC User's Guide.

For information on more advanced topics regarding simulating using EMTDC, please see the EMTDC User's Guide.

ORGANIZATION

The PSCAD User's Guide is organized in the following manner:

- **Chapter 1: Welcome to the World of PSCAD** briefly introduces the PSCAD product, its use and development, what is new, and also provides important information regarding both compiler and PSCAD restrictions.
- **Chapter 2: License Management** discusses issues related to licensing and the license manager software.
- **Chapter 3: Workspace and System Settings** describes all parameters included within the Workspace and Systems Settings dialogs.
- **Chapter 4: The Application Environment** describes the look and feel, including terminology and features. This chapter also provides a tutorial on running a simulation.
- **Chapter 5: Operations and Feature Overview** is essentially a 'how to' guide, which outlines most of the basic features and operations that are available. This chapter also includes a tutorial on creating a new project from scratch.
- **Chapter 6: Online Plotting and Control** provides details on creating and using the online plotting tools. Creating and using the online controls and meters is also discussed.
- **Chapter 7: Project Settings** describes all parameters included within the Project Settings dialog.

- **Chapter 8: Transmission Lines and Cables** provides details on constructing and defining transmission corridors for both overhead transmission lines, and underground cable systems. Related output files are also discussed.
- **Chapter 9: Component Design** is essentially a manual in itself, which describes how to design components using the Definition Editor. This chapter also includes a tutorial on designing a component from scratch.
- **Chapter 10: Definition Script** is a complimentary chapter for Component Design above. It describes all features and coding conventions involved in the Definition Script language.
- **Chapter 11: Project Debug and Refinement** describes common warning and error messages associated with running a project, as well as methods for debugging user code, creating library files and searching.
- **Chapter 12: MATLAB/Simulink Interface** provides details on the interface subroutine and how to set-up the interface and connect with MATLAB/Simulink.
- **Chapter 13: Migrating from Older Versions** outlines all procedures for migrating projects into the latest version. Important conversion issues are also discussed.
- **Index**: Alphabetical index of keywords with page numbers.

Chapter 1:

Welcome to the World of 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, analyze 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 30 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 PC 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 successfully released in 2002. 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 module locking, transmission line mutual coupling, and many others.

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 to 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 to 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 v4.1 or v4.2 only. Older component definition files with extension **.cmp* may be imported as well.

- **Downwards Compatibility**: The PSCAD X4 release is <u>not backwards compatible</u>. That is, X4 format project files (**.pscx* and **.pslx*) cannot be converted back to **.psc* or **.psl* format.

- **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
  - **Compaq Visual Fortran 6**: *.cf6
  - **Intel Visual Fortran 9 to 11**: *.if9

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.

- **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. 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.

- **Module Locking**: Module canvases may now be locked from viewing (locked and password protected). See *Module Locking* in Chapter 11 of this manual for more details.

- **Oscilloscope**: A new meter utility has been added as yet another avenue for viewing data online. See *Oscilloscopes* in Chapter 6 for more details.

- **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.

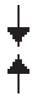
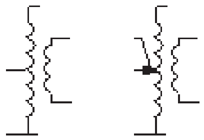
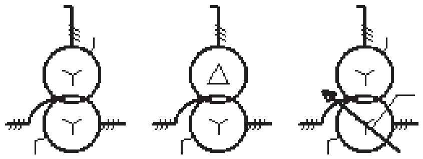
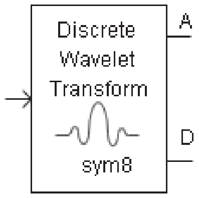
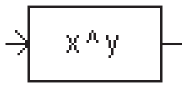
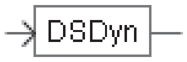
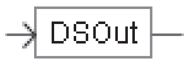
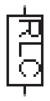
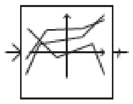
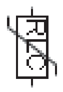
- **Saving Graphics to File**: Graphic objects used in the Graphics section of the component design environment can now be stored in and imported from files. See *The Graphic Section* in Chapter 9 of this manual for more details.

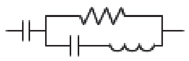
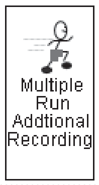
- **Transmission Line/Cable Error Messaging**: Messages sourced from the transmission segment solve step in the build process, are now displayed in the output window.

- **Zoom Rectangle/Extents**: See *Zooming* in Chapter 5 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 v4.2.1 release.

Graphic	Name
	Saturable Reactor

	Spark Gap
	1-Phase 3-Winding Auto Transformer
	3-Phase Star-Star Auto Transformer with Tertiary
	Discrete Wavelet Transform (DWT)
	X to the Power Y
	Force to DSDYN
	Force to DSOUT
	Runtime Configurable Passive Branch
	XY Table
	Variable Series Impedance Branch

	C-Type Filter
	Multiple Run Additional Recording

Master Library Component Updates
The following lists which existing master library components have been updated since the PSCAD v4.2.1 release.

- **Real Pole, Differential Pole, Lead-Lag**: Added initial value option when resetting at TIMEZERO.
- **On-Line Frequency Scanner (FFT)**: Added single-line diagram support.
- **1-Phase Auto Transformer, 3-Phase Star-Star Auto Transformer**: Added on-load tap changing capability.
- **Variable RLC**: Added dL/dt or dC/dt effects.
- **Feedback Loop Selector**: Added support of non-scalar input.
- **Single-Phase Breaker, Three-Phase Breaker**: Added breaker voltage output signal.
- **6-Pulse Bridge**: Added snubber circuit current measurement. Added reverse withstand voltage input. Added ability to set an unblock time.
- **3-Phase to SLD Electrical Wire Converter (Breakout)**: Added a compact graphical view option.
- **Multi-Mass Torsional Shaft Interface**: Increased maximum total masses from 6 to 26.
- **Random Number Generator**: Added Gaussian type random number distribution.

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.

- **DC Correction**: Two unique DC correction algorithms have been added, which ensure perfectly accurate DC parameters for time domain simulations.

- **Trace Fitting**: An alternative algorithm for fitting the propagation function H has been added to the Frequency Dependent (Phase) model. This method derives poles by fitting the trace of H (sum of diagonal elements), instead of fitting the modes of H. This method avoids the problem of occasional unstable poles inherent in the fitting of the modes of H.

- **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.

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.

GETTING IN CONTACT WITH US

The Manitoba HVDC Research Centre Inc. (the Centre) and its representatives are committed to providing you with the best sales and technical support available.

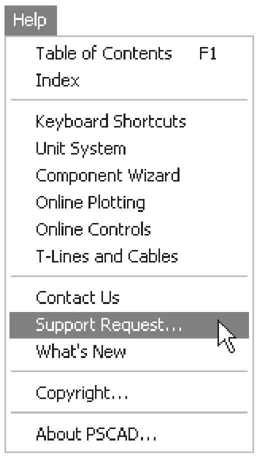
Contact your local PSCAD representative first for fast and efficient service. If you do not have their contact address from the time you purchased PSCAD, you can get it either from the PSCAD Web Site (www.pscad.com) or by contacting the Centre directly. To make the best use of our technical support and 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:

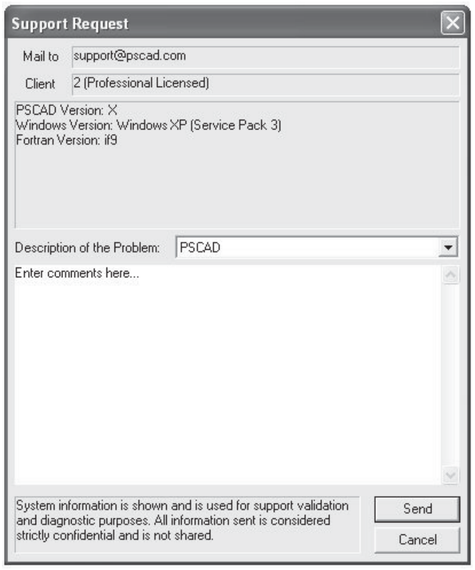
E-mail:	support@pscad.com
Phone:	+1 (204) 989-1240
Fax:	+1 (204) 989-1277
Web Site:	www.pscad.com
Address:	PSCAD Support Services Manitoba HVDC Research Centre Inc. 211 Commerce Drive Winnipeg, Manitoba R3P 1A3 CANADA

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 dialog window. Simply add your comments in the space provided. Please ensure that you select a **Description of the Problem** from the drop list, which most adequately describes your problem type. When finished, press the **Send** button. An email will be sent directly to the PSCAD Support Desk.


Support Request

Mail to: support@pscad.com

Client: 2 (Professional Licensed)

PSCAD Version: X
Windows Version: Windows XP (Service Pack 3)
Fortran Version: if9

Description of the Problem: PSCAD

Enter comments here...

System information is shown and is used for support validation and diagnostic purposes. All information sent is considered strictly confidential and is not shared.

PSCAD Community Forum
On August 18, 2004, the Manitoba HVDC Research Centre unleashed the PSCAD Community Forum to the online world. This forum is an online environment where users of PSCAD (and other related products) from around the world can meet and discuss the issues important to them. The forum includes several separate topic areas, and allows for the exchange of files through post attachments. Users may contribute to a growing list of example case projects, and show off what they have accomplished with PSCAD and/or other related software.

Please note that there are other regional users groups as well: Please contact PSCAD Technical Support (support@pscad.com) to find out if your region has one.

Membership
To become a member of the PSCAD Community Forum, you must have a passion for PSCAD/EMTDC! To add your name to the forum's worldwide member list, simply access the site (details below) and register online. There is no membership fee.

The PSCAD Community Forum should not be used as a support medium. If you have queries regarding problems simulating with PSCAD, bug reports, etc., please contact the PSCAD Support desk at support@pscad.com. Contacting the support desk will ensure a prompt response to your query.

Forum Details

Users can submit their questions/comments/examples regarding PSCAD or related software by posting under the appropriate forum. Your post will immediately be viewable to all registered members. If you are responding to an existing topic, the forum will automatically send an email to the member who posted the topic.

PSCAD Community Forum: bb.pscad.com

PSCAD Sales

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

E-mail:	sales@pscad.com
Phone:	+1 (204) 989-1240
Fax:	+1 (204) 989-1277
Web Site:	www.pscad.com
Address:	PSCAD Sales Manitoba HVDC Research Centre Inc. 211 Commerce Drive Winnipeg, Manitoba R3P 1A3 CANADA

Chapter 2:

License Management

All PSCAD software editions, except the *Student Edition*, must be licensed. Licensing is organized and controlled using *License Manager* software, which can exist either as a separate, standalone program, or embedded within PSCAD itself.

The type of licensing arrangement that is best for you depends on how you wish to utilize PSCAD, and on what type of license(s) you purchased. The following sections describe each type of licensing arrangement in detail.

Multi-User Licensing

The multi-user licensing (MUL) method was first introduced with PSCAD V3, and was then the only licensing option available. With this configuration, the license manager software is installed as a standalone program, which can be accessed by any computer (including the host), on a *Local Area Network* (LAN).

The license manager can only license PSCAD clients that are 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 PSCAD and license manager machines may need to be adjusted, so that PSCAD can contact the license manager machine

A multi-user set-up is capable of providing multiple users on multiple machines 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 several PSCAD users.

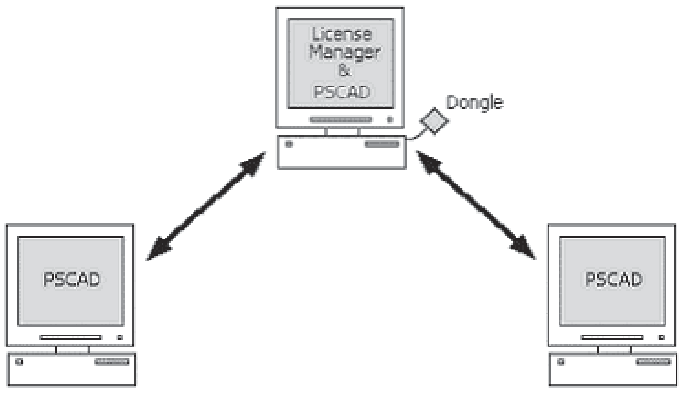
In multi-user licensing, the license manager is used in conjunction with a hardware lock (also known as a dongle), which contains licensing information for the purpose of validation. When an instance of PSCAD is started somewhere on the LAN, a license will be requested from the license manager. The license 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. Each time a user opens or closes the PSCAD program, it will request or relinquish a license respectively. Each licensable edition of PSCAD

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

possesses a separate product number, so that several products can be licensed by the same license manager.

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



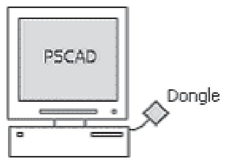
EXAMPLE: A PSCAD user owns three professional edition licenses programmed onto one dongle. The above diagram illustrates how the user could install the license manager on a LAN, so that up to three users can run PSCAD simultaneously.

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

Single-User Licensing

Single-user licensing (SUL) still involves the use of a dongle, but a standalone license manager program is not required. Here, an embedded license manager, within the PSCAD program itself, manages the licenses.

A single-user license will only allow a single user on a single machine to run only one instance of a licensable edition of PSCAD. Single-user licensing is designed especially for use on standalone machines and laptops, and will NOT allow access from other machines, even if it is connected to a LAN.



EXAMPLE: A PSCAD user owns one professional edition license programmed onto a single dongle. 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 infinite (no time limit).

Trial Licensing

In order to simplify the evaluation process, and to provide access to full PSCAD functionality, trial licensing is available for the educational and professional editions. A trial license is a special type of single-user license, where a time limit is mandatory and the user may only possess a single trial license. V4.3 is used

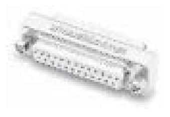
With trial licensing, neither the license manager nor the dongle is required. However, the user must request a trial license after PSCAD is installed. Note that until the trial license is approved and sent by the Manitoba HVDC Research Centre Inc., the installed copy of PSCAD will function as a Student Edition (SE), with all SE limitations imposed (i.e. 15 electrical nodes maximum).

Do not upgrade your operating system, hard drive, or network card during the duration of the trial license period. Any one of these changes will cause your trial license to be invalidated.

Trial licenses are locked to the hardware and registry of a particular machine and are therefore not transferable from one machine to another. The latest installed trial license will overwrite any previously existing trial licenses.

Hardware Locks (Dongles)

The license manager software supports the following types of hardware locks:



Sentinal SuperPro™ 797, Low Profile Parallel Port Hardware Lock

Sentinal SuperPro™ USB, USB Port Hardware Lock

The parallel port dongle is perfect for laptop computers, as it features a low profile and does not protrude from the back of the machine (and typically does not ever need to be removed). The serial and parallel port locks are compatible with all Windows platforms on which PSCAD is supported.

License Key

The license key is an ASCII text file named ‘license.txt,’ which usually exists as an electronic file, is required for both the PSCAD professional and educational editions. This file contains encrypted information designed to act as a ‘Key’ to adding or upgrading licenses (both single-user and standalone) to the license database file, which is used directly by the license manager.

LICENSE MANAGER SOFTWARE

In addition to the standalone license manager software that was first introduced with PSCAD V3, the PSCAD package now includes two additional types of license manager. The type of license manager you will need depends on your licensing arrangement.

There are a few important points to discuss regarding the maintenance and operation of the license manager. These are outlined in the following sections as well.

Standalone License Manager

The *Standalone License Manager* (SLM) is used only with multi-user licenses. The SLM can operate on a dedicated server, or on any machine running PSCAD. Any combination of PSCAD products can be included and used interchangeably with this license manager. Installation instructions for the SLM are given later in this chapter.

Maintenance and Support

Once the license manager is installed and running, it is a good idea for the person operating the designated license manager server to maintain the software and ensure proper operation.

The SLM usually starts automatically when the machine is booted and runs as a background process. All SLM events and transactions are recorded in a log file entitled ‘Lmgrd-log.txt,’ which is located under *...\Documents and Settings\All Users\Application Data\Manitoba HVDC Research Centre\License Manager*. When troubleshooting, this file contains important information about the cause of any problems.

If problems arise with the operation of your license manager software, first please try to address the problem using the clues found in the log file. If the solution remains elusive, send us an email at PSCAD Support Services (support@pscad.com) with a detailed description of the problem. To ensure efficient response time, please attach the log file (i.e. **Lmgrd-log.txt**). Also, go to **START** | **Programs** | **HVDC Lmgr** and run the ‘Get License Info’ utility and attach the generated ‘Get License Info’ file (**getinfo.txt**) to the email.

Local License Manager

The *Local License Manager* (LLM) is utilized only when running under a single-user license. Although the LLM will support multiple product licenses, it will not support multiple instances of any individual PSCAD product. Attempting to start a second instance of a certain product (say professional edition), will force the second instance to run as the student edition.

Trial License Manager

The *Trial License Manager* (TLM) is utilized only when running under a lockless trial license. The TLM supports only a single, time limited license of the professional or educational edition (multiple trial licenses are not supported). As with the local license manager, the TLM will not grant licenses to other machines on the LAN, and only one instance of a trial license may be active at one time. Any additional instances of PSCAD will run as the student edition.

PSCAD Usage Log

The usage of all licensed PSCAD editions, such as 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 *...\Documents and Settings\All Users\Application Data\Manitoba HVDC Research Centre\License Manager* directory on the machine 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,   Pro, Remote,    7327,    177, Mon Aug 22 10:45:42 2005, Mon Aug 22 10:48:39 2005
 lha,   Pro, Remote,    7328,    176, Mon Aug 22 10:46:21 2005, Mon Aug 22 10:49:17 2005
```

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* or *Edu*. Use of student edition is not logged as it is not a licensed edition and does not require the LM.
- **<user location>**: *Remote* for users using PSCAD on machines other than the LM machine. *Local* for users using PSCAD on the same machine as the LM machine.
- **<LicenseID>**: This is the license ID of the granted license, and appears in the LM log file (lmgrd-log.txt), and in the PSCAD usage log file (PscadLmgr.txt).
- **<start time>**: The time that the LM granted the PSCAD license to the user.
- **<finish time>**: The time that the user quit the PSCAD application.

Once you have submitted trial license request, a new license key will be sent pending approval. Trial licensing is only available for the professional and educational editions.

TRIAL LICENSES

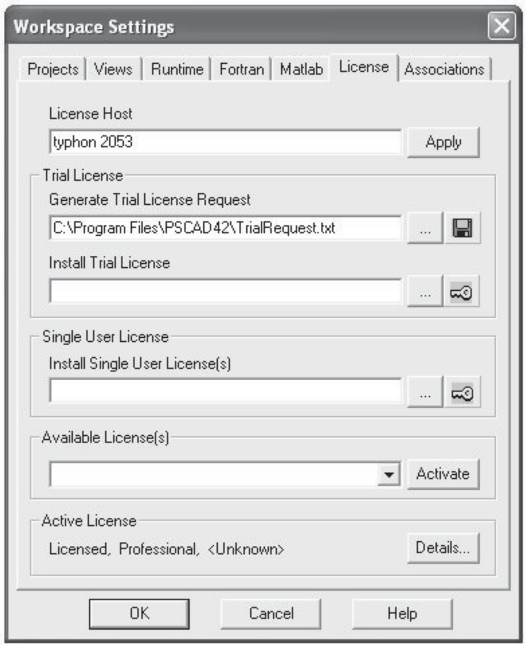
If you have already installed the student edition and require more than the fixed 15 electrical node limit, or if you would like to review the component design features, you may be interested in a time limited evaluation license for the professional edition. If interested, please follow the procedures below to receive your trial license.

Requesting a Trial License

As part of the *requesting a trial license* procedure, you must provide us with some specifics on the PC you will be using to run PSCAD. This information is supplied to us in the form of a text file, which contains important machine metrics, such as computer name and other operating system information. This data is required to create a trial license file and is held in the strictest confidentiality.

The following procedure outlines how to request a trial license:

1. Open the PSCAD student edition and go to **Edit | System Settings...** in the main menu bar to open the system settings dialog.
2. Select the **License** tab near the top of the dialog.



License Host: typhon 2053

Trial License

Generate Trial License Request: C:\Program Files\PSCAD42\TrialRequest.txt

Install Trial License

Single User License

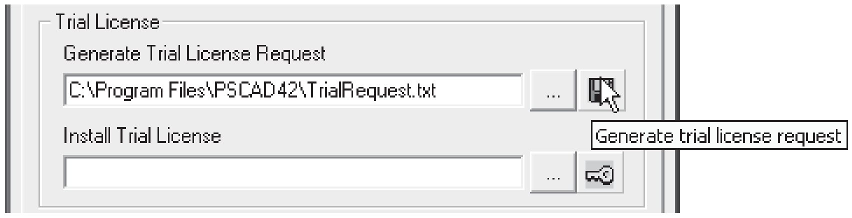
Install Single User License(s)

Available License(s)

Active License

Licensed, Professional, <Unknown>

3. In the **Trial License** area of this dialog, enter a filename and path in the input field entitled **Generate Trial License Request** or use the **Browse** button to navigate to the desired directory. This will specify where PSCAD is to place the trial license request file once it is generated. To avoid confusion, leave this path and filename as default. Press the '**Generate**' button directly to the right of this field.

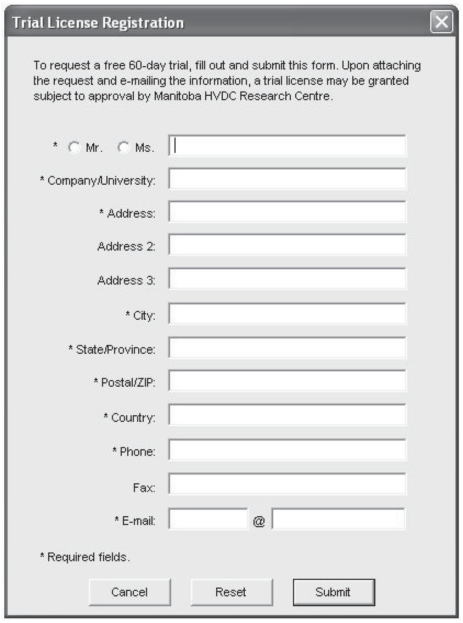


Trial License

Generate Trial License Request: C:\Program Files\PSCAD42\TrialRequest.txt

Install Trial License

4. A *Trial License Registration* entry form will appear. Enter all required information into the form and then click the **Submit** button near the bottom of the form.



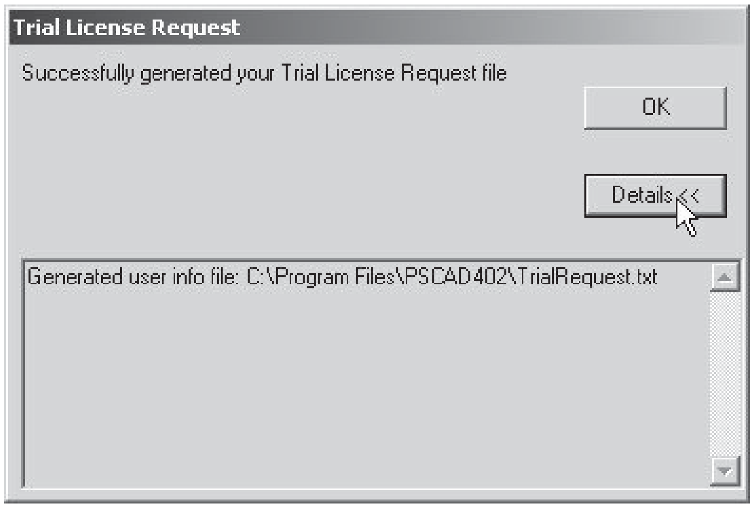
Trial License Registration

To request a free 60-day trial, fill out and submit this form. Upon attaching the request and e-mailing the information, a trial license may be granted subject to approval by Manitoba HVDC Research Centre.

* ○ Mr. ○ Ms.	
* Company/University:	
* Address:	
Address 2:	
Address 3:	
* City:	
* State/Province:	
* Postal/ZIP:	
* Country:	
* Phone:	
Fax:	
* E-mail:	@

* Required fields.

5. If all fields in *Step 4* were entered correctly, PSCAD will generate the trial license request file and place it in the directory specified in *Step 3*. A trial license request dialog will appear confirming the placement of the file.

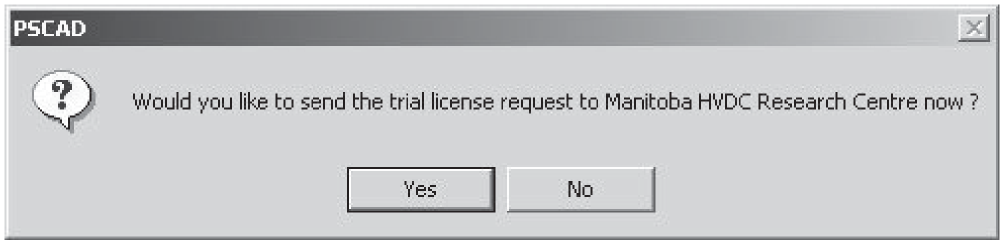


Trial License Request

Successfully generated your Trial License Request file

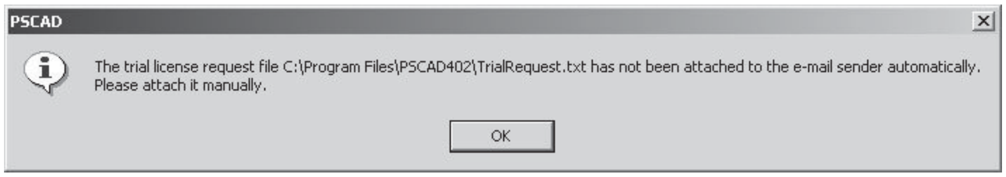
```
Generated user info file: C:\Program Files\PSCAD402\TrialRequest.txt
```

6. A dialog will appear asking if you would like to email the trial license request file now. Click **Yes** to proceed to *Step 7*. Click **No** if you wish to email the trial license request file at a later time. If you decide to send the file at a later time, you can start at *Step 8* of this procedure, as the file has already been created.



Would you like to send the trial license request to Manitoba HVDC Research Centre now ?

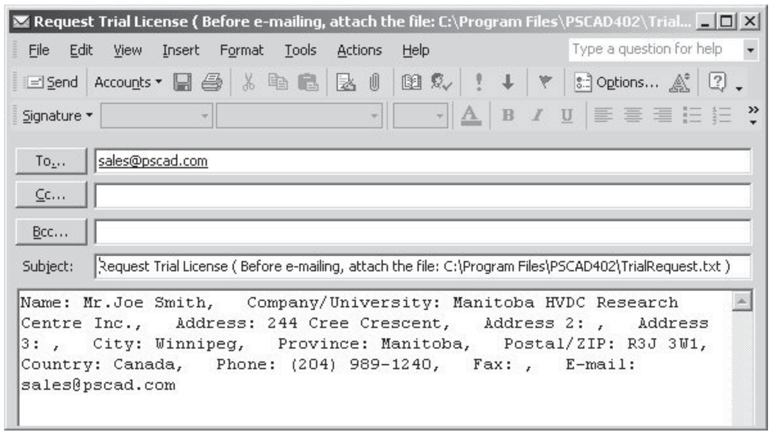
7. Another dialog will appear with a simple warning. Please read carefully:



The trial license request file C:\Program Files\PSCAD402\TrialRequest.txt has not been attached to the e-mail sender automatically. Please attach it manually.

If you generated your trial license request file at an earlier time, you do not need to add the text to the body of the email, as this is contained within your trial license request file anyway.

8. PSCAD will then open a new email form from your default email tool. The body of the email will contain the information given in *Step 4* and the sending address is set to *PSCAD Sales* (sales@pscad.com).


Request Trial License (Before e-mailing, attach the file: C:\Program Files\PSCAD402\Trial...
File Edit View Insert Format Tools Actions Help
Type a question for help
Send Accounts Options...
Signature
To...
Cc...
Bcc...
Subject:

To: sales@pscad.com

Subject: Request Trial License (Before e-mailing, attach the file: C:\Program Files\PSCAD402\TrialRequest.txt)

```
Name: Mr.Joe Smith,   Company/University: Manitoba HVDC Research
Centre Inc.,   Address: 244 Cree Crescent,   Address 2: ,   Address
3: ,   City: Winnipeg,   Province: Manitoba,   Postal/ZIP: R3J 3W1,
Country: Canada,   Phone: (204) 989-1240,   Fax: ,   E-mail:
sales@pscad.com
```

Do not upgrade your operating system, change your system clock, or change any hardware configuration (i.e. hard drives, motherboard, etc.) while waiting for your trial license Key

9. Remember to manually attach the trial license request file to email in *Step 8* (the file will located in the directory specified in *Step 3*). **Send** the email to *PSCAD Sales* (sales@pscad.com). You should receive a response within the next couple of days. See the next section entitled *Installing a Trial License* upon receipt of your trial license key.

Installing a Trial License

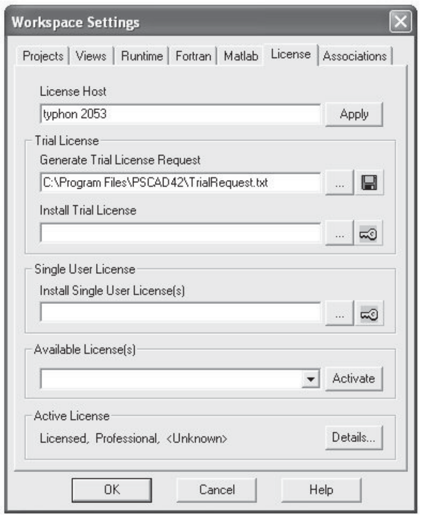
Upon approval of your trial license request, you will receive an email from us with your trial license key attached (i.e. 'Trial.txt').

Before proceeding with installation, please note the following important points:

- The trial license can only be installed on the computer from which it was requested.
- When operating under a trial license, PSCAD will only run if it is the only PSCAD instance active on your machine. Quit all other instances of PSCAD prior to starting with a trial license.
- The trial license will not operate if certain computer system configurations are changed. Do not: Upgrade your operating system, change your system clock, or change any hardware configuration (i.e. hard drives, motherboard, etc.) during the course of the trial license period.
- If you encounter problems during this process, contact us at *PSCAD Support Services* (support@pscad.com).

The following procedure outlines how to install a trial license:

1. Save the trial license key (i.e. 'Trial.txt' file) somewhere on your hard drive (say C:\temp).
2. Open the student edition and go to **Edit | System Settings...** in the main menu bar to open the system settings dialog.
3. Select the **License** tab near the top of the dialog.



License Host: typhon 2053

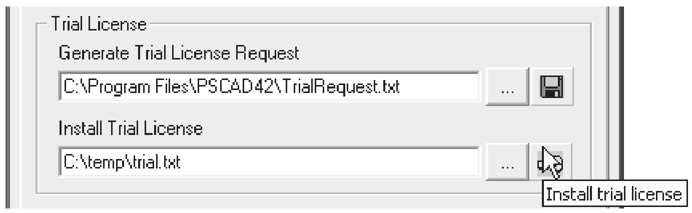
Trial License
Generate Trial License Request: C:\Program Files\PSCAD42\TrialRequest.txt
Install Trial License

Single User License
Install Single User License(s)

Available License(s)

Active License
Licensed, Professional, <Unknown>

4. In the trial license area of this dialog, enter the filename and path to your 'Trial.txt' file in the input field entitled **Install Trial License** or press the **Browse** button to navigate to the file. Once entered, press the **Install** button directly to the right of this field.

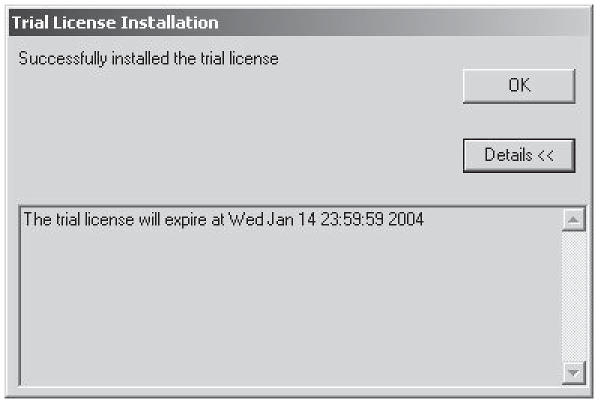


C:\Program Files\PSCAD42\TrialRequest.txt

C:\temp\trial.txt

If the license does not install correctly, please contact us at *PSCAD Support Services* (support@pscad.com).

5. If trial license is installed correctly, a trial license installation dialog will appear confirming this, along with the expiry date of the Trial License. Click the **OK** button.



Successfully installed the trial license

The trial license will expire at Wed Jan 14 23:59:59 2004

6. Close the student edition and then open the licensed edition. See *Viewing Active License Information* for details on checking if your trial license is running properly.

ADDING/UPGRADING STANDALONE LICENSE MANAGER LICENSES

You do not need to reinstall the license manager in order to update or add a license. You will simply need to update the license manager database file, which resides on your computer's hard drive.

To update or add a new standalone license manager license to your license database file, use the *Enter License Key* program. Do not attempt to manually edit this file:

1. Go to **START | Programs | HVDC Lmgr** and select *Enter License Key*.

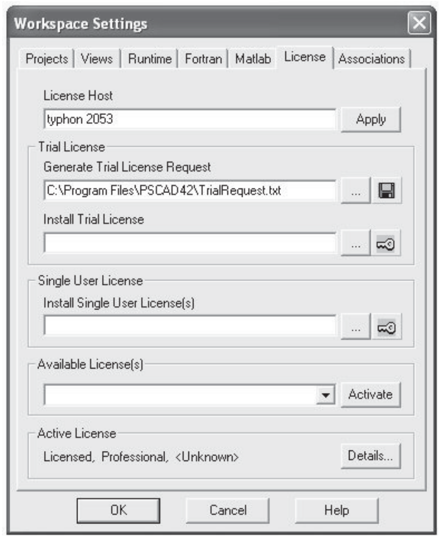
2. Follow instructions given.

ADDING/UPGRADING A SINGLE-USER LICENSE

You do not need to reinstall the license manager in order to update or add a single-user license. You will simply need to update the license manager database file, which resides on your computer's hard drive.

To update or add a new license to your license database file:

1. Save your license file to a convenient directory (say C:\temp).
2. Open the student edition and go to **Edit | System Settings...** in the main menu bar. Click the **License** tab.



License Host: typhon 2053

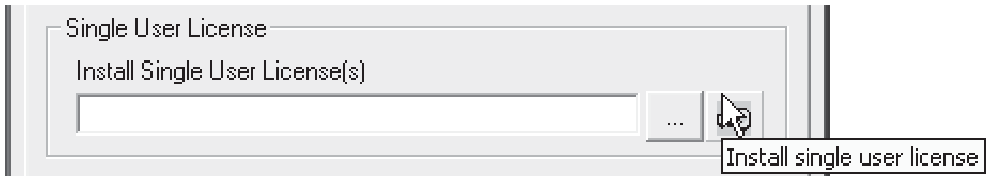
Trial License
Generate Trial License Request: C:\Program Files\PSCAD42\TrialRequest.txt
Install Trial License

Single User License
Install Single User License(s)

Available License(s)

Active License
Licensed, Professional, <Unknown>

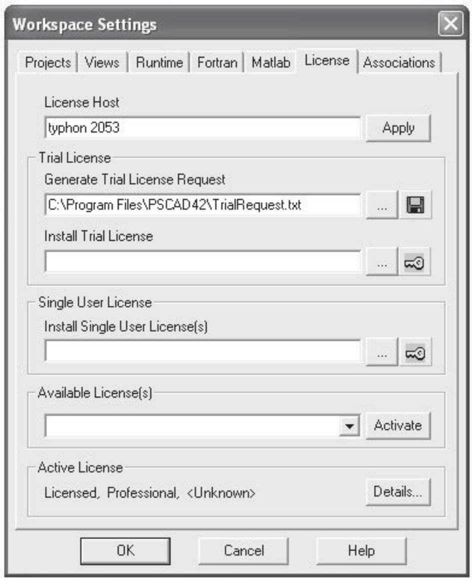
3. In the **Single User License** area of this dialog, enter the filename and path to your single user license file in the input field entitled **Install Single User License(s)**. Press the **Install** button directly to the right of the input field and navigate to the saved 'license.txt' file. Select **OK** in the pop-up dialog window.



4. Click **OK** on the *System Settings* dialog and exit the student edition.
5. You should now be able to start the edition for which you are licensed.

CHANGING ACTIVE LICENSING SETTINGS

PSCAD allows you to alter your present (i.e. 'active') license settings without the need to close and reopen the program in a different mode. You can change the license manager host computer, and/or the license type (i.e. professional, educational or student) directly from the *System Settings* dialog. To open this dialog, go to **Edit | System Settings...** in the main menu bar and click the **License** tab.



License Host: typhon 2053

Generate Trial License Request: C:\Program Files\PSCAD42\TrialRequest.txt

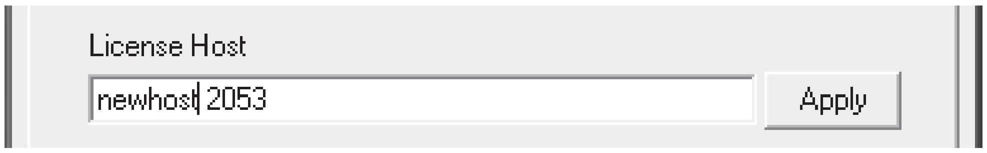
Install Trial License

Install Single User License(s)

Licensed, Professional, <Unknown>

Changing License Manager Host

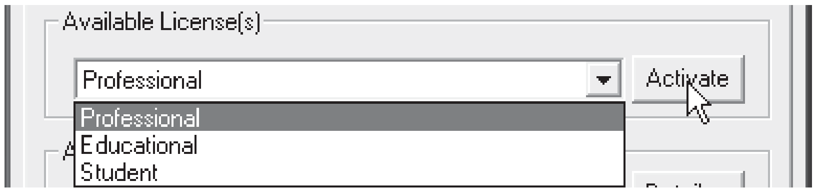
To change the license manager host computer, simply enter the name of the new host (along with 2053) and click the **Apply** button.



License Host: newhost 2053

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

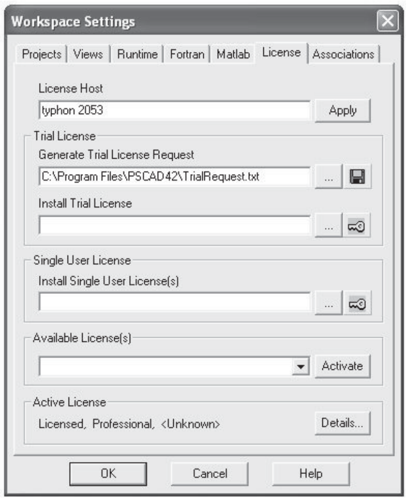
Changing Active License Type
To change the active license type, click the down arrow in the selection box under the **Available License(s)** area and select the required license. Click the **Activate** button.



VIEWING ACTIVE LICENSE INFORMATION

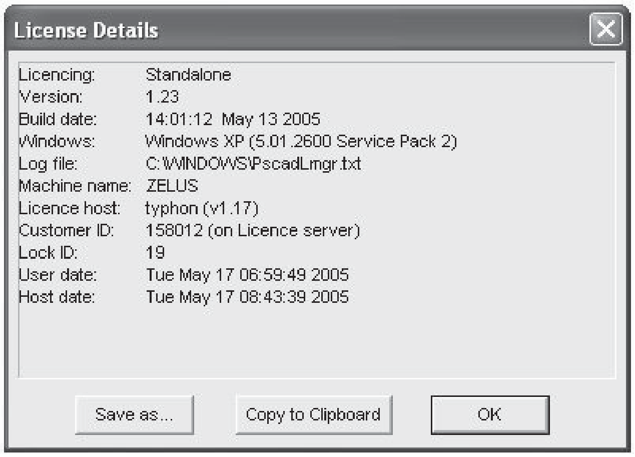
At any time, you can review the status of your active license.

1. Open PSCAD and go to **Edit | System Settings...** in the main menu bar. Select the **License** tab.



License Host
typhon 2053
Apply
Trial License
Generate Trial License Request
C:\Program Files\PSCAD42\TrialRequest.txt
Install Trial License
Single User License
Install Single User License(s)
Available License(s)
Activate
Active License
Licensed, Professional, <Unknown>
Details...
OK
Cancel
Help

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. A dialog should pop-up displaying all available information regarding your current licensing. If you wish, you can save this data to a text file or copy it to the Windows clipboard. Select **OK** to close the dialog.



```
Licencing:      Standalone
Version:        1.23
Build date:     14:01:12  May 13 2005
Windows:        Windows XP (5.01.2600 Service Pack 2)
Log file:       C:\WINDOWS\PscadLmgr.txt
Machine name:   ZELUS
Licence host:   typhon (v1.17)
Customer ID:    158012 (on Licence server)
Lock ID:        19
User date:      Tue May 17 06:59:49 2005
Host date:      Tue May 17 08:43:39 2005
```

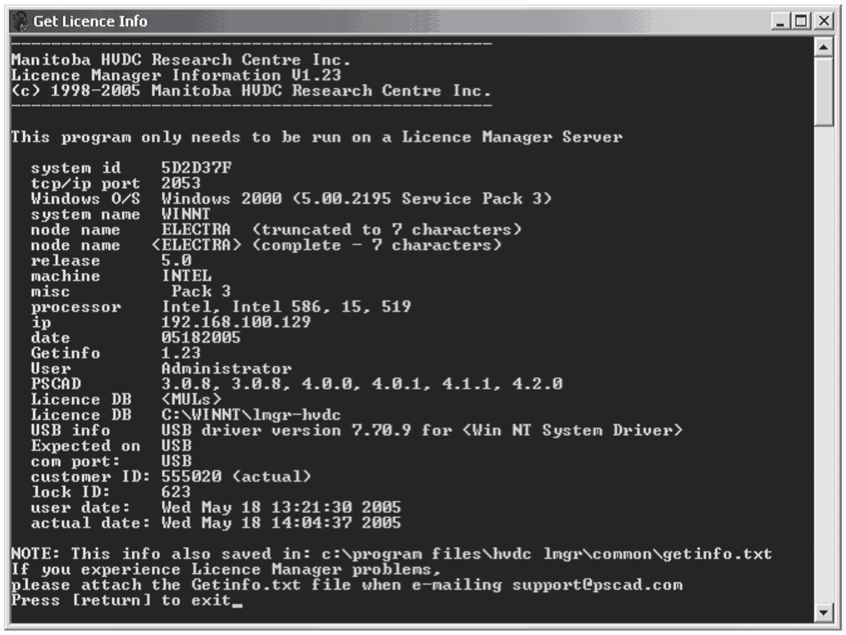
GETTING LICENSING INFORMATION

If you would like information about your licenses, you can get it directly from the dongle itself. This information is also helpful in troubleshooting if a problem arises, and is used by *PSCAD Support* to help solve license manager issues.

Standalone License Manager Information

To view your standalone license manager information, use the ‘Get License Info’ program:

1. Go to **START | Programs | HVDC Lmgr** and select ‘Get License Info.’



```
--------------------------------------------------
Manitoba HVDC Research Centre Inc.
Licence Manager Information V1.23
(c) 1998-2005 Manitoba HVDC Research Centre Inc.
--------------------------------------------------

This program only needs to be run on a Licence Manager Server

  system id     5D2D37F
  tcp/ip port   2053
  Windows O/S   Windows 2000 (5.00.2195 Service Pack 3)
  system name   WINNT
  node name     ELECTRA  (truncated to 7 characters)
  node name    <ELECTRA> (complete - 7 characters)
  release       5.0
  machine       INTEL
  misc           Pack 3
  processor     Intel, Intel 586, 15, 519
  ip            192.168.100.129
  date          05182005
  Getinfo       1.23
  User          Administrator
  PSCAD         3.0.8, 3.0.8, 4.0.0, 4.0.1, 4.1.1, 4.2.0
  Licence DB    <MULs>
  Licence DB    C:\WINNT\lmgr-hvdc
  USB info      USB driver version 7.70.9 for <Win NT System Driver>
  Expected on   USB
  com port:     USB
  customer ID: 555020 (actual)
  lock ID:      623
  user date:    Wed May 18 13:21:30 2005
  actual date: Wed May 18 14:04:37 2005

NOTE: This info also saved in: c:\program files\hvdc lmgr\common\getinfo.txt
If you experience Licence Manager problems,
please attach the Getinfo.txt file when e-mailing support@pscad.com
Press [return] to exit_
```

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

A text file will also be created at the same time called ‘getinfo.txt’ (as indicated above). This is an important file to include when contacting *PSCAD Support Services* by email (support@pscad.com).

Active Licensing Information
See *Viewing Active License Information* for more details.

MANUALLY STARTING THE LICENSE MANAGER

The method for starting the 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 LICENSE MANAGER

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

Windows XP/Vista/7

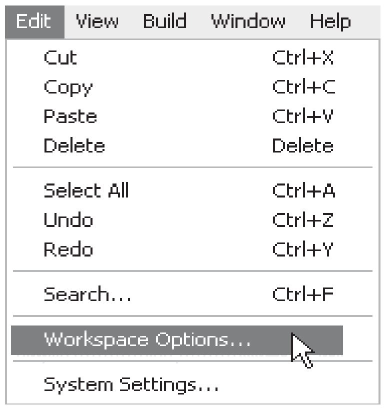
- Right-click on your **My Computer** desktop icon and select **Manage**.
- Double-click on the **Services and Applications** entry and then double-click **Services**.
- Highlight **HVDC License Manager**, right-click and select **Stop**.

Chapter 3:

Workspace and System Settings

Now that the application has been installed successfully, you may want to adjust some of the work environment settings and preferences that are available. Referred to as *system settings*, they affect all projects currently loaded in the workspace window. These parameters are contained within a couple of dialogs: The main dialog is called *Workspace Settings*, as well as another called *Graphics Settings*.

Go to **Edit | Workspace Settings...** or **Edit | Graphics Settings...** in the main menu bar to access either of these dialogs.



The settings are organized into several categories:

Workspace Settings

- Projects
- Runtime
- Fortran
- Matlab
- License
- Associations

Graphics Settings

- Options
- Diagnostics

WORKSPACE OPTIONS

Environment

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

- **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.
- **Update Interval**: Update the message table by the time interval set here.
- **Hovering appearance**: If set to *Show Bounding Box,* a dashed box will appear surrounding the component over which the mouse pointer is situated.
- **Tool-tip appearance**: Select *Animated Popup* to enable animated tool tip pop-up windows.
- Use shift key to create controls and curves: 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.
- **Navigate into a module**: 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.
- **Navigate project focus**: This option controls the focus of the circuit canvas when selecting a project in the workspace window.
- **Colour for definition view**: 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 Chapter 5 of this manual for more details.
- **Rendering**: 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.

- **Text anti-aliasing**: Enabling this option ensures that all text is anti-aliased (smoothed). It may be left to *System Default* if desired.
- **Frame and panel appearance**: Select a preferred panel style for control panels and graph frames. The choices are *Plain Paper, 3D Shadow* and *Metal Chisel*.
- **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.

Projects
The projects section contains a variety of user preferences related specifically to projects.

- **Save document every**: PSCAD will save all project documents loaded in the workspace at the intervals specified here.
- **Before building**: If set to *Save all changes*, case projects are automatically saved whenever they are compiled.
- **Messages**: This option allows the user to save all build and runtime messages, displayed in the *Output Window*, as part of project (*.pscx) files. While this option is enabled, project messages will be included in any project that is saved. To remove the saved messages from any project, set this option to *Discard on unload* and then save that project.
- **Change tracking**: Enables clipboard undo and redo if set to *Use Undo/Redo Stack*.
- **On startup**: This setting will ensure that all library and case project files, which were loaded in the workspace when the application was previously exited, are reloaded again at start-up. When disabled, only the master library will be loaded at start-up.
- **Most recently used listing:** Enter the number of recently loaded projects that you would like to appear in the **File | Recent Files** menu in the main menu bar. You may enter a number up to 16 recently used projects.
- **Source folder**: 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.

- **Update parameter keys**: 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*.

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

- **If type conversion mismatch**: Set to *No action* if you do not want to view signal type conversion warnings.
- **If unit system converter is disabled**: Set to *No action* if you do not want to be notified when the unit system converter is disabled.
- **Environment variables**: Using private process-based environment settings can eliminate conflicts that occur when multiple FORTRAN compilers are installed.

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

- **Notify if storage needs exceed**: If the storage required for a simulation exceeds the selected value, a warning will be issued before the simulation is run.
- **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.
- **Waveform file will be overwritten**: 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 file will be overwritten**: 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.

Dependencies
This area contains settings related to dependent files and folders:

- **Compiler information file**: Selects the compiler information file. This file includes configuration information about compatible vendors and versions.
- **Master library**: Select the file to be used as the master library.
- **EMTDC binaries folder**: Enter the path to the EMTDC binaries folder. You can either enter the path directly, or use the browse button to select the directory. This path is set to the installation directory by default, and should normally be left as such.
- **User binaries folder**: The path entered here is automatically appended to files entered into the *Additional Library (*.lib) and Object (*.obj) Files* input field in the Project Settings dialog.
- **SVG script folder**: Enter the path to the SVG script folder. This folder is where graphics scripts have been stored. See *Graphic Objects* in Chapter 9 of this manual for more detail on graphic scripts.

External Tools

This area contains settings related to external software tools:

- **Browser**: Enter the path to your preferred HTML browser executable file. You can either enter the path directly, or use the browse button to select the directory. This browser will be used for viewing your user-written component help files. See the section entitled *Script Section* in Chapter 9 of this manual for details on setting up component help files.

Graphics Diagnostics

This area contains settings related to the painting of graphics in PSCAD:

- **Component scaffolding**: Enables scaffolding on components to show footprint.
- **Screen buffering**: Enables off-screen draw for best performance.
- **Fast scrolling**: Enables off screen scroll draw for best performance.
- **Graphics mapping**: Re-use graphics mapping for best performance.

SYSTEM SETTINGS

FORTRAN

The FORTRAN section is used to set-up FORTRAN related settings.

- **Installed 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 compilers.

MATLAB

The MATLAB section is used to set-up MATLAB/Simulink interface related settings.

Interface Settings

The interface settings area is used to set-up a directory path to the MATLAB installation libraries.

- **Installed 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.
- **Library Path**: If you are using MATLAB version 5.0, you must specify a path to the MATLAB installation library directory. This path will be used for all projects using the MATLAB/Simulink interface.   You can either enter the path directly, or use the browse button to select the directory.

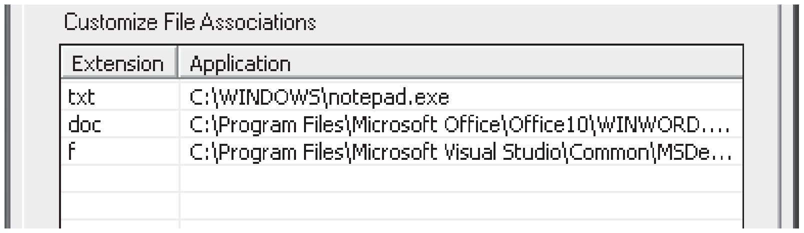
Enabling the MATLAB/Simulink interface is also a project-specific option. See the section entitled *Link* in Chapter 7 of this manual for more details.

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.

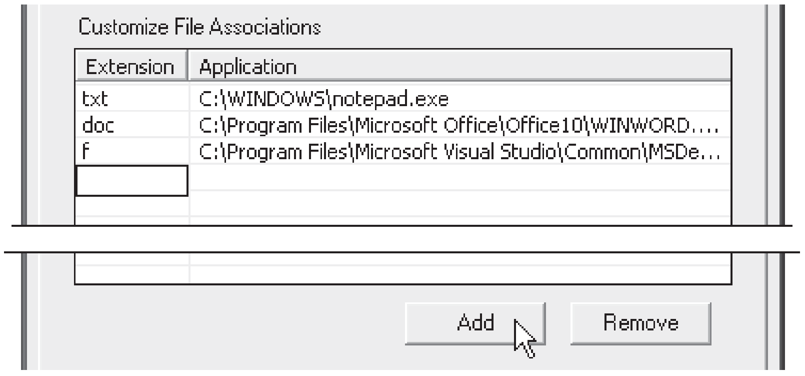


Customize File Associations

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

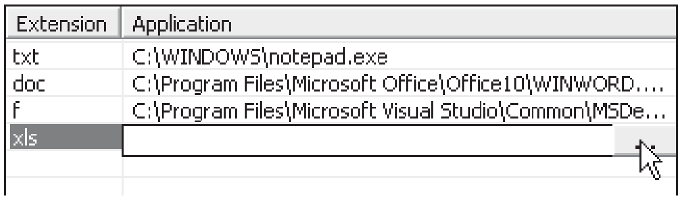
To enter a new file association, press the **Add** button. The next available input box in the **Extension** column will appear as selected.



Customize File Associations

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

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.



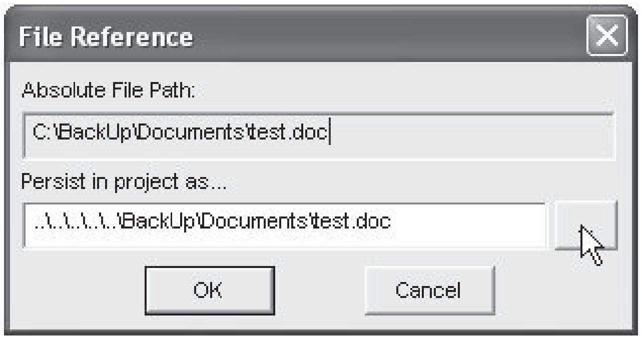
Extension	Application
txt	C:\WINDOWS\notepad.exe
doc	C:\Program Files\Microsoft Office\Office10\WINWORD....
f	C:\Program Files\Microsoft Visual Studio\Common\MSDe...
xls	

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 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.



File Reference

Absolute File Path:
C:\BackUp\Documents\test.doc

Persist in project as...
..\..\..\..\..\BackUp\Documents\test.doc

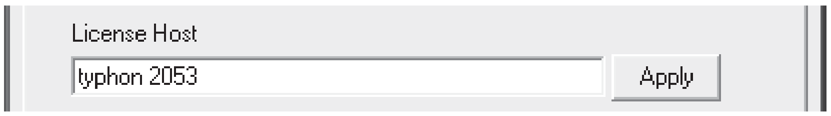
That's all there is to it! The next time the *File Reference* component is double-clicked, it will start the associated application and open the specified file.

License

The License section is devoted to license manager and other licensing preferences and features.

License Host

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

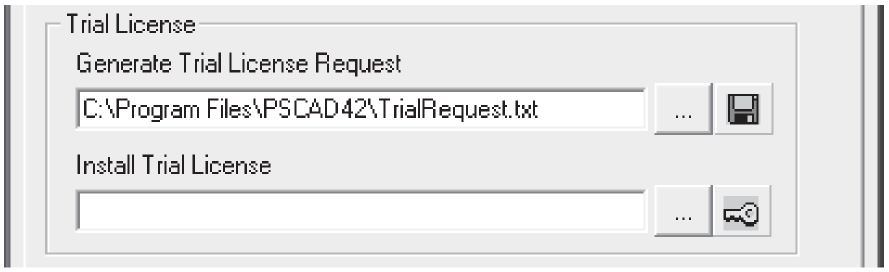


typhon 2053

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 *Changing Active Licensing Settings* in chapter 2 of this manual.

Trial License

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

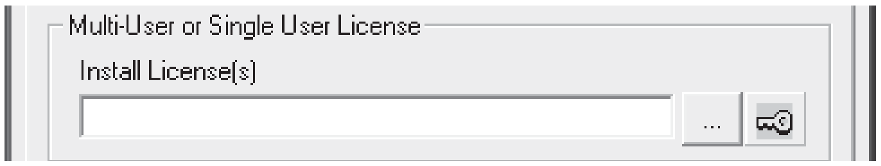


C:\Program Files\PSCAD42\TrialRequest.txt

See *Trial Licenses* in chapter 2 of this manual for more details.

Multi-User or Single User License

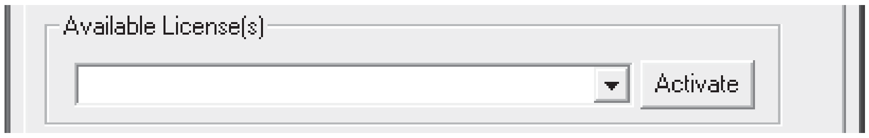
The multi-user or single user license area is used only when installing and activating these types of licenses.



See *Adding/Upgrading a Single-User License* and *Getting Licensing Information* in chapter 2 of this manual for more details.

Available License(s)

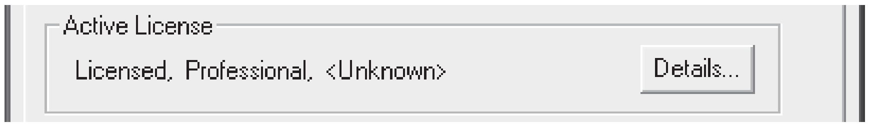
The Available License(s) area is used for switching between available licenses.



See *Changing Active Licensing Settings* in chapter 2 of this manual for more details.

Active License

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


Active License

Licensed, Professional, <Unknown>

See *Trial Licenses* in chapter 2 of this manual for more details.

Chapter 4:

The Application Environment

The term *Application Environment* refers not only to how PSCAD is organised 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.

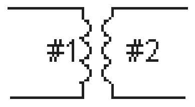
This chapter discusses a wide range of introductory topics from simple terms and definitions, to a general overview of most important elements that make up the environment. The tutorial section *My First Simulation* is a must read for all users new to PSCAD.

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.'

TUTORIAL: MY FIRST SIMULATION

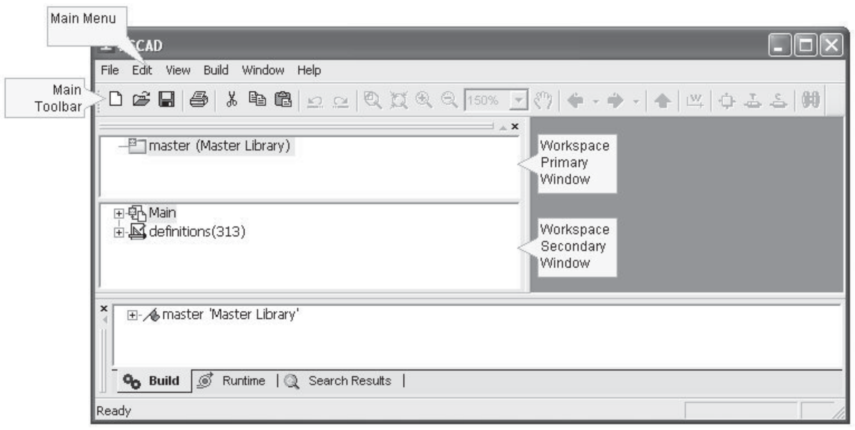
This tutorial is designed to give new users a 'jump start' in learning how to use PSCAD, as well as to provide a chance to try out a PSCAD simulation before proceeding further in this chapter.

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!

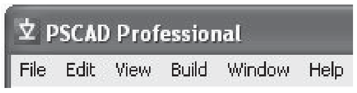
Starting PSCAD
Go to **Start | All Programs** | **PSCAD** in the Windows Start menu, and select the PSCAD application to start it. You will see the main PSCAD environment, as shown below.



There is a list of items across the top of the environment (**File**, **Edit**, etc.), which are part of the main menu. The buttons directly below the main menu are part of the main toolbar.

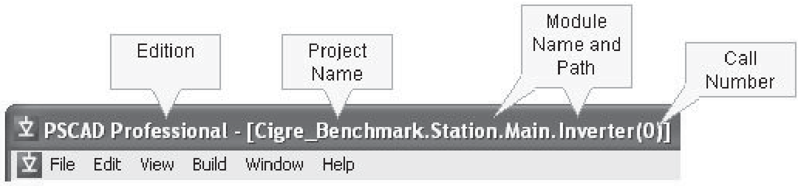
Title, Menu and Main Tool Bar

The image below illustrates the PSCAD title and menu bars.



Title Bar and Active Project

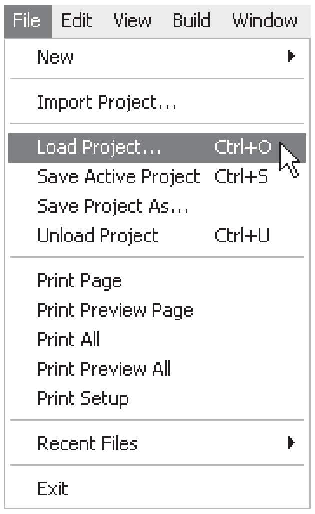
The top most part of the window that displays the application name is called the title bar. The title bar will also display the project name, along with the name, path and call number of the module whose canvas is currently being viewed. A call number of *-1* indicates that the current view is a module definition (non-white canvas background).


Edition
Project Name
Module Name and Path
Call Number
PSCAD Professional - [Cigre_Benchmark.Station.Main.Inverter(0)]
File Edit View Build Window Help

Menu Bar and Menu Items

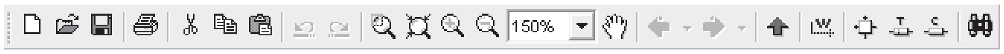
The area directly under the title bar is called the main menu. All menus are drop down, so if you left-click one with your mouse pointer, you will see a list of items appearing below it.

To select an item from this list, first move the pointer onto that item and then left click. The following menu shows how to load a project using the **File** menu from main menu bar.



Toolbar Buttons

The row of buttons directly below the main menu bar comprises the main toolbar.

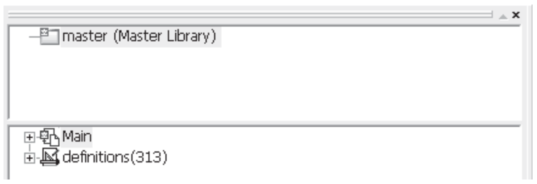


The main toolbar buttons initiate actions as soon as you click on them and hence are more convenient to use. For this reason, most frequently used operations have toolbar button equivalents.

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.

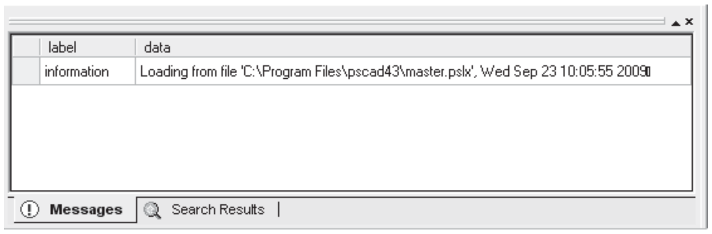
Workspace and Output Windows

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 main menu bar and click on **View | Workspace**.



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 for example.

Directly below the workspace, you should see another docked window referred to as the *Output* window. Click on **View | Output** if the output window is not visible.



label	data
information	Loading from file 'C:\Program Files\pscad43\master.pslx', Wed Sep 23 10:05:55 2009

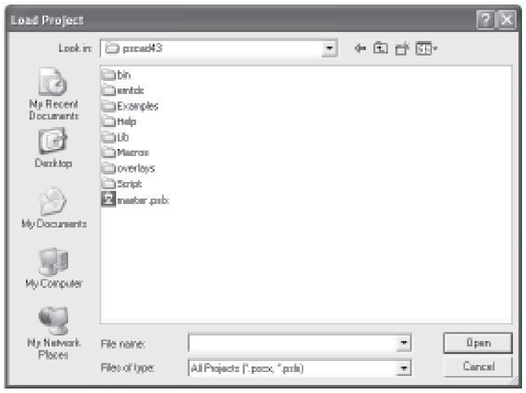
All the status (informational), warning and error messages involved in both *Build* and *Runtime* procedures are logged in this window - so, it is a good idea to keep this window visible at all times.

Loading 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*.

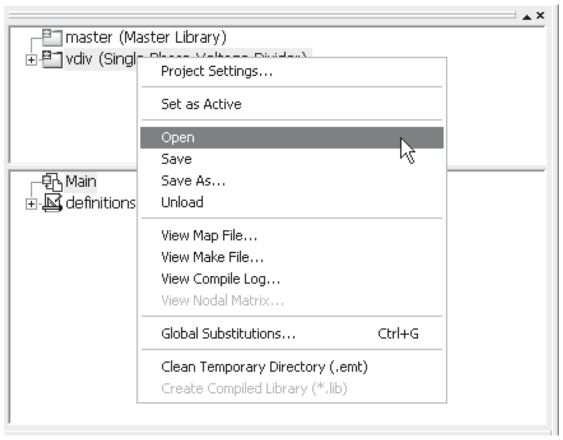
To load an existing case project, click on **File** in the main menu bar and select **Load Project...** You can also either press **Ctrl + O** or click the **Load** button in the main toolbar.

You should see the **Load Project** dialog appear on your screen. By default, the selected file type is **All Projects (*.pscx, .pslx)** at the bottom of the dialog.

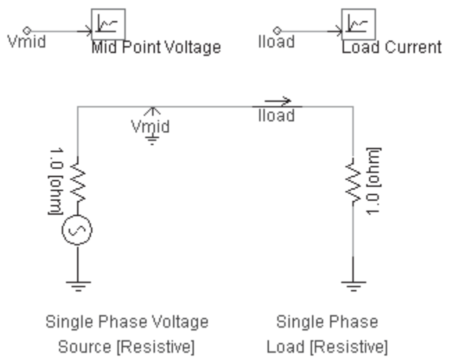


Navigate to the *tutorial* directory inside your PSCAD installation directory (i.e. ...\Program Files\PSCADxx\examples\tutorial). Click on the *vdiv.pscx* file and then click on the **Open** button to load this project.

The workspace window will now list an additional project entitled *vdiv - vdiv (Single Phase Voltage Divider)* directly under the master library project. Double-click on the project title (or right-click and select **Open**) in the workspace window to open and view the main canvas of the project in the Circuit window.



You should see the assembled voltage divider circuit as shown below, which is located at the top left corner of the main page of the project that you just opened. The plots are situated directly to the right of the circuit.


Vmid
Mid Point Voltage
Iload
Load Current
Vmid
Iload
1.0 [ohm]
1.0 [ohm]
Single Phase Voltage Source [Resistive]
Single Phase Load [Resistive]

The circuit consists of a single-phase resistive voltage source connected to a resistive load. Since the magnitude of the source resistance (1 Ω) 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.

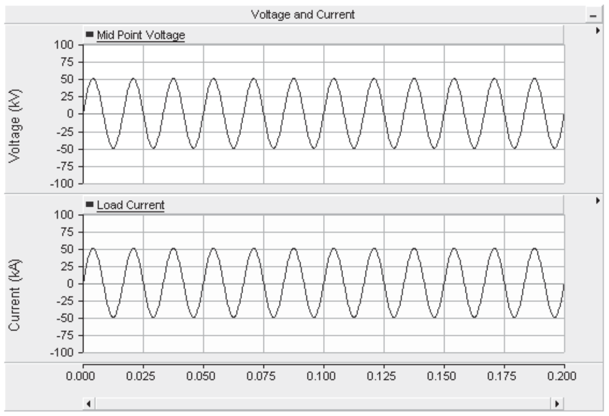
Running a Simulation

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 properties - note that the source voltage magnitude is *70.71 kV RMS* (or *100 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 kV* source voltage, we know that the mid-point voltage should then be *50 kV peak*, and the load current should be *50 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 runtime toolbar. 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:

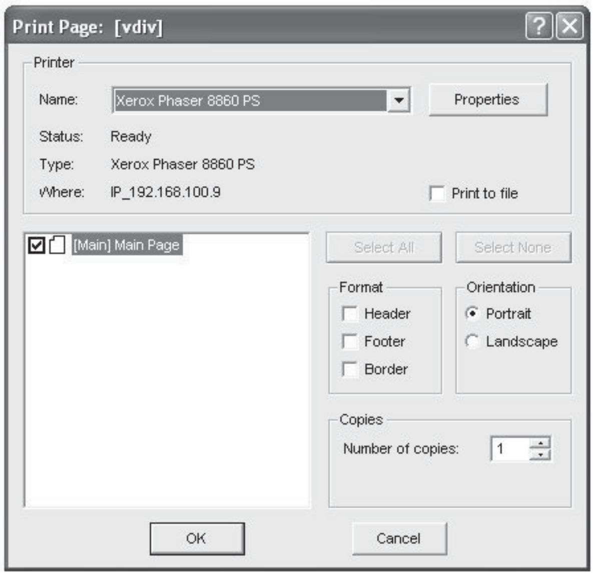


Make sure that your simulation produces the same result as shown here. This is one step towards ensuring that your PSCAD is installed correctly.

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 main circuit page and either select **Print Page** or **Print Preview Page** items.


Print Page: [vdiv]

Printer
Name: Xerox Phaser 8860 PS
Status: Ready
Type: Xerox Phaser 8860 PS
Where: IP_192.168.100.9
☐ Print to file

☑ [Main] Main Page

Format: ☐ Header ☐ Footer ☐ Border
Orientation: (● Portrait) ○ Landscape
Copies — Number of copies: 1

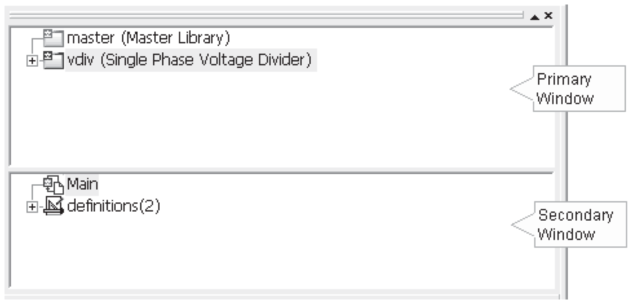
This should bring up either the Print dialog or the Print Preview Viewer. The contents of the Print dialog depend on what you are printing - click the **OK** button to proceed.

THE WORKSPACE

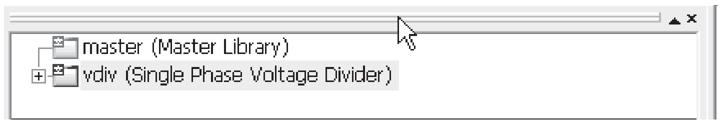
The master library is always the first project loaded into the workspace. The master library cannot be unloaded.

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

The workspace window 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.

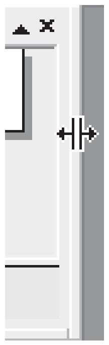


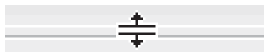
Moving and Resizing the Workspace Window
The workspace window may be moved and dropped anywhere in the environment. To do this, left click and hold directly over the top window bar.



Drag the window to where you want it placed (you should see a boxed outline of the window) and let go of the mouse button. Depending on where the window outline is when the mouse button is released, the window will either dock itself into position or appear as a floating window.

The workspace window can be resized by moving the mouse pointer over one of the four window edges until the pointer changes to that shown below:



or 

Click and hold the left mouse button and drag the mouse pointer to resize in either the horizontal or vertical direction as shown above.

The Primary Window

When a case or library project is loaded, the project filename and description will appear in the primary workspace window. It is of course possible to load multiple projects, which will be listed in the order in which they were loaded.

The primary window is used mainly for inter-project navigation, or for viewing project related data files.

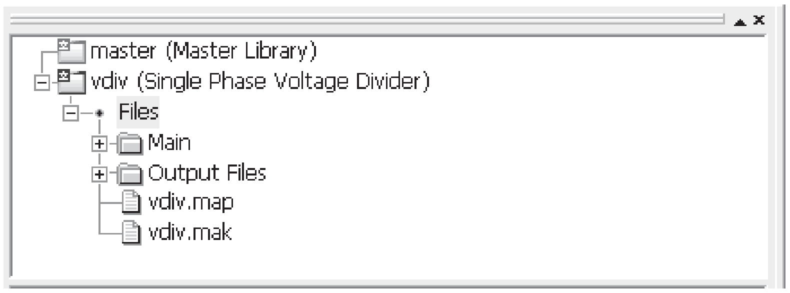
Icons are included for visual differentiation between case and library projects. These are listed below:

- Library Project
- Inactive Case Project
- Active Case Project

The Files Branch

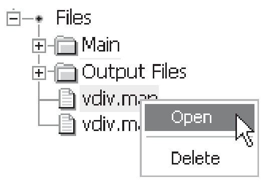
The Files branch is used specifically for the organization of all files found in the project temporary directory (see *The PSCAD Temporary Directory* in this chapter), such as FORTRAN, Data, Output, etc. files. The main trunk of the Files tree is arranged according to module hierarchy in the corresponding project, with an additional **Output Files** branch specifically to harbour miscellaneous output files, unrelated to specific modules.

To view the Files branch, left-click the [+] box beside the project name:



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

The files shown on the main trunk are the map (*.map) and make (*.mak) files associated with the project. 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:

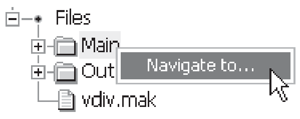


The functions in this menu are as described below:

- **Open**: Open the selected file for viewing in the main window.
- **Delete**: Delete the selected file from the temporary directory.

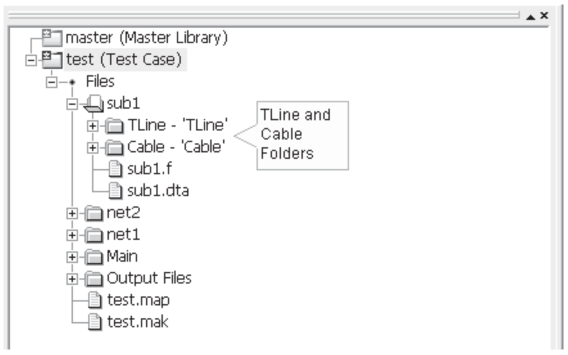
Module Folders

Most folders in the Files branch represent a module definition. All files within a particular folder pertain only to the associated module definition (i.e. data (*.dta) and FORTRAN (*.f) files). It is possible to navigate to the definition of a module by right-clicking on the folder and selecting **Navigate to...**.



Transmission Lines and Cable Files

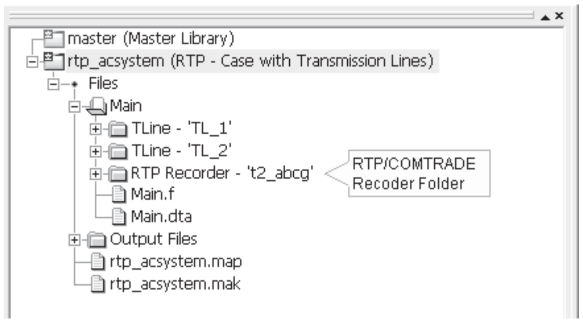
If there are any existing transmission lines or cables within a module, all related line constants files will appear within a separate directory, organised by the transmission system name. In the example below, the module 'sub1' contains both an overhead transmission line and an underground cable system, in addition to a FORTRAN and Data file:



It is possible to navigate to the definition of a transmission line or cable by right-clicking on the folder and selecting **Navigate to...** (as described in *Module Folders* above).

RTP/COMTRADE Recorder Files

If there are any existing *RTP/COMTRADE Recorder* components within a module, all related files will appear within a separate folder, designated by the *RTP/COMTRADE Recorder* output filename. In the example below, the module *Main* contains an *RTP/COMTRADE Recorder* component with an output filename labelled as *t2_abcg*, in addition to a FORTRAN and Data file:



```
master (Master Library)
rtp_acsystem (RTP - Case with Transmission Lines)
  Files
    Main
      TLine - 'TL_1'
      TLine - 'TL_2'
      RTP Recorder - 't2_abcg'
      Main.f
      Main.dta
    Output Files
    rtp_acsystem.map
    rtp_acsystem.mak
```

is possible to navigate to the definition of a *RTP/COMTRADE Recorder* component by right-clicking on the folder and selecting **Navigate to...** (described *Module Folders* above).

Active Project

As mentioned earlier, multiple projects may be listed in the workspace primary window. If a case project is to be compiled and run, the application needs to know which one. This is accomplished by selecting an *Active* case project (right-click on the project and select **Set as Active**).

See *Setting the Active Project* in the next chapter for more details.

The Master Library

The master library is always the first project listed in 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 double-click the title in the workspace or right-click on the title and select **Open** from the pull down menu. The main master library canvas will open (Circuit 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

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.

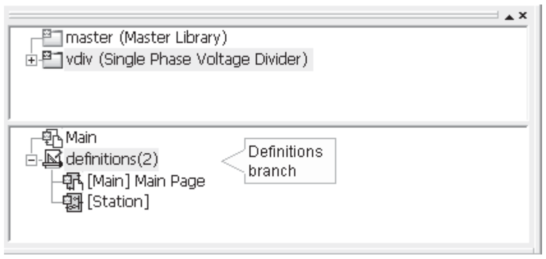
users may find this format allows for easier navigation to the correct component.

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

The Secondary Window

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

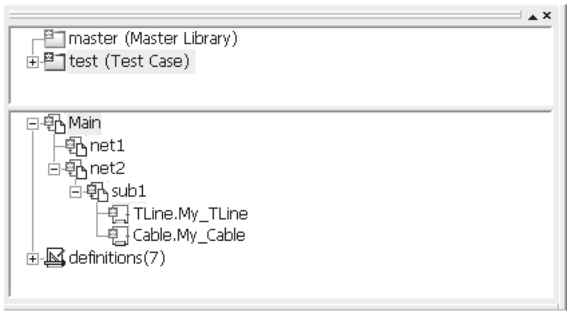
Each project contains information regarding stored definitions, as well as module hierarchy, organised into a tree-type structure. The listings under the **definitions** branch also contain information on transmission system wires, controls, panels and curves.



You can expand and detract each branch by clicking on either the [+] or the [-] symbols respectively.

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 organised according to their hierarchy, which helps to simplify navigation, as well as provides a complete overview of the how the project is structured. Transmission lines and cables will appear inside their parent module.

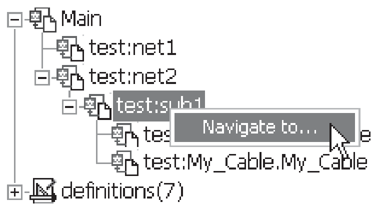


For example, the case project above (with namespace name *test*) contains a main page with two modules called *net1* and *net2*. The module *net1* contains another called *sub1*. The display format of the names in this branch is as follows:

```
<namespace_name>:<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 tlines/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. *My_TLine* and *My_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 Circuit 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
- Transmission Line or Cable Instance

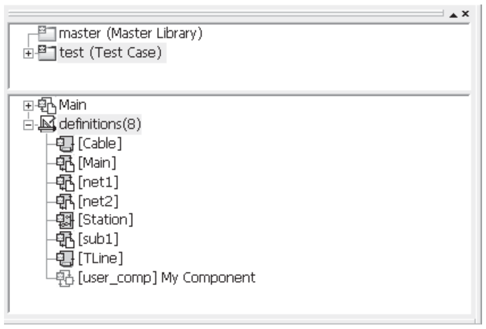
The instance that is currently being viewed will appear blue.

Definitions Branch

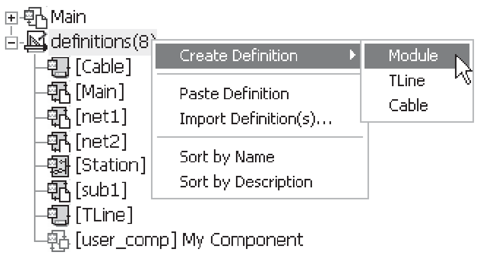
The definitions branch contains a list of all definitions that are stored within that particular project: Definitions for component or module instances that exist in a project, but are stored in other projects (ex. master library components), will not appear in this local definitions branch.

The image below shows the list of component definitions for the same case project discussed in the previous section. In addition to the definitions of each module, the existence of a component definition (called *user_comp*) is also indicated.

It is not usually a good idea to store component definitions within case projects. All component definitions should exist exclusively in library projects to avoid having to maintain multiple versions of the same definitions.



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

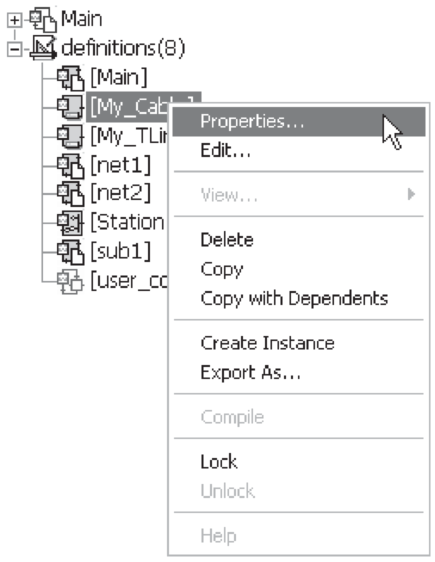


The following list describes the functions of this pop-up menu:

- **Create Definition**: This menu item contains a sub-menu list that allows you to specifically create either **Module**, **TLine** or **Cable** definitions. The definition will appear as *Untitled* at the bottom of the definitions list once created. The definition will need to be instantiated in order to place it on the Circuit canvas. See *Create Instance* menu function described below for more details.
- **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 Definition* 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.

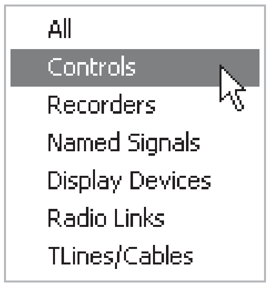
- **Sort by Name**: Select this option to sort the definitions list by name. Continually selecting this option will toggle the alphabetical order from A to Z or Z to A.
- **Sort by Description**: Select this option to sort the definitions list by description. Continually selecting this option will toggle 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 Properties* dialog window for editing the definition Name and Description. See *Editing Definition Properties* in the next chapter for more details.
- **Edit ...**: Select this option to edit the definition of the component (opens the Graphic window for components, the Circuit window for modules or the Editor window for transmission lines and cables). See *Editing a Component or Module Definition* in chapter 9 of this manual 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 the next section *Runtime Objects in the Definitions Branch* for more details.



- **Delete:** Select this option to delete the definition (and all existing instances of course!).
- **Copy:** Select this option to copy the definition.
- **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 Circuit 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. Drag and Drop may also be used to perform this feature.
- **Export As...**: This option brings up the **Export 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.
- **Lock:** Select this option to encrypt and password protect a module definition.
- **Unlock:** Select this option to decrypt a module definition. You will need to enter a password. See *Module Locking* in chapter 11 of this manual for more details.

Icons are included in this branch for visual distinctness:

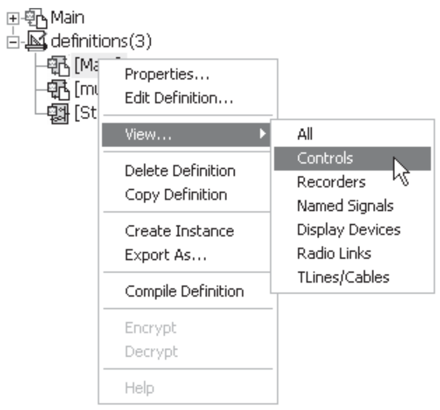
- Component Definition
- Module Definition
- Transmission Line or Cable Definition

A double-click on a module definition will bring you directly to its definition canvas in the *Circuit* window, 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.

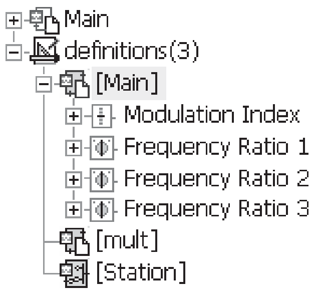
Runtime Objects in the Definitions Branch

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



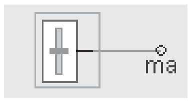
Resulting List of Controls-Type Runtime Objects in definition *Main*

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.

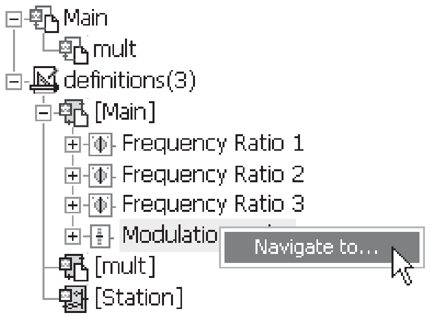
Runtime objects can be categorized into several main groups:

- 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 Circuit 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

<u>Transmission Lines and Cables</u>
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.

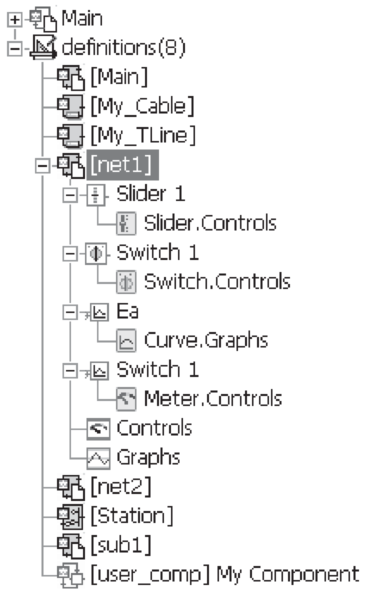
If the line is an overhead line with Termination Style in **Remote Ends** mode, then a [+] box will appear beside the branch name. Expanding this branch (press the [+] symbol) will show the two interface components representing both ends of the transmission line. Note that cables will always have remote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

TLine/Cable Interface Components:

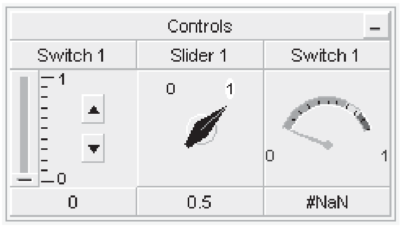
- Remote End Interface Component Instance

<u>Observers</u>
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 Circuit view:



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

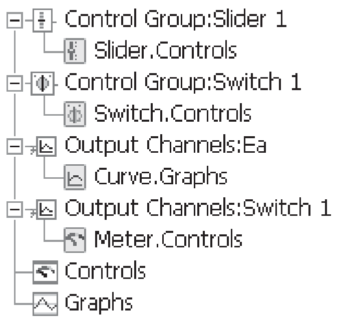
Recorders:

- 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:



```
Control Group:Slider 1
    Slider.Controls
Control Group:Switch 1
    Switch.Controls
Output Channels:Ea
    Curve.Graphs
Output Channels:Switch 1
    Meter.Controls
Controls
Graphs
```

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* in Chapter 6.

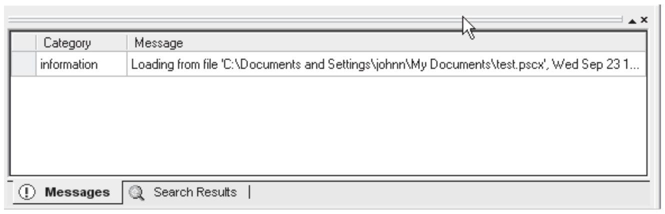
THE OUTPUT WINDOW

The primary purpose of the output window 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 here. In the *Messages* section, the messages are divided into *build*, *runtime* and *information* categories.

The output window will also provide the results of a project search, using the *Search* utility, placed under the *Search Results* tab.

Moving and Resizing the Output Window

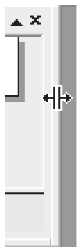
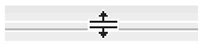
The output window may be moved and dropped anywhere in the environment. To do this, click and hold the left mouse button with the mouse pointer directly over the top window bar.



Category	Message
information	Loading from file 'C:\Documents and Settings\johnn\My Documents\test.pscx', Wed Sep 23 1...

Drag the window to where you want it placed (you should see a boxed outline of the window) and let go of the mouse button. Depending on where the window outline is when the mouse button is released, the window will either dock itself into position or appear as a floating window.

The output window can be resized by moving the mouse pointer over one of the four window edges until the pointer changes that shown below:

 or 

Click and hold the left mouse button, and drag the mouse pointer to resize in either the horizontal or vertical direction.

Errors and Warnings

A distinction can be made between error and warning messages simply by the colour of the symbol preceding the message. The colour code is as follows:

- ⓘ Warning (yellow)
- ● Error (red)

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 (i.e. a mistakenly disconnected node, for instance). It is therefore important to study the output window every time the project is built and run if there are any warning messages.

If an error is reported, the simulation build or run 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.

Build Messages

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 *Common Output Window Messages* for more information.

Any warning or error messages defined in the Checks segment of any component definition will be displayed as a build message.

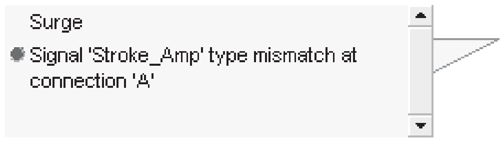
Runtime Messages

Runtime messages provide error and warning messages related to the simulation run -that is, messages from 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 *Common Output Window Messages* for more information.

Locating the Problem Source

The output window provides an easy method of locating the source of any displayed message: Either right-click the message itself and select **Navigate to...**, or simply double-click the message. PSCAD will then automatically open the source page in Circuit view and point directly at the problem with a message box:

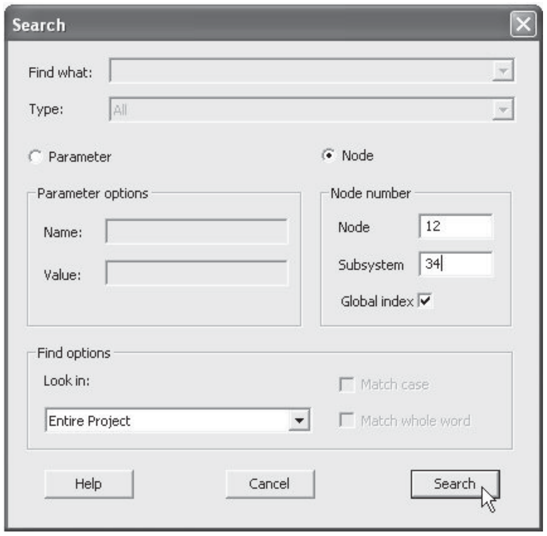


Surge

Signal 'Stroke_Amp' type mismatch at connection 'A'

Search

If a message indicates that the problem is arising at a particular subsystem and node, then you can utilize the Search utility to search for this exact location. Simply invoke the Search dialog (**ctrl + f**) and in the select **Node**, enter the **Subsystem** and **Node** indicated in the output window message and select the **Search** button.



Search

Find what:

Type: All

Parameter [✓ Node]

Parameter options

Name:

Value:

Node number

Node: 12

Subsystem: 34

Global index: ✓

Find options

Look in: Entire Project

Match case

Match whole word

The feedback from this search will be displayed in the *Search Results* section of the output window. Simply double-click the search feedback messages and PSCAD will automatically open the source page in Circuit view and point directly at the problem (with the same arrow as described above).

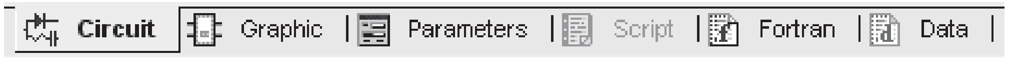
THE DEFINITION EDITOR

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. Circuit view), component graphic design, as well as scripting.

The definition editor window is invoked (in Circuit view by default) by either a left double-click on a project in the workspace window, or by right clicking on the project and selecting **Open**.

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.

Circuit

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

While the Circuit window is open, the Control Palette and Electrical Palette toolbars are enabled.

Graphic

This Graphic tab is used for editing the graphics of a definition.

Parameters

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

Script

This Script tab 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 Circuit 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 Circuit window. For example, if you are viewing the main page of a project in Circuit view, the Data window (click the **Data** tab) will show the EMTDC input data for the main page.

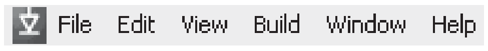
OTHER IMPORTANT FEATURES

Menus

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

Main Menu Bar

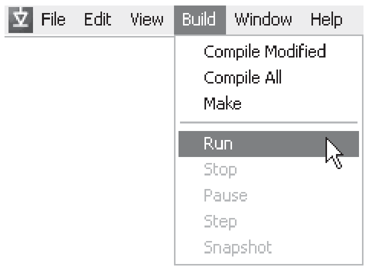
The main menu bar contains an assortment of the most common commands. It is located near the top of the environment, appearing as shown below:



The main menu bar contains only some of the standard functions. There are many others available using right-click pop-up menus.

This bar may appear differently depending on what section of the environment you are currently in.

To select a menu item, move the mouse pointer onto that item and then click the left mouse button. The following menu shows how to create a new project using the *Build* menu.



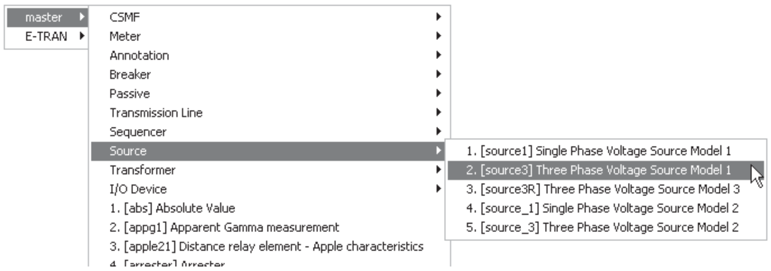
Right-Click Pop-Up Menus

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 Circuit view.

It is a good idea 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 menus will display the actual filename of the component.


master
E-TRAN
CSMF
Meter
Annotation
Breaker
Passive
Transmission Line
Sequencer
Source
Transformer
I/O Device
1. [abs] Absolute Value
2. [appg1] Apparent Gamma measurement
3. [apple21] Distance relay element - Apple characteristics
1. [source1] Single Phase Voltage Source Model 1
2. [source3] Three Phase Voltage Source Model 1
3. [source3R] Three Phase Voltage Source Model 3
4. [source_1] Single Phase Voltage Source Model 2
5. [source_3] Three Phase Voltage Source Model 2

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 towards the *Three Phase Voltage Source Model 1* component in the Sources module, in the master library. A left-click at this point will add this component to whichever canvas is open in Circuit view.

In general, library pop-up menus construct a system of menus according to the hierarchal 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 properties editor. See *Editing Definition Properties* in the next chapter for details.

Toolbars

Toolbars are useful in any program to provide a quick and easy method to access menu functions. The application includes multiple toolbars, which contain the most commonly used menu tasks.

The drawing related toolbars allow you to add the most common drawing objects and components by simply using the left mouse button: A single click on the toolbar to select an object, and a single click to add the object to the drawing canvas.

Main Toolbar

The main toolbar resides near the top of the environment. This bar primarily contains tasks found in the main menu bar, but also includes others, as well. The individual toolbar buttons are listed below with a short description:

Button	Description
🗋	Creates a new case project (*.pscx file)
📂	Loads a case project (*.pscx file)
💾	Saves changes to the active project (*.pscx file)
🖨	Print
✂	Cuts the current selection
⎘	Copies the current selection
📋	Pastes from the clipboard
↶	Undo (select the down arrow for history)
↷	Redo (select the down arrow for history)
🔍	Zoom rectangle
⛶	Zoom extents
🔍	Zoom in one step
🔍	Zoom out one step

150%	Zoom control list box
	Pan mode
	Navigate backward (select the down arrow for history)
	Navigate forward (select the down arrow for history)
	Up to parent canvas
	Invoke wire mode
	Create a new definition
	Create new transmission line
	Create new cable
	Launches search utility

Status Bar

The status bar is located at the very bottom of the application, and is not actually a toolbar, but used for the purpose of display only (does not contain any command buttons). Most of the time, the status bar is used to monitor the status of a simulation build, link and run, and will display the percentage complete and the simulation time during a run.

Rotation Bar

The individual rotation bar buttons are listed below with a short description:

Button	Description
	Rotate selection counter-clockwise
	Rotate selection clockwise
	Mirror selection
	Flip Selection

Runtime Bar

The individual runtime bar buttons are listed below with a short description:

Button	Description
f	Compile (build) modified modules (active project only)
⇩	Compile (build) entire project (active project only)
0101 1001	Make project (active project only)
▶	Run simulation (active project only)
●	Stop simulation (active project only)
⏸	Pause simulation (active project only)
⏯	Advances run by one time step (while pause is invoked)
📷	Take a snapshot
50.0 ▾	Change plot step
▤	Scenarios menu button
<Default> ▾	Scenarios template list

Electrical Palette

The electrical palette contains common electrical components used when building circuits in Circuit view. Most of these button functions can also be found under the **Add Component** menu item in the Circuit view pop-up menu. The individual toolbar buttons are listed below with a short description:

Button	Description
—	Add wire
-/\/\/\-	Add resistor
∿∿∿	Add inductor
⊣⊢	Add capacitor
⏚	Add ground

	Add node label
	Add external node (Xnode)
	Add breakout
	Add pin
	Add current meter
	Add voltmeter
	Add voltmeter with ground
	Add multimeter
	Add a t-line interface
	Add a cable interface

Control Palette

The control palette contains common control components used when building circuits in Circuit view. Most of these button functions can also be found under the **Add Component** menu item in the Circuit view pop-up menu. The individual toolbar buttons are listed below with a short description:

Button	Description
	Add data tap
	Add data merge
	Add data label
	Add integer constant
	Add real constant
	Add import
	Add export
	Add radio link

	Add output channel
	Add slider
	Add switch
	Add dial
	Add pushbutton
	Add graph frame
	Add xy plot
	Add control panel
ab	Add annotation box
🗨	Add sticky note
—	Add divider

Graphic/Script Bar

The graphic/script toolbar contains functions used only when editing a definition in either the Graphic section or the Script section. The individual toolbar buttons are listed below with a short description:

Button	Description
	Set layers
	Show/hide all layers
╲	Add line
□	Add rectangle
⬭	Add ellipse
◜	Add arc (90°)
◠	Add arc (180°)
A	Add text label

	Add connection
■	Default line colour
0.2 pt	Default line weight
—	Default line style
Transparent	Default fill style
Configuration	Navigate segments drop list
	Invoke segment manager
	Search for text
	Search for and replace text

Graphic Filter Bar

The Graphic Filter toolbar is used specifically within the Graphic section. These buttons provide a quick means to show or hide certain types of graphic objects, so as to create a less cluttered environment in which to work.

Button	Description
	Show/hide ellipses
	Show/hide arcs
	Show/hide lines
	Show/hide rectangles
A	Show/hide text
	Show/hide connections
	Show/hide all graphics

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 *Cut/Copy/Paste keys* in the *Workspace Options* is set to *x,c,v*. See *Environment* in Chapter 3 of this manual for more.

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 **Assign Sequence Numbers Automatically** must be enabled in the Canvas Settings dialog.
Ctrl + a	Sellect all
Ctrl + z	Undo
Ctrl + y	Redo
Ctrl + f	Search utility
Ctrl + o	Load project
Ctrl + n	New project
Ctrl + s	Save active project

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

Wires

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

Shortcut	Description
v	Insert a wire vertex
i	Reverse wire vertexes
d	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.

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

w	Show y-intercept
c	Show cross-hairs (follows curve traces)
Space Bar	Switch curves while in cross-hair mode
← →	Graph frame dynamic aperture adjustment

PSCAD TEMPORARY DIRECTORIES

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 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**: *.gf42
- **Compaq Visual Fortran 6**: *.cf6
- **Intel Visual Fortran 9 to 11**: *.if9

All files located in the temporary directory can be viewed in an organized manner in the *Files* branch in the primary workspace window. 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 *Files* branch, will be those from the **.if9* temporary folder. Same goes for cleaning the temporary folder.

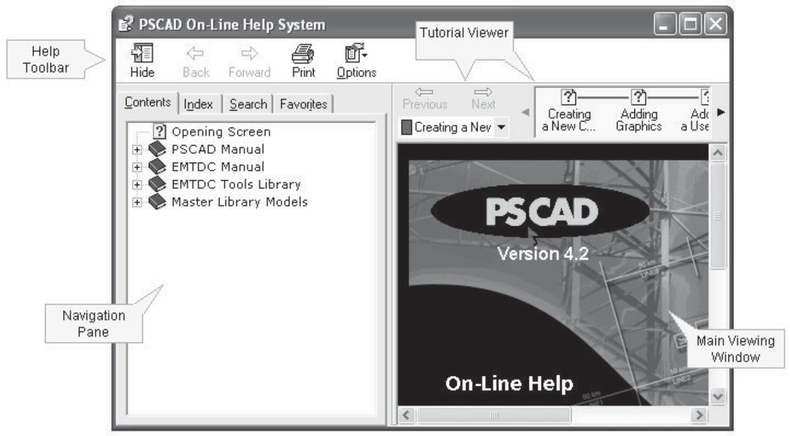
PSCAD ON-LINE HELP SYSTEM

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:

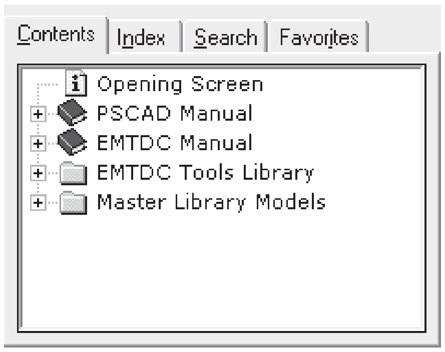
Button	Description
Hide	Hides navigation pane
Back	Moves backward in the viewed topic list
Forward	Moves forward in the viewed topic list
Print	Prints currently viewed topic
Options	Provides some options to allow the user to customize the online help viewer

The Navigation Pane

The navigation pane is separated into four sections: The *Table of Contents (TOC)*, the *Index*, the *Search* engine and the *Favourites* 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.



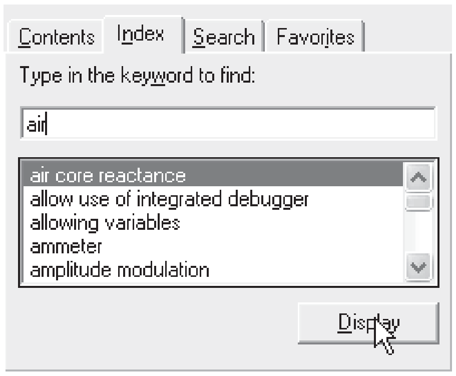
There are various icons associated with the TOC, as described below:

- **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.



Type in the keyword to find:

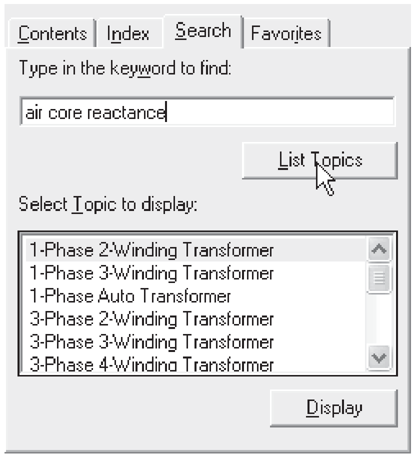
air

air core reactance
allow use of integrated debugger
allowing variables
ammeter
amplitude modulation

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.

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



Type in the keyword to find:

air core reactance

Select Topic to display:

1-Phase 2-Winding Transformer
1-Phase 3-Winding Transformer
1-Phase Auto Transformer
3-Phase 2-Winding Transformer
3-Phase 3-Winding Transformer
3-Phase 4-Winding Transformer

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

Favourites

The Favourites section allows users to bookmark their favourite help pages. To add a favourite topic to the favourites list, simply left click the **Add** button. To remove a topic from Favourites, select it and left click the **Remove** button. To display a topic from the favourites 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.psc	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.psc	Shows the use of Fast Fourier analysis component to perform online fft on signals.

Simple AC System with a Transmission Line

simpleac.psc simpleac_sld1.psc simpleac_sld2.psc	A simple AC system with transmission lines. Introduces transformers, transmission lines, and the concept of subsystems in PSCAD.

Use of Control Arrays

pagearray.psc	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.psc	Shows the use of dynamic input devices: Slider, Switch, Push Button and the Dial.

Interpolation

interpolation.psc	A simple case illustrating the use of interpolation.

Generating a Legend and using PSCAD Macros

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

Chatter Elimination

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

Multiple Run

multirun.psc multirun_sld.psc	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 providing 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.

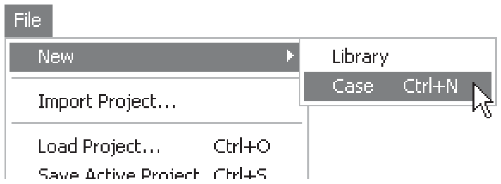
This chapter includes a tutorial entitled *Creating a New Project*, which is designed to jump start the learning process and to get you using the application. If you have not done so already, try the tutorial entitled *My First Simulation* in Chapter 4 of this manual as well.

PROJECTS

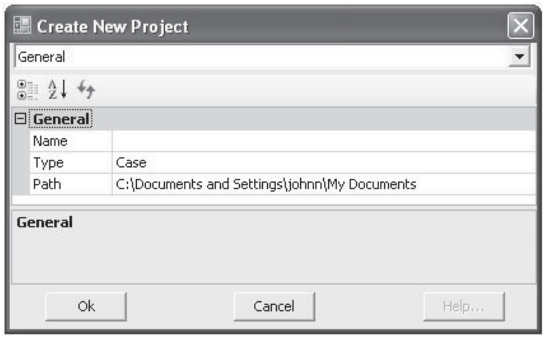
The following topics focus on the usage and manipulation of case and library projects.

Creating a New Project

Click the left mouse button on the **File** item in the main menu bar and select **New**. A secondary menu should open listing a choice of either a **Library** project or a **Case** project.



Select the desired project type to open the *Create New Project* dialog.



General

Name	
Type	Case
Path	C:\Documents and Settings\johnn\My Documents

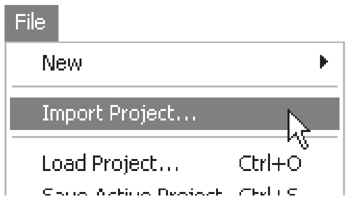
General

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 be become both the *namespace* name and the file name of the project. Either of these names can be modified after the project is created. 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 unique name for the project in the **Name** field. A new project entitled *<name> - <name>* will appear in the workspace window. You can also invoke the *Create New Project* dialog by simply pressing the **New Project** button in the main toolbar, or by pressing **Ctrl + N** on your keyboard.

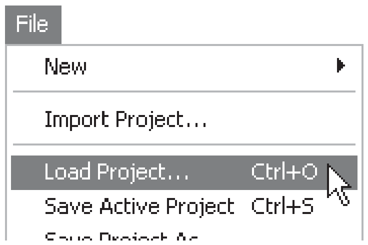
Importing a Project
Click the left mouse button on the **File** item in the main menu bar and select **Import Project...**.



The *Import Project* dialog window will open with a default **Files of type** as **All v4.1 or v4.2 Projects (*.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 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 chapter 13 of this manual.

Loading a Project
Click the left mouse button on the **File** item in the main menu bar and select **Load Project...**.



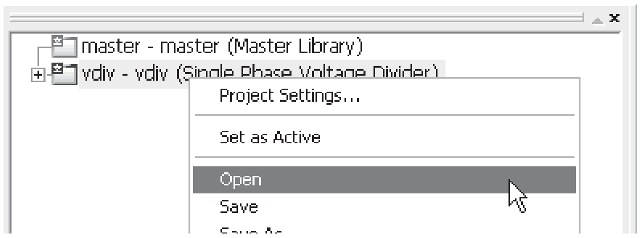
The *Load Project* dialog window will open with a default **Files of type** as **All Projects (*.pscx, *.pslx)**. Navigate to the desired project and select it so that its name appears in the **File name** field. Click the **Open** button to open the project. The project name should appear in the workspace window, indicating that the project was loaded.

You can also bring up the *Load Project* dialog by simply pressing the **Load Project** button in the main toolbar, or by pressing **Ctrl + O** on your keyboard.

Recent Files
You can load projects that have been loaded previously by accessing the **Files** | **Recent Files** sub-menu. Hold the mouse pointer over the **Recent Files** item in the **File** menu and a list of previously loaded and <u>saved</u> projects will appear. Select the desired project from the list to load it.

The number of projects listed can be adjusted up to 16 in the *Projects* section of the *Workspace Options* dialog. See *Projects* in chapter 3 for details.

Opening and Viewing a Project
To open a project for viewing, right-click on the project in the workspace window and select **Open** from the pop-up menu. A **left double-click on the project name** in the workspace window will perform the same operation.


master - master (Master Library)
vdiv - vdiv (Single Phase Voltage Divider)
Project Settings...
Set as Active
Open
Save

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

Navigating an Opened 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 Circuit view.

Panning (Dynamic Scroll) Mode

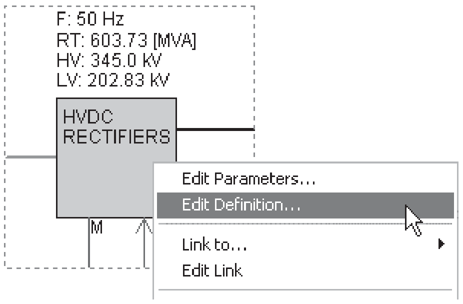
The panning feature allows you to scroll through the *Circuit* or *Graphic* tab windows in a fluid motion. You can invoke the panning mode by one of the following methods:

- 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 though the page.

- Press the **Pan** button in the main toolbar to invoke panning mode. To indicate that you are in panning mode, the mouse pointer will change into a hand shape. Press **Esc** or press the **Pan** button again to exit from panning mode.

Moving In and Out of Modules

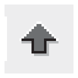
Opening the Circuit canvas of a module component (i.e. moving into) is analogous to editing the definition of the module. Therefore, right-click on the module and select **Edit Definition...**.



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

- **Double-Click**: If you want to use the old method of navigating through module components, set the *Navigate into a module* 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 *Navigate into a module* 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 on the main toolbar.



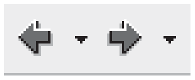
Up to Parent Canvas Button

Instead of the **Up to parent canvas** button, you can also use the workspace secondary window to specifically select a module to navigate to. See *The Secondary Window* in chapter 4 for more.

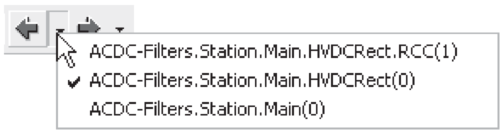
Forward/Back Buttons

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 on the main toolbar. If you cannot see this bar, select **View | Main Toolbar** in the main menu.



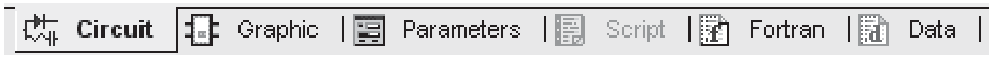
Once a navigational history is started, you may then use these buttons to navigate into and out of modules, or to jump directly to previously viewed canvases. To access the navigational history, press the down arrow on either button as shown below:


ACDC-Filters.Station.Main.HVDCRect.RCC(1)
ACDC-Filters.Station.Main.HVDCRect(0)
ACDC-Filters.Station.Main(0)

These functions are also included in the **View** menu.

Tabs

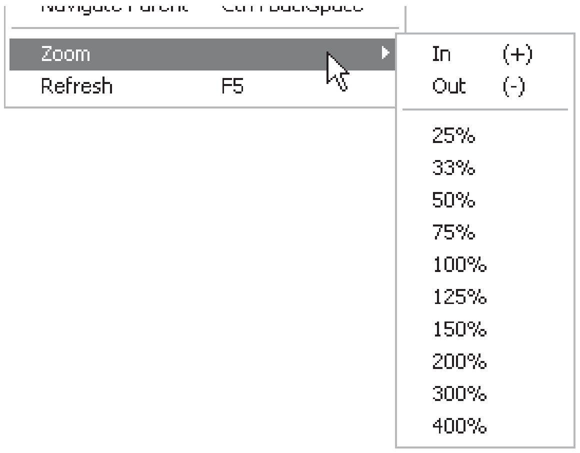
Tabs allow you to jump from one viewing area to another. There is a tab bar included at the bottom of some windows (as shown below), including the *Output Window* and others.



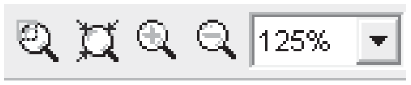
Zoom

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

- From the main menu bar, select **View** | **Zoom**. You then have a choice to select either **In**, **Out** or a specified zoom percentage.



- From the main toolbar, 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 main toolbar. Click this button to enter zoom rectangle mode. On the Circuit canvas, **hold down the left-mouse button** and drag the pointer to create the desired rectangle. Release the mouse button to zoom.

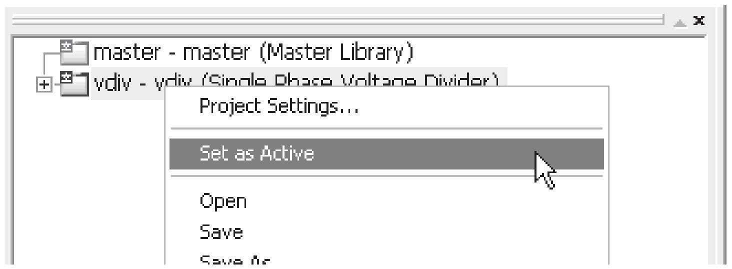
Zoom Extents
The **Zoom Extents** button is located in the main toolbar. Click this button to zoom to the extents of the currently viewed Circuit canvas.

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

- From the main menu bar, select **View** | **Refresh**.
- Press the **F5** key on your keyboard.
- Right-click on a blank part of the page and select **Refresh**.

Setting the Active Project
When there is more than one project loaded in the workspace, PSCAD needs to know which project to run when a simulation is requested (i.e. **Run** or **Compile** button is pressed). This is accomplished by setting the *Active Project*.

In the workspace window, right-click on the title of an inactive project and select **Set as Active**.


master - master (Master Library)
vdiv - vdiv (Single Phase Voltage Divider)
Project Settings...
Set as Active
Open
Save
Save As

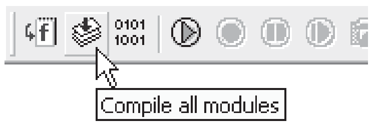
Compiling and Building a Project
Before compiling and building a project, ensure that the desired project is the *Active Project* in the workspace window.

Running a project without compiling is perfectly fine. If a **Run** command is issued, PSCAD will perform the **Compile modified modules** and then the **Make** procedure before the run starts. See *Running a Simulation* in this chapter for more details.

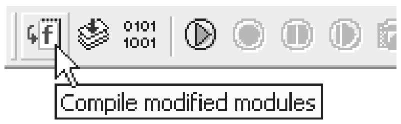
Compile

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

To compile the entire project from scratch, select the **Compile all modules** button in the main toolbar:

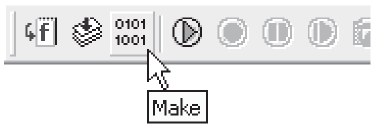


To compile the only modules and transmission segments that have been modified since the last compile, select the **Compile modified modules** button in the main toolbar:



Make

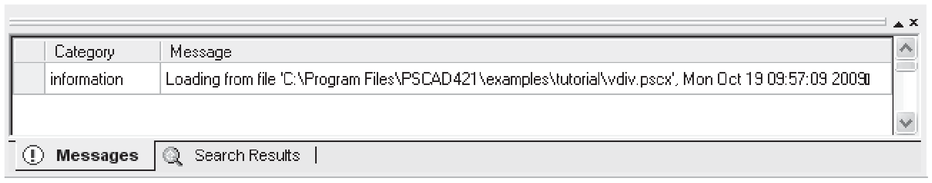
Making the project refers to the assembly of the associated executable file (*.exe) to be run by EMTDC. To make your project, press the **Make** button.



You may also perform these functions by selecting **Build** | **Compile Modified**, **Compile All** or **Make** in the main menu.

Viewing Error and Warning Messages

If a problem is detected when a simulation is run, all error and warning messages will appear in the *Output Window*.

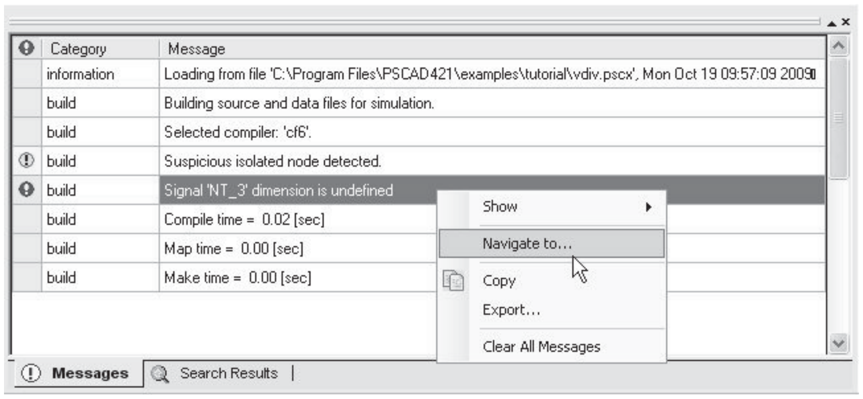


Category	Message
information	Loading from file 'C:\Program Files\PSCAD421\examples\tutorial\vdiv.pscx', Mon Oct 19 09:57:09 2009

Output window messages are divided into three main groups: *Build, Runtime* and *Information*. Compilation messages will appear as *Build*, while EMTDC runtime messages will appear as *Runtime*. The *Search Results* area is used to view results from a search (see *Searching* later in this chapter).

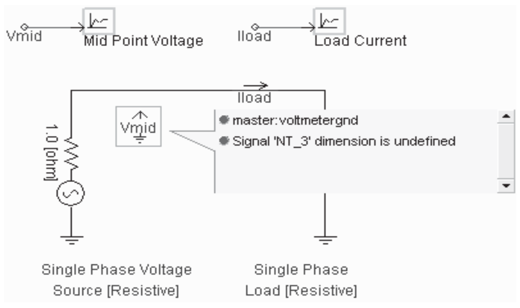
For more on this, see *Error and Warning Messages* in Chapter 11 of this manual.

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 double-click** or right-click and select **Navigate to...**. The problem source will be highlighted in the Circuit window.



	Category	Message
	information	Loading from file 'C:\Program Files\PSCAD421\examples\tutorial\vdiv.pscx', Mon Oct 19 09:57:09 2009
	build	Building source and data files for simulation.
	build	Selected compiler: 'cf6'.
	build	Suspicious isolated node detected.
	build	Signal 'NT_3' dimension is undefined
	build	Compile time = 0.02 [sec]
	build	Map time = 0.00 [sec]
	build	Make time = 0.00 [sec]

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



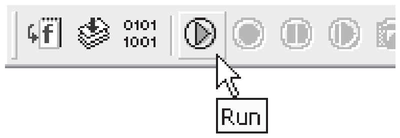
master:voltmetergnd
Signal 'NT_3' dimension is undefined

Vmid
Mid Point Voltage
Iload
Load Current
Iload
Vmid
1.0 [ohm]
Single Phase Voltage Source [Resistive]
Single Phase Load [Resistive]

Running a Simulation
Before running a simulation, ensure that the desired project is the *Active Project* in the workspace window.

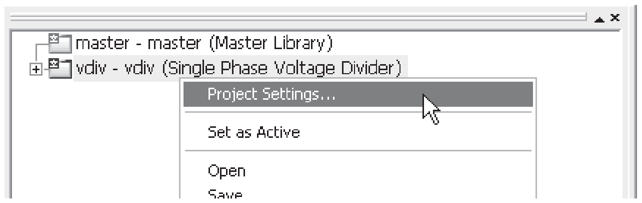
Use either of the following methods to run a project simulation:

- Press the **Run** button in the main toolbar.

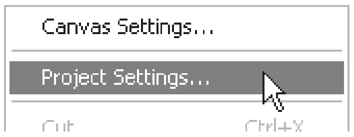


- In the main menu, select **Build** | **Run**.

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

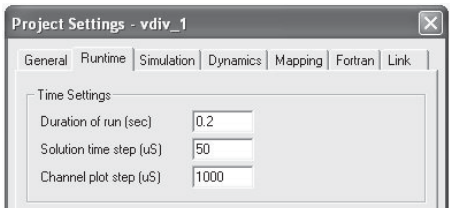

master - master (Master Library)
vdiv - vdiv (Single Phase Voltage Divider)
Project Settings...
Set as Active
Open
Save

Also, you can right-click over a blank area of any project page in Circuit 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.

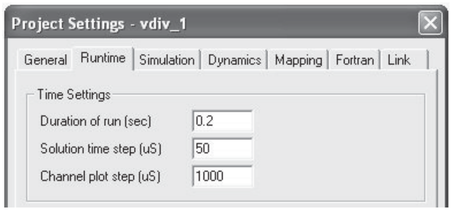
Changing the Run Duration
Open the *Project Settings* dialog window as described in *Editing Project Settings* above. Click the **Runtime** tab and, in the **Time Settings** area, enter a new **Duration of Run** time in seconds.



Duration of run (sec) 0.2
Solution time step (uS) 50
Channel plot step (uS) 1000

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 (μs).


Project Settings - vdiv_1

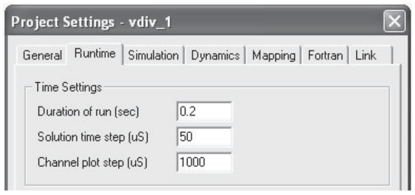
Time Settings

Duration of run (sec) 0.2

Solution time step (uS) 50

Channel plot step (uS) 1000

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 (μs).


Project Settings - vdiv_1

Time Settings

Duration of run (sec) 0.2

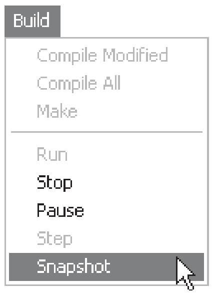
Solution time step (uS) 50

Channel plot step (uS) 1000

The plot step may also be varied through the **Plot Step drop list** in the runtime bar. Plot step can be changed at any point during a simulation.

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 Chapter 2 of the EMTDC manual for more details on snapshot files.

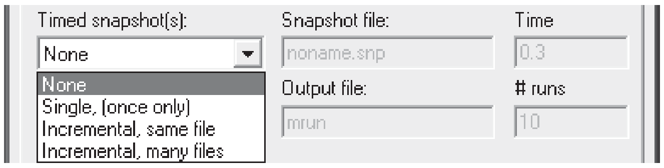
Manual
You can take a snapshot manually by either pressing the **Snapshot** button on the main toolbar, or selecting **Build** | **Snapshot** from the main menu bar (while the simulation is running).



Pre-Defined

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



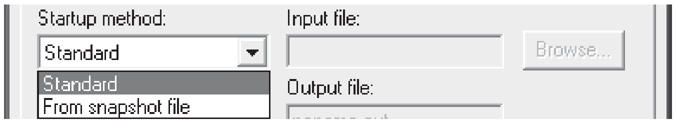
Make sure that the run duration is greater than the snapshot time, or no snapshot file will be created!

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.

Starting from a Snapshot

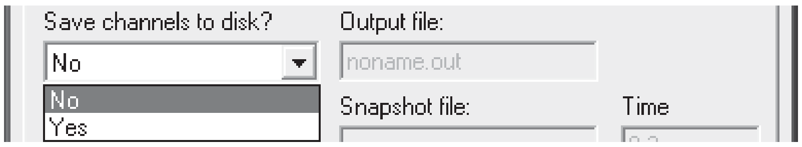
Open the *Project Settings* dialog window as described in *Editing Project Settings* above. In the **Runtime** section of the dialog, choose **From snapshot file** in the **Startup method** drop list. Enter a name for the snapshot file in the **Input File** field or use the **Browse...** button to select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



Click the **OK** button to exit the dialog and save changes. Run the project and it should start initialized from the snapshot time. See *Initialization and Initial Conditions* in Chapter 2 of the EMTDC manual for more details on snapshot files.

Saving Output to File

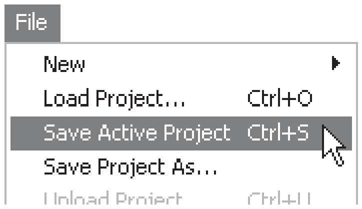
Open the *Project Settings* dialog window as described in *Editing Project Settings* above. In the **Runtime** section of the dialog, choose **Yes** from the **Save channels to disk?** drop list. Enter the name for the output file in the **Output File** field.



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.

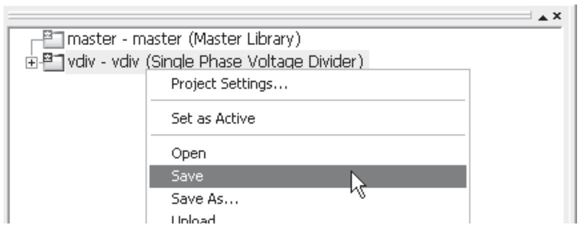
Saving Project Changes

To save the active project, click **File** in the main menu bar. Select **Save Active Project...**.



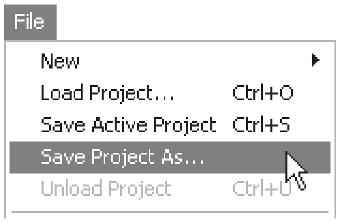
You can also save the active project by pressing the **Save Active Project** button in the main toolbar, or by pressing **Ctrl + S** on your keyboard.

To save a specific project (either active or inactive), right-click on the project title within the workspace window, and select **Save** from the pop-up menu.

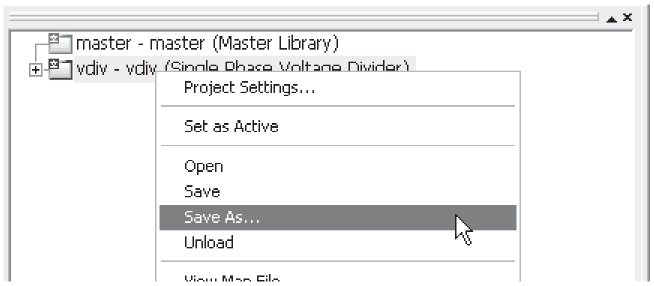

master - master (Master Library)
vdiv - vdiv (Single Phase Voltage Divider)
Project Settings...
Set as Active
Open
Save
Save As...
Unload

Save Project As...

To save the active project as another file, click **File** in the main menu bar. Select **Save Project As...**.



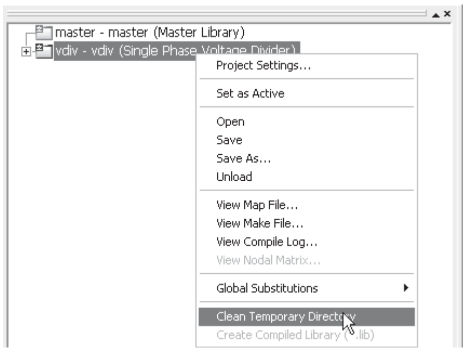
To save a specific project (either active or inactive) as another file, right-click on the project title in the workspace window and select **Save As...** from the pop-up menu.


master - master (Master Library)
vdiv - vdiv (Single Phase Voltage Divider)
Project Settings...
Set as Active
Open
Save
Save As...
Unload
View Map File

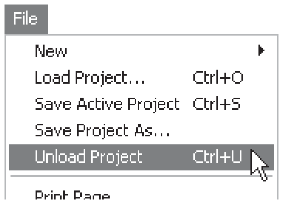
In either case, the *Save As...* dialog window will open with a default **Save as type** as **Project File (*.pscx)** or **Project File (*.pslx)** for case or library project respectively. Change the file name of the project in the **File name** field.

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.

Cleaning the Temporary Directory
To delete all temporary project files (i.e. compilation, make, data, etc. files), right-click on the project title in the workspace window, and select **Clean Temporary Directory** from the pop-up menu. See *PSCAD Temporary Directories* in Chapter 4 of this manual for more details.



Unloading a Project
In the workspace window, select the project that is to be unloaded. Click **File** in the main menu bar and select **Unload Project**.



You can 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.

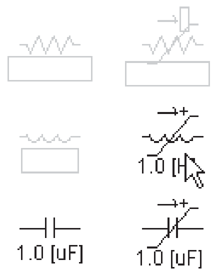
COMPONENTS AND MODULES

The following topics focus on the usage and manipulation of component and module objects.

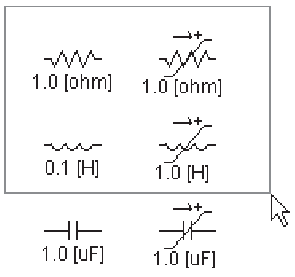
Selecting Objects

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.



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

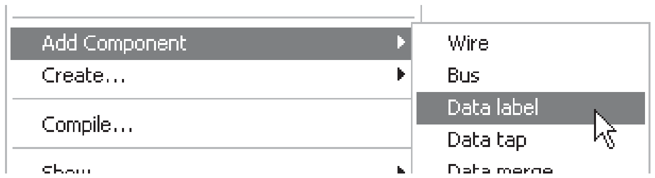


To ungroup any objects, hold the **Ctrl** key and then select (**left-click**) all objects to be ungrouped.

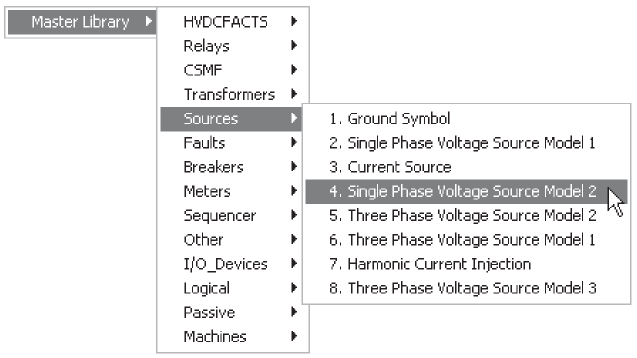
Adding Components to a Project
There are a variety of ways to insert components onto the Circuit canvas. Before proceeding, ensure that you are viewing the desired page in the Circuit window.

- **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**).

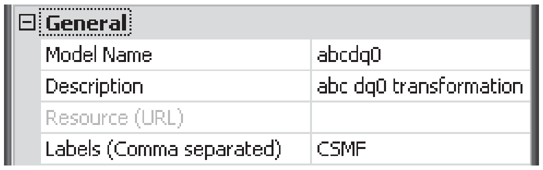
- **Right-Click Menu:** Right-click over a blank area of the page and select **Add Component**. A sub-menu will appear containing the most commonly used components from the master library. Select a component and it will be automatically added.



- **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.


Master Library
HVDCFACTS
Relays
CSMF
Transformers
Sources
Faults
Breakers
Meters
Sequencer
Other
I/O_Devices
Logical
Passive
Machines
1. Ground Symbol
2. Single Phase Voltage Source Model 1
3. Current Source
4. Single Phase Voltage Source Model 2
5. Three Phase Voltage Source Model 2
6. Three Phase Voltage Source Model 1
7. Harmonic Current Injection
8. Three Phase Voltage Source Model 3

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 properties are set as follows:

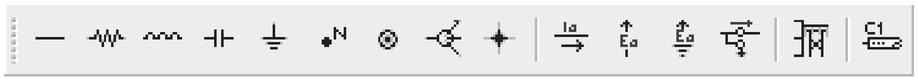


General	
Model Name	abcdq0
Description	abc dq0 transformation
Resource (URL)	
Labels (Comma separated)	CSMF

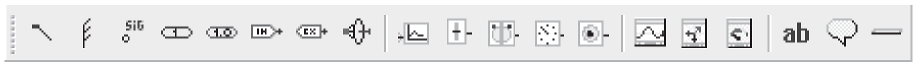
See *Editing Definition Properties* later in the chapter for more details.

- **Control and Electrical Palettes: Left-click** on any of the palette buttons and then move the mouse pointer over the Circuit 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.

The *Control and Electrical Palettes* are usually located to the right of the main canvas window. If you cannot see them, select **View | Control Palette** and/or **View | Electrical Palette** from the main menu bar.



Electrical Palette



Control Palette

Adding Multiple Instances of a Component

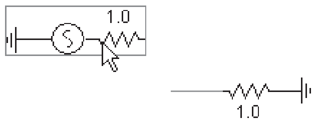
Multiple instances of the same component can be added when using the *Control* and *Electrical Palettes*:

1. **Hold down the Ctrl** key and **left-click** on the desired component from either the Control or the Electrical Palette.
2. **Continue holding the Ctrl key** and move the mouse pointer over the Circuit 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.

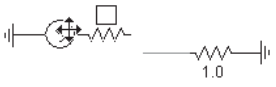
You may continue this process and add as many instances as desired, provided the **Ctrl** key remains depressed. To escape from this function, simply release the **Ctrl** key at any time.

Moving or Dragging an Object

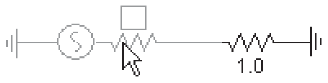
To move an object, place the mouse pointer over the object icon. Press and hold the left mouse button. Now drag the mouse to move the component. When you move a component, it will always snap to the nearest drawing grid, even if the grid dots are not visible.



Click the left mouse button and hold



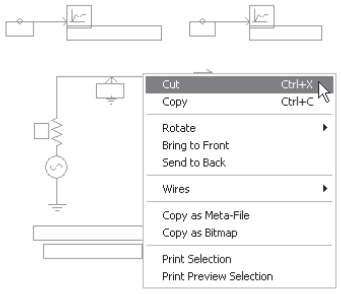
Drag the object



Release the left mouse button

Cut/Copy/Paste

Objects can be cut, copied and pasted as many times as desired within the Circuit 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.

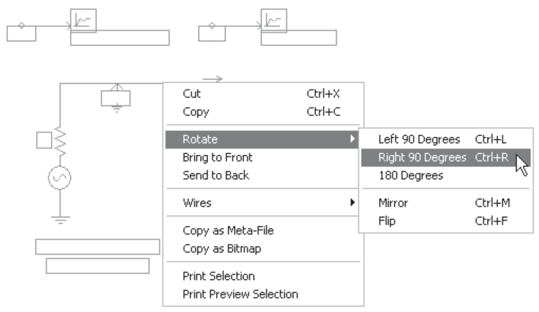


Once cut or copied, you can paste the object by right-clicking over a blank area of the Circuit canvas and selecting **Paste** (or press **Ctrl + v**).

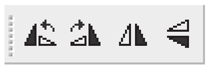
Rotate/Mirror/Flip

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.



- **Rotation Bar:** Select the object or group of objects and then press one of the four **Rotation Bar** buttons.



Rotation Bar

If you cannot see the rotation bar, select **View | Rotation Bar** from the main menu bar.

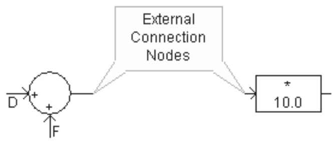
Deleting Objects
Select the object (or objects) and press the **Delete** key.

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 main toolbar, or press **Ctrl + z** or **Ctrl + y** respectively. The undo and redo features will store most manipulations and changes, however some limitations may apply.

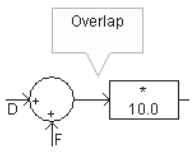
If this feature does not appear to be working, check the *Projects* section of the *Workspace Options* dialog to see if *Change tracking* is set to *Use Undo/Redo Stack* (see chapter 3 on accessing the *Workspace Options* dialog).

Connecting Components Together
A connection between components can be made in one of the following ways:

- Overlapping one component port connection over top another.



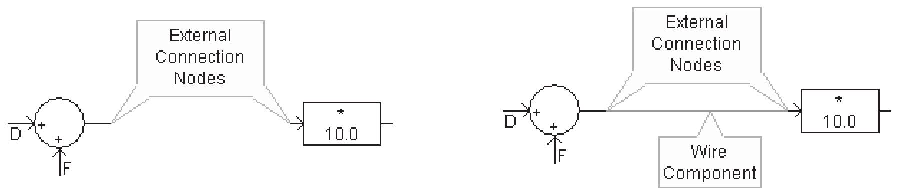
Not Connected



Connected

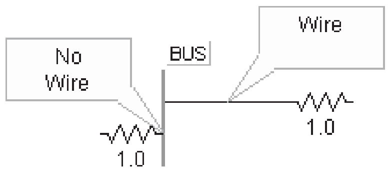
- The endpoint of a *Wire* component can be used to make contact with port connections, or the endpoint of another Wire:

See *Valid Connections* in the *Wire* help page for more on connecting objects.



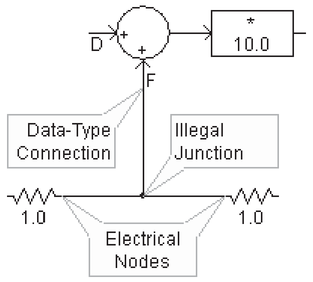
Not Connected Connected

- 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*:

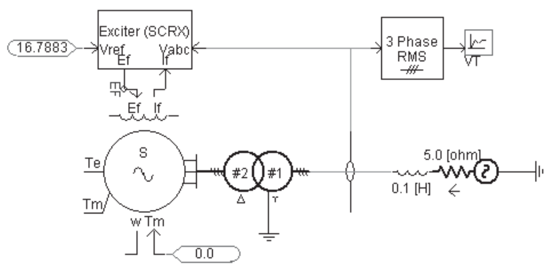


Invalid Connection

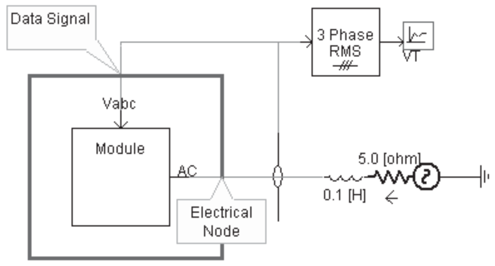
Wires are universal in that they assume the signal type of the nodes they are connected to. If a wire is connected to an electrical node, it will become an electrical wire, for example.

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. 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
- Input 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 appear as either a subroutine argument (if it is ported as an input 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

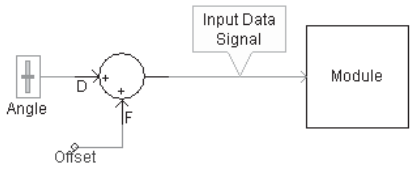
If you create a new module component by using the *Component Wizard*, the process of adding port connections is fully automated. See *Creating a New Component or Module* later in this section for more details.

If you are editing an existing module, follow these general steps:

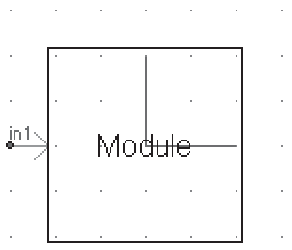
1. Edit the definition of the module. See *Editing a Component or Module Definition* in chapter 9 for more details.
2. Define an input or output port connection in the *Graphic* section (Graphic tab) of the module. See *The Graphics Section* in Chapter 9 of this manual for more details.
3. Add a corresponding *Import* or *Export* component on the module Circuit canvas. Each *Import* or *Export* must be named the same as its corresponding port connection.

EXAMPLE 5-1:

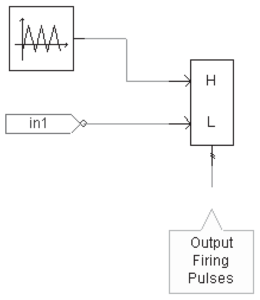
Consider a project that consists of a simple phase angle offset control being input into a module. The module *Graphic* section consists of a single port connection named *in1*, which is to be connected to the incoming data signal from its parent module. The input signal *in1* is used as a reference angle to a firing pulse generator inside the module definition Circuit canvas.



Parent Canvas Containing Module Component and Greater Circuit

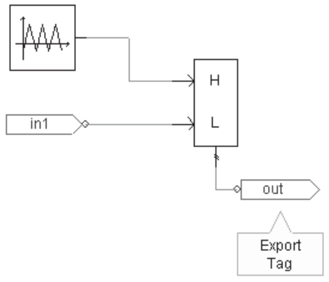


Graphic View of Module Definition

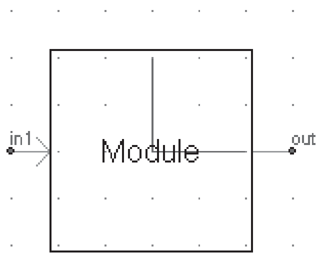


Circuit Canvas of Module

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.

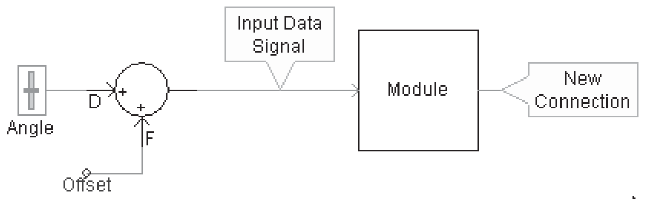


Added Export tag in Canvas of Module



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 input parameters for you when creating new components. You must therefore create a new module first (if necessary), before editing the definition and defining parameters.

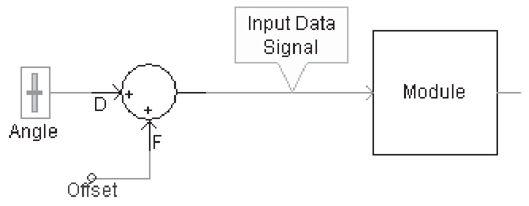
Also note that parameters may only be used for porting *into* the module. Porting data signals out of the module is supported only through port connections or *Radio Links*.

Follow these general steps:

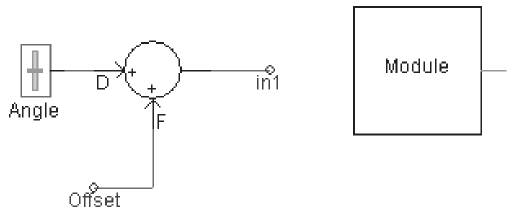
1. Edit the definition of the module. See *Editing a Component or Module Definition* in this section for more details.
2. Define an input parameter in the *Parameters* section (Parameter tab) of the module. See *The Parameters Section* in Chapter 9 of this manual for more details.
3. Add a corresponding *Import* component on the module Circuit canvas. Each Import must be named the same as its corresponding parameter.

EXAMPLE 5-2:

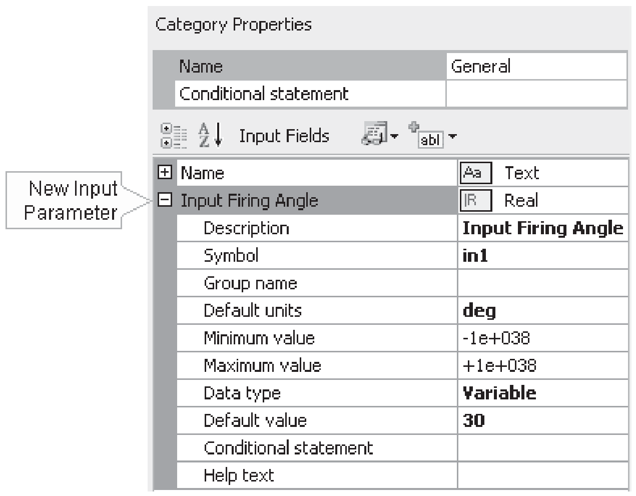
Consider the project outlined in *Example 5-1*.



This time, instead of using a port connection, the user wants to port the *Input Data Signal* through an input 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 an input parameter in the module definition *Parameters* section.



Data Signal Named and Port Connection Removed from Module



Category Properties

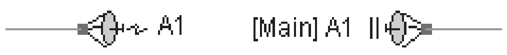
Name	General
Conditional statement	

Name	Text
Input Firing Angle	Real
Description	**Input Firing Angle**
Symbol	**in1**
Group name	
Default units	**deg**
Minimum value	-1e+038
Maximum value	+1e+038
Data type	**Variable**
Default value	**30**
Conditional statement	
Help text	

New Input Parameter in Parameters Section of Module Definition

Porting Data Signals by Radio Links

The aforementioned data signal porting can also be accomplished by means of *Radio Link* components.

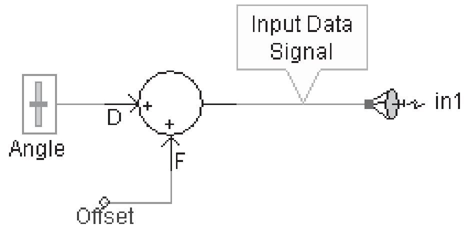


Radio Links offer a 'wireless' method of porting the signals; that is there is no need for 'hard-wired' port connections, input parameters and Import/Export components. In fact, *Radio Links* make it possible to port signals simultaneously to multiple module instances, at multiple layers in the hierarchy. 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.

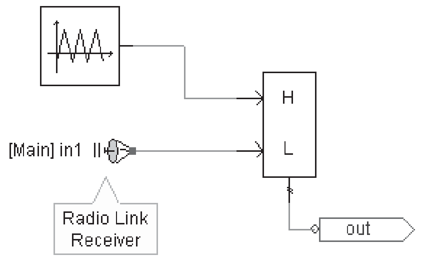
EXAMPLE 5-3:

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

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*.

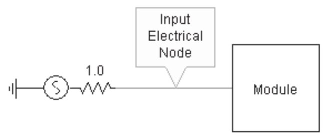
If you create a new module component by using the *Component Wizard*, the process of adding port connections and *XNodes* is fully automated. See *Creating a New Component or Module* later in this section for more details.

If you are editing an existing module, follow these general steps:

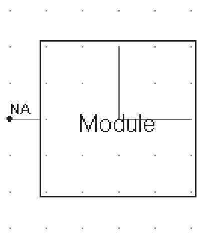
1. Edit the definition of the module. See *Editing a Component or Module Definition* in this section for more details.
2. Define an electrical port connection in the *Graphic* section (Graphic tab) of the module. See *The Graphics Section* in Chapter 9 of this manual for more details.
3. Add a corresponding *XNode* component on the module Circuit canvas. Each *XNode* must be named the same as its corresponding port connection.

EXAMPLE 5-4:

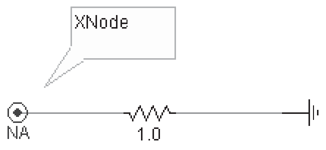
Consider a simple project that consists of a single-phase source bus connected to a module. The module *Graphic* section includes a single port connection named *NA*, which is connected to the source bus on the main Circuit canvas. The module canvas (i.e. Circuit view) consists simply of a *Resistor* component connected directly to a *Ground* component.



Main Canvas Containing a Module and Source



Graphic View of Module Definition



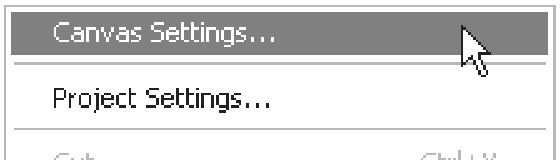
Circuit Canvas View of Module

Here are a few important factors to remember when porting electrical signals into and out of modules:

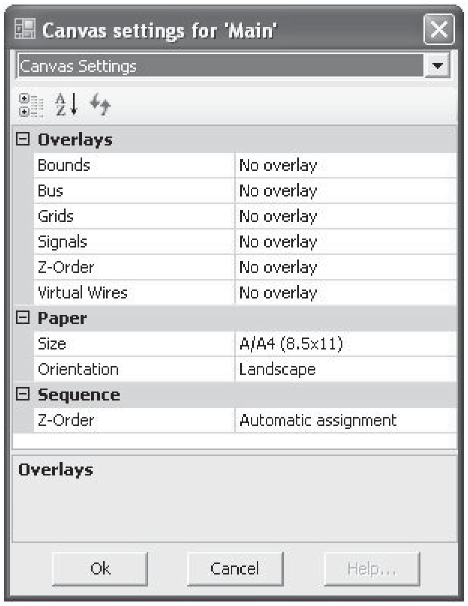
- *XNodes* cannot be placed on the project main page.
- Only one instance of an *XNode* may exist within the module for a single port connection.

- 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 Circuit 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:



Overlays	
Bounds	No overlay
Bus	No overlay
Grids	No overlay
Signals	No overlay
Z-Order	No overlay
Virtual Wires	No overlay
Paper	
Size	A/A4 (8.5x11)
Orientation	Landscape
Sequence	
Z-Order	Automatic assignment

Overlays

The *Canvas Settings* dialog inputs are described in the following sections.

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**: 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.
- **Z-Order**: 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.

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.

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.

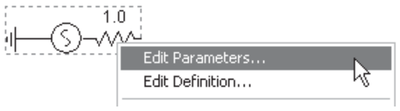
Sequence
These options are related to the way in which *system dynamics* code is ordered. For more on this option, see *Component Ordering* in chapter 11 of this manual.

- **Z-Order**: 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*.

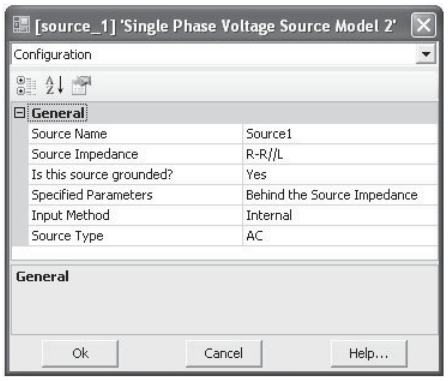
This option is by default enabled, and should remain that way unless the user wants to reorder these components manually.

Editing Component or Module Parameters
Either double left-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.

The double left-click operation is affected by the *Navigate into module* workspace option. If this setting is set to *Double Click*, a double left-click on a module component will perform an *Edit Definition* operation (i.e. will open the module canvas in Circuit view).

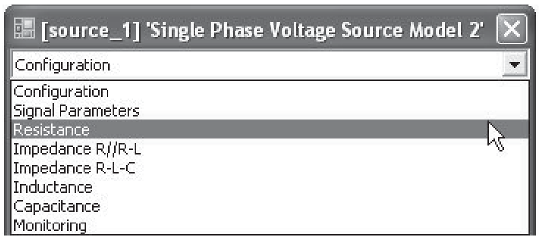


The *Edit Parameters* operation will bring up the component instance parameters dialog. The figure below shows the first page of the dialog for one of the single-phase source models in the master library (called *source_1*).



General	
Source Name	Source1
Source Impedance	R-R//L
Is this source grounded?	Yes
Specified Parameters	Behind the Source Impedance
Input Method	Internal
Source Type	AC

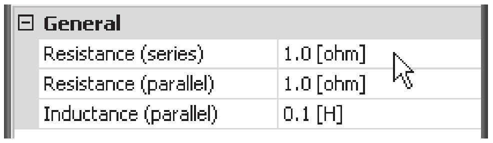
Dialogs for other components will differ, but all include the same basic features and most contain multiple pages: At the top of the dialog is a drop list, which contains a list all of the dialog pages (called *categories*). In this example, the first page is entitled *Configuration*. To move through or view any other pages, left-click the down arrow on the field as shown below:



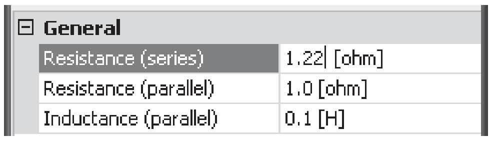
If there are too many items in the list, you will need to scroll to see all the category names. To do this, click the left mouse button on the list to expand it, and then select a particular category with another left click. Or, press the up or down arrow on your keyboard while the list is expanded.

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.



General	
Resistance (series)	1.0 [ohm]
Resistance (parallel)	1.0 [ohm]
Inductance (parallel)	0.1 [H]

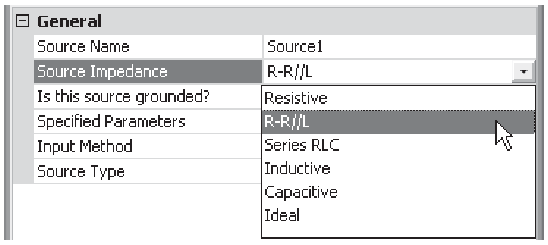


General	
Resistance (series)	1.22 [ohm]
Resistance (parallel)	1.0 [ohm]
Inductance (parallel)	0.1 [H]

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 *toggle* 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.



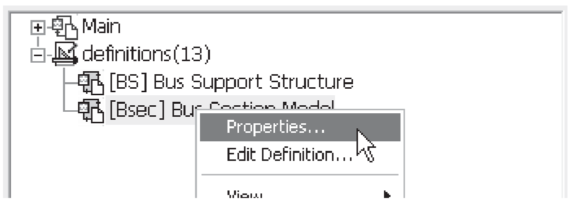
General	
Source Name	Source1
Source Impedance	R-R//L
Is this source grounded?	
Specified Parameters	
Input Method	
Source Type	

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.

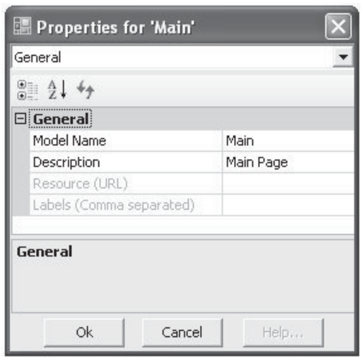
Editing Definition Properties

Definition properties 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 Properties* dialog window:


Properties for 'Main'

General

General	
Model Name	Main
Description	Main Page
Resource (URL)	
Labels (Comma separated)	

General

The following list describes the functions of this dialog:

- **Model 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.
- **Resource (URL)**: This is a future property and is currently disabled.
- **Label (Comma separated)**: 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* in this chapter for more details.

Linking and Re-linking Definitions

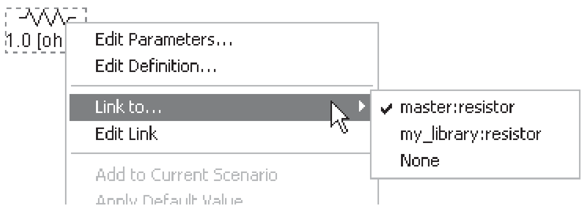
In PSCAD versions X4 or greater, users are given the ability to re-link any particular component instance to another definition. Obviously, the new definition must be compatible with the original for this to work, but this feature can be convenient when working with several versions of the same definition. Re-linking definitions is also important when copying and pasting module components or transmission segments between projects.

For more details on namespace and other definition linking issues, see *Changing the Namespace Name* in this chapter and *Duplicate Definition Linking Priority* in chapter 13.

Link List

All definitions that share the same name in any loaded project in the workspace will appear in the link 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 **Link to...**.



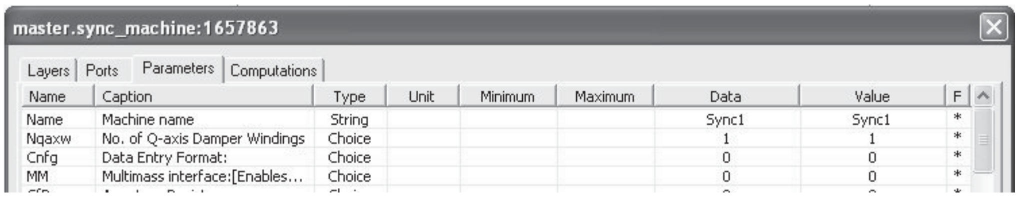
As can be seen above, both resistor definitions have been detected. From here you may freely switch between definitions.

Resource Link

If a component instance is not currently linked to a definition, or if you want to point to a specific definition of a different name, then use **Resource Link...**. This will invoke the *Resource Link* dialog in which you can adjust both the namespace and definition name.

Viewing Component Properties

The complexity of a component may range 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.



master.sync_machine:1657863

Layers | Ports | Parameters | Computations

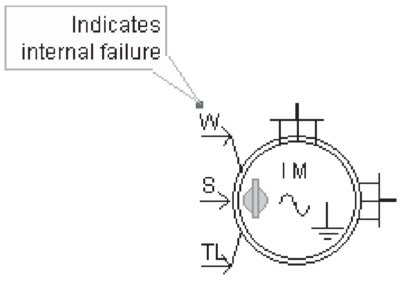
Name	Caption	Type	Unit	Minimum	Maximum	Data	Value	F
Name	Machine name	String				Sync1	Sync1	*
Nqaxw	No. of Q-axis Damper Windings	Choice				1	1	*
Cnfg	Data Entry Format:	Choice				0	0	*
MM	Multimass interface:[Enables...	Choice				0	0	*

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):

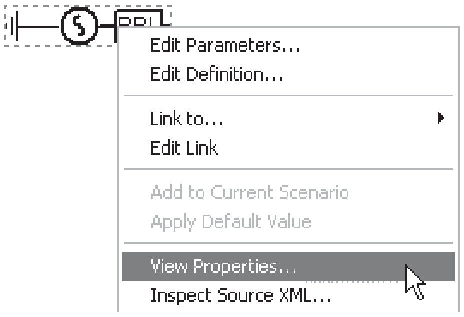
- **Layers**: Provides information on graphical layers and related conditional statements set in the component *Graphics* section.
- **Ports**: Provides information on external connections (nodes) and related conditional statements set in the component *Graphics* section.
- **Parameters**: Provides information 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
- **Failures**: Provides a list of pre-compilation errors (only appears if errors are detected). For example, *Computations* segment errors.

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 failures are indicated with a **#NaN** in the **Value** column.



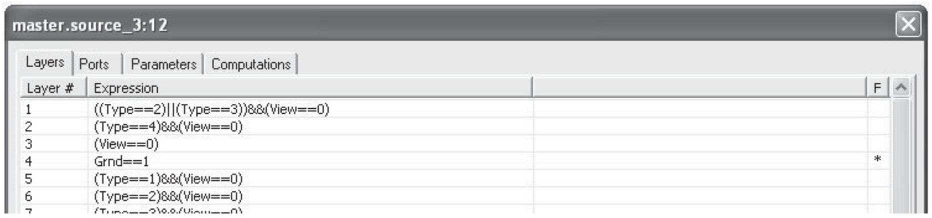
Invoking the Component Properties Viewer

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.

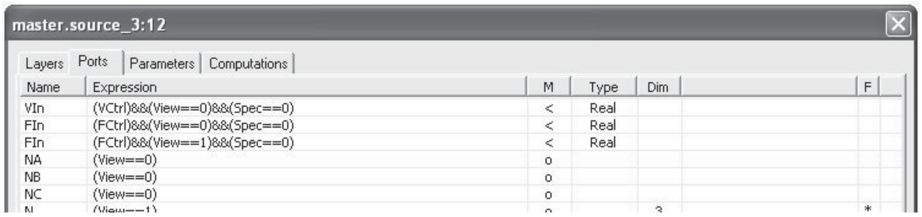


Layer #	Expression		F
1	((Type==2)||(Type==3))&&(View==0)		
2	(Type==4)&&(View==0)		
3	(View==0)		
4	Grnd==1		*
5	(Type==1)&&(View==0)		
6	(Type==2)&&(View==0)		
7	(Type==3)&&(View==0)		

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).



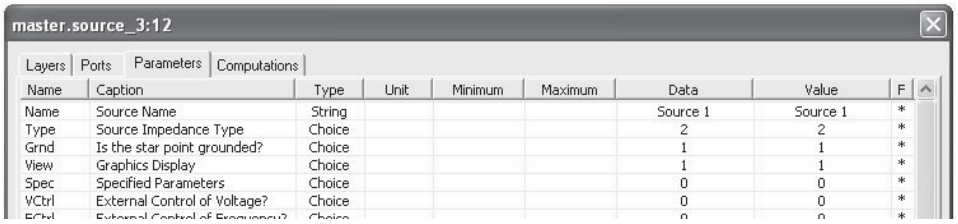
Name	Expression	M	Type	Dim		F
VIn	(VCtrl)&&(View==0)&&(Spec==0)	<	Real			
FIn	(FCtrl)&&(View==0)&&(Spec==0)	<	Real			
FIn	(FCtrl)&&(View==1)&&(Spec==0)	<	Real			
NA	(View==0)	o				
NB	(View==0)	o				
NC	(View==0)	o				
N	(View==1)	o		3		*

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).

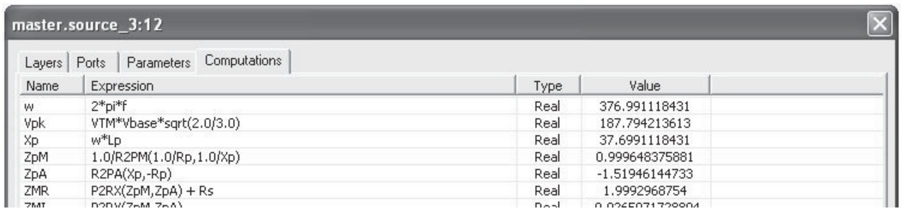


Name	Caption	Type	Unit	Minimum	Maximum	Data	Value	F
Name	Source Name	String				Source 1	Source 1	*
Type	Source Impedance Type	Choice				2	2	*
Grnd	Is the star point grounded?	Choice				1	1	*
View	Graphics Display	Choice				1	1	*
Spec	Specified Parameters	Choice				0	0	*
VCtrl	External Control of Voltage?	Choice				0	0	*
FCtrl	External Control of Frequency?	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 Field.
- **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).

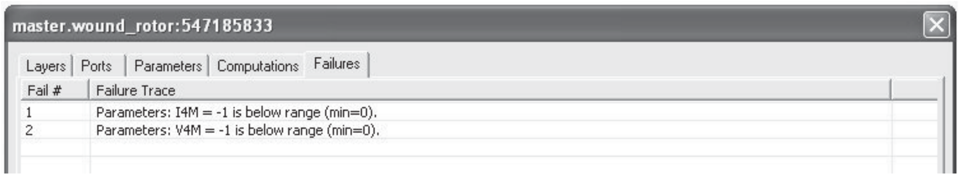


Name	Expression	Type	Value
w	2*pi*f	Real	376.991118431
Vpk	VTM*Vbase*sqrt(2.0/3.0)	Real	187.794213613
Xp	w*Lp	Real	37.6991118431
ZpM	1.0/R2PM(1.0/Rp,1.0/Xp)	Real	0.999648375881
ZpA	R2PA(Xp,-Rp)	Real	-1.51946144733
ZMR	P2RX(ZpM,ZpA) + Rs	Real	1.9992968754
ZMI	[illegible]	Real	[illegible]

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.



Fail #	Failure Trace
1	Parameters: I4M = -1 is below range (min=0).
2	Parameters: V4M = -1 is below range (min=0).

Failures Tab

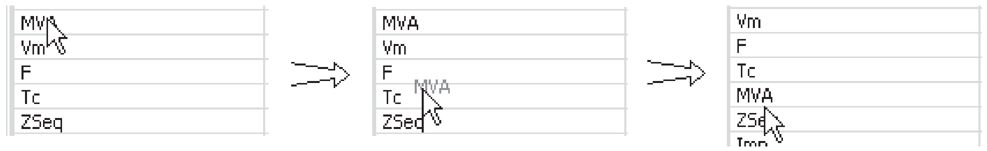
Failures:

- **Fail #**: The sequential failure number (i.e. order in list).
- **Failure Trace**: A description of the actual failure. This is normally a message from the internal compiler.

Formatting Viewed Properties

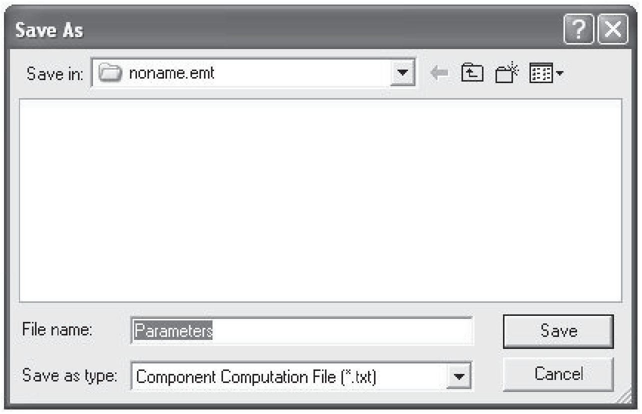
Once the *Component Properties Viewer* has been opened, the data is open to some simple formatting. Although property attributes cannot be altered, users may adjust the order in which properties appear –this is useful if the data is to be saved to a file for example.

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.



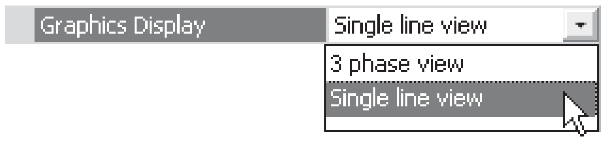
Enter a name for the file and press the **Save** button. Once saved, PSCAD will open the file automatically for viewing.

Copying Data to Clipboard

Specific rows of property data can be saved to the clipboard at any time. Simply right-click 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 and select the desired view from this field.

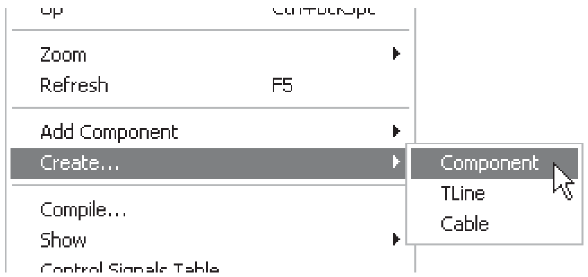
Creating a New Component, Module or Transmission Wire

New components (including modules and transmission segments) can be created by using the *Component Wizard*. The *component wizard* generates a new definition, and then creates a single instance of it for placement on the Circuit canvas. The user may then continue 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* in this manual.

Using the Component Wizard

Press either the **New Component** button in the main toolbar, or move the mouse pointer over a blank area of any page, right-click and select **Create...** | **Component**.



The first page of the component wizard will appear:

Step 1

Enter all parameters required according to the descriptions below. When finished, click the **Next >** button. If no port connections are specified, skip to Step 4.

- **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.
- **Connections:** If you would like to add any *port connections* to your component, enter the number required in any or all of the **Left**, **Top**, **Right** or **Bottom** fields. For example, entering 2 in the **Left** field indicates that you want two port connections on the left side of the component graphic.

- **Page Module:** Select this input if you want the new component to be a module (also referred to as a page component or a sub-page).

Step 2
Enter the parameters required according to the descriptions below. This dialog page will be repeated for each port connection specified in *Step 1*, starting with the left side of the component graphic. The text box at the top of the dialog will indicate which port connections are currently being defined.

When finished each page, click the **Next >** button. If **Page Module** was not selected in Step 1, skip to Step 4. If a mistake is made, press the **< Back** button at any time.

- **Port 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).
- **Port Dimension:** 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 Type:** Select the type of electrical port connection here. This input is enabled only if the *Port Type* is selected as *Electrical*.
- **Data Type:** Select the type of data port connection here. This input is enabled only if the *Port Type* is selected as either *Input Data* or *Output Data*.

For more information on port connection types (i.e. electrical, data, etc.) see *Port Connections* in Chapter 9 of this manual.

Step 3
When finished, click the **Next >** button. If a mistake is made, press the **< Back** button at any time. Step 3 will appear only if the **Page Module** input was selected in Step 1.

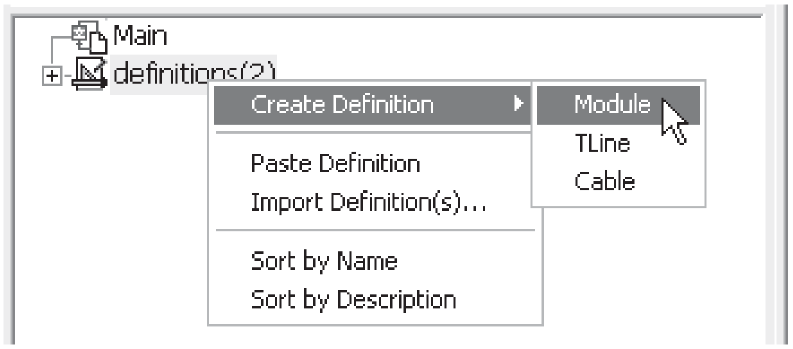
- **Paper Size:** Select a paper size from the list provided.
- **Orientation:** Select either Portrait or Landscape page orientation.

Step 4

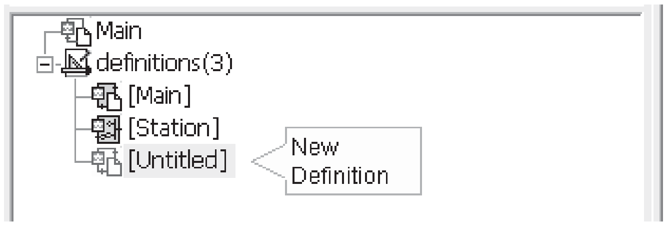
Step 4 is simply a confirmation step. Press the **Finish** button if you are satisfied that everything is correct. Press the **< Back** button to go back to the previous step. Press **Cancel** to cancel the whole process.

Manually Creating a New Module or TLine/Cable Definition

Go to the Workspace secondary window, right-click on the ***definitions*** branch and select Create Definition. A sub-menu will appear with the options Module, TLine or Cable.



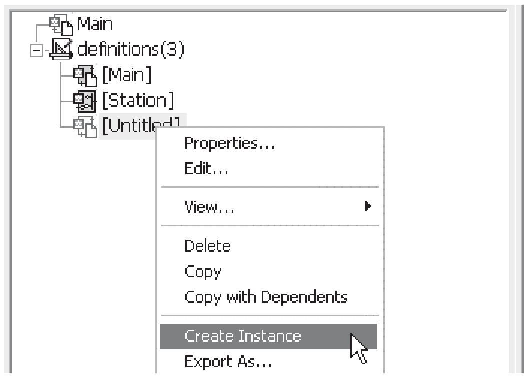
Select the definition type to create. A new definition will appear in the *definitions* branch, given a default name.



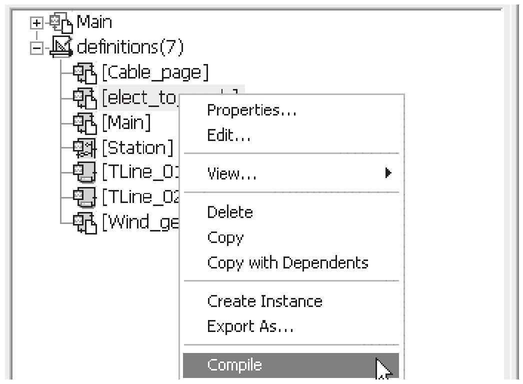
See the next section for instructions on how to create the first instance of the new definition.

Creating the First Instance of a Definition

In the workspace secondary window, navigate to the *definitions* branch. Right-click the desired definition and then select **Create Instance**. Then, paste the newly created instance on the Circuit canvas (right-click on the canvas and select **Paste**).



Compiling an Individual Module Definition
In the workspace secondary window, navigate to the *definitions* branch. Select the desired module definition from the list, right-click and select **Compile**.



Lock/Unlock Modules
To lock a module component Circuit canvas from being viewed: Right-click on the module definition list and select **Lock**. You will be prompted to enter a password. Enter it (and do not lose it!) and press **OK**.

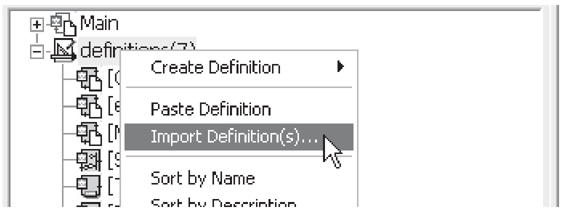
The locking mechanism prevents users from looking at the canvas of the module. To unlock the module, right-click on the module definition list and select **Unlock**. You will be prompted to re-enter the password.

For more details on locking modules, see *Module Locking* in chapter 11 of this manual.

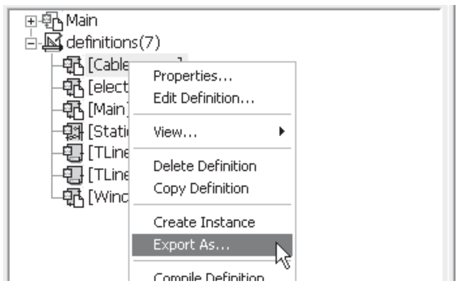
Importing/Exporting Definitions
Definitions may be imported or exported to and from projects, or exchanged between users, by saving the definition as a *Definition (*.psdx)* file. A definition file contains the XML data elements defining

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.

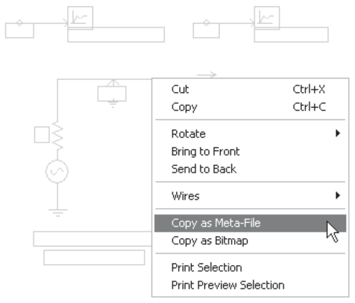
Import a Definition
In the workspace secondary window, left-click on the [+] box beside the *definitions* branch to expand the definitions list. Right-click on the *definitions* branch itself and select **Import Definition(s)...**.



Export a Definition
To export a 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...**.



Copy as Meta-File or Bitmap
Bring up the desired area to be copied in the Circuit 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 Selected Objects

Bring up the desired area to be printed in the Circuit canvas. Select the objects to be printed (see *Selecting Objects* in this chapter), right-click and select **Print Selection**. A print dialog window should appear - select your desired printer properties and click **OK**.

More than one object must be selected to print a selection.

To preview the selection before printing, select the objects to be printed, right-click and select **Print Preview Selection**.

Printing a Module Page

Bring up the desired module canvas in Circuit view. Right-click on a blank portion of the canvas and select **Print Page** (or press the **Print** button in the main toolbar) –a print dialog window should appear. Adjust the printer properties and click **OK**.

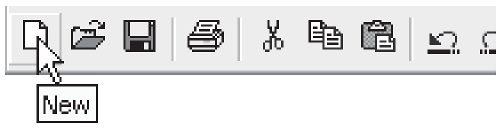
To preview before printing, bring up the desired module canvas in Circuit view and right-click on a blank portion of the canvas. Select **Print Preview Page**.

TUTORIAL: CREATING A NEW PROJECT

This tutorial is meant to introduce the user to creating a new project from scratch. In it, we will describe how to create, construct and simulate a very simple voltage divider circuit.

Creating a New Case Project

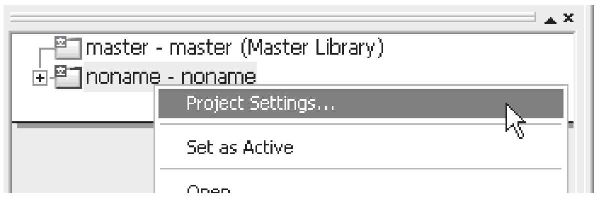
Press the **New** button in the main toolbar to open the *Create New Project* dialog.



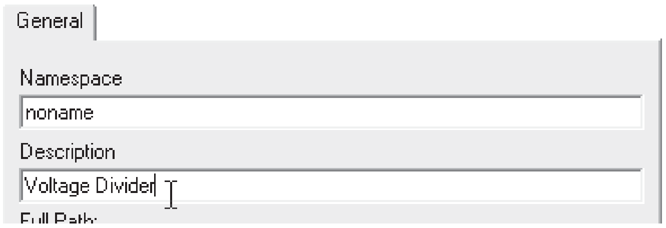
If the workspace window is not visible, select **View** | **Workspace** in the main menu.

Enter a name for the project, as discussed in the section entitled *Creating a New Project* previously in this chapeter. A new project will appear in the workspace window.

Right-click on the project name and select **Project Settings...**.



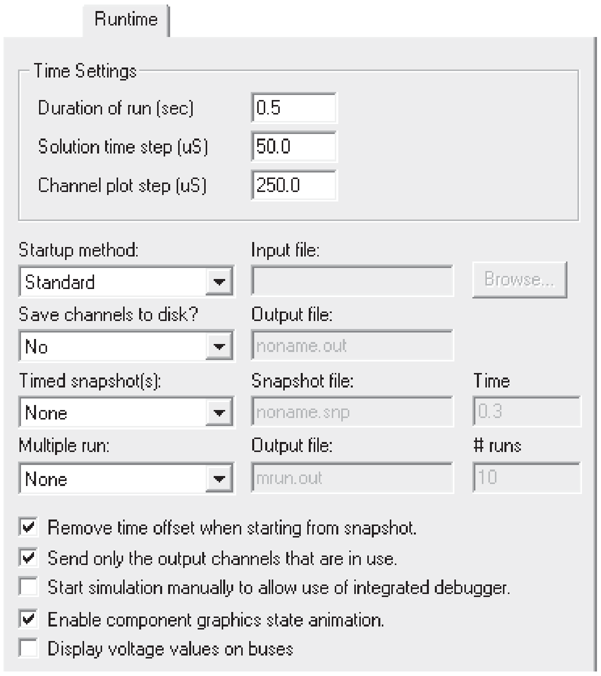
This action will open the *Project Settings* dialog. Click the **General** tab at the top of the dialog, left-click inside the **Description** field and type a description for the case, say, 'Voltage Divider'.



Namespace: noname

Description: Voltage Divider

Click the **Runtime** tab and get familiar with the inputs displayed therein (see *Runtime* for details on parameters).



Duration of run (sec): 0.5

Solution time step (uS): 50.0

Channel plot step (uS): 250.0

Startup method: Standard

Input file:

Save channels to disk?: No

Output file: noname.out

Timed snapshot(s): None

Snapshot file: noname.snp

Time: 0.3

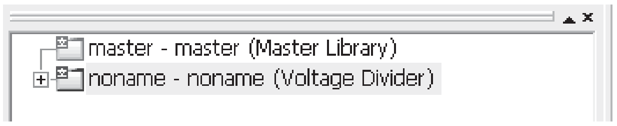
Multiple run: None

Output file: mrun.out

# runs: 10

- ✓ Remove time offset when starting from snapshot.
- ✓ Send only the output channels that are in use.
- ☐ Start simulation manually to allow use of integrated debugger.
- ✓ Enable component graphics state animation.
- ☐ Display voltage values on buses

The project will be automatically set up with a run duration of *0.5* seconds with a *50* μ*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.

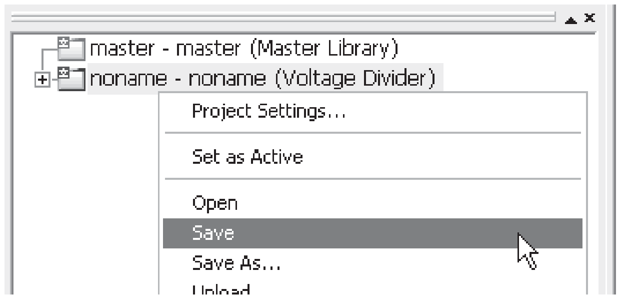


master - master (Master Library)

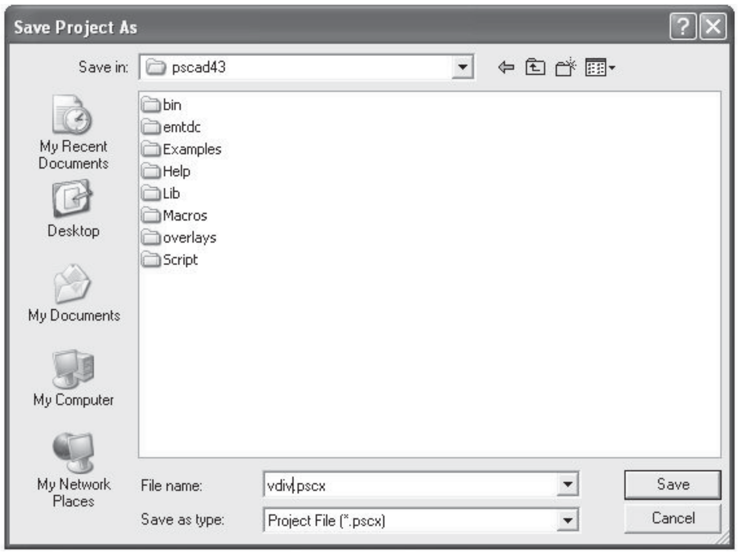
noname - noname (Voltage Divider)

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.


master - master (Master Library)
noname - noname (Voltage Divider)
Project Settings...
Set as Active
Open
Save
Save As...
Unload

This will open the **Save Project As** dialog.



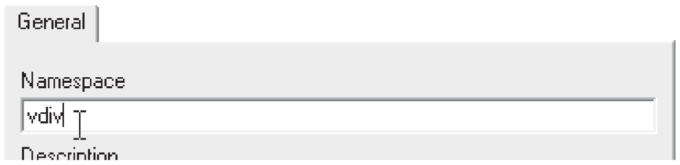
File name: vdiv.pscx

Save as type: Project File (*.pscx)

Use the buttons at the top of this dialog to navigate to where you want to save the project file. Type the name of the file (say *vdiv.pscx*) in the **File name** field. Click the **Save** button. The project icon in the workspace primary window will turn to blue and the new filename will appear.

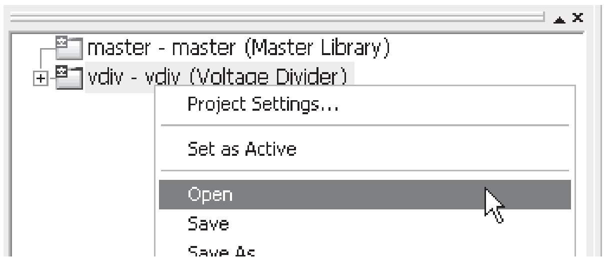
Changing the Namespace Name

Open the *Project Settings* dialog as described above and again click the **General** tab at the top of the dialog. Left-click inside the **Namespace** field and declare a namespace name for the case. To avoid confusion, the namespace should be the same as the case name (i.e. *vdiv* in this example).



The namespace name will not be utilized in this tutorial. For more details on what the namespace is and what it is used for, see *Linking and Re-Linking Definitions* in this chapter and *Duplicate Definition Linking Priority* in chapter 13.

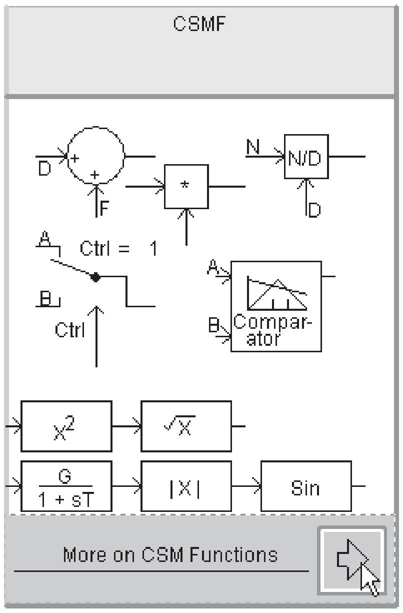
Opening the Project Main Canvas
To open the main Circuit canvas of your new project, left double-click on the project name in the workspace primary window, or right-click on the name and select **Open**.


master - master (Master Library)
vdiv - vdiv (Voltage Divider)
Project Settings...
Set as Active
Open
Save
Save As

An empty canvas should open in the Circuit window. This is the main canvas of your new project, on which you will draw the circuit.

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.

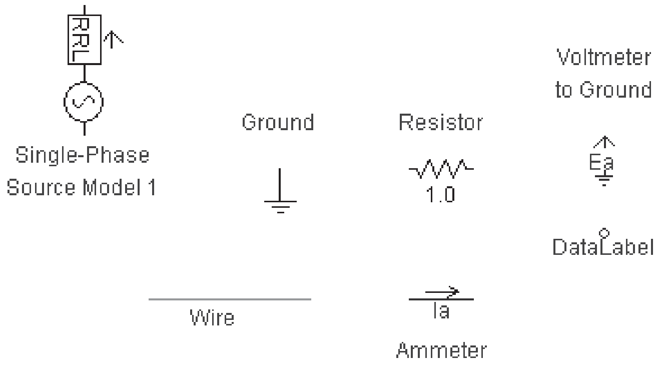
To open the master library, follow the same procedure as outlined in *Opening the Project Main Canvas*. The master library is organized as a group of modules; each opens into another canvas window, giving access to all of the components belonging to that group. To open a particular group, hold down the **Ctrl** key and **left double-click** the arrow in the bottom-right corner of the module:



Assembling the Voltage Divider Circuit

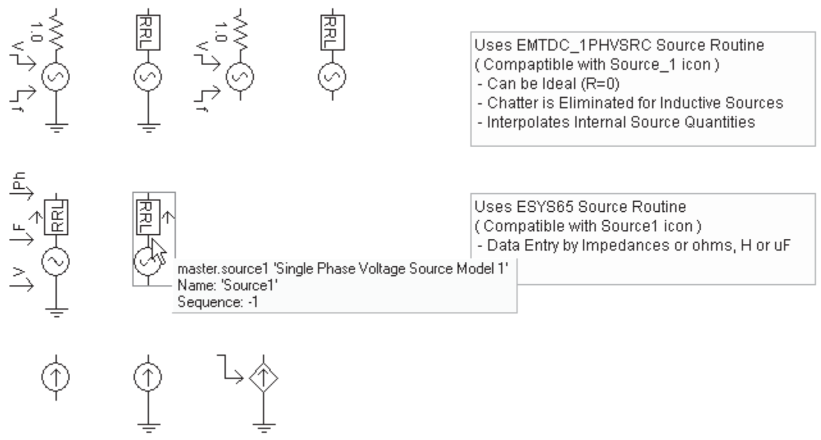
The voltage divider circuit in this example will use eight different components as shown below.

Initially, it may be a challenge to navigate through the master library and find these components one by one.



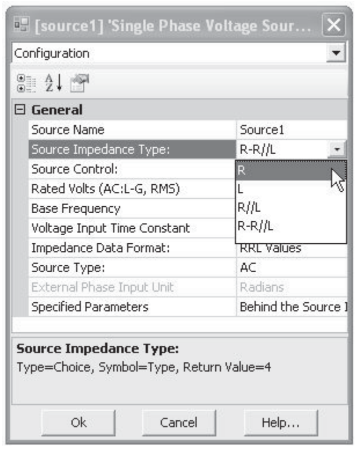
Locating the Single-Phase Source

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 1** component.


Uses EMTDC_1PHVSRC Source Routine
(Compaptible with Source_1 icon)
- Can be Ideal (R=0)
- Chatter is Eliminated for Inductive Sources
- Interpolates Internal Source Quantities
Uses ESYS65 Source Routine
(Compatible with Source1 icon)
- Data Entry by Impedances or ohms, H or uF
master.source1 'Single Phase Voltage Source Model 1'
Name: 'Source1'
Sequence: -1
RRL

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 'R' (for purely resistive).


[source1] 'Single Phase Voltage Sour...

Configuration

General

Source Name	Source1
Source Impedance Type:	R-R//L
Source Control:	R
Rated Volts (AC:L-G, RMS)	L
Base Frequency	R//L
Voltage Input Time Constant	R-R//L
Impedance Data Format:	RRL Values
Source Type:	AC
External Phase Input Unit	Radians
Specified Parameters	Behind the Source I

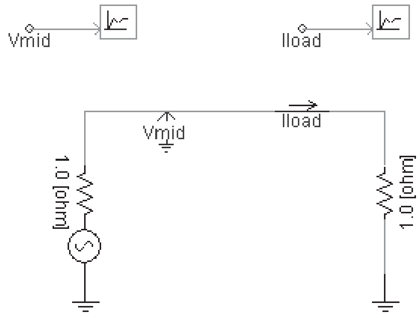
Source Impedance Type:
Type=Choice, Symbol=Type, Return Value=4

On the **AC Source Initial Values** page, change the **Initial Source Mag (L-G, RMS)** input field to *70.71 [kV]*. This will give an internal source voltage magnitude of *100 kV* peak.

Save the project.

Add and Assemble

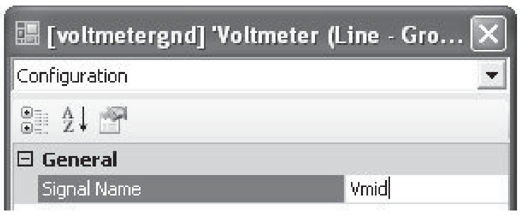
The next step is to add the remaining components (i.e. *Wire*, *Resistor*, *Current Meter*, *Data Label*, *Output Channel*, and *Ground* components). Arrange all components to form the simple voltage divider shown in the following diagram:



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

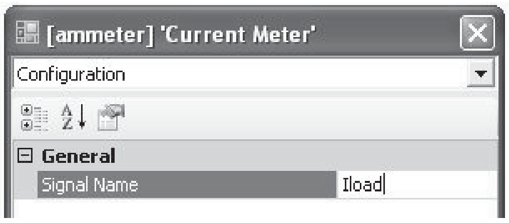
Use the properties shown below for the remaining components (see *Editing Component Parameters* in this chapter). Only the properties to be changed from their default values are mentioned. Use the resistor with its default properties.


[voltmetergnd] 'Voltmeter (Line - Gro...

Configuration

General

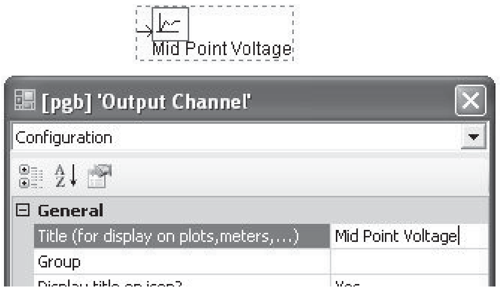
Signal Name: Vmid


[ammeter] 'Current Meter'

Configuration

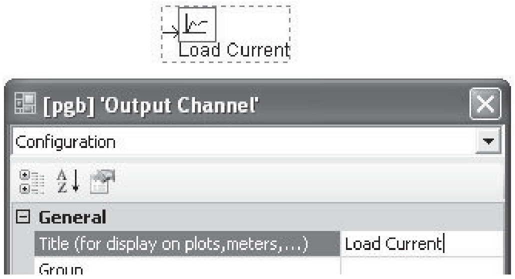
General

Signal Name: Iload



Title (for display on plots,meters,...): Mid Point Voltage

Group



Title (for display on plots,meters,...): Load Current

Group

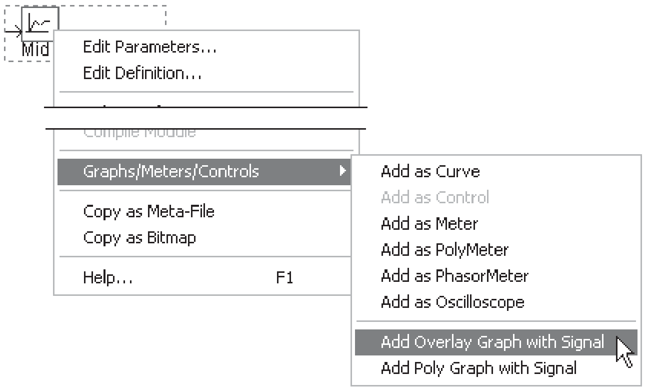
Save the project.

Plotting Results

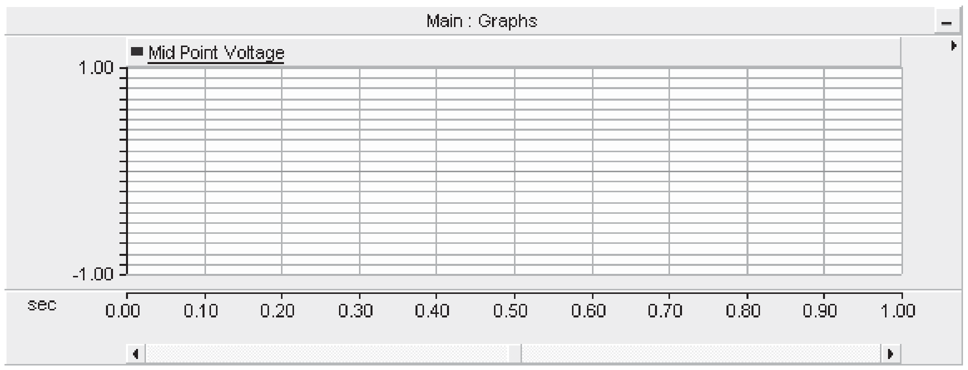
In order to view any results from our voltage divider circuit, we must add a *Graph Frame* and set it up to display the waveforms. The following sections describe only those aspects of plotting needed for this exercise. See *Preparing Data for Control or Display* in chapter 6 of this manual for further reading.

Adding a Graph Frame

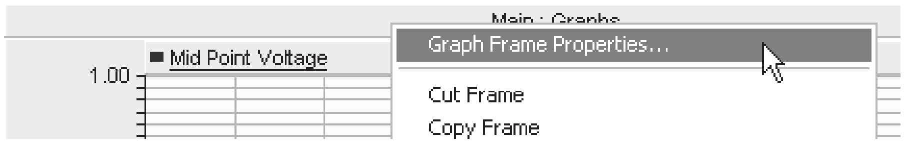
Right-click on the *Output Channel* component called *Mid Point Voltage* to bring up the pop-up menu. Select **Graphs/Meters/ Controls | Add Overlay Graph with Signal**.



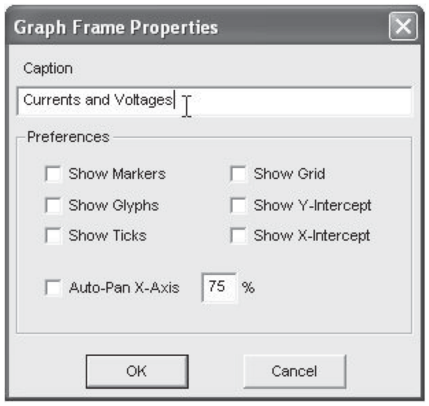
This will create a new *Graph Frame*, *Overlay Graph* and a *Curve* simultaneously as shown below:



Right-click on the graph frame title bar (top bar on the plot labelled 'Main : Graphs') and select **Graph Frame Properties...** from the pop-up menu.



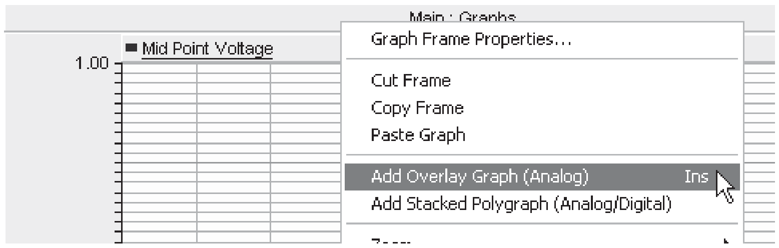
This will bring-up the *Graph Frame Properties* dialog window. In the **Caption** field, change the title to *Currents and Voltages*. See *Graph Frames* in chapter 6 for more details on the options in this dialog.



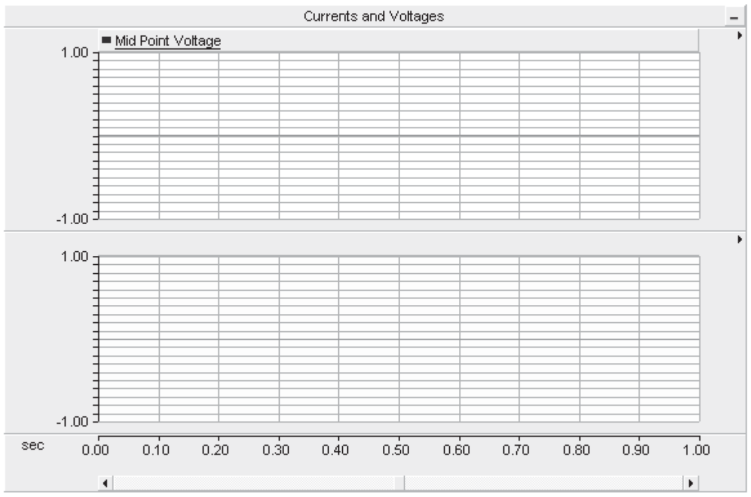
To resize the graph frame at any time, left click on the frame title bar so that grips appear. Left-click and hold on a corner grip and drag mouse outwards. Resize the graph frame to approximately *8 x 8* centimetres.

Additional Overlay Graph and Curve

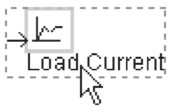
Right-click on the graph frame title bar and select **Add Overlay Graph (Analog)** (or press the **Insert** key while the mouse pointer is over the graph frame).


Main : Graphs
Mid Point Voltage
1.00
Graph Frame Properties...
Cut Frame
Copy Frame
Paste Graph
Add Overlay Graph (Analog) Ins
Add Stacked Polygraph (Analog/Digital)

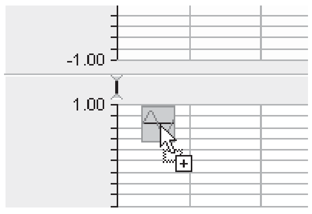
A new *Overlay Graph* will appear within the frame directly below the existing graph (see *Graphs* in chapter 6 for more).



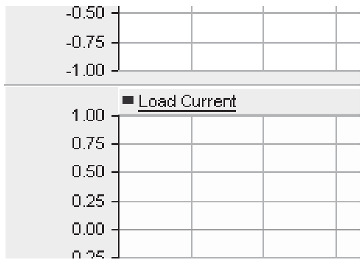
Add a curve to this graph to monitor load current: Hold down the **Ctrl** key and click and hold the **left mouse button** while over the *Output Channel* with title *Load Current*. Drag the mouse pointer over the new graph and release the mouse button. A curve will appear.



Select the Output Channel (**Ctrl + left mouse hold**)



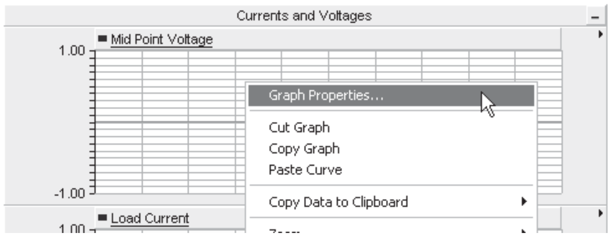
Drag the Mouse Pointer Over the New Graph



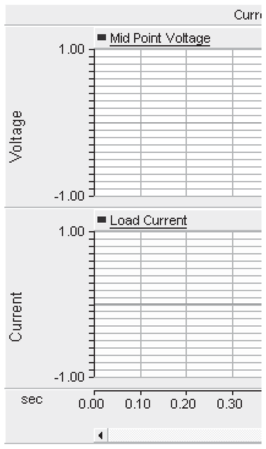
Release the Mouse Button

Editing the Graph Properties

To customize an individual graph title and/or vertical axis label, right click over top the graph and select **Graph 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 display *Voltage* on the voltage graph, and *Current* on the current graph. You may also want to turn on/off **Grid Lines** and adjust the scaling. The graph frame should appear similar to that shown below once completed.



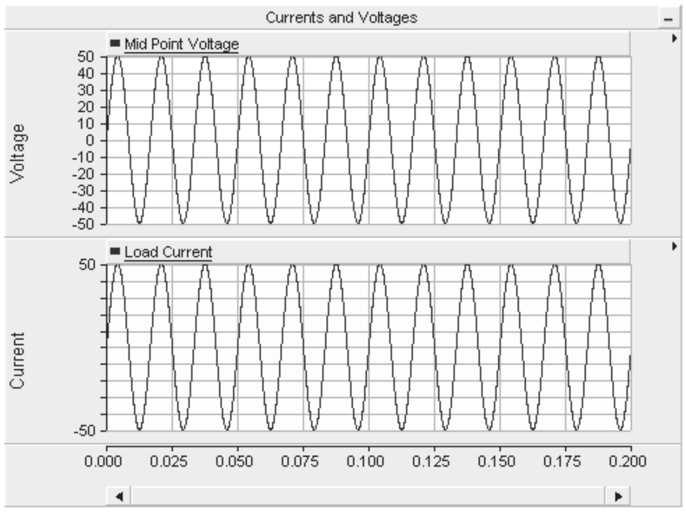
Save the project.

Running the Project

Before running the project, ensure that it is the active project in the workspace primary window (see *Setting the Active Project* in this chapter). Press the **Run** button in the main toolbar to run the simulation (see *Running a Simulation* in this chapter).

It may be necessary to readjust your x and y-axes, as well as zoom and time frame. See *Chapter 6 – Online Plotting and Control* for more details.

This is the last step, assuming there are no errors. If there are any, they will be logged in the *Output Window*. See *Error and Warning Messages* in chapter 11 for more details. Your simulated results should look similar to the following once the simulation has completed.



MULTIPLE INSTANCE MODULES

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.

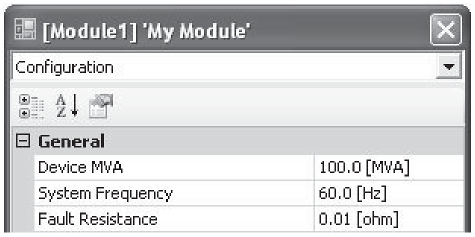
The absence of the ability to multiply-instance modules remained throughout the life of both the V3 and V4 releases, but was finally remedied in PSCAD X4. This was made possible by a complete rework of the PSCAD program architecture towards a more data-centralized model. This new architecture greatly 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.

Input Parameters

With the establishment of multiple instance module capabilities in PSCAD, it was also a prudent time to introduce input parameter functionality as well. As veteran users know, module components did not support input parameters in the past, and relied purely on port connections to transfer data signals into and out of the canvas. Input 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 input parameter values.

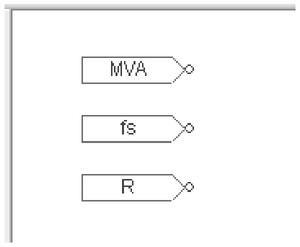
When it comes to porting input parameter values onto the canvas, they are treated in exactly the same way as port connections, that is, each parameter requires a matching *Import* component on the canvas. The name of the *Import* component must match the *Symbol* name of the input parameter.



General

Device MVA	100.0 [MVA]
System Frequency	60.0 [Hz]
Fault Resistance	0.01 [ohm]

Module Input Parameters



Import Connection Components on Canvas

For more details on designing component parameter dialogs, see *The Parameters Section* in chapter 9 of this manual.

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.

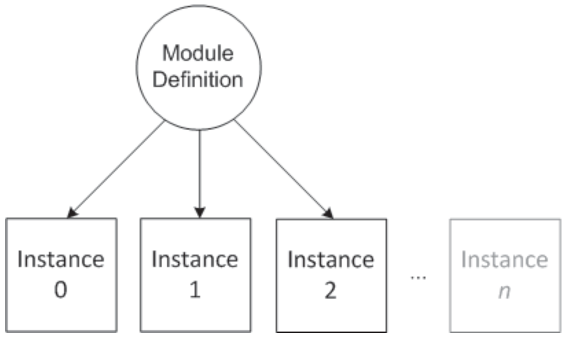
	Circuit	Graphic	Parameters	Script
Standard Component	✗	✓	✓	✓
Module Component	✓	✓	✓	✗

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 root module, and it, along with its definition, are included for you automatically whenever a new project is created.

The Circuit canvas s 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), but this is not the case any more.

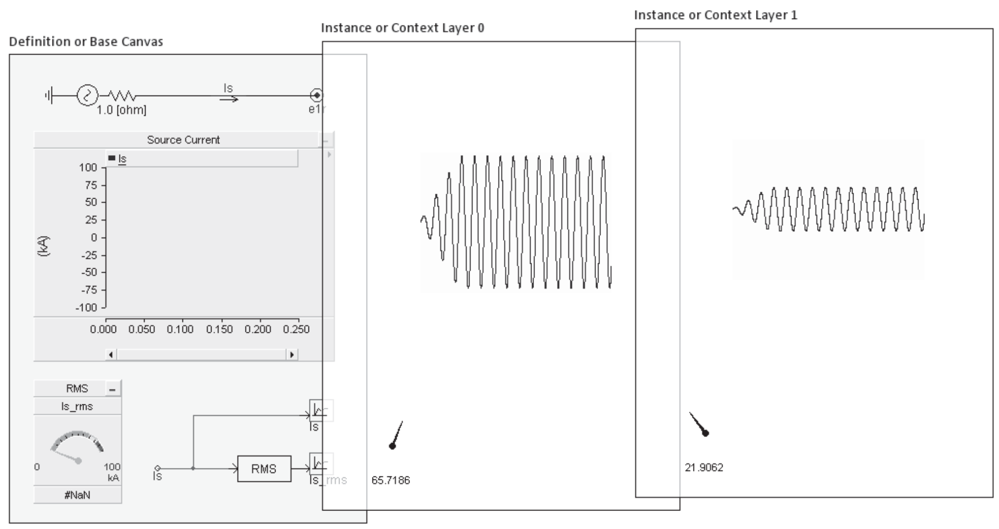


Multiple Instances Based on a Single Module Definition

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*.

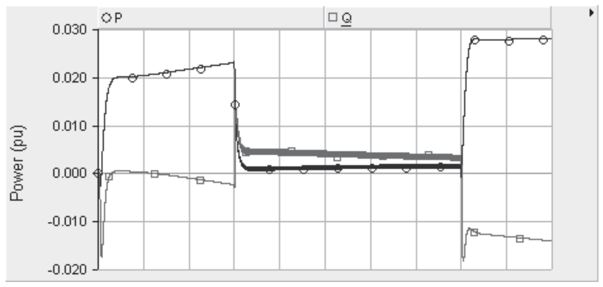
Every component, graph, control panel, etc. that is placed on a module canvas, becomes part of that modules definition. If a plot panel is resized, or a new graph is added to the panel, it modifies the definition. Changing graph settings, curve colours, zoom settings all affect the definition. The same 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.

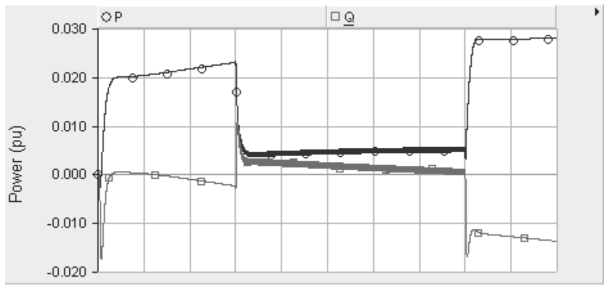
It is important to distinguish between the settings of plotting and control tools, and the actual signals themselves. The signals being displayed on graphs or meters may not necessarily be defined purely as part of the module definition. If a signal is sourced from within a module, say for example a 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.

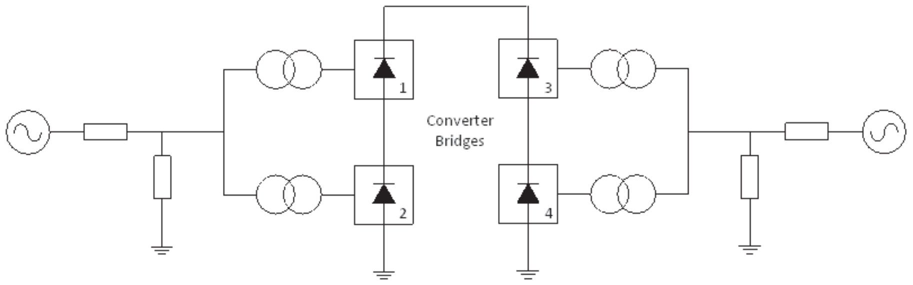




Same Graph from Two Separate Module Instances Based on the Same Definition

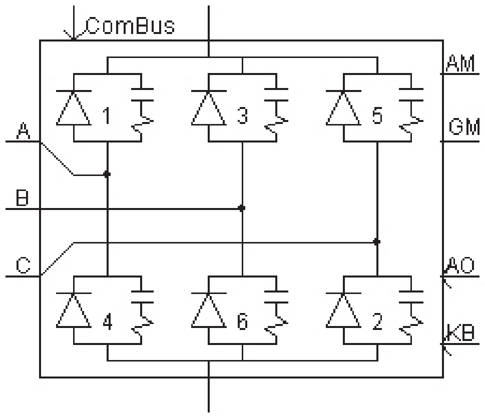
A Practical Example of Multiple Module Instances in Use

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.

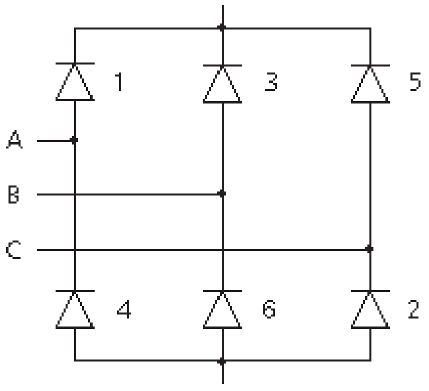


Monopolar HVDC System with Four Converter Bridge Units

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.



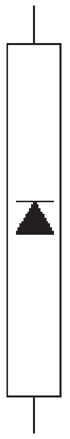
6 Pulse Bridge Component Graphic

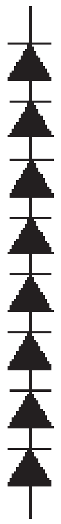


6 Pulse Bridge Equivalent Circuit

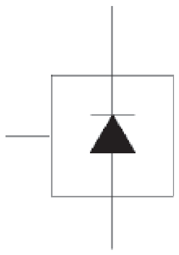
Standard components however are limited in flexibility, in that if they are to be combined with other components to form another single device, they must be reprogrammed into a single unit (i.e. combining two bridge units in series to form a 12-pulse converter for example). Reprogramming these models 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.

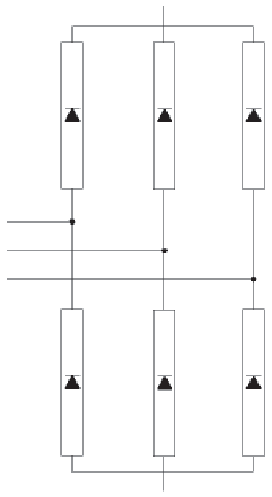
Let us conceptually build a three-phase bridge from scratch, using only module components: The most elemental (i.e. indivisible) component of the converter is the individual thyristor valves (ignoring the snubber circuits): This is where we should start the design. In practice, thyristor valves are combined 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:

```
0:   6-Pulse Bridge
          0:  Valve Group
          1:  Valve Group
          2:  Valve Group
          3:  Valve Group
          4:  Valve Group
          5:  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:

```
0:   Main
        0:   6-Pulse Bridge
                  0:Valve Group
                  1:Valve Group
                  2:Valve Group
                  3:Valve Group
                  4:Valve Group
                  5:Valve Group
        1:   6-Pulse Bridge
                  0:Valve Group
                  1:Valve Group
                  2:Valve Group
```

```
                3:Valve Group
                4:Valve Group
                5:Valve Group
2:    6-Pulse Bridge
                0:Valve Group
                1:Valve Group
                2:Valve Group
                3:Valve Group
                4:Valve Group
                5:Valve Group
3:    6-Pulse Bridge
                0:Valve Group
                1:Valve Group
                2:Valve Group
                3:Valve Group
                4:Valve Group
                5:Valve Group
```

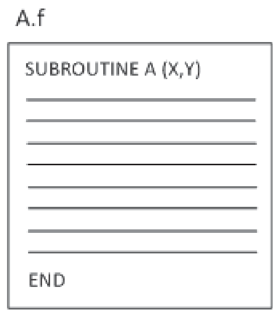
Coding Analogies

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.

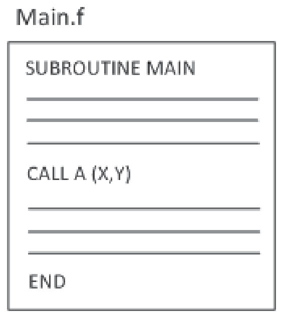
PSCAD Object	Code Equivalent
Project	Program
Module Definition	Subroutine
Module Instance	Subroutine Call
Non-Module Component	Inline Code
Port Connections/Input Parameters	Subroutine Arguments

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.

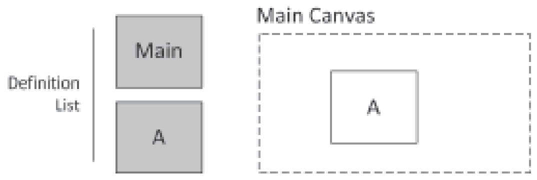


Module A Definition Fortran File

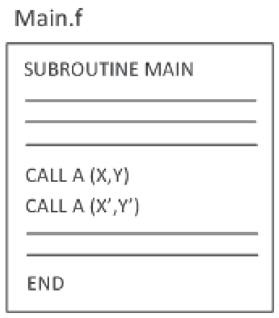


Call to Subroutine from Main Module

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*.

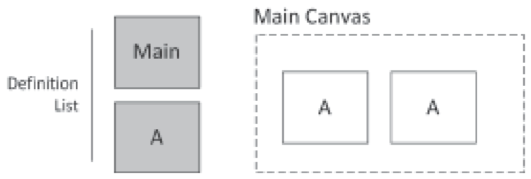


In the above example, there are two module definitions (one for *Main* and one for *A*). What would happen then if *A* was instantiated again so that there are two instances of it on the *Main* canvas? Well a new definition is not required; only an additional call statement to *A* within the *Main* subroutine is needed.



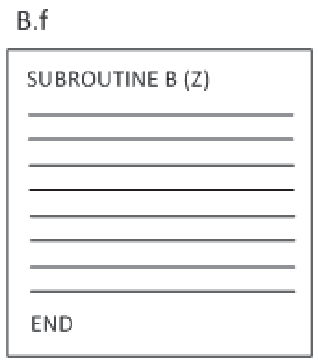
```
Main.f
SUBROUTINE MAIN
CALL A (X,Y)
CALL A (X',Y')
END
```

Main Definition Fortran File



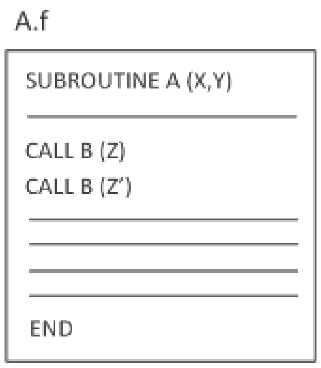
Two Instances of A on the Main Canvas

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*.



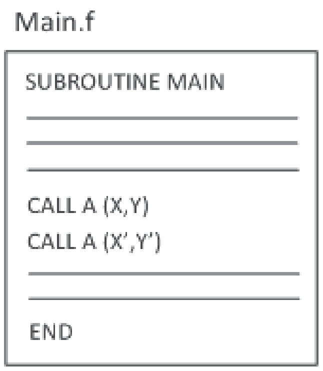
```
B.f
SUBROUTINE B (Z)
END
```

Module B Definition Fortran File



```
A.f
SUBROUTINE A (X,Y)
CALL B (Z)
CALL B (Z')
END
```

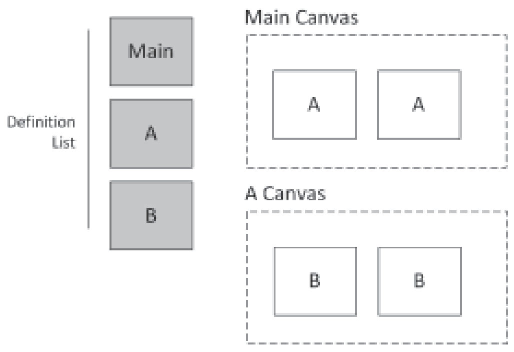
Module A Definition Fortran File with Calls to B



```
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

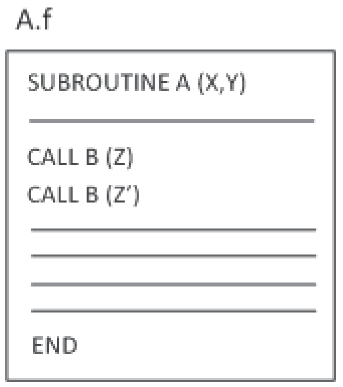
Instance and Call Numbers for the Given Example

It is important to understand this concept in order understand fully how multiple instance modules work, and how signal values are mapped and accessed within PSCAD and EMTDC.

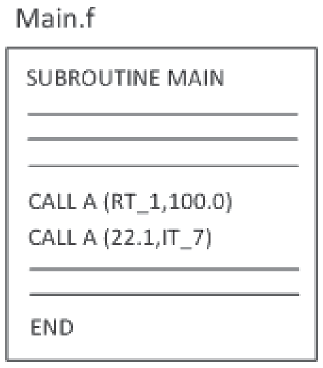
Subroutine Arguments

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.

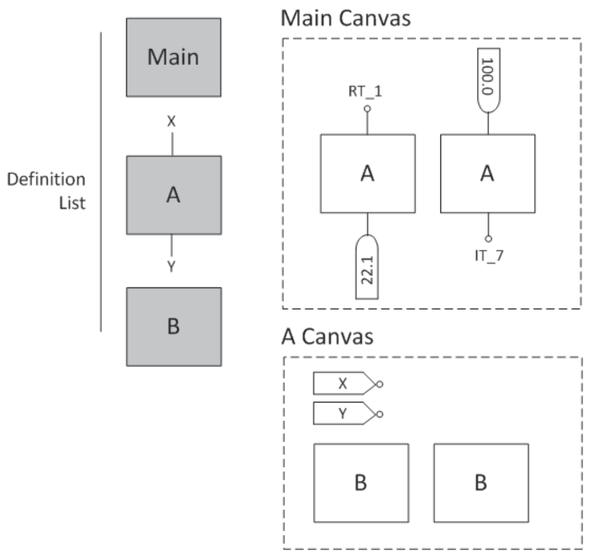


Module A Definition Fortran File



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.



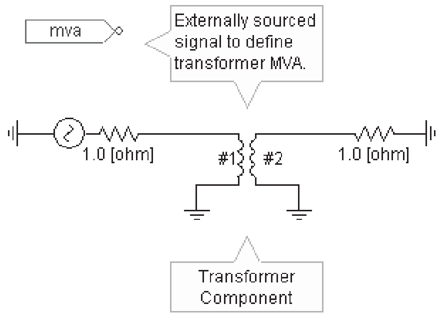
For more details on defining input parameters and port connections, see the chapter entitled *Component Design* in this manual.

Important Design Considerations

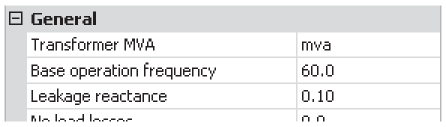
Up until now, we have discussed multiple instance modules in a conceptual manner. In practice however, there are certain important things to consider when designing your system, so as to ensure maximum compile and simulation efficiency. It is essential to consider where signals are sourced. That is, 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:



Module Definition Canvas with Transformer Component



General	
Transformer MVA	mva
Base operation frequency	60.0
Leakage reactance	0.10
No load losses	0.0

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.

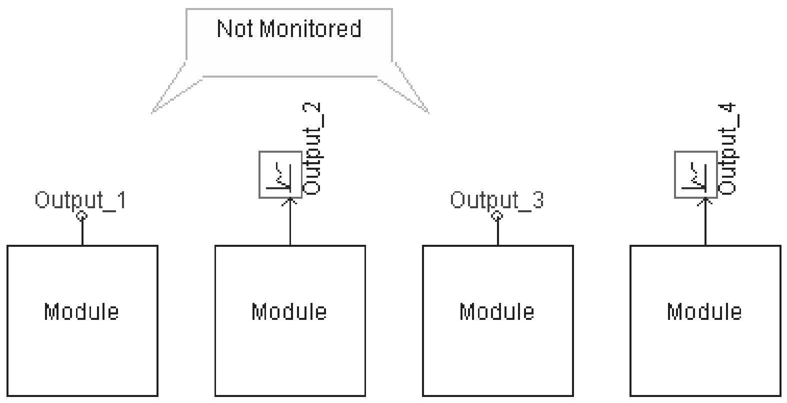
See *The Parameters Section* in the chapter called *Component Design* in this manual for more details on the proper design of module parameters, specifically the definitions of *Literal*, *Constant* and *Variable* type parameters, and how they pertain to multiple instance modules.

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.

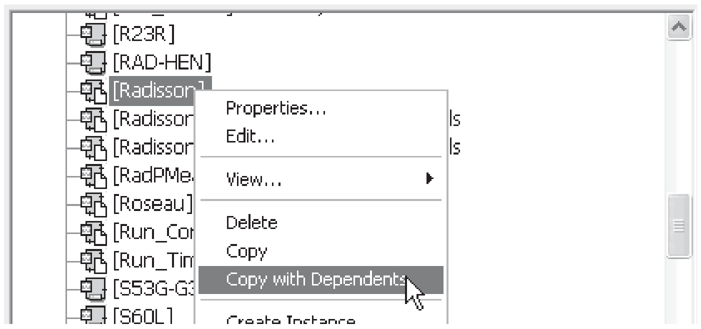
Copy with Dependents

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 hierarchal 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.

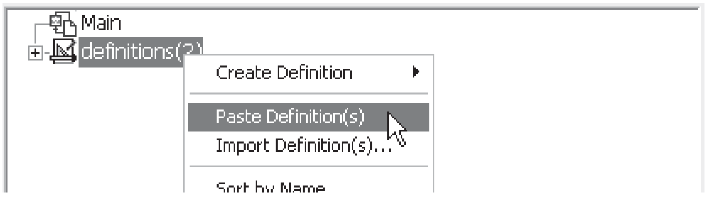
This feature is provided mainly for inter-project transfer of modules, alleviating the need to manually reconstruct the module and its hierarchy. *Copy with Dependents* may also be used for copying and pasting within the same project, providing a mechanism to create a unique copy of the module (i.e. not multiply-instanced), based on new unique definitions. Note however, that additional steps must be taken when doing this.

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 in the secondary window. 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).

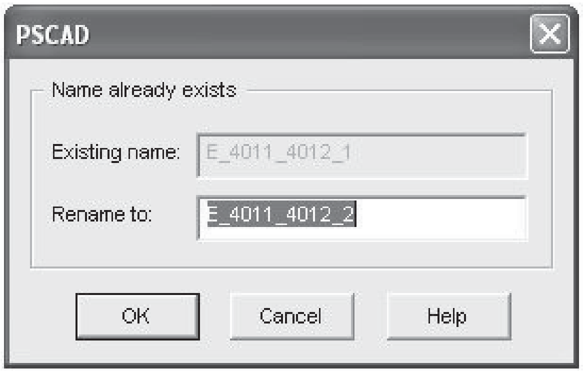
Intra-Project

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.

Import Notes

There are some important issues to consider when using the *Copy with Dependents* feature: As you may know, it is illegal for two definitions of the same name to exist in the same project. This complicates matters, as when you perform a copy with dependents, you are attempting to add a group of definitions (by pasting) into an existing list of definitions. What if one or all of these existing definitions shares a name with one or all of the definitions being pasted? No two definitions of the same name can exist, which means that the pasted definitions must be renamed if needed.

This is taken care of by PSCAD. If a duplicate definition name is detected, a dialog will appear, asking the user to rename the definition.


PSCAD

Name already exists

Existing name: E_4011_4012_1

Rename to: E_4011_4012_2

All definitions that require a renaming retain their links to the original definitions. What this means is that when the parent module (or top module) is instantiated on the Circuit canvas, all module instances based on definitions that were renamed, will still be linked back to the original.

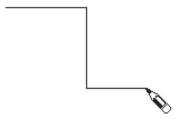
For example, if one of the dependent module definitions was named *A*, and its parent was copied to a project that already has a module definition called *A*, then the user would prompted to rename the *A* definition to be pasted to say *A_1*. However, when the parent module is instantiated on the canvas, the instance or instances of module A will still be linked to definition *A*, instead of *A_1*. The user must manually re-link these module instances. See *Linking and Re-linking Definitions* in this chapter on how to do this.

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 main toolbar or press **Ctrl + W** on your keyboard. With the project open in Circuit 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.



To turn-off Wire Mode, either press the **Esc** key, 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 Circuit canvas, or definitions may be instantiated by drag and drop from the workspace window.

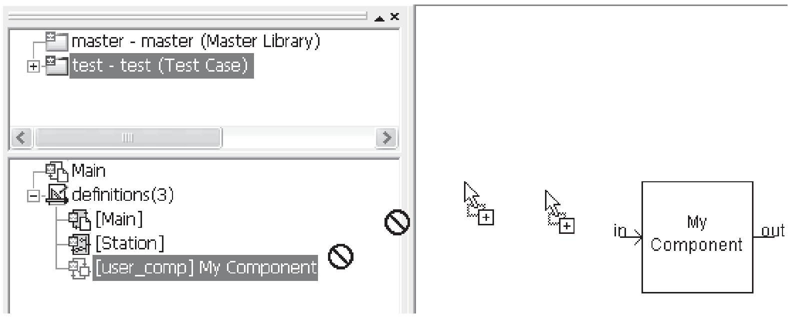
The drag and drop feature uses mouse pointer icons to indicate whether or not a dragged object may be placed under the current mouse position:

- 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 your left mouse button.
3. **Drag** the mouse pointer to a blank area on the Circuit canvas and release the button to **drop**.

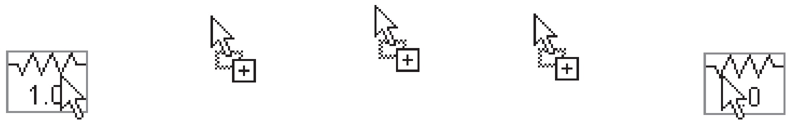


Drag and Drop onto Circuit Canvas from Workspace

Copy a Component Instance

All component instances appearing on the Circuit 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 Circuit canvas and release the mouse button to paste.

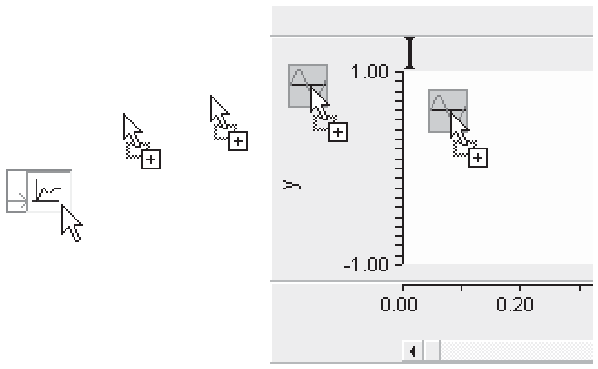


Copying a Component Instance

Adding a Curve to a Graph

Curves can be added directly to graphs using drag and drop.

1. Hold down the **Ctrl** key (use the **Shift** key if *Use shift key to create controls & curves* is enabled in the *Graphic Settings* 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.

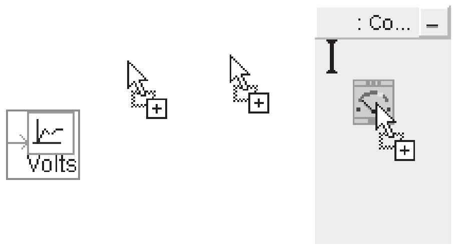


Adding a Curve to a Graph from a Output Channel

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 *Use shift key to create controls & curves* is enabled in the *Graphic Settings* 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.

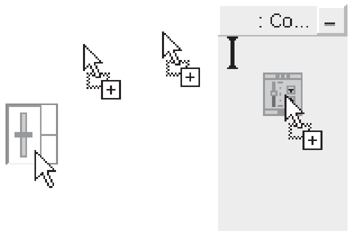


Adding a Meter to a Control Panel from an Output Channel

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 *Use shift key to create controls & curves* is enabled in the *Graphic Settings* 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 an Output Channel

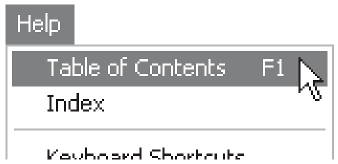
Move/Copy Meters and Curves between Graphs/Panels

Once a curve or meter interface has been placed in a graph or control panel, it may then be copied or moved to any other graph or control panel present in the project. In fact, control/meters interfaces and curves can be moved/copied within the same graph or control panel.

- To *copy* a curve/meter interface from one graph/control panel to another, 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 from one graph/control panel to another, left-click and hold the object, 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 main menu 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.

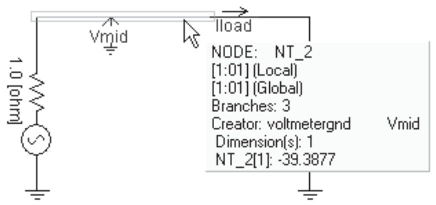
In addition to the above, other dialog windows in PSCAD will possess a **Help** button (usually in the bottom right corner). Click this button to bring-up the online help topic specific to that dialog. See *PSCAD Manual | The Application Environment | PSCAD On-Line Help System* for more details on the 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 data quantities during a simulation.

To monitor an electrical or data signal during a run:

1. **Left-click** on a blank part of the Circuit canvas (this is to ensure that the Circuit 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:



NODE: NT_2
[1:01] (Local)
[1:01] (Global)
Branches: 3
Creator: voltmetergnd
Dimension(s): 1
NT_2[1]: -39.3877

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.

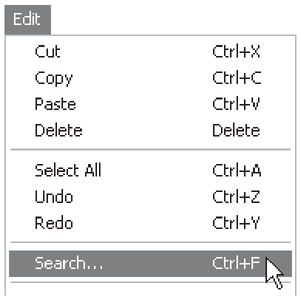
If the mouse pointer is held over a wire carrying an electrical signal, the flyby will indicate the voltage in kilovolts at that electrical node. The flyby shown above indicates that this is EMTDC node *NT_2*, which is a scalar quantity (i.e. single-phase node). The value of the node voltage at the instant 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.

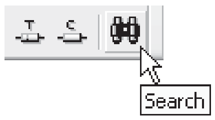
SEARCHING

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:

- Select **Edit** | **Search...** from the main menu bar



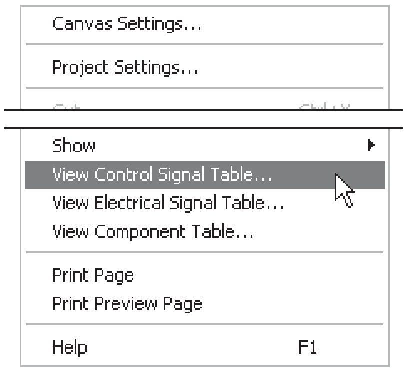
- Press **Ctrl + F** on your keyboard, while the Circuit window is open
- Press the **Search** button in the Query Bar.



For more details on searching, see *Searching* in chapter 11 of this manual.

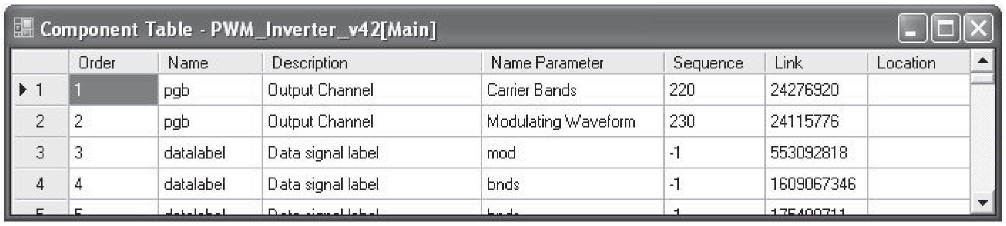
COMPONENT AND SIGNAL TABLES

The component and signal table viewers provide a module-specific list of either components and/or signals (depending on which viewer you are looking at), organized in spreadsheet form. Right-click on a blank part of the Circuit canvas in order to invoke either viewer. Select either **View Control Signal Table...**, **View Electrical Signal Table...** or **View Component Table....**



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.


Component Table - PWM_Inverter_v42[Main]

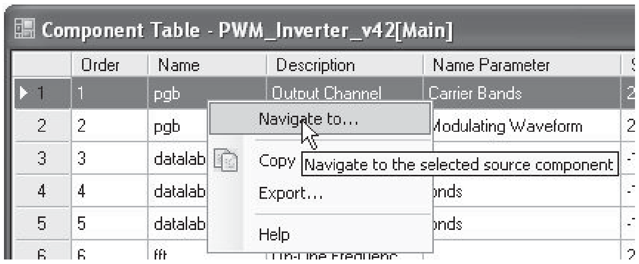
	Order	Name	Description	Name Parameter	Sequence	Link	Location
1	1	pgb	Output Channel	Carrier Bands	220	24276920	
2	2	pgb	Output Channel	Modulating Waveform	230	24115776	
3	3	datalabel	Data signal label	mod	-1	553092818	
4	4	datalabel	Data signal label	bnds	-1	1609067346	

The columns displayed in this viewer are described below:

- **Order**: Simple order number for display.
- **Name**: The component definition name.

- **Description**: The description of the component definition. See *Editing Definition Properties* in this chapter.
- **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.
- **Location**: The location within the system dynamics where the instance resides. This can be either DSDYN or DSOUT.

This viewer also includes a feature by which you may navigate directly to any component instance in the list. Simply right-click on the row representing the Instance and select **Navigate to...**.

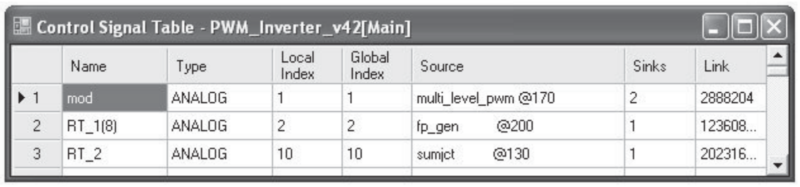


Component Table - PWM_Inverter_v42[Main]

	Order	Name	Description	Name Parameter
1	1	pgb	Output Channel	Carrier Bands
2	2	pgb		Modulating Waveform
3	3	datalab		
4	4	datalab		onds
5	5	datalab		onds
6	6	fft	On-Line Frequenc	

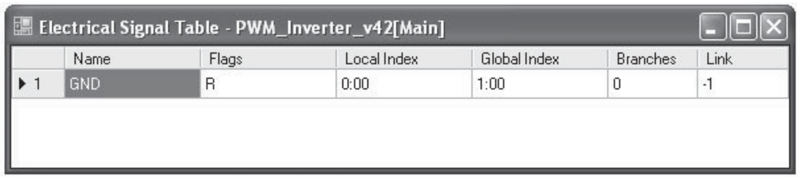
Signal Tables

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 is given.



Control Signal Table - PWM_Inverter_v42[Main]

	Name	Type	Local Index	Global Index	Source	Sinks	Link
1	mod	ANALOG	1	1	multi_level_pwm @170	2	2888204
2	RT_1(8)	ANALOG	2	2	fp_gen @200	1	123608...
3	RT_2	ANALOG	10	10	sumjct @130	1	202316...



Electrical Signal Table - PWM_Inverter_v42[Main]

	Name	Flags	Local Index	Global Index	Branches	Link
1	GND	R	0:00	1:00	0	-1

The columns displayed in these viewers are described below:

Control Signals:

- **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 component instance from which the signal is sourced.
- **Sinks**: The number of points at which the signal is received.

Electrical Signals:

- **Name**: EMTDC electrical node name. That is, the name given to the electrical node by the PSCAD compiler for use by EMTDC.
- **Flags**: Electrical node type: S = Switched, R = Removable. See *Electrical Node Types* for more.
- **Local Index**: A local (i.e. module based) index number given to the signal. For example, **1:01** -> Subsystem #1, Node #1.
- **Global Index**: A global (i.e. project based) index number given to the signal.
- **Branches**: The total number of electrical branches attached to the node.

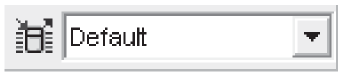
This viewer also includes a mechanism by which you may navigate directly to a control signal or electrical node in the list. Simply right-click on the row representing the instance and select **Navigate to...**.

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 **Runtime Bar**.

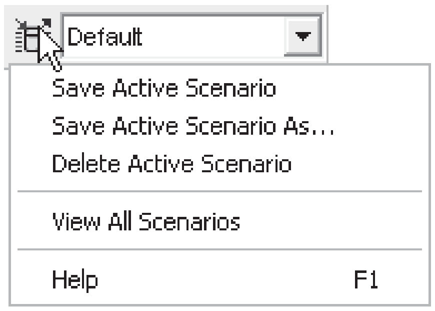


If this tool bar is not visible, go to the main menu and select **View | Runtime Bar**.

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** butto

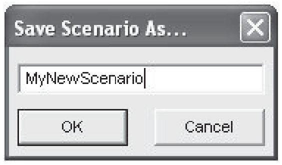
n to bring up the scenarios pop-up menu as shown below:

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 Active Scenario**. Also, ensure that all controls possess a distinct name; otherwise an error message will be posted.

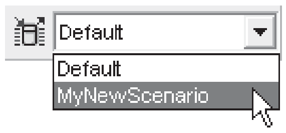


Select either **Save Active Scenario** or **Save Active Scenario As...**, depending on what you want to do. If this is the first scenario created for this project, select the latter.

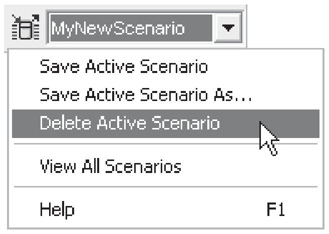
If creating a new scenario (i.e. **Save Active Scenario As...**), a dialog window will pop-up – enter a name for the scenario.



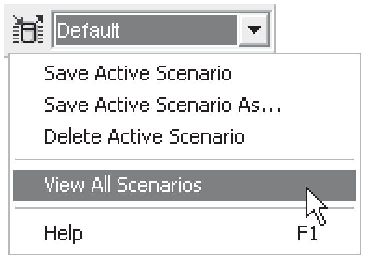
When finished, press the **OK** button. Your dynamic control settings will now be stored in this new scenario. Whenever you want to revert back to this scenario, simply left-click the down arrow on the scenario drop list and select it.



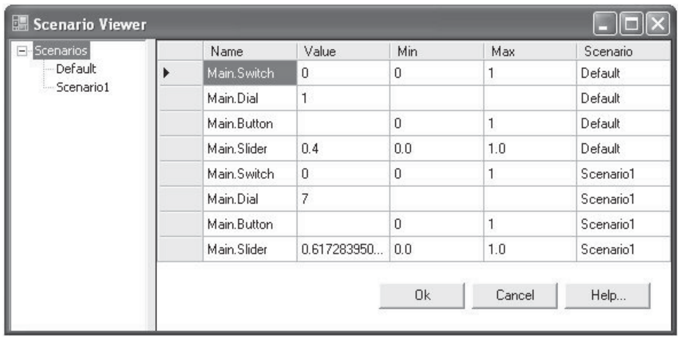
Delete Active Scenario
You can delete any scenario by first making the scenario to be deleted the active scenario. Then select **Delete Active Scenario** from the scenario pop-up menu as shown below:



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 **View All Scenarios** from the scenario pop-up menu as shown below:



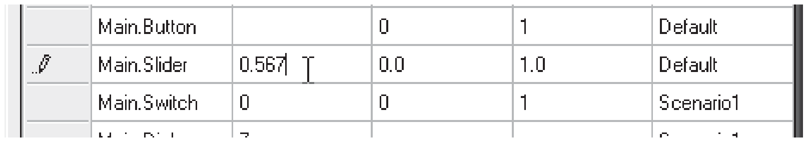
Features and Functions
The scenario viewer window contains a few simple features to help view and organize scenario data. The viewer consists of two sections: A scenarios tree on the left, which displays all scenarios saved in the project and a data sheet view to the right. The data sheet view lists the settings (specific to whatever scenario is selected in the scenario tree) of all control interfaces in the project.



Name	Value	Min	Max	Scenario
Main.Switch	0	0	1	Default
Main.Dial	1			Default
Main.Button		0	1	Default
Main.Slider	0.4	0.0	1.0	Default
Main.Switch	0	0	1	Scenario1
Main.Dial	7			Scenario1
Main.Button		0	1	Scenario1
Main.Slider	0.617283950...	0.0	1.0	Scenario1

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 are 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:

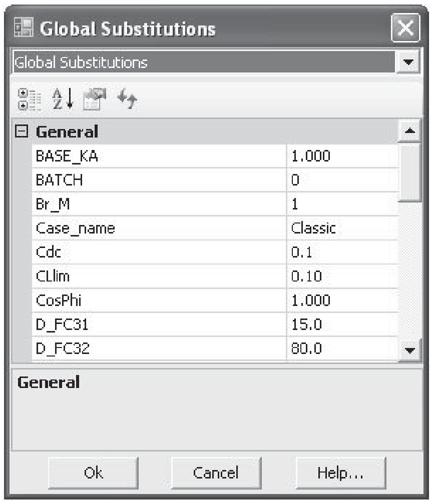
```
$(<Key>)
```

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.

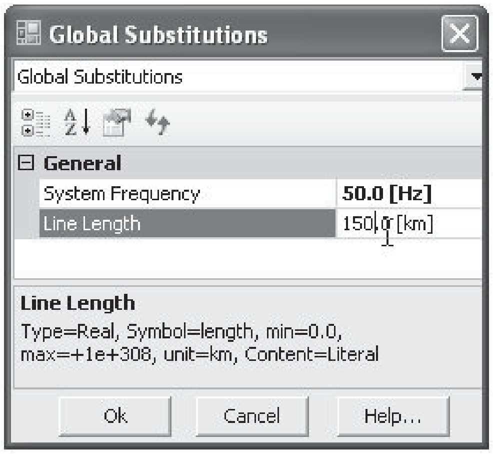
Viewing Defined Global Substitutions
In the workspace window, right-click on a project and select **Global Substitutions** | **View...** (or simply press **Ctrl + G** on your keyboard), to bring up the *Global Substitutions* dialog window.



BASE_KA	1.000
BATCH	0
Br_M	1
Case_name	Classic
Cdc	0.1
CLlim	0.10
CosPhi	1.000
D_FC31	15.0
D_FC32	80.0

General

Existing global substitution values may be modified directly within the dialog in the same manner you would edit a component input parameter. 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 *150 km* in a project. This can be modified as follows in the global substitutions dialog:



System Frequency	**50.0 [Hz]**
Line Length	150.0 [km]

Line Length
Type=Real, Symbol=length, min=0.0, max=+1e+308, unit=km, Content=Literal

Adding/Editing Global Substitutions

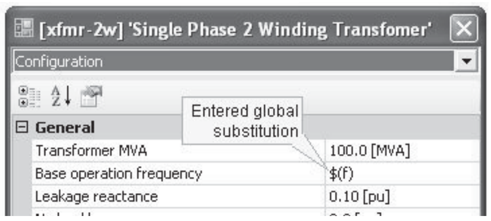
Global substitutions are handled in a very similar manner as component input parameters, in that they are defined as 'parameters' of the *Main* page. As such, the user can add new, and edit existing global substitutions by using the parameter editor for the *Main* module definition (i.e. the Parameters section).

To add or edit a global substitution, right-click on a project and select **Global Substitutions** | **Edit...**. 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*.

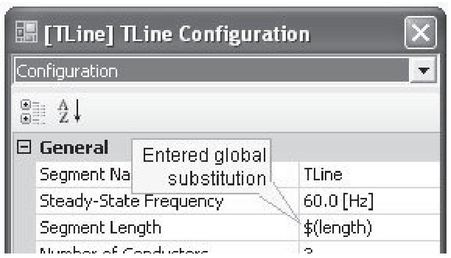
Utilizing Global Substitutions

To utilize these substitutions, enter either of the keys in the appropriate component input parameter field, according to the syntax given above:


[xfmr-2w] 'Single Phase 2 Winding Transfomer'
Configuration
General
Entered global substitution

Transformer MVA	100.0 [MVA]
Base operation frequency	$(f)
Leakage reactance	0.10 [pu]

Entering *f* in Single-Phase Transformer Parameters Dialog


[TLine] TLine Configuration
Configuration
General
Entered global substitution

Segment Na	TLine
Steady-State Frequency	60.0 [Hz]
Segment Length	$(length)

Entering *length* in Transmission Line Configuration Dialog

UNIT SYSTEM

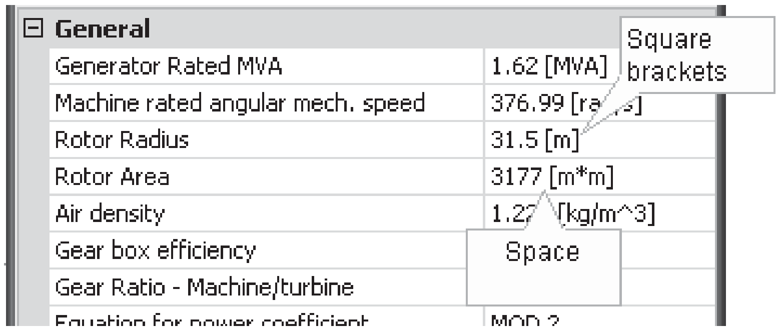
Component parameter units perform limited conversion and scaling, depending on the unit entered and the defined default (or *Target*) unit for that particular input parameter. The unit system includes base units for time, length, weight and speed (both translational and rotational), as well as electrical 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 [ra s]
Rotor Radius	31.5 [m]
Rotor Area	3177 [m*m]
Air density	1.2 [kg/m^3]
Gear box efficiency	
Gear Ratio - Machine/turbine	
Equation for power coefficient	MOD 2

Examples of Units Entered in Component Input Parameters

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
	Horse-power	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		Cycles per second
	Metre	m		Length in metres
	Inch	in	1 in = 0.0254 m	Length in inches
Length	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
	Radians	rad		Angle in radians
Rotational	Revolutions	rev	1 rev = 2π rad	Angle in revolutions
	Degree	deg	1 deg = π/180 rad	Angle in degrees
	Revolutions per Minute	rpm	1 rpm = 1/60 rev/s	Rotational speed in revolutions per minute
	Revolutions per Minute	RPM	1 RPM = 1/60 rev/s	Rotational speed in revolutions per minute 2
Other	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}

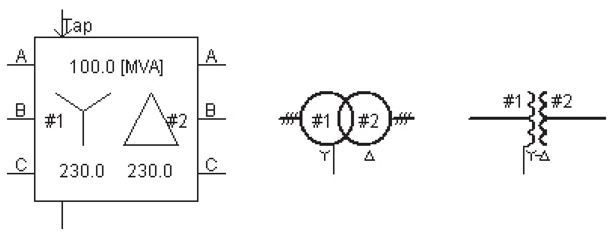
You can view the target units of any component by invoking the *View Properties* dialog: Right-click on the component and select **View Properties**.

Target Units
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.

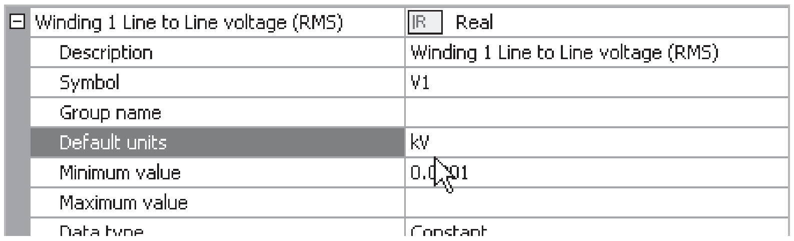
Target units are not limited to the base units alone, and may include prefixes by default (i.e. *kA*): In instances such as these, any prefixes in the target unit will be considered if further scaling is performed later on. In fact, this is quite common in the master library, where many target units 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)	IR Real
Description	Winding 1 Line to Line voltage (RMS)
Symbol	V1
Group name	
Default units	kV
Minimum value	0.0001
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.

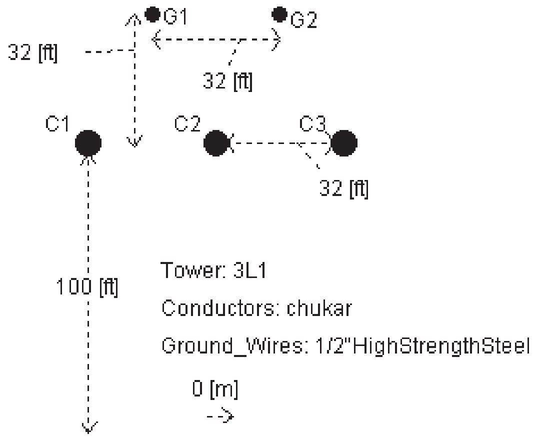
Unit Conversions

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.

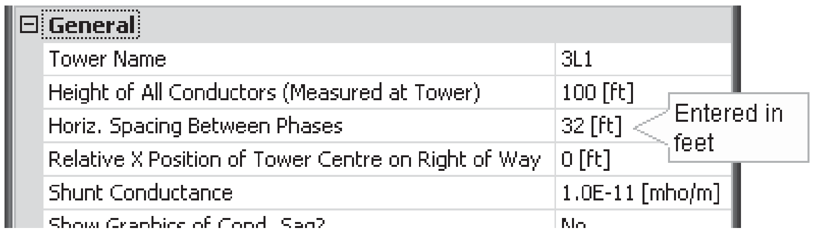
Converting units from one form to another is relatively straightforward; the only rule is that the conversion must take place within the same base unit class; for example, [m] to [ft] (both units measure length) or [sec] to [hr] (both units measure time). Valid prefixes may be included in the conversion 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

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
Correct:	[hp*min/MVA]	[MW*s/MVA]
Incorrect:	[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'.

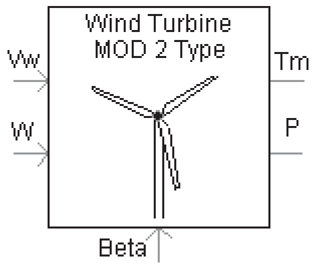
	Entered	Target
Correct:	[ft*ft]	[m*m]
Incorrect:	[ft^2]	

- Multiple 'divide' symbols are not allowed - only one 'divide' per compound unit may exist.

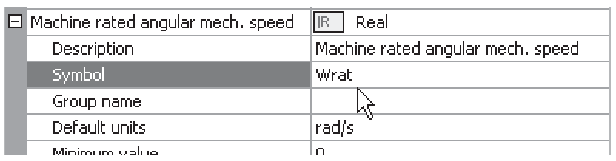
	Entered	Target
Correct:	[lb*ft/s]	[kg*m/s]
Incorrect:	[lb/s/ft]	

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



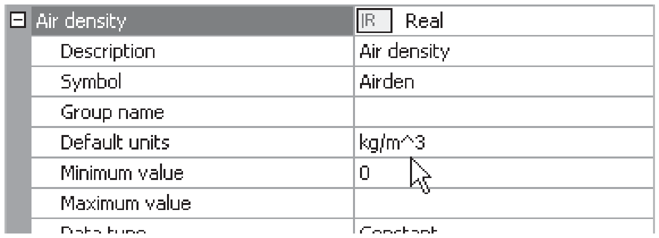
Machine rated angular mech. speed	IR Real
Description	Machine rated angular mech. speed
Symbol	Wrat
Group name	
Default units	rad/s
Minimum value	0

Input Field Property Settings (*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 2π *[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.



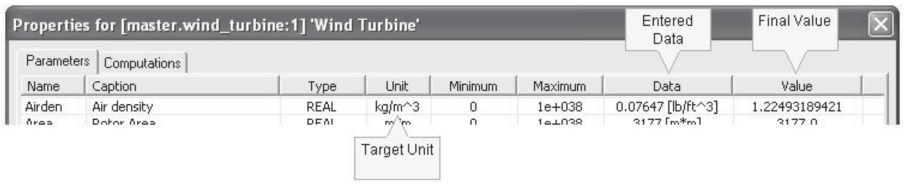
Air density	IR Real
Description	Air density
Symbol	Airden
Group name	
Default units	kg/m^3
Minimum value	0
Maximum value	

Input Field Property Settings (*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 a good idea 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.



Properties for [master.wind_turbine:1] 'Wind Turbine'

Name	Caption	Type	Unit	Minimum	Maximum	Data	Value
Airden	Air density	REAL	kg/m^3	0	1e+038	0.07647 [lb/ft^3]	1.22493189421
Area	Rotor Area	REAL	m*m	0	1e+038	3177 [m*m]	3177.0

Wind Turbine Air Density Parameter

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:

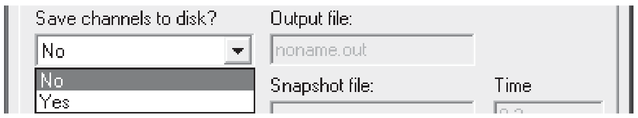
```
Test Case
    0.0000000000000          0.0000000000000          0.0000000000000
   0.10000000000000E-02      0.0000000000000          0.0000000000000
   0.20000000000000E-02     0.86727047422974         0.86727047422974
   0.30000000000000E-02      1.6650619394029          1.7674567004163
   0.40000000000000E-02      1.9545665157651          2.2542364437667
   0.50000000000000E-02      2.0221282586499          2.8373003607589
   0.60000000000000E-02      1.9264422562260          4.0202957514613
   0.70000000000000E-02      2.3912531836698          5.5547061594874
   0.80000000000000E-02      2.8769239640036          6.5791178737352
   0.90000000000000E-02      2.8473253982397          6.9456914675731
   0.10000000000000E-01      2.3232656122503          7.1369144410646
   0.11000000000000E-01      1.2446128466462          7.1669414384869
   0.12000000000000E-01      1.6208317681211          7.0023750435365
   0.13000000000000E-01      1.6458908605563          7.1529925186834
   0.14000000000000E-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. EMTDC output files are given the extension '*.out' and are stored in the project temporary directory.

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.



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.out*, and there are three files as described above, the output files would be named *abc_01.out*, *abc_02.out* and *abc_03.out*. This sequential numbering is important when identifying data columns.

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 34.

Here is a simple formula to help identify an output file column:

Single Output File:

Output Channel # = Output File Column # - 1

Multiple Output Files:

Output Channel # = Output File Column # - 1 + (10 x Output File #)

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 specialty 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 transferrable 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, Rotary Switch, Switch and Push Button

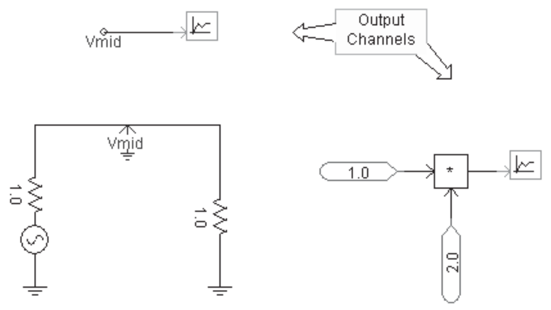
- **Recorders**: Output Channel and RTP/COMTRADE Recorder
- **Display Devices**: Control Panel, Graph, XY Plot, PolyMeter, PhasorMeter and Oscilloscope.

Each runtime object performs a specific task, and may be used in combination with others to control and/or display data.

Channelling Output Data

Channelling output refers to directing 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 Circuit canvas:

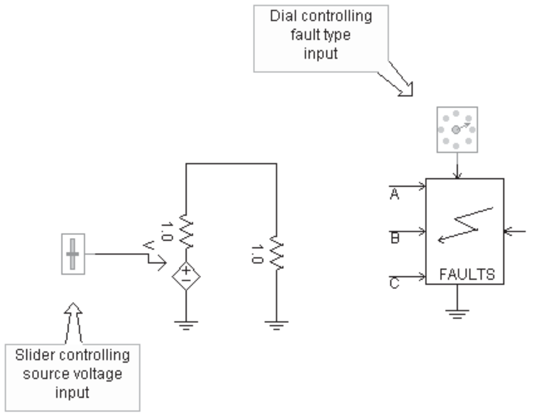
Output channel components cannot be directly connected to an electrical wire.



Controlling Input Data

Controlling input data refers to using one of the control type runtime objects (i.e. Slider, Rotary Switch, Switch or Push Button) as a source to control a data signal. This is accomplished by simply adding a control object to the Circuit canvas. This signal can be input at any valid location in the schematic as shown below:

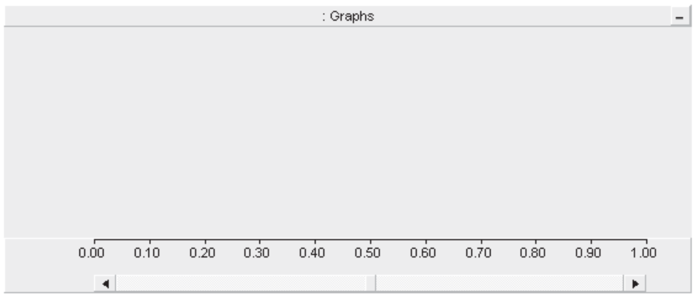
The control objects cannot be manually adjusted (as they appear above) until they are linked to a control interface. See *Online Controls and Meters* in this chapter for details.


Dial controlling fault type input
1.0
V
1.0
A
B
C
FAULTS
Slider controlling source voltage input

GRAPH FRAMES

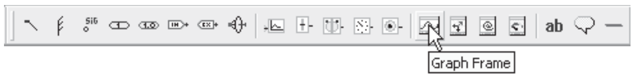
A Graph Frame is a special runtime object used for encapsulating *Overlay* or *PolyGraphs*, and can be placed anywhere on the Circuit canvas. Once a graph frame has been added, you may then proceed to add as many *Graphs* to it as you wish.

Graph frames are used exclusively for plotting curves versus time. That is, the graph frame horizontal axis is always the EMTDC 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 the Circuit view. Right-click on a blank portion of the page and select **Add Component | Graph Frame**, or press the **Graph Frame** button in the Control Palette bar.

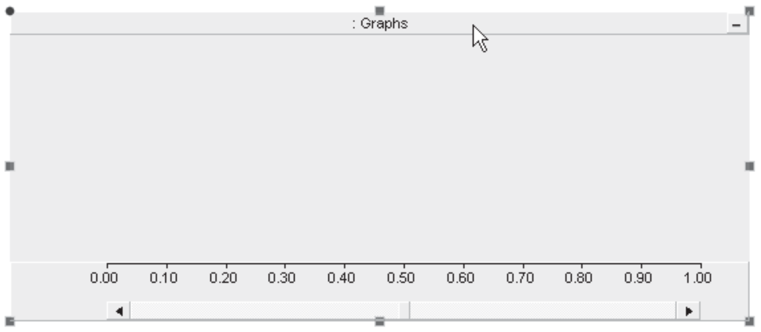


If you cannot see the Control Palette, go to the **View** menu and ensure *Control Palette* is selected.

Moving and Resizing a Graph Frame

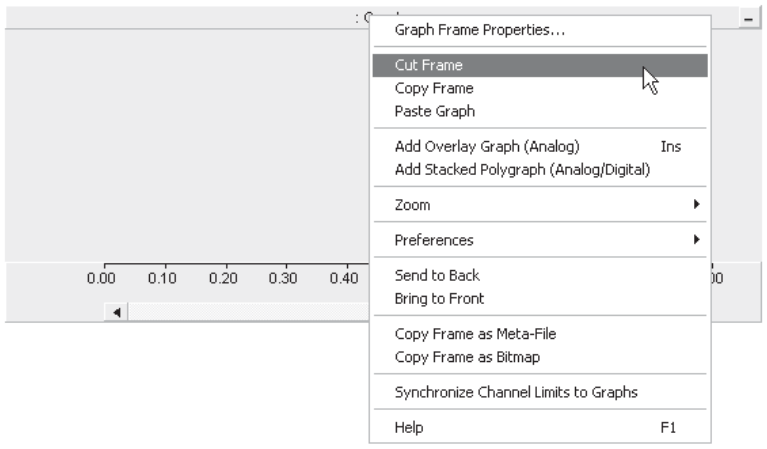
To move a graph frame, move the mouse pointer over the title bar and then left-click and hold. Drag the frame to where it is to be placed and release the mouse button.

To resize, 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, hold and then drag, then move the pointer to resize.

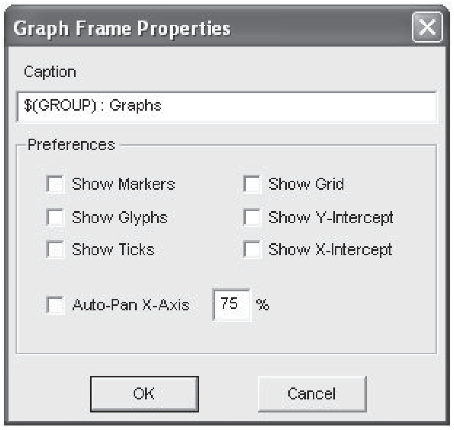
Cut/Copy Frames
Right-click over the graph frame title bar and select **Cut Frame** or **Copy Frame** respectively.



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 Circuit 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 **Graph Frame Properties....**


Graph Frame Properties

Caption

$(GROUP) : Graphs

Preferences

- [ ] Show Markers
- [ ] Show Glyphs
- [ ] Show Ticks
- [ ] Show Grid
- [ ] Show Y-Intercept
- [ ] Show X-Intercept
- [ ] Auto-Pan X-Axis 75 %

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: 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*.

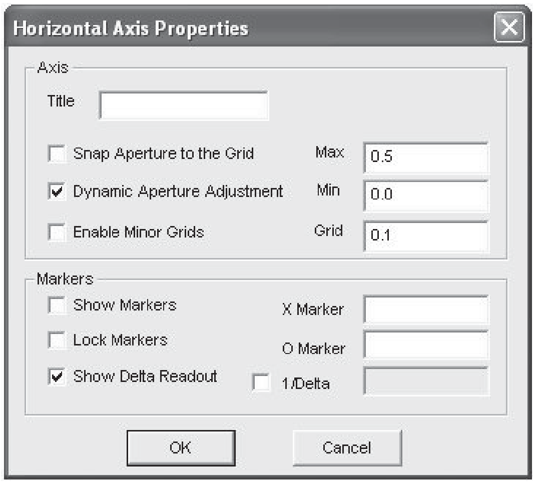
Preferences:

- **Show 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...**.

Adjusting the horizontal axis properties will affect all graphs in the frame.



Snap Aperture to the Grid: unchecked
Dynamic Aperture Adjustment: checked
Enable Minor Grids: unchecked
Max: 0.5
Min: 0.0
Grid: 0.1
Show Markers: unchecked
Lock Markers: unchecked
Show Delta Readout: checked
X Marker:
O Marker:
1/Delta: unchecked

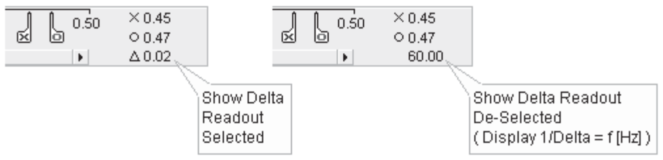
Axis:

- **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. Δt) between the *X* and *O* markers. When this

option is disabled, the equivalent 1/∆t value (i.e. frequency in Hz) will be displayed.

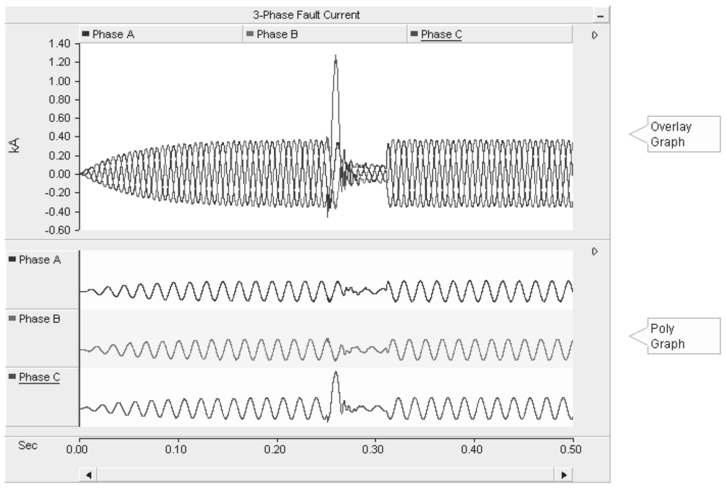

0.50
× 0.45
○ 0.47
∆ 0.02
Show Delta Readout Selected
0.50
× 0.45
○ 0.47
60.00
Show Delta Readout De-Selected (Display 1/Delta = f [Hz])

- **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) between markers.

GRAPHS

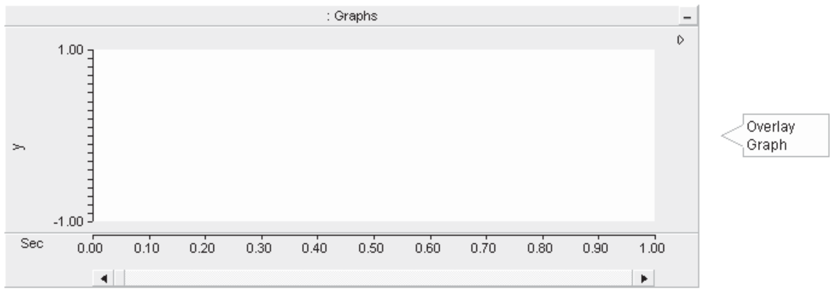
A Graph is a special runtime object, which can reside only inside a graph frame. There are two types of graphs available: *Overlay* and *PolyGraphs*. A single graph may hold and display multiple *Curves*, where all curves in a graph are based on the same y-axis scale.

The following illustrates a graph frame with an overlay graph at the top and a poly graph below it.



Adding Graphs to a Graph Frame

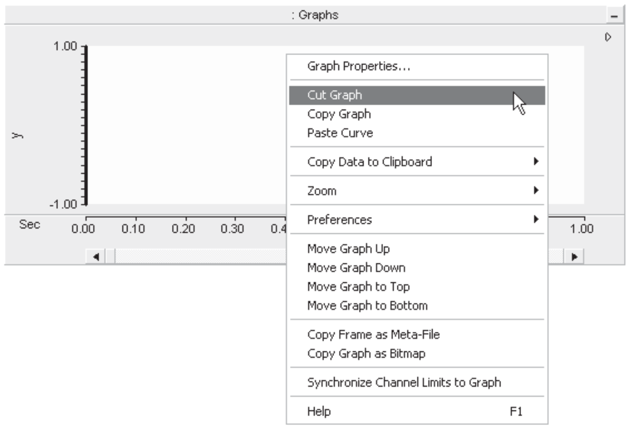
Graph frames may accommodate single or multiple graphs. To add one or more graphs, right-click on the frame title bar and select **Add Overlay Graph (Analog)** or **Add Poly Graph (Analog/Digital)**. You can also add an overlay graph directly by pressing the Insert key on your keyboard with your mouse 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.



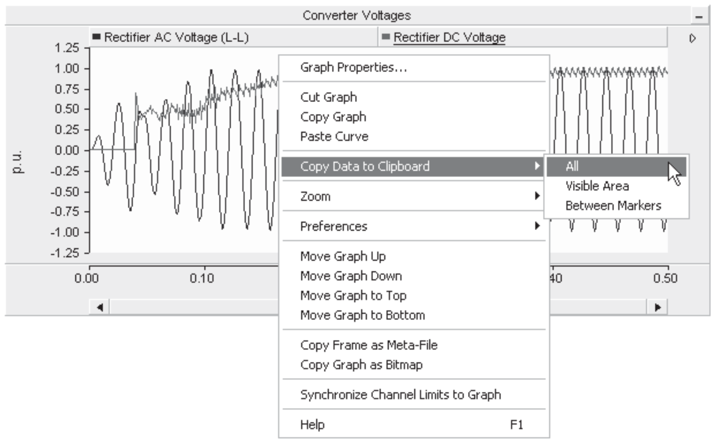
Once a graph has been cut or copied it may then be pasted into the same or another graph frame.

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.



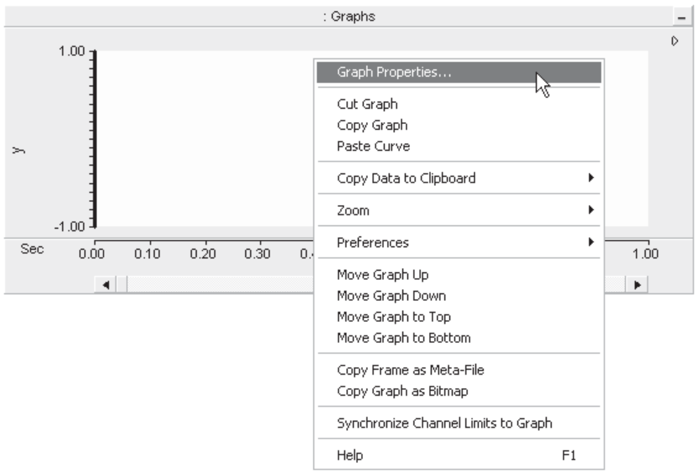
The data is copied as *Comma Separated Variables (*.csv)* format for easy migration into common data analysis software.

Overlay Graphs

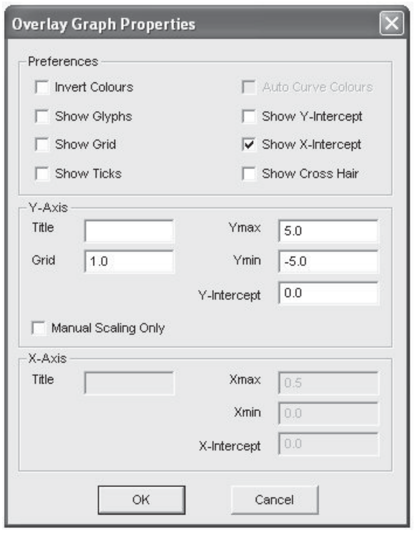
Overlay graphs are the most common and familiar type of online plotting tool in PSCAD. These graphs display measured data as a function of time, where multiple curves may be added (or *overlaid* on top of each other) onto a single graph.

Adjusting Overlay Graph Properties

Left double-click on the desired overlay graph, or right-click on the graph and select **Graph Properties....**



This will bring up the Overlay Graph Properties dialog window as shown below:


Overlay Graph Properties

Preferences
- [ ] Invert Colours
- [ ] Auto Curve Colours
- [ ] Show Glyphs
- [ ] Show Y-Intercept
- [ ] Show Grid
- [✓] Show X-Intercept
- [ ] Show Ticks
- [ ] Show Cross Hair

Y-Axis
Title:
Ymax: 5.0
Grid: 1.0
Ymin: -5.0
Y-Intercept: 0.0
- [ ] Manual Scaling Only

X-Axis
Title:
Xmax: 0.5
Xmin: 0.0
X-Intercept: 0.0

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.

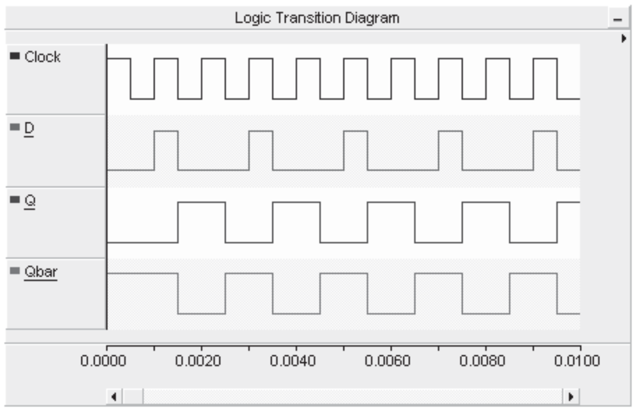
- **Auto Curve Colours**: Select this option to use automatic colouring of curves in the graph. You cannot change curve colour manually when this option is selected.
- **Show Y-Intercept**: Select this option to display the y-intercept (horizontal) intercept line. The y-intercept line can be adjusted 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:

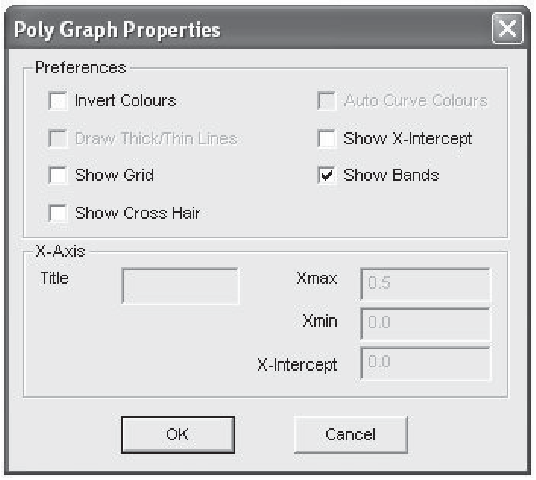
- **Title**: Enter text for display as the graph title (located on the left side of the graph).
- **Grid**: Specifies the y-axis grid interval. To view the y-axis grid lines, select the option Show Grid described above.
- **Ymin**: Specifies the minimum y-axis viewing limit on the graph.
- **Ymax**: Specifies 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.

Poly Graphs

Poly graphs 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 poly graph 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 poly graph, or right-click on the graph and select **Graph Properties....** This should bring up the Poly Graph Properties dialog:


Poly Graph Properties

Preferences

- [ ] Invert Colours
- [ ] Auto Curve Colours
- [ ] Draw Thick/Thin Lines
- [ ] Show X-Intercept
- [ ] Show Grid
- [✓] Show Bands
- [ ] Show Cross Hair

X-Axis

Title

Xmax 0.5

Xmin 0.0

X-Intercept 0.0

OK Cancel

There are various parameters that may be edited though this window, each of which are described below.

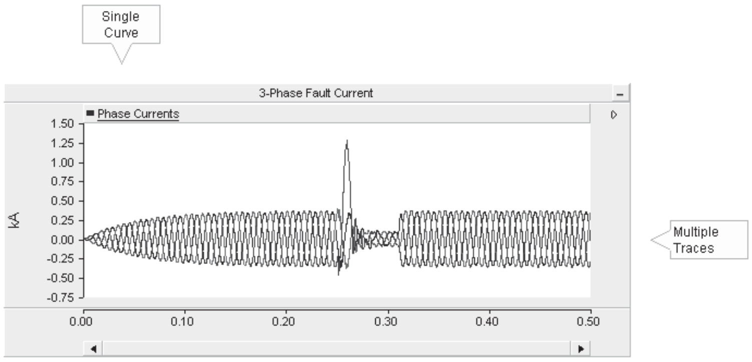
Preferences:

- **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 poly graphs.
- **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 a special runtime object best described as a graphical representation of a string of data points, where each point is associated with a simulation time 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* 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*.

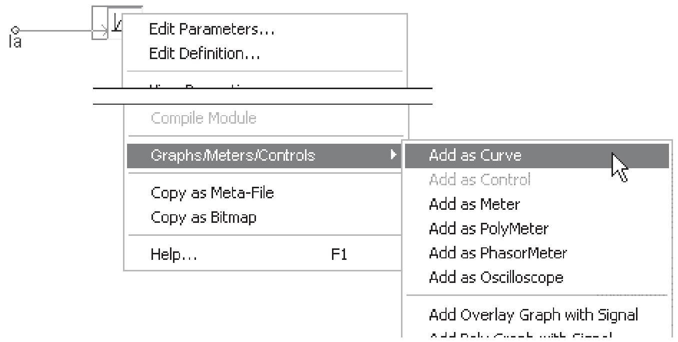
Adding a New Curve to a Graph

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.

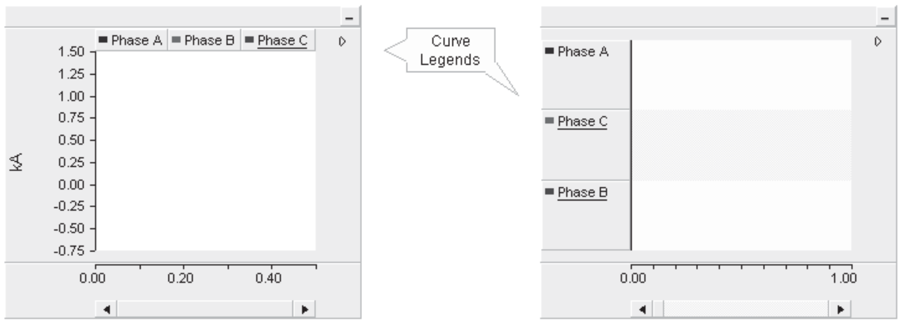
2. **Graphs/Meters/Controls Method**: Right-click on the *Output Channel* component from which you would like to extract the

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.

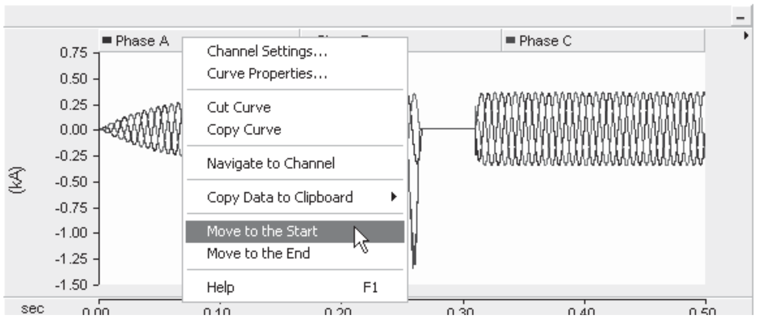


Curve Order

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

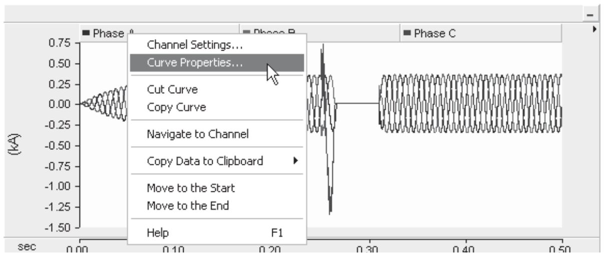
If a simulation has been run and a particular curve contains data, you have the option of copying all or a portion of this single curve data set to the clipboard. Right-click over the corresponding curve name, select **Copy Data to Clipboard** and then select one of the following from the pop-up menu:

- **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.

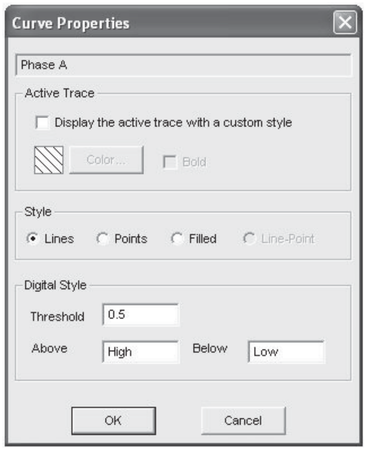
The data is copied as *Comma Separated Variables (*.csv)* format for easy migration into common data analysis software.

Adjusting Curve Properties

Left double-click on the desired curve in the curve legend, or right-click the curve and select **Curve Properties....**



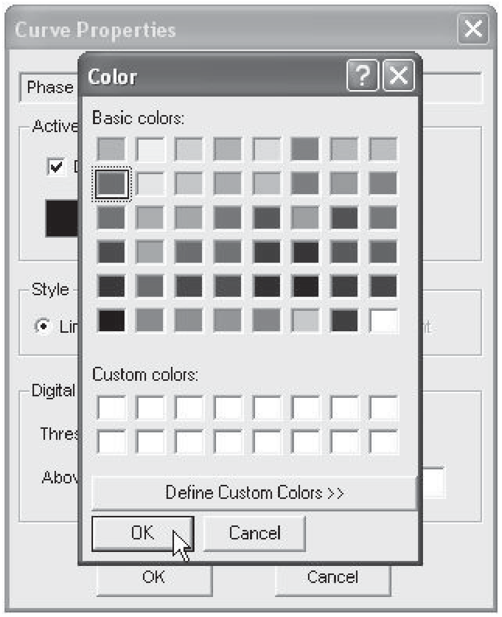
This should bring up the Curve Properties dialog window.



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 bolded. 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 poly graph. 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.

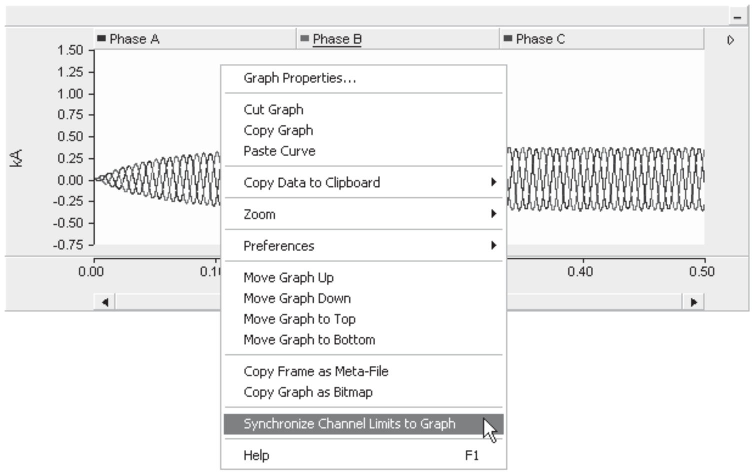
Adjusting Channel Settings

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

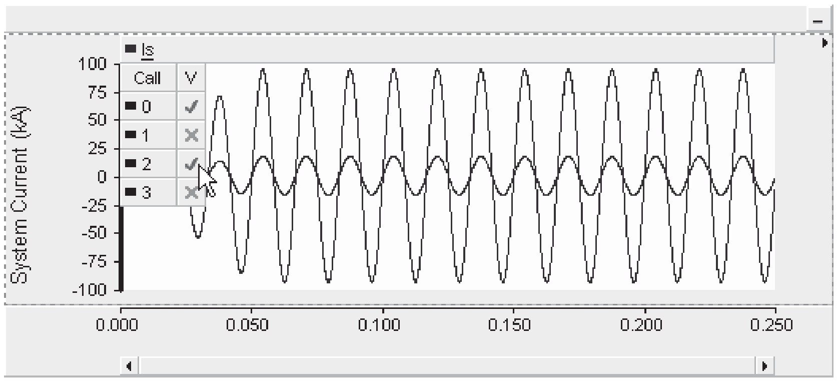
Once one or more curves have been added to an overlay graph, all corresponding *Output Channel* component **Min / Max** limits can be synchronized to the graph y-axis minimum and maximum limits.

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 technically only a single *Output Channel* object. If the curve associated with the *Output Channel* is placed in a graph that exists on a parent canvas, 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 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 Is is sourced from four different module instances that are based on the same definition. In this particular example, instance 0 (base 0) and instance 2 are being displayed. You may choose to display all calls in a single graph, or you can place the Is curve in multiple graphs, and select a unique call to display in each graph.

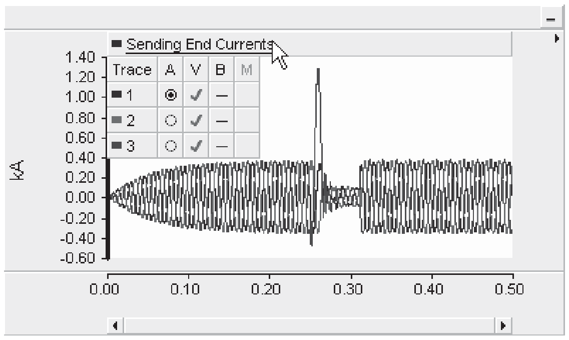
For more information on multiple instance modules, see Multiple Instance Modules (MIM) in chapter 5 of this manual.

TRACES

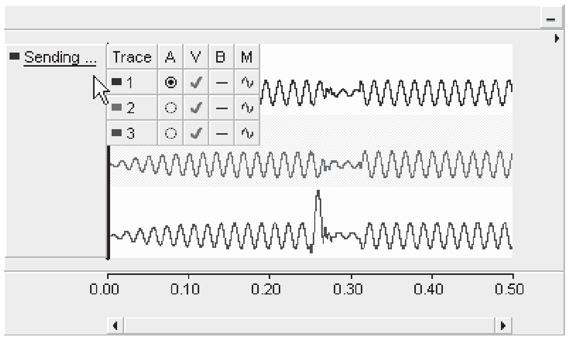
Array signal curves can be plotted online as a single entity, where each array element or 'sub-curve' is referred to as a *Trace*. Each trace may be enabled or disabled separately (i.e. shown or hidden).

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.



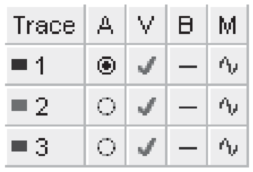
Trace	A	V	B	M
1	●	✓	—	
2	○	✓	—	
3	○	✓	—	



Trace	A	V	B	M
1	●	✓	—	∿
2	○	✓	—	∿
3	○	✓	—	∿

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 allowing for easy access to certain trace properties. These columns are described below:

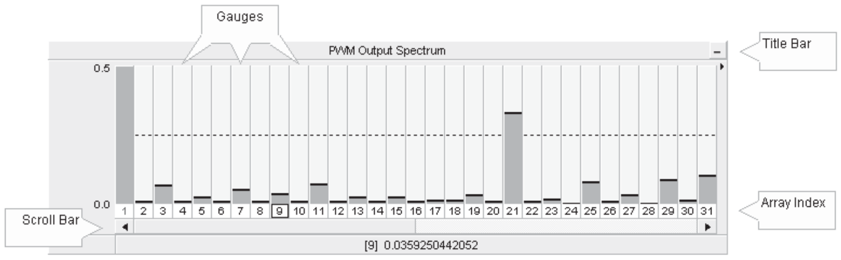


Trace	A	V	B	M
■ 1	◉	✓	—	∿
■ 2	○	✓	—	∿
■ 3	○	✓	—	∿

- **Trace:** Indicates the trace number and corresponding colour setting. The number corresponds to the array index number of the multi-signal curve. To change individual trace colour, 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 poly graph. 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 *Poly Graphs*.

POLYMETERS

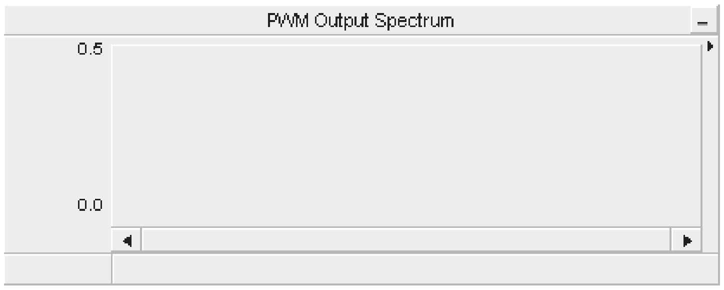
A polymeter is a special runtime object used specifically for monitoring a single, multiple-trace curve. The polymeter dynamically displays the magnitude of each trace in bar type format (called gauges), which results in an overall appearance similar to a spectrum analyzer. The power of this device lies 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 *On-Line Frequency Scanner (FFT)* component.



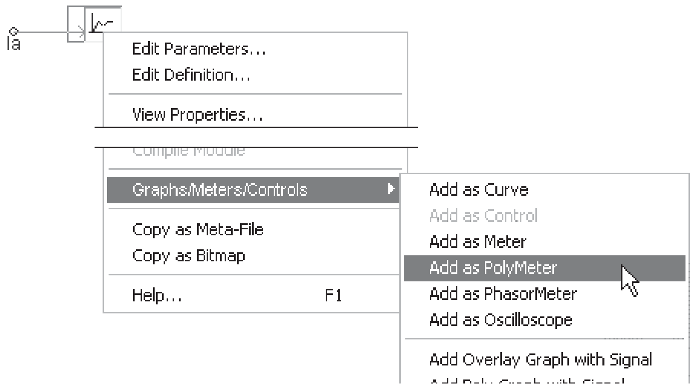
The relative gauge width is fixed, and so if the polymeter is not wide enough to display all data, a simple horizontal scroll bar is provided. An array Index display is included directly under the gauges for easy identification.

Polymeters will appear as a blank container (as shown below) until the project is compiled and run.

Polymeters are special objects that cannot be added directly from the toolbar. Each polymeter is linked with a single curve from a single *Output Channel* component.



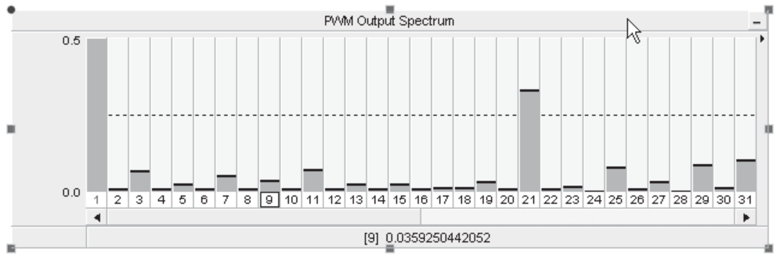
Adding a PolyMeter
To add a polymeter, right-click on an *Output Channel* component within the Circuit canvas and select **Graphs/Meters/Controls | Add as PolyMeter**.



The polymeter 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 Circuit canvas.

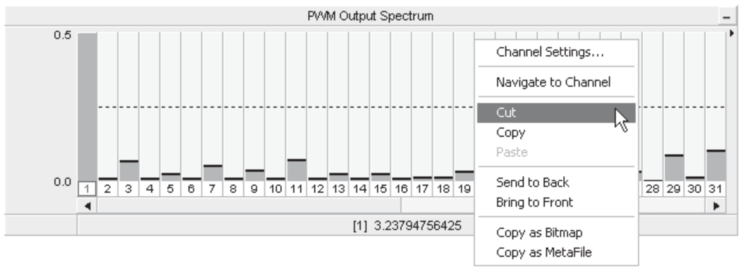
Moving and Resizing a PolyMeter
To move a polymeter, move the mouse pointer over the title bar and then left-click and hold. Drag the meter to where it is to be placed and release the mouse button.

To resize, move the mouse pointer over the title bar and left-click to select the polymeter. 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, then move the pointer to resize.

Cut/Copy PolyMeter
Right-click over the polymeter and select **Cut** or **Copy** respectively.



Once a polymeter has been cut or copied it may then be pasted to another Circuit canvas in the project.

Paste PolyMeter
Cut or copy a polymeter as described above. Right-click over a blank area in Circuit view and select **Paste**. A polymeter may be pasted multiple times.

Copy PolyMeter as Meta-File or Bitmap

The entire polymeter display can be copied to the Windows clipboard in either meta-file (*.wmf) or bitmap (*.bmp) format. Right-click over the polymeter title bar and select **Copy as Bitmap** or **Copy as Meta-File**. Go to your report document and paste the image.

Navigate to Channel

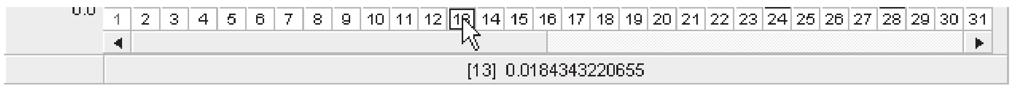
You can navigate directly to the associated *Output Channel* component by selecting this option. Right-click over the polymeter and select **Navigate to Channel**. PSCAD will automatically find the output channel and highlight it.

Adjusting Channel Settings

The y-axis properties of the polymeter are set in the corresponding *Output Channel* Properties dialog. This dialog may be accessed directly from the polymeter by right-clicking over the title bar and selecting **Channel properties....**

Displaying Specific Data

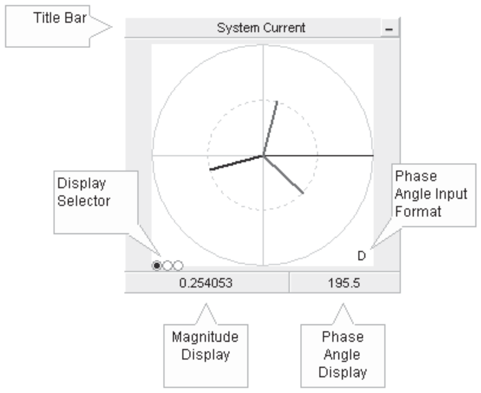
The magnitude of individual array elements can be displayed in the status bar at the bottom of the polymeter. To view the magnitude of that element, left-click on a particular index number in the array index.



[13] 0.0184343220655

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 *On-Line Frequency Scanner (FFT)* component.

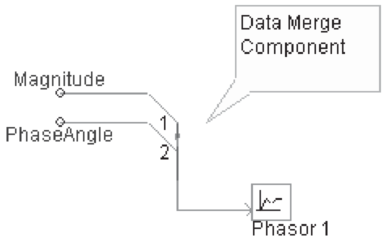


Phasormeters are special objects that cannot be added directly from the toolbar. Each phasormeter is directly linked with a single curve from a single *Output Channel* component. In the case of the phasormeter, a curve with at least two traces (magnitude and phase angle) is the minimum requirement for 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:

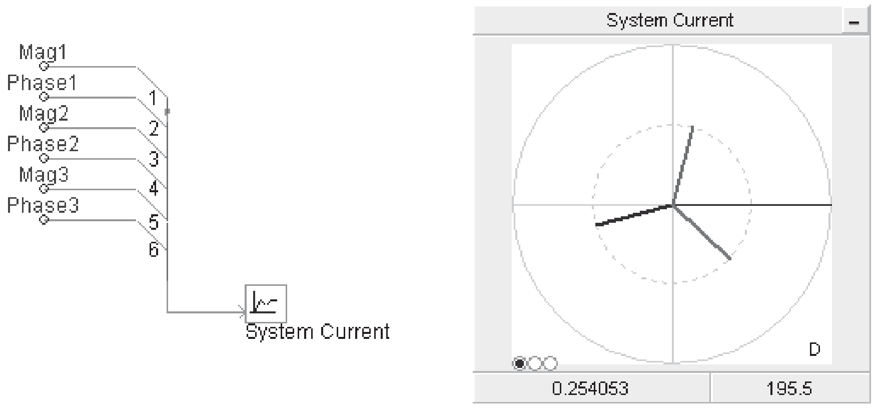
$$X\angle\varphi$$

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:

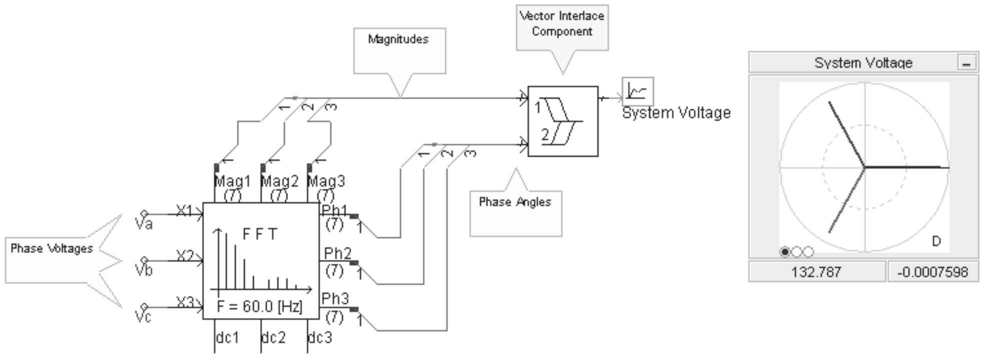


The phasormeter always assumes that the elements of the array are in the above displayed order. That is element *1* is the magnitude and element *2* is the phase angle.

The phasormeter 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:

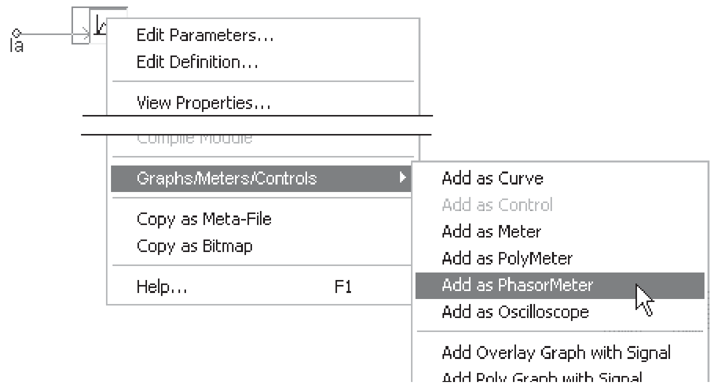


More often than not, the *On-Line Frequency Scanner (FFT)* component will be used to derive the polar quantities for display in this meter. If so, a special component, called the *Vector Interlace* component was developed to make it easier to prepare data for display. The image below illustrates a situation where it could be used:



Adding a PhasorMeter

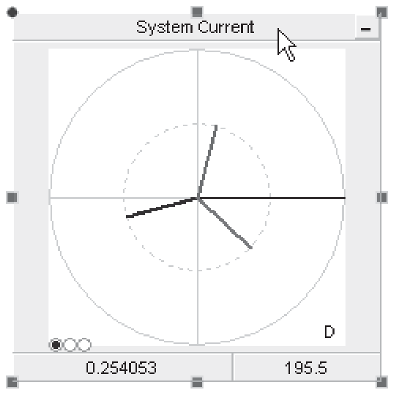
To add a phasormeter, right-click on an *Output Channel* component within the Circuit canvas and select **Graphs/Meters/Controls | Add as PhasorMeter**.



The phasormeter 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 Circuit canvas.

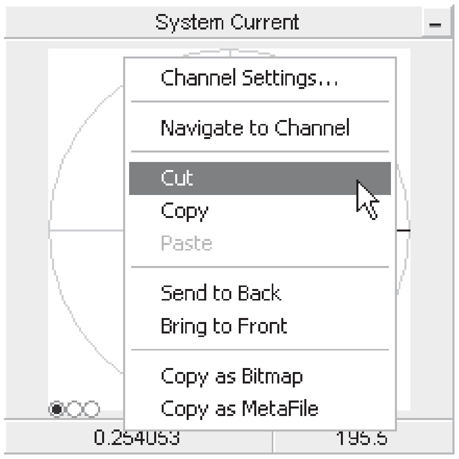
Moving and Resizing a PhasorMeter
To move a phasormeter, move the mouse pointer over the title bar and then left-click and hold. Drag the meter to where it is to be placed and release the mouse button.

To resize, 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 resize.

Cut/Copy PhasorMeter
Right-click over the meter and select **Cut** or **Copy** respectively.



Once a phasormeter 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 Circuit view and select **Paste**. A phasormeter may be pasted multiple times.

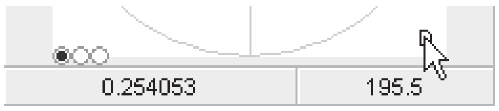
Copy PhasorMeter as Meta-File or Bitmap
The entire phasormeter display can be copied to the Windows clipboard in either meta-file (*.wmf) or bitmap (*.bmp) format. Right-click over the phasormeter title bar and select **Copy as Bitmap** or **Copy as Meta-File**. Go to your report document and paste the 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 phasormeter by right-clicking over the device title bar and selecting **Channel properties....**

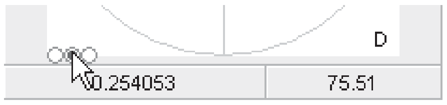
Displaying Specific Data
The magnitude and phase angle of individual array elements can be displayed in the status bar at the bottom of the phasormeter. Left-

click the corresponding button on the display selector, where the buttons represent phasors 1 through 6 from left to right.



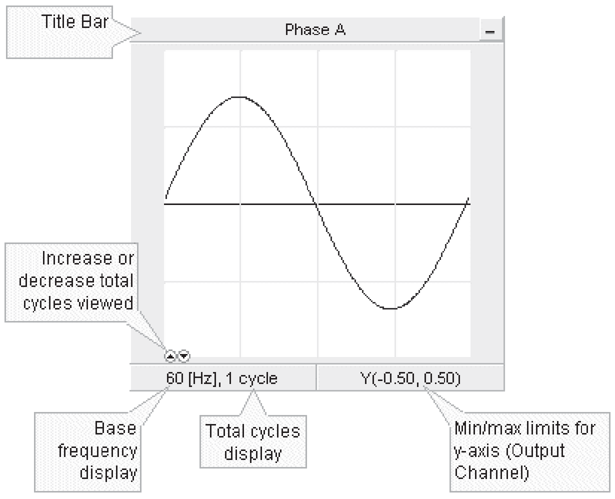
Adjusting Phase Angle Input Format
The format that the incoming phase angle data is in must be specified. This is easily accomplished by toggling the *D* or *R* display in the bottom-right corner of the graph, where:

- **D**: Degrees
- **R**: Radians



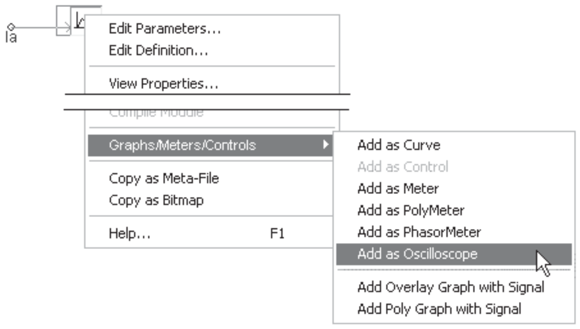
OSCILLOSCOPES

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 toolbar. 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.

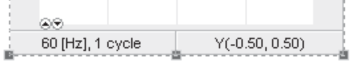
Adding an Oscilloscope
To add an oscilloscope, right-click on an *Output Channel* component within the Circuit canvas and select **Graphs/Meters/Controls | Add as Oscilloscope**.



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 Circuit canvas.

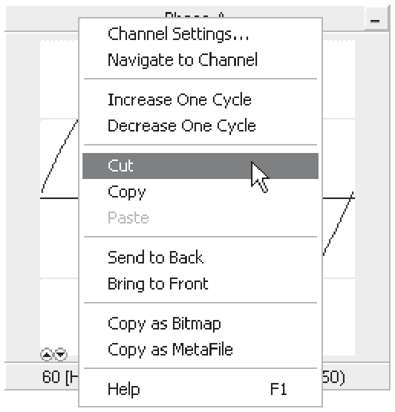
Moving and Resizing an Oscilloscope
To move an oscilloscope, move the mouse pointer over the title bar and then left-click and hold. Drag the meter to where it is to be placed and release the mouse button.

To resize, 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 resize.

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.

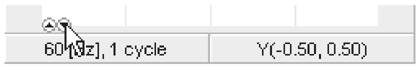
Paste Oscilloscope
Cut or copy an oscilloscope as described above. Right-click over a blank area of the project page in Circuit view and select **Paste**. A scope may be pasted multiple times.

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.

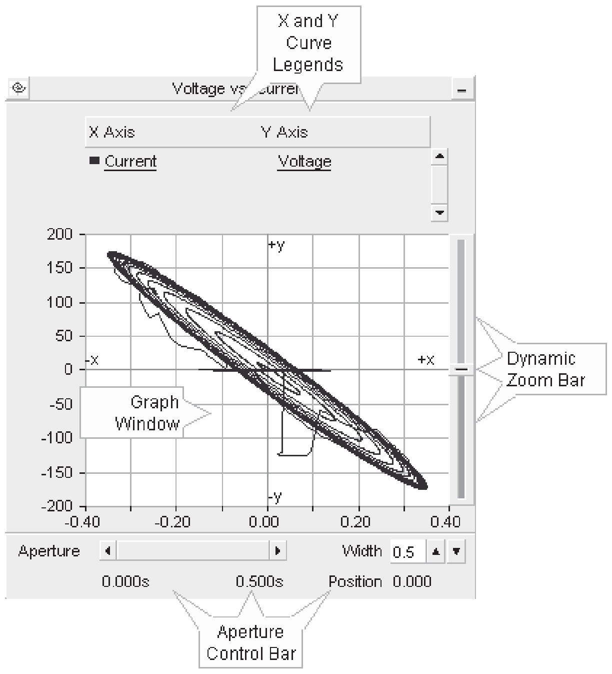
Adjusting Channel Settings
The y-axis limits of the oscilloscope are set in the corresponding *Output Channel* Properties dialog. This dialog may be accessed directly from the scope by right-clicking over the device title bar and selecting **Channel Settings....**

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

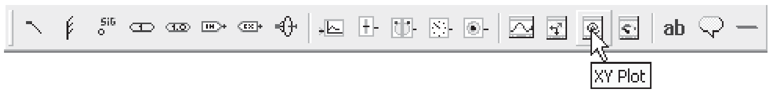
The XY Plot is comprised of both a graph frame and a single, specialized graph window for the purpose of plotting one curve versus another. An xy plot can accommodate multiple curves on each of the *x* and *y-axes*, and includes dynamic zoom and polar grid features.



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

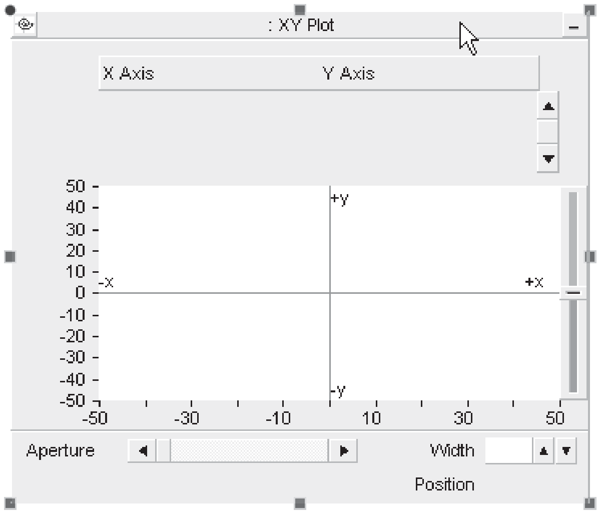
Open the project in the Circuit view. Right-click on a blank portion of the page and select **Add Component | XY Plot**, or press the **XY Plot** button in the Control Palette bar.



Moving and Resizing 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.

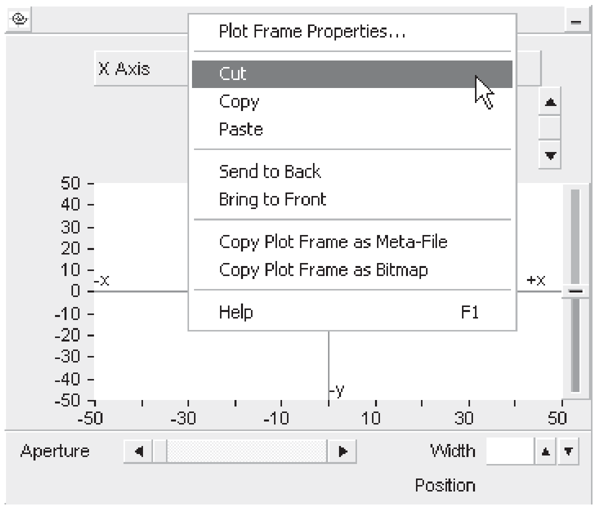
To resize the frame, move the mouse pointer over the title bar and left-click to select it. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. Left-click, hold and drag the pointer to resize.

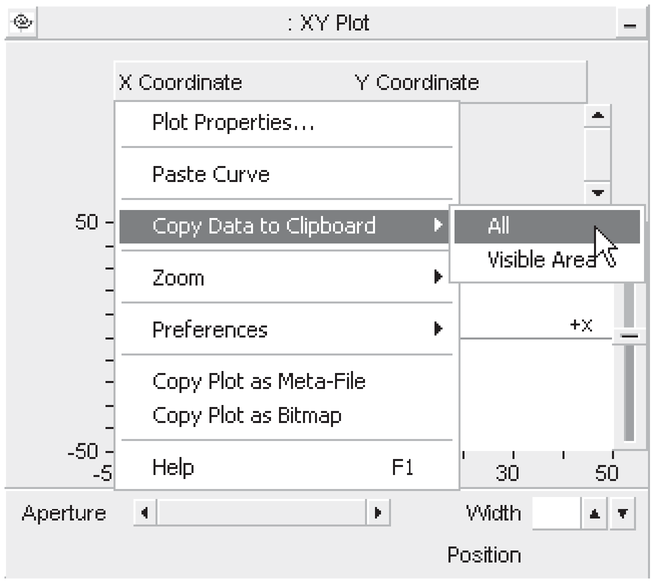
Cut/Copy XY Plot Frame

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.



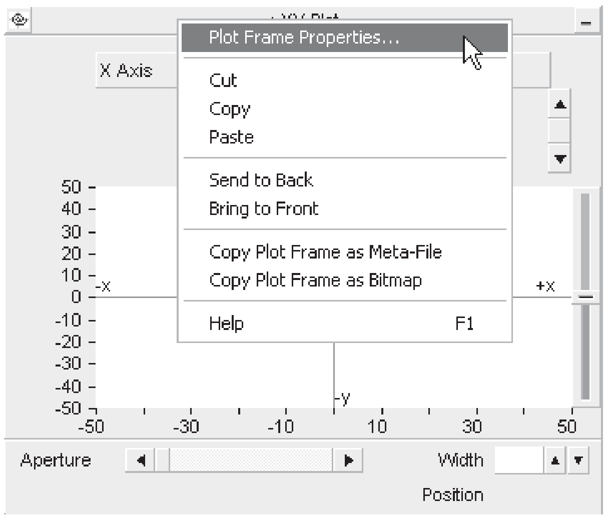
The data is copied as *Comma Separated Variables (*.csv)* format for easy migration into common data analysis software.

Two choices are given:

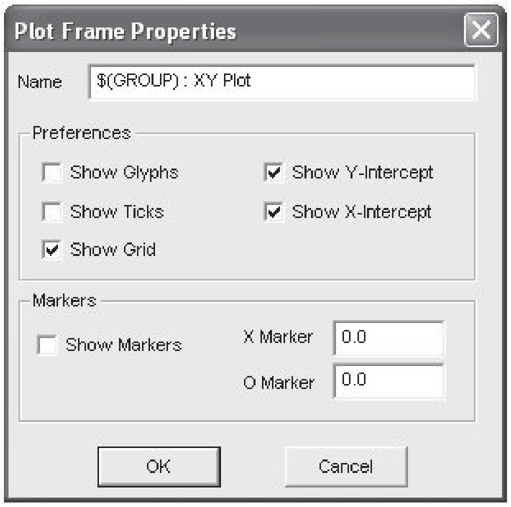
- **All**: Copies all curve data available.
- **Visible Area**: Copies all curve data visible in the plot window.

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.



Name: $(GROUP) : XY Plot

[ ] Show Glyphs
[ ] Show Ticks
[✓] Show Grid
[✓] Show Y-Intercept
[✓] Show X-Intercept

[ ] Show Markers
X Marker: 0.0
O Marker: 0.0

There are various parameters that may be edited though 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*.

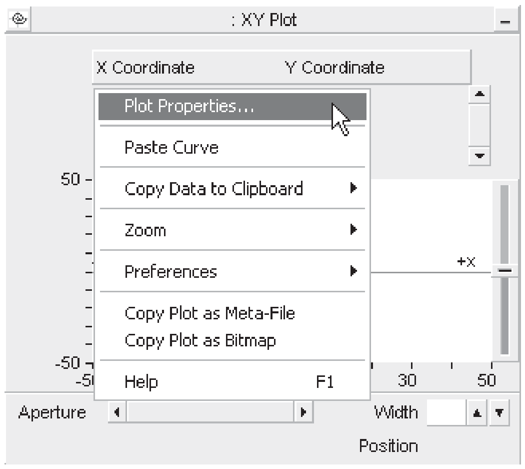
Preferences:

- **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.

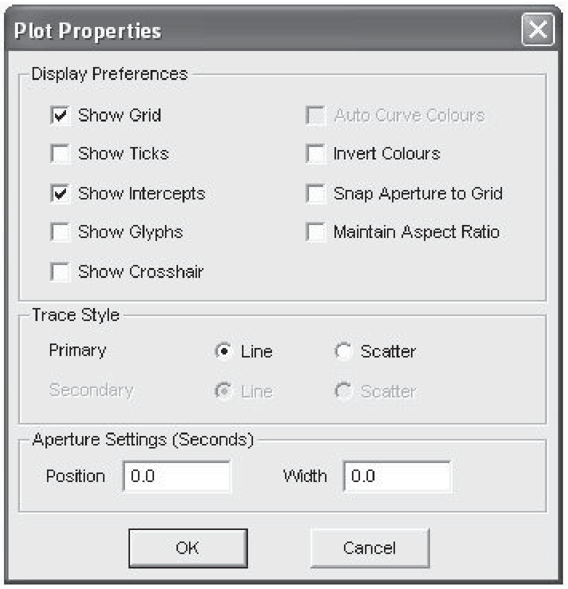
Markers:

- **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.



Plot Properties

Display Preferences

- [✓] Show Grid
- [ ] Auto Curve Colours
- [ ] Show Ticks
- [ ] Invert Colours
- [✓] Show Intercepts
- [ ] Snap Aperture to Grid
- [ ] Show Glyphs
- [ ] Maintain Aspect Ratio
- [ ] Show Crosshair

Trace Style

Primary [✓ Line] Scatter

Secondary [✓ Line] Scatter

Aperture Settings (Seconds)

Position 0.0 Width 0.0

OK Cancel

There are various parameters that may be edited though 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.

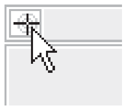
Trace Style:

- **Primary:** Select whether to draw traces as **Line** or **Scatter** view. Scatter view simply adds a single dot for each X-Y coordinate.

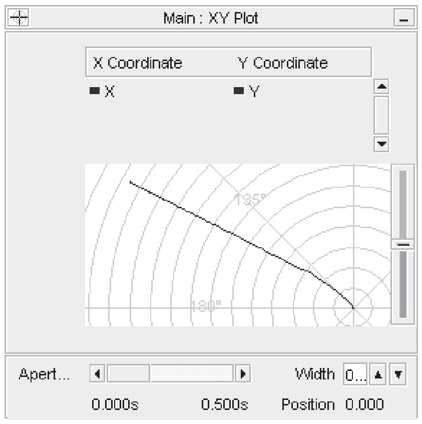
Aperture Settings (seconds):

- **Position:** Enter the starting position in seconds of the aperture window.
- **Width:** Enter the width in seconds of the aperture window.

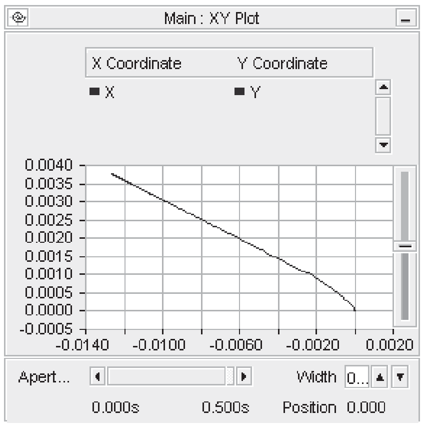
Polar Grid
The default cartesian (xy) grid display can be switched to a polar grid by simply toggling the xy/polar display button at the top-left of the plot frame.



The polar and xy grids are simply overlaid on top of the plotted data:



Polar Grid



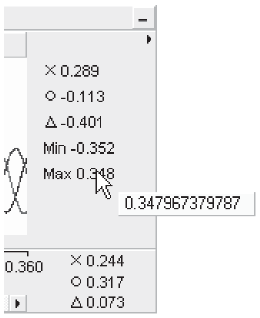
Cartesian Grid

Dynamic Zoom
See *Dynamic Zoom in XY Plot*s for more details.

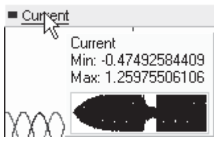
TOOL TIPS IN PLOTS

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.

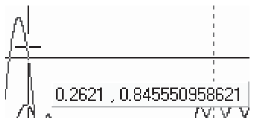
By default, the most important quantities involved in plotting are displayed directly on the graph frame. This is especially apparent when viewing marker data, which is displayed to four significant digits. For most studies, four significant digits are not enough to provide the user with accurate assessments 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.



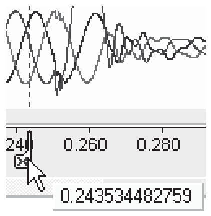
Existing tool tips, such as cross-hair and min/max curve are also standardized to 12 significant digits. The following images show the locations of other tool tips in the plotting environment:



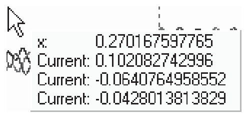
Min/Max Tool Tip



Cross-Hair Tool Tip



Marker Tool Tip

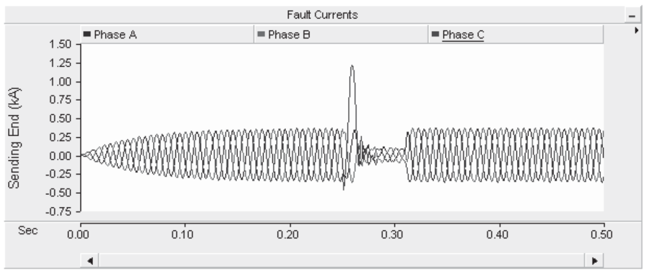


Data Tool Tip

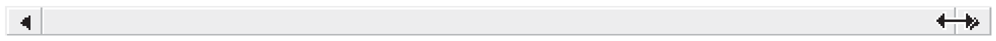
DYNAMIC APERTURE ADJUSTMENT

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.

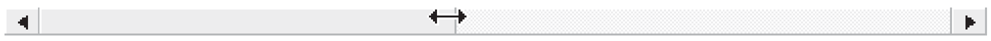
Although the following example uses a graph frame, it may be easily applied to the xy plot as well. Run the simulation so that your graph displays the curve data. The following image shows a simulation that has been run for *0.5 s*.



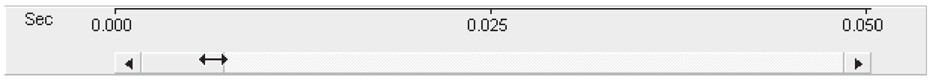
As can be seen from the graph, the fault occurs around *0.25 s* and lasts for about *0.07 s*. Dynamic aperture adjustment can be used to close the viewing window to a smaller time width, so that the fault waveforms can be more easily studied. To do this, move your mouse pointer over the horizontal scroll bar at the far right so that the mouse pointer turns into a double-headed arrow.



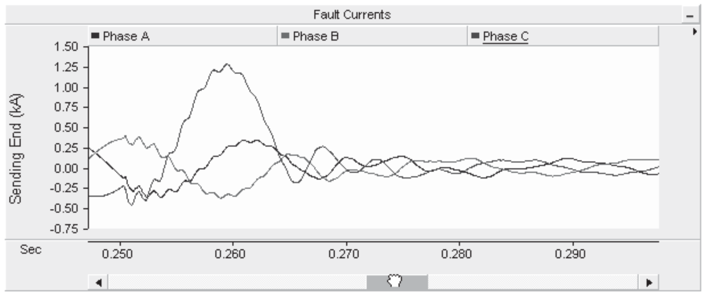
Click and hold the left mouse button and drag the scroll bar aperture slowly to the left.



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.



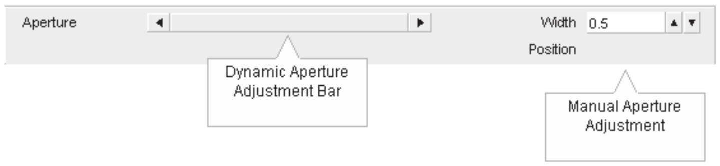
If desired, you can shrink the aperture further for even more detail. An aperture will also be created automatically when you zoom into a certain data range.

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.

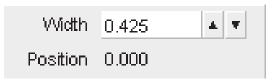
You may also use either the left and right arrow buttons at each end of the scroll bar, or the arrow keys on your keyboard to scroll through data: To scroll in small increments, simply left-click either arrow; for larger increments hold down the **Ctrl** key and then left-click. When using the arrow 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.

Adjusting and Controlling the Aperture in XY Plots

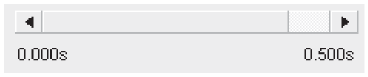
As mentioned previously, the user is able to set and control the time viewing window for the xy plot. At the bottom of the xy plot frame, there is an area devoted solely for the adjustment and control of this viewing window - otherwise known as the aperture.



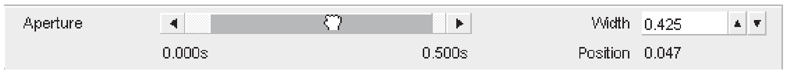
This area contains two sections; The dynamic aperture adjustment scroll bar on the left, and the manual aperture adjustment field on the right. You can increment the aperture width manually by pressing the up/down arrows to the right of the manual aperture adjustment field.



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

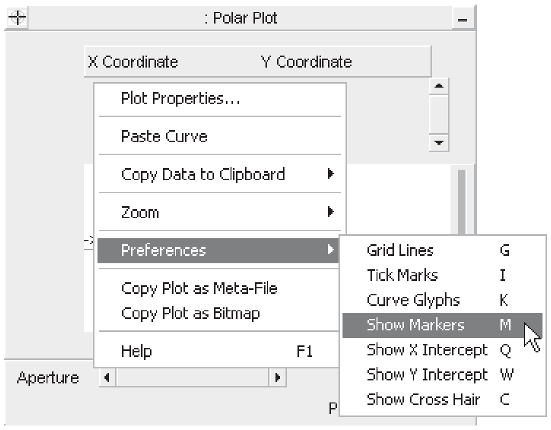
Markers are a special feature included in both graph frames and xy plot frames to help users with the analysis of their online data. Specifically, they are used to delineate the data so as to focus analysis to that specific range. Depending on marker positions, legend displays will indicate the difference 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.

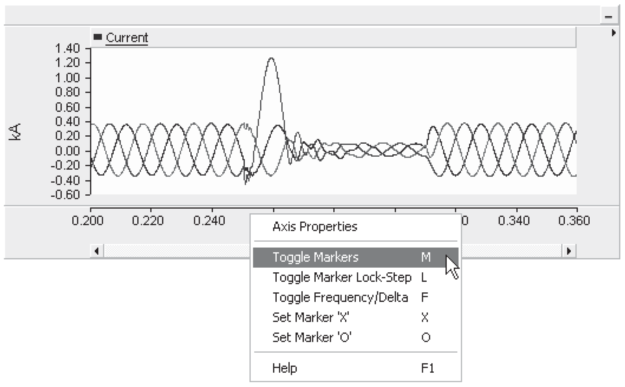
Markers are used in a slightly different manner between graph frames and xy plots. Any differences are noted in the sections below.

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**.

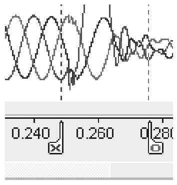


- Left double-click the graph frame horizontal axis, or right-click over the horizontal axis and select **Axis Properties...** to bring up the **Axis Properties** dialog. Select the **Show Markers** selection box.

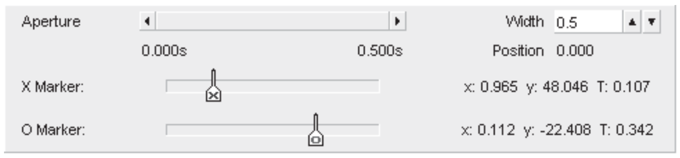


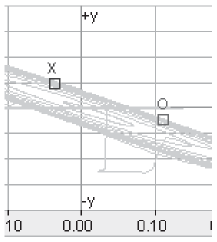
- Left double-click an overlay or poly graph, or right-click over the graph and select Graph Properties... to bring up the **Graph Properties** dialog. Select the **Show Markers** selection box.

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
0.000s
0.500s
Width 0.5
Position 0.000
X Marker: x: 0.965 y: 48.046 T: 0.107
O Marker: x: 0.112 y: -22.408 T: 0.342

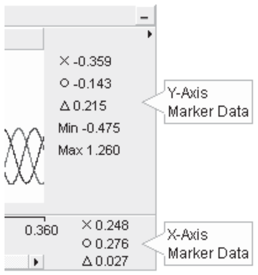


Aperture Control Bar

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.



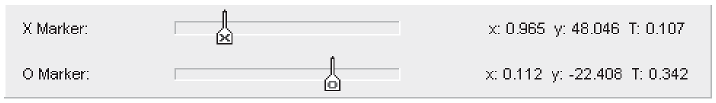
Each graph in the graph frame will have a legend directly to the right of the graph. The values displayed are specific to each graph and are described as follows:

- **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

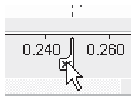
The markers will only monitor the active trace of one curve at a time. If multiple curves exist in an overlay or poly graph, you can scroll through them by simply pressing the **Space Bar** on your keyboard. You may also conveniently select the desired curve in the graph legend.

When using xy plots, you must select the desired curve in the graph legend, as the **Space Bar** function is not available.

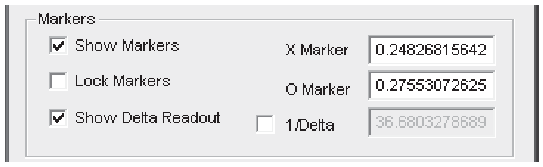
Adjusting Marker Positions

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.

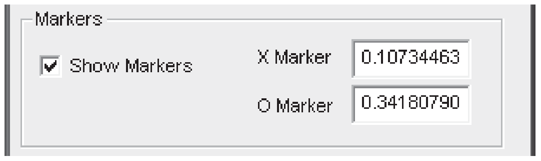


- When using graph frames, left double-click the frame horizontal axis, or right-click over the horizontal axis and select **Axis Properties...** to bring up the Axis Properties dialog. Select the **Show Markers** selection box and then set the x-axis values of each marker directly in the input fields provided. Note that it is also possible to directly set the frequency (i.e. *1/Delta*) between the two markers.



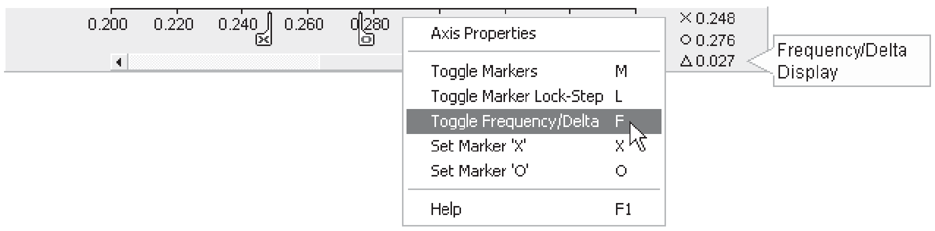
Markers
☑ Show Markers X Marker 0.24826815642
☐ Lock Markers O Marker 0.27553072625
☑ Show Delta Readout ☐ 1/Delta 36.6803278689

- 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.



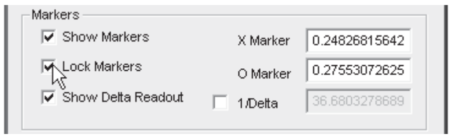
Markers
☑ Show Markers X Marker 0.10734463
O Marker 0.34180790

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):

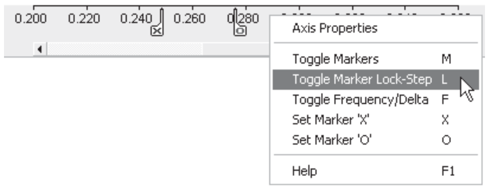


This feature is not available in xy plots.

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.


Markers
- [✓] Show Markers
- [✓] Lock Markers
- [✓] Show Delta Readout

X Marker: 0.24826815642
O Marker: 0.27553072625
[ ] 1/Delta: 36.6803278689

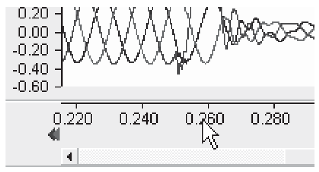


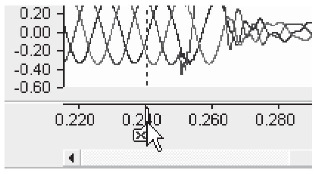
Axis Properties Dialog

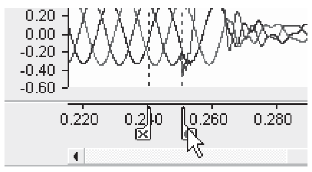
Right-Click Menu

Setting Markers
Markers can be set to a certain position on the graph frame time axis as follows:

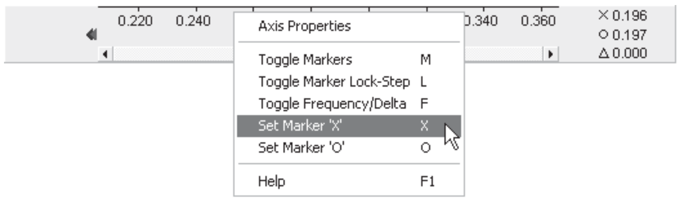
1. Show the markers by pressing the **M** key.
2. Left-click on the axis at the approximate position where you wish to place the *X* marker. Press the **X** key on your keyboard.
3. Repeat this procedure to set the *O* marker, except press the **O** key.


Press the **M** Key

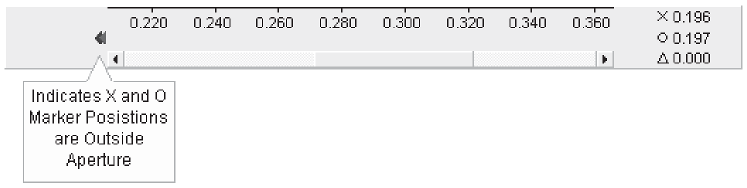

Press the **X** Key


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.



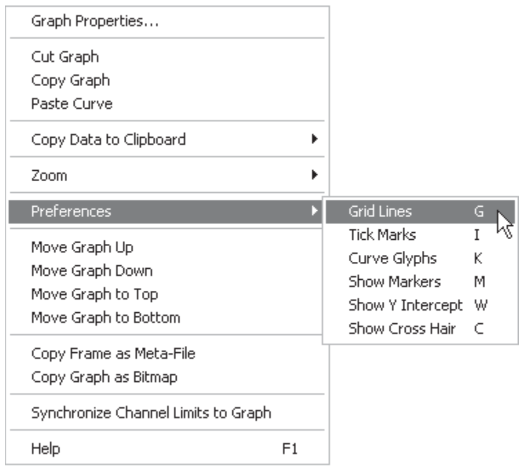
Using Markers as Bookmarks
If only a fraction of the total graph frame time scale is currently being viewed in a graph, the markers may appear outside the viewing frame when they are first enabled. If this happens, blue and red arrow symbols will appear on the time axis indicating the markers are set 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.

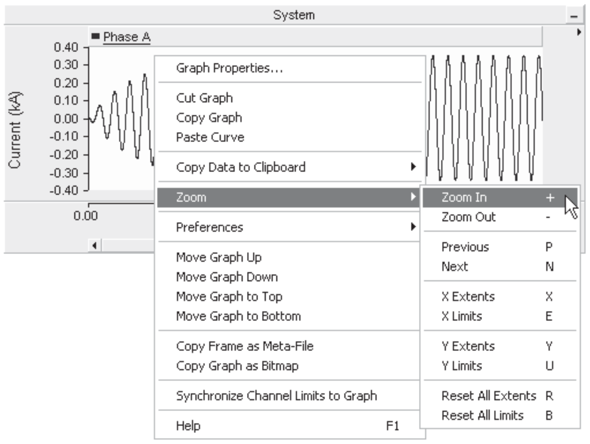
Many of the above preferences can be set within the various dialog windows involved with graphs and xy plots.

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

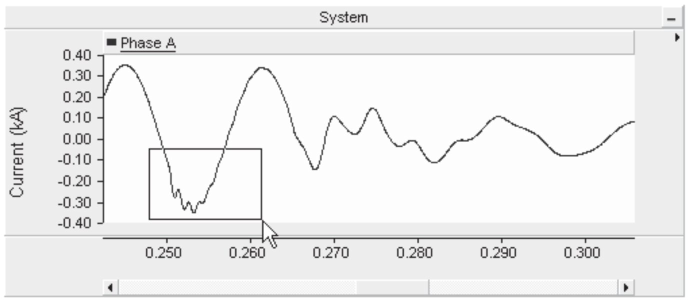
Select the desired graph with a left-click on the graph display area. Right-click over the graph to generate a pop-up menu and select **Zoom | Zoom In** or **Zoom | Zoom Out**.



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

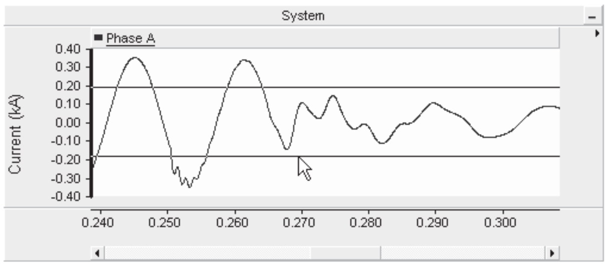
Select the desired graph with a left-click on the graph display area. Click and hold the left mouse button and drag the mouse pointer to create a boxed region. Release the left mouse button to zoom to that region.



You can also zoom along a single axis by drawing out a narrow box. The zoom will ignore the narrow side of the box.

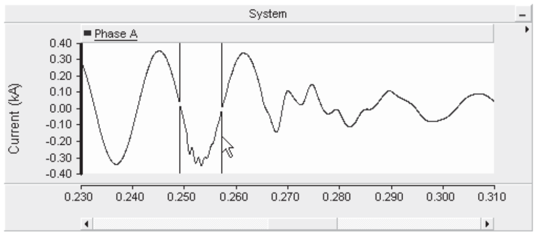
Vertical Zoom

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.



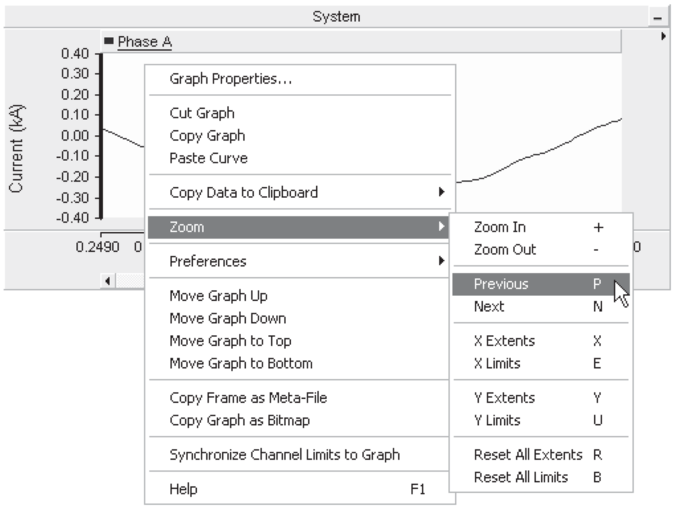
Horizontal Zoom

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**.

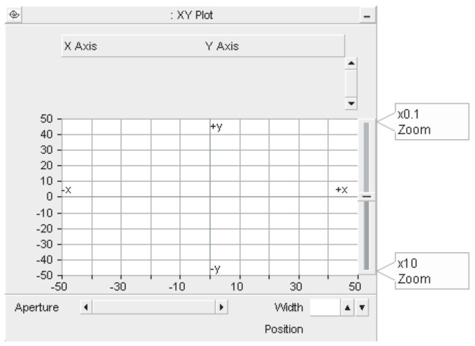
As an alternative, you can also use the **E** or **U** keyboard shortcuts for *Zoom X Limits* or *Zoom Y Limits* respectively (once the desired graph has been selected).

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.

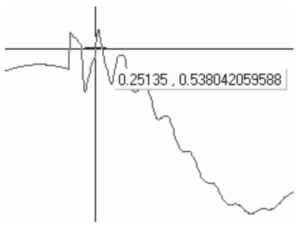
CROSS HAIR MODE

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.

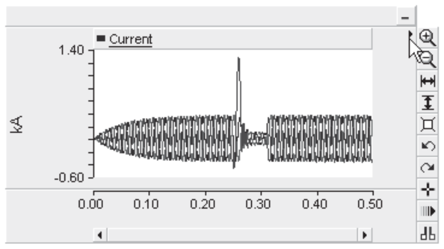


As shown above, the curve xy data will be displayed beside the cross hair. When your left mouse button is released, the cross hair will disappear, but cross hair mode will remain invoked until you press the **C** key again (or select **Preferences | Show Cross Hair**).

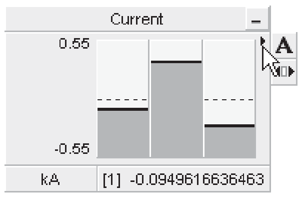
In xy plots, the cross hair will follow the mouse pointer only.

POP-UP TOOL BARS

Pop-up toolbars are provided with graphs and polymeters in order to provide a quick and easy means for accessing the most common functions. Pop-up toolbars may be invoked by left-clicking on the small arrow in the top-right corner of the graph or polymeter windows:



Graph



PolyMeter

The pop-up toolbar 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:

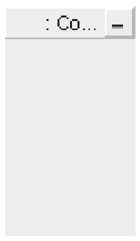
Button	Description
A	Toggle index labels
◂▯▸	Toggle scroll view mode

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.

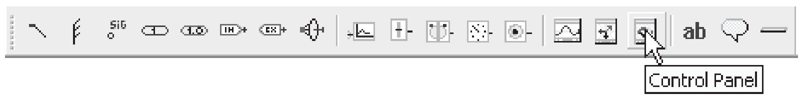
Control Panels

A Control Panel is a special component used for accommodating control or meter interfaces and can be placed anywhere on a Circuit 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

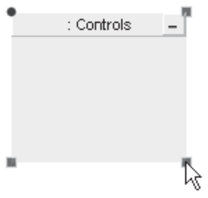
Open the project in the Circuit view. Right-click on a blank portion of the page and select **Add Component | Control Panel**, or press the **Control Panel** button in the Control Palette bar.



Moving and Resizing 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.

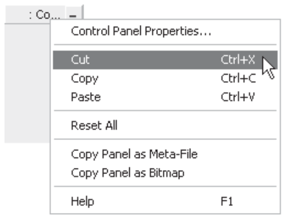
To resize, move the mouse pointer over the title bar and left-click to select the panel. Grips should then appear around the outer edge as shown below.



Move the mouse pointer over one of the grips. Left-click, hold and then drag, then move the pointer to resize.

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.



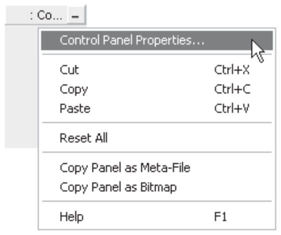
Once a control panel has been cut or copied, it may then be pasted into another location in the project (along with its contents).

Paste Panel

Cut or copy a control panel as described above. Right-click over a blank area of the Circuit canvas and select **Paste**. A control panel may be pasted multiple times.

Adjusting Panel properties

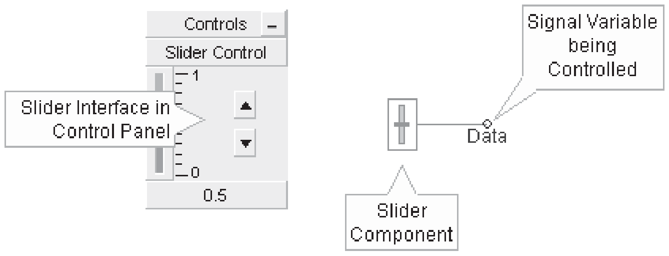
To access the Control Panel Properties dialog, left double-click the control panel title bar, or right-click over the title bar and select **Control Panel Properties....**



Presently, the only adjustable panel property is the caption:

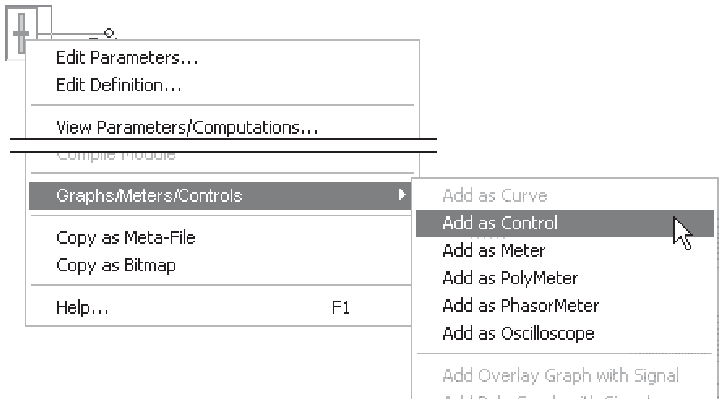
- **Caption**: Enter a title for the control panel (this text will appear in the panel title bar). The default text may appear a bit cryptic: This syntax is used as a naming convention for grouping objects in the workspace. For more information on this syntax, see *Grouping of Runtime Objects*.

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.

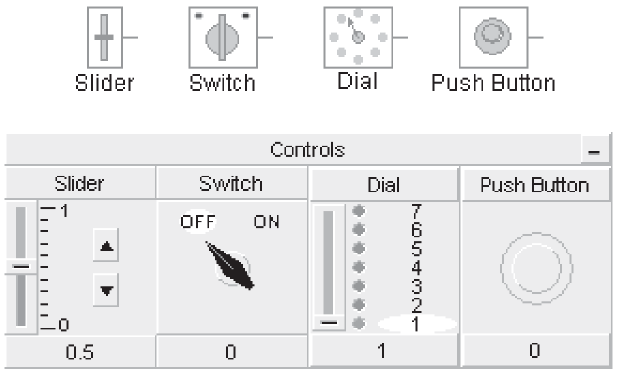

Controls
Slider Control
1
0
0.5
Slider Interface in Control Panel
Signal Variable being Controlled
Data
Slider Component

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**.



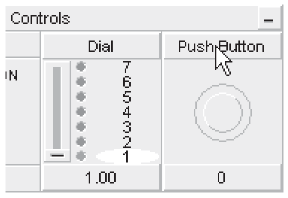
Of course, each type of control component will have a different control interface when added to a control panel. The following figure shows the available control components and their corresponding control interfaces:

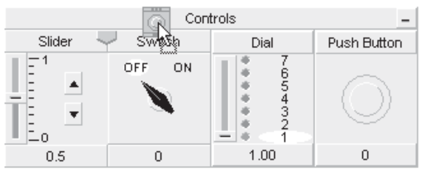


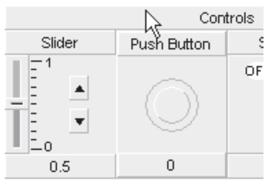
Control Interface Order

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.

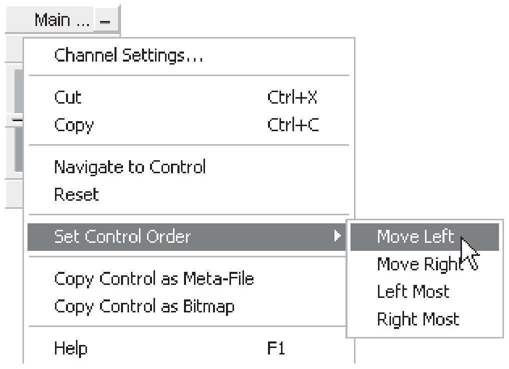






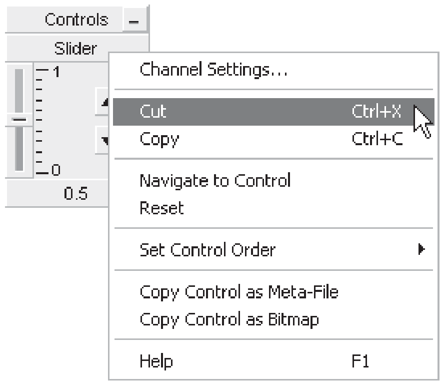
Left-Click and Hold   Drag   Release Button

- 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.



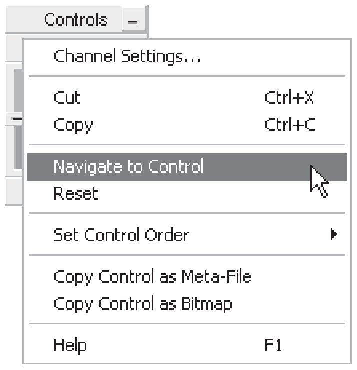
Once a control interface has been cut or copied, it may then be pasted into control panel in the project.

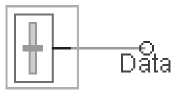
Paste Control Interface

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.

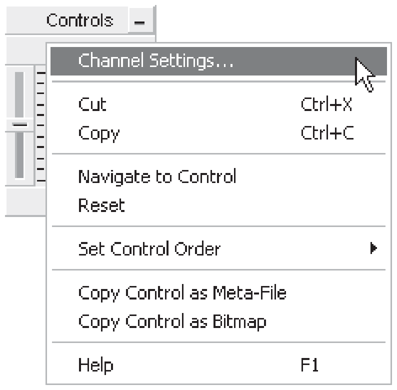




Select Navigate to Control PSCAD Highlights Object

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....**



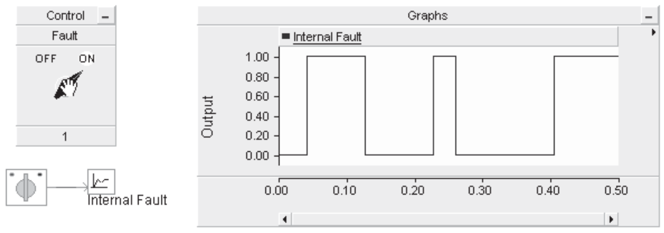
Using the Control Interfaces

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.

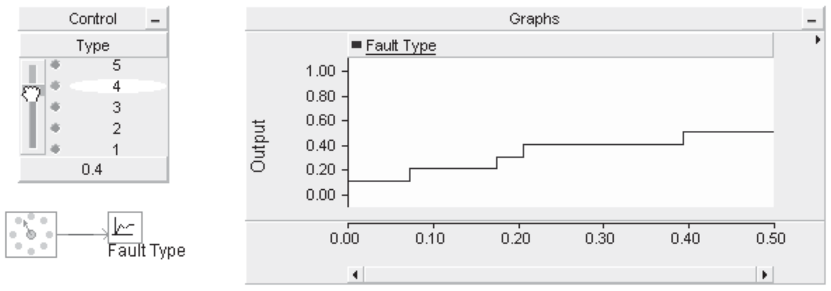
The image below shows a plot of a *Switch* component output, where the user has changed its state (i.e. turned it on and off) during a *1.0* second simulation run.



Rotary Switch Component:

To operate the *Rotary Switch* component, left-click and hold the slider control knob and move the knob up or down. As the control knob is moved, the *Rotary Switch* interface will indicate graphically, which of the multiple states is currently being output from the corresponding *Rotary Switch* component.

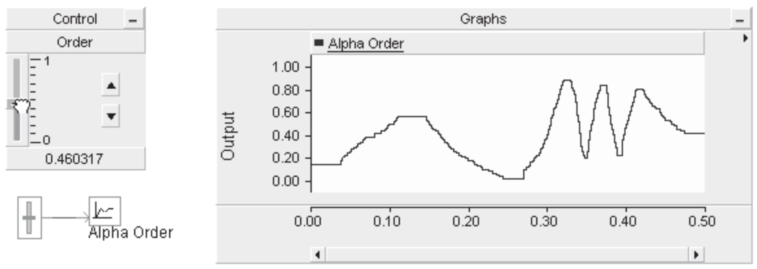
The image below shows a plot of a *Rotary Switch* component output, where the user has changed its state during a *1.0* second simulation run.



Slider Component:

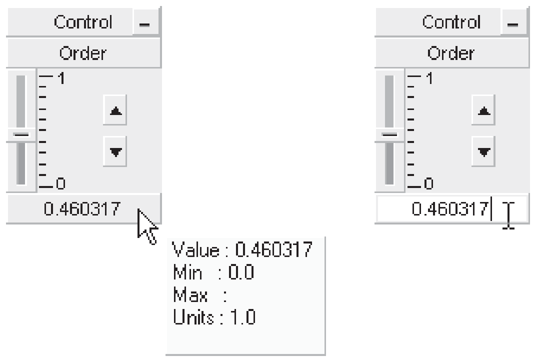
The *Slider* component is operated in the same manner as the *Rotary Switch* interface described above. The only difference is that the *Slider* output does not operate in discrete states, but provides a continuously variable output.

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.



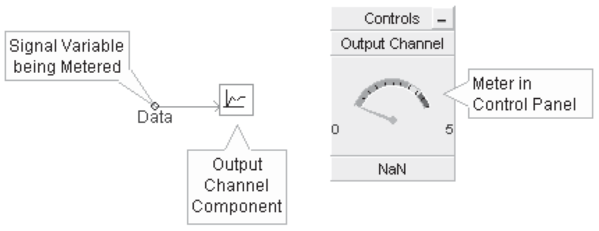
It is important to note that the *Slider* component will output exactly what is shown on the *Slider* interface. The *Slider* interface has a maximum precision of six significant digits. If more precision is required, you can add successive Slider outputs together, each handling different data ranges. 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 six 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.



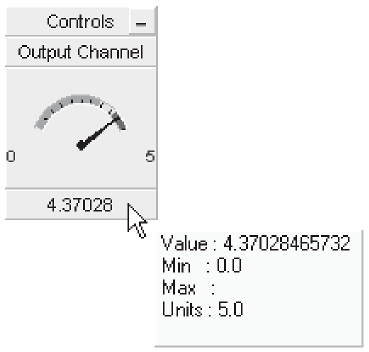
Meters

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 six significant digits. However, twelve significant digits can be viewed via the tool tip.

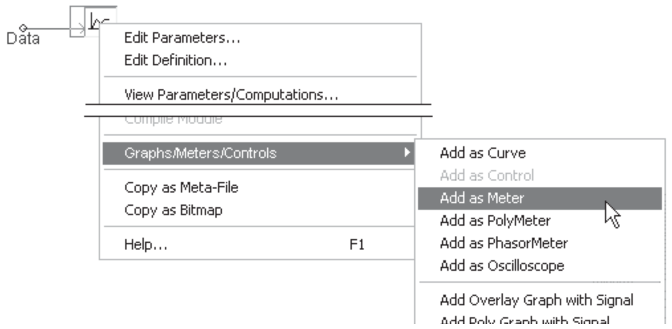


Value : 4.37028465732
Min : 0.0
Max :
Units : 5.0

Adding a Meter to a Control Panel

There are a couple of methods to accomplish this:

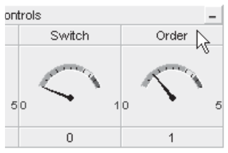
- **Drag and Drop**: Hold down the **Ctrl** key, left-click and hold the associated *Output Channel* component and drag and release over the control panel. See *Drag and Drop* for more details.
- Right-click on the associated *Output Channel* component and select **Graphs/Meters/Controls | Add as Meter**. Right-click on the desired control panel title bar and select **Paste**.



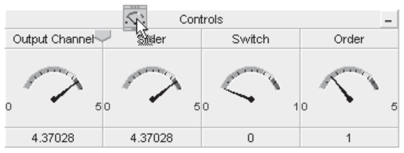
Meter Order

Once multiple meter interfaces have been added to a particular control panel, you may change the order in which they appear.

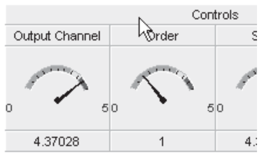
- **Drag and Drop**: Press **Ctrl** and then **left-click and hold** over the title bar of a particular meter interface and then drag the mouse pointer over the control panel title bar. A downward arrow will appear between existing interfaces depending on the position of the mouse pointer. Release the left button to drop the interface.



Left-Click and Hold

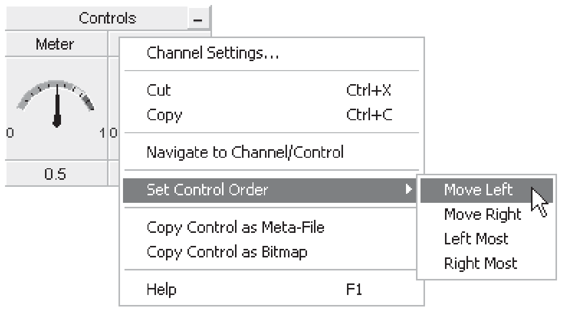


Drag



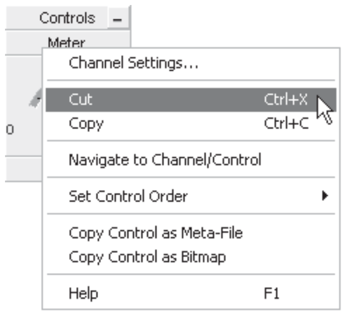
Release Button

- Right-click over the meter interface and select **Set Control Order**. Choose one of the options given.



Cut/Copy Meter

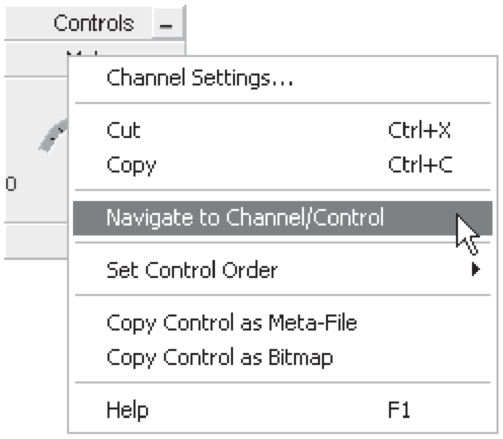
Right-click over the meter title bar and select **Cut** or **Copy** respectively.



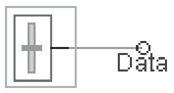
Once a meter has been cut or copied, it may then be pasted into any control panel in the project.

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 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



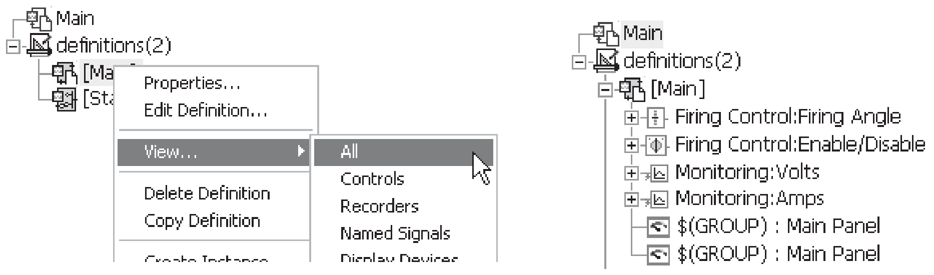
PSCAD Highlights Object

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 secondary workspace window.

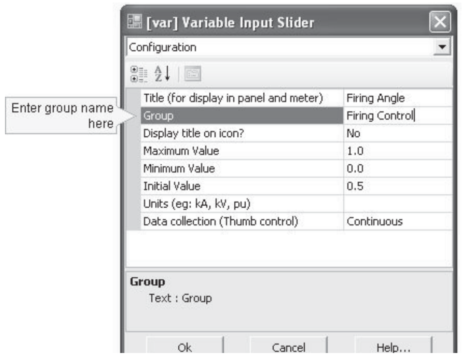
See *Runtime Objects in the Definitions Branch* in chapter 4 of this manual for more details on accessing runtime objects.



Runtime Objects Listed According to Group Name in the Definitions Branch

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.

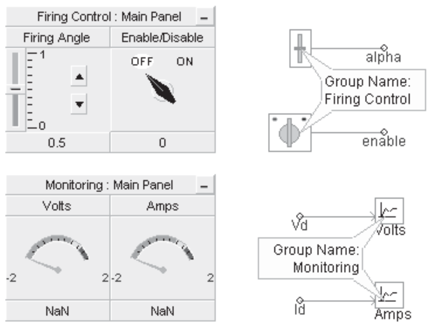

[var] Variable Input Slider

Configuration

Title (for display in panel and meter)	Firing Angle
Group	Firing Control
Display title on icon?	No
Maximum Value	1.0
Minimum Value	0.0
Initial Value	0.5
Units (eg: kA, kV, pu)	
Data collection (Thumb control)	Continuous

Group
Text : Group

For example, consider the following project circuit 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*.

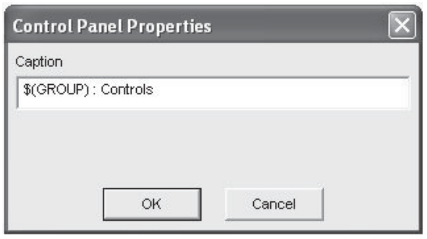
Displaying Group Name on Graph Frames and Control Panels
The group name of any control or meter interface, or any curve in a graph, can be automatically displayed through the use of a special syntax:

```
$(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.

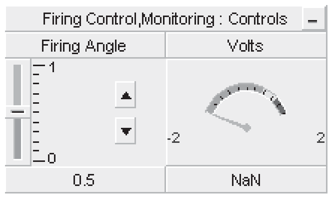
EXAMPLE 6-1:

A user wants to display the group name of all Interfaces within a control panel. The default *Caption* parameter in the control panel properties 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 Properties

Caption: $(GROUP) : Controls

Control Panel Properties

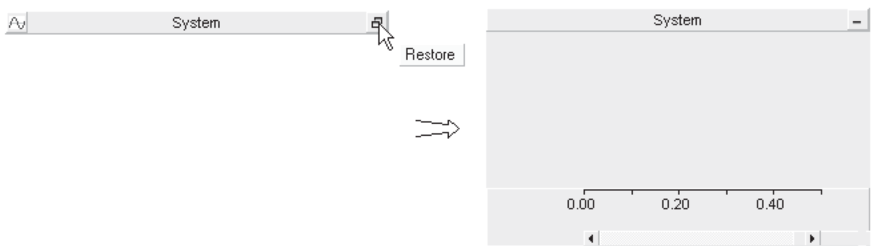


Control Panel with Objects from Two Different Groups

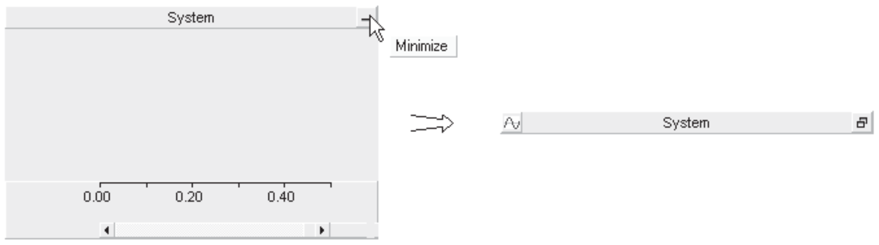
FRAME/PANEL MINIMIZATION

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. To access the *Workspace Options* dialog, go to **Edit | Workspace Options...** in the main menu bar. See *Environment* for more details.

Copy as Meta-File or Bitmap

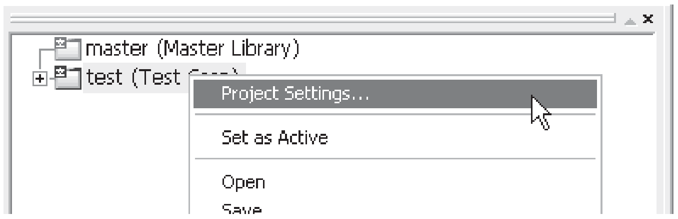
Right-click over the plotting or control object and select **Copy <object> as Meta-File** or **Copy <object> as Bitmap** from the pop-up menu. Depending on the object selected, the <object> part of the command will be replaced by the object type.

You may then go directly to a report document and paste the image, while it is still in the Windows clipboard.

Chapter 7: Project Settings

Most 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 features and processes.

This dialog can be accessed through a right-click on the project title itself (either a case or a library) in the workspace window, and selecting *Project Settings*... from the pop-up menu.



The features and parameters available in the Project Settings dialog control window are divided into several specific areas:

- **General**: Settings such as revision tracking and other file related information.
- **Runtime**: Project related simulation controls.
- **Simulation**: EMTDC related simulation controls.
- **Dynamics**: PSCAD signal control and other features.
- **Mapping**: PSCAD electric network related features.
- **Fortran**: Fortran compiler and debug settings.
- **Link**: Fortran and MATLAB linking options.

There are a number of more advanced features available for the enhancement of speed, accuracy and efficiency of simulations performed in PSCAD. The Project Settings dialog gives the user access and control of most of these features. Most users need not concern themselves with the more advanced settings, and should leave most to their default. However, there may be some instances where users may wish to disable or enable some processes at their discretion.

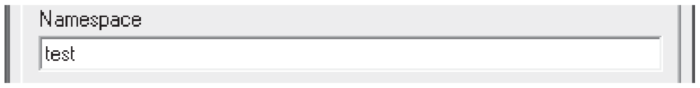
The Project Settings window for a library project contains only the General tab, as simulations cannot be performed on library projects.

GENERAL

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

Namespace

The namespace is the project attribute through which all component definitions are linked to a project. As a project attribute, it can only be changed from this project setting field.

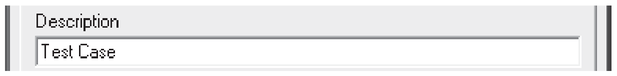


Namespace
test

Using a project attribute to ensure proper linking to component definitions ensures that these links are not dependent on the actual file name. See *Linking and Re-linking Definitions* in Chapter 5 of this manual for more details.

Description

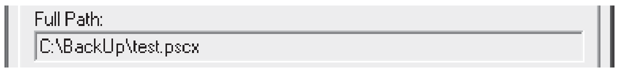
This field allows for the entry of a single-line description of the project. This description will be displayed beside the project filename in the workspace window. Do not use quotations (“) or apostrophes (‘) in this field.



Description
Test Case

Full Path

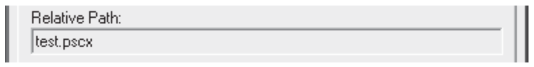
Full Path displays the path and filename information for the project file. This field is for display only and cannot be changed through the Project Settings dialog.



Full Path:
C:\BackUp\test.pscx

Relative Path

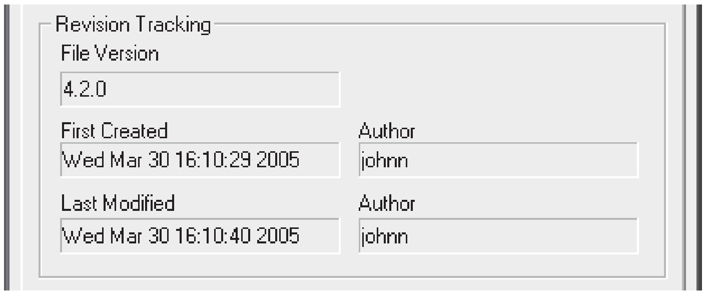
Relative path displays the project filename only. This field is for display only and cannot be changed through the Project Settings dialog.



Relative Path:
test.pscx

Revision Tracking

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



Revision Tracking

File Version: 4.2.0

First Created: Wed Mar 30 16:10:29 2005 — Author: johnn

Last Modified: Wed Mar 30 16:10:40 2005 — Author: johnn

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 was last modified.

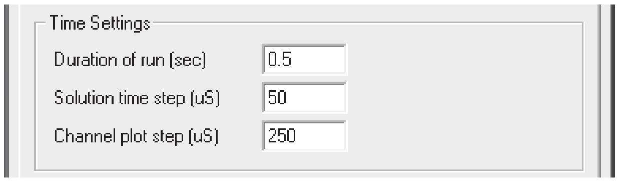
Author
The name (userid) of the person who created and/or modified the project.

RUNTIME

The properties contained within the Runtime section are the most commonly accessed project parameters.

Time Settings

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



Time Settings

Duration of run (sec): 0.5

Solution time step (uS): 50

Channel plot step (uS): 250

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 (μs)

This is the EMTDC simulation time step, entered in microseconds. The default is *50* μ*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 (μs)

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* μ*s* plot step provides a reasonable resolution and speed.

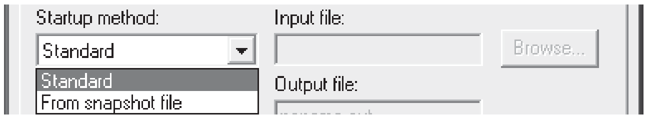
Smaller sampling intervals (higher sampling rate) can decrease the simulation speed considerably due to an excessive transfer of data from EMTDC to PSCAD (without adding much to the plot resolution). Users can experiment with this number for a given project. If the sampling interval is too large, the 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.

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 from the Runtime bar.

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.

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.

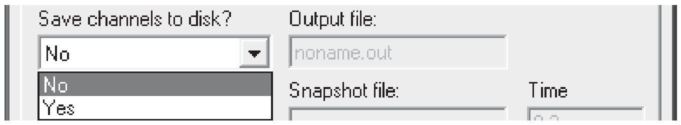
When you start from a Snapshot File, make sure that you have not altered the circuit from which the snapshot was taken, otherwise an error may occur.

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* in Chapter 5 of this manual for more details on how to create and start from a snapshot file.

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 project temporary directory (*.emt).

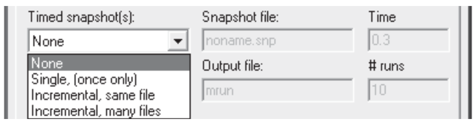


Save channels to disk?: No
Output file: noname.out

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.



Timed snapshot(s): None
Snapshot file: noname.snp
Time: 0.3
Output file: mrun
# runs: 10

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 overwritten 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.

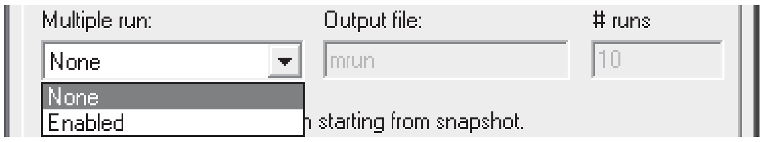
The file naming convention is:

```
base_name_##.snp
```

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

Multiple Run
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.

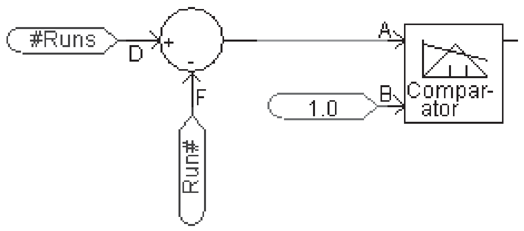
The other multiple run method involves the *Multiple Run* component, which is available in the master library. DO NOT enable this multiple run feature when using the *Multiple Run* component.


Multiple run: None
None
Enabled
Output file: mrun
# runs 10
n starting from snapshot.

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).

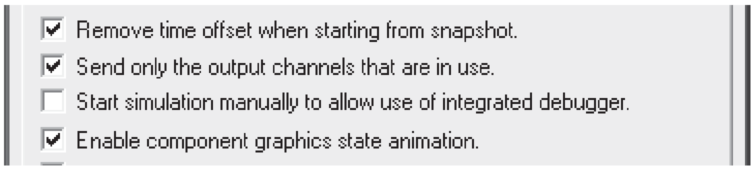
EXAMPLE 7-1:

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 snapshot.
- [✓] Send only the output channels that are in use.
- [ ] Start simulation manually to allow use of integrated debugger.
- [✓] Enable component graphics state animation.

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).

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

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.

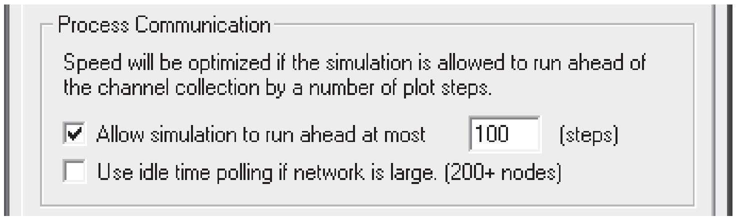
Start Simulation Manually to Allow Use of an Integrated Debugger
This option allows you to start a PSCAD run that can be connected with a manually started EMTDC case. See *Deployment of an Integrated Debugger* in Chapter 11 of this manual for more details.

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.

Process Communication
The input parameters involved here are outlined below.


Process Communication
Speed will be optimized if the simulation is allowed to run ahead of the channel collection by a number of plot steps.
[✓] Allow simulation to run ahead at most 100 (steps)
[ ] Use idle time polling if network is large. (200+ nodes)

Allow Simulation to Run Ahead at Most
This parameter controls the maximum amount of time (in time step increments), by which EMTDC is allowed to run ahead of the PSCAD display. Adjusting this parameter can have the following implications:

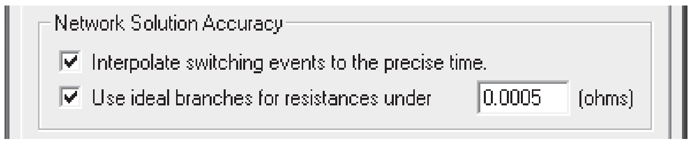
1. If the run-ahead time is too small, PSCAD will hold EMTDC back and the overall simulation speed will be reduced.
2. If the run-ahead time is too large, interactive control actions (such as resetting a slider value or pushing a button) are

potentially conveyed to EMTDC with a maximum delay equal to this margin.

Run-ahead time is something users can optimize based on the case and has no implication on the accuracy of simulation. The default value of 100 steps is a good compromise.

Use Idle Time Polling if Network is Large (200+ Nodes)
To ensure a fast response in the data communication between PSCAD and EMTDC, the simulation of smaller systems is continuously polled, providing maximum performance. On larger networks however, the simulation can run significantly slower and it is not necessary to poll the simulation continuously. To free up processor cycles (and to allow other applications to run more smoothly), this control allows the polling to only occur during periods when the application is not busy, referred to as 'idle time.' If this option is set, larger networks will use this method instead, reducing the processor loading from PSCAD.

Network Solution Accuracy
The following input parameters can affect solution speed.


Network Solution Accuracy
[✓] Interpolate switching events to the precise time.
[✓] Use ideal branches for resistances under 0.0005 (ohms)

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 behaviour of EMTDC and is essential for the accurate simulation of frequently switching devices, such as FACTS models.

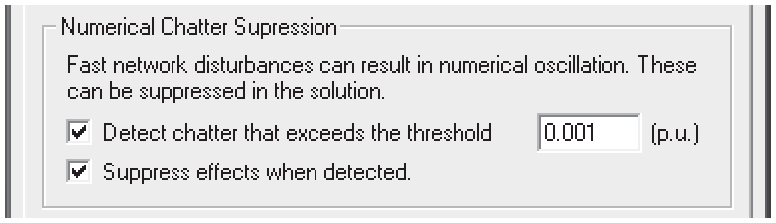
Use Ideal Branches for Resistances Under
The Ideal Branch algorithm allows for zero resistances and true infinite bus voltage sources in EMTDC.

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* Ω for the source resistance. Similarly, for a zero resistance branch, enter a value less than the threshold or *0.0* Ω for the ON resistance of a diode, close resistance of a breaker, etc.

The default threshold value is preset to *0.0005* Ω. As this algorithm involves extra computations, 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.



Numerical Chatter Supression

Fast network disturbances can result in numerical oscillation. These can be suppressed in the solution.

[✓] Detect chatter that exceeds the threshold 0.001 (p.u.)

[✓] Suppress effects when detected.

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.

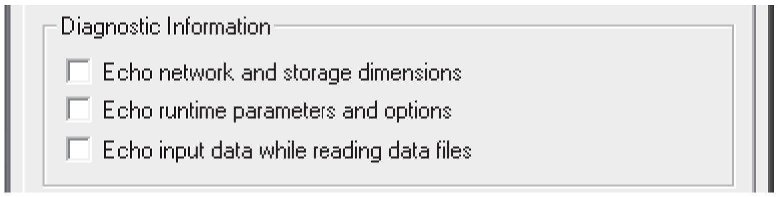
An input field is included directly beside this option. Enter a threshold value for chatter detection, below which chatter will be ignored (the default is *0.001 pu*).

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 tab window.



Diagnostic Information

[ ] Echo network and storage dimensions

[ ] Echo runtime parameters and options

[ ] Echo input data while reading data files

Echo Network and Storage Dimensions

Prints network and storage array dimensions.

Echo Runtime Parameters and Options
Prints runtime related options.

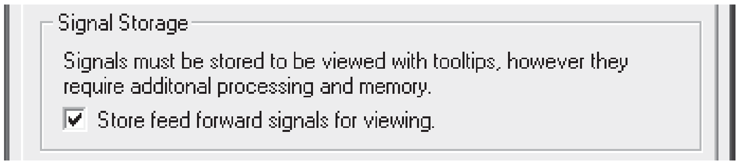
Echo Input Data While Reading Data Files
Prints all data read from the data and map files.

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.


Signal Storage
Signals must be stored to be viewed with tooltips, however they require additonal processing and memory.
☑ Store feed forward signals for viewing.

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 <u>must</u> 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.

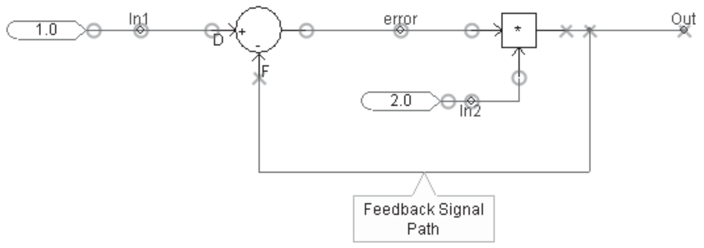
It is important to note that tool tips (flybys) extract the value of the signal being monitored from storage. Therefore, if this control is disabled, <u>all feed-forward signal values will not appear in the tool tips</u>!

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 O symbol), and one is a feedback variable (indicated by the X symbol).

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:

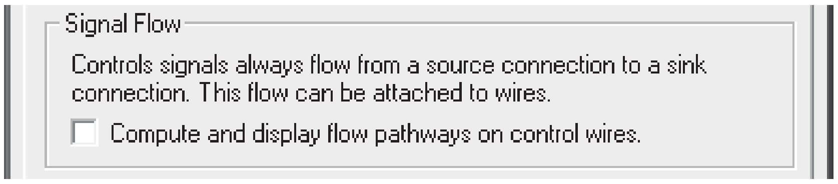
```
!==================================================
      SUBROUTINE DSDyn()
.
.
.
!----------------------------------------
! Transfers from storage arrays
!----------------------------------------
      In2       = STOF(ISTOF + 1)
      In1       = STOF(ISTOF + 2)
      error     = STOF(ISTOF + 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.


Signal Flow

Controls signals always flow from a source connection to a sink connection. This flow can be attached to wires.

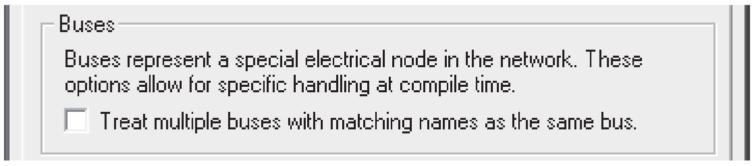
☐ Compute and display flow pathways on control wires.

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 Flow Indicators* in Chapter 11 of this manual for more details on this.

Buses

The input parameters involved here are outlined below.


Buses

Buses represent a special electrical node in the network. These options allow for specific handling at compile time.

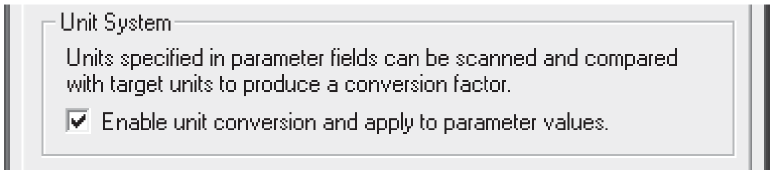
☐ Treat multiple buses with matching names as the same bus.

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:


Unit System

Units specified in parameter fields can be scanned and compared with target units to produce a conversion factor.

✓ Enable unit conversion and apply to parameter values.

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:

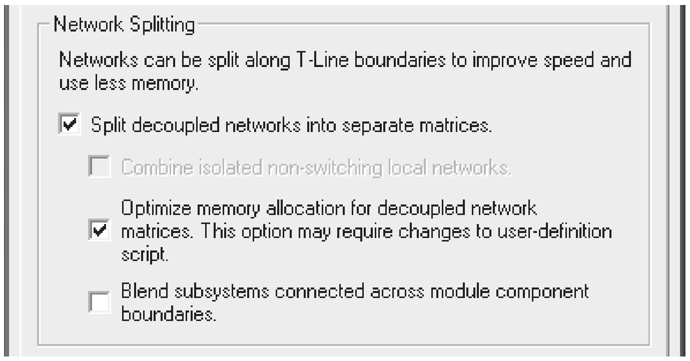
1. Consult the output window for error messages: Point to the error source (see *Locating the Problem Source*).
2. View the parameters of the component.
3. Scan through the *Value* column and look for a *#NaN* (Not a Number) symbol. This usually indicates that the entered unit does not match the *Target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


Network Splitting

Networks can be split along T-Line boundaries to improve speed and use less memory.

☑ Split decoupled networks into separate matrices.

☐ Combine isolated non-switching local networks.

☑ Optimize memory allocation for decoupled network matrices. This option may require changes to user-definition script.

☐ Blend subsystems connected across module component boundaries.

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 Chapter 3 of the EMTDC manual for more details.

Combine Isolated Non-Switching Local Networks

This feature is available only if **Split Decoupled Networks into Separate Matrices** is enabled. When enabled, all subsystems that do not contain switching devices are lumped together. This process can substantially lower the memory usage by EMTDC when storing electric network data.

This setting is obsolete and remains for compatibility reasons only. Instead of this option, you should choose *Optimize Memory Allocation for Decoupled Network Matrices*, which is much more effective.

Optimize Memory Allocation for Decoupled Network Matrices

This setting should remain enabled always. This algorithm dynamically dimensions subsystems in the EMTDC conductance matrix and manages memory allocation efficiently.

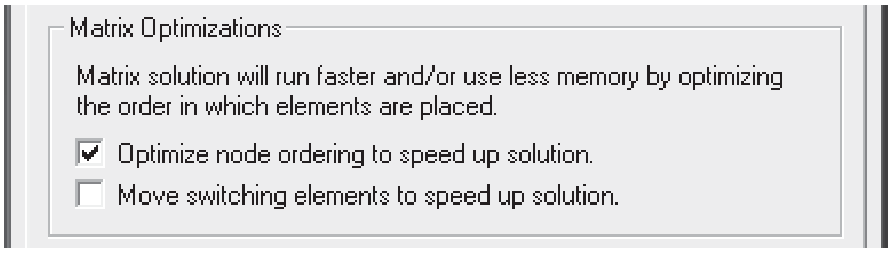
Blend Subsystems Connected Across Module Component Boundaries

This setting enables the older subsystem splitting algorithm that was used in PSCAD versions previous to X4. When enabled, this algorithm will merge (or blend) all subsystems that span module component boundaries. In other words, if two electrical subsystems are connected through a module component, they will be merged into a single subsystem.

This setting is obsolete and remains for compatibility reasons only. This option should remain disabled.

Matrix Optimizations

The input parameters involved here are outlined below.



Matrix Optimizations

Matrix solution will run faster and/or use less memory by optimizing the order in which elements are placed.

- [✓] Optimize node ordering to speed up solution.
- [ ] Move switching elements to speed up solution.

Optimize Node Ordering to Speed Up Solution

Optimize Node Ordering is a PSCAD based algorithm, which renumbers mapped nodes for 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 references [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.).

Electrical Connections in user-defined components are set as node type 2 by changing the electrical Connection 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].

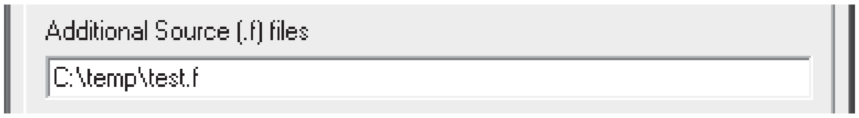
FORTRAN

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.

Additional Source (.f) Files

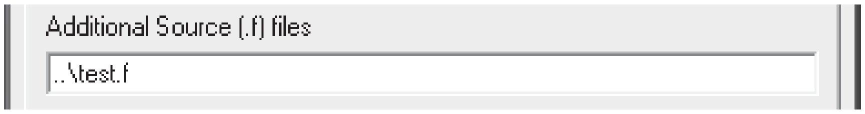
This input field allows you to link one or more external source code files, so as to be included during compilation of the associated project. An external source code file could contain one or more subroutines, which must be called from within the FORTRAN segment of the component that uses it.

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:


Additional Source (.f) files

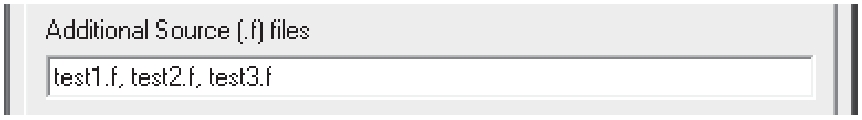
C:\temp\test.f

If only the filename is entered, PSCAD will assume that the source file is <u>located in the same directory as the project file itself</u>, 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:


Additional Source (.f) files

..\test.f

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

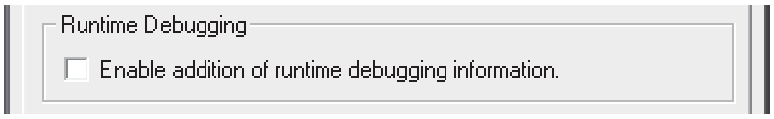

Additional Source (.f) files

test1.f, test2.f, test3.f

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

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

Runtime Debugging
The input parameters involved here are outlined below.

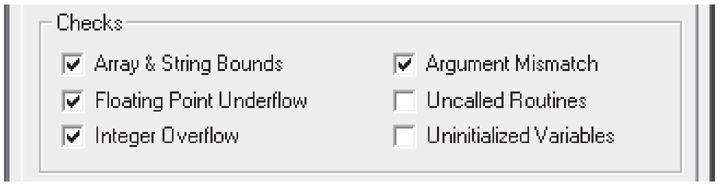

Runtime Debugging

☐ Enable addition of runtime debugging information.

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.


Checks

☑ Array & String Bounds
☑ Floating Point Underflow
☑ Integer Overflow
☑ Argument Mismatch
☐ Uncalled Routines
☐ Uninitialized Variables

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.

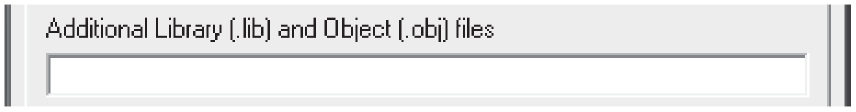
Uncalled Routines
The FORTRAN compiler issues a warning if a routine is not called in the program.

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.

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) Files
This input field allows you to indicate extra FORTRAN object (*.obj) or library (*.lib) 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 files.



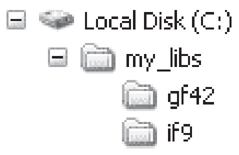
Object and library files provide a way to avoid the compilation of user written subroutines every time they are used in different projects. They are also useful for sharing custom models with others, where you do not want to share the source code itself.

User Library Path Method
A compiler specific directory 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 corresponding sub-directory of the specified *User Library Path* directory. Then, the user may freely switch between compilers without the need to re-specify the files.

The user must manually add these sub-directories to the directory specified by the **User Library Path** input in the Workspace dialog (go to *Edit* | *Workspace Settings* | *Fortran* in the main menu). Each sub-directory 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-directories specified. These must be named as follows:

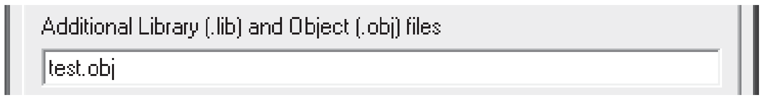
- gf42 (GNU GFortran 95 compiler)
- cf6 (Compaq Fortran 90 compiler)
- if9 (Intel Visual Fortran compiler versions 9 & 10)

For example, if a user specifies *C:\my_libs* as the **User Library Path**, and intends to use either the *GFortran compiler* or the *Intel Visual Fortran compiler*, then two sub-directories should be added to 'my_libs' as shown below:



Any library or object files required should be created with both compilers, and then added to the corresponding sub-directory.

If only the filename is entered, PSCAD will assume that the library or object file is located in the specified sub-directory in the **User Library Path** directory. For example, if a user specifies *C:\my_libs* as the User Library Path 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*.

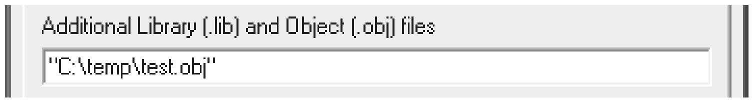

Additional Library (.lib) and Object (.obj) files

test.obj

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.

Absolute Path Method

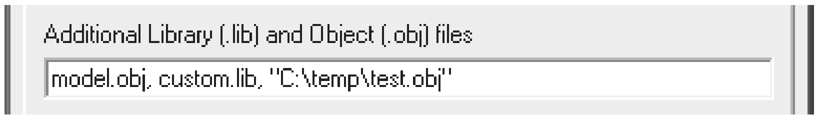
Files may also be referenced by either an absolute or relative path specification. If this method is chosen, then the directory specified in the *User Library Path* directory will be overridden. For example, a file referenced with an absolute path may appear as follows:


Additional Library (.lib) and Object (.obj) files

"C:\temp\test.obj"

Entering Multiple Files

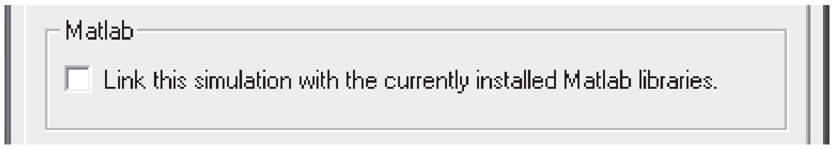
When entering multiple files into this field, they must be separated with a space, a comma or a semicolon. Both *User Library Path* and *Absolute Path* methods may be used simultaneously:


Additional Library (.lib) and Object (.obj) files

model.obj, custom.lib, "C:\temp\test.obj"

Matlab

The input parameters involved here are outlined below.



Matlab

Link this simulation with the currently installed Matlab libraries.

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 Workspace Settings dialog.

Chapter 8:

Tranmission 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 secondly, 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* μ*s* time step), may also be represented using an equivalent π-section representation. This is accomplished by using either the π-section component in the *master library*, where only the admittance and impedance data of the line segment is required, or by using the π-section equivalent component creator feature in the *Line Constants Program (LCP)*. π-sections are essentially a network of passive elements, and hence do not represent propagation delay.

A *15 km* line length to a *50 μs* 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 three distributed (travelling wave) models:

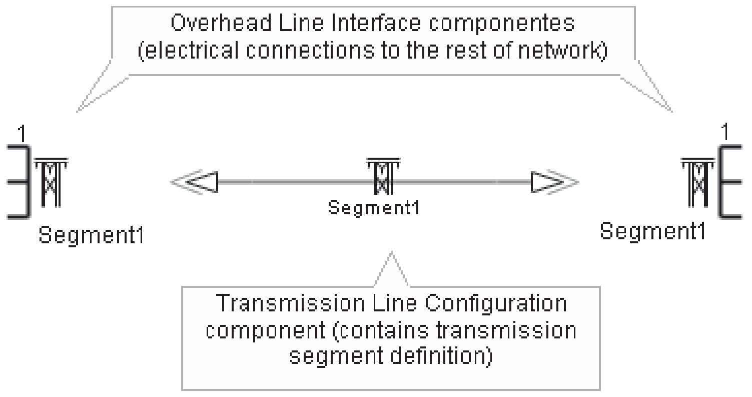
- Bergeron
- Frequency Dependent (Mode)
- Frequency Dependent (Phase)

The most accurate of these is the Frequency Dependent (Phase) model, which represents all frequency dependent effects of a transmission line, and should be used whenever in doubt. When using the Bergeron model, impedance/admittance data can also be entered directly to define the transmission segment.

For both of the frequency dependent models, detailed conductor information (i.e. segment geometry, conductor radius) must be given. For more details on the differences between models available in PSCAD, see *Chapter 9* - *Transmission Lines and Cables* in the EMTDC Manual.

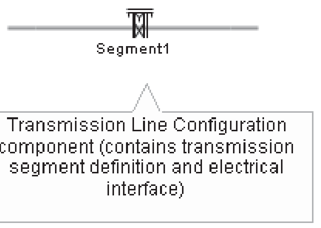
CONSTRUCTING OVERHEAD LINES

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:


Overhead Line Interface componentes
(electrical connections to the rest of network)
1
1
Segment1
Segment1
Segment1
Transmission Line Configuration component (contains transmission segment definition)

An Overhead Transmission Line (Remote Ends Method)

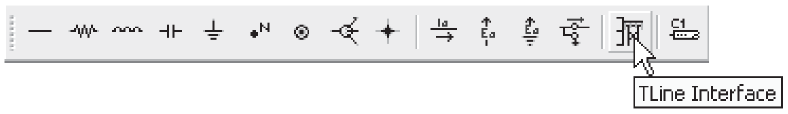
The second and more recently introduced method is to use the *Direct Connection* mode, where both the interfaces and the segment properties are housed within a single component.


Segment1
Transmission Line Configuration component (contains transmission segment definition and electrical interface)

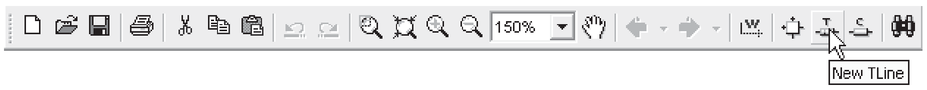
An Overhead Transmission Line (Direct Connection Method)

Building an Overhead Line (/w Remote Ends)

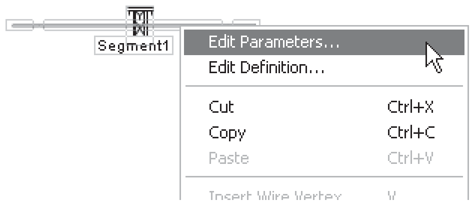
Add one *Transmission Line Configuration* component and two *Overhead Line Interface* components to the page. The most straightforward way to add the interface components is to use the *Electrical Palette* toolbar. See the section entitled *Adding Components to a Project* in Chapter 5 for details.

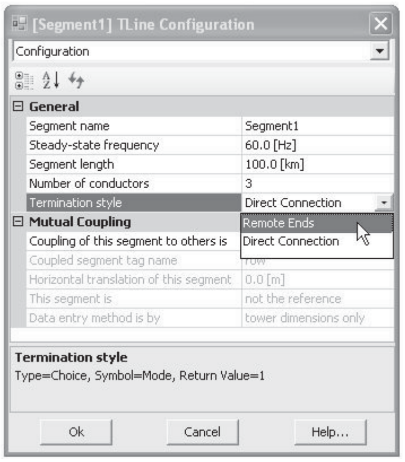


The *Transmission Line Configuration* component must be added by using the *Component Wizard*. In fact, there is a separate button (entitled **New TLine**) that provides a quick means by which to do so. These are located as part of the *Main Toolbar*.



Next, edit the parameters of the configuration component and ensure that **Termination Style | Remote Ends** is selected.





[Segment1] TLine Configuration

Configuration

General

Segment name	Segment1
Steady-state frequency	60.0 [Hz]
Segment length	100.0 [km]
Number of conductors	3
Termination style	Direct Connection

Mutual Coupling

Coupling of this segment to others is	Remote Ends / Direct Connection
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

Termination style
Type=Choice, Symbol=Mode, Return Value=1

Ok Cancel Help...

Edit the Parameters of the Transmission Line Configuration Component

Set Termination Style to Remote Ends

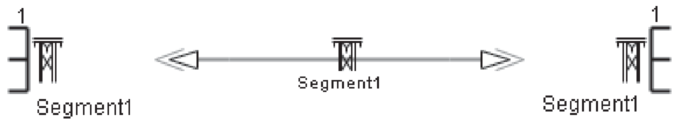
The *Transmission Line Configuration* component and two *Overhead Line Interface* components are interlinked through input parameters. Ensure that the following steps are taken:

1. **Segment Name**: Ensure that the *segment name* inputs in both the Transmission Line Configuration component and two Overhead Line Interface components are identical.

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.

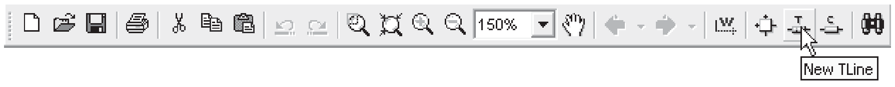
These components do not need to be in proximity to one another. As long as they are connected by name, each component can be located anywhere in the project, including different modules.

When finished, you should have something similar to that shown below on your project page:

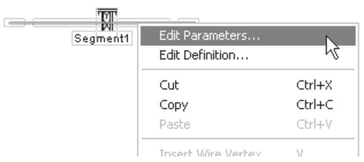


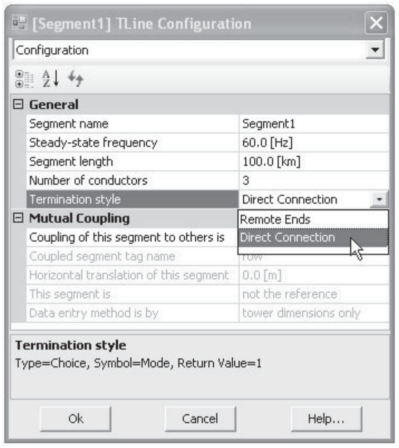
A 3-Conductor Overhead Transmission Line
(Remote Ends Mode)

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*. In fact, there is a separate button (entitled **New TLine**) that provides a quick means by which to do so. These are located as part of the Main Toolbar.



Next, edit the parameters of the configuration component and ensure that **Termination Style | Direct Connection** is selected.





[Segment1] TLine Configuration

Configuration

General

Segment name	Segment1
Steady-state frequency	60.0 [Hz]
Segment length	100.0 [km]
Number of conductors	3
Termination style	Direct Connection

Mutual Coupling

Coupling of this segment to others is	Remote Ends / Direct Connection
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

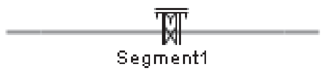
Termination style
Type=Choice, Symbol=Mode, Return Value=1

Edit the Parameters of the Transmission Line Configuration Component

Set Termination Style to Direct Connection

When finished, you should have something similar to that shown below on your project page:

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.



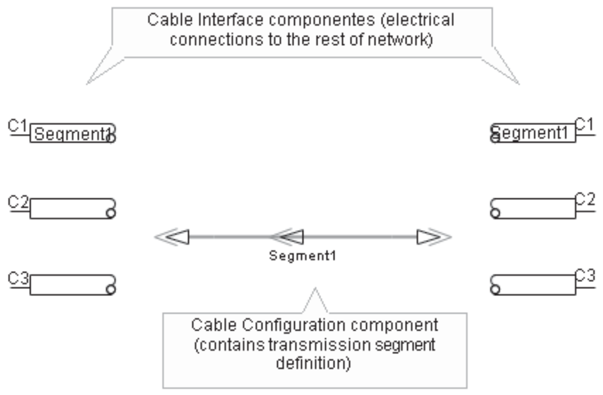
An Overhead Transmission Line (Direct Connection Mode)

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 of *Edit Properties...*. See *Editing an Overhead Line Segment Cross-Section* later in the section for details.

CONSTRUCTING UNDERGROUND CABLE SYSTEMS

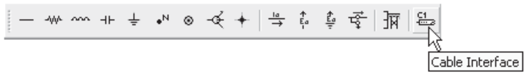
Unlike overhead lines, the *Direct Connection* method cannot be used with cable systems. This leaves just the *Remote Ends* method, which involves a *Cable Configuration* component with two *Cable Interface* components, representing the sending and receiving ends of the cable. The purpose of the interface components is to provide connection of the cable to the greater electric network. An underground cable system is illustrated below:


Cable Interface componentes (electrical connections to the rest of network)
C1
Segment1
C2
C3
Segment1
C1
C2
C3
Segment1
Cable Configuration component (contains transmission segment definition)

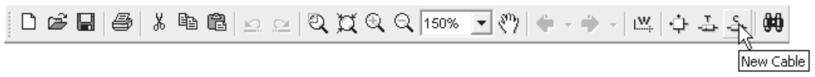
An Underground Cable

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 to use the *Electrical Palette* toolbar. See the section entitled *Adding Components to a Project* in Chapter 5 for details.



The *Cable Configuration* component must be added by using the *Component Wizard*. In fact, there is a separate button (entitled **New Cable**) that provides a quick means by which to do so. These are located as part of the *Main Toolbar*.

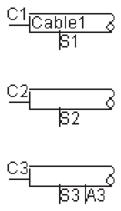


The *Cable Configuration* component and two *Cable Interface* components are interlinked through input parameters. Ensure that the following steps are taken:

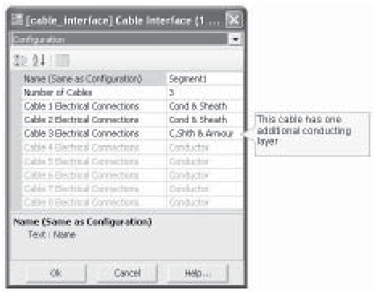
1. **Segment Name**: Ensure that the *Segment Name* inputs in both the *Cable Configuration* component and two *Cable Interface* components are identical.

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*.

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.



Cable Interface Component



[cable_interface] Cable Interface (1 ...

Configuration

Name (Same as Configuration)	Segment1
Number of Cables	3
Cable 1 Electrical Connections	Cond & Sheath
Cable 2 Electrical Connections	Cond & Sheath
Cable 3 Electrical Connections	C,Shth & Armour
Cable 4 Electrical Connections	Conductor
Cable 5 Electrical Connections	Conductor
Cable 6 Electrical Connections	Conductor
Cable 7 Electrical Connections	Conductor
Cable 8 Electrical Connections	Conductor

This cable has one additional conducting layer

Name (Same as Configuration)
Text : Name

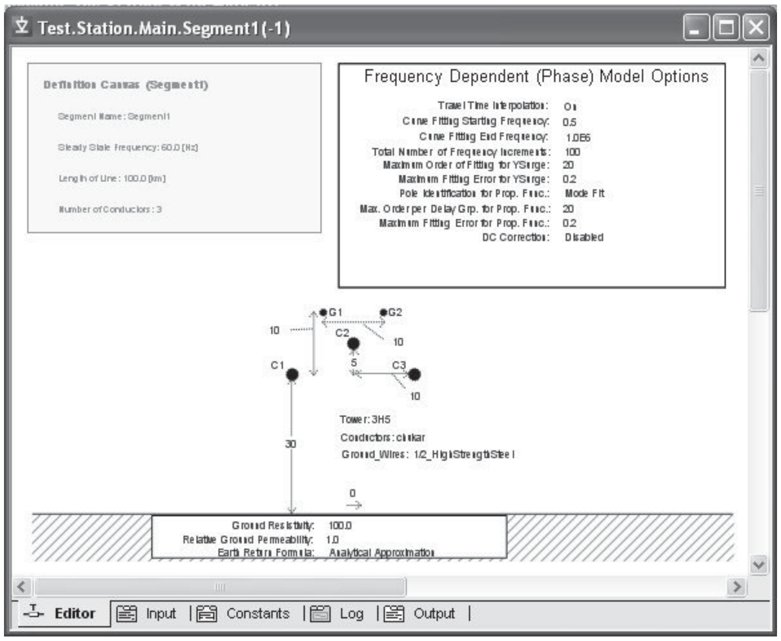
Specifying Cable Set-up in Cable Interface Parameters Dialog

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 on these parameters, see the *Cable Configuration* component help. To edit parameters specific to towers, conductors and ground, select **Edit Definition...**instead of *Edit Properties...*. 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.


Test.Station.Main.Segment1(-1)

Definition Canvas (Segment1)

Segment Name: Segment1

Steady State Frequency: 60.0 [Hz]

Length of Line: 100.0 [km]

Number of Conductors: 3

Frequency Dependent (Phase) Model Options

Travel Time Interpolation:	On
Curve Fitting Starting Frequency:	0.5
Curve Fitting End Frequency:	1.0E6
Total Number of Frequency Increments:	100
Maximum Order of Fitting for YSurge:	20
Maximum Fitting Error for YSurge:	0.2
Pole Identification for Prop. Func.:	Mode Fit
Max. Order per Delay Grp. for Prop. Func.:	20
Maximum Fitting Error for Prop. Func.:	0.2
DC Correction:	Disabled

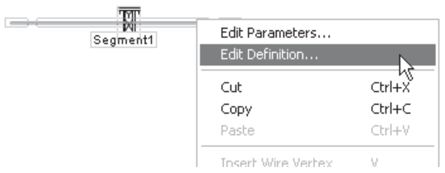
Tower: 3H5
Conductors: chukar
Ground_Wires: 1/2_HighStrengthSteel

Ground Resistivity:	100.0
Relative Ground Permeability:	1.0
Earth Return Formula:	Analytical Approximation

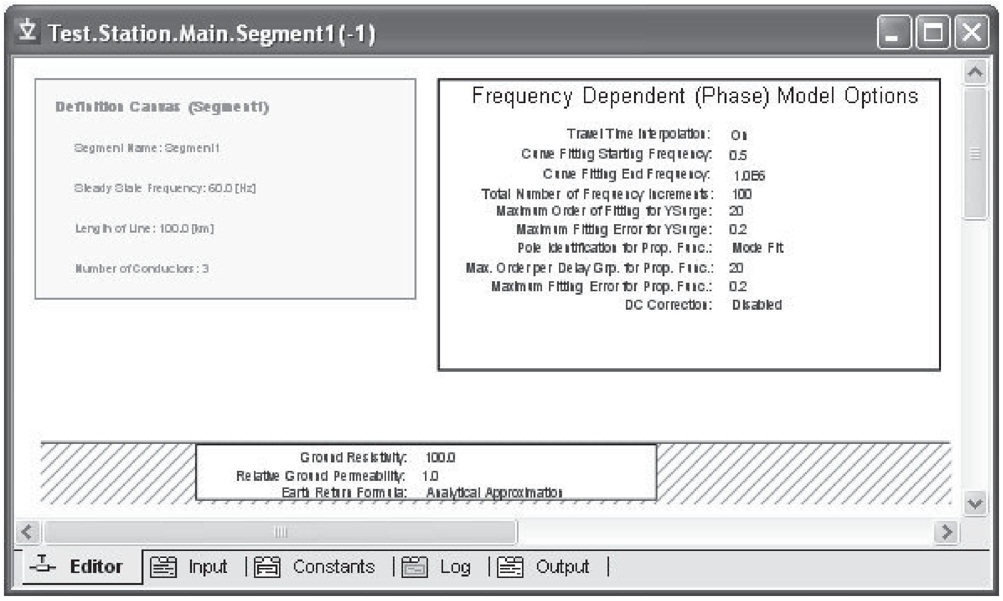
The main section tab is entitled *Editor*, and is the default section view when the editor is invoked. The four remaining sections are for viewing files related to the segment. Note that these files will not exist unless either the segment has been solved manually, or the project has been compiled.

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...**, 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.



Definition Canvas (Segment1)

Segment Name: Segment1

Steady State Frequency: 60.0 [Hz]

Length of Line: 100.0 [km]

Number of Conductors: 3

Frequency Dependent (Phase) Model Options

Travel Time Interpolation:	On
Curve Fitting Starting Frequency:	0.5
Curve Fitting End Frequency:	1.0E6
Total Number of Frequency Increments:	100
Maximum Order of Fitting for YSurge:	20
Maximum Fitting Error for YSurge:	0.2
Pole Identification for Prop. Func.:	Mode Fit
Max. Order per Delay Grp. for Prop. Func.:	20
Maximum Fitting Error for Prop. Func.:	0.2
DC Correction:	Disabled

Ground Resistivity:	100.0
Relative Ground Permeability:	1.0
Earth Return Formula:	Analytical Approximation

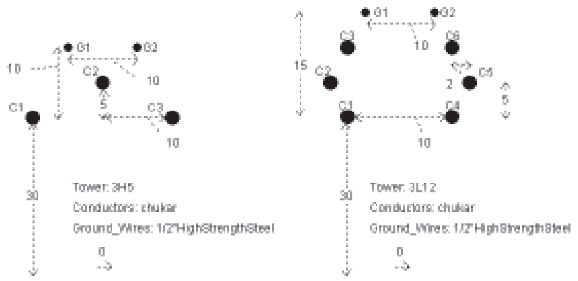
By default, the editor section will contain three graphical objects:

- **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.

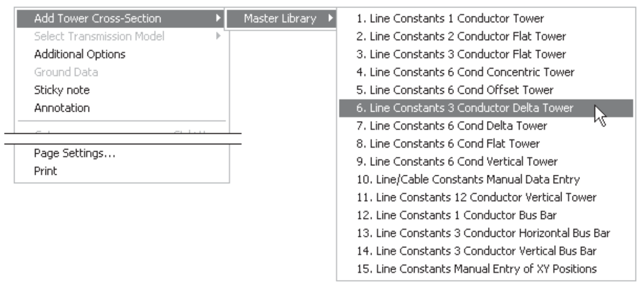
A fourth object required is the definition of the transmission line itself. This definition can be a geometrical cross-section of the *Transmission Line Tower* component, or can be a *Manual Entry of Admittance/Impedance Data* (Bergeron model only). In either case, this is left for the user to add manually.



Transmission Line Tower Components

Adding a Tower Component

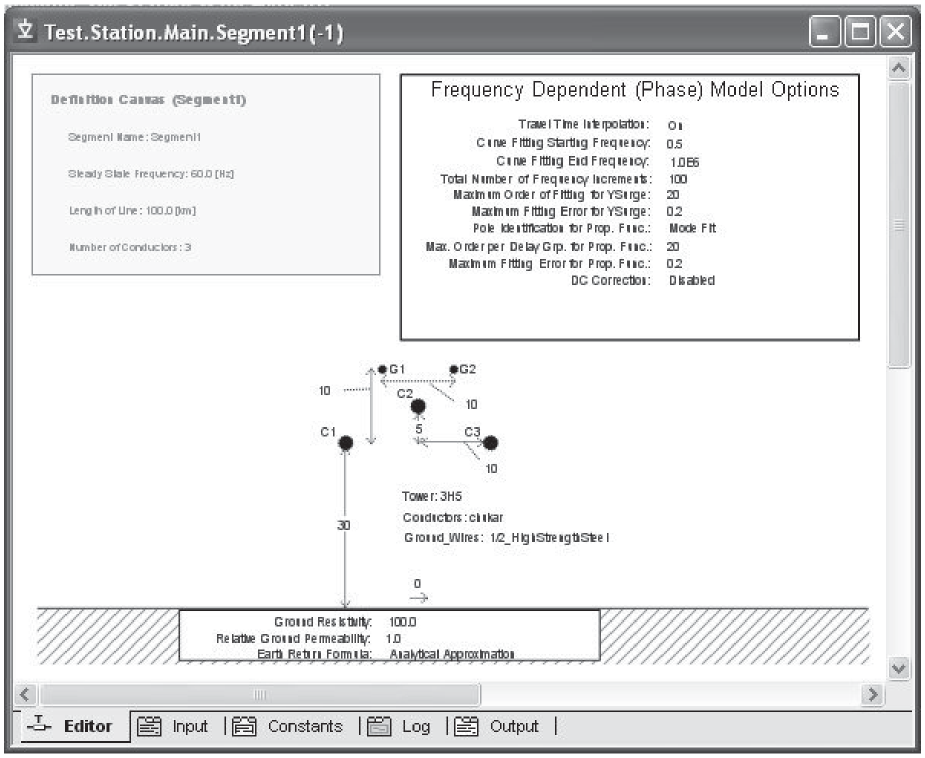
Tower components can be added to the editor in one of two ways. The most straightforward method is to use the right-click pop-up menu. In the *Editor* section (tab), move the mouse pointer over a blank area of the window. Right-click and select **Add Tower Cross-Section**. A sub-menu will appear containing 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.


Add Tower Cross-Section
Select Transmission Model
Additional Options
Ground Data
Sticky note
Annotation
Page Settings...
Print
Master Library
1. Line Constants 1 Conductor Tower
2. Line Constants 2 Conductor Flat Tower
3. Line Constants 3 Conductor Flat Tower
4. Line Constants 6 Cond Concentric Tower
5. Line Constants 6 Cond Offset Tower
6. Line Constants 3 Conductor Delta Tower
7. Line Constants 6 Cond Delta Tower
8. Line Constants 6 Cond Flat Tower
9. Line Constants 6 Cond Vertical Tower
10. Line/Cable Constants Manual Data Entry
11. Line Constants 12 Conductor Vertical Tower
12. Line Constants 1 Conductor Bus Bar
13. Line Constants 3 Conductor Horizontal Bus Bar
14. Line Constants 3 Conductor Vertical Bus Bar
15. Line Constants Manual Entry of XY Positions

You can also copy and paste tower components directly from any library project. Open the library in Circuit 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:

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.



Definition Canvas (Segment1)

Segment Name: Segment1

Steady State Frequency: 60.0 [Hz]

Length of Line: 100.0 [km]

Number of Conductors: 3

Frequency Dependent (Phase) Model Options

Travel Time Interpolation:	On
Curve Fitting Starting Frequency:	0.5
Curve Fitting End Frequency:	1.0E6
Total Number of Frequency Increments:	100
Maximum Order of Fitting for YSurge:	20
Maximum Fitting Error for YSurge:	0.2
Pole Identification for Prop. Func.:	Mode Fit
Max. Order per Delay Grp. for Prop. Func.:	20
Maximum Fitting Error for Prop. Func.:	0.2
DC Correction:	Disabled

Tower: 3H5

Conductors: chukar

Ground_Wires: 1/2_HighStrengthSteel

Ground Resistivity:	100.0
Relative Ground Permeability:	1.0
Earth Return Formula:	Analytical Approximation

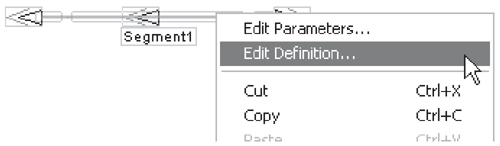
Multiple towers may be added to a single configuration. Just remember to ensure that the conductors are numbered properly within each of the tower components, and that the *x-positions* of the new towers in the corridor are adjusted. Also, any conductors added by the additional towers must be reflected in the corresponding *Transmission Line Interface* components (if in *Remote Ends* mode).

Editing Tower Parameters

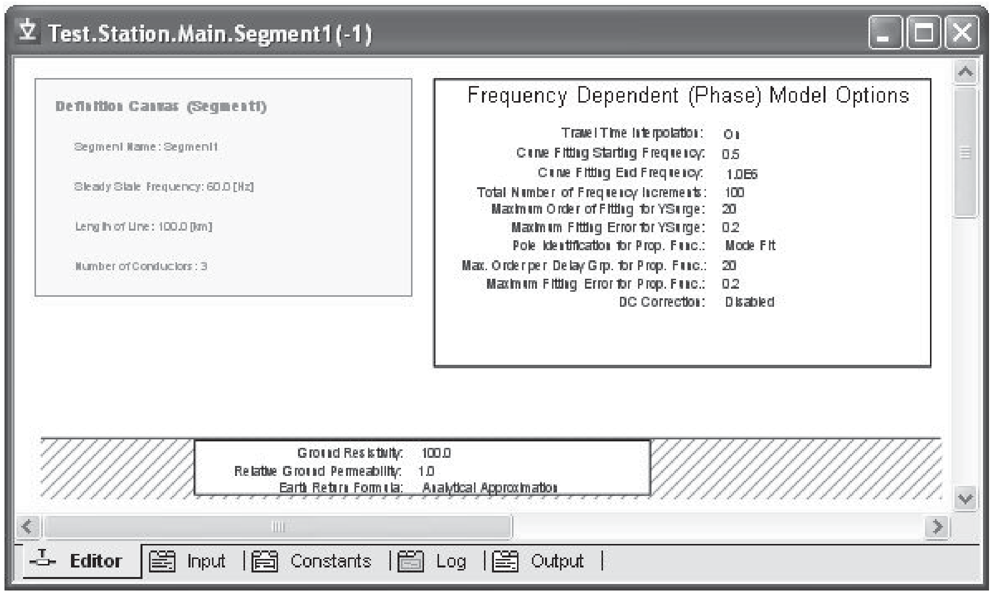
Tower parameters can be edited through the corresponding tower parameters dialog window. Right-click over the tower component (without selecting it) and select **Edit Parameters...** to access this dialog.

Editing a Cable Segment Definition

To invoke the editor, right-click over the *Cable Configuration* component (without selecting it) and select **Edit Definition...**, 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.



Definition Canvas (Segment1)

Segment Name : Segment1

Steady State Frequency: 60.0 [Hz]

Length of Line : 100.0 [km]

Number of Conductors : 3

Frequency Dependent (Phase) Model Options

Travel Time Interpolation:	On
Curve Fitting Starting Frequency:	0.5
Curve Fitting End Frequency:	1.0E6
Total Number of Frequency Increments:	100
Maximum Order of Fitting for YSurge:	20
Maximum Fitting Error for YSurge:	0.2
Pole Identification for Prop. Func.:	Mode Fit
Max. Order per Delay Grp. for Prop. Func.:	20
Maximum Fitting Error for Prop. Func.:	0.2
DC Correction:	Disabled

Ground Resistivity:	100.0
Relative Ground Permeability:	1.0
Earth Return Formula:	Analytical Approximation

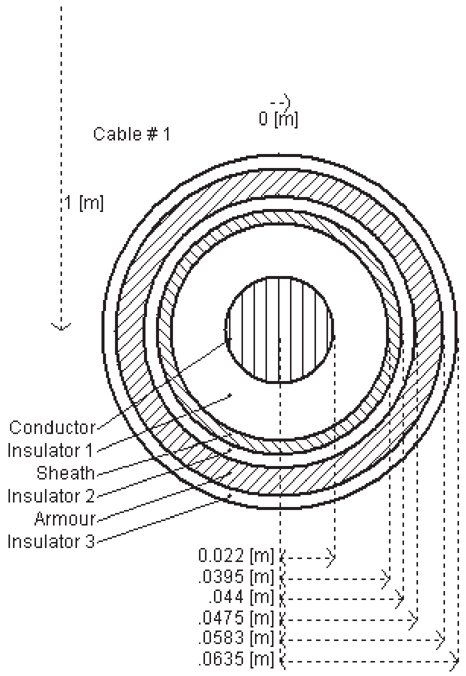
By default, the *Editor* section will contain three graphical objects:

- **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.

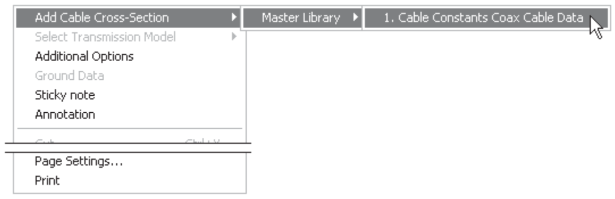
A fourth object required is the definition of the cable itself. This definition is a geometrical cross-section of the cable called a *Cable Cross-Section*. The user must add this manually.



Cable Cross-Section Component

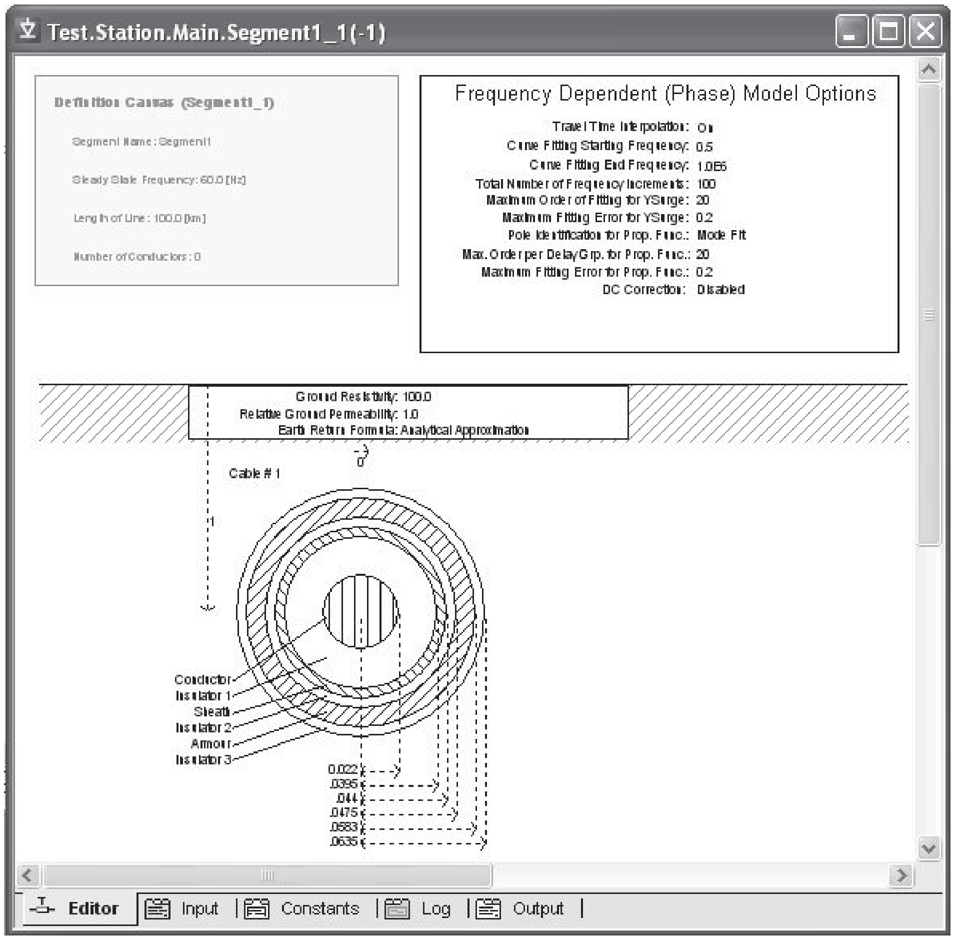
Adding a Cable Cross-Section Component

Cable Cross-Section components can be added to the editor in one of two ways. The most straightforward method is to use the right-click pop-up menu: In the *Editor* section (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 Circuit 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:


Test.Station.Main.Segment1_1(-1)

Definition Canvas (Segment1_1)

Segment Name : Segment1

Steady State Frequency: 60.0 [Hz]

Length of Line : 100.0 [km]

Number of Conductors : 0

Frequency Dependent (Phase) Model Options

```
                 Travel Time Interpolation:  On
          Curve Fitting Starting Frequency:  0.5
               Curve Fitting End Frequency:  1.0E6
      Total Number of Frequency Increments:  100
        Maximum Order of Fitting for YSurge:  20
          Maximum Fitting Error for YSurge:  0.2
     Pole Identification for Prop. Func.:  Mode Fit
  Max. Order per Delay Grp. for Prop. Func.:  20
       Maximum Fitting Error for Prop. Func.:  0.2
                             DC Correction:  Disabled
```

Ground Resistivity: 100.0
Relative Ground Permeability: 1.0
Earth Return Formula: Analytical Approximation

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.

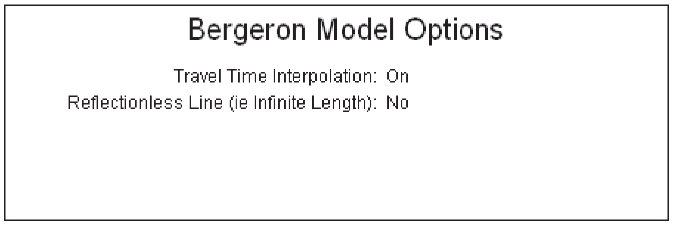
Multiple cables may be added to a single configuration. Just remember to ensure that the cables are numbered properly, and that the *xy*-positions of the new cross-sections in the corridor are adjusted. Also, any conductors added by the additional cables must be reflected in the corresponding *Cable Interface* components.

Editing Cross-Section Parameters

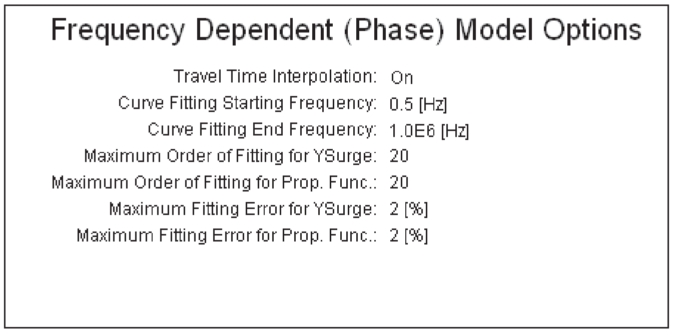
Cable cross-section parameters can be edited through the corresponding cross-section parameters dialog window. Right-click over the cross-section component (without selecting it) and select **Edit Parameters...** to access this dialog.

Selecting the Proper Line Model

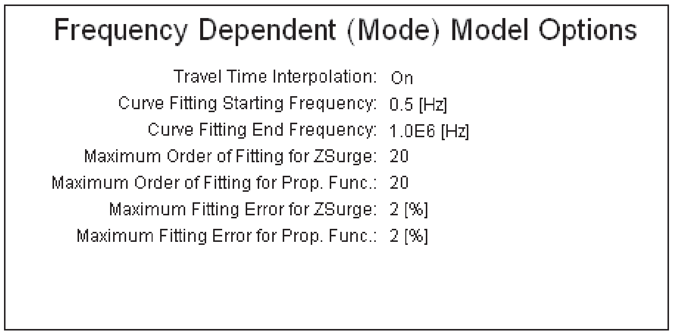
There are three types of distributed (i.e. travelling wave) transmission models that may be selected to represent your transmission corridor: The *Bergeron* model, the *Frequency-Dependent (Mode)* model, and the *Frequency-Dependent (Phase)* model. These models exist as components in the master library 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 [%]

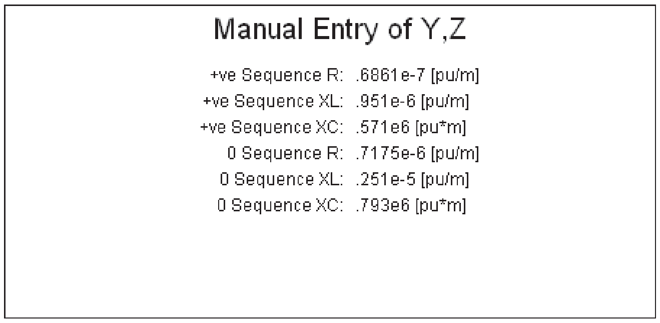

Frequency Dependent (Mode) Model Options

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 π-section in a distributed manner (not using lumped parameters like π-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 two 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.

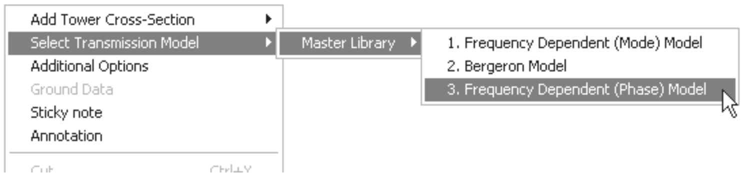
For further technical detail or more information on choosing a line model, see *Chapter 9 - Transmission Lines and Cables* in the EMTDC Manual.

The line configuration cannot contain more than one model component. This menu item will be disabled if a model already exists in the editor.

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!

Adding a Line Model

Line model components can be added to the editor in one of two ways. The most straightforward method is to use the right-click pop-up menu. In the *Editor* section (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 Circuit 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

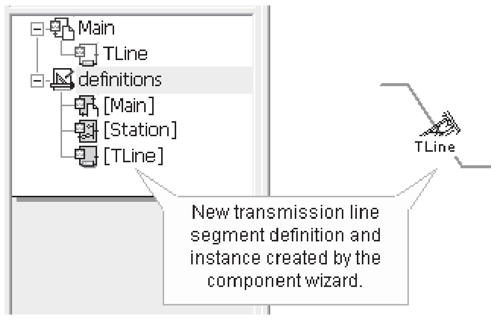
From the perspective of the PSCAD application itself, transmission segments are not unlike modules; they possess a canvas, input parameters, and a definition. It would seem then that transmission segments could also be multiple instanced, like their module component counterparts – this is indeed possible. 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.

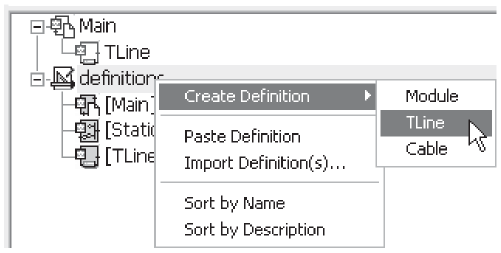
Surely the segment length could easily be made part of the definition, instead of existing as an instance-based parameter, but then a new segment definition would need to be created whenever the length changed. This would mean that for the same tower configuration, a new definition would need to be created 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.


Main
TLine
definitions
[Main]
[Station]
[TLine]
TLine
New transmission line segment definition and instance created by the component wizard.

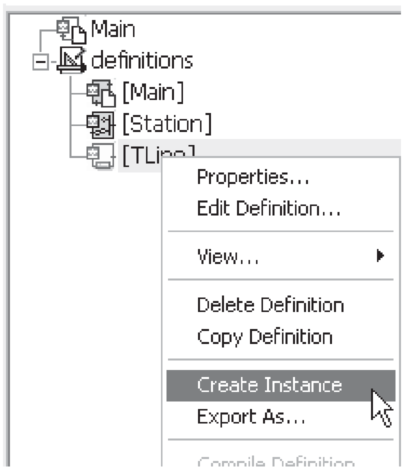
You may however, forego the use of the component wizard and instead create a definition directly. Right-click on the definitions branch in the workspace window and select either **Create Definition | TLine or Create Definition | Cable**.



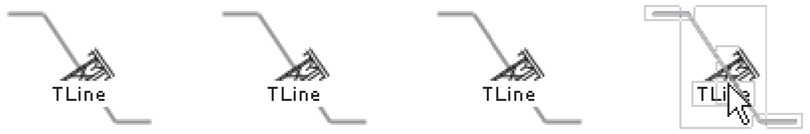
To produce an instance of this newly created definition, right-click on the definition in the workspace and select **Create Instance**. See *Creating the First Instance of a Definition* in Chapter 5 of this manual.

Multiple Instances of the Same Segment Definition
Once a new transmission line or cable definition has been created, you may modify it in the same manner as you would a module. Of course, the definitions of the two component types are different, but in general they are similar.

Creating the first instance of the new segment definition is performed the same as any other component definition: Right-click on the definition and select **Create Instance**. Then right-click on the Circuit canvas and select **Paste**.



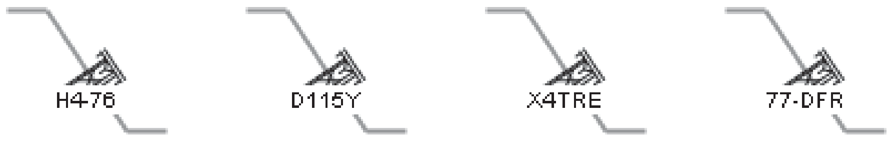
From this point forward, any instance of the transmission segment can be instantiated by simply copying and pasting the component on the Circuit canvas.



As discussed above, the final step in multiply instancing 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.

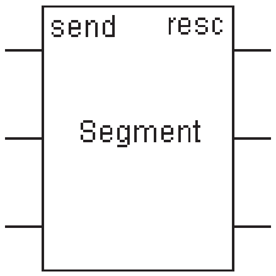
Multiple Instances without Solving Multiple Times
It is possible to instantiate a transmission segment many times, while ensuring it is only solved once. This is desirable for example, if the transmission line or cable segments possess the same cross-sectional configuration (i.e. definition) *and* the same length. A practical situation would be perhaps a cross-bonded cable or transposed aerial line, where the transposed segments are all equal.

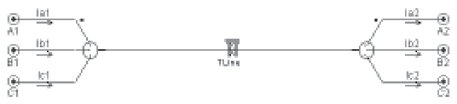
The above is accomplished by ‘wrapping’ the segment within a module canvas. This essentially makes the transmission line or cable object part of the definition of its parent module. Since a module definition is only solved once, so shall it be for the transmission line. EMTDC will simply access the same *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 *six* 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*.

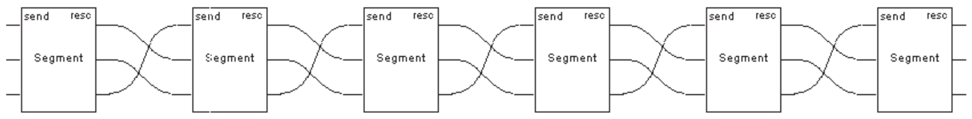




New Module Graphic

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

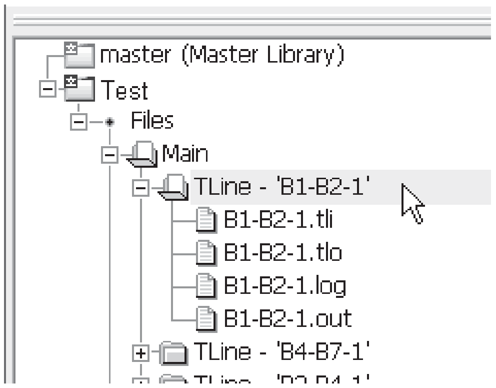
Whenever a project containing a transmission line or a cable segment is compiled (or the line constants are solved manually), text files involved in the calculation are created. PSCAD uses information extracted from the various components involved with the transmission system, and constructs a *Segment Input (*.tli)* file. This file is then used as input by the *Line Constants Program (LCP)* - a separate executable supplied with your PSCAD software.

For more on the Line Constants Program, see *Chapter 9 - Transmission Lines and Cables* in the EMTDC Manual.

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*)

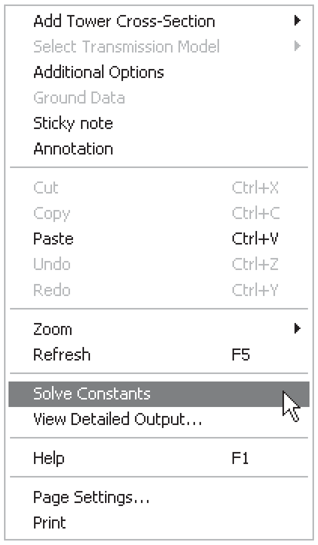
These files are located in the project temporary directory (*<project name>.emt*) and may be viewed either directly within the *Transmission Segment Definition Editor* (by clicking the corresponding tab window), or in the *Files* tree in the workspace.



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.

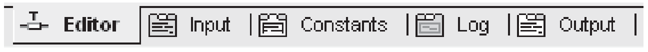
Solving the Transmission Segment Constants Manually
To manually solve a specific transmission line or cable segment in your project, first open the *Transmission Segment Definition Editor* (see previous section for instructions, if needed).

Right-click over a blank area of the canvas and select **Solve Constants**:



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, 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
The constants 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.

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 the user scan through this file to ensure that no problems occurred while the line constants were being solved.

If questions arise regarding the output in this file, please contact the PSCAD Support Desk (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 ***Z*** per-unit length (Ω/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*:

$$Z = \begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1N} \\ Z_{21} & Z_{22} & \cdots & Z_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Z_{N1} & Z_{N2} & \cdots & Z_{NN} \end{bmatrix} [\Omega/m]$$

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*:

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1N} \\ Y_{21} & Y_{22} & \cdots & Y_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{N1} & Y_{N2} & \cdots & Y_{NN} \end{bmatrix} [S/m]$$

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 (Z_{LL})
This data represents the long-line corrected, series impedance matrix ***Z*** (Ω). 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 π-section equivalent is being derived to represent the entire line length at a specific frequency.

Long-Line Corrected Shunt Admittance Matrix (Y_{LL})
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 π-section equivalent is being derived to represent the entire line length at a specific frequency.

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***:

$$T = \frac{1}{2\sqrt{3}} \cdot \begin{bmatrix} 2 & 2 & 2 \\ 2 & -1-j\sqrt{3} & -1+j\sqrt{3} \\ 2 & -1+j\sqrt{3} & -1-j\sqrt{3} \end{bmatrix}$$

Transformation Matrix

Sequence Impedance Matrix (Zsq)
This data represents the system sequence impedance matrix ***Z**$_{sq}$* per-unit length (Ω/m). ***Z**$_{sq}$* is derived directly from the series impedance matrix ***Z*** and the sequence transform matrix ***T*** (both described above) as follows:

$$Z_{sq} = T^{-1} \cdot Z \cdot T$$

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:

$$Z_{sq} = \begin{bmatrix} Z_{01} & 0 & 0 & Z_{0M} & 0 & 0 \\ 0 & Z_{+1} & 0 & 0 & 0 & 0 \\ 0 & 0 & Z_{-1} & 0 & 0 & 0 \\ Z_{0M} & 0 & 0 & Z_{02} & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_{+2} & 0 \\ 0 & 0 & 0 & 0 & 0 & Z_{-2} \end{bmatrix}$$

Sequence Impedance Matrix (Two, Balanced 3-Phase Circuits)

Where,

Z_{0n} =	Zero sequence impedance of the *n*th circuit [Ω/m]
Z_{+n} =	Positive sequence impedance of the *n*th circuit [Ω/m]
Z_{-n} =	Negative sequence impedance of the *n*th circuit [Ω/m]
Z_{0M} =	Zero sequence mutual impedance [Ω/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 six conductors are included in the transposition, a sequence matrix will not be provided.

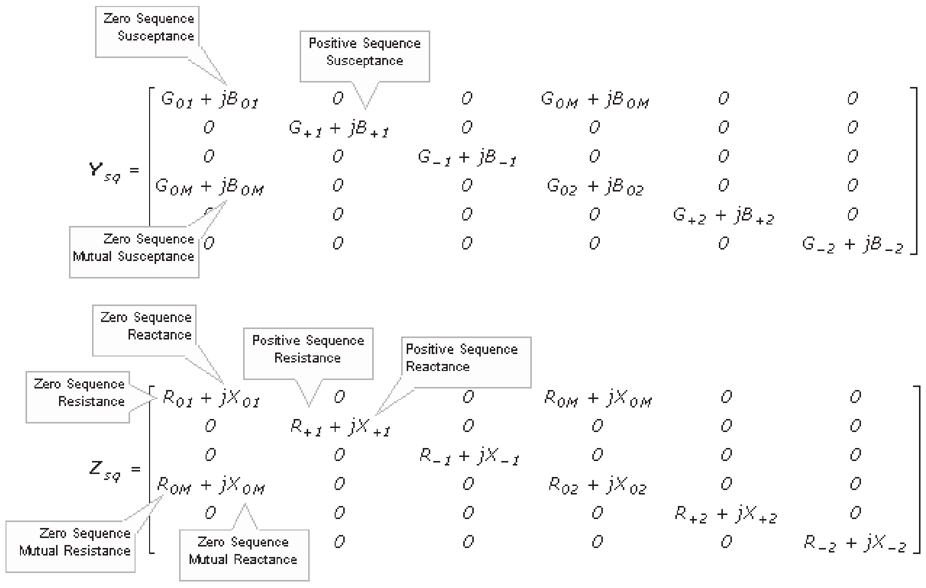
The positions of elements in this matrix are dependent on the manner in which the conductors/ground wires have been numbered. The type of ideal transposition that has been selected will also affect this matrix.

Sequence Admittance Matrix (Ysq)
This data represents the system sequence admittance matrix $\boldsymbol{Y_{sq}}$ per-unit length (S/m). $\boldsymbol{Y_{sq}}$ is derived directly from the shunt admittance matrix $\boldsymbol{Y}$ and the sequence transform matrix $\boldsymbol{T}$ (both described above) as follows:

$$Y_{sq} = T^{-1} \cdot Y \cdot T$$

The same concepts described for the sequence impedance matrix $\boldsymbol{Z_{sq}}$, apply to $\boldsymbol{Y_{sq}}$.

Load Flow RXB Formatted Data
This data is simply the data given in the Sequence Data section above, but reformatted for easy reading. The following diagram shows from where the data is taken:


Zero Sequence Susceptance
Positive Sequence Susceptance
Zero Sequence Mutual Susceptance
Zero Sequence Reactance
Positive Sequence Resistance
Positive Sequence Reactance
Zero Sequence Resistance
Zero Sequence Mutual Resistance
Zero Sequence Mutual Reactance

$$Y_{sq} = \begin{bmatrix} G_{01} + jB_{01} & 0 & 0 & G_{0M} + jB_{0M} & 0 & 0 \\ 0 & G_{+1} + jB_{+1} & 0 & 0 & 0 & 0 \\ 0 & 0 & G_{-1} + jB_{-1} & 0 & 0 & 0 \\ G_{0M} + jB_{0M} & 0 & 0 & G_{02} + jB_{02} & 0 & 0 \\ 0 & 0 & 0 & 0 & G_{+2} + jB_{+2} & 0 \\ 0 & 0 & 0 & 0 & 0 & G_{-2} + jB_{-2} \end{bmatrix}$$

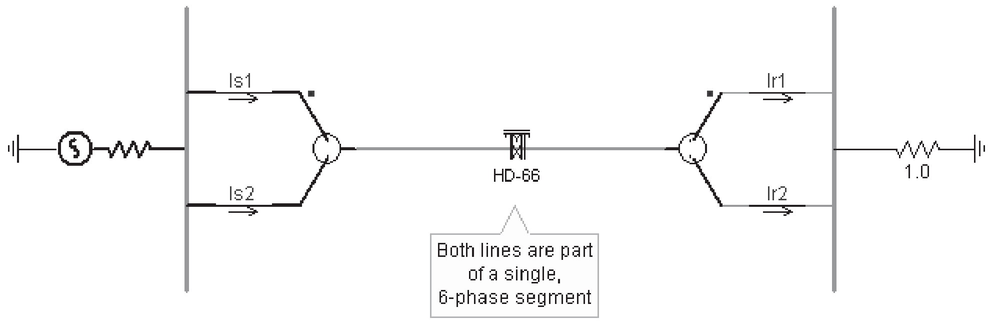
$$Z_{sq} = \begin{bmatrix} R_{01} + jX_{01} & 0 & 0 & R_{0M} + jX_{0M} & 0 & 0 \\ 0 & R_{+1} + jX_{+1} & 0 & 0 & 0 & 0 \\ 0 & 0 & R_{-1} + jX_{-1} & 0 & 0 & 0 \\ R_{0M} + jX_{0M} & 0 & 0 & R_{02} + jX_{02} & 0 & 0 \\ 0 & 0 & 0 & 0 & R_{+2} + jX_{+2} & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{-2} + jX_{-2} \end{bmatrix}$$

Sequence Admittance and Sequence Impedance Matrices
(Two, Balanced 3-Phase Circuits)

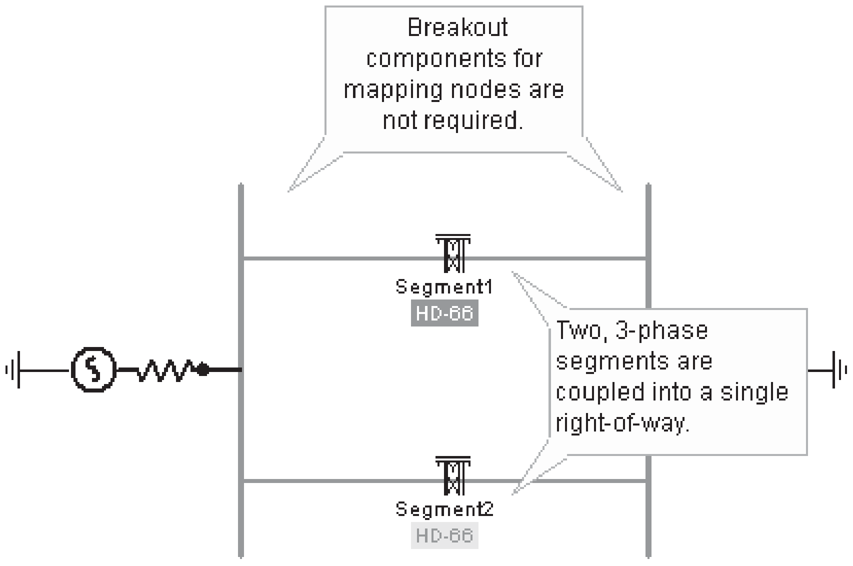
MUTUAL COUPLING

Mutual coupling of transmission system segments enables the user to couple (or parallel) multiple transmission segments together, provided their lengths are all identical. In the past, this was possible only by manually combining two or more separate systems of towers (or cable cross-sections) into the same transmission segment. Now PSCAD can be programmed to perform this task automatically, and even toggle between coupled and non-coupled states, all the while maintaining the individuality of each segment.

Mutual coupling can also circumvent certain node mapping issues when dealing with multi-phase wires and buses. For example, you can now couple multiple transmission segments connected to the same bus, without the need for *Breakout* components!



Single, 6-Phase Segment

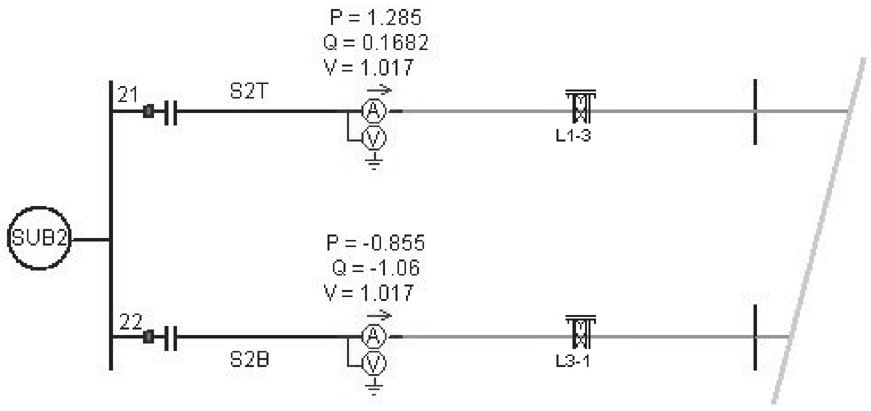

Breakout components for mapping nodes are not required.
Segment1
HD-66
Two, 3-phase segments are coupled into a single right-of-way.
Segment2
HD-66

Two Mutually Coupled, 3-Phase Segments

Mutual Coupling of Transmission Segments

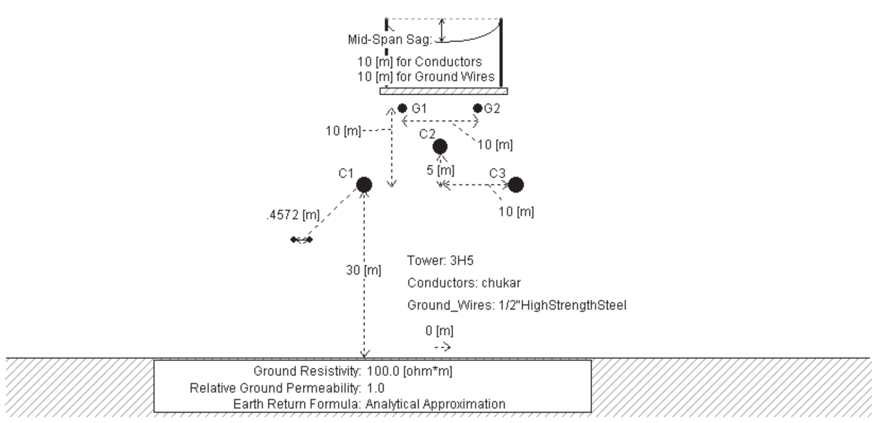
In order to couple two or more segments together, they first must possess the same length. Each unique segment in a collection of coupled segments (referred to as a *Right-Of-Way* or *ROW*) must possess a *Horizontal Translation* value, which serves to orientate segments on a global *x*-axis.

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:

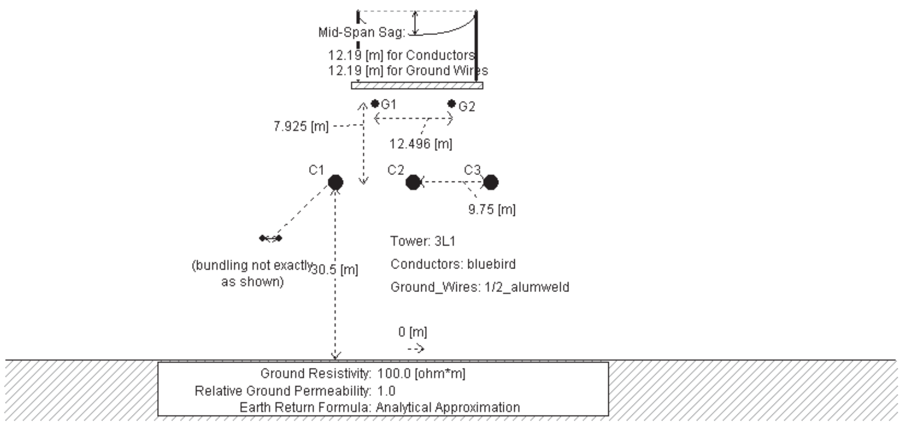


Transmission Segments on the Project Canvas

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.


Mid-Span Sag:
10 [m] for Conductors
10 [m] for Ground Wires
G1
G2
10 [m]
C2
10 [m]
C1
5 [m]
C3
10 [m]
.4572 [m]
30 [m]
Tower: 3H5
Conductors: chukar
Ground_Wires: 1/2"HighStrengthSteel
0 [m]
Ground Resistivity: 100.0 [ohm*m]
Relative Ground Permeability: 1.0
Earth Return Formula: Analytical Approximation

Transmission Segment L1-3 Editor Canvas

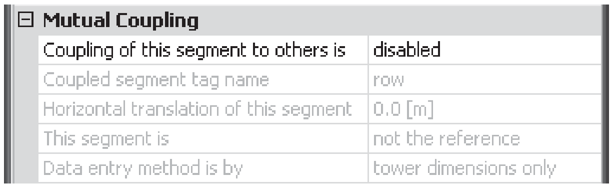

Mid-Span Sag:
12.19 [m] for Conductors
12.19 [m] for Ground Wires
G1
G2
7.925 [m]
12.496 [m]
C1
C2
C3
9.75 [m]
(bundling not exactly as shown)
30.5 [m]
Tower: 3L1
Conductors: bluebird
Ground_Wires: 1/2_alumweld
0 [m]
Ground Resistivity: 100.0 [ohm*m]
Relative Ground Permeability: 1.0
Earth Return Formula: Analytical Approximation

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

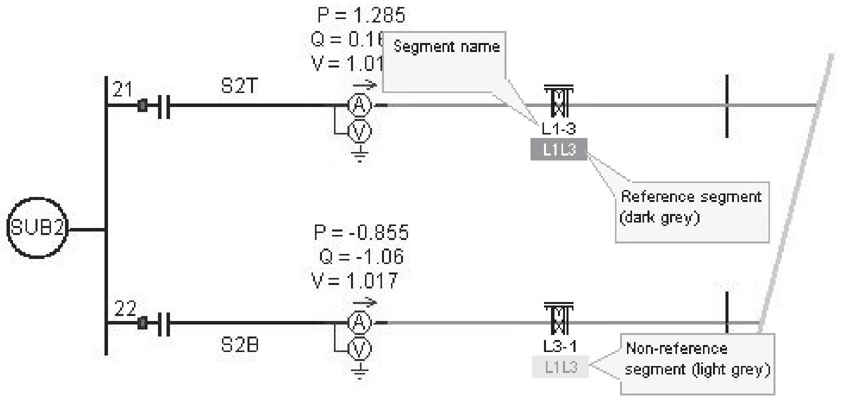
There a few parameters involved in defining a Right-Of-Way:

- **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 segments *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:

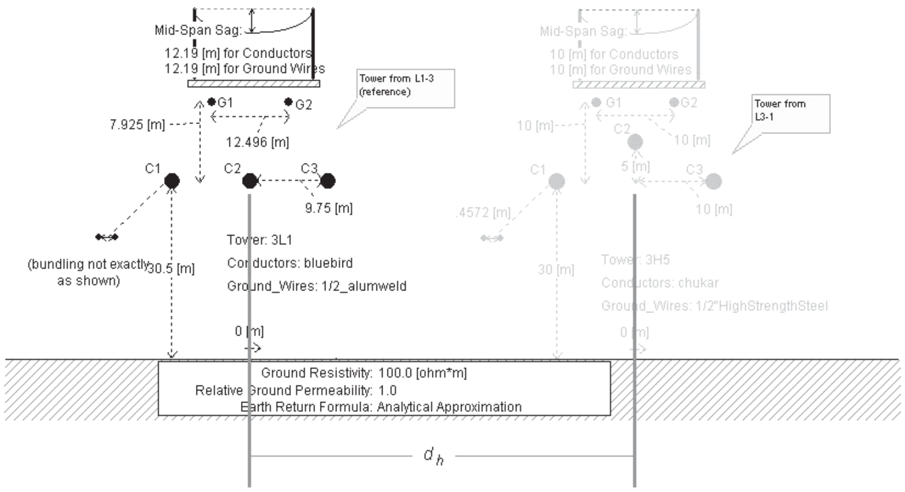


Behind the Scenes

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 by PSCAD 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.

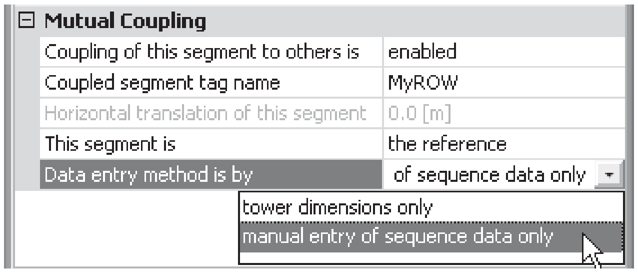
The mutual coupling algorithm takes care of all conductor phasing and cable numbering automatically.

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.

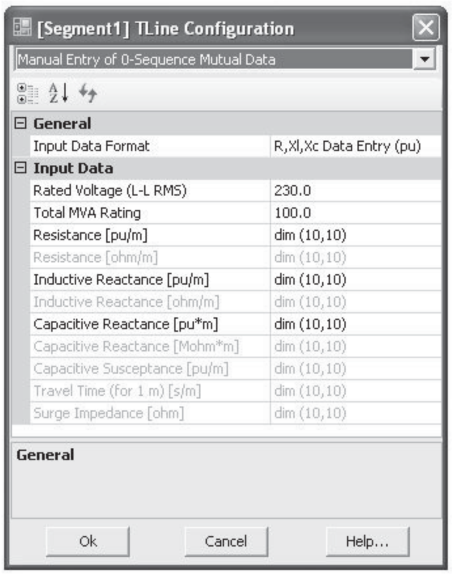
What occurs instead with sequence data entry is that PSCAD will construct ***Y*** and ***Z*** matrices for the entire coupled system; these matrices are based on information from both the individual *Manual Entry of Y,Z* components in each segment involved, as well as additional *0-sequence mutual data*. 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*.


Mutual Coupling

Coupling of this segment to others is	enabled
Coupled segment tag name	MyROW
Horizontal translation of this segment	0.0 [m]
This segment is	the reference
Data entry method is by	of sequence data only

This will enable the second category page in the dialog called *Manual Entry of 0-Sequence Mutual Data*.

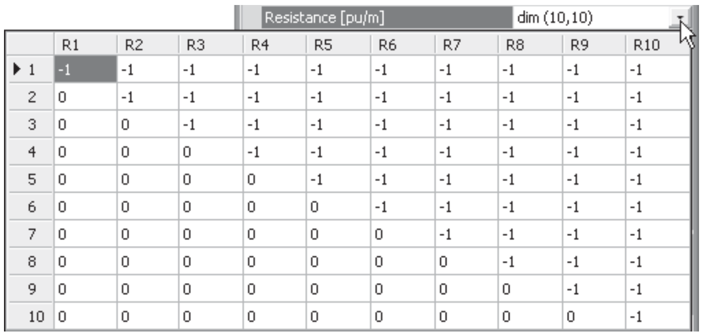

[Segment1] TLine Configuration
Manual Entry of 0-Sequence Mutual Data
Ok
Cancel
Help...

General	
Input Data Format	R,Xl,Xc Data Entry (pu)
Input Data	
Rated Voltage (L-L RMS)	230.0
Total MVA Rating	100.0
Resistance [pu/m]	dim (10,10)
Resistance [ohm/m]	dim (10,10)
Inductive Reactance [pu/m]	dim (10,10)
Inductive Reactance [ohm/m]	dim (10,10)
Capacitive Reactance [pu*m]	dim (10,10)
Capacitive Reactance [Mohm*m]	dim (10,10)
Capacitive Susceptance [pu/m]	dim (10,10)
Travel Time (for 1 m) [s/m]	dim (10,10)
Surge Impedance [ohm]	dim (10,10)

General

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.

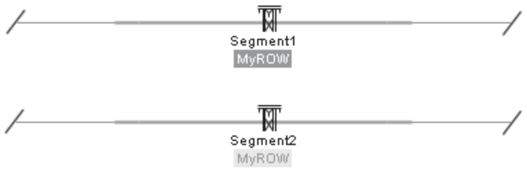

Resistance [pu/m] dim (10,10)

	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:

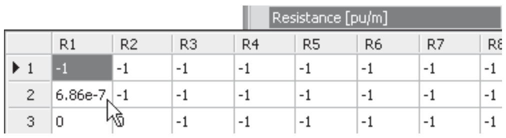
A user wishes to couple two, 3-phase transmission segments (called *Segment1* and *Segment2*), which are each defined by directly entered sequence data (using the *Manual Entry of Y,Z* component).



The local data from each segment forms two, individual *3x3* matrices each representing both ***Y*** and ***Z***. If we just concentrate on ***Z*** (***Y*** is treated in an identical manner), then the two segments will be represented by two unique ***Z*** matrices called ***Z1*** and ***Z2***. When two, 3-phase systems are coupled to form a single 6-phase system, ***Z1*** and ***Z2*** become the diagonals of the coupled *6x6* matrix as follows:

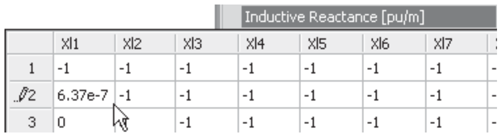
$$Z_{ROW} = \left|\begin{array}{c:c} Z_1 & Z_m \\ \hdashline Z_m & Z_2 \end{array}\right|$$

The matrix $\boldsymbol{Z_m}$ 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,Xl,Xc Data Entry (pu)* and so three separate values must be entered:



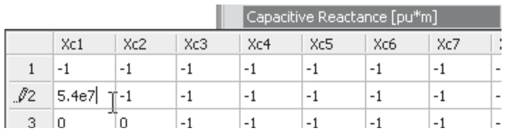
Resistance [pu/m]

	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	0	-1	-1	-1	-1	-1	-1



Inductive Reactance [pu/m]

	Xl1	Xl2	Xl3	Xl4	Xl5	Xl6	Xl7
1	-1	-1	-1	-1	-1	-1	-1
2	6.37e-7	-1	-1	-1	-1	-1	-1
3	0	0	-1	-1	-1	-1	-1



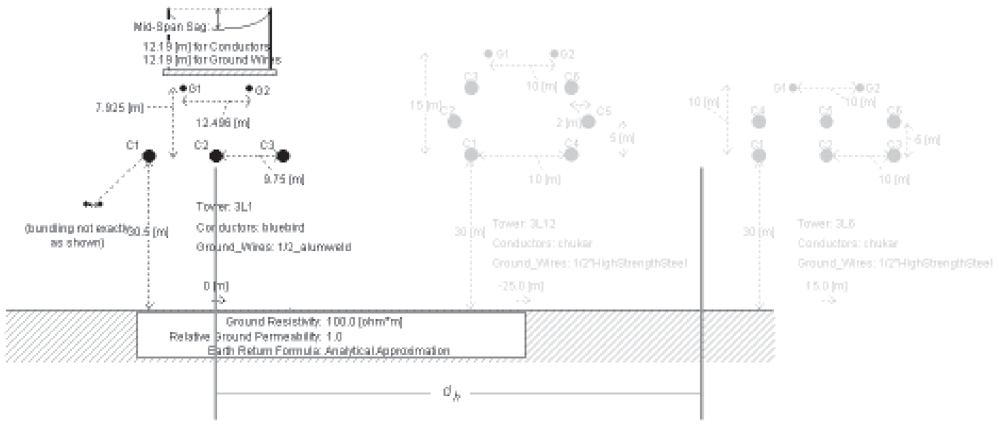
Capacitive Reactance [pu*m]

	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:


Mid-Span Sag:
12.19 [m] for Conductors
12.19 [m] for Ground Wires
7.925 [m]
G1
G2
12.496 [m]
C1
C2
C3
9.75 [m]
(bundling not exactly as shown)
30.5 [m]
Tower: 3L1
Conductors: bluebird
Ground_Wires: 1/2_alumweld
0 [m]
Ground Resistivity: 100.0 [ohm*m]
Relative Ground Permeability: 1.0
Earth Return Formula: Analytical Approximation
Tower: 3L12
Conductors: chukar
Ground_Wires: 1/2"HighStrengthSteel
-25.0 [m]
Tower: 3L6
Conductors: chukar
Ground_Wires: 1/2"HighStrengthSteel
15.0 [m]
10 [m]
15 [m]
2 [m]
5 [m]
30 [m]
6 [m]
d_h

Mutually Coupled ROW With Mutli-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.

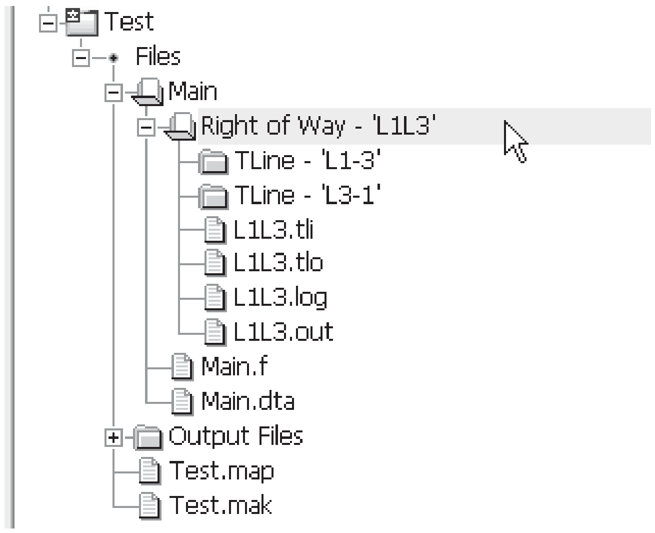
Remote End Mode Support

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* in Chapter 4 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 *Scoping 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:

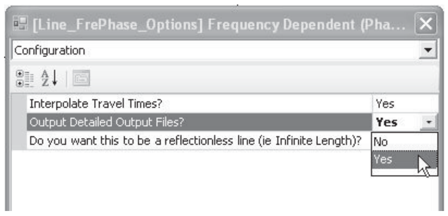
- Series Impedance
- Shunt Admittance
- Eigenvectors/Eigenvalues
- Curve Fitting Output

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** <u>before</u> the constants are solved.



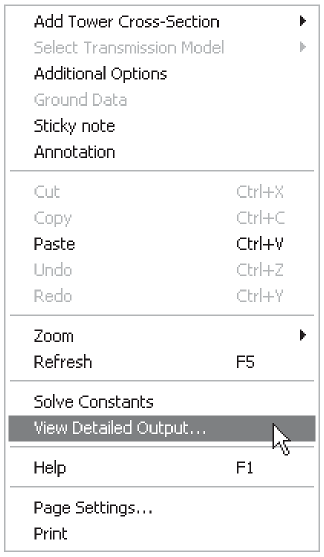
Interpolate Travel Times?	Yes
Output Detailed Output Files?	Yes
Do you want this to be a reflectionless line (ie Infinite Length)?	

Although mutually coupled ROW detailed output files can be generated, the Detailed Output Viewer does not currently support viewing of these files.

For more details on solving the constants, see the section entitled *Solving the Transmission Segment Constants Manually* in this chapter.

Invoking the Detailed Output Viewer

The Detailed Output Viewer can be opened from within the *Transmission Segment Definition Editor* pop-up menu.



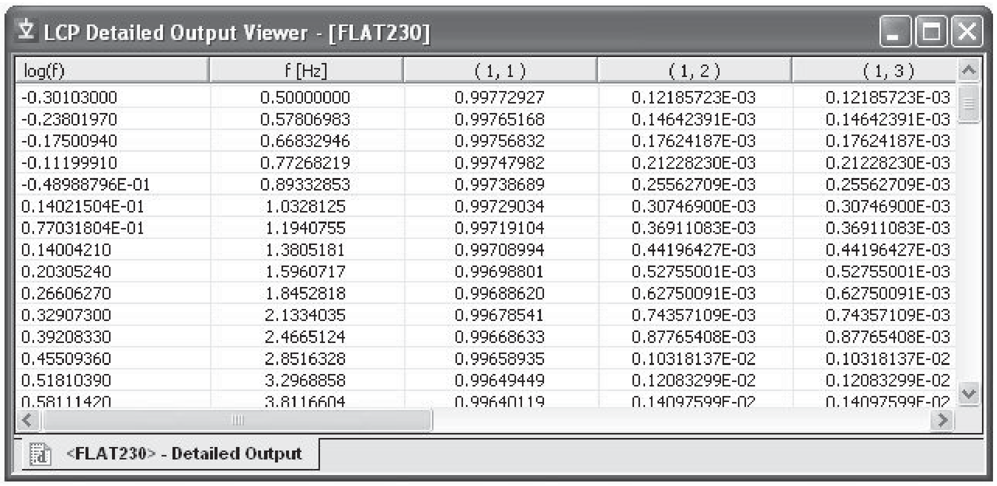
Detailed output is not available for the *Bergeron* model, as this model solves at only one frequency. See the section entitled *Segment Output (*.out) File* in this chapter for more on viewing Bergeron model output.

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]

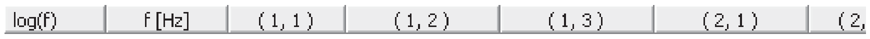
log(f)	f [Hz]	(1, 1)	(1, 2)	(1, 3)
-0.30103000	0.50000000	0.99772927	0.12185723E-03	0.12185723E-03
-0.23801970	0.57806983	0.99765168	0.14642391E-03	0.14642391E-03
-0.17500940	0.66832946	0.99756832	0.17624187E-03	0.17624187E-03
-0.11199910	0.77268219	0.99747982	0.21228230E-03	0.21228230E-03
-0.48988796E-01	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.

<u>Column Organization</u>
Depending on the output parameter being viewed, the data may exist as either matrix or array, or whether curve fitting results are included.

<u>Matrix:</u>

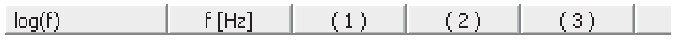


log(f)	f [Hz]	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2,

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.

<u>Array:</u>

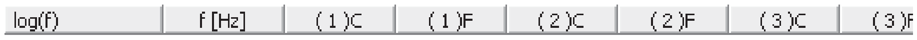


log(f)	f [Hz]	(1)	(2)	(3)	

etc...

Array quantities are simply listed in order.

<u>Calculated/Fitted Quantities:</u>



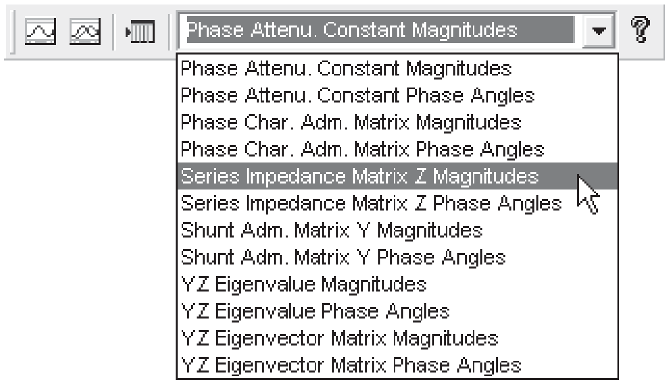
log(f)	f [Hz]	(1)C	(1)F	(2)C	(2)F	(3)C	(3)F

etc...

When curve fitting results are included, the calculated and fitted quantities are staggered, as indicated by the ‘C’ and ‘F’ respectively.

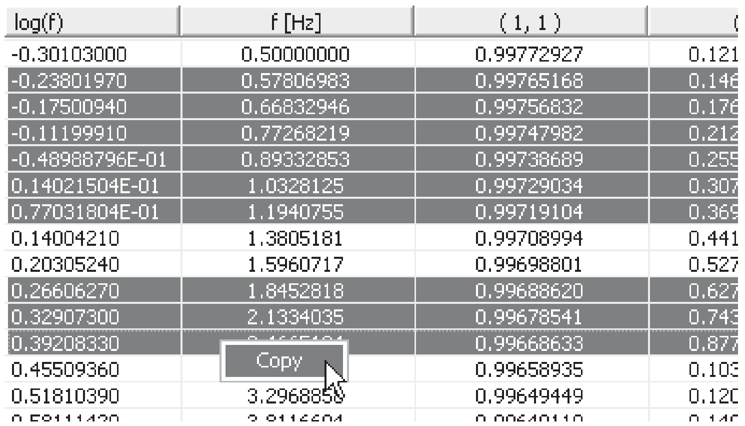
Switching between Spreadsheets

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:


Phase Attenu. Constant Magnitudes
Phase Attenu. Constant Magnitudes
Phase Attenu. Constant Phase Angles
Phase Char. Adm. Matrix Magnitudes
Phase Char. Adm. Matrix Phase Angles
Series Impedance Matrix Z Magnitudes
Series Impedance Matrix Z Phase Angles
Shunt Adm. Matrix Y Magnitudes
Shunt Adm. Matrix Y Phase Angles
YZ Eigenvalue Magnitudes
YZ Eigenvalue Phase Angles
YZ Eigenvector Matrix Magnitudes
YZ Eigenvector Matrix Phase Angles

Copying Data Rows

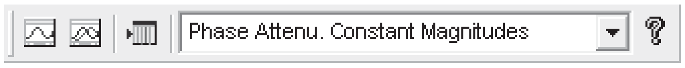
One or more rows of data may be copied to the Windows clipboard. Simply click on a single row, or hold down the *Ctrl* key and select multiple rows, then right-click on the selection and select *Copy*.



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	[illegible]	0.99668633	0.877
0.45509360	Copy	0.99658935	0.103
0.51810390	3.2968858	0.99649449	0.120
[illegible]	[illegible]	[illegible]	[illegible]

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

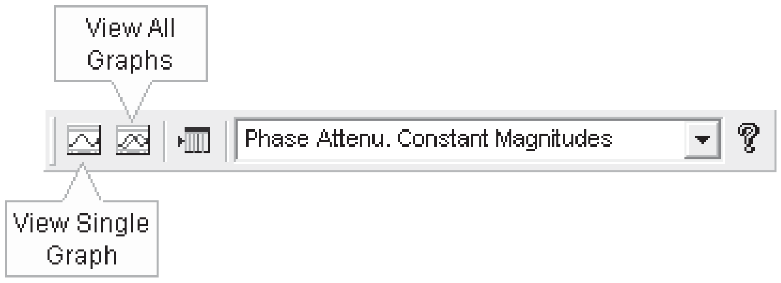
Curve Viewer

The Curve Viewer is a simple utility used 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.

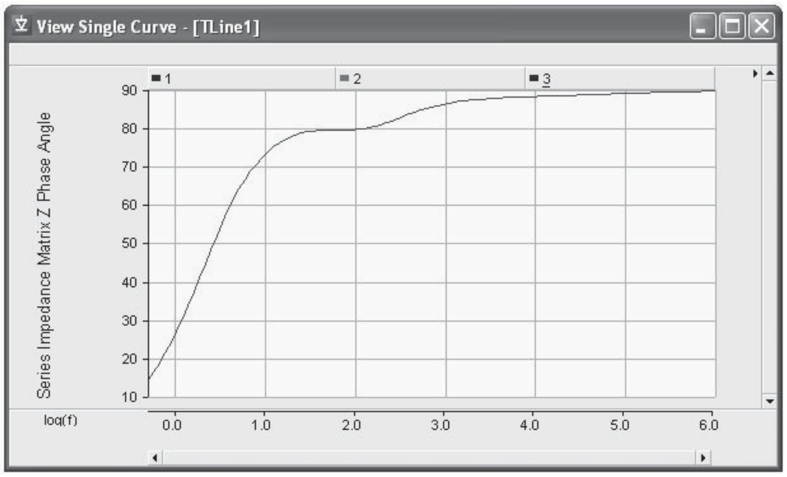
For detailed information on using these plotting tools, see Chapter 6 - *Online Plotting and Control*.

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:

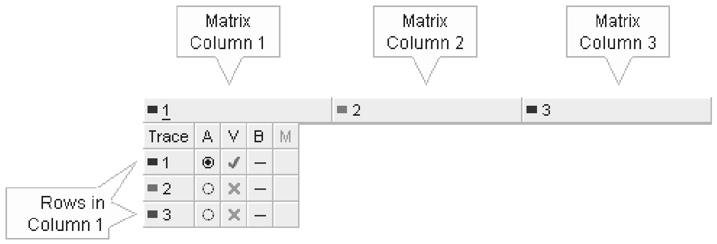


This will bring up the Curve Viewer as a floating window, as shown below for the Single Graph Viewer:



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.

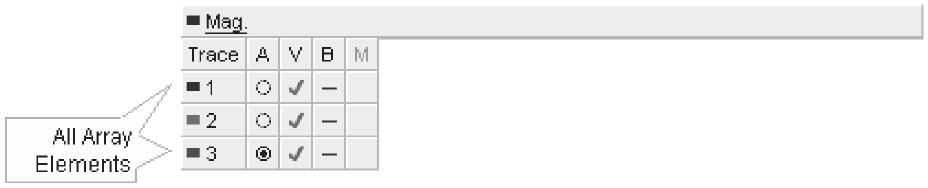
Element Identification
The Curve Viewer makes use of the multi-trace Curve feature available in the PSCAD online plotting tools, in order to allow plotting of 2-dimensional parameters, such as matrices. For example, if plotting a 3x3 matrix, such as the Y or Z matrix magnitudes, the matrix elements are 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 9:

Component Design

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.

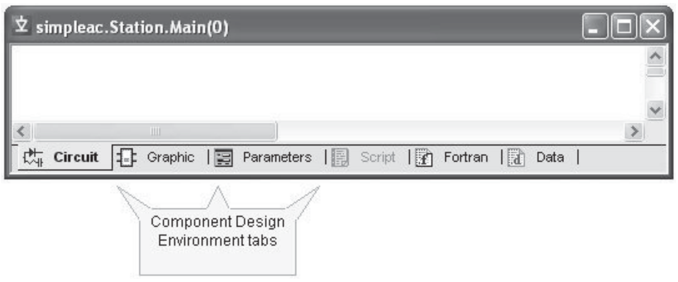
Custom components may be constructed in one of two ways: Either graphically by means of a module type component, or by direct coding. Regardless of the method used, in order for a custom model to be included in either the system dynamics or the electric network, a component must first be created to define 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 COMPONENT DESIGN ENVIRONMENT

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. Now, components are designed in a more integrated environment, which is no longer a separate utility. In fact, what is referred to as the *Component Design Environment* is incorporated as part of the main tabbed window.

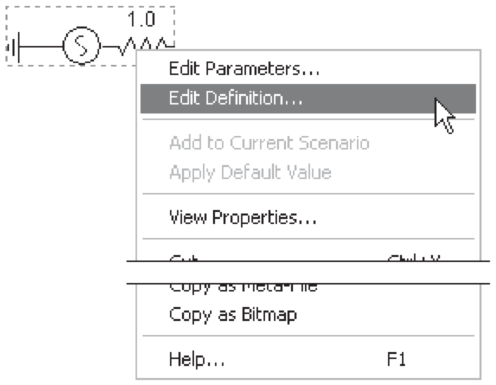


Four of the tabs represent the main sections of the design environment, and will become enabled depending on whether the user is editing the definition of a regular component or a module. For example, modules do not possess a *Script* section, and so therefore this tab is disabled while editing module definitions:

- **Circuit**: The *Circuit* window displays the schematic canvas of whatever module is being viewed (including the *Main* 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 *Circuit* 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 environment (i.e. editing the 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.

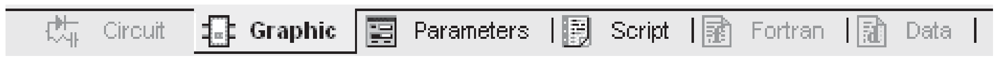


Other methods include:

- 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 Definition...**.

THE GRAPHIC SECTION

The component graphic is important as it represents your model visually on the Circuit canvas. In the Graphic section, component graphic objects, text labels and port connections can be added, removed and manipulated.



Navigation and Zoom

There are many 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

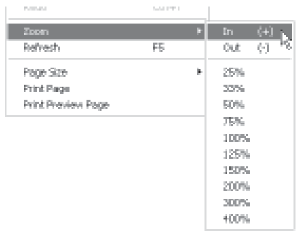
The panning or dynamic scroll feature allows you to scroll in a fluid motion. You can invoke panning mode by one of the following methods:

- 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 main toolbar 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 few different methods for zooming available:

- From the main menu, select **View | Zoom**. You then have a choice to select either **In**, **Out** or a specified zoom percentage.
- From the main toolbar, select either the **Zoom In**, **Zoom Out**, **Zoom Extents**, or **Zoom Rectangle** buttons, or select a percentage zoom directly from the **Zoom In/Out** drop down list.
- Right-click on a blank part of the Graphic canvas and select **View | Zoom**. You then have a choice to select either **In**, **Out** or a specified zoom percentage.



Cut, Copy, Paste and Delete Graphic Objects and Text

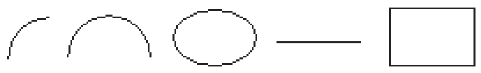
You can cut, copy or paste any graphic object or text label by using one of the following methods:

- Select the object with a left mouse click. Click the **Cut** or **Copy** buttons from the main toolbar. Click the **Paste** button main toolbar 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:

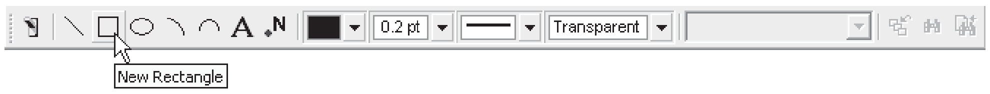
- ¼ Arc
- ½ Arc
- Ellipse
- Line
- Rectangle



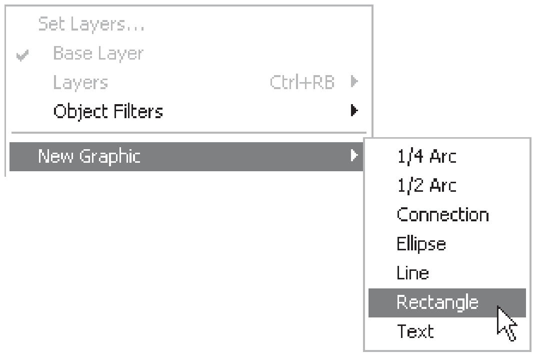
Object properties, such as colour, fill and size can be changed to suit the component graphical needs. A good graphical design can enhance user understanding of the purpose of the component and what its primary function is.

Adding New Graphic Objects

To add a graphic object to your component definition, the most straightforward method is to use the *Graphic Palette*:



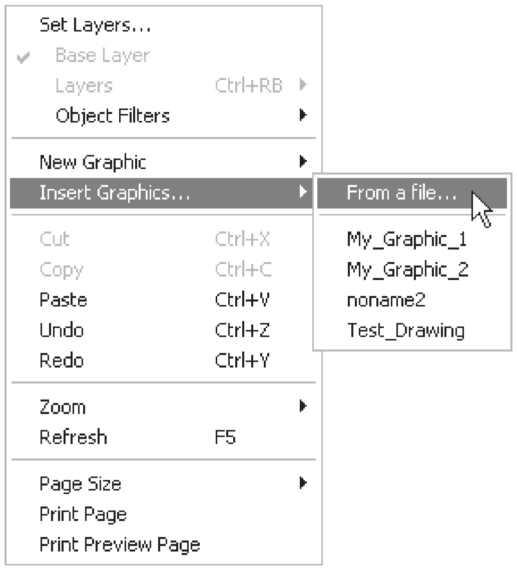
Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic window. Right-click and select **New Graphic**.



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:

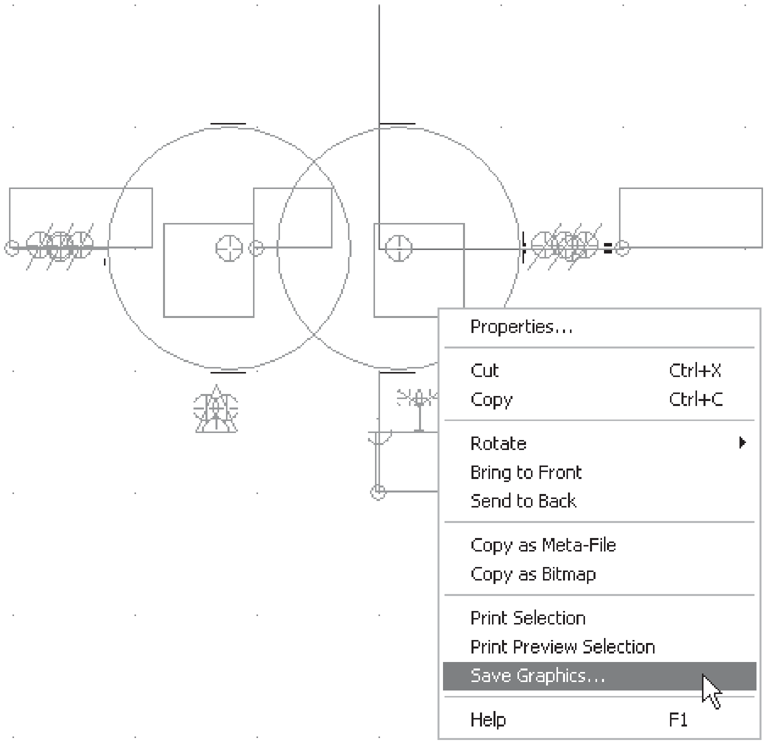
```
<svg id='noname' viewBox='-200 -200 200 200'>
  <port model="Natural" name="G1" x="-18" ... ></port>
  <line x1="6" y1="23" x2="3" y2="21" stroke="Black" ... />
  <ellipse cx="3" cy="0" rx="18" ry="18" stroke="Black" ... />
  <text x="6" y="6" stroke="Black" fill="Black" ... ></text>
</svg>
```

For more details on *Scalable Vector Graphics*, see http://en.wikipedia.org/wiki/Svg.

Saving Graphics to File

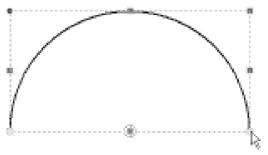
One or more graphic objects may be saved to a *Scalable Vector Graphics (*.svg)* file format. In this way, collections of selected graphic objects may be duplicated for transfer from one component to another.

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.



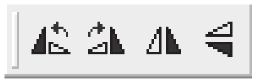
Rotate, Flip, Mirror and Resize Graphic Objects

Graphic objects can be rotated, flipped, mirrored or resized. To resize, left-click on the object so that grips appear.



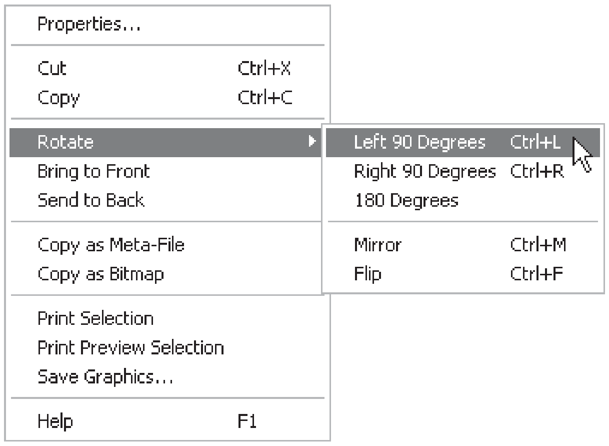
Place the mouse pointer over a selected grip, press and hold the left mouse button, and move the mouse. Note that the corner grips will allow resizing in both directions, while the mid-point grips will only allow movement in either the horizontal or vertical directions.

Rotate, flip or mirror can be accomplished in one of three ways. The first is to use the speed buttons on the Rotation Bar.



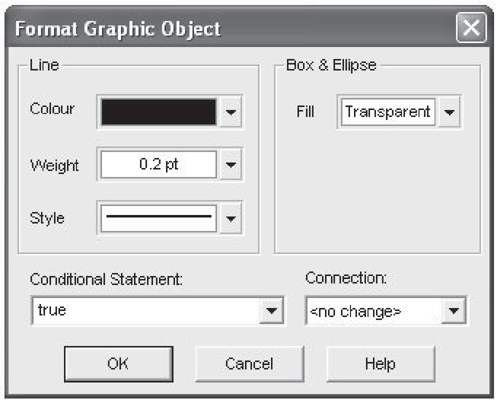
If you cannot see this tool bar, select **View | Rotation Bar** from the main menu. Select the object with a left-click of your mouse. Then, click one of the four buttons shown above.

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 toolbar, 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....**


Format Graphic Object

Line
Colour: ■
Weight: 0.2 pt
Style: ―――

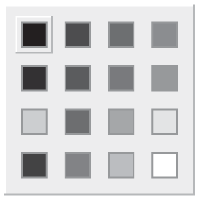
Box & Ellipse
Fill: Transparent

Conditional Statement: true
Connection: <no change>

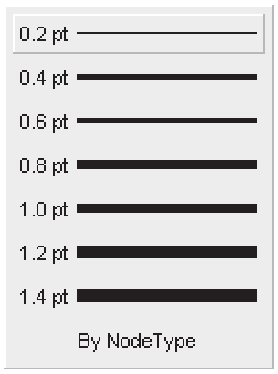
As shown above, graphical features, such as styles and colours, can be adjusted.

Line:

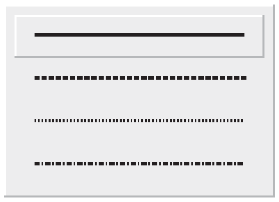
- **Colour**: Select the down arrow to bring up the colour palette shown below. Left-click on a colour button to change the colour of the object line. If the object is a *Rectangle* or *Ellipse*, this will change the border colour.



- **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.

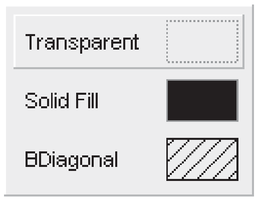


- **Style**: Select the down arrow to bring up the line style palette shown below. Left-click on a style button to change the style of the object line. If the object is a *Rectangle* or *Ellipse*, this will change the border style.

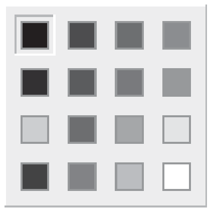


Rectangle & Ellipse:

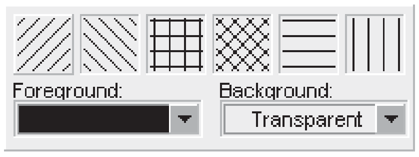
- **Fill**: Select the down arrow to bring up the fill palette shown below. With the mouse pointer over the **Transparent**, **Solid Fill** or **Pattern** buttons, left click the down arrow to bring up the respective sub-palettes shown below. This input is disabled for Line objects!



Fill Palette



Solid Fill Sub-Palette



Pattern Sub-Palette

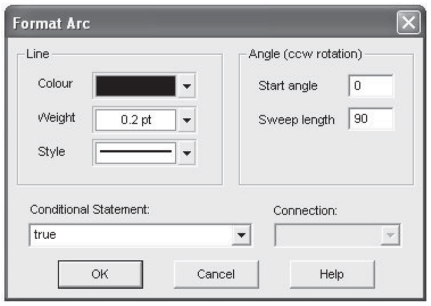
Other:

- **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 *Graphic Palette*. 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:

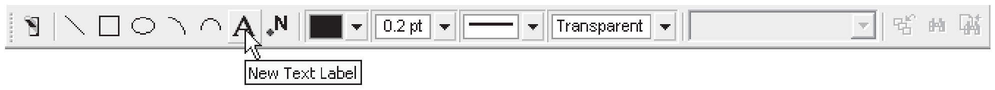
The remaining properties (i.e. **Conditional Statement** and **Connection**) function exactly as described for graphic objects. See *Changing Graphic Object Properties* above.

Text Labels

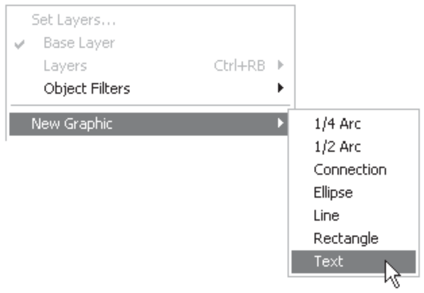
Text can be displayed on component graphics by using a *Text Label* object. Text labels can be one line of text only, and the user may select alignment and size.

Adding Text Labels

To add a *Text Label* to your component definition, the most straightforward method is to use the Graphic Palette:



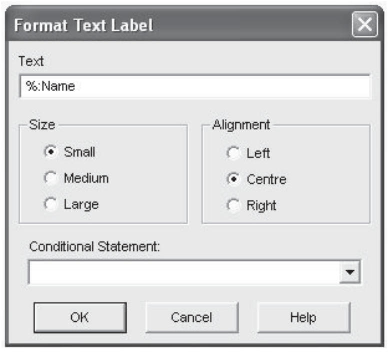
Simply left-click the **New Text Label** button, drag the label to where you want it placed and left-click again. Another method is to use the right-click menu: Move the mouse pointer over a blank area of the Graphic window. Right-click and select **New Graphic | Text**.



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. To change the *Text Label* properties: Either left double-click or right-click over the label and select **Properties....**



Format Text Label

Text: %:Name

Size: (● Small) ○ Medium ○ Large

Alignment: ○ Left (● Centre) ○ Right

Conditional Statement:

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 wil be visible. See *Conditional Statements, Layers & Filters* for more details.

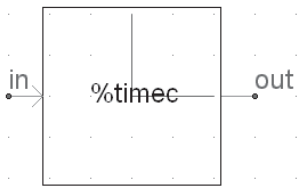
Linking a Text Label to an Input Field

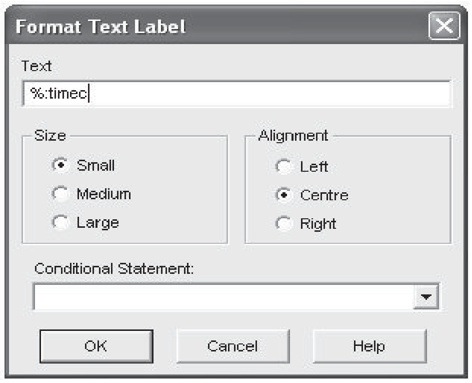
It is possible to link *Text Label* text to a specific input parameter within the same definition. Once linked, the label can be used to graphically display the parameter value (and unit). For example, you can display the MVA rating of a transformer model according to what is entered in its *Rated MVA* 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:




Format Text Label

Text: %:timec

Size: [✓ Small] Medium Large

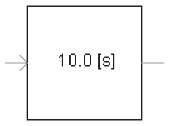
Alignment: Left [✓ Centre] Right

Conditional Statement:

Component Graphic Section Text Label Property Settings

If the % prefix is replaced by the $ prefix, the units will not be displayed.

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:

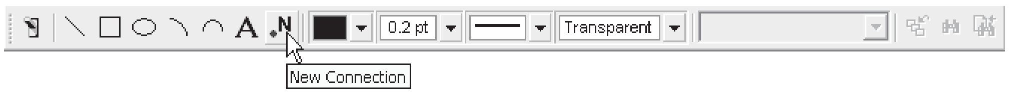


Port Connections

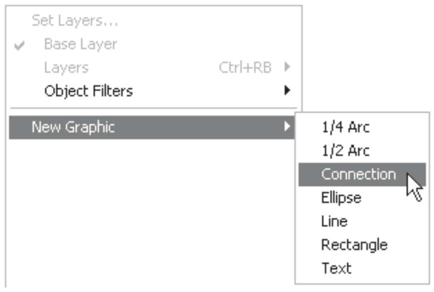
Port Connections are used to provide graphical access to signals defined within the Circuit canvas where the component instance resides. They provide a means to either read signals from, or output signals to the external system each simulation time step. These ports play an essential part in the construction 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 Graphic Palette:



Simply left-click the **New Connection** button on the toolbar, 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**.

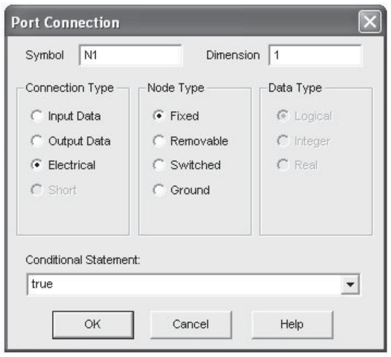


You may notice that port connections are always snapped to the Graphic canvas drawing grid. This is to ensure that when other components are connected to yours, their respective connection points will overlap.

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.

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...**.


Port Connection

Symbol: N1
Dimension: 1

Connection Type: ( ) Input Data ( ) Output Data [✓ Electrical] ( ) Short

Node Type: [✓ Fixed] ( ) Removable ( ) Switched ( ) Ground

Data Type: [✓ Logical] ( ) Integer ( ) Real

Conditional Statement: true

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 is 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**.
- **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.

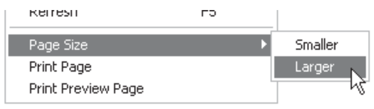
Only **Fixed**-type electrical ports are allowed in module component definitions.

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**.

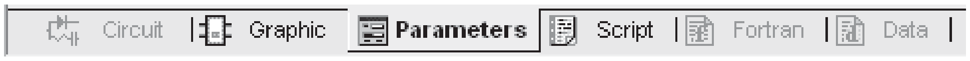


The default canvas size is **Smaller**.

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:



When you first create a new component, a default category called *General* 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, specifically for the identification of module components with multiple instances. See *The Secondary Window* in Chapter 4 of this manual for details.

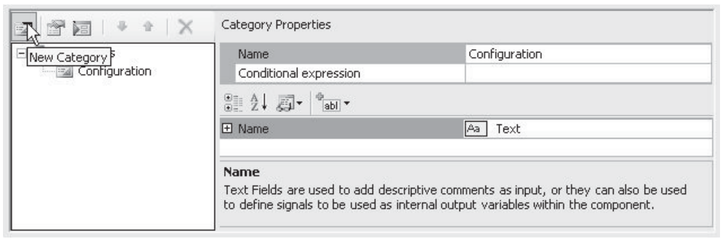
Categories

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 four types of input parameter fields that may be added categories. These include:

- Choice List
- Integer
- Real
- Text
- Table
- Toggle

Adding a New Category

To add a new category page to your component definition, simply click the **New Category** button:



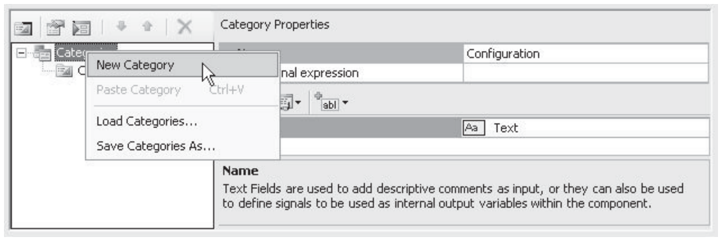
Category Properties

Name	Configuration
Conditional expression	

⊞ Name	Aa Text

Name
Text Fields are used to add descriptive comments as input, or they can also be used to define signals to be used as internal output variables within the component.

Or, right-click on the Categories tree and select **New Category**.



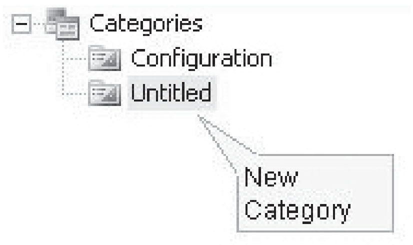
Category Properties

	Configuration
nal expression	

	Aa Text

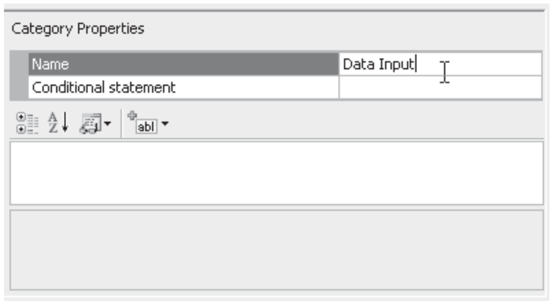
Name
Text Fields are used to add descriptive comments as input, or they can also be used to define signals to be used as internal output variables within the component.

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.

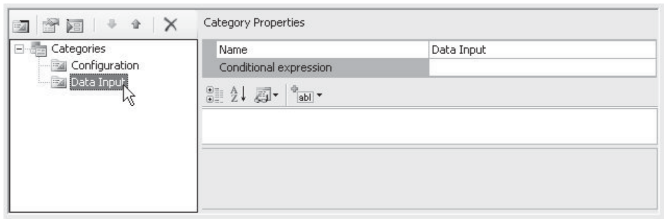


Name	Data Input
Conditional statement	

- **Name**: Enter a descriptive name in 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

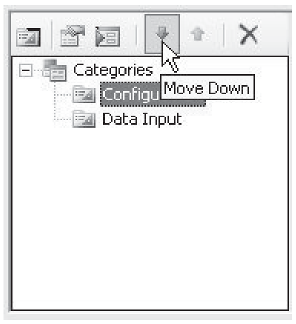
Once you have added more than one category, there will of course be a need to navigate through these pages. In the category tree, left-click on the desired category page to view it.



Name	Data Input
Conditional expression	

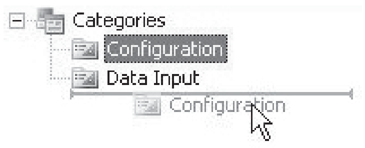
Ordering Categories

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:



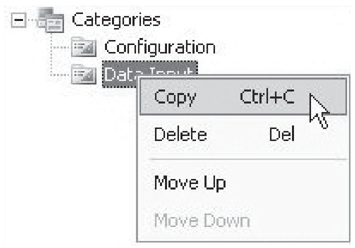
In the above diagram, the *Configuration* category page was moved below the *Data Input* page.

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.

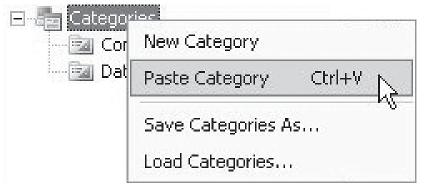


Duplicating a Category

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**).



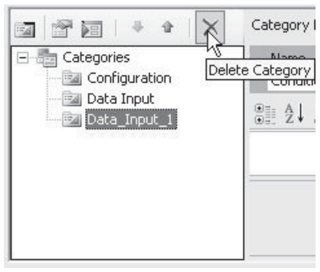
Copy



Paste

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.

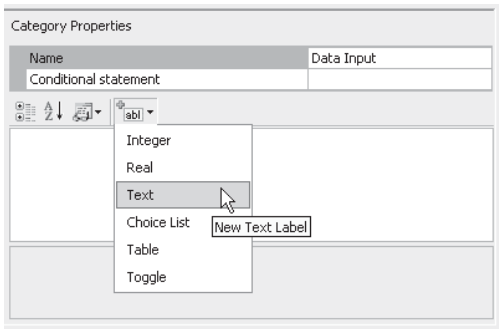
Deleting a Category
To delete a particular category, select it in the Categories tree and then click the **Delete Category** button in the toolbar (or press the **Delete** key).



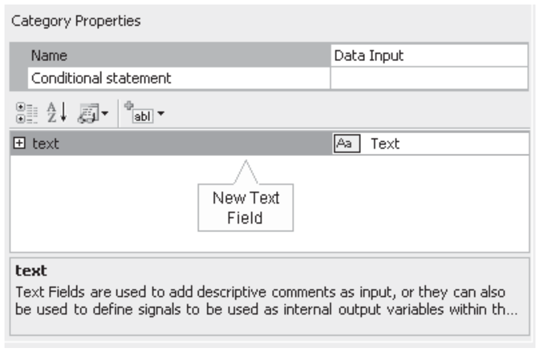
You may also right-click on the category page itself and select **Delete**.

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:



A new text field will appear:


Category Properties

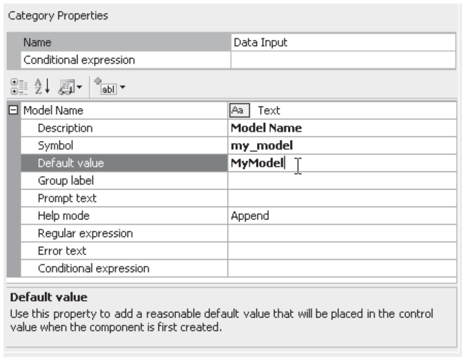
Name	Data Input
Conditional statement	

⊞ text	Text

text
Text Fields are used to add descriptive comments as input, or they can also be used to define signals to be used as internal output variables within th...

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

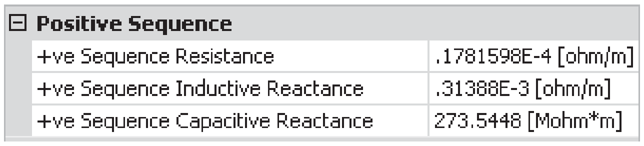
Name	Data Input
Conditional expression	

⊟ Model Name	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 next 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*):



Positive Sequence	
+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.

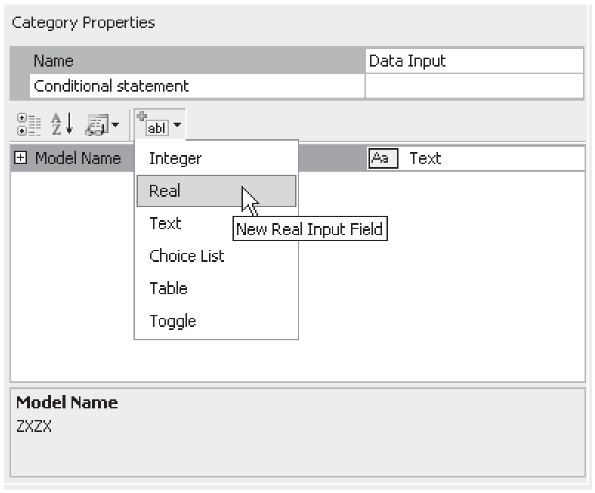
Input Fields

Input fields allow the user to import numerical or signal input into the component. Input fields include the following numerical sub-types:

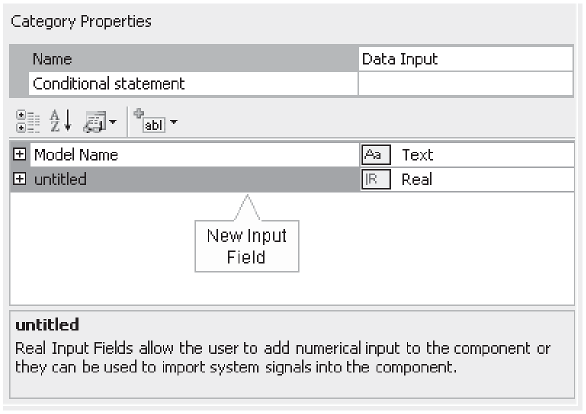
- Real
- Integer

Adding Input Fields

To add a new input field to a category page, select the category in the Categories tree and then select **Real** or **Integer** within the **Add Parameter Field** drop list button:



A new input field will appear:


Category Properties

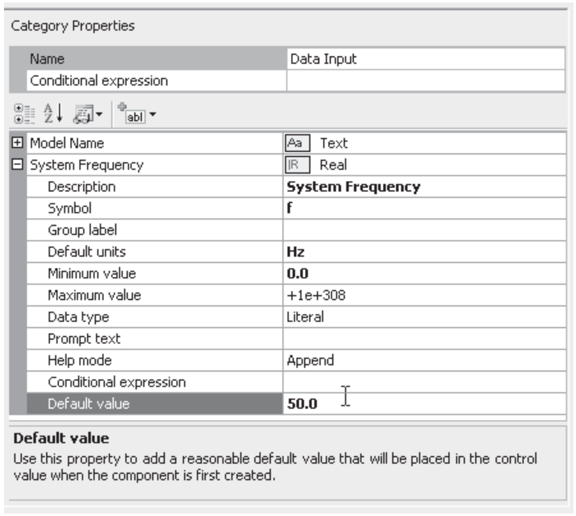
Name	Data Input
Conditional statement	

⊞ Model Name	Text
⊞ untitled	Real

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 Input Field Properties

Input field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the input field tree node.


Category Properties

Name	Data Input
Conditional expression	

⊞ Model Name	Text
⊟ System Frequency	Real
Description	**System Frequency**
Symbol	**f**
Group label	
Default units	**Hz**
Minimum value	**0.0**
Maximum value	+1e+308
Data type	Literal
Prompt text	
Help mode	Append
Conditional expression	
Default value	**50.0**

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 input field.
- **Symbol**: Enter a symbolic name for the input 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 a value outside this range is entered, PSCAD will provide a warning in the *Output Window* upon compilation.
- **Data Type**: Select **Literal**, **Constant** or **Variable**. The selection of data type is very important. Please see the following section entitled *Input Field Data Types* for a complete description of each.
- **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 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.

Input Field Data Types

Input 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, both Real and Integer type input fields can be further categorized into three sub-types: *Literal*, *Constant* and *Variable*.

PSCAD versions prior to X4 allowed only type *Literal* and *Variable* input fields. Seasoned users will remember the use of the *Allow Signal Names* option in the Input Field dialog: By default, all input 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 input 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 input 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 input fields accept both numeric and non-numeric input. Input values may remain fixed, or they can vary throughout the simulation. This input type is equivalent to having the *Allow Signal Names* option enabled in PSCAD versions previous to X4.

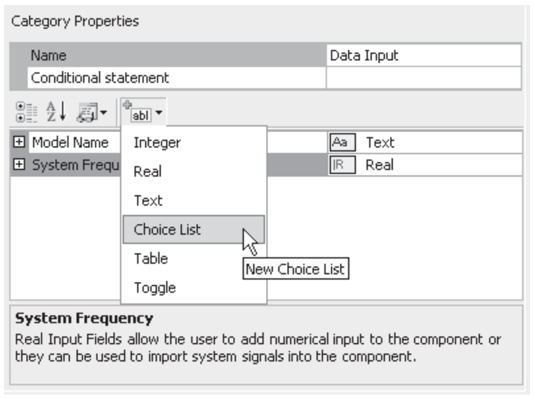
See *Multiple Instance Modules* in Chapter 5 for more details on using Constant type input fields.

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.

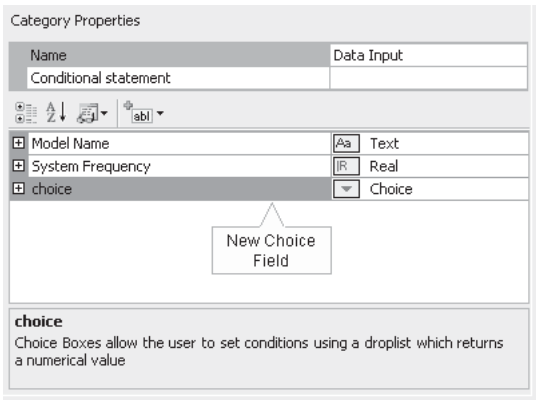
Adding Choice Lists

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 toolbar:



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.

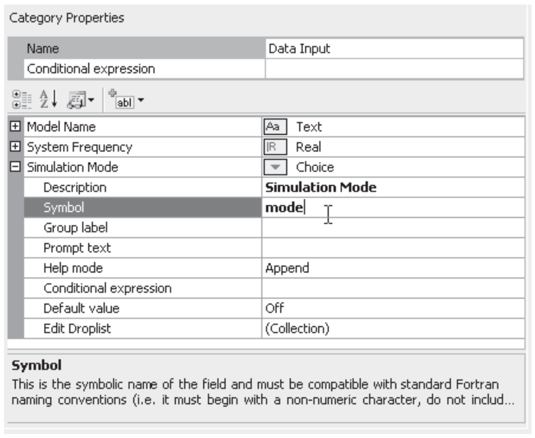
A new input field will appear.



choice
Choice Boxes allow the user to set conditions using a droplist which returns a numerical value

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.


Category Properties

Name	Data Input
Conditional expression	

⊞ Model Name	Text
⊞ System Frequency	Real
⊟ Simulation Mode	Choice
Description	**Simulation Mode**
Symbol	**mode**
Group label	
Prompt text	
Help mode	Append
Conditional expression	
Default value	Off
Edit Droplist	(Collection)

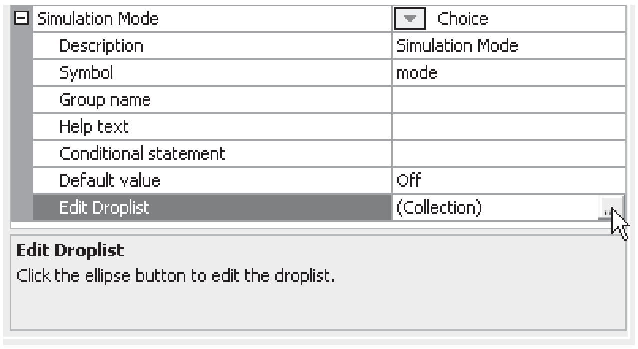
Symbol
This is the symbolic name of the field and must be compatible with standard Fortran naming conventions (i.e. it must begin with a non-numeric character, do not includ...

The properties available are described as follows:

- **Description**: Enter a caption to act as the visible title of the 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

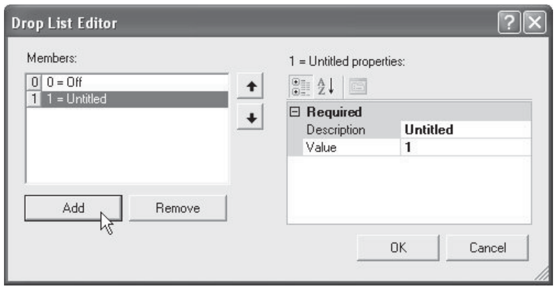
Choice lists are constructed using the *Drop List Editor*. To open the editor, select the **Edit Drop List** field and then left-click the [...] button.



Simulation Mode	Choice
Description	Simulation Mode
Symbol	mode
Group name	
Help text	
Conditional statement	
Default value	Off
Edit Droplist	(Collection)

Edit Droplist
Click the ellipse button to edit the droplist.

The Drop List Editor will appear. To add choices to the list, click the **Add** button.

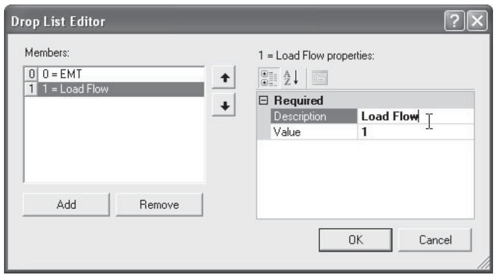


Members:
0 0 = Off
1 1 = Untitled

1 = Untitled properties:

Required	
Description	Untitled
Value	1

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.



Members:
0 0 = EMT
1 1 = Load Flow

1 = Load Flow properties:

Required	
Description	Load Flow
Value	1

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, depending on the setting.

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

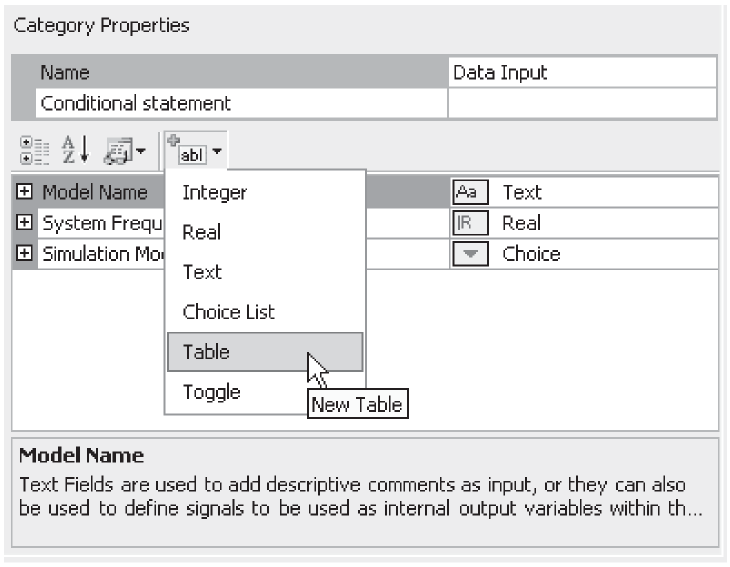
Table fields can currently be used only in standard, non-module components.

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

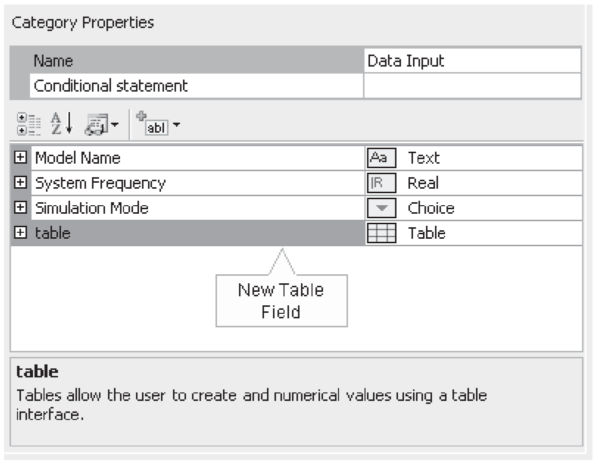
Adding Table Fields

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:



Model Name
Text Fields are used to add descriptive comments as input, or they can also be used to define signals to be used as internal output variables within th...

A new table field will appear:



Category Properties

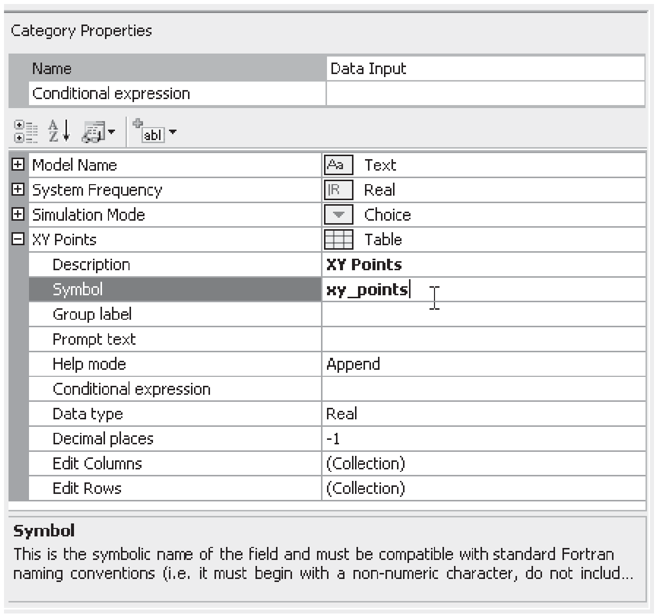
Name	Data Input
Conditional statement	

⊞ Model Name	Text
⊞ System Frequency	Real
⊞ Simulation Mode	Choice
⊞ table	Table

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	

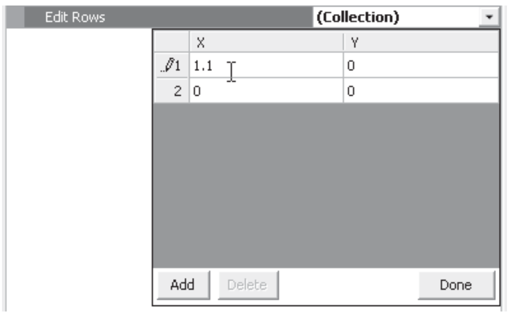
⊞ Model Name	Text
⊞ System Frequency	Real
⊞ Simulation Mode	Choice
⊟ XY Points	Table
Description	**XY Points**
Symbol	**xy_points**
Group label	
Prompt text	
Help mode	Append
Conditional expression	
Data type	Real
Decimal places	-1
Edit Columns	(Collection)
Edit Rows	(Collection)

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.



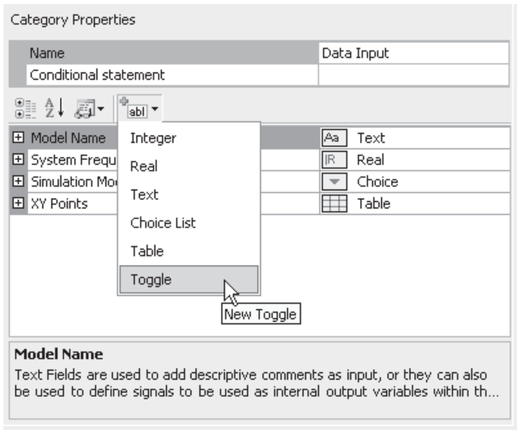
	X	Y
1	1.1	0
2	0	0

Toggle Fields

Toggle fields are Boolean parameters designed for simple on/off or yes/no parameters.

Adding Toggle Fields

To add a new toggle field to a category page, select the category in the Categories tree and then select **Toggle** within the **Add Parameter Field** drop list button:

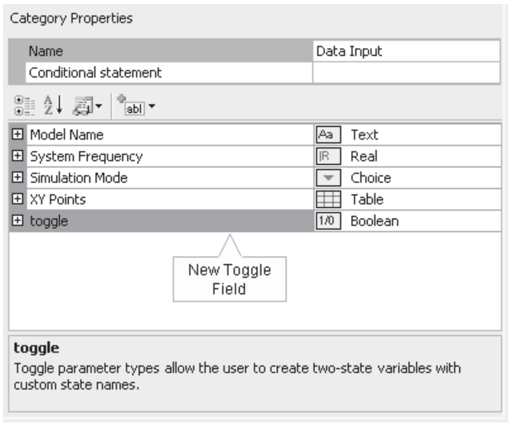


Category Properties

Name	Data Input
Conditional statement	

Model Name
Text Fields are used to add descriptive comments as input, or they can also be used to define signals to be used as internal output variables within th...

A new toggle field will appear:



Category Properties

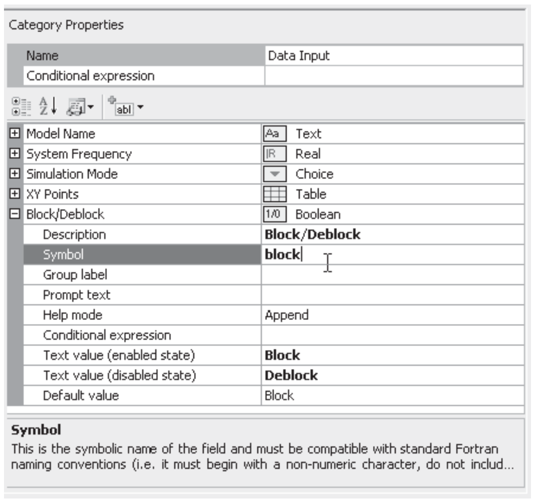
Name	Data Input
Conditional statement	

Model Name	Text
System Frequency	Real
Simulation Mode	Choice
XY Points	Table
toggle	Boolean

toggle
Toggle parameter types allow the user to create two-state variables with custom state names.

Changing Toggle Field Properties

Toggle field properties can be adjusted directly on the category page. Left-click on the [+] box to expand the toggle field tree node.



Category Properties

Name	Data Input
Conditional expression	

⊞ Model Name	Text
⊞ System Frequency	Real
⊞ Simulation Mode	Choice
⊞ XY Points	Table
⊟ Block/Deblock	Boolean
Description	**Block/Deblock**
Symbol	**block**
Group label	
Prompt text	
Help mode	Append
Conditional expression	
Text value (enabled state)	**Block**
Text value (disabled state)	**Deblock**
Default value	Block

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...

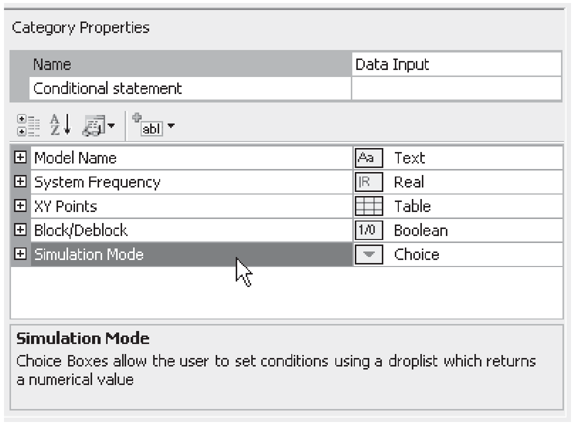
The properties available are described as follows:

- **Description**: Enter a caption to act as the visible title of the 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.
- **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 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.


Category Properties

Name	Data Input
Conditional statement	

Model Name	Text
System Frequency	Real
XY Points	Table
Block/Deblock	Boolean
Simulation Mode	Choice

Simulation Mode
Choice Boxes allow the user to set conditions using a droplist which returns a numerical value

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, LAYERS 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

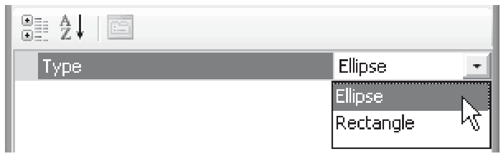
The primary purpose of conditional statements is to either enable or disable an object (such as an input field), or to make an object visible or invisible (such as a graphic object).

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.

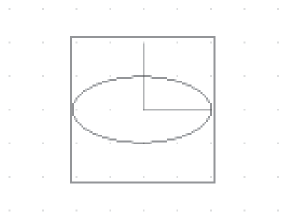
EXAMPLE 9-2:

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*.

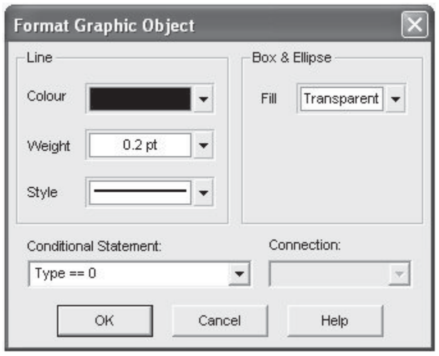
In the component Parameters section, the designer adds a single choice list:



In the component Graphic section, the designer draws a simple ellipse and a simple rectangle:

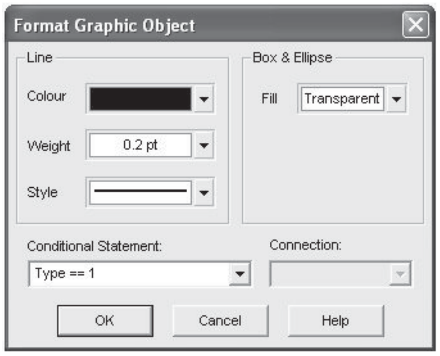


In the properties dialog for the graphic objects, the following text is added to the respective **Conditional Statement** input fields:



Colour
Weight 0.2 pt
Style
Fill Transparent
Conditional Statement: Type == 0
Connection:

Ellipse Property Settings



Colour
Weight 0.2 pt
Style
Fill Transparent
Conditional Statement: Type == 1
Connection:

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.

Care must be exercised when using the divide (/) function, as a *divide by zero* error can occur.

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 layer will be created, even though the statements may indicate the same thing.

Layers

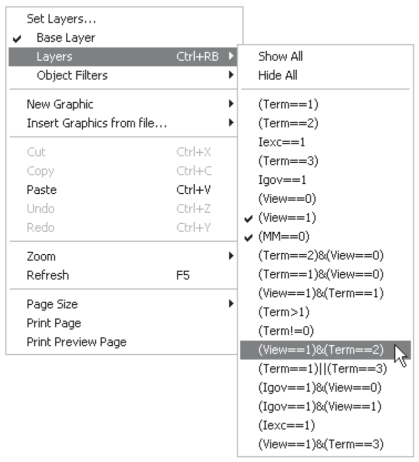
As component graphics become larger and more complicated, the graphical working environment can become quite unruly – especially when many conditional statements are used.

Fortunately, there is a way to avoid the potential clutter by utilizing the graphical layers available in the Graphic section. Layers are based solely on conditional statements, in that whenever a unique conditional statement is entered into an object or field, a new graphical layer is created. Any other graphic object, which utilizes an identical conditional statement, will also be visible in that particular layer.

Viewing Layers

By default, only the layers 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 layers 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 layers will be based on the component default conditions.

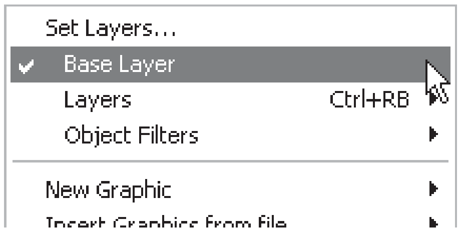
To adjust the layer visibility, move the mouse pointer over a blank area of the Graphic canvas. Right-click and select **Layers**. The resulting sub-menu will list all of the available layers (or conditional statements) that currently exist in the component graphics.



Simply select or de-select each layer to make visible or invisible respectively. You can also select **Show All** or **Hide All** to toggle the visibility of all layers.

Base Layer

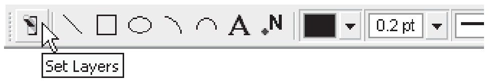
The base layer contains any graphical objects and port connections that do not possess a defined conditional statement (i.e. are not part of a defined layer). The base layer may be toggled on and off using the pop-up menu: Move the mouse pointer over a blank area of the Graphic canvas. Right-click and select **Base Layer**.



Setting Layers

Instead of turning layers on and off through the right-click menu (which can become cumbersome as the number of layers increases), you can set and test the visible layers directly using the component parameters dialog.

Right-click on a blank part of the canvas and select **Set Layers...** from the pop-up menu. Or, press the **Set Layers** button on the Graphic Palette:

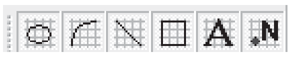


This feature provides a fully functional preview of the parameter dialogs as they would be seen by a user. Simply set all the choice lists to the desired settings and click the **OK** button – the graphical layers pertaining to that setting will become visible in the Graphic canvas.

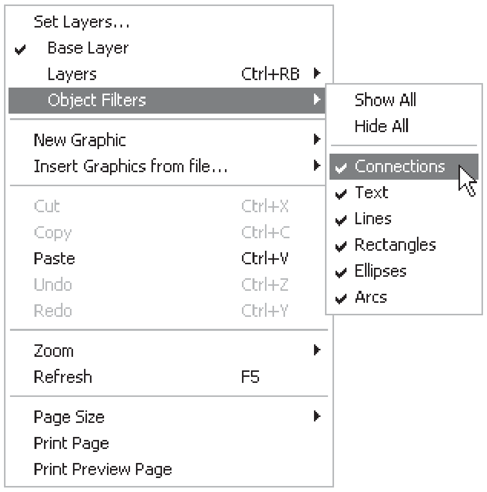
Object Filters

In addition to graphical layers, you may also use **Object Filters** to help alleviate graphical clutter. Object filters allow the user to view objects based on the graphic object type (i.e. port connections, text labels, etc.).

To adjust Object Filters, the most straightforward method is to use the Object Filter 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 **Object 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.

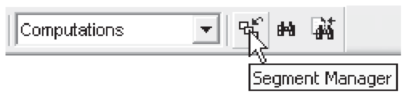
The segments that your component definition will require depend on what function your component is to perform. Are pre-calculations required? Is it an electrical component? Is there source code involved? The following sections describe each available script segment in detail.

Segments

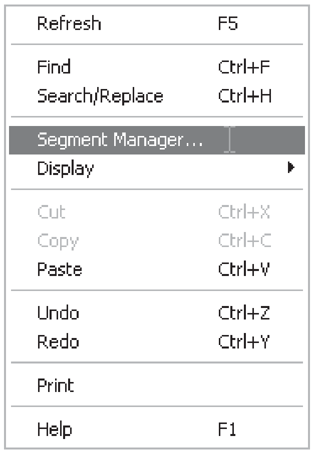
Segments are ordered similar to pages in a book. Each segment is itself a simple text editor, where data, script and other code is added to perform a specific task. Segments may be added or deleted at will, but must follow a strict naming convention.

Managing Segments

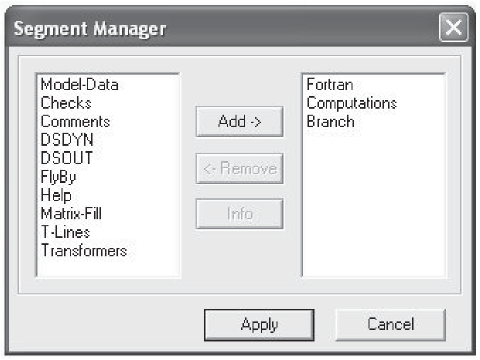
The *Segment Manager* enables users to add or remove segments from within a simple dialog. To invoke the segment manager, the most straightforward method is to use the script bar:



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:

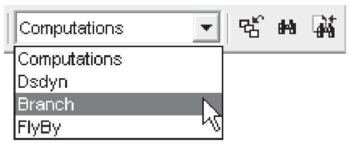


The list on the left-side of the dialog displays the segments available to add to this definition, while the right-side lists are the segments already in the definition. To add or remove segments to/from the definition, simply select the **Add->** or **<-Remove** buttons. Once satisfied with the selection, click the **OK** button. If you chose to remove any existing segments, a dialog will appear asking you to confirm the removal. Press the **Cancel** button to cancel the operation.

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 on the script bar to navigate segments.



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.

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 segments.

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.

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.

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.

A typical Computations Segment entry must have the following standard format:

```
<DataType> <Name> = <Expression>
```

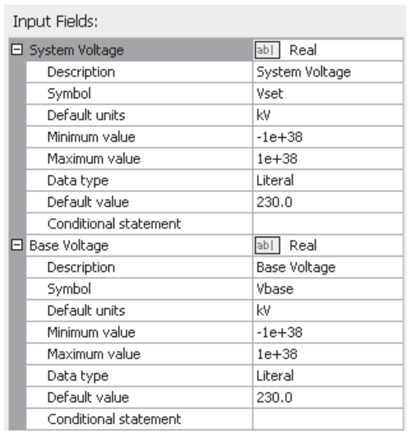
Where,

- <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.



Input Fields:

⊟ System Voltage	Real
Description	System Voltage
Symbol	Vset
Default units	kV
Minimum value	-1e+38
Maximum value	1e+38
Data type	Literal
Default value	230.0
Conditional statement	
⊟ Base Voltage	Real
Description	Base Voltage
Symbol	Vbase
Default units	kV
Minimum value	-1e+38
Maximum value	1e+38
Data type	Literal
Default value	230.0
Conditional statement	

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 of 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>]
```

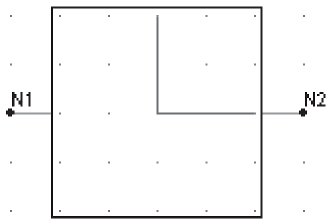
Where,

If a branch does not contain all three types of elements, simply substitute *0.0* for the value of the element that is not present.

- <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 [Ω].
- <L> is the inductance of the branch (optional) [H].
- <C> is the capacitance of the branch (optional) [μF].
- <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* ($).

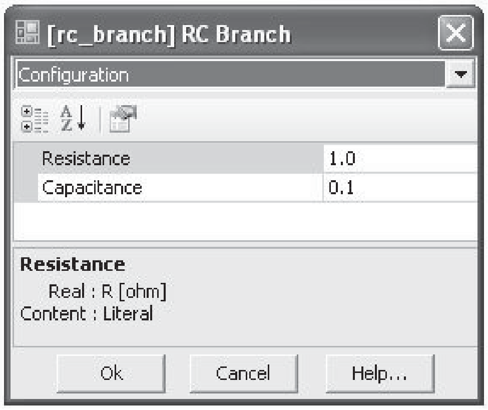
EXAMPLE 9-4:

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 through 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.



[rc_branch] RC Branch

Configuration

Resistance	1.0
Capacitance	0.1

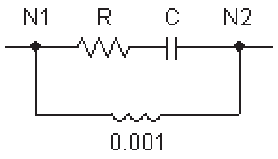
Resistance
Real : R [ohm]
Content : Literal

Component Parameters Dialog

The Branch segment would then contain the following two statements:

```
$N1  $N2  $R   0.0    $C
$N1  $N2  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*

If *BRN1* and *BRN2* are used in other segments, they must be preceded by the *Substitution Prefix Operator ($)*.

and *BRN2* are defined. The designer modifies the above branch statements as follows:

```
BRN1 = $N1  $N2   $R   0.0    $C
BRN2 = $N1  $N2   0.0  0.001  0.0
```

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>]
```

For more examples on this, edit the definition of the *Voltage Source Model 2* in the master library project.

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.

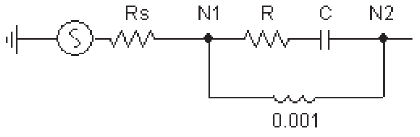
EXAMPLE 9-5:

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 = $GND $N1  SOURCE  $RS  0.0    0.0
BRN1 = $N1  $N2          $R   0.0    $C
BRN2 = $N1  $N2          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 [Ω]. 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 *VT*, 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:

$$v(t) = V_{peak} \cdot \sin(2\pi ft + \varphi)$$

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 *2πft* function should never be allowed to simply grow larger and larger as the simulation progresses. It should be reset to zero whenever *2πft* = 2π is reached.

There are pros and cons to adding inline 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 simply be called from the Fortran segment. Also, if Fortran line numbering is used in your code, these same numbers 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 *Output Window* 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 Output Window. 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*, then all that is required is 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 *Chapter 3 - Workspace and System Settings*.

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

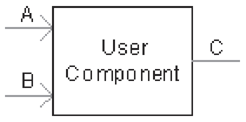
The FlyBy segment provides an avenue for the designer to apply fly-by (or pop-up) help to a component. Fly-by windows can help to provide users with a quick description of the model itself, or even individual port connections on the component.

The FlyBy section 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:

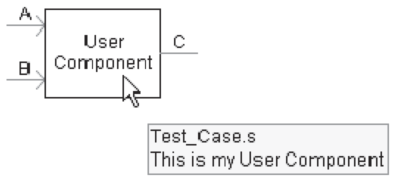


In addition to a general description of the component, the designer wants to provide a fly-by for each of the input port connections *A* and *B*.

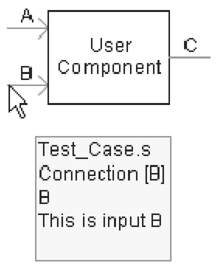
The FlyBy section should then contain something similar to the following:

```
This is my User Component
:A
This is input A
:B
This is input B
```

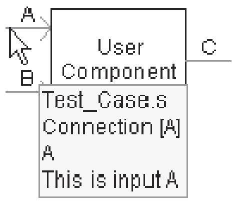
If the mouse pointer is now moved over the component while on the Circuit canvas, the following fly-by windows will appear:



Test_Case.s
This is my User Component



Test_Case.s
Connection [B]
B
This is input B



Test_Case.s
Connection [A]
A
This is input A

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 three main parts:

- **Number of transformers (matrices)**: This is accomplished through the use of a *#TRANSFORMERS* directive. See *Chapter 10 - Definition Script* for more details on this directive.
- **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>
<Prefix><Number_of_Windings> /
<Node_1> <Node_2>   <R_11> <L_11> /
<Node_2> <Node_3>   <R_12> <L_12>  <R_22> <L_22> /
...
```

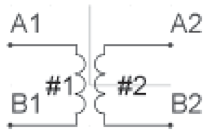
The Transformers segment format is different when defining a UMEC type transformer. For more information on the UMEC format, see the UMEC transformer component definitions in the master library, or contact the PSCAD Support Desk (support@pscad.com).

<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 segments have 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*.

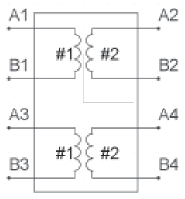
The Transformers segment should then appear similar to the following:

```
#TRANSFORMERS 1
!
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
!
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.

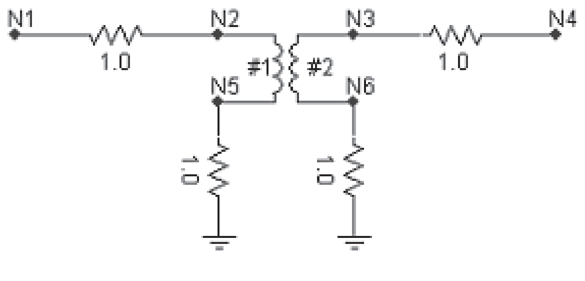
The general format for a Matrix-Fill Segment statement would appear as follows:

```
<Node_1> <Node_2> <Node_3> <Node_4> ...
```

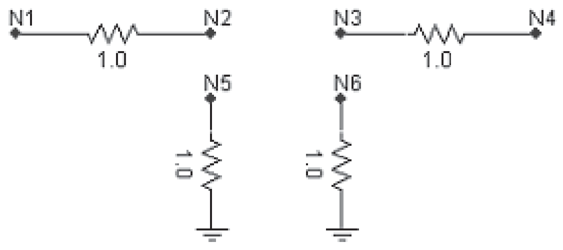
Where <Node_#> is the Symbol name of the port connection involved. All port connections in a single statement are assumed to be connected to each other.

EXAMPLE 9-11:

A user creates a single-phase electrical circuit in PSCAD, which includes a classical transformer component connected as follows:



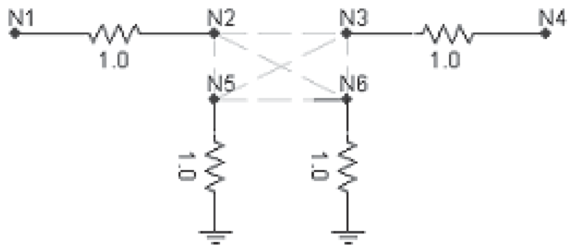
As can be seen above, there are six electrical nodes in this system, and therefore the system matrix will have dimension 6 x 6. Connections internal to the transformer component cannot be determined initially by PSCAD, and so when a logical matrix for the Optimize Node Ordering algorithm is created, 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:

$$\begin{bmatrix} X & X & 0 & 0 & 0 & 0 \\ X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 \\ 0 & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & 0 & 0 & X \end{bmatrix}$$

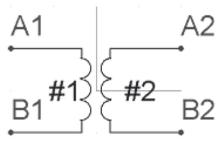
The matrix above does not contain all of the connection information required however. We know through the Transformers segment in the transformer component, that nodes *N2*, *N3*, *N5* and *N6* are all part of a mutual impedance matrix and hence are all connected together, as shown below:



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:

$$\begin{bmatrix} X & X & 0 & 0 & 0 & 0 \\ X & X & + & 0 & + & + \\ 0 & + & X & X & + & + \\ 0 & 0 & X & X & 0 & 0 \\ 0 & + & + & 0 & X & + \\ 0 & + & + & 0 & + & X \end{bmatrix}$$

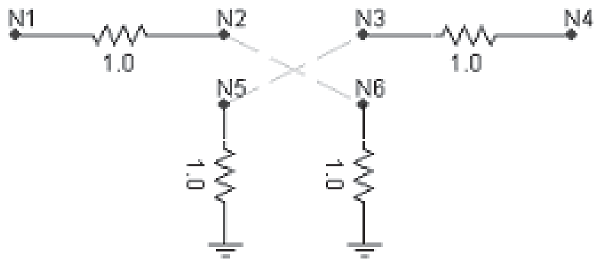
If the four electrical port connections on the transformer were given as follows in the Graphic section of the definition,



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:

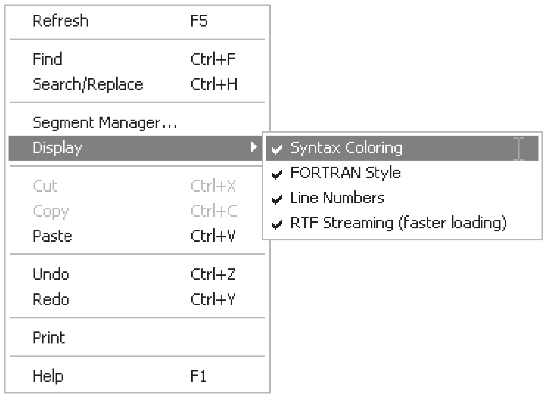
```
<Subsystem_#> <Cond_1> <Cond_2> <Cond_3> ...
```

<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.

Undo and Redo
Undo/redo is presently not supported in the Script section.

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 segment pop-up menu.



- **Syntax Colouring**: Select this option to enable code colouring. 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.
- **RTF Streaming (faster loading)**: Select this option to enable RTF (Rich Text Format) streaming. This option

ensures that script segment text is parsed into a Rich Text Editor control as part of a single block of text (as opposed to parsing the segment line-by-line). RTF streaming provides a significant speed improvement when loading very large script segment blocks.

INTERNAL OUTPUT VARIABLES

An *Internal Output Variable* is internal component data, which has been made available as a signal to be plotted or monitored. Before any internal quantities are made available however, they must 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*.

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.

The following should then appear in the Fortran segment:

```
#OUTPUT REAL Vdiff {$VDC:N1:N2}
```

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 Circuit canvas and select **Project Settings....** Under the Fortran tab, add a reference to the file in the *Additional Source (.f) Files* input field.
2. Add a *File Reference* component: Right-click over a blank area of the Circuit canvas and select **Add Component | File Reference**. Open the File Reference and select the source file.

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 pre-compiled object (*.obj) file.

C source cannot be inserted directly into the Fortran, DSDYN or DSOUT segments, as these are reserved for Fortran only.

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* 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
As mentioned above, in many instances it is perfectly alright 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* in Chapter 5 of the EMTDC manual 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.

Compaq Visual Fortran 6

If you are using the Compaq Visual Fortran 6 compiler, you must call your C procedures through an interface subroutine. The CVF 6 compiler requires 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
```

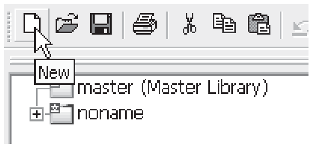
A standard call statement is then added to the Fortran segment of the user component definition:

```
        CALL AUX_CSUB($IN,$OUT)
```

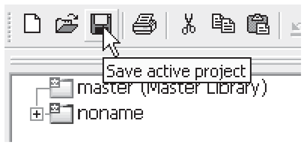
Where $IN and $OUT are input and output port connections on the component respectively.

TUTORIAL: CREATING A NEW COMPONENT

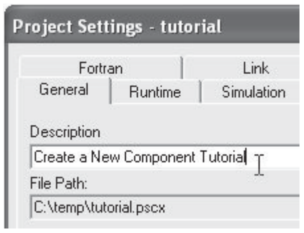
Open PSCAD, create a new case project, save it under a unique name, edit the project settings and add a description.



Create a New Case Project ('noname')



Save the Project Under a Unique Name (ex. 'tutorial.pscx')


Project Settings - tutorial

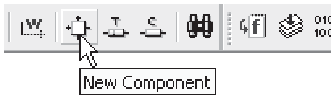
Fortran | Link | General | Runtime | Simulation

Description: Create a New Component Tutorial

File Path: C:\temp\tutorial.pscx

Edit the Project Description in the Project Settings Dialog

Create a new component using the *Component Wizard*.

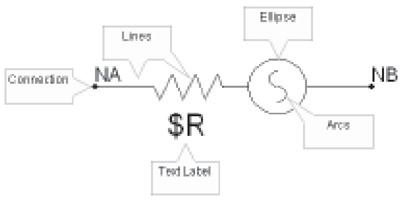


Invoke the Component Wizard

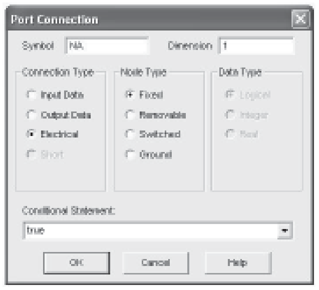
Adding Graphics

Edit the component definition and in the Graphic section, draw the following diagram (minus the sticky notes!); where *$R* is a text label, and *NA* and *NB* are port connections.

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 *$R*:


Port Connection

Symbol: NA Dimension: 1

Connection Type: Input Data [✓ Electrical] Output Data Short

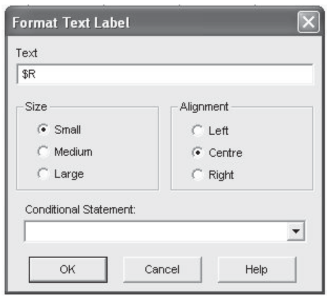
Node Type: [✓ Fixed] Removable Switched Ground

Data Type: [✓ Logical] Integer Real

Conditional Statement: true

OK | Cancel | Help

Connection Property Settings (*NA*)


Format Text Label

Text: $R

Size: [✓ Small] Medium Large

Alignment: Left [✓ Centre] Right

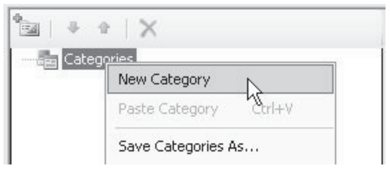
Conditional Statement:

OK | Cancel | Help

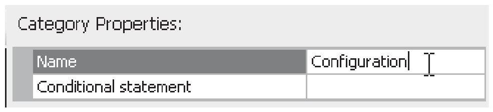
Text Label Property Settings (*$R*)

Save the project.

Adding a User Interface
Edit the component definition and in the Parameters section, add a new category page. Give the new category a name (say, 'Configuration').



Add New Category in Parameters Section


Category Properties:

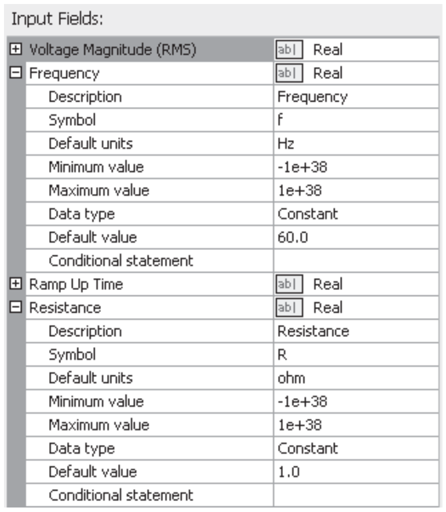
Name	Configuration
Conditional statement	

Give Category a Name

Add input 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*, *sec* 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.

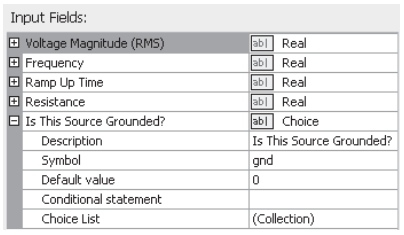


Input Fields:

⊞ Voltage Magnitude (RMS)	abl Real
⊟ Frequency	abl Real
Description	Frequency
Symbol	f
Default units	Hz
Minimum value	-1e+38
Maximum value	1e+38
Data type	Constant
Default value	60.0
Conditional statement	
⊞ Ramp Up Time	abl Real
⊟ Resistance	abl Real
Description	Resistance
Symbol	R
Default units	ohm
Minimum value	-1e+38
Maximum value	1e+38
Data type	Constant
Default value	1.0
Conditional statement	

Input Field Properties
(ex. *Frequency & Resistance*)

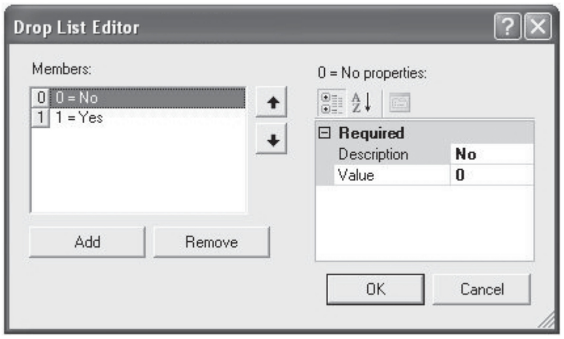
Now add a choice list to the category with the **Description** *Is This Source Grounded?*. Give it an appropriate **Symbol** name (say, *gnd*) and then add two choices: *1=Yes* and *0=No*.



Input Fields:

⊞ Voltage Magnitude (RMS)	abl Real
⊞ Frequency	abl Real
⊞ Ramp Up Time	abl Real
⊞ Resistance	abl Real
⊟ Is This Source Grounded?	abl Choice
Description	Is This Source Grounded?
Symbol	gnd
Default value	0
Conditional statement	
Choice List	(Collection)

Choice List Properties (*Is This Source Grounded?*)



Drop List Editor

Members:

0 0 = No
1 1 = Yes

0 = No properties:

⊟ Required

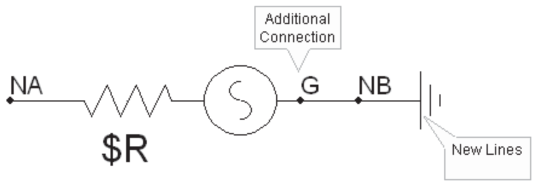
Description	No
Value	0

Drop List Editor

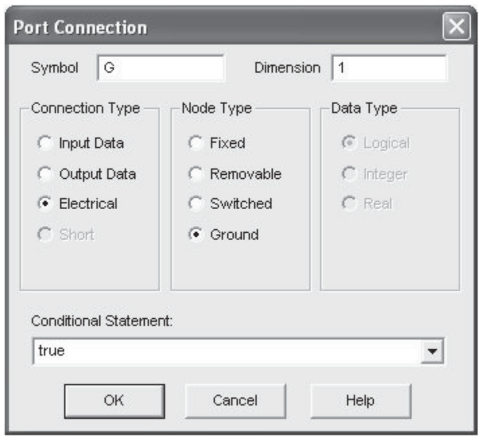
Save the project.

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 condition 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


Port Connection

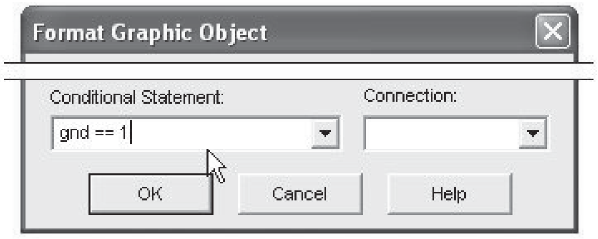
Symbol: G
Dimension: 1

Connection Type: ( ) Input Data ( ) Output Data (•) Electrical ( ) Short
Node Type: ( ) Fixed ( ) Removable ( ) Switched (•) Ground
Data Type: (•) Logical ( ) Integer ( ) Real

Conditional Statement: true

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.

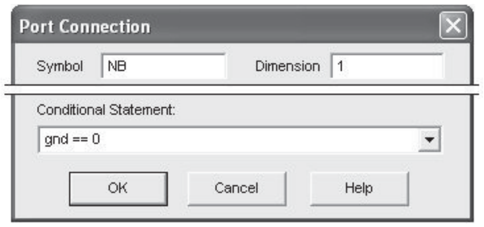


Conditional Statement: gnd == 1

Connection:

Editing Conditional Statement in Line Objects

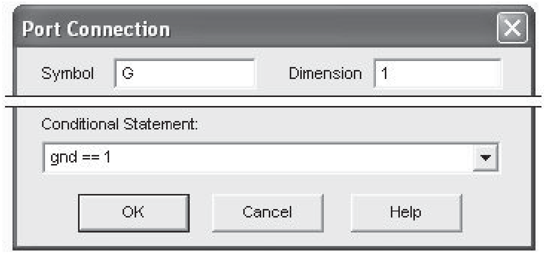
In the **Conditional Statement** field of port connections *G* and *NB*, add conditions so that they become enabled or disabled according to the setting specified in the *Is This Source Grounded?* choice list. That is, when *Is This Source Grounded?* is set to Yes, port connection *G* should be enabled and port connection *NB* should be disabled (and vice-versa):



Symbol NB Dimension 1

Conditional Statement: gnd == 0

Editing Conditional Statement in Port Connection *G*



Symbol G Dimension 1

Conditional Statement: gnd == 1

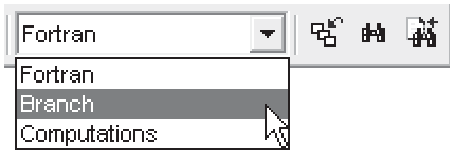
Editing Conditional Statement in Port Connection *NB*

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).

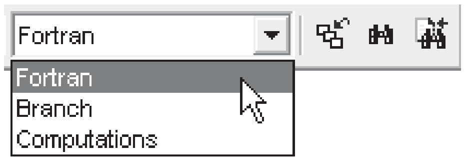
For more information on branch naming and the SOURCE syntax, see *Branch segment*.

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,$R,0.0,0.0)
      CALL E_1PVSRC_CFG(1,0,1,$Vrms,$f,0.0,$R,0.0,0.0,0.0,0.0,$tr)
#ENDBEGIN
#STORAGE LOGICAL:1 INTEGER:6 REAL:8 RTCF:4
#LOCAL REAL RVD1_1
#LOCAL REAL RVD1_2
#LOCAL REAL RVD1_3
#LOCAL REAL RVD1_4
! Single Phase AC source: Type: R
     RVD1_1 = RTCF(NRTCF)
     RVD1_2 = RTCF(NRTCF+1)
     RVD1_3 = RTCF(NRTCF+2)
     RVD1_4 = RTCF(NRTCF+3)
     NRTCF = NRTCF + 4
     CALL EMTDC_1PVSRC($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 [Ω]
- **HL**: Inductance value [H]
- **FC**: Capacitance value [μF]

```
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 [Ω]
- **L**: Inductance value [H]
- **RP_C**: Capacitance value [μF]
- **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:

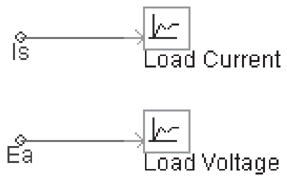
```
ERROR Source resistance must be greater than zero : R > 0.0
```

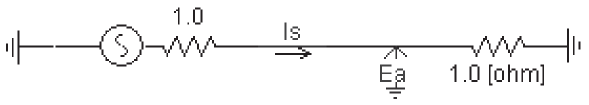
Save the project.

Testing Your New Component

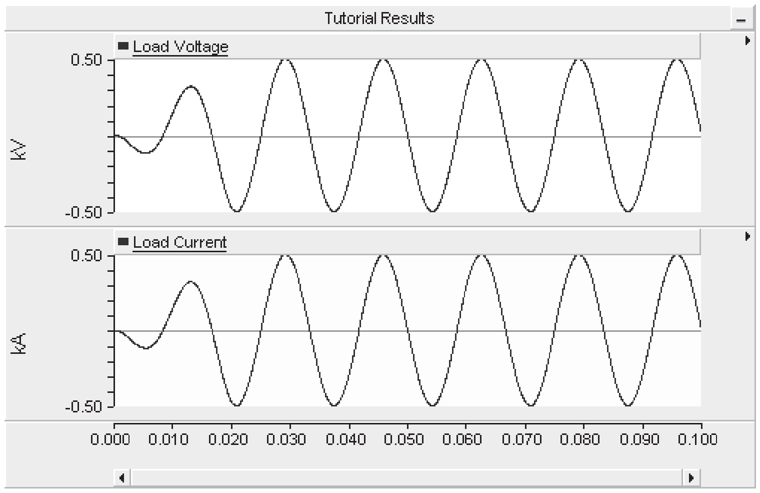
At this point it would be a good idea to check if your component will actually run, and if it does, whether the results are correct. Below are what the results should be if the source model was connected to the indicated circuit, and if the source is set to:

- Voltage Magnitude (RMS) = 0.7071 kV
- Frequency = 60 Hz
- Ramp-Up Time = 0.02 s
- Source Resistance = 1.0 Ω





The results are as shown below:

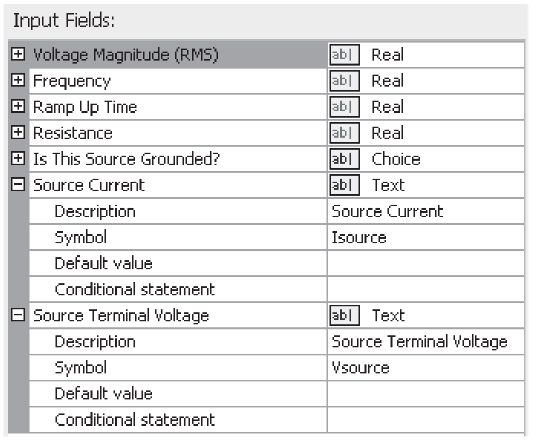


Save the project.

Adding Internal Output Variables

It is possible to monitor quantities internal to the component, for display on plots and meters. One way to accomplish this is by setting up an internal output variable. In this tutorial, we will set-up internal variables to monitor source voltage and branch current.

Edit the component definition and in the Parameters section, add two text fields: One with **Symbol** name *Isource* and **Description** *Source Current*; the other with **Symbol** name *Vsource* and **Description** *Source Terminal Voltage*.



Input Fields:

⊞ Voltage Magnitude (RMS)	abl Real
⊞ Frequency	abl Real
⊞ Ramp Up Time	abl Real
⊞ Resistance	abl Real
⊞ Is This Source Grounded?	abl Choice
⊟ Source Current	abl Text
Description	Source Current
Symbol	Isource
Default value	
Conditional statement	
⊟ Source Terminal Voltage	abl Text
Description	Source Terminal Voltage
Symbol	Vsource
Default value	
Conditional statement	

Text Field Properties (*Isource & Vsource*)

Save the project.

Edit the component definition and navigate to the Fortran segment. With an *#OUTPUT* directive statement, extract the current in this source branch (i.e. from branch *BRN*), using the *CBR* internal variable as follows:

```
#OUTPUT REAL Isource {$CBR:BRN}
```

The *#OUTPUT* statement above will define a variable *Isource* and output to it the current flowing in branch *BRN*.

Extract the node voltage from port connection *NA* to *G* or *NB* with another *#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:

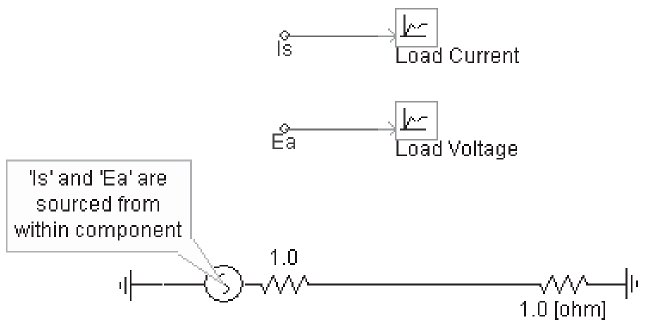
```
#BEGIN
      CALL E_BRANCH_CFG($BRN,$SS,1,0,0,$R,0.0,0.0)
      CALL E_1PVSRC_CFG(1,0,1,$Vrms,$f,0.0,$R,0.0,0.0,0.0,0.0,$tr)
#ENDBEGIN
#STORAGE LOGICAL:1 INTEGER:6 REAL:8 RTCF:4
#LOCAL REAL RVD1_1
#LOCAL REAL RVD1_2
#LOCAL REAL RVD1_3
#LOCAL REAL RVD1_4
! Single Phase AC source: Type: R
     RVD1_1 = RTCF(NRTCF)
     RVD1_2 = RTCF(NRTCF+1)
     RVD1_3 = RTCF(NRTCF+2)
     RVD1_4 = RTCF(NRTCF+3)
     NRTCF = NRTCF + 4
     CALL EMTDC_1PVSRC($SS,$BRN,RVD1_4,.TRUE.,RVD1_1,RVD1_2,RVD1_3)
!
#OUTPUT REAL Isource {$CBR:BRN}
#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 *Vs* and *Is* 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 10:

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 *Chapter 9 – 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. Chapter 9 contains a tutorial on designing your own component.

SUBSTITUTIONS

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 listed below:

Substitution	Description
$	Value Substitution Prefix Operator
%	Data Substitution Prefix Operator
{}	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.

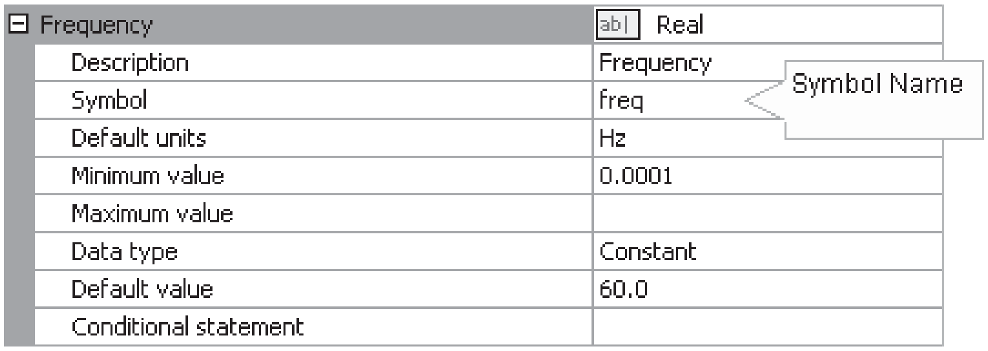
The $ operator may <u>only precede</u> variable names defined as one of the following:

- 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.

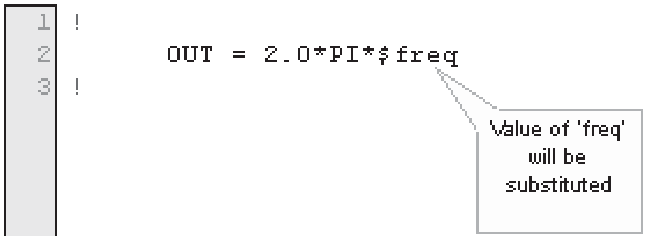
EXAMPLE 10-1:

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	Real
Description	Frequency
Symbol	freq
Default units	Hz
Minimum value	0.0001
Maximum value	
Data type	Constant
Default value	60.0
Conditional statement	

Input Parameter Field Properties



```
!
      OUT = 2.0*PI*$freq
!
```

Substituting Symbol Name in Script

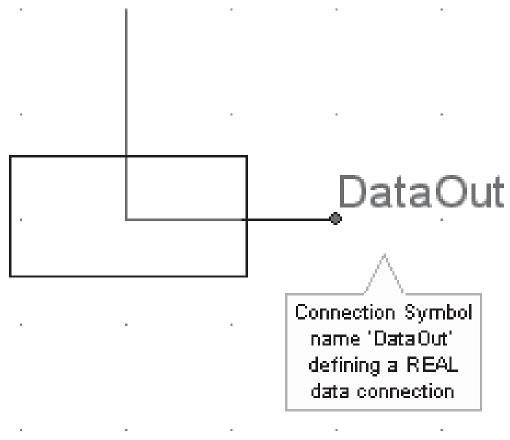
```
!
      REAL     RT_1             ! EMTDC variable to represent 'freq'
      REAL     OUT

      RT_1 = STOF(ISTOF + 2)   ! EMTDC extracts parameter value
                                ! from stored data array
      OUT = 2.0*PI*RT_1
!
```

Actual Fortran Code Following Compile

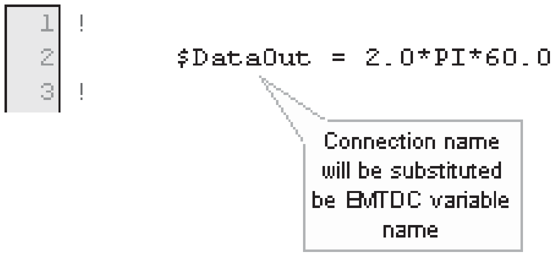
The input parameter value substituted for *freq* is the resulting value *following* unit processing. See the section entitled *Unit System* in *Chapter 5 – Operations and Feature Overview* for more details on units.

EXAMPLE 10-2:

The value of an output port connection entitled *DataOut* is defined by a Fortran statement. Upon compilation 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.


DataOut
Connection Symbol name 'DataOut' defining a REAL data connection

Component Graphic Containing a Connection



```
!
       $DataOut = 2.0*PI*60.0
!
```

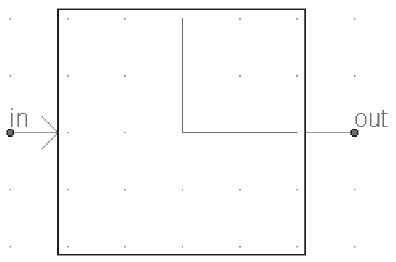
Substituting a Port Connection Name into a Fortran Statement

```
REAL    RT_1       ! EMTDC variable to represent ‘DataOut’

RT_1 = 2.0*PI*60.0
```

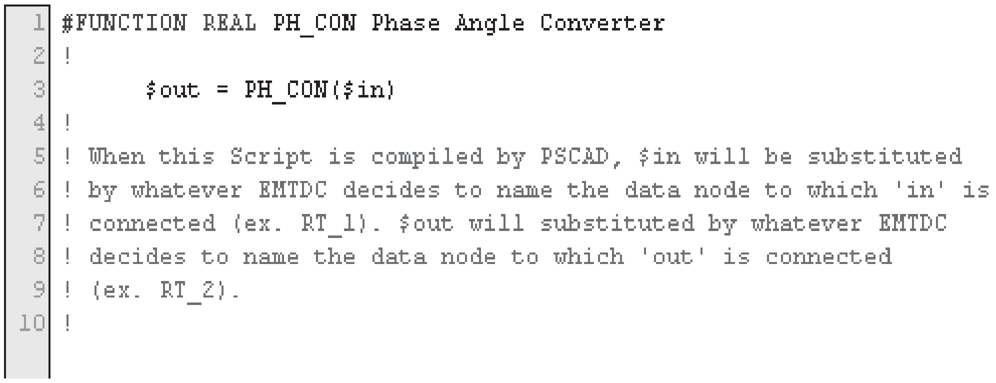
Actual Fortran Code Following Compile

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



```
#FUNCTION REAL PH_CON Phase Angle Converter
!
      $out = PH_CON($in)
!
! When this Script is compiled by PSCAD, $in will be substituted
! by whatever EMTDC decides to name the data node to which 'in' is
! connected (ex. RT_1). $out will substituted by whatever EMTDC
! decides to name the data node to which 'out' is connected
! (ex. RT_2).
!
```

Fortran Segment Script

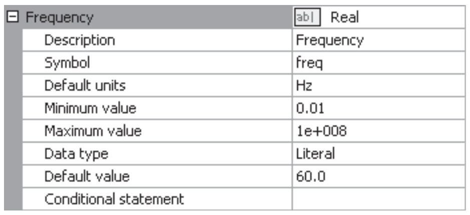
```
!
      REAL    PH_CON          ! Phase Angle Converter
      REAL    RT_1, RT_2
!
      RT_2 = STOF(ISTOF + 2)  ! EMTDC extracts port connection value
                              ! from stored data array
      RT_1 = PH_CON(RT_2)
!
```

See the #FUNCTION Directive for more details on declaring Fortran functions.

Actual Fortran Code Following Compile

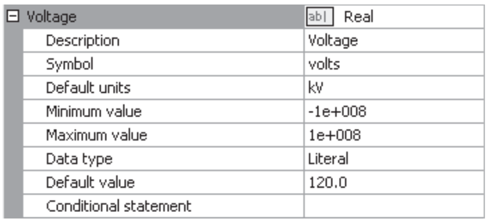
EXAMPLE 10-4:

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, with 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*)

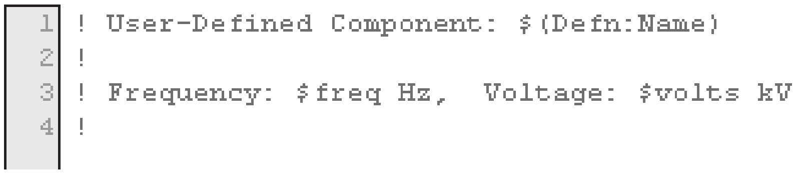


Voltage	Real
Description	Voltage
Symbol	volts
Default units	kV
Minimum value	-1e+008
Maximum value	1e+008
Data type	Literal
Default value	120.0
Conditional statement	

Input Field Property Settings (*volts*)

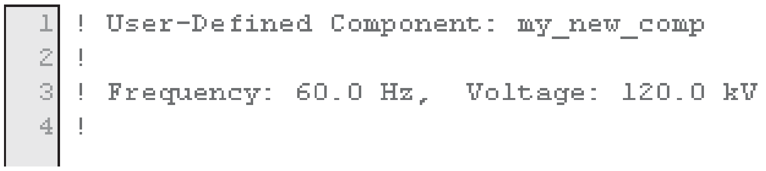
When the component is evaluated, the Symbol names (*freq* and *volts*) are given the value entered in their respective fields. In this case, *freq = 60.0* and *volts = 120.0*. The user is free to insert these variables elsewhere within the component script by using the $ operator.

Something similar to the following comments could appear in either the Fortran, DSDYN, or DSOUT segments:



```
! User-Defined Component: $(Defn:Name)
!
! Frequency: $freq Hz,  Voltage: $volts kV
!
```

Fortran Segment Script



```
! User-Defined Component: my_new_comp
!
! Frequency: 60.0 Hz,  Voltage: 120.0 kV
!
```

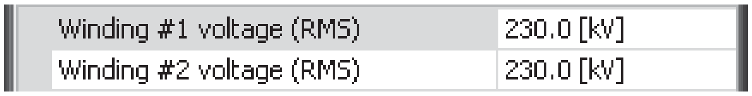
Actual Fortran Code Following Compile

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, for example, input parameters.

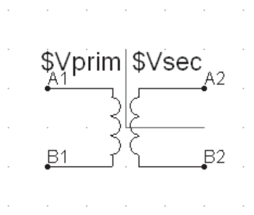
EXAMPLE 10-5:

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:

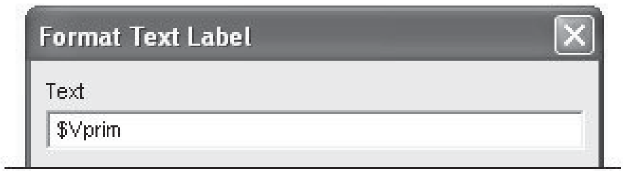


Winding #1 voltage (RMS)	230.0 [kV]
Winding #2 voltage (RMS)	230.0 [kV]

In the component Graphic Section, the user adds two new text labels (positioned accordingly) and enters *$Vprim* and *$Vsec* as shown below:



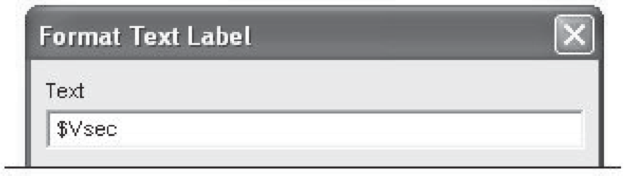
Component Graphic Containing Text Labels



Format Text Label

Text

$Vprim



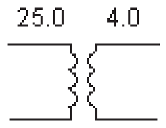
Format Text Label

Text

$Vsec

Text Label Property Settings

When the component instance is viewed on the Circuit canvas, input parameter values for *Vprim* and *Vsec* 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:

```
$(<Context>:<Key>)
```

The substitution contains both a <Context> and item <Key> separated by a scope operator and delimited by parentheses, where:

- **<Context>**: The context that is to be used to scope the item <Key>. See below for a list of valid <Context> names.
- **<Key>**: Item key (actual substitution) that is to be placed at the substitution point. See below for a list of valid <Key> 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 of the Script section. 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 <Context> and <Key> as described above (both Context and Key names are case sensitive):

Context Name	Description	Available Keys
<none>	An empty context key is by default the context of the component instance.	

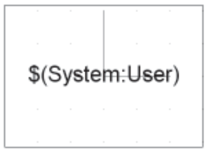
Defn	Context is that of the component definition.	Name, Desc, Path
Project	Context is that of the project that contains the component instance.	Name, Desc, Version, Author, Path
Session	Context is that of the current running session of the application (PSCAD).	Name, Version
System	Context is that of 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 hence, it may be excluded along with the scope operator. In such cases the syntax is:

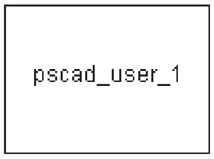
```
$(<Key>) or simply $<Key>
```

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

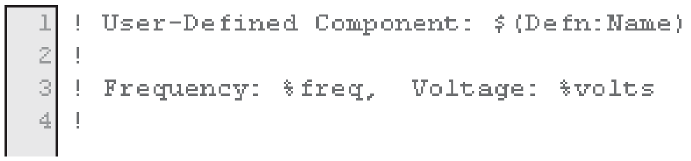
% Data Substitution Prefix Operator

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.

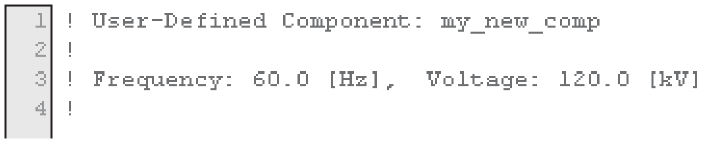
EXAMPLE 10-7:

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:



```
! User-Defined Component: $(Defn:Name)
!
! Frequency: %freq,  Voltage: %volts
!
```

Fortran Segment Script

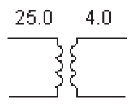


```
! User-Defined Component: my_new_comp
!
! Frequency: 60.0 [Hz],  Voltage: 120.0 [kV]
!
```

Actual Fortran Code Following Compile

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 *Vprim* and *Vsec* will appear on the component graphic as follows:



{ } Enfolding Operators (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:

- **$**: Optional. The *$ 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).

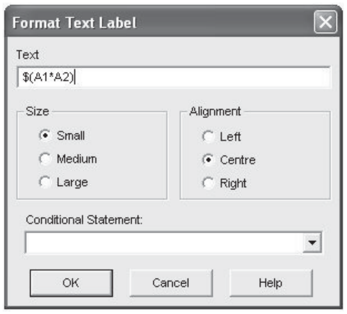
Although simple examples are provided here, you can find more examples by studying the definitions of master library components.

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 Label = A1*A2 = 6.0

The following is entered into the text label properties dialog:



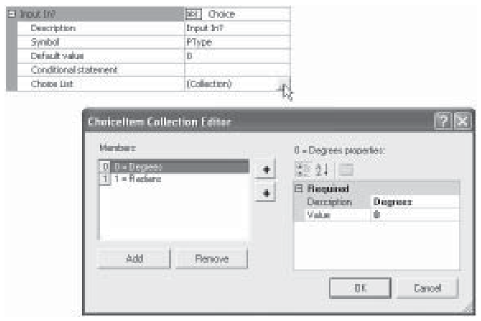
Text: $(A1*A2)

Size: Small

Alignment: Centre

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)

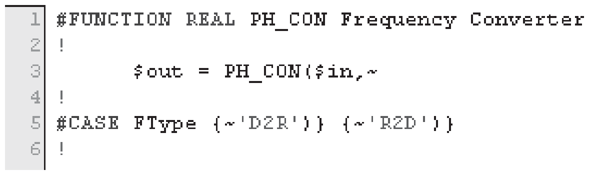
Members:

0 0 = Degrees
1 1 = Radians

Required	
Description	Degrees
Value	0

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*:



```
#FUNCTION REAL PH_CON Frequency Converter
!
      $out = PH_CON($in,~
!
#CASE FType {~'D2R'} {~'R2D'}
!
```

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 Operator
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 and Data files.

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.

See the #STORAGE directive for more details.

The following example illustrates the use of comments in the Fortran or DSDYN segments in the user component definition:

```
! MY FIRST PSCAD SCRIPT
! --------------------
!
! Storage:
!
#STORAGE REAL:1
!
! Main body of script:
!
```

SCRIPT DIRECTIVES

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>
      ...script...
#ELSEIF <Logic>
      #IF <Logic>
            ... script...
      #ELSE
            ... script...
      #ENDIF
#ELSE <Logic>
      ... script...
#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
!
```

Type == 1 in the above code is a *Logical Expression*. See *Expression Evaluation* for more details.

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.

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 <u>only if</u> 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
      $Z = XYZFUNC($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*.

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 in order to 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>
```

<TYPE> can be either REAL, INTEGER, or LOGICAL. <Name> is the given name for the local variable. <Array_Size> is an optional integer or substituted constant that defines the size of the array. If the variable has only a single dimension, then leave <Array_Size> 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
```

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.

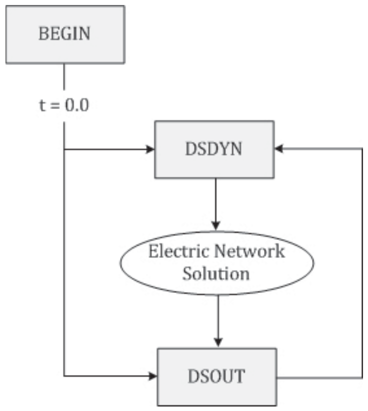
A > 1 in the above code is a *Logical Expression*. See *Expression Evaluation* for more details.

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)
!
```

#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 are 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 π-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 M^{th} winding of the N^{th} 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.

~ 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.

The line continuation operator can appear anywhere on the segment line, and it can be used with the #CASE Conditional Directive.

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
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
LOG(x)	Natural logarithm
EXP(x)	Exponential
LOG10(x)	Base 10 logarithm
SQRT(x)	Square root

ABS(x)	Absolute value
REAL(x)	Real part of complex number
IMAG(x)	Imaginary part of complex number
NORM(x)	Norm of complex number (x2 + y2)
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
INT(x)	Removes fractional part of REAL value (right side of decimal)
FRAC(x)	Removes integer part of REAL value (left side of decimal)
RAND(x)	Random value between 0 and x
P2RX(m,θ)	Polar to rectangular conversion (θ in degrees)
P2RY(m,θ)	Polar to rectangular conversion (θ in degrees)
R2PM(x,y)	Rectangular to polar conversion
R2PA(x,y)	Rectangular to polar conversion

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.

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
!
```

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*.

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.

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 11:

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

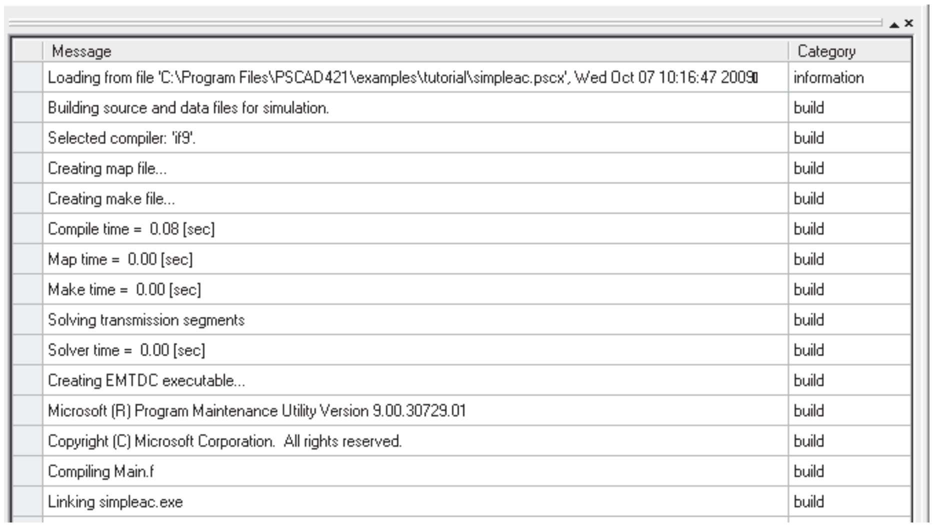
PSCAD does not include a real time error manager, so in order to generate and view any warning or error feedback you must first compile and build your project. Error messaging is displayed in the *Output Window*, and is organized by compilation step. This classification can be very helpful in determining just what the problem is, and how to fix it.

As discussed in The *Output Window* in Chapter 4 of this manual, the window is divided into tabbed sub-windows –these are called *Messages* and *Search Results*. The *Messages* tab includes messages sourced from the two main parts of the simulation: The building or compilation of the project (*build*), and the simulation run itself (*runtime*). As it pertains to error messaging, these two parts of the simulation are exclusive of each other. The *Messages* tab will also display messages involving transmission line and cable solving, as well as 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 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. However, 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 resulting from the steps described above in the output window (for the *simpleac* tutorial project):



	Message	Category
	Loading from file 'C:\Program Files\PSCAD421\examples\tutorial\simpleac.pscx', Wed Oct 07 10:16:47 2009	information
	Building source and data files for simulation.	build
	Selected compiler: 'if9'.	build
	Creating map file...	build
	Creating make file...	build
	Compile time = 0.08 [sec]	build
	Map time = 0.00 [sec]	build
	Make time = 0.00 [sec]	build
	Solving transmission segments	build
	Solver time = 0.00 [sec]	build
	Creating EMTDC executable...	build
	Microsoft (R) Program Maintenance Utility Version 9.00.30729.01	build
	Copyright (C) Microsoft Corporation. All rights reserved.	build
	Compiling Main.f	build
	Linking simpleac.exe	build

There was not problem building the *simpleac* project. Any warning or error messages that may have occurred would have been indicated in the column at the far left. See The *Output Window* in Chapter 4 of this manual for details.

Runtime Messaging
Upon the completion of the build process, the runtime process will begin. Runtime messaging is organized into the following categories:

1. **Communications**: Messages related to the running of the executable file are placed here.
2. **Copyright**: Displays EMTDC software copyright information.
3. **Non-Standard Messages**: These messages are the most important of the runtime process, and include all messages sourced from EMTDC (such as 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 these messages.
4. **Time Summary**: Displays a summary of simulation execution times.

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 M*Ω) 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.

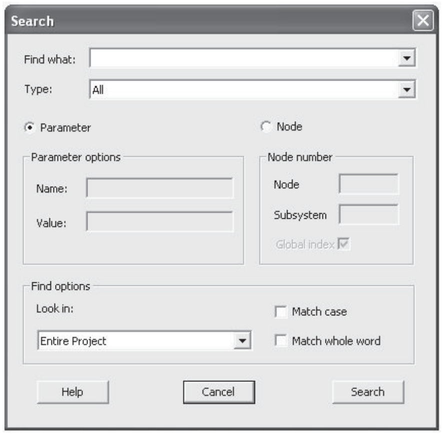
<*Node*> is the name of the Breakout connection which is shorted.

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.

SEARCHING

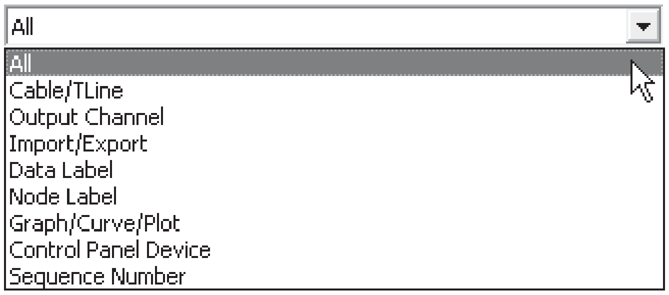
If the user wishes to search a project for a particular object, the search utility may be utilized. To invoke the Search dialog window, simply select **Edit | Search...** from the main menu bar, press the **Search** button in the main toolbar or press **Ctrl + f** on your keyboard, while viewing the Circuit canvas.



The input parameters available in this dialog are described below.

Find What
Enter the character string that you would like to search for. Note that if you are performing a *Node* search, this input is disabled.

Type
It is possible to narrow a search to a certain type of object in the project:



Parameter/Node
The *Parameter* and *Node* radio buttons toggle the search utility between a parameter-type lookup (i.e. search of string or sub-string), with that of a node number search. A node number search is a different animal, in that it requires the project be compiled first, thereby generating the node numbers (See *Node Number* below).

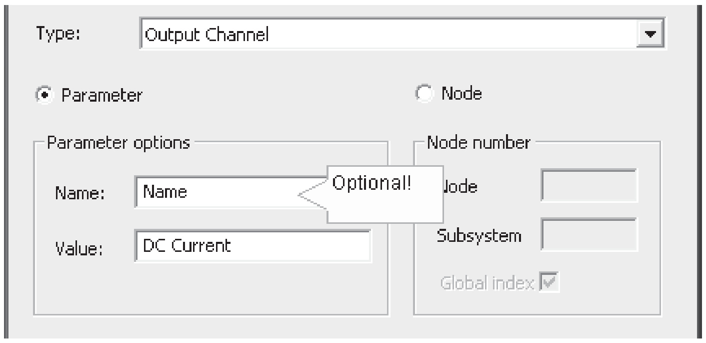
Parameter Options
The options available in this section are specifically for searching XML parameter nodes. Note that these options will appear disabled when *Type* is set to either *All* or *Sequence Number*.

Parameter Name and Value
All XML parameter nodes in the project file will possess both a *name* and a *value* attribute. This can be confusing when you are searching for a particular parameter.

For example, say you are searching for a specific output channel component with a *Title* parameter as 'DC Current.' The XML node for this component instance appears as follows in the project file:

```
<User classid="UserCmp" name="pgb" id="23921160" x="414" ...
link="-1">
   <paramlist link="-1" name="">
      <param name="Name" value="DC Current" />
      <param name="Group" value="Inverter" />
      <param name="Display" value="1" />
      <param name="Scale" value="0.5" />
      <param name="Units" value="" />
      <param name="mrun" value="1" />
      <param name="Pol" value="0" />
      <param name="Min" value="-0.5" />
      <param name="Max" value="2.0" />
   </paramlist>
</User>_
```

Notice that the XML parameter name in this case is ‘Name.’ Therefore, to find this object in the project, based on its *Title* parameter, you must enter the parameter options as follows:


Type: Output Channel

[✓ Parameter] Node

Parameter options

Name: Name

Value: DC Current

Note that most of the time you will not know what the parameter *Name* is (only its *Value*) -- you can leave the *Name* field blank and just search for the *Value* parameter as *DC Current*.

Node Number
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:

- **Node**: The actual node number.
- **Subsystem**: The electrical subsystem number (if desired)
- **Global Index**: Select this option if the node you are searching for is a global node. Otherwise, the utility will search for local nodes.

To navigate to a specific object connected to this node, right-click on a search result and select **Navigate To...**.

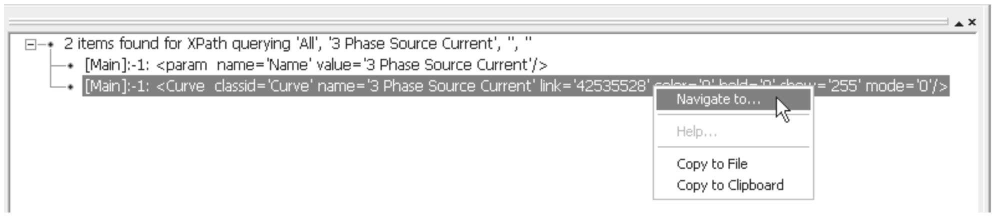
Find Options
There are a few options available to help focus your search:

- **Look In**: Select **Entire Project**, **Current Circuit** or **Search Results**. *Entire Project* is selected by default, but you can narrow your search by confining it to the current module canvas (*Current Circuit*). The *Search Results* selection allows you to search only within the results of a previous search.

- **Match Case:** The search will be case sensitive.
- **Match Whole Word**: The search will be limited to the string as a complete word, as opposed to just a sub-string.

Viewing Search Results

Search results are located in the **Search Results** section tab of the *Output Window*. This tab will automatically open upon performing a search. Any result matching the search criteria will appear in a list form as shown below:



```
2 items found for XPath querying 'All', '3 Phase Source Current', '', ''
    [Main]:-1: <param  name='Name' value='3 Phase Source Current'/>
    [Main]:-1: <Curve  classid='Curve' name='3 Phase Source Current' link='42535528' color='0' bold='0' show='255' mode='0'/>
```

Each entry in the list represents an XML node. Simply left double-click on a particular list entry (or right-click and select **Navigate To...**) and PSCAD will automatically highlight the source in the Circuit canvas.

VIEWING BUILD AND DATA FILES

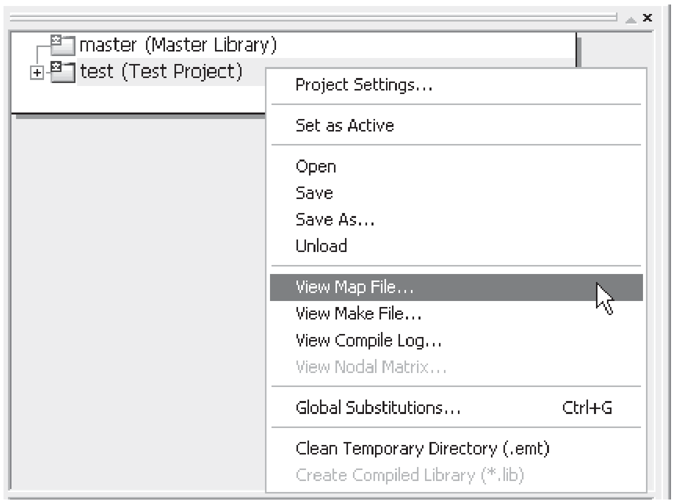
Whenever a project is compiled and built, several files are created and stored in the associated project temporary (*.emt) directory. Some of these files, such as Fortran, Data and Map files, can be useful in debugging. All of these files can be viewed directly from within PSCAD.

Fortran and Data Files

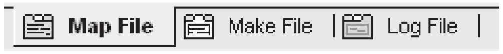
Fortran and Data files can be viewed by simply clicking the *Fortran* or *Data* tabs at the bottom of the main canvas (following compilation of course). These files are module specific and are automatically generated by PSCAD. These files may not be edited.

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...*.



This will invoke a separate file viewer window. This viewer possesses its own tab bar, which allows you to switch between files quickly.



Note that the Log file is a simple file created by the compiler to log the compilation process.

COMPONENT ORDERING

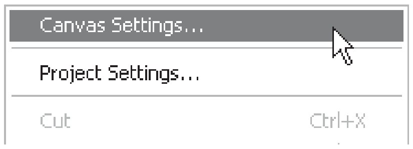
PSCAD includes a smart algorithm, which will automatically sequence (or order) all components involved in the *EMTDC System Dynamics*. This is carried out automatically in order to ensure that variables are calculated in their proper sequence, and that time step delays are minimized: The algorithm iteratively 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.

Component ordering functions are performed by PSCAD on a per module basis. By default, the **Assign Sequence Numbers** option in the *Canvas Settings* dialog is enabled 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.

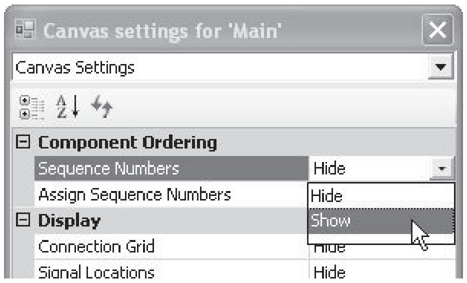
This algorithm should be left on by default at all times; however there may be instances where it is desirable to turn it off and manually order the components during the debugging process. This may be the case if you wish to manually control the feedback points. Alternatively, a feedback can be introduced by inserting a *Feedback Loop Selector* component in the signal path.

Component ordering features may be accessed through the *Canvas Settings* dialog. See *Editing Canvas Settings* in Chapter 5 of this manual for more details.

Showing Sequence Numbers
Before manually ordering any components, you must first compile the case and then ensure that the **Sequence Numbers** setting is set to Show in the Canvas Settings dialog. To bring up this dialog, right-click on a blank part of the Circuit canvas and select *Canvas Settings...*



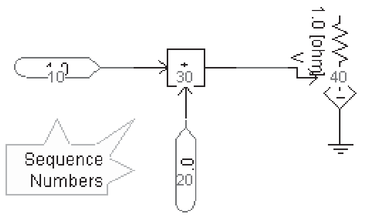
Circuit Canvas Pop-Up Menu



Component Ordering
Sequence Numbers: Hide (Hide / Show)
Assign Sequence Numbers
Display
Connection Grid: Hide
Signal Locations: Hide

Module 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:



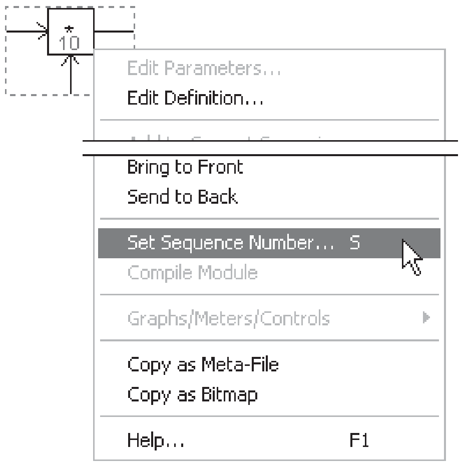
There are two possible locations within the system dynamics (i.e. DSDYN or DSOUT) where the code for a particular component can reside. As a result, the sequence numbers are colour coded so that the user can graphically determine where code is. These colours are listed below:

Colour Legend:

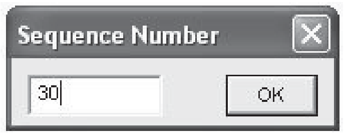
- **Aqua**: The component code resides within DSDYN for the current module.
- **Olive**: The component code resides within DSOUT for the current module.

Manually Setting Sequence Numbers

To manually set sequence numbers, first ensure that the **Sequence Numbers** option is set to show (as described above), and that the **Assign Sequence Numbers** option is <u>disabled</u> in the *Canvas Settings* dialog. Then, right-click on a component and select the **Set Sequence Number...** option from the pop-up menu.



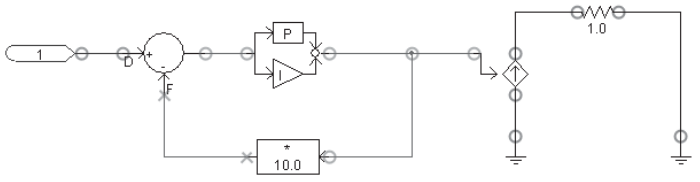
This will bring up the *Sequence Number* dialog.



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 **Signal Locations** 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:

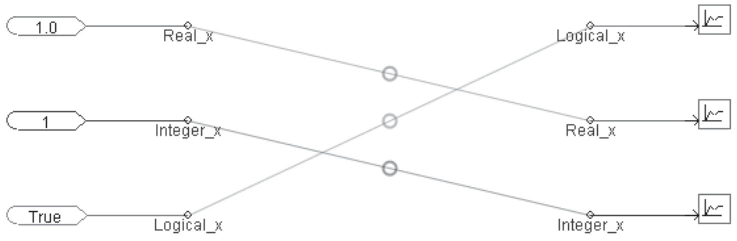
- ○ **Feed-Forward Connection:** This symbol indicates that the signal passing through the connection point is classified as a feed-forward signal. This means that the value of the signal is always defined within the present time step.
- × **Feedback Signal:** This symbol indicates that the signal passing 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 - Δt). 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.

Colour Legend:

- **Green**: Indicates a signal of type REAL
- **Blue**: Indicates a signal of type INTEGER
- **Magenta**: Indicates a signal of type LOGICAL
- **Grey**: Indicates an electrical signal

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:



The dashed connections are <u>purely visual</u>, 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 control wires.

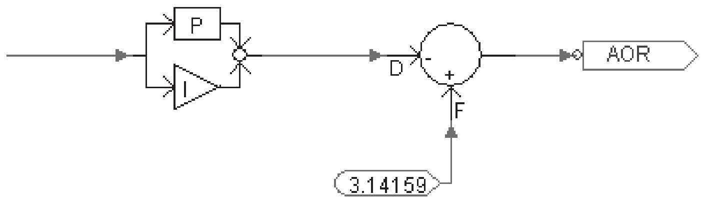
To enable this feature, right-click on a blank part of the module canvas and select **Canvas Settings...** to bring up the *Canvas Settings* dialog. Select the **Virtual Wires** option and change it to *Show*.

The colour of the virtual control wire indicates the data type of the data label 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.



The indicators are orientated by default <u>according to the Wire component direction</u> (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.

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**.

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

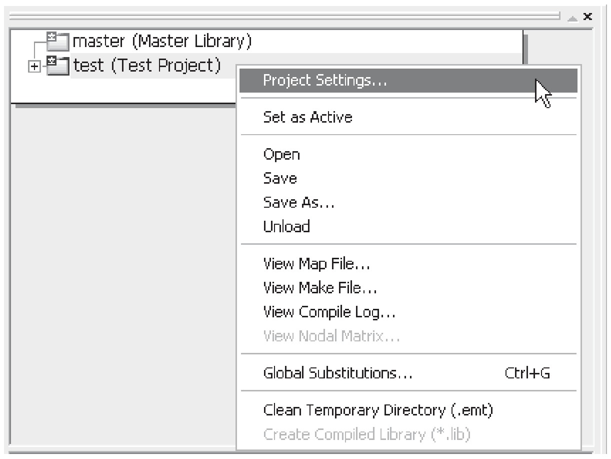
DEPLOYMENT OF AN INTEGRATED DEBUGGER

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.

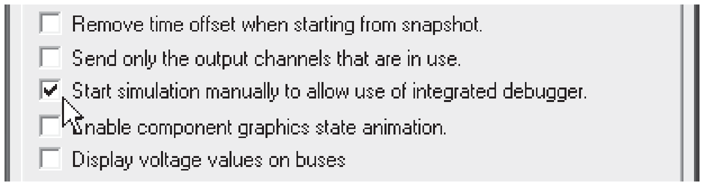
If you are using the free GFortran compiler supplied with PSCAD, there are debugging tools available. See *http://sources.redhat.com/insight/index.php*.

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. This dialog can be accessed through a right-click on the project title itself (either a case or a library) in the Workspace window, and selecting **Project Settings...** from the pop-up menu.

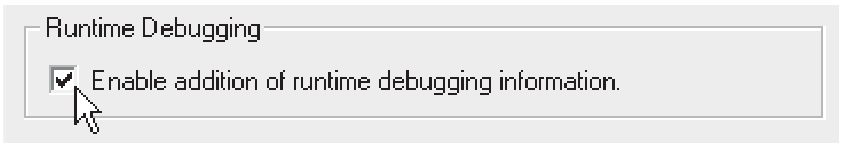


In the *Runtime* section of the dialog, enable the **Start simulation manually to allow use of integrated debugger** option. Enabling this option allows EMTDC to be started manually.



☐ Remove time offset when starting from snapshot.
☐ Send only the output channels that are in use.
☑ Start simulation manually to allow use of integrated debugger.
☐ Enable component graphics state animation.
☐ Display voltage values on buses

Next, navigate to the *Fortran* section of the dialog and enable the **Enable addition of runtime debugging information** option in the *Runtime Debugging* area.

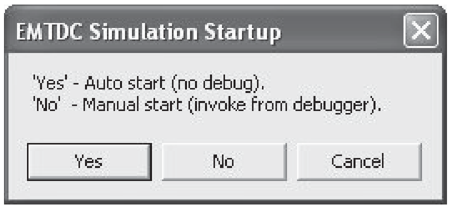

Runtime Debugging

☑ Enable addition of runtime debugging information.

Ensure that any user source files are referenced in the Additional Source (.f) files 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 (press the **Run** button in the main toolbar). A pop-up message should appear as follows:


EMTDC Simulation Startup

'Yes' - Auto start (no debug).
'No' - Manual start (invoke from debugger).

2. Select the **No** button to continue with debugging.

3. On the *Status Bar* (at the bottom of the PSCAD environment) you should see a command line indicated. If you cannot see the status bar, select **View | Status Bar** from the main menu. The general format of this command is as follows:

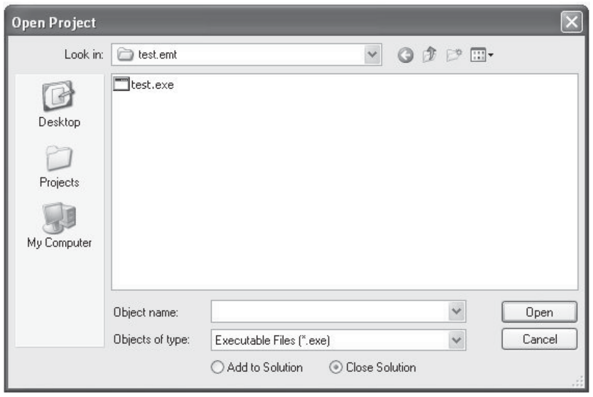
```
Command: <project_name> -v4 localhost ####
```

Jot down the four-digit number at the end of the command line so as not to forget it (you will need it soon).

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 temporary directory (*.emt) associated with your project file (located in the same directory), select the executable file (*.exe) for your project and then click the **Open** button. For example, if your project is entitled 'test.psc,' then you would select 'test.exe' from the 'test.emt' directory as shown below:



test.emt

test.exe

Object name:

Objects of type: Executable Files (*.exe)

Add to Solution [✓ Close Solution]

5. While still in Intel Fortran, select **Project | Properties...** (CVF: **Project | Settings...**) to bring up the Property Pages dialog. Click the **Debugging** branch (CVF: **Debug** tab) and in the **Command Arguments** field, enter the following:

```
-v4 localhost ####
```

Note that the number (represented by #### above) must be the same as that in Step 3. Click the **OK** button.

6. While still in Intel Fortran, 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. 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. Also, the above steps must be repeated during subsequent runs, as the four-digit link number will change each time.

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

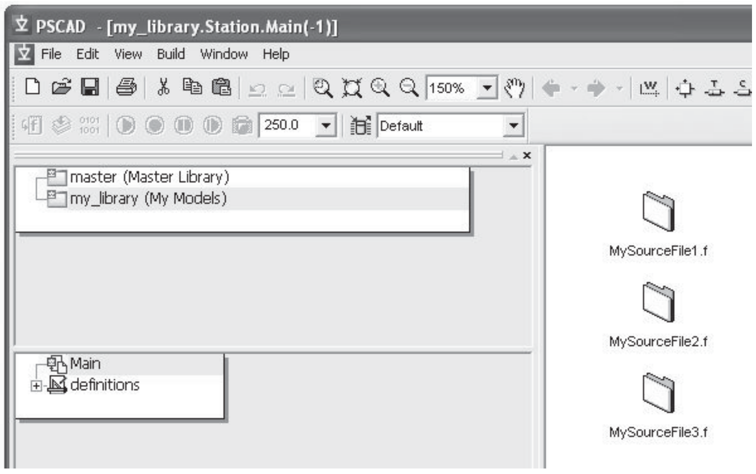
Normally, when a source file is linked to a project, the Fortran compiler used to compile the project will automatically create a compiled object (*.obj) file for each linked source file. This file is placed in the project temporary (*.emt) directory. The user may choose to supply clients with these 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 *Chapter 7 – Additional Library (*.lib) and Object (*.obj) Files*.

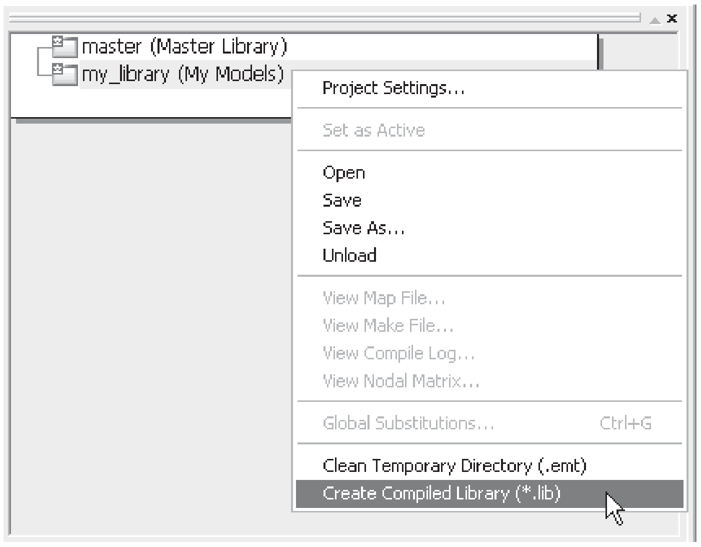
Step 1:

Create a new library project as described in *Chapter 5 – 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 (*.emt) directory 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 *Chapter 7 – 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).

MODULE LOCKING

Starting in PSCAD X4, it is possible to lock the canvas of any module from being viewed. The locking mechanism uses a sophisticated algorithm to encrypt the XML data, describing the contents of the module canvas in the project (*.*pscx*) file. Once locked, the module definition can then be unlocked by means of a password, which is a required user input at the time of locking the module. For details on how to lock or unlock a module, see *Lock/Unlock Modules* in chapter 5 of this manual.

It is important to note that this feature should *not* be viewed as a means to secure proprietary information. Regardless of whether or not the module has been locked, there will still be an associated Fortran file generated for the module definition on compile. Proprietary control system designs can be reconstructed given this code. In the future, the ability to link pre-compiled object and library (*.obj and *.lib) files to a module (similar to how this can be accomplished with non-module components) will be provided.

The best way to ensure your proprietary designs are protected, is to use non-module components with linked, pre-compiled object and library (*.obj and *.lib) files.

Chapter 12:

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. MATLAB must be installed on your computer in order to use the MATLAB interface.
2. 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 '*.mdl' file path
MFILE	CHARACTER	Name of module within '*.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:

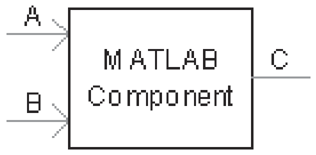
1. Create a new component.
2. Write a MATLAB file (.m) to perform the required modeling.

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:

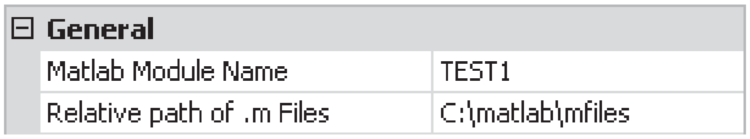
Consider a simple example for a PSCAD component, which has two REAL input connection nodes (A and B), and a single REAL output connection node (C).



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).

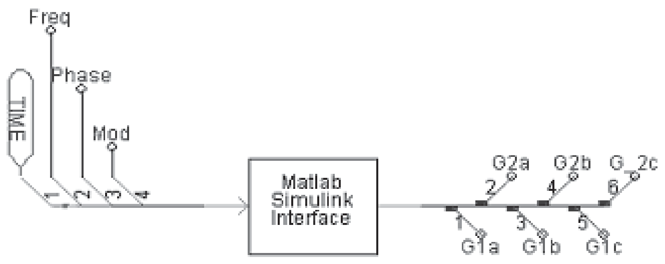
DESIGNING A MATLAB/SIMULINK COMPONENT

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.

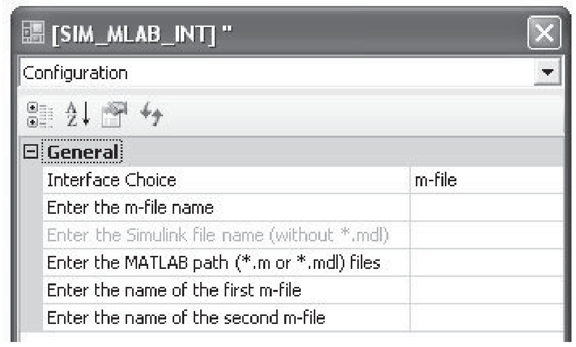


The following code appears in the Fortran segment of the component definition:

```
#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:



Interface Choice	m-file
Enter the m-file name	
Enter the Simulink file name (without *.mdl)	
Enter the MATLAB path (*.m or *.mdl) files	
Enter the name of the first m-file	
Enter the name of the second m-file	

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

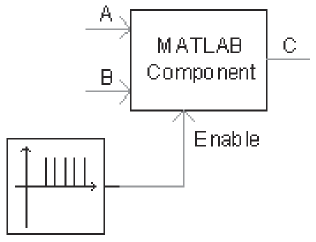
There is an alternative way to invoke a Simulink module. Instead of calling the 'SIMULINK_INT' subroutine, users may call 'MLAB_INT' subroutine to invoke an '*.m' file, which uses the MATLAB command 'sim' to handle the Simulink module. However, the use of 'SIMULINK_INT' is highly recommended due to the 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 should also be possible to convert ‘*.m’ source code directly to ‘C’ code using the MATLAB ‘C’ compiler, and then directly compile and link the ‘C’ source code into the EMTDC executable. This has not been tested thoroughly, but no obvious technical reasons exist as to why it should not. 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 System Settings dialog under the MATLAB tab.

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.

Once the MATLAB® interface particulars are set up, you must then enable the interface in your case Project Settings under the Link tab.

Chapter 13:

Migrating from Older Versions

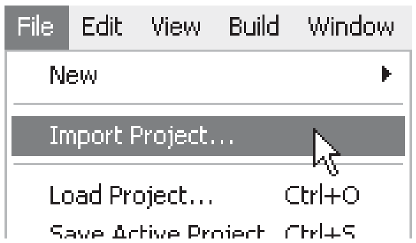
The PSCAD X4 release represents a vast transformation in program architecture. It is also a prudent point, at which to continue unfettered by old operating systems, Fortran compilers, and older PSCAD versions. This latest version allows you to import *.psl and *.psc format project files that were last saved in PSCAD v4.1.x or v4.2.x 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.

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 v4.1.x or v4.2.x. 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 V4.1 AND V4.2 PROJECTS TO X4

Importing a v4.1 or v4.2 project into PSCAD X4 is for the most part, as simple as a couple of mouse clicks. Simply select **File** | **Import Project…** from the main menu, to import a v4.1 or v4.2 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 v4.1 or v4.2 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 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	New in X4
Master Library	Local Project
Other Library Projects	Master Library
Local Project	Other Library Projects

Definition Linking Priority on Import/Load Project

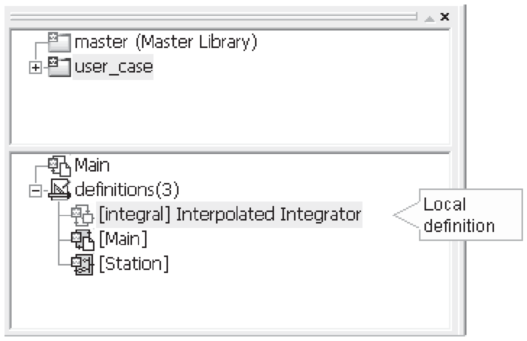
Problems to be Aware of

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, 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.



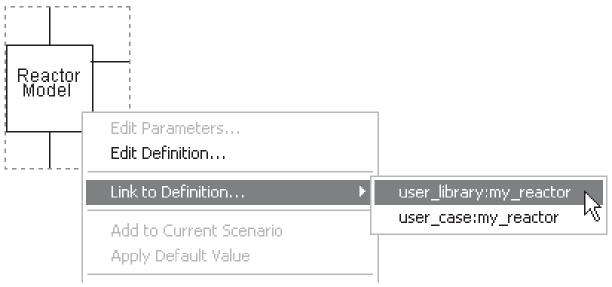
This will most likely result in compilation errors and other problems, or it may even go undetected.

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 v4.1 or v4.2 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.

Custom Definition Linking (Namespace)

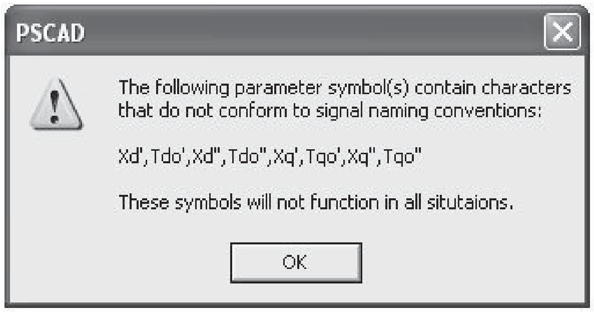
A new feature exists in PSCAD X4 that enables users to manually re-link any component or module instance to a specific definition. Once a project has been imported and saved under the new file format, the user may manually re-link any instance to any definition from a project loaded in the workspace. This is accomplished through a technique called *Namespacing*, which provides absolute path information to a definition by pre-pending the project name where the definition is stored.



This is especially useful in dealing with duplicate definitions, as the user may select a specific definition to link to, regardless of how many duplicates there are. See *Namespace* in chapter 7 of this manual for more details.

Initially, when an older project is imported, all links to definitions are set according to the linking priority of the importer, and are absolute paths. It is therefore especially important that the rules outlined in this topic are followed before importing projects, as the initial links to definitions may be incorrect and would need to be hunted down and corrected.

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:



The following parameter symbol(s) contain characters that do not conform to signal naming conventions:

Xd',Tdo',Xd'',Tdo'',Xq',Tqo',Xq'',Tqo''

These symbols will not function in all situtaions.

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

Manual Modification of Script

Although the project importer will remove the illegal characters from the areas listed above, they will not be removed from the component definition script sections. This means that there will be undeclared *Symbol* names in the definition script, which will be detected when the project is compiled.

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
`Xd''`	`Xd__`

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__)
```

Adding Multiple Instance Module Support to User-Defined Components

With the inclusion of Multiple Instance Module (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*.

There are a few new sections that have been added to the manual documentation that describe this issue. They are listed below:

PSCAD Manual:

- Chapter 5 – Operations and Feature Overview: *Multiple Instance Modules (MIM)*
- Chapter 9 – Component Design: *The Parameters Section | Input Fields*

EMTDC Manual:

- Chapter 2 – Program Structure: *System Dynamics* (various sections)
- Chapter 5 – 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

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.

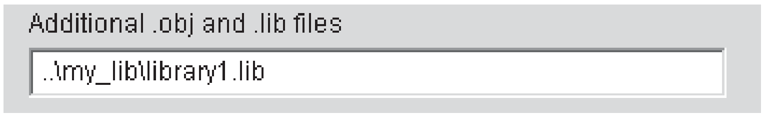
This important feature should be considered when initially verifying your V3 project results in PSCAD V4. The new sequencing algorithm may adjust the order in which components appear in the Fortran code. Depending on where the components within the EMTDC System Dynamics (i.e. DSDYN or DSOUT), a single 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 *Additonal .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.

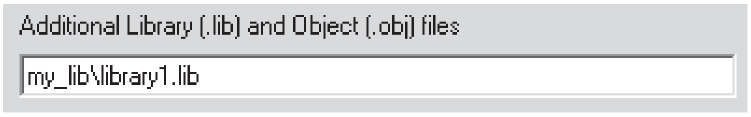
These paths must be changed accordingly in your V4 projects. Open the respective Project Settings dialog for the project (right-click on the project filename in the Workspace and select *Project Settings...*) and select the *Link* tab. Modify the paths in the *Additional Fortran Library (*.lib) and Object (*.obj) Files* input field. For example, a V3 reference is shown below:



Additional .obj and .lib files

..\my_lib\library1.lib

would appear as follows in PSCAD V4:



Additional Library (.lib) and Object (.obj) files

my_lib\library1.lib

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 and 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
```

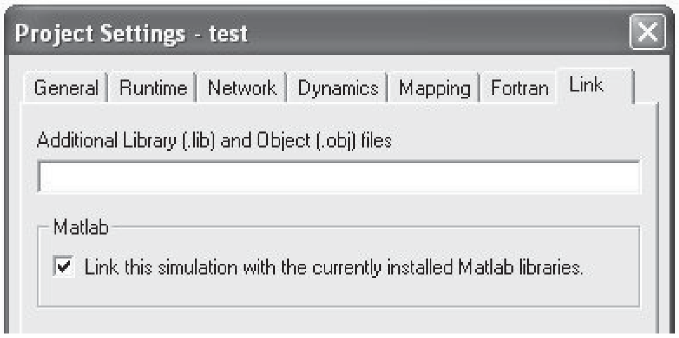
V4 format (multi-subsystem):

```
$Loop:SS, $Loop:NA, $Loop:NB, $Loop:NC
```

MATLAB Libraries and Interface

Project and Workspace settings should be adjusted slightly when migrating 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.



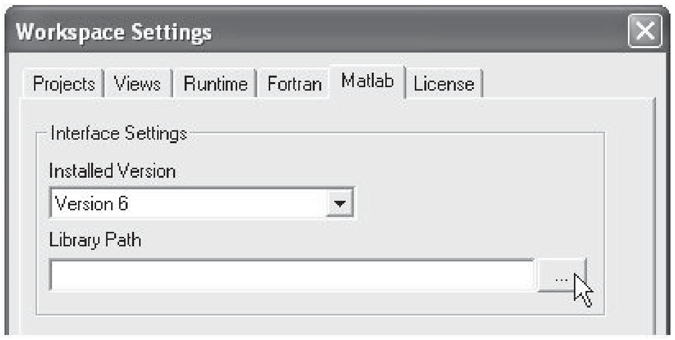
Project Settings - test

Additional Library (.lib) and Object (.obj) files

Matlab

✓ Link this simulation with the currently installed Matlab libraries.

2. Open the *System Settings* dialog by selecting *Edit* | *System 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)

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

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 two 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 users to overwrite 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 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.

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.

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.
- 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 I*mporting V2 User Libraries* or *Importing Individual V2 Components* below for more details.

If a V2 draft project containing undefined components is loaded into PSCAD V4, all undefined components will be substituted with a temporary ‘placeholder’ component.

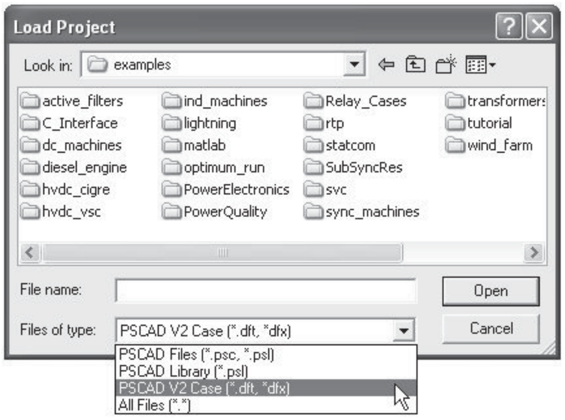
Please follow these steps to avoid problems:

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.

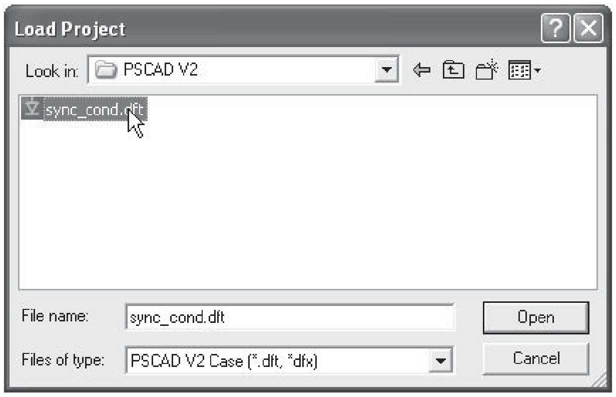
2. Near the bottom of the Load Project dialog, change the *Files of Type* drop list to *PSCAD V2 Case (*.dft, *.dfx)* so as to view PSCAD draft files only. If you do not have a direct

network connection from your PC to your UNIX system, then you must first copy (or ftp) the files to your PC from UNIX.

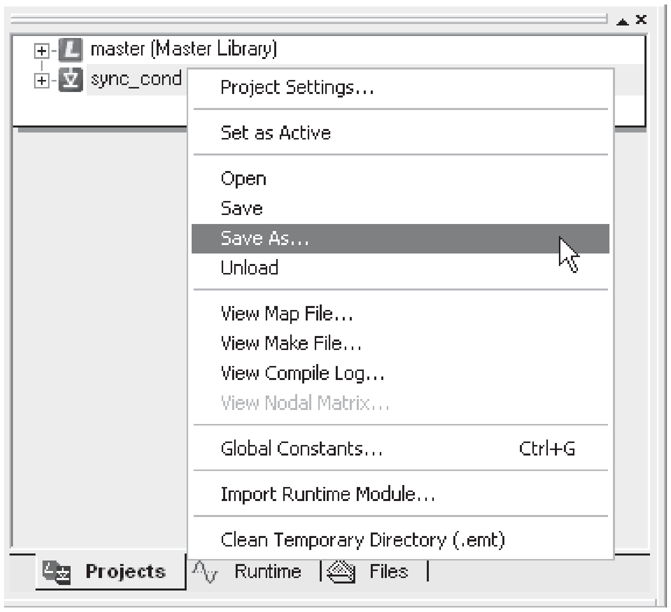
Note your V2 files may still be in UNIX format. In order to successfully import the file to PSCAD V4, you must ensure that all files are in DOS format. This can be accomplished by using the 'unix2dos' UNIX utility, or FTP in ASCII mode.



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:

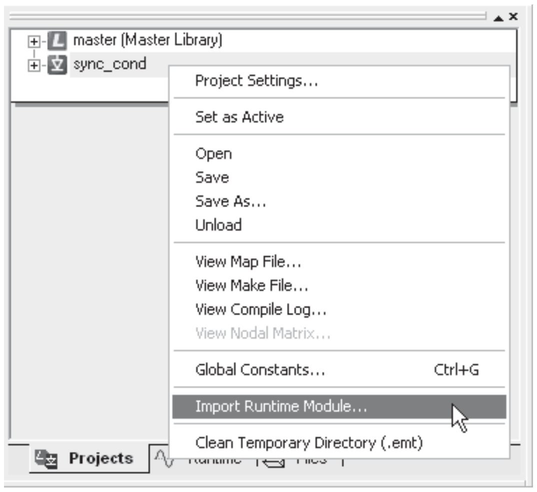
- **Component Definition 'xxx' not resolved**: This means that a custom component was used in PSCAD V2 but was not available in any of the library projects loaded in the PSCAD V4 Workspace. If you have previously converted your PSCAD V2 libraries into PSCAD V4 library projects, then simply ensure that 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.

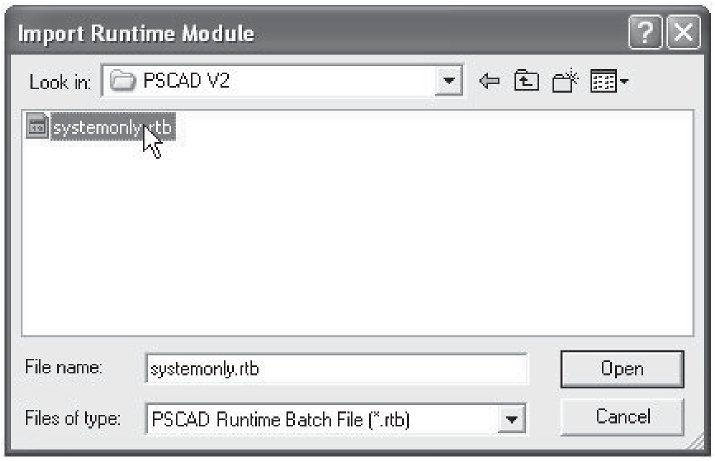
Importing Additional Runtime Batch Files

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:

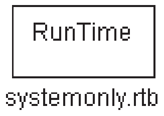
Once steps 1 to 4 above have been completed (i.e. the draft file has been successfully imported into PSCAD V4), right-click on the case filename in the Workspace and select *Import Runtime Module...*.



This will bring up the *Import Runtime Module* dialog window. Select the *.rtb file you wish to import and click the *Open* button.


Import Runtime Module
Look in: PSCAD V2
systemonly.rtb
File name: systemonly.rtb
Files of type: PSCAD Runtime Batch File (*.rtb)
Open
Cancel

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.



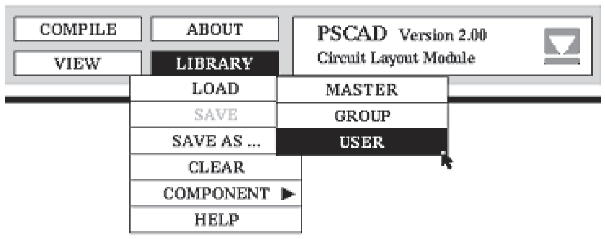
Importing V2 User Libraries

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.

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

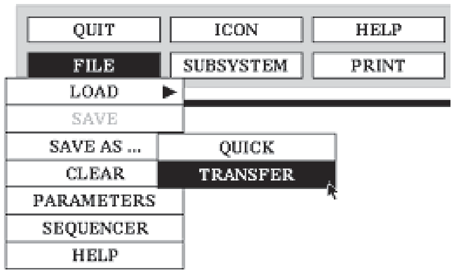
To transfer this palette into PSCAD V4:

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*.



2. Select the desired user library file from the Load User Library dialog and then press the *PROCEED* button.

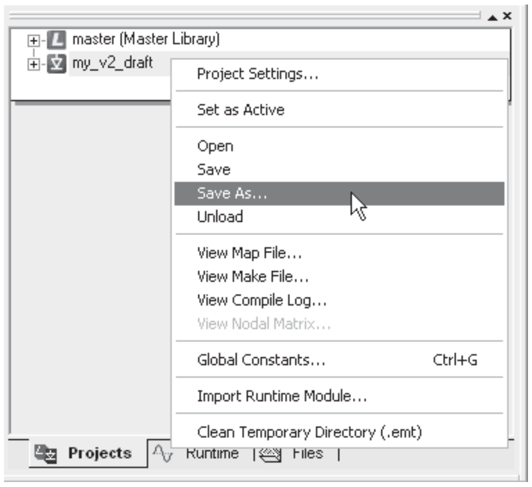
3. Copy the desired component instances from the user library palette over to the Draft canvas. From the *FILE* menu, select *SAVE AS... | TRANSFER* to save the Draft project as a transfer file (i.e. as a *.dfx file). This will ensure that the component definitions are included within the V2 draft file.



4. 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.

5. Follow Steps 1 to 4 in Importing V2 Draft and Runtime Batch Files to load the V2 draft project into PSCAD V4.

6. In the PSCAD V4 Workspace, right-click on the project filename and select *Save As...* from the pop-up menu.



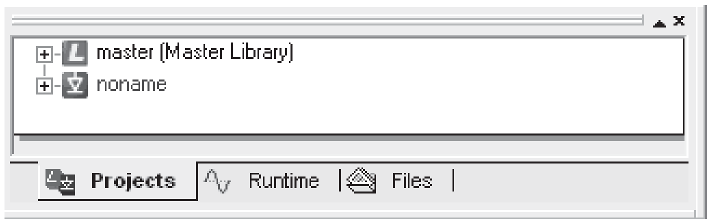
7. Rename the case project to a library project in the *File Name* field of the Save Project As dialog window. Press the *Save* button.

You now have converted your V2 user library to a PSCAD V4 library project!

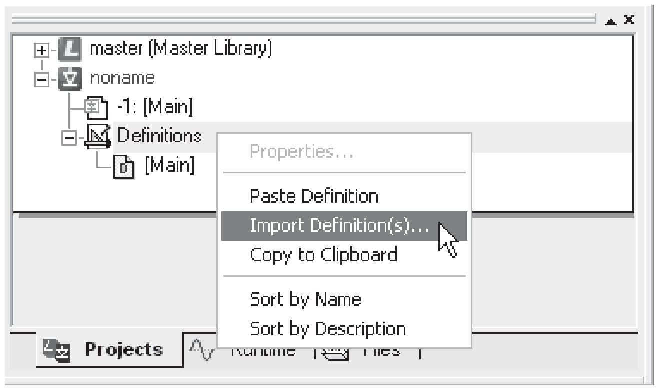
Importing Individual V2 Components
It is possible to import individual PSCAD V2 component definition files, directly into a PSCAD V4 library project. Once this is accomplished, an instance of the definition may be created and displayed within the library.

Please follow these steps to avoid problems:

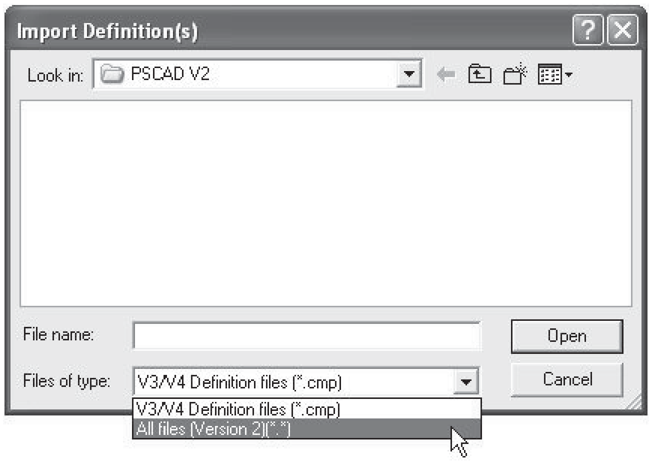
1. Create a new library project in PSCAD V4 (or open an existing library). To create a new library, select *File* | *New* | *Library*. A new library called 'noname' should appear in the Workspace window.



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)...*.

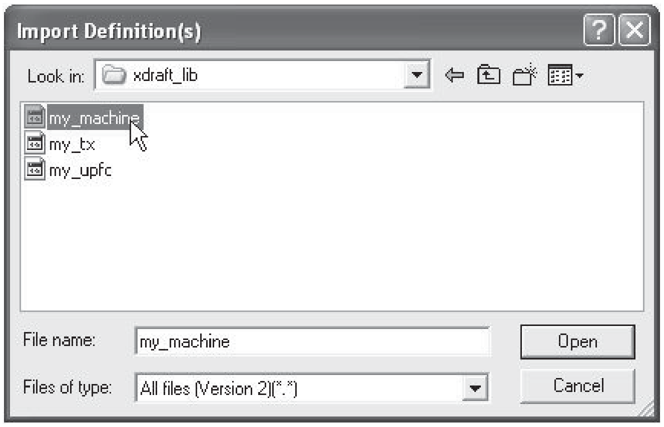


3. In the Import Definition(s) dialog window, change the *Files of Type* drop list to 'All Files'.


Import Definition(s)
Look in: PSCAD V2
File name:
Files of type: V3/V4 Definition files (*.cmp)
V3/V4 Definition files (*.cmp)
All files (Version 2)(*.*)
Open
Cancel

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.

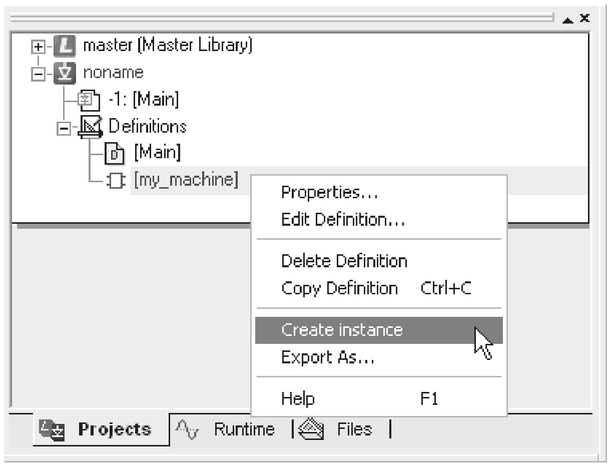
Do not select any PSCAD V2 macro files (*.g), library files (*.lib), or any file beginning with a 'period.'



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.

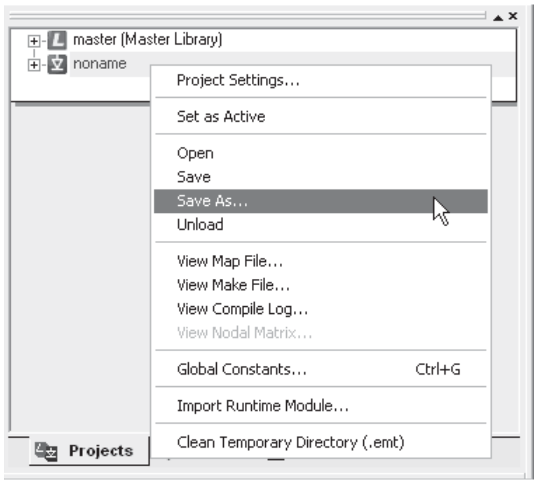
6. Check for warning or error messages in the Output window. If you receive an error message similar to "Macro file 'xxx.g' not found," you need to move all associated V2 macro files (*.g) from the '~/PSCAD/script' directory on your UNIX system to the same directory from where you are importing your 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. Too 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.

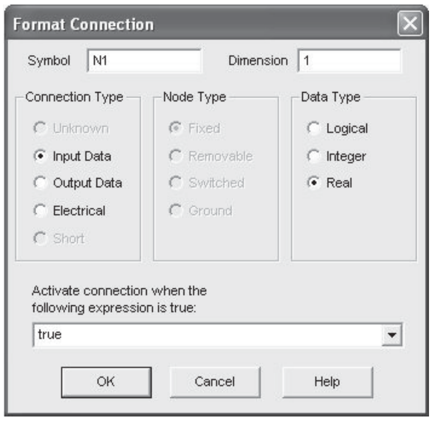
Import/Export Tags with Array Variables
Each module may also require editing if you used any import or export tags with array variables in the original PSCAD V2 draft project. The old import and export components did not contain any information regarding the dimension or type of variable (all import/exports 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.



Format Connection

Symbol: N1
Dimension: 1

Connection Type: ( ) Unknown [✓ Input Data] ( ) Output Data ( ) Electrical ( ) Short

Node Type: [✓ Fixed] ( ) Removable ( ) Switched ( ) Ground

Data Type: ( ) Logical ( ) Integer [✓ Real]

Activate connection when the following expression is true: true

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.

Index

Index