Additional Resources
PSS®E 34.8.2

February 2021

SIEMENS

Siemens Industry, Inc.
Siemens Power Technologies International
400 State Street
Schenectady, NY 12301-1058 USA
+1 518-395-5000
www.siemens.com/power-technologies

Copyright © 1997 - 2021 Siemens Industry, Inc., Siemens Power Technologies International

Information in this manual and any software described herein is confidential and subject to change without notice and does not represent a commitment on the part of Siemens Industry, Inc., Siemens Power Technologies International. The software described in this manual is furnished under a license agreement or nondisclosure agreement and may be used or copied only in accordance with the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, for any purpose other than the purchaser's personal use, without the express written permission of Siemens Industry, Inc., Siemens Power Technologies International.

PSS®E high-performance transmission planning software is a registered trademark of Siemens Industry, Inc., Siemens Power Technologies International in the United States and other countries.

The Windows 7 and Windows 10® operating systems, the Visual C++® development system, Microsoft Office Excel® and Microsoft Visual Studio® are registered trademarks of Microsoft Corporation in the United States and other countries.

Intel® Visual Fortran Compiler for Windows is a trademark of Intel Corporation in the United States and other countries.

The Python™ programming language is a trademark of the Python Software Foundation.

Other names may be trademarks of their respective owners.

Table of Contents

PSS®E Applications and Utilities 1
Overview 2
Auxiliary Program Descriptions 6
ACCC Post Processor 6
User Dynamics Data Table 6
Motor Parameters 7
IPLAN Programming Language Compiler 7
Line Properties 7
Eigenvalue and Eigenvector Calculations 8
Generation Cost Curves 8
PSS®E Plotting Tool 8
Transmission Constants 8
V Curves 8
Utilities Descriptions 9
Dynamics User Model DLLs 9
Compiling CONEC and CONET 10
User-Written Models 10
Incorporating User-Written Routines into IPLAN 11
Adding a New Activity to PSS®E 11
Creating a Dynamic Simulation DLL 11
Application Notes 15
Flecs Fortran-to-Fortran Translator 15
Conversion Program Descriptions 16
PSS®E Dynamics Data File Conversion to IEEE Dynamics FIle 16
IEEE Dynamics File Conversion to PSS®E Dynamics Data File 16
Generator Data 16
Excitation System Data 18
Governor Data 19
IEEE Power Flow Data Conversion 19
Bus Data 19
Generator Data 19
Transformer Data 19
Area Interchange Data 20
Zone Data 20
Legacy Drawing Coordinate Data File Conversion 20
Legacy Sequence Data File Conversion 20
Convert Old Power Flow Raw Data File 22
Create Power Flow Raw Data File for Use by Legacy PSS®E Releases 23
PECO Power Flow Conversion 24
Bus Data 24
Generator Data 24
Branch Data 25
Transformer Data 25
Area Interchange Data 25
Convert to New WECC Format 25
Convert New WECC File to PSS®E DYRE File 25
Convert New WECC File to PSS®E RAWD File 25
Deprecated Auxiliary Programs 26

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Printing 27
Printer Drivers: Microsoft and Siemens PTI 28
Siemens PTI Printing Parameter Files 29
PARMPR 29
Supplemental Printing Parameter Files 30
Printing 31
Graphics 31
Text 32
FLECS-to-Fortran Translator 35
FLECS User's Manual 36
Introduction 36
Retention of Fortran Features 36
Correlation of FLECS and Fortran Sources 37
Structured Statements 37
Indentation, Lines, and the Listing 39
Control Structures 41
Decision Structures 41
Internal Procedures 49
Additional Statements: EXITPROC/EXITLOOP/NEXTPASS 51
EXITPROC 51
EXITLOOP and NEXTPASS 52
Restrictions and Notes 52
Errors 54
Syntax Errors 55
Context Errors 55
Undetected Errors 55
Other Errors 56
FLECS Summary Sheet 57
Using FLECS for PSS®E 59
Introduction 59
Use of FLECS32 59
FLECS32 Options 59
Examples of Using FLECS32 62

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

List of Figures

PSS®E Applications Available from Start Menu 2
PSS®E Data Converters Available from Start Menu 4
Create User DLL Dialog 12
Sample PARMPR.DAT File 29

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

List of Tables

PSS®E Auxiliary Programs 2
PSS®E Utilities 3
PSS®E Conversion Programs 4
Createusrdll Application Options 12
IEEE Excitation System Types 18

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Chapter 1
PSS®E Applications and Utilities

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.1. Overview

The programs which comprise PSS®E are listed below. Those not otherwise marked are found in the PSSBIN subdirectory. You may have all or some of these programs, depending upon your particular license agreement.

Most of these programs are normally executed through shortcut icons (see Figure 1-1 and Figure 1-2); others, as noted below, are primarily intended to be executed from the Command Prompt. Note that all PSS®E programs may be executed from the Command Prompt if the user prefers.

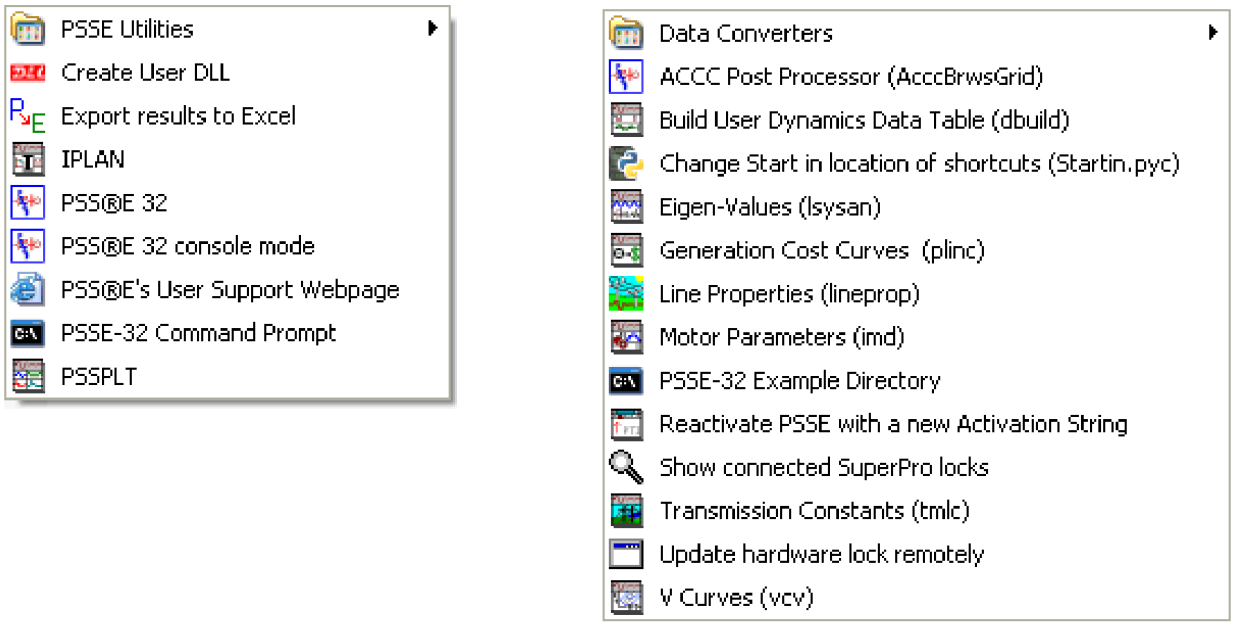


Figure 1.1. PSS®E Applications Available from Start Menu

Table 1.1. PSS®E Auxiliary Programs

Application [a]	Description
AcccBrwsGrid	The ACCC Post-processor is an application that reports the results of the PSS®E contingency screening analysis performed by the PSS®E AC Contingency Calculation.
DBUILD	Incorporates information on user-written plant-related models into the data file used by activity CCON.
IMD	Induction motor model data verification program.
IPLAN	Compiles programs written in the IPLAN programming language in preparation for their specification to the PSS®E activity EXEC.
LINEPROP	GUI program for calculating transmission line properties.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Application [a]	Description
LSYSAN	Linear System Dynamic Analysis program. Performs small disturbance dynamic analysis.
PLINC	Plots incremental cost curve data as contained in an Economic Dispatch Data File.
PSSPLT	Channel output file processing program.
TMLC	Calculates the transmission line constant data required by many power system analysis programs.
VCV	Plots generator V-curves; see PLINC and Generator Reactances and Saturation Data.

[a]These applications, when viewed in the PSSBIN directory, will have the current PSS®E version number appended to the name of the executable (*.exe) file. A corresponding .BAT file with the names listed in the table initiate the corresponding .exe file. In most applications the name of the .BAT file should be used.

Table 1.2. PSS®E Utilities

Application [a]	Description
ACTV	Reactivate one or more PSS®E programs without requiring reinstallation.
CHECKPDD	Check to see if the system driver for the Activator locks has been installed.
CMDLUSR	Command file to compile any number of FOR, F90, F, C, and CPP files. (Be sure to read the restrictions listed at the top of this file!)
Createusrdll	An application program used for creating a PSS®E dynamic simulation user dll. This combines the functions of compile.bat and cload4.bat of PSS®E dynamic simulation.
FLECS32	FLECS language pre-processor program.
PARSE32	Utility program used by CLOAD4.BAT
PARSEXT	Utility program used by CMDLUSR.BAT.
REACTPSSE32	Reactivate PSS®E with a new activation string (will not install any additional files).
REMOTEUP	Program for updating data stored in certain types of hardware locks.
ShowLockNum	Program to display the SuperPro lock number connected to the computer.
STARTIN	Command-Line Only program to set the Start In directory in the PSSLF4 and PSSDS4 shortcuts to the current directory.

[a]These applications, when viewed in the PSSBIN directory, will have the current PSS®E version number appended to the name of the executable (*.exe) file. A corresponding .BAT file with the names listed in the table initiate the corresponding .exe file. In most applications the name of the .BAT file should be used.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

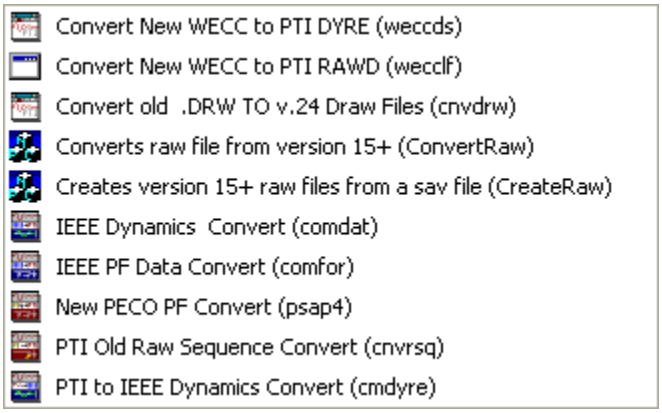


Figure 1.2. PSS®E Data Converters Available from Start Menu

Table 1.3. PSS®E Conversion Programs

Application [a]	Description
CMDYRE	From a PSS®E Dynamics Data File and a Machine Impedance Data File, builds a file containing card image records in the IEEE format for the exchange of stability data.
COMDAT	From a file containing card image records in the IEEE format for the exchange of stability data, builds a PSS®E Dynamics Data File and a Machine Impedance Data File.
COMFOR	From a file containing power flow data records in IEEE Common Tape Format, builds a PSS®E-24 Power Flow Raw Data File.
CNVDRW	From a Drawing Coordinate Data File in the format required for PSS®E-15 through PSS®E-23, builds a Drawing Coordinate Data File in the form required by PSS®E-24 or later.
CNVRSQ	From a Sequence Data File in the format required for PSS®E-21 or earlier, builds a Sequence Data File in the form required by PSS®E-22 through PSS®E-26.
CONVERTRAW	Reads any previous PSS®E Power Flow Raw Data File produced by PSS®E-15 or later, and builds a PSS®E Power Flow Raw Data File in any later format.
CREATERAW	Reads a PSS®E Saved Case File, and builds a PSS®E Power Flow Raw Data File compatible with PSS®E-15 or later.
PSAP4	From a file containing card image records in the PJM PSAP Version 4 or 5 power flow program data format, builds a PSS®E-23 Power Flow Raw Data File.
RAWECC	Converts PSS®E saved case format to Western Electricity Coordinating Council (WECC) format.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Application [a]	Description
WECCDS	Converts Western Electricity Coordinating Council (WECC) dynamics format to PSS®E DYRE.
WECCLF	Converts Western Electricity Coordinating Council (WECC) power flow program data format to PSS®E RAWD format.

[a]These applications, when viewed in the PSSBIN directory, will have the current PSS®E version number appended to the name of the executable (*.exe) file. A corresponding .BAT file with the names listed in the table initiate the corresponding .exe file. In most applications the name of the .BAT file should be used.

The following sections describe each auxiliary program and references an execution time parameter file, along with the name of its parameter file (see PSS®E Program Operation Manual, Section 3.3.3: Program Run-Time Option Settings):

`IMD (WINIMD.PRM) PSSPLT (WINPLT.PRM)`

The remaining auxiliary programs do not reference an execution time parameter file.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.2. Auxiliary Program Descriptions

1.2.1. ACCC Post Processor

AcccBrwsGrid

The auxiliary program AcccBrwsGrid is a Windows® application that post-processes the ACCC Solution Output file (*.acc) and provides spreadsheet-style reports of these results. This application is documented in its own Help window.

1.2.2. User Dynamics Data Table

DBUILD

The auxiliary program DBUILD constructs the binary data file used by the plant-related model data changing activity, CCON. In addition to information on the standard models that are supplied with PSS®E, DBUILD allows the user to include in the binary data file information on user-written plant-related models being used in his system model.

DBUILD first processes the source file containing data on the PTI-supplied plant-related models. It then instructs the user to enter the name of a data file that contains records describing user-written models. This file is then processed and the request for a user file is again made. This cycle is repeated until a zero or simply a carriage return is entered in response to the request for a filename.

Information on each model is contained on a series of records in the file. The first record for each model block is of the form:

```
$ name,NC,NI,'desc'
```

where:

$	Is contained in column one.
name	Is the name of the model.
NC	Is the number of CONs used by the model.
NI	Is the number of ICONs used by the model.
desc	Is model descriptive text.

This is followed by NC records describing the NC CONs used by the model in the same order as they appear on the model's data sheet, followed by NI records describing the model's NI ICONs in the same order as they appear on the data sheet. Each of these records is of the form:

```
NW,ND,'text'
```

where:

NW	Is the field width.
ND	Is the number of digits to the right of the decimal point.
text	Is up to 52 characters of descriptive text for this constant.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Currently, the 'desc' field on the first record and the NW and ND items on the remaining data records are ignored; these data items may used in a future update.

Following is an example for the block of records which may be used for the model CSVGN4:

```
$ CSVGN4,11,1,'Static Shunt Compensator'
  13,4,'J      K'
  13,4,'J+1    T1'
  13,4,'J+2    T2'
  13,4,'J+3    T3 (> 0)'
  13,4,'J+4    T4'
  13,4,'J+5    T5'
  13,4,'J+6    RMIN (Reactor Minimum MVAR)'
  13,4,'J+7    VMAX'
  13,4,'J+8    VMIN'
  13,4,'J+9    CBASE (Capacitor MVAR)'
  13,4,'J+10   VOV (Override Voltage)'
  13,4,'M      IB , Remote Bus To Reg., 0 To Reg. Term. Volt.'
```

1.2.3. Motor Parameters

IMD

The auxiliary program IMD plots induction motor torque, current and power factor versus slip or speed. These values may also be presented in tabular form. IMD may also calculate and report motor conditions for specified values of terminal voltage, speed, motor base, and system base.

Initial estimates of model data may be provided in a data file that contains a DYRE format data record for one of the models CIM5BL, CIMTR1, CIMTR2, CIMTR3, or CIMTR4. Model parameters in equivalent circuit form may then be modified interactively. IMD outputs equivalent circuit parameter data as used by CIM5BL; output may be either in tabular form or in the form of a dynamics data input record for use by activity DYRE.

Machine manufacturers typically supply the user with a set of induction machine curves and/or a partial set of machine parameters. Users must estimate the remaining equivalent circuit parameters. IMD is used to plot and/or print induction motor characteristics based on available data and user estimates. Curves may be compared to those supplied by the manufacturer and parameters may be modified interactively. This process is repeated until a set of plots reasonably match those supplied by the manufacturer.

The *PSS®E Program Application Guide* contains additional details on motor modeling in PSS®E and the use of IMD.

1.2.4. IPLAN Programming Language Compiler

IPLAN

The auxiliary program IPLAN compiles programs written in the IPLAN programming language in preparation for their specification to the PSS®E activity EXEC. IPLAN is documented in the *PSS®E IPLAN Program Manual*.

1.2.5. Line Properties

LINEPROP

The auxiliary program LINEPROP calculates electrical parameters for overhead transmission and distribution lines. It is described in the *PSS®E Line Properties Calculator Manual*. LINEPROP is supplied to those PSS®E users whose installation includes the Transmission Line Characteristics Program section of PSS®E. PC lessees

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

who receive LINEPROP also receive the TMLC program and its documentation, *Transmission Line Characteristics (TMLC) Program Manual*. LINEPROP is replacing the older TMLC program and should be used in lieu of TMLC. TMLC will be dropped in a future release.

1.2.6. Eigenvalue and Eigenvector Calculations

LSYSAN

The auxiliary program LSYSAN is used to perform small disturbance dynamic analysis. LSYSAN is supplied to those installations whose lease includes the Linear Dynamic Analysis Program section and is documented in the *PSS®E Program Application Guide*.

1.2.7. Generation Cost Curves

PLINC

The auxiliary program PLINC plots incremental heat rate curves from an Economic Dispatch Data File of the form used by activity ECDI (see PSS®E Program Operation Manual, Section 5.32: Performing Unit Commitment and Economic Dispatch). Curves for up to four machines may be plotted on each page.

The dialog of PLINC is self-explanatory.

1.2.8. PSS®E Plotting Tool

PSSPLT

The auxiliary program PSSPLT processes PSS®E dynamic simulation Channel Output Files. It is documented in the *PSSPLT Program Manual*.

1.2.9. Transmission Constants

TMLC

The auxiliary program TMLC calculates the transmission line constant data required by many power system analysis programs. It is described in the *Transmission Line Characteristics (TMLC) Program Manual*. TMLC is supplied to those PSS®E users whose installation includes the Transmission Line Characteristics Program section of PSS®E. PC lessees who receive TMLC also receive the LineProp program and its documentation, *PSS®E Line Properties Calculator Manual*. TMLC is being replaced by LINEPROP and will be dropped in a future release.

1.2.10. V Curves

VCV

The auxiliary program VCV calculates full load EFD for a user specified set of machine parameters, terminal voltage and machine loading. Optionally, the generator V curves may then be plotted. VCV accepts machine parameters as used in the PSS®E generator models GENROE, GENROU, GENSAE, and GENSAL.

Machine manufacturers typically supply the user with a set of synchronous machine V curves and/or a partial set of machine parameters. Users must estimate the remaining parameters. VCV is used to plot V curves based on available data and user estimates. Curves can be compared to those supplied by the manufacturer and parameters can be improved interactively. This process is repeated until a set of plots reasonably match those supplied by the manufacturer. This approach is particularly useful if the machine's saturation constants are suspect.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.3. Utilities Descriptions

1.3.1. Dynamics User Model DLLs

PSS®E advanced features involving user-written code are implemented through the use of user-written dynamically linked libraries (DLLs). When the user starts PSS®E, the program searches for the user-written DLL by name. These library names are:

DSUSR.DLL	This library may contain CONEC, CONET, and/or user-written dynamics models.
IPLUSR.DLL	This library may contain user-defined routines for IPLAN (see IPLAN Program Manual, Section 3.28: Graphics).
PSSUSR.DLL	This library may contain a user-written USERAC subroutine which will be called the USER activity.

Consequently, if you have a customized DSUSR.DLL (created by using CLOAD4) in your directory, and you start Dynamics with C:\WORKING\ONE set as your working directory, your custom DSUSR will be loaded instead of the default. If, on the other hand, you start Dynamics C:\WORKING\TWO set which does not contain a customized DSUSR.DLL, the default copy will be loaded from PSSLIB.

Default copies of all of the user-replaceable DLLs, DSUSR, DSUSRNEW, PSSUSR, PSSUSRNEW, IPLUSR, and IPLUSRNEW are stored in PSSLIB. If you wish to permanently change the default for one of these DLLs, you may, for example, create a new DSUSR.DLL file by running activity DYRE, compiling your new CONEC.FLX and CONET.FLX, running CLOAD4, and then copying the resulting DSUSR.DLL file into the PSSLIB directory. Once placed there, your new copy will become the default for all future executions of Dynamics.

If your results are not what you expect, first make certain that you are loading the correct copies of the DLL(s) you wish to use.

If you create a customized DLL for the use of PSS®E, you can create a corresponding icon or shortcut in order to use that DLL. The new icon or shortcut should identify as the working directory the directory that contains the customized DLL. Alternatively, if you choose to start programs from the Command Prompt, simply change directories into your working directory before starting the program and the correct DLL will be loaded.

Instead of creating new shortcuts, it is possible to change the working directory (i.e., the Start In directory) of the existing PSS®E-32 shortcuts. PSS®E-32 includes a program called STARTIN, which should be run from the PSS®E-32 Command Line. When executed, this program (after prompting) will change the working directories of all standard (noncustom) shortcuts associated with PSS®E-32 to be the current directory (i.e., the directory from which the STARTIN command was given.) The STARTIN command may be used to switch the working directories as frequently as desired. To return the working directories to their original settings, use the shortcut "PSSE-32 Example Directory" under the PSS®E Utilities menu, to bring up a command prompt and give the STARTIN command.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Compiling CONEC and CONET

The compiling file generated by activities DYRE and SRRS for compiling the connection subroutines CONEC and CONET is in the form of a BATCH file. The name of this file must have the extension ".BAT". If no extension is specified when this file is created, the extension ".BAT" is automatically appended to the filename. The compiling file is executed by entering the command:

```
filename.bat
```

which will compile the CONEC and CONET subroutines. The compilation must be run from the Command Prompt. Once the compilation is complete, you must execute CLOAD4 at the Command Prompt to create a new DSUSR.DLL file.

In addition to compiling CONEC.FLX and CONET.FLX, the PSS®E-generated COMPILE.BAT file will allow a single user written-model to be compiled at the same time. If you have such a model, which must be written in the FLECS language, simply specify it on the COMPILE command line, e.g.:

```
COMPILE MY_MODEL.FLX
```

The result will be a correctly compiled MY_MODEL.OBJ, ready to process with the CLOAD4 step.

The CMDLUSR command can be used if you have more than one user-written model, or if some of the files are not in FLECS. This command will compile any number of FLX, FOR, F90, F, C, and CPP files at one time. For example, the command:

```
CMDLUSR MY_MODEL.FLX EXTRA1.FLX EXTRA2.FOR
```

will FLECS and compile MY_MODEL.FLX and EXTRA1.FLX, and then compile EXTRA2.FOR. The object files MY_MODEL.OBJ, EXTRA1.OBJ and EXTRA2.OBJ should be generated in the current directory.

When compiling FLX, FOR, F90, and F files, CMDLUSR will search the current directory and the PSSLIB directory in an attempt to find any INCLUDEd files or MOD files. If other directories are to be searched, make sure that your INCLUDE environment variable defines those extra directories.

When compiling C and CPP files, only the current directory will be automatically searched for #include files. Therefore, you will almost certainly need to set your INCLUDE variable appropriately before compiling such files!

The top section of the CMDLUSR.BAT file discusses the INCLUDE variable in more detail.

User-Written Models

The CLOAD4 linking procedures create a custom DSUSR.DLL, which allows for inclusion of user-written models in the dynamics program. The CLOAD4 command will automatically link in the CONEC.OBJ and CONET.OBJ files. Up to 35 additional object files may be specified on the CLOAD4 command line, for example:

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

```
CLOAD4 MYMDL.OBJ E:\PSSE32W\PSSLIB\MOREMDLS.OBJ
```

The result of running CLOAD4 is a new DSUSR.DLL.

Incorporating User-Written Routines into IPLAN

CLIPLU provides a means for incorporating user-written Fortran routines into IPLAN and PSSLF4 and PSSDS4. If your license includes IPLAN, nine dummy files (USREX1.FOR through USREX9.FOR) are placed in the IPLUSR subdirectory of PSS®E. You may edit these files to incorporate your own routines and, in addition, you may include routines on the command line as with CLOAD4. CLIPLU will compile and link these routines to create a new IPLUSR.DLL.

Adding a New Activity to PSS®E

CLPSSUSR enables the user to include a new activity in PSS®E. By editing USERAC.FOR (found in the PSSLIB subdirectory of PSS®E) and then running CLPSSUSR, you can compile and link your new activity into a new PSSUSR.DLL. Starting PSS®E from the directory containing PSSUSR.DLL will automatically include the new user function.

Creating a Dynamic Simulation DLL

Createusrdll

Createusrdll is an application program used for creating a dynamic simulation user dll. This combines the functions of compile.bat and cload4.bat. You can launch the program from the desktop icon created at PSS®E installation or from the Windows® *Start* menu.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

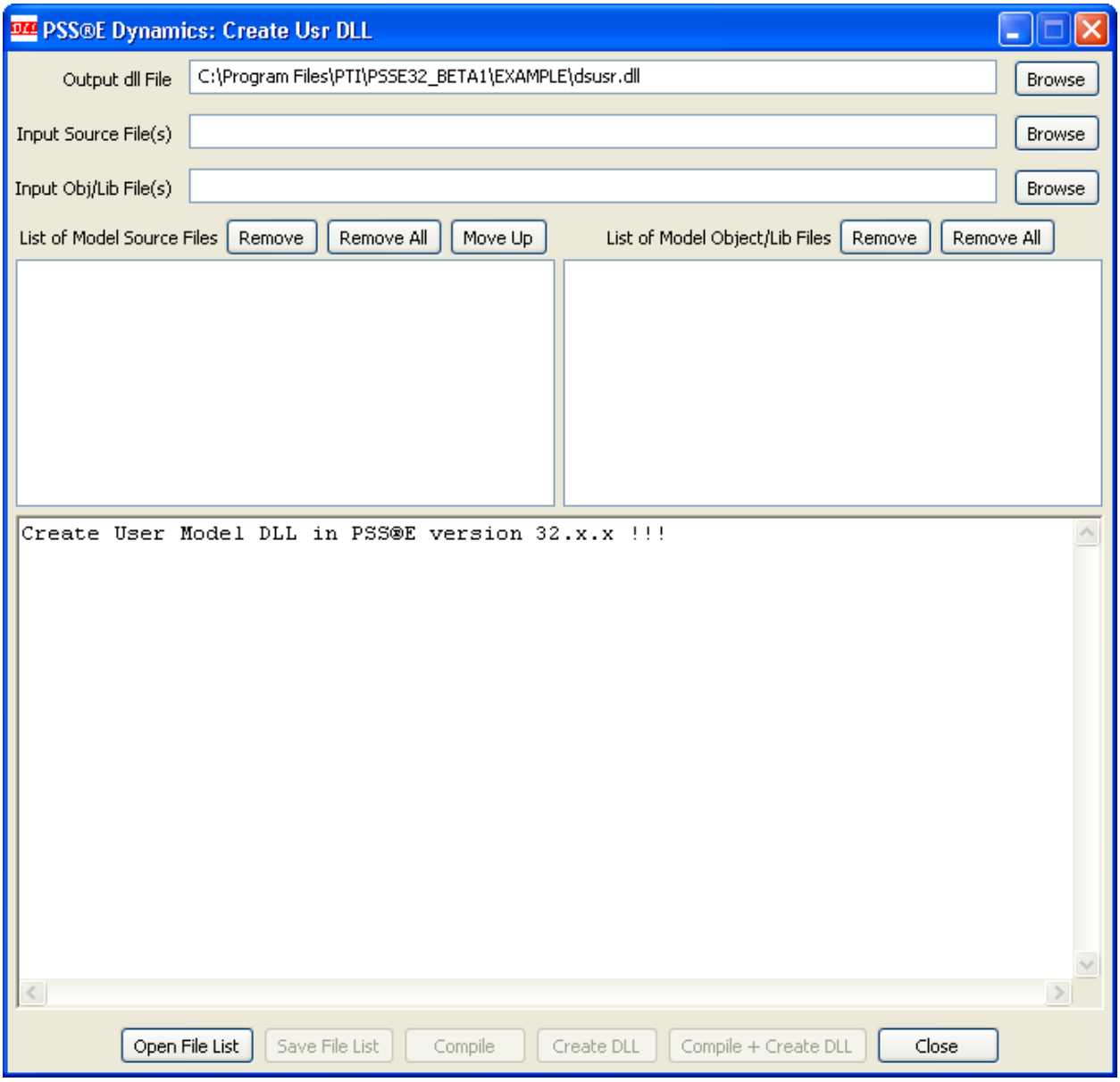


Figure 1.3. Create User DLL Dialog

Table 1.4. Createusrdll Application Options

Option	Description
Output dll File	The name of the dll file to be created. The default name is *dsusr.dll*. However, it can be any name. To edit the dll name, enter the dll filename or select the dll filename using [Browse] button.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Option	Description
	If the name is anything other than 'dsusr.dll', then after opening PSS®E, but before running dynamic simulation, users have to load this dll using the *addmodellibrary* API. For example, if a user dll called myusrmodel.dll is created in the work_dir folder, this dll can be loaded using the psspy module as follows: `ierr = psspy.addmodellibrary(r"c:\work_dir\myusrmodel.dll")`
Input Source File(s)	The FORTRAN or FLX source filenames. Use one of the following methods to add source files. • If the names of all the source files that would be needed to create the user DLL were saved in a text file, use *Open File List* and select the text filename. • Select one or more files using [Browse] button. • Enter the filename. Addition of a source file by one of these methods will populate the left side box titled *List of Model Source Files*.
Input Object/Library File(s)	The successful compilation of the source files, corresponding "object" codes are automatically added to *List of Model Object/Lib Files*. Additionally, other pre-compiled (i.e., existing) object/library files can be added to the list using one of the following methods. • Object/Library files from input text file when opened using *Open File List*. • Select one or more files using [Browse] button. • Enter the filename. This will populate the right side box titled *List of Model Object/Lib Files*.
Remove and Remove All	These controls are used for removing Source or Object/Lib Files from Selection. The following methods can be used to remove files from either the *List of Model Source Files*, or the *List of Object/Lib Files*.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Option	Description
	• Select a file and use *Remove* button to remove selected file. • Use *Remove All* to remove all files. • Double click a filename to remove selected file. When a file is removed from *List of Model Source Files* the corresponding file from the *List of Model Object/Lib Files* is not removed.
Move Up	Moves one or more source files up the *List of Model Source Files*. When a FORTRAN source file is a FORTRAN module file, this file needs to compiled first, so that the *.mod* created by compiler is available for other FORTRAN source files during their compilation. To move a file up, select the appropriate source file in *List of Model Source Files* and use *Move Up* button to bring the selected file ahead of another FORTRAN file.
Compile	Compiles all the files in *List of Model Source Files*, and adds the resulting objects to *List of Model Object/Lib Files*.
Create DLL	This creates a dll by linking all the files in *List of Model Object/Lib Files*. The dll thus created will be stored in the file whose name was specified in *Output dll File*.
Compile+Create DLL	Compiles all the files in *List of Model Source Files*, adds objects to *List of Model Object/Lib Files*, and creates a dll by linking all the files in *List of Model Object/Lib Files*.
Open File List	Specification of the user dll filename, the source files to be compiled, and the names of object/library files (collectively called *File List*) that are needed for creating the user dll can be done via GUI as explained above. Having specified the files needed for creating the dll, *File List* can be saved in a text file using the *Save File List*. Alternatively the text file containing *File List* can be created using a text editor in a format as given in Application Notes.
Save File List	This saves the source filenames, object/library filenames and dll name (i.e., *File List*) to a text file. This file then can be opened using *Open File List* button. The file is saved in a format described in Application Notes.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Application Notes

The format of Input Text File is as given below.

```
# File: mymodelfiles.txt

# Input to "createusrdll" program.

# Any text on a line after "#" is treated as comment.

# The file has three sets data records, with each record title as:

# [dll name], [source filenames] and [object/lib filenames].

# Provide only one dll name in [dll name] record.

# Provide each source filename (.flx, .for, .f90, .f) on a separate line in
[source filenames]

# record.

# Provide each object (.obj) or library (.lib) on a separate line in [source
filenames] record.


[dll name]

c:\work_dir\myusrmodel.dll


[source filenames]

c:\work_dir\mymodel_module.for

c:\work_dir\mymodel.for

c:\work_dir\mycontroller.flx


[object/lib filenames]

c:\work_dir\mymodel.dll

c:\work_dir\pssewind.lib

c:\work_dir\myother.obj
```

1.3.2. Flecs Fortran-to-Fortran Translator

FLECS32

Refer to Section 3.2 Using FLECS for PSS®E for details.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.4. Conversion Program Descriptions

1.4.1. PSS®E Dynamics Data File Conversion to IEEE Dynamics FIle

CMDYRE

The auxiliary program CMDYRE32 reads a Dynamics Data file (*.dyr) and its corresponding Machine Impedance Data file (*.rwm) and outputs the data in IEEE dynamics data format [1] . See PSS®E Program Operation Manual, Section 14.1.1: Dynamics Model Raw Data File Contents and Section 5.4.1: Machine Impedance Data File Contents, respectively, for content information.

Through a brief dialog, the user is instructed to designate the two input files and the output file. Progress messages may be directed either to the user's terminal or written to a message file.

The user is given the option of including or omitting data records for machines designated as out-of-service in the Machine Impedance Data File. If such machines are included, their power fractions are set to zero on their output file data records.

The user has the opportunity to designate the system base MVA and to specify descriptive comment lines to be included in the output file.

Not all PSS®E models correspond to models in the IEEE format. Any model references which are ignored or approximated are tabulated at the message device.

For machines for which XTRAN is nonzero in the Machine Impedance Data File, the same bus number and name is included on the generator data record as the terminal and high voltage side buses. It is the user's responsibility to reconcile the bus identification with the power flow case in the target program.

1.4.2. IEEE Dynamics File Conversion to PSS®E Dynamics Data File

COMDAT

The auxiliary program COMDAT reads a data file in IEEE dynamics data format[1] and outputs the data in the form of a Dynamics Data file (*.dyr) and its corresponding Machine Impedance Data file (*.rwm). See PSS®E Program Operation Manual, Section 14.1.1: Dynamics Model Raw Data File Contents and Section 5.4.1: Machine Impedance Data File Contents, respectively, for content information.

Through a brief dialog, the user is instructed to designate the input file and the two output files. Program messages may be directed either to the user's terminal or written to a message file.

The user is given the option of upgrading classical and transient level machine representations to subtransient level models; details on this conversion are given below.

Upon completion of COMDAT, the user should review and reconcile any program messages before reading the data into PSS®E.

The following assumptions and data changes are made in converting the stability data into PSS®E format.

Generator Data

If the generator model designation is not specified, COMDAT assumes the following: [2]

[1] *The IEEE dynamics data format is defined in Procedures for the Exchange of Power Plant and Load Data for Synchronous Stability Studies*, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-100, No. 7 July 1981, pp. 3229-3245.

[2] All time constants are entered in seconds, and all reactances are entered in per unit on a machine MVA base and machine rated terminal voltage.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Initially, 2.2 IEEE model

but, set to 2.1 IEEE model if $T'_{qo} = 0.0$ or $X'_q = 0.0$ pu

then set to 1.1 IEEE model if $T'_{do} = 0.0$ s.

COMDAT assumes the following PSS®E generator models:

GENROU	For the 2.2 and 2.2M IEEE model designations.
GENSAL	For the 2.1 and 2.1M IEEE model designations.
GENCLS	For the 0.0, 1.0 and 1.1 IEEE model designations if the model upgrade option is not selected.
GENROU	For the 0.0, 1.0 and 1.1 IEEE model designations if the model upgrade option is selected; typical exciter and governor model data is also assigned.

In upgrading classical or transient analysis models to GENROU, COMDAT assumes:

S1 = 0.11

S2 = 0.48

If $T''_{do} < 0.04$, $T''_{do} = 0.04$ s.

If $T''_{qo} < 0.06$, $T''_{qo} = 0.06$ s.

If $X_d \leq 0.0$:

For machines with MVA $\leq$ 100.0, $X_d = 8.0 * X'_d$ pu

For machines with MVA > 100.0, $X_d = 6.0 * X'_d$ pu

For machines with MVA $\leq$ 300.0:

$T'_{do} = 6.0$ s.

$X_q = 0.96 * X_d$ pu

If $T'_{qo} \leq 0.0$ or $T'_{qo} > 2.0$, $T'_{qo} = 1.0$ s.

For machines with MVA > 300.0:

$T'_{do} = 5.0$ s.

$X_q = 0.95 * X_d$ pu

If $T'_{qo} \leq 0.0$ or $T'_{qo} > 2.0$, $T'_{qo} = 0.5$ s.

If $X''_d \leq 0.0$, $X''_d = 0.7 * X_d$ pu

If $X_l \leq 0.0$ or $X_l > X''_d$, $X_l = 0.67 * X''_{d\ pu}$

If $X'_q \leq 0.0$, $X'_q = 2.5 * X''_{d\ pu}$

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

If $X'_q > X_q$, $X'_q = 0.5*X_{q\ pu}$

For IEEE "2.n" models (i.e., GENSAL and GENROU), COMDAT assumes:

If $X''_d \leq 0.0$, $X''_d = (X'_d + X_l)/2.0$ pu

If $X''_d > X'_d$, $X''_d = 0.75*X'_d$ pu

If $X_l \leq 0.0$ or $X_l > X''_d$, $X_l = 0.67*X''_d$ pu

If $T''_{do} \leq 0.04$, $T''_{do} = 0.04$ s.

If $T''_{qo} \leq 0.06$, $T''_{qo} = 0.06$ s.

In addition, for GENSAL models, COMDAT assumes:

If $X_d > 15.0*X'_d$, $X_d = 4.5*X'_d$ pu

If $X_d \leq X_q$ or $X_q \leq 0.0$, $X_q = 0.6*X_d$ pu

and for GENROU models, COMDAT assumes:

If $X_d > 15.0*X'_d$, $X_d = 8.0*X'_d$ pu

If $X_d \leq X_q$ or $X_q \leq 0.0$, $X_q = 0.96*X_d$ pu

If $X'_q \leq 0.0$, $X'_q = 2.5*X''_d$ pu

If $X'_q > X_q$, $X'_q = 0.5*X_q$ pu

If $X'_d > X'_q$:

For machines with MVA <300.0, $X'_q = 2.0*X'_d$ pu

For machines with MVA > 300.0, $X'_q = 1.5*X'_d$ pu

Excitation System Data

The following table summarizes the PSS®E models used for each of the IEEE excitation system types. For each exciter model, if either RC or XC is nonzero, a record for the PSS®E model IEEEVC is generated.

Table 1.5. IEEE Excitation System Types

IEEE Type	PSS®E Model
DC1	IEEEX1
DC2	EXDC2
DC3	IEEEX4
AC1	EXAC1
AC2	EXAC2
AC3	EXAC3
AC4	EXAC4
ST1	EXST1
ST2	EXST2

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

IEEE Type	PSS®E Model
ST3	EXST3

Governor Data

The IEEE type one, two, and three speed governing models are handled as the PSS®E models IEEEG1, IEEEG2, and IEEEG3, respectively. For the type one model, the following data changes are made:

If $P_{max} = 0.0$, $P_{max} = 1.0$ pu on turbine MW rating or generator MVA rating

If $U_o = 0.0$, $U_o = 1.0$

If $U_c = 0.0$, $U_c = -1.0$

1.4.3. IEEE Power Flow Data Conversion

COMFOR

The auxiliary program COMFOR reads a data file in IEEE power flow data format [3] and outputs the data in the form of a PSS®E-24 Power Flow Raw Data File (see PSS®E Program Operation Manual, Section 5.2.1: Power Flow Raw Data File Contents). Only the 132 column format is recognized.

Through a brief dialog, the user is instructed to designate the input and output files. Program messages may be directed either to the user's terminal or written to a message file.

The user is given the option of specifying a "generator netting file". If specified, this file contains bus numbers, entered one per line in free format, of buses whose generation is to be netted with their loads. Such buses become PSS®E Type 1 buses.

The following assumptions and data changes are made in converting the Common Format data into PSS®E format.

Bus Data

All single quotes (') in bus names are changed to dashes (-).

Generator Data

IEEE Type 0 buses with nonzero generation are set to Type 2 buses with fixed generator output.

IEEE Type 1 buses are set to fixed generator output Type 2 buses. If the desired voltage (V_{sched}) is zero, the scheduled voltage is set to the average of the voltage limits.

If $V_{sched} < 0.0$,

If final voltage > 0.0, V_{sched} = final voltage

If final voltage < 0.0, $V_{sched} = 1.0$ pu on machine terminal voltage base

Transformer Data

If step size < 0.0,

[3] *The IEEE power flow data format is defined in* Common Format for Exchange of Solved Load Flow Cases, IEEE Transactions on Power Apparatus and Systems, Vol. PAS-92, No. 6 November/December 1973, pp. 1916-1925.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

If Type 4 transformer (phase shifter), STEP = 1.25°

Otherwise, STEP = 0.00625 pu on bus voltage base

For voltage controlling transformers, if (VMA-VMI) < STEP,

VMI = VMA - STEP

VMA = VMA + STEP

For flow controlling transformers, if VMA < VMI,

VMI = VMA - 2.5 MW/Mvar

VMA = VMA + 2.5 MW/Mvar

If RMA < RMI,

RMI = RMA - 4.0*STEP

RMA = RMA + 4.0*STEP

Area Interchange Data

All single quotes (') in area names are changed to dashes (-).

Zone Data

All single quotes (') in zone names are changed to dashes (-).

1.4.4. Legacy Drawing Coordinate Data File Conversion

CNVDRW

The auxiliary program CNVDRW reads a PSS®E-15 through PSS®E-23 Drawing Coordinate Data File and outputs data in the Drawing Coordinate Data File format required by the current release of PSS®E (see PSS®E Program Operation Manual, Section C.4.1: Drawing Coordinate Data File Contents).

Through a brief dialog, the user is instructed to designate the input and output files.

All data records are copied verbatim except for data records describing loads. The interpretation of the option field on the "LO" record has been changed, and "LP", "LC" and "LY" records have been introduced (see PSS®E Program Operation Manual, Load Records - LO, LP, LC, and LY). In addition, the load symbol is larger than it was at PSS®E-23 and earlier releases of PSS®E in order to accommodate the load identifier when an individual load is being drawn. The endpoint coordinates on load records are modified so that the end of each load symbol lies at the same location that it did before.

Following the data conversion process, another pair of data files may be specified.

1.4.5. Legacy Sequence Data File Conversion

CNVRSQ

The auxiliary program CNVRSQ reads a PSS®E-21 (or earlier) Sequence Data File and outputs the data in the Sequence Data File format required by PSS®E-22 through PSS®E-26.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Through a brief dialog, the user is instructed to designate the input and output files.

All data records are copied verbatim except for data records describing zero sequence mutual couplings on which the geographical "B" factors are converted to conform to their more flexible definitions introduced in PSS®E-22 (see PSS®E Program Operation Manual, Zero Sequence Mutual Impedance Data).

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.5. Convert Old Power Flow Raw Data File

CONVERTRAW

The auxiliary program CONVERTRAW reads a PSS®E Power Flow Raw Data File in any of the formats required by PSS®E-15 through PSS®E-31, and outputs data records in the Power Flow Raw Data File format required by a selected PSS®E release later than that of the input file. The following format conversions are supported:

Input File Format	*Output File Format*
PSS®E-15 through PSS®E-23	PSS®E-24 or later
PSS®E-24 through PSS®E-26	PSS®E-27 or later
PSS®E-27 and PSS®E-28	PSS®E-29 or later
PSS®E-29	PSS®E-30 or later
PSS®E-30	PSS®E-31 or later
PSS®E-31	PSS®E-32

The user is instructed to designate the Power Flow Raw Data File input and output filenames, as well as the PSS®E revisions of the input and output files. The user also indicates if any of the input file records use extended bus names as bus identifiers. The output records are then written.

The "old" format data file is expected to be in the form required by activity READ or READ,NAME; specifically, the first three records are assumed to be the Case Identification Data records.

This auxiliary program can be run from GUI (Start>Programs>PTI>PSSE Utilities>Data Converters>ConvertRaw) or from command prompt as below:

CONVERTRAW INNAME INVERSTR NUMNAM OUTNAME OUTVERSTR

where:

Character	INNAME*260	Is the name of Input Power Flow Raw Data file (input; no default allowed).
Character	INVERSTR*12	Is the version number corresponding to the format of INNAME (input; no default allowed). INVERSTR is in the format of a PSS®E release number. Example: If INNAME format is of PSS®E-29.5.1: INVERSTR = ’29’ = ’29.5’ = ’29.5.1’
Integer	NUMNAM	*Is the flag for bus number or name specification on input records*

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

		(input; no default allowed). *NUMNAM = 0 bus numbers.* *NUMNAM = 1 bus names.*
Character	OUTNAME*260	Is the name of Output Power Flow Raw Data file (input; no default allowed).
Character	OUTVERSTR*12	Is the version number corresponding to the format of OUTNAME (input; no default allowed). PSS®E release number corresponding to OUTVERSTR must be later thanINVERSTR PSS®E release number.

1.5.1. Create Power Flow Raw Data File for Use by Legacy PSS®E Releases

CREATERAW

The auxiliary program CREATERAW reads a Saved Case File written by the PSS®E activity CASE of PSS®E-16 or later, and outputs data in the Power Flow Raw Data File format required by a release of PSS®E which preceded the current release. CREATERAW can output data records in formats compatible with the following PSS®E releases:

- PSS®E-15 through PSS®E-23
- PSS®E-24 through PSS®E-26
- PSS®E-27 and PSS®E-28
- PSS®E-29
- PSS®E-30
- PSS®E-31

The user is instructed to designate the input and output filenames, as well as the PSS®E revision to be used for the Power Flow Raw Data File records. The output records are then written.

If the specified Saved Case File was written by PSS®E-30 or later and contains any six digit bus numbers, and a Raw Data file prior to PSS®E-30 is to be created, an appropriate error message is printed and CREATERAW is terminated.

If the Saved Case File specified to CREATERAW was written by PSS®E-30 or later, control modes for switched shunts are left at their current values.

This auxiliary program can be run from GUI (Start>Programs>PTI>PSSE Utilities>Data Converters>CreateRaw) or from command prompt as below:

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

CREATERAW INNAME OUTNAME OUTVERSTR

where:

Character	INNAME*260	Is the name of Input Power Flow Saved Case file (input; no default allowed).
Character	OUTNAME*260	Is the name of Output Power Flow Raw Data file (input; no default allowed).
Character	OUTVERSTR*12	Is the version number corresponding to the format of OUTNAME (input; no default allowed). OUTVERSTR is in the format of a PSS®E release number. Example: If INNAME format is of PSS®E-29.5.1: INVERSTR = '29' = '29.5' = '29.5.1'

1.5.2. PECO Power Flow Conversion

PSAP4

The auxiliary program PSAP4 reads a data file in the format of the PJM Power System Analysis Package (PSAP) power flow program version 4 or 5, and outputs the data in the form of a PSS®E-23 Power Flow Raw Data file. This file may be converted to a file in the current PSS®E Power Flow Raw Data File format (*.raw) using the auxiliary program CONVERTRAW. See PSS®E Program Operation Manual, Section 5.2.4: Subsystem READ.

Through a brief dialog, the user is instructed to designate the input and output files. Program messages may be directed either to the user's terminal or written to a message file.

The following assumptions and data changes are made in converting the PSAP data into PSS®E format.

Bus Data

All single quotes (') in bus names are changed to dashes (-).

If the bus voltage VM = 0.0, VM = 1.0 pu on bus voltage base

Base voltages are decoded from the last four characters of the 12-character bus name. If a value greater than 999 is decoded, the last three characters are used. If a valid numerical value is not found in the final four, three, or two characters, the bus base voltage is set to zero.

Generator Data

Nonregulating buses with generation are set to Type 2 buses with fixed generator output.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Branch Data

Impedance is converted from "MVA BASE" or percent to per unit.

Charging is converted from percent to per unit.

BUSTIEs are given an impedance of j0.0001 pu on system bus voltage and MVA base.

Transformer Data

If phase shift angle is nonzero:

Phase shift angle is negated

If TAP = 0.0, TAP = 1.0 pu

If second line data card is read and TAP = 0.0, TAP = 1.0

For voltage controlling transformers, when no second line data card is entered, the tap step is set to 0.00625 per unit on system bus voltage base. The desired voltage band is set to the desired voltage setpoint +0.00625 per unit if a nonzero controlled bus is specified, and to 0.9 pu through 1.1 pu if no controlled bus is specified. When the second line data card is entered, the designated voltage band is used and the tap step is determined from the ratio limits and the number of tap positions available.

For MW controlling phase shifters, the MW flow limits are set to the desired MW flow +2.5 MW. Phase shift angle limits are negated and interchanged.

For Mvar controlling phase shifters, the Mvar flow limits are set to the desired Mvar flow +2.5 Mvar. The tap step is determined from the ratio limits and the number of tap positions available.

Area Interchange Data

All single quotes (') in area names are changed to dashes (-).

1.5.3. Convert to New WECC Format

RAWECC

1.5.4. Convert New WECC File to PSS®E DYRE File

WECCDS

1.5.5. Convert New WECC File to PSS®E RAWD File

WECCLF

The auxiliary programs WECCLF, RAWECC, and WECCDS, which convert data between the WECC power flow and stability program data formats and PSS®E data input file formats, are described in the *PSS®E–WECC Data Conversion Manual*.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

1.6. Deprecated Auxiliary Programs

The conversion capabilities of the following programs are now comprehensively covered by CONVERTRAW and CREATERAW and will be removed in a future release of PSS®E. The CONVERTRAW and CREATERAW programs should be used in lieu of the individual conversion programs listed here.

CNV27	From a Power Flow Raw Data File in the format required for PSS®E-24 through PSS®E-26, builds a Power Flow Raw Data File in the form required by PSS®E-27 or PSS®E-28, and then provides for the conversion to PSS®E-27 or later of any corresponding Sequence Data File which is in the form required for PSS®E-22 through PSS®E-26.
CNV29	From a Power Flow Raw Data File in the format required for PSS®E-27 or PSS®E-28, builds a Power Flow Raw Data File in the form required by PSS®E-29.
CNV30	From a Power Flow Raw Data File in the format required for PSS®E-29, builds a Power Flow Raw Data File in the form required by PSS®E-30.
CNV31	From a Power Flow Raw Data File in the format required for PSS®E-30, builds a Power Flow Raw Data File in the form required by PSS®E-31.
CNV32	From a Power Flow Raw Data File in the format required for PSS®E-31, builds a Power Flow Raw Data File in the form required by PSS®E-32.
RAW23	From a PSS®E Saved Case File, builds a PSS®E-23 Power Flow Raw Data File.
RAW26	From a PSS®E Saved Case File, builds a PSS®E-26 Power Flow Raw Data File.
RAW28	From a PSS®E Saved Case File, builds a PSS®E-28 Power Flow Raw Data File.
RAW29	From a PSS®E Saved Case File, builds a PSS®E-29 Power Flow Raw Data File.
RAW30	From a PSS®E Saved Case File, builds a PSS®E-30 Power Flow Raw Data File.
RAW31	From a PSS®E Saved Case File, builds a PSS®E-31 Power Flow Raw Data File.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Chapter 2
Printing

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

2.1. Printer Drivers: Microsoft and Siemens PTI

Printing from PSS®E is controlled primarily by Microsoft Windows® printer drivers. The Windows drivers, including display drivers, operate independently of PSS®E. The Windows default printer driver is selected or changed using the Windows Control Panel. Refer to your Microsoft Windows manuals for further documentation on selecting device drivers. Siemens PTI does not supply Microsoft display or printer drivers for its Windows products. Contact Microsoft or your device vendor for drivers.

In a *very* small number of cases, you may wish to use a Siemens PTI graphics driver. If so, the details are described in Siemens PTI Printing Parameter Files. Siemens PTI spool device drivers operate by referencing the primary PSS®E printing parameter file, PARMPR.DAT. When a Siemens PTI output device is selected, PARMPR.DAT is accessed (along with supplemental parameter files) and its user-specified printing parameters are applied to the output. In the following sections, all PSS®E parameter files are referred to without their ".DAT" file extensions.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

2.2. Siemens PTI Printing Parameter Files

PSS®E printing parameter files provide you with flexibility in the routing and formatting of both text and graphics output. They are located in the subdirectory PSSPRM. If PSS®E is installed on a network drive, the parameter files may be made available to all PSS®E users. If a network user has special needs, parameter files may be copied to the user's working directory, HOME directory, or other local directory and modified accordingly (be sure the PSSPRM setting in PSSE3200.INI contains the proper path). Parameter files are ASCII text files and can be modified with any text editor.

2.2.1. PARMPR

PARMPR is the printing parameter file referenced by PSS®E. PARMPR contains instructions to PSS®E on how, where, and in what format to send output to a printer, plotter, or file. These instructions are referred to as printing parameters. For example, you could customize such printing characteristics of a Siemens PTI spool device as start and end strings, banner pages, queue types, and page orientation.

Below is a sample PARMPR file with five printer entries including comment lines. Each entry in the file contains one or more parameter assignments in the following form:

```
parameter_name device_code = new_value
```

where:

parameter_name	Name of the parameter.
device_code	Number used to specify the device this parameter modifies. T = Terminal (device code 1). F = File (device code 2). 1 = Printer 1 (device code 3). 2 = Printer 2(device code 4). 3-20 = Alternate spool devices (device code 6).
new_value	Is the value assigned to the parameter.

Figure 2.1. Sample PARMPR.DAT File

Printer #1 is reserved for access to the default Microsoft printer and should not be redefined.

Printer #1 contains the keyword, WIN-PRINTER, to access the Microsoft Windows default printer. This entry uses Microsoft drivers. All other entries use Siemens PTI spool device drivers.

Printers #2 and #3 are different entries for the same HP Laserjet Series II printer attached to LPT1. The start string for Printer #2 indicates a portrait page orientation and condensed print. Printer #3 includes a start string indicating a landscape page orientation and condensed print.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Printer #4 is a QMS 800 II PostScript printer on COM2. Before using, be sure COM2 is set properly via the MODE command.

Printer #5 is a STAR NX-1000 dot-matrix printer on LPT1. An END_STRING is used to give a form-feed (12) at the end of the print job.

The beginning-of-job and end-of-job strings for a PostScript printer are specified in the PSCRIPT.DAT file. Use of such strings is HIGHLY recommended.

Comment lines may be used anywhere in the parameter files. The format for comment lines is:

```
! string
```

```
* string
```

where *string* can be any sequence of characters.

If any modifications are made to PARMPR while PSS®E is running, the program must be restarted to implement the changes. Refer to the Guide to Printing and Plotting for detailed descriptions of the available parameters for PARMPR.

When used for graphics output, the <default> PostScript printer must have PSPLOT listed as one of its name "aliases."

2.2.2. Supplemental Printing Parameter Files

In addition to PARMPR, there are several supplemental printing parameter files used to further define specific characteristics of PSS®E output. They are:

PARMPS	Used for graphics printing to PostScript printers.
PARMPW	Used for graphics printing controlled by Microsoft drivers (does not read PARMPR, since it's not a Siemens PTI spool device).
PSCRIPT	Used for text printing to PostScript printers.

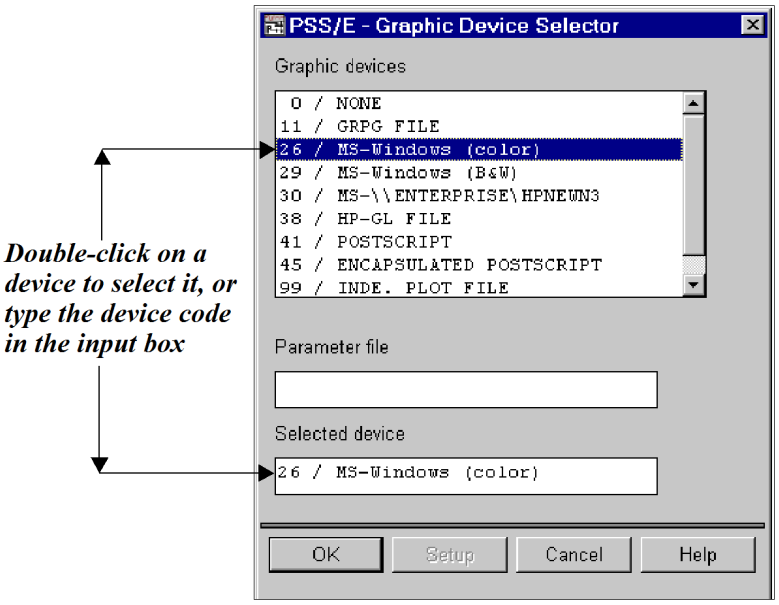
These parameter files are discussed in detail under the sections where they apply. The parameters available for PARMPS, PARMPW, and PSCRIPT are located in the Guide to Printing and Plotting along with instructions on their usage.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

2.3. Printing

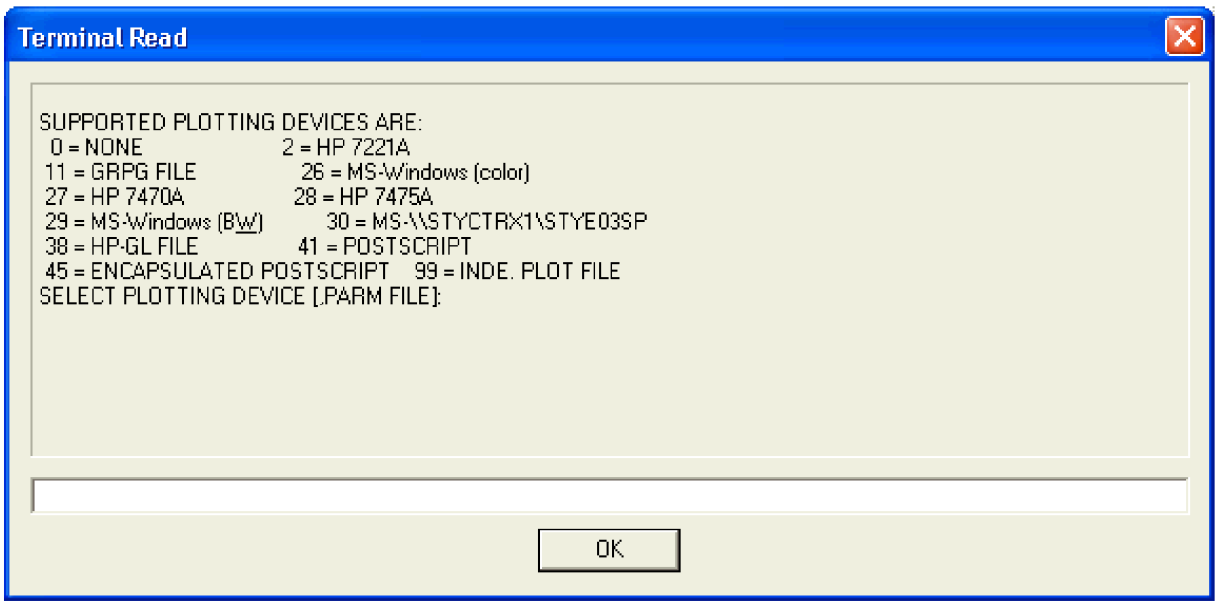
2.3.1. Graphics

When a graphics activity such as activity GRPG is selected from a PSS®E program, a Graphic Device Selector dialog box is displayed prompting you to choose the destination of the output. A list of available output device names is displayed with their corresponding device codes. Note also that an optional parameter file containing directives similar to those in PARMPW, may be supplied in the box provided.



OR

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

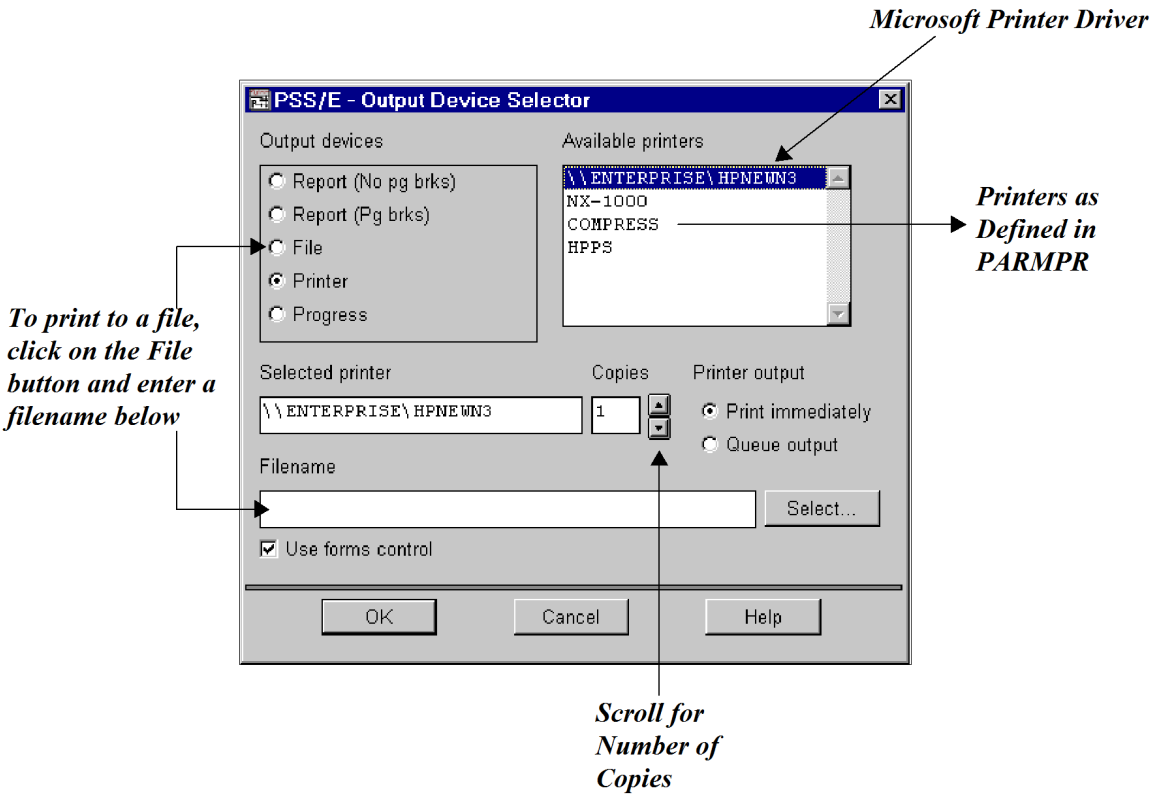


0/None	Cancel printing; no output.
11/GRPG FILE	Outputs to a file in GRPG format.
26/MS-Windows (color)	Outputs to the screen in color. Uses the default Windows display driver.
29/MS-Windows (B&W)	Outputs to the screen in black and white. Uses the default Windows display driver.
30/MS-\\ENTERPRISE\HPCENTR2	Outputs to the printer selected as the default printer under Windows, which may be (as in this case) a network device.
38/HP-GL FILE	Outputs to a file in HP-GL format.
41/POSTSCRIPT	Siemens PTI Spool Device. Sends output to a PostScript printer as defined in PARMPR. Also reads PARMPS.
45/ENCAPSULATED POSTSCRIPT	Same as 41, except an encapsulated PostScript file is generated. (Note: No "preview" bitmap is included.)
99/INDEPLOT	Outputs in Siemens PTI "INDEPLOT" format.

2.3.2. Text

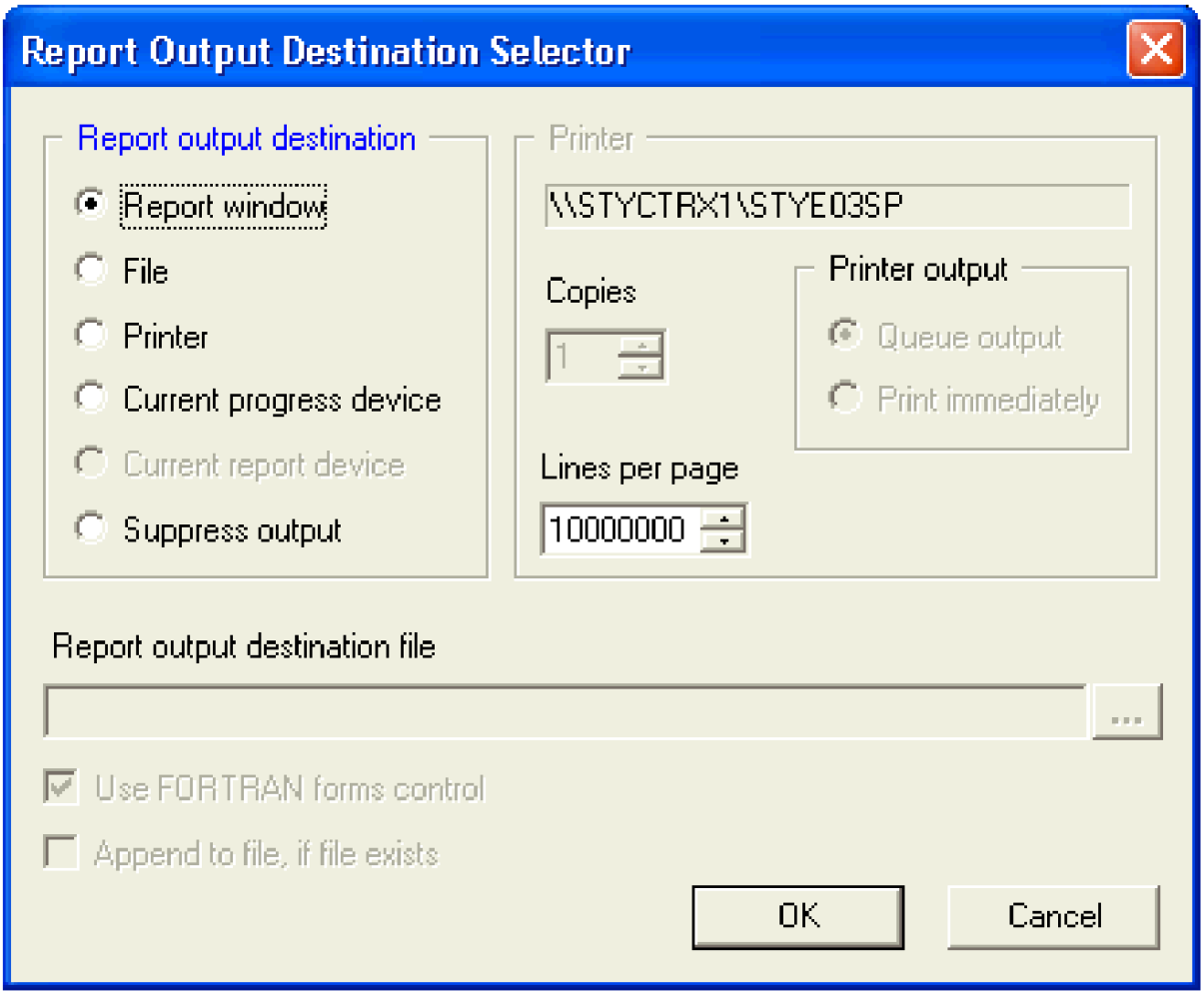
The IO CONTROL,OPEN dialog box displays a list of devices for text output. Available printers, both Microsoft and Siemens PTI driven, are listed, too.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International



OR

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International



Report	Outputs to the screen in a PSS®E Reports Window.
File	Outputs to a file. Must specify a filename. If the TEMP Directory is not set in PSS3200.INI, output goes to the Windows TEMP file.
Printer	Outputs to a Microsoft-driven printer if available, or to a printer as defined in PARMPR (Siemens PTI spool device). Also reads the PSCRIPT parameter file which is used to override the default PostScript translation characteristics.
Progress	Outputs to the screen in the PSS®E Progress window.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Chapter 3
FLECS-to-Fortran Translator

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

3.1. FLECS User's Manual

This FLECS general user's manual was originally written by Terry Beyer at the University of Oregon. It has been updated by Wayne F. B'Rells at Power Technologies International to describe features which have been added to the FLECS language.

3.1.1. Introduction

Fortran v66 contains four basic mechanisms for controlling program flow: CALL/RETURN, IF, DO, and various forms of the GO TO.

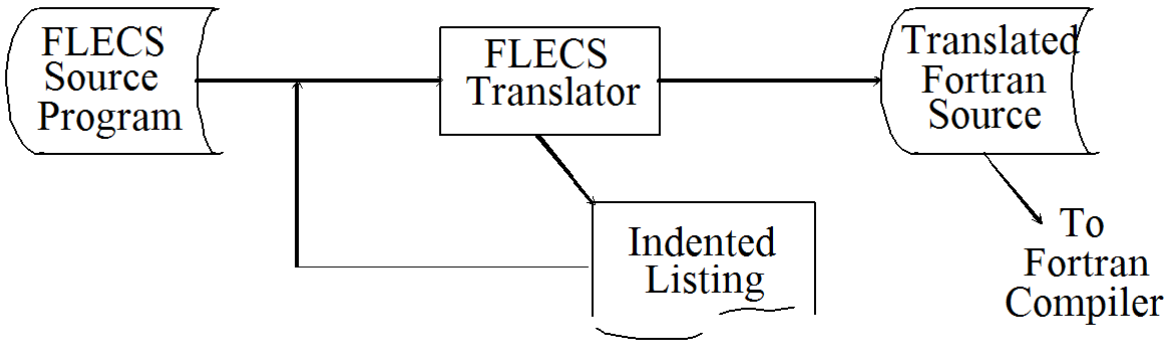
FLECS is a language extension of Fortran which has additional control mechanisms. These mechanisms make it easier to write Fortran by eliminating much of the clerical detail associated with constructing Fortran programs. FLECS is also easier to read and comprehend than Fortran.

This manual is intended to be a brief but complete introduction to FLECS. It is not intended to be a primer on FLECS or structured programming. The reader is assumed to be a knowledgeable Fortran programmer.

For programmers to whom transportability of their programs is a concern, it should be noted that the FLECS translator source code is in the public domain and is made freely available. The translator was written with transportability in mind and requires little effort to move from one machine to another. The version of FLECS described in this manual has been implemented on machines on which Siemens PTI software is supported.

The manner of implementation is that of a preprocessor which translates FLECS programs into Fortran programs. The resulting Fortran program is then processed in the usual way. The translator also produces a nicely formatted listing of the FLECS program which graphically presents the control structures used. FLECS will accept this nicely formatted listing file as input, so only one form of a FLECS source program needs be maintained.

The following diagram illustrates the translating process.



3.1.2. Retention of Fortran Features

The FLECS translator examines each statement in the FLECS program to see if it is an extended statement (a statement valid in FLECS but not in Fortran). If it is recognized as an extended statement, the translator generates the corresponding Fortran statements. If, however, the statement is not recognized as an extended statement, the translator assumes it must be a Fortran statement and passes it through unaltered. Thus, the FLECS system does not restrict the use of Fortran statements, it simply provides a set of additional statements which may be used. In particular, GO TOs, arithmetic IFs, CALLs, arithmetic statement functions, and any other Fortran statement, compiler dependent or otherwise, may be used in a FLECS program.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

3.1.3. Correlation of FLECS and Fortran Sources

One difficulty of preprocessor systems like FLECS is that error messages which come from the Fortran compiler must be related back to the original FLECS source program. This difficulty is reduced by allowing the placement of line numbers (not to be confused with Fortran statement numbers) on FLECS source statements. These line numbers then appear on the listing and in the Fortran source. When an error message is produced by either the FLECS translator or the Fortran compiler, it will include the line number of the offending FLECS source statement, making it easy to locate on the listing.

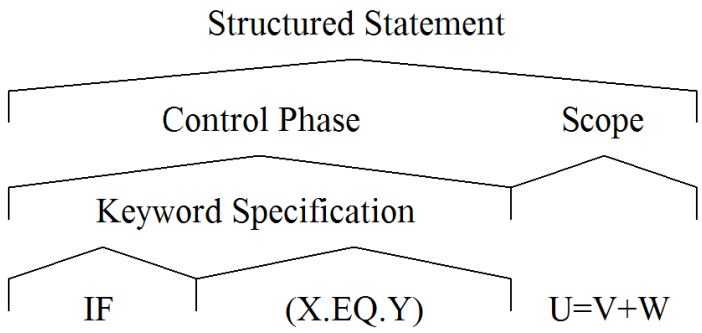
If the programmer chooses not to supply line numbers, the translator may be instructed to assign sequential numbers and place them in the last five columns of the listing and Fortran output files. Thus, errors from the compiler may still be related to the FLECS listing.

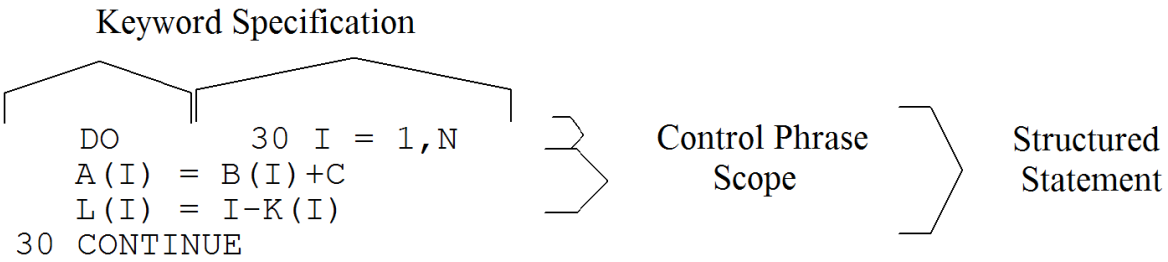
The beginning FLECS programmer should discover and make special note of the details of the mechanism by which Fortran compiler error messages may be traced back to the FLECS listing on the system being used.

3.1.4. Structured Statements

A basic notion of FLECS is that of the structured statement which consists of a control phrase and its scope. FORTRAN 66 has two structured statements, the logical IF and the DO.

The following examples illustrate this terminology:





Each structured statement consists of a control phrase which controls the execution of a set of one or more statements called its scope. Also note that each control phrase consists of a keyword plus some additional information called the specification. A statement which does not consist of a control phrase and a scope is

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

said to be a simple statement. Examples of simple statements are assignment statements, subroutine CALLs, arithmetic IFs, and GO TOs.

In FLECS there is a uniform convention for writing control phrases and indicating their scopes. To write a structured statement, the keyword is placed on a line beginning in Column 7 followed by its specification enclosed in parentheses. The remainder of the line is left blank. The statements comprising the scope are placed on successive lines. The end of the scope is indicated by a FIN statement. This creates a multiline structured statement.

Examples of multiline structured statements:

```
IF (X.EQ.Y)
   U = V+W
   R = S+T
   FIN
```

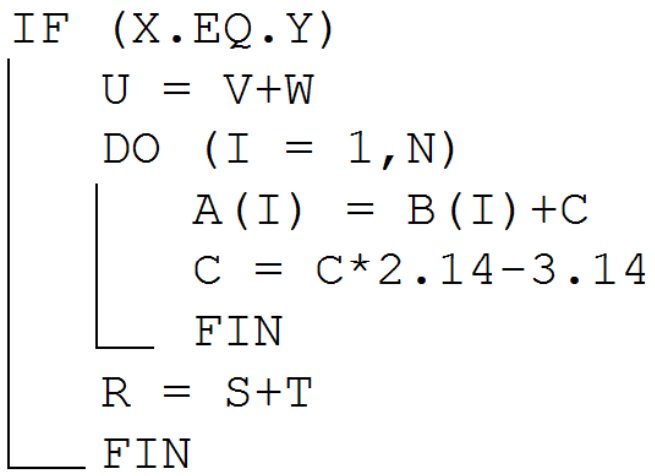
The FLECS form of the IF statement will generally override the Fortran IF statement. Therefore, FLECS programs should use only the FLECS form of IF statements! See x-refsection\ (Restrictions and Notes) for more details.

```
DO (I = 1,N)
   A(I) = B(I)+C
   C = C*2.14-3.14
   FIN
```

The statement number has been eliminated from the DO specification since it is no longer necessary, the end of the loop being specified by the FIN.

Nesting of structured statements is permitted to any depth. Example of nested structured statements:

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International



When the scope of a control phrase consists of a single simple statement, it may be placed on the same line as the control phrase and the FIN may be dispensed with. This creates a *one-line structured statement*.

Examples of one-line structured statements:

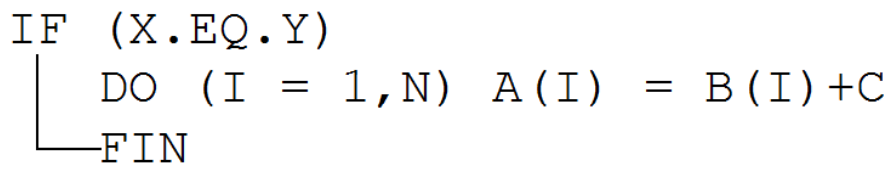
```
IF (X.EQ.Y) U = V+W DO (I = 1,N) A(I) = B(I)+C
```

Since each control phrase must begin on a new line, it is not possible to have a one-line structured statement whose scope consists of a structured statement.

Example of invalid construction:

```
IF (X.EQ.Y) DO (I = 1,N) A(I) = B(I)+C
```

To achieve the effect desired above, the IF must be written in a multiline form. Example of valid construction:



In addition to IF and DO, FLECS provides several useful structured statements not available in Fortran. After a brief excursion into the subject of indentation, we will present these additional structures.

3.1.5. Indentation, Lines, and the Listing

In the examples of multiline structured statements above, the statements in the scope were indented and an 'L' shaped line was drawn connecting the keyword of the control phrase to the matching FIN. The resulting graphic effect helps to reveal the structure of the program. The rules for using indentation and FINs are quite simple and uniform. The control phrase of a multiline structured statement always causes indentation of

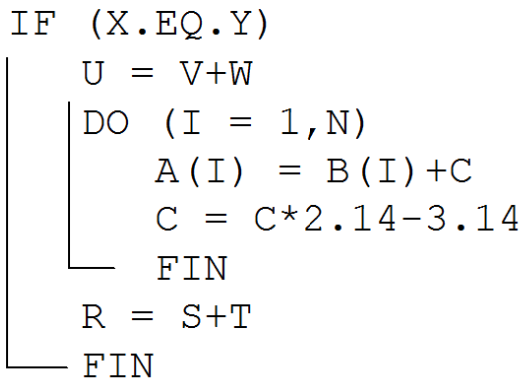
All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

the statements that follow. Nothing else causes indentation. A level of indentation (i.e., a scope) is always terminated with a FIN. Nothing else terminates a level of indentation.

When writing a FLECS program on paper, the programmer should adopt the indentation and line drawing conventions shown below. When preparing a FLECS source program in machine readable form, however, each statement may begin in Column 7. When the FLECS translator produces the listing, it will reintroduce the correct indentation and produce the corresponding lines. The programmer may introduce his or her own indentation if he or she is careful to insert blanks and .'s in exactly the format used by FLECS. This approach is sometimes useful when making corrections to existing FLECS programs and there is no need to generate a new listing file (which, if generated, would presumably replace the original input file as the FLECS source file).

Example of indentation:

1. Program as written on paper by programmer.



2. Program as entered into computer. Program could also be entered as shown in step 3.

```
IF (X.EQ.Y)
U = V+W
DO (I = 1,N)
A(I) = B(I)+C
C = C*2.14-3.14
FIN
R = S+T
FIN
```

3. Program as listed by FLECS translator.

```
IF (X.EQ.Y)
.  U = V+W
.  DO (I = 1,N)
.  .  A(I) = B(I)+C
.  .  C = C*2.14-3.14
.  ...FIN
.  R = S+T
...FIN
```

The correctly indented listing is a tremendous aid in reading and working with programs. Except for the dots and spaces used for indentation, the lines are listed exactly as they appear in the "raw" source program. That

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

is, the internal spacing of columns is preserved. Once a file has been processed by FLECS the first time there will seldom be any need to refer to a straight listing of the unindented source. Comment lines are treated in the following way on the listing to prevent interruption of the dotted lines indicating scope. A comment line which contains only blanks in Columns 2 through 6 will be listed with Column 7 indented at the then-current level of indentation as if the line were an executable statement. If, however, one or more nonblank characters appear in Columns 2 through 6 of a comment line, it will be listed without indentation. Blank lines may be inserted in the source and will be treated as empty comments.

In-line comments may appear on any line in a FLECS program. The beginning of an in-line comment is signalled by an exclamation mark (!). That is, neither an exclamation mark nor any characters following it are passed to the Fortran output file. If an actual exclamation mark is needed in a FLECS program it may be entered by placing two exclamation marks together (i.e., "!!" will result in a single "!" being passed to the Fortran output file).

3.1.6. Control Structures

The complete set of control structures provided by FLECS is given below together with their corresponding flow charts. The symbol "L" is used to indicate a logical expression. The symbol "S" is used to indicate a scope of one or more statements. Some statements, as indicated below, do not have a one-line construction. A convenient summary of the information in this chapter may be found in x-refsection\

Decision Structures

Decision structures are structured statements which control the execution of their scopes on the basis of a logical expression or test.

IF

Description: The IF statement causes a logical expression to be evaluated. If the value is true, the scope is executed once and control passes to the next statement. If the value is false, control passes directly to the next statement without execution of the scope.

General Form:

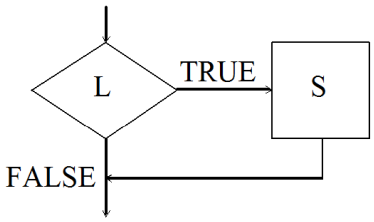
```
IF (L) S
```

Examples:

```
IF (X.EQ.Y) U = V+W

IF (T.GT.0.AND.S.LT.R)
.   I = I+1
.   Z = 0.1
...FIN
```

Flow Chart:



UNLESS

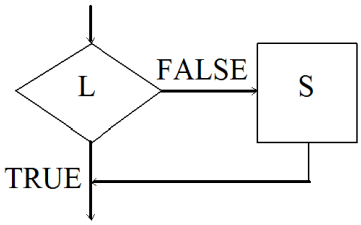
Description: "UNLESS (L)" is functionally equivalent to "IF(.NOT.(L))", but is more convenient in some contexts.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

```
UNLESS (L) S
```

Flow Chart:



Examples:

```
UNLESS (X.NE.Y) U = V+W

UNLESS (T.LE.0.OR.S.GE.R)
.   I = I+1
.   Z = 0.1
...FIN
```

WHEN...ELSE

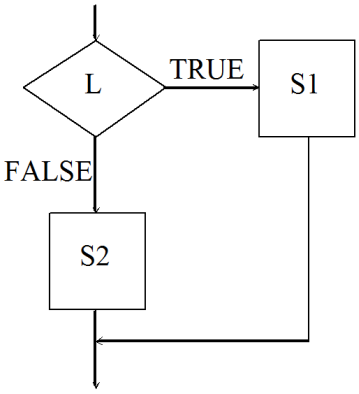
Description: The WHEN...ELSE statements correspond to the IF...THEN...ELSE statement of FORTRAN 77, PL/1, Pascal, etc. In FLECS, both the WHEN and the ELSE act as structured statements although only the WHEN has a specification. The ELSE statement must immediately follow the scope of the WHEN. The specifier of the WHEN is evaluated and exactly one of the two scopes is executed. The scope of the WHEN statement is executed if the expression is true and the scope of the ELSE statement is executed if the expression is false. In either case, control then passes to the next statement following the ELSE statement.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

```
WHEN (L) S1
ELSE S2
```

Flow Chart:



Examples:

```
WHEN (X.EQ.Y) U = V+W ! Sample in-line
ELSE U = V-W          ! comment

WHEN (X.EQ.Y)
.   U = V+W
.   T = T+1.5
...FIN
ELSE U = V-W

WHEN (X.EQ.Y) U = V+W
ELSE
.   U = V-W
.   T = T+1.5
...FIN

WHEN (X.EQ.Y)
.   U = V+W
.   T = T-1.5
...FIN
ELSE
.   U = V-W
.   T = T+1.5
...FIN
```

WHEN and ELSE always come as a pair of statements, never separately. Either the WHEN or the ELSE or both may assume the multiline form. ELSE is considered to be a control phrase, hence it cannot be placed on the same line as the WHEN. Thus "WHEN (L) S1 ELSE S2" is not valid.

CONDITIONAL

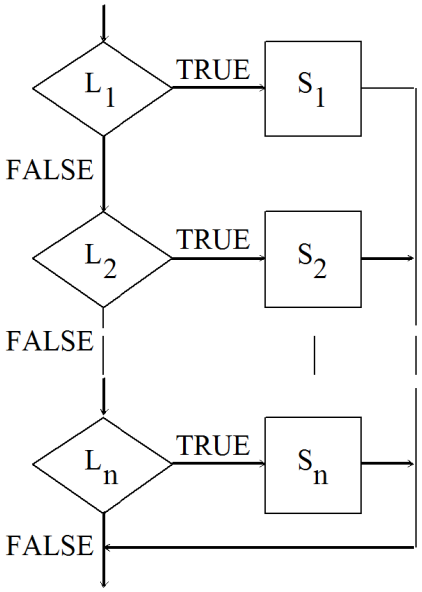
Description: The CONDITIONAL statement is based on the LISP conditional. A list of logical expressions is evaluated one by one until the first expression to be true is encountered. The scope corresponding to that expression is executed, and control then passes to the first statement following the CONDITIONAL. If all expressions are false, no scope is executed. (See, however, the note about OTHERWISE below.)

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

```
CONDITIONAL
.   (L1)   S1
.   (L2)   S2
.     .    .
.     .    .
.     .    .
.   (LN)   SN
...FIN
```

Flow Chart:



Examples:

```
CONDITIONAL
.   (X.LT.-5.0) U = U+W
.   (X.LE.1.0)  U = U+W+Z
.   (X.LE.10.5) U = U-Z
...FIN

CONDITIONAL
.   (A.EQ.B)  Z = 1.0
.   (A.LE.C)
.   .  Y = 2.0
.   .  Z = 3.4
.   ...FIN
.   (A.GT.C.AND.A.LT.B)  Z = 6.2
.   (OTHERWISE)  Z = 0.0
...FIN
```

The CONDITIONAL itself does not possess a one-line form. However, each "(Li) Si" is treated as a structured statement and may be in one-line or multiline form.

The reserved word OTHERWISE represents a catchall condition. That is, "(OTHERWISE) Sn" is equivalent to "(.TRUE.) Sn" in a CONDITIONAL statement.

SELECT

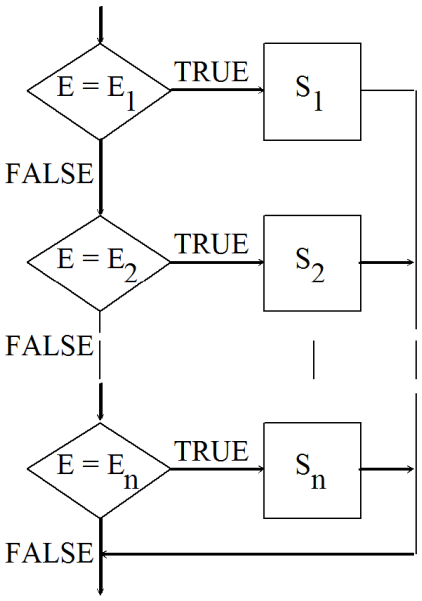
Description: The SELECT statement is similar to the CONDITIONAL but is more specialized. It allows an expression to be tested for equality to each expression in a list of expressions. When the first matching expression is encountered, a corresponding scope is executed and the SELECT statement terminates. In the description below, E, E1,...En represent arbitrary but compatible expressions. Any type of expression (integer, real, complex, character...) is allowed as long as the underlying Fortran system allows such expressions to be compared with an .EQ. or .NE. operator.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

```
SELECT    (E)
.   (E1)   S1
.   (E2)   S2
.     .    .
.     .    .
.     .    .
.   (EN)   SN
...FIN
```

Flow Chart:



Examples:

```
SELECT (OPCODE(PC))
.   (JUMP) PC = AD
.   (ADD)
.   .   A = A+B
.   .   PC = PC+1
.   ...FIN
.   (SKIP)   PC = PC+2
.   (STOP)   CALL STOPCD
...FIN
```

As in the case of CONDITIONAL, at most one of the Si will be executed.

The catchall OTHERWISE may also be used in a SELECT statement. Thus, "(OTHERWISE) Sn" is equivalent to "(E) Sn" within a "SELECT (E)" statement.

The expression E is reevaluated for each comparison in the list. Thus lengthy, time consuming, or irreproducible expressions should be precomputed, assigned to a variable, and the variable used in the specification portion of the SELECT statement.

LOOP Structures

The structured statements described below all have a scope which is executed a variable number of times depending on specified conditions.

DO

Description: The FLECS DO loop is functionally identical to the Fortran DO loop. The only differences are syntactic. In the FLECS DO loop, the statement number is omitted from the DO statement, the incrementation parameters are enclosed in parenthesis, and the scope is indicated by either the one line or multiline convention. Since the semantics of the Fortran DO statement vary from one Fortran compiler to another, a flowchart cannot be given. The symbol "I" represents any legal incrementation specification.

	General Form:	*Equivalent Fortran:*

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

	DO (I) S	DO 30 I S 30 CONTINUE
	Examples:	
	DO (I = 1,N) A(I) = 0.0 DO (J = 3,K,3) . B(J) = B(J-1)*B(J-2) . C(J) = SIN(B(J)) ...FIN	

WHILE

Description: The WHILE loop causes its scope to be repeatedly executed while a specified condition is true. The condition is checked prior to the first execution of the scope. Thus, if the condition is initially false the scope will not be executed at all.

General Form:

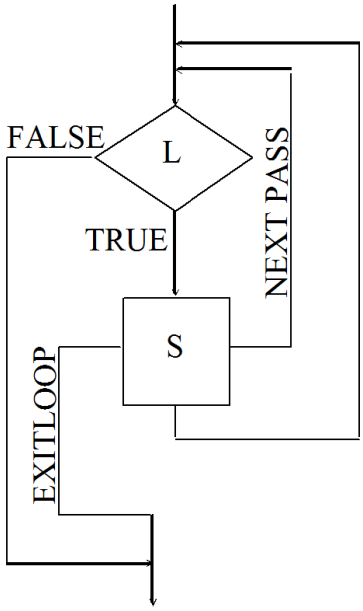
```
WHILE (L) S
```

Examples:

```
WHILE (X.LT.A(I)) I = I+1

WHILE (P.NE.O)
.   VAL(P) = VAL(P)+1
.   P = LINK(P)
...FIN
```

Flow Chart:



REPEAT WHILE

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT WHILE loop causes its scope to be repeatedly executed while a specified condition remains true. The condition is not checked until after the first execution of the scope. Thus, the scope will always be executed at least once and the condition indicates under what conditions the scope is to be repeated.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

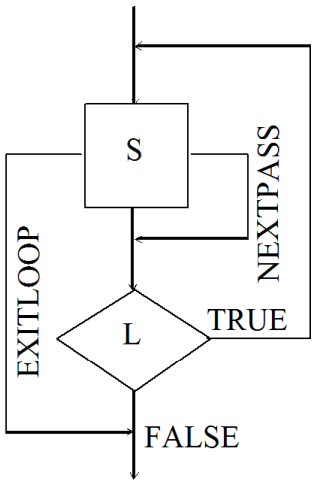
```
REPEAT WHILE (L) S
```

Examples:

```
REPEAT WHILE(N.EQ.M(I)) I = I+1

REPEAT WHILE (LINK(Q).NE.0)
.   R = LINK(Q)
.   LINK(Q) = P
.   P = Q
.   Q = R
...FIN
```

Flow Chart:



"REPEAT WHILE(L)" is functionally equivalent to "REPEAT UNTIL(.NOT.(L))."

UNTIL

Description: The UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is checked prior to the first execution of the scope, thus if the condition is initially true, the scope will not be executed at all. Note: "UNTIL (L)" is functionally equivalent to "WHILE (.NOT.(L))."

General Form:

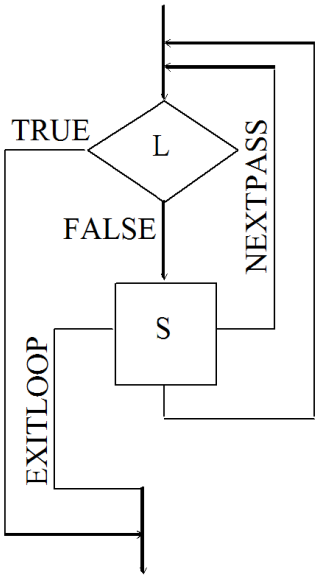
```
UNTIL (L) S
```

Examples:

```
UNTIL (X.EQ.A(I)) I = I+1

UNTIL (P.EQ.O)
.   VAL(P) = VAL(P)+1
.   P = LINK(P)
...FIN
```

Flow Chart:



All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

REPEAT UNTIL

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is not checked until after the first execution of the scope. Thus, the scope will always be executed at least once and the condition indicates under what conditions the repetition of the scope is to be terminated.

General Form:

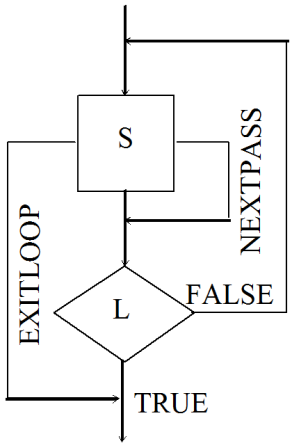
```
REPEAT UNTIL (L) S
```

Examples:

```
REPEAT UNTIL(N.EQ.M(I)) I = I+1

REPEAT UNTIL (LINK(Q).EQ.0)
.   R = LINK(Q)
.   LINK(Q) = P
.   P = Q
.   Q = R
...FIN
```

Flow Chart:



REPEAT FOREVER

Description: By using the keyword FOREVER it is possible to set up an "infinite loop" that will normally be terminated by an END= or ERR= branch. Alternatively, an EXITLOOP, EXITPROC, or GO TO statement can be used to exit from such a loop. Notice that this statement has no one-line form.

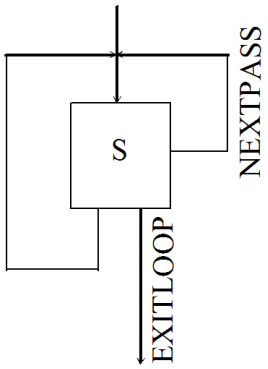
REPEAT FOREVER is functionally equivalent to REPEAT UNTIL(.FALSE.).

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

General Form:

```
REPEAT FOREVER
.
.  S
.
...FIN
```

Flow Chart:



Examples:

```
   REPEAT FOREVER
   .  READ (8,10,END=90) X
10 .  FORMAT (F10.2)
   ...FIN
90 I=I+1
```

3.1.7. Internal Procedures

In FLECS a sequence of statements may be declared an *internal procedure* and given a name. The procedure may then be invoked from any point in the program by simply giving its name.

Procedure names may be any string of letters, digits, and hyphens (i.e., minus signs) beginning with a letter and containing at least one hyphen. Internal blanks are not allowed. The only restriction on the length of a name is that it may not be continued onto a second line.

Examples of valid internal procedure names:

```
INITIALIZE-ARRAYS
GIVE-WARNING
SORT-INTO-DESCENDING-ORDER
INITIATE-PHASE-3
```

A *procedure declaration* consists of the keyword "TO" followed by the procedure name and its scope. The set of statements comprising the procedure is called its scope. If the scope consists of a single simple statement, it may be placed on the same line as the "TO" and procedure name, otherwise the statements of the scope are placed on the following lines and terminated with a FIN statement. These rules are analogous with the rules for forming the scope of a structured statement.

General form of procedure declaration:

```
TO procedure-name
```

Examples of procedure declarations:

```
TO RESET-POINTER P = 0
TO DO-NOTHING CONTINUE

TO SUMMARIZE-FILE
.  INITIALIZE-SUMMARY
```


All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

```
.  OPEN-FILE
.  REPEAT UNTIL (EOF)
.  .  ATTEMPT-TO-READ-RECORD
.  .  WHEN (EOF) CLOSE-FILE
.  .  ELSE UPDATE-SUMMARY
.  ...FIN
.  OUTPUT-SUMMARY
...FIN
```

An *internal procedure reference* is a procedure name appearing where an executable statement would be expected. In fact an internal procedure reference is an executable simple statement and thus may be used in the scope of a structured statement as in the last example above. When control reaches a procedure reference during execution of a FLECS program, a return address is saved and control is transferred to the first statement in the scope of the procedure. When control reaches the end of the scope, control is transferred back to the statement logically following the procedure reference.

A typical FLECS program or subprogram consists of a sequence of Fortran declarations: (e.g., INTEGER, DIMENSION, COMMON, etc.) followed by a sequence of executable statements called the body of the program followed by the FLECS internal procedure declarations, if any, and finally the END statement.

Here is a complete (but uninteresting) FLECS program which illustrates the placement of the procedure declarations:

```
C  INTERACTIVE PROGRAM TO COMPUTE X**2.
C  ZERO IS USED AS A SENTINEL VALUE TO TERMINATE EXECUTION

       REAL X,XSQ
       REPEAT UNTIL (X.EQ.0)
       .  GET-A-VALUE-OF-X
       .  IF (X.NE.0)
       .  .  COMPUTE-RESULT
       .  .  TYPE-RESULT
       .  ...FIN
       ...FIN
       CALL EXIT

     -----------------------------------

       TO GET-A-VALUE-OF-X
       .  WRITE (1,10)
  10   .  FORMAT ('X = ')
       .  READ (1,*) X
       ...FIN

     -----------------------------------

       TO COMPUTE-RESULT XSQ = X*X

     -----------------------------------

       TO TYPE-RESULT
       .  WRITE (1,30), XSQ
```


All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

```
30    .  FORMAT ('X-SQUARED = ',F7.2)
      ...FIN
      END
```

Notes concerning internal procedures:

1. All internal procedure declarations must be placed at the end of the program just prior to the END statement. The appearance of the first "TO" statement terminates the body of the program. The translator expects to see nothing but procedure declarations from that point on. Care must be taken that the program flow of the main body of the program does not accidentally "fall" into a procedure, that is, the last statement of the main body should be a RETURN or GOTO statement.
2. The order of the declarations is not important. Alphabetical by name is an excellent order for programs with a large number of procedures.
3. Procedure declarations may not be nested. In other words, the scope of a procedure may not contain a procedure declaration. It may of course contain executable procedure references.
4. Any procedure may contain references to any other procedures (excluding itself).
5. Dynamic recursion of procedure referencing is not permitted.
6. All program variables within a main or subprogram are global and are accessible to the statements in all procedures declared within that same main or subprogram.
7. There is no formal mechanism for defining or passing parameters to an internal procedure. When parameter passing is needed, the Fortran function or subroutine subprogram mechanism may be used or the programmer may invent his own parameter passing methods using the global nature of variables over internal procedures.
8. The FLECS translator separates procedure declarations on the listing by dashed lines as shown in the preceding example.

3.1.8. Additional Statements: EXITPROC/EXITLOOP/NEXTPASS

Three additional statements have been added to the FLECS language as described below. These statements may (within the constraints described below) appear anywhere that a simple Fortran statement could occur (e.g., on a line by themselves or following an IF or UNLESS clause). The loop-related statements (EXITLOOP and NEXTPASS) can be used within WHILE, UNTIL, REPEAT UNTIL, REPEAT WHILE, REPEAT FOREVER and FLECS-type Do loops; they cannot, however be used to control a Fortran-type (e.g., DO 10 I=1,20) DO loop. This should not be much of a problem since there is never any need to use a Fortran-type Do loop within FLECS. A detailed description of each command follows.

EXITPROC

The EXITPROC statement is used to exit from an internal procedure before the final FIN statement is encountered at the bottom of the procedure. An example would be:

```
   TO TEST-EXITPROC
C  .
   .  READ(1,*) IFLAG
   .  IF (IFLAG.EQ.-1) EXITPROC   ! All done
   .
   .     (Read & process data)
```


All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

```
   .
   ...FIN                          ! Normal return location
```

Using EXITPROC outside of an internal procedure (i.e., in the main program) will result in an error message.

EXITLOOP and NEXTPASS

The EXITLOOP and NEXTPASS statements are very similar and will be described together. These statements can only be used within a loop structure as described above; any other use will generate an error message.

The EXITLOOP statement will cause a jump to the statement following the last statement making up the current loop. Use a GO TO statement (or redesign your program!) if you need to exit more than one looping level.

The NEXTPASS statement generates a jump to the incrementing statement (in the case of a FLECS- type DO loop) or to the "test" clause (in all other cases). The following example shows how both of these statements can be used:

```
   EOF = .FALSE
   REPEAT UNTIL (EOF)
   .  DO (IAC=1,NACROS)
   .  .  READ (LUNIT,40,END=50) NADR
   .  .
   .  .     (Process NADR)
   .  .
   .  .  NEXTPASS         ! On to the next DO-loop index
50 .  .  EOF = .TRUE.
   .  .  EXITLOOP         ! Get out of DO-loop
   .  ...FIN
   .
   .       (More processing)
   .
   ...FIN
```

3.1.9. Restrictions and Notes

If FLECS were implemented by a nice intelligent compiler, this section would be much shorter. Currently, however, FLECS is implemented by a sturdy but naive translator. Thus the FLECS programmer must observe the following restrictions:

1. FLECS must invent many statement numbers in creating the Fortran program. It does so by beginning with a large number (usually 32767) and generating successively smaller numbers as it needs them. Do not use a number which will be generated by the translator. A good rule of thumb is to *avoid using 5 digit statement numbers*.

2. The FLECS translator must generate integer variable names. It does so by using names of the form "Innnnn" when nnnnn is a 5 digit number related to a generated statement number. *Do not use variables of the form Innnnn and avoid causing them to be declared other than INTEGER.* For example the declaration "IMPLICIT REAL (A-Z)" leads to trouble. Try "IMPLICIT REAL (A-H, J-Z)" instead.

3. The translator does not recognize continuation lines in the source file. Thus Fortran statements may be continued since the statement and its continuations will be passed through the translator without alteration. However, *an extended FLECS statement which requires translation may not be continued*. The

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

reasons one might wish to continue a FLECS statement are: a) it is a structured statement or procedure declaration with a one-statement scope too long to fit on a line, or b) it contains an excessively long specification portion, or c) both of the above. Problem a) can be avoided by going to the multiline form. Frequently problem b) can be avoided when the specification is an expression (logical or otherwise) by assigning the expression to a variable in a preceding statement and then using the variable as the specification.

4. IF statements must not be continued on a following line. FLECS treats all logical IF statements as FLECS statements, none of which can be continued in the normal Fortran fashion. (i.e., by putting a nonblank character in Column 6). This problem may be circumvented by setting a temporary LOGICAL variable equal to (condition) and then having an IF statement which tests the LOGICAL variable. Alternatively (or in addition) the IF statement can be rewritten to be a multiline FLECS-type statement.

5. FORTRAN 77 block-IF statements should not be used in a FLECS program. The syntax of FORTRAN 77 block-IF statements will confuse the FLECS translator and generate error messages.

6. *Blanks are meaningful separators in FLECS statements*; don't put them in dumb places like the middle of identifiers or key words and do use them to separate distinct words like REPEAT and UNTIL.

7. Let FLECS indent the listing. *Start all statements in Column 7* and the listing will always reveal the true structure of the program. (As understood by the translator, of course.) As mentioned in Section 3.1.5, Indentation, Lines, and the Listing it is *possible* to type in an indented listing, but this practice should generally be limited to making minor changes to existing programs.

8. As far as the translator is concerned, FORMAT statements are executable Fortran statements since it doesn't recognize them as extended FLECS statements. Thus, *only place FORMAT statements where an executable Fortran statement would be acceptable*. Don't put them between the end of a WHEN statement and the beginning of an ELSE statement. Don't put them between procedure declarations.

	Incorrect Examples:	Corrected Examples:
	`WHEN (FLAG) WRITE(3,30) 30 FORMAT(7H TITLE:) ELSE LINE = LINE+1`	`WHEN (FLAG) . WRITE (3,30) 30 . FORMAT(7H TI- TLE:) ...FIN ELSE LINE = LINE+1`
	`TO WRITE-HEADER . PAGE = PAGE+1 . WRITE(3,40) H,PAGE ...FIN 40 FORMAT (70A1,I3)`	`TO WRITE-HEADER . PAGE = PAGE+1 . WRITE(3,40) H,PAGE 40 . FORMAT(70A1,I3) ...FIN`

9. The translator, being simple-minded, recognizes extended FLECS statements by the process of scanning the first identifier on the line. If the identifier is one of the FLECS keywords IF, WHEN, UNLESS, FIN, etc., the line is assumed to be a FLECS statement and is treated as such. Thus, *the FLECS keywords are reserved and may not be used as variable names*. In case of necessity, a variable name, say WHEN, may be slipped past the translator by embedding a blank within it. Thus "WH EN" will look like "WH" followed by "EN" to the translator which is blank sensitive, but like "WHEN" to the compiler which ignores blanks. That is, if a keyword is to be recognized by FLECS it must not contain blanks. Moreover, the construction '(OTHERWISE)' must be *exactly* in this form - no blanks are allowed between the word and the parentheses.

10. In scanning a parenthesized specification, the translator scans from left to right to find the parenthesis which matches the initial left parenthesis of the specification. The translator, however, is ignorant of Fortran syntax including the concept of Hollerith constants and will treat Hollerith parentheses as syn-

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

tactic parentheses. Thus, *avoid placing Hollerith constants containing unbalanced parentheses within specifications*. If necessary assign such constants to a variable, using a DATA or assignment statement, and place the variable in the specification.

	Incorrect Examples:	Corrected Examples:
	`IF (J.EQ.'(')`	`LP = '(' IF(J.EQ.LP)`

11. The FLECS translator will not supply the statements necessary to cause appropriate termination of main and subprograms. Thus *it is necessary to include the appropriate RETURN, GOTO, STOP, or CALL EXIT statement prior to the first internal procedure declaration*. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

12. A RETURN or STOP statement just before a FIN can cause trouble because FLECS may insert an unlabeled statement (normally a GO TO) directly after the RETURN or STOP. The Fortran compiler may then complain because there is no path to the unlabeled statement. The error message from the compiler will normally be in the form of a warning and will not affect the generated code. However, with some compilers you can eliminate the message by using a procedure call such as GO-RETURN or GO-STOP in place of the RETURN or STOP. The GO-RETURN procedure would look like:

    ```
    TO GO-RETURN RETURN 99 CONTINUE FIN
    ```

    where a dummy CONTINUE statement has been added to 'fool' the compiler. A similar set up would be used for GO-STOP.

13. A problem similar to that in (12) will arise if a GO TO statement is the last statement before the FIN in a SELECT block:

    ```
    SELECT (X)
    .  (A)
    .  .  J=1
    .  .  Y=3.5
    .  .  GO TO 10
    .  ...FIN
    .  (B)
    .  .  I=9
    .  .  Z=8.6
    .  ...FIN
    ...FIN
    ```

    The GO TO 10 may upset the Fortran compiler in the same way as in (12). That is, a 'NO PATH TO STATEMENT' error will be generated. With some compilers, this problem can, if desired, be eliminated by putting a dummy labelled CONTINUE statement following the GO TO 10.

3.1.10. Errors

This section provides a framework for understanding the error handling mechanisms of the FLECS Translator. The system described below has proven to be quite workable.

The FLECS translator examines a FLECS program on a line by line basis. As each line is encountered it is first subjected to a limited syntax analysis followed by a context analysis. Errors may be detected during either of these analysis. It is also possible for errors to go undetected by the translator.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

Syntax Errors

When a syntax error is detected by the translator, it ignores the statement. On the FLECS listing, the characters *IGNORED* are printed in Columns 1 to 9 to indicate that the statement has been ignored. The nature of the syntax error is given in a message on the following line.

The fact that a statement has been ignored may, of course, cause some context errors in later statements. For example, the control phrase "WHEN (X(I).LT.(3+4)" has a missing right parenthesis. This statement will be ignored, causing, as a minimum, the following ELSE to be out of context. The programmer should of course be aware of such effects. More is said about them in the next section.

Context Errors

If a statement successfully passes the syntax analysis, it is checked to see if it is in the appropriate context within the program. For example, an ELSE must appear following a WHEN and nowhere else. If an ELSE does not appear at the appropriate point, or if it appears at some other point, then a context error has occurred. A frequent source of context errors in the initial stages of development of a program comes from miscounting the number or FIN's needed at some point in the program.

With the exception of excess FIN's which do not match any preceding control phrase and are ignored, (as indicated by *IGNORED* being printed in Columns 1 to 9 of the listing file) all context errors are treated with a uniform strategy. When an out-of-context source statement is encountered, the translator generates a "STATEMENT(S) NEEDED" message. It then invents and processes a sequence of statements which, if they had been included at that point in the program, would have placed the original source statement in a correct context. A message is given for each such statement invented. The original source statement is then processed in the newly created context.

By inventing statements the translator is not trying to patch up the program so that it will run correctly. Rather, it is simply trying to adjust the local context so that the original source statement and the statements which follow will be acceptable on a context basis. As in the case of context errors generated by ignoring a syntactically incorrect statement, such an adjustment of context frequently causes further context errors later on. This is called *propagation of context errors*.

One nice feature of the context adjustment strategy is that context errors cannot propagate past a recognizable procedure declaration. This is because the "TO" declaration is in context only at indentation level 0. Thus, to place it in context, the translator must invent enough statements to terminate all open control structures which precede the "TO." The programmer who modularizes his program into a collections of relatively short internal procedures, limits the potential for propagation of context errors.

Undetected Errors

The FLECS translator is ignorant of most details of Fortran syntax. Thus, most Fortran syntax errors will be detected by the Fortran compiler, not the FLECS translator. In addition, there are two major classes of FLECS errors which will be caught by the compiler rather than the translator.

The first class of undetected errors involves misspelled FLECS keywords. A misspelled keyword will not be recognized by the translator. The line on which it occurs will be assumed to be a Fortran statement and will be passed unaltered to the compiler which will no doubt object to it. For example, a common error is to spell UNTIL with two L's. Such statements are passed to the compiler, which then produces an error message. The fact that an intended control phrase was not recognized frequently causes a later context error since a level of indentation will not be triggered.

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

The second class of undetected errors involves unbalanced parentheses. (See also Note 10 in Section 3.1.9, Restrictions and Notes.) When scanning a parenthesized specification, the translator is looking for a matching right parenthesis. If the matching parenthesis is encountered before the end of the line, the remainder of the line is scanned. If the remainder is blank or consists of a recognizable internal procedure reference, all is well. If neither of the above two cases hold, the remainder of the line is assumed (without checking) to be a simple Fortran statement which is passed to the compiler. Of course, this assumption may be wrong. Thus the statement

```
WHEN (X.LT.A(I)+Z))  X = 0
```

is broken into

```
keyword "WHEN"
specification "(X.LT.A(I)+Z)"
Fortran statement ") X = 0"
```

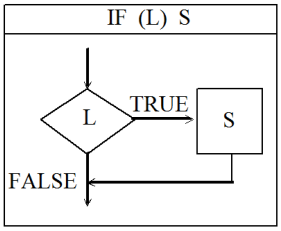
Needless to say, the compiler will object to ") X = 0" as a statement.

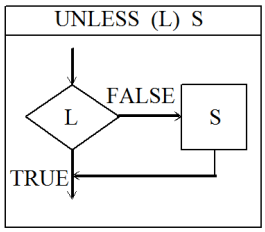
An undetected error in one line may trigger a context error in a much later line. Noticing the context error, the programmer does not proceed with compilation and hence is not warned by the compiler of the genuine cause of the error. One indication of the true source of the error may be an indentation failure at the corresponding point in the listing.

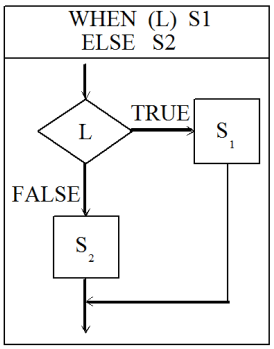
Other Errors

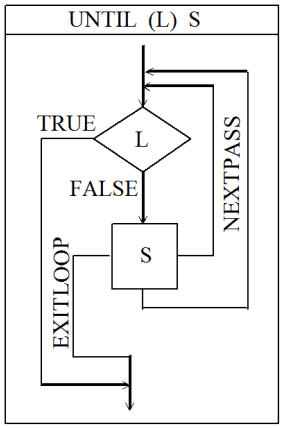
The translator detects a variety of other errors such as multiply defined, or undefined procedure references. The error messages are self-explanatory. (Really and truly!)

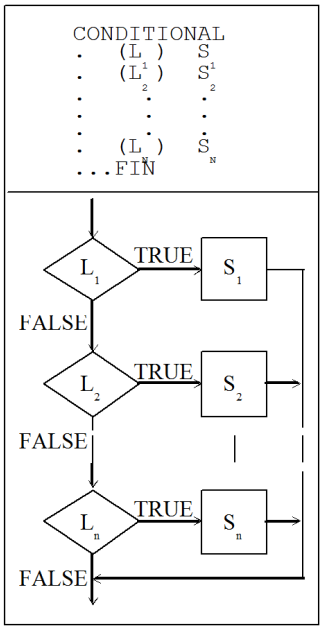
All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

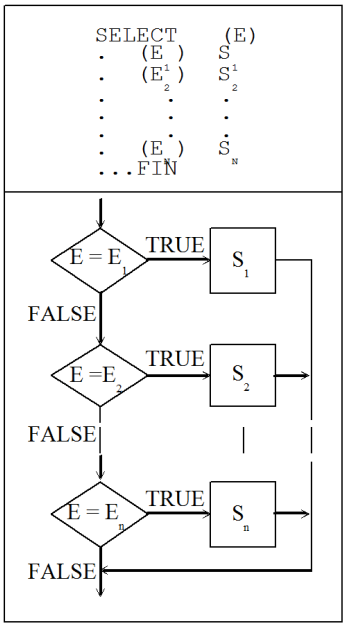


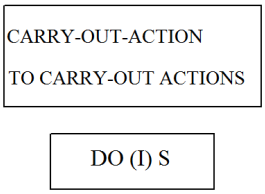


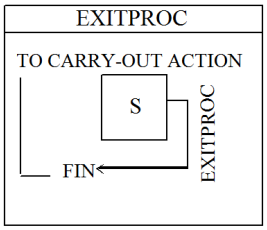








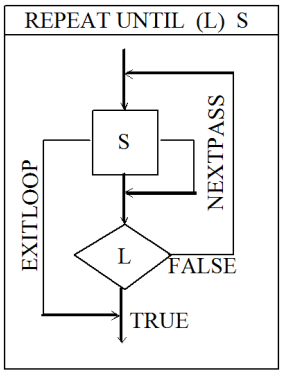


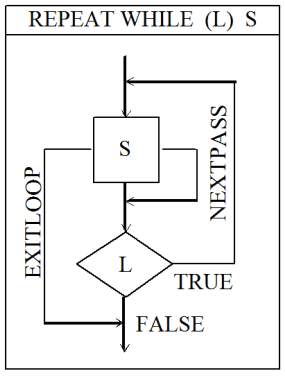


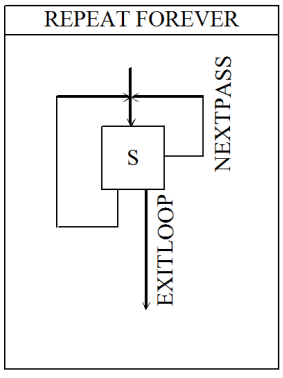
Note: Place RETURN, STOP, or CALL EXIT statement ahead of the first TO statement.

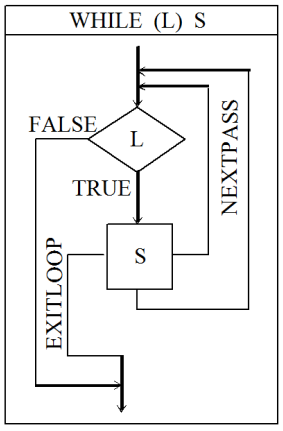
Note: OTHERWISE can be used as a catchall condition or expression in CONDITIONAL and SELECT statements.

Legend:
L = Logical Expression
S = Statement(s)
E = Expression
I = DO Specification









All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

3.2. Using FLECS for PSS®E

3.2.1. Introduction

This document describes the specific manner in which the FLECS32 preprocessor is used on PC systems. You should be familiar with the FLECS-to-Fortran Translator which describes the features found in all implementations of FLECS as supplied by Siemens PTI. The write-up below also describes a number of additional limitations and enhancements in the FLECS32 package.

3.2.2. Use of FLECS32

FLECS32 is a command-line driven program. Simply type:

```
FLECS32 filename
```

where 'filename' is either the name of a file in the current directory or a pathname specifying a file in some other directory. If the input file ends with ".FLX" there is no need to specify the ".FLX" when giving the FLECS32 command. Examples are:

```
FLECS32 DUM       (file DUM.FLX is in the current
directory)
FLECS32 C:\WFB\DUM(file DUM.FLX is in the WFB directory on drive
C)
```

The default filenames generated by FLECS32 are:

- Listing file will be "filename.LIS"
- Fortran file will be "filename.FOR"

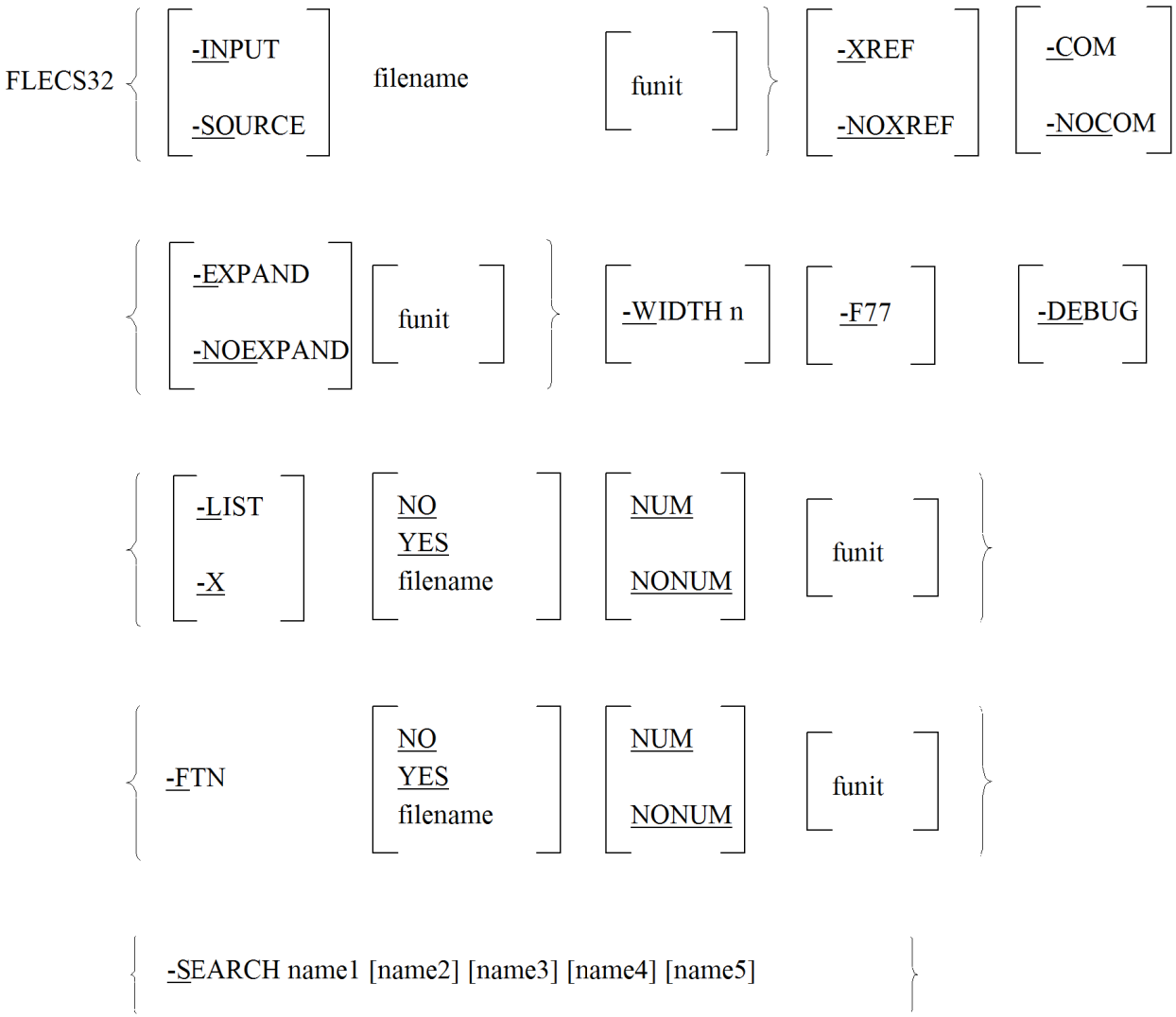
The above filenames are used only if the user has NOT specified names for the Listing and Fortran output files as described below.

Any errors detected by FLECS32 will be displayed on the terminal as well as being written to the listing file.

3.2.3. FLECS32 Options

The operation of FLECS32 as described above may be modified by specifying one or more options after the "pathname". The general form of the FLECS32 command line is:

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International



That is, the user may specify:

- An input tree-filename.
- A FLECS32 LIST output tree-filename.
- A Fortran output tree-filename.
- The unit numbers to be used for reading the input file and the $INCLUDE file(s), and writing the LIST and Fortran output files. If nested $INCLUDE files are used, the file unit used to read the second-level file(s) will be obtained by adding 1 to the "funit" used to read the first-level $INCLUDE file(s).
- Whether or not (YES/NO) an output file is to be produced.
- Whether or not (NUM/NONUM) line numbers are to be inserted on the right side of the output file(s).

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

- Whether or not (XREF/NOXREF) a PROCEDURE cross-reference map is to be produced. (Specifying XREF will automatically cause line numbers to be inserted in the LIST file.)

- Whether or not (WIDTH n) the width of the listing file is to be expanded from 80 characters. If so, specify -WIDTH n, where n is the desired width up to 134 characters.

If you are FLECSing a file that has line numbers on the right hand side, DO NOT specify a listing width that is GREATER than the current width. If you do, the line numbers will be taken as part of the input file and great confusion will result. To “shrink” the width of such a line number file, FLECS32 it without specifying line numbers for the output LISTING file. Then, if desired, re-FLECS32 the file specifying the desired width and numbering options.

- Whether or not (EXPAND/NOEXPAND) the $INCLUDE files are to be processed.

- If the FLECS32 command line includes an -EXPAND option all $INCLUDE files will be read and their contents interpreted as FLECS32 source statements just as though they were part of the main input file. The “translated” statements are then written to the Fortran output file, but not to the LISTING file (which will contain the $INCLUDE statements only).

- Thus, with the -EXPAND option $INCLUDE files can contain FLECS32 statements (including “!” comments). If the -EXPAND option is NOT specified, $INCLUDE files are not read and no “translation” of their contents is attempted.

- The EXPAND option can be very useful when used on computers that do not have a $INCLUDE statement since the function of $INCLUDE can now be handled entirely within FLECS32. In other words, programs using $INCLUDE statements are more “portable” and can be installed on other machines more easily.

- Whether FORTRAN 77 DO-loop rules are to be assumed. Specifically, specifying the command line option -F77 causes FLECS32 to treat the invoking of FLECS32 procedures from within DO-loops as errors.

- Whether or not (COM/NOCOM) all comment (“C”) lines are to be passed through to the Fortran output file. If -COM is specified the beginning of each internal PROCEDURE will also be indicated.

- Whether the lines with a “D” in column 1 are treated as debug code or comment lines.

- If the -DEBUG option is specified, lines with a “D” in column 1 are output to the Fortran file with the “D” removed. This allows the inclusion of debug code in the FLECS32 file without forcing its overhead in the resulting production version of the Fortran program.

- A list of names (name1, name2, ... name5) that are to be used as prefixes whenever a $INCLUDE file must be found. For example, specifying -SEARCH C:\SYSCOM would allow a $INCLUDE statement of the form $INCLUDE KEYPTI to be used. Otherwise, the more system-dependent form $INCLUDE C:\SYSCOM\KEYPTI would be needed.

- Each “name” will be used (in the order specified) as a prefix whenever a simple filename (not a pathname) is found in a $INCLUDE statement.

- Whenever -SEARCH is specified, the current directory will NOT BE SEARCHED for a specified $INCLUDE file UNLESS a “*” is given as one of the “names.” A typical -SEARCH option would then be:

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

```
-SEARCH * C:\SYSCOM
```

which would cause the current directory to be searched first, followed by a search of the C:\SYSCOM directory.

If you use '-SEARCH', you should also specify '-EXPAND' for proper results.

The -INPUT and -SOURCE specifications are optional. If neither is specified, the first name following the FLECS32 command is assumed to be the input tree-filename. Except for the YES/NO/NUM/NONUM key words, all the "-" specifiers may be abbreviated as shown by the underlining in the description of the FLECS32 command line shown at the beginning of this section. The order of items following the -I, -SO, -L, -X and -F specifications is immaterial. (As indicated above, the order of "names" following the -SEARCH option *IS* important.) Any items which are not specified will be set to their default values. Not specifying any options is equivalent to:

```
FLECS32 -INPUT path name 13 -NOXREF -NOCOM -NOEXPAND
16
-WIDTH 80 -LIST YES NONUM 14 -FTN YES NONUM 15
-SEARCH *
```

Note that in the default case no line numbers will be added to either the LIST or Fortran output files. By default, FLECS32 will use file units 13, 14, 15, 16 and 17 for INPUT, LIST output, Fortran output, first level $INCLUDE files, and second level $INCLUDE files, respectively. These file units are unlikely to be in use for other purposes when FLECSing a file; therefore, it is unlikely that these numbers will have to be altered.

3.2.4. Examples of Using FLECS32

1. Normal case - FORTRAN 77-based FLECS32 program

   Assume the input file is 'FILE.FLX'.

   ```
   FLECS32 FILE -F77
   ```

   Generates FILE.FOR, FILE.LIS

   ```
   DEL FILE.FLX
   ```

   Make the new FLECS32 listing

   ```
   REN FILE.LIS FILE.FLX
   ```

   file into the input for next time

   ```
   DF FILE
   ```

   Compile the Fortran output

   ```
   DEL FILE.FOR
   ```

   Delete the Fortran output

All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International

2. Generate only a FLECS32 listing file with line numbers and a procedure cross-reference map. Store the listing in the WFB directory on drive D under the name 'LIST'.

```
FLECS32 FILE -F NO -L D:\WFB\LIST -XREF
```


All material contained in this documentation is proprietary to Siemens Industry Inc., Siemens Power Technologies International