Easy5 2021.1

Reference Manual

For Windows® and Linux®

Corporate
MSC Software Corporation
4675 MacArthur Court, Suite 900
Newport Beach, CA 92660
Telephone: (714) 540-8900
Toll Free Number: 1 855 672 7638
Email: americas.contact@mscsoftware.com

Europe, Middle East, Africa
MSC Software GmbH
Am Moosfeld 13
81829 Munich, Germany
Telephone: (49) 89 431 98 70
Email: europe@mscsoftware.com

Japan
MSC Software Japan Ltd.
KANDA SQUARE 16F
2-2-1 Kanda Nishikicho, Chiyoda-ku
Tokyo 101-0054, Japan
Telephone: (81)(3) 6275 0870
Email: MSCJ.Market@mscsoftware.com

Asia-Pacific
MSC Software (S) Pte. Ltd.
100 Beach Road
#16-05 Shaw Tower
Singapore 189702
Telephone: 65-6272-0082
Email: APAC.Contact@mscsoftware.com

Worldwide Web
www.mscsoftware.com

Support
http://www.mscsoftware.com/Contents/Services/Technical-Support/Contact-Technical-Support.aspx

January 5, 2021

Documentation Feedback

At MSC Software, we strive to produce the highest quality documentation and welcome your feedback. If you have comments or suggestions about our documentation, write to us at: documentation-feedback@mscsoftware.com.

Please include the following information with your feedback:

- Document name
- Release/Version number
- Chapter/Section name
- Topic title (for Online Help)
- Brief description of the content (for example, incomplete/incorrect information, grammatical errors, information that requires clarification or more details and so on).
- Your suggestions for correcting/improving documentation

You may also provide your feedback about MSC Software documentation by taking a short 5-minute survey at: http://msc-documentation.questionpro.com.

Note:	The above mentioned e-mail address is only for providing documentation specific feedback. If you have any technical problems, issues, or queries, please contact Technical Support.

Contents

Reference Manual

Preface

Conventions Used in This Guide . . . . **xviii**

1 Reference Manual Topics

Overview **5**

Accelerator Keys **5**

Adding Components **6**

Add Components Window 6

Adding Components to the Schematic 9

Add Components by Name Reference 10

Analyses **11**

Nonlinear Analyses 11

Linear Analyses 12

Analysis Data Form **12**

Analysis Data Form Header 13

Analysis Data Form 14

Auxiliary Input File **15**

Analysis Title 15

Time of the Analysis 16

Initial Operating Point 16

Model Explorer “Pickable” Fields 17

Auxiliary Input File 18

Creating an Auxiliary Input File 18

Using an “*auxfile*” To Enter Blocks of Data 19

Specifying a Label in an Auxiliary Input File 20

Auxiliary Input File Data Format **21**

PARAMETER VALUES Command 22

Scalar Parameter Data 22

Array Parameter Data 23

Expressions 26

Tabular Data 26

“Analysis Only” Mode **29**

Operations Allowed 30
Disabled Functionality 30
Creating a “Locked Configuration” Model for Distribution Purposes 31
Using a Locked Configuration Easy5 Model 32

Background Processes 32

C Component 34
Adding C Code 35
Example C Code Component 36
Adding C Declarations 38
C Code Files and Structure 38

Code Components 39

Command Line Options 40
Option Examples 43

Compiling External Code 43
Default Compiler Options 44
Obtaining Current Compiler Options 44
Setting Debug Compiler Options. 45
User Specified Compiler Options 45
Examples of Compiling External Code. 45
Compiling and Linking Mixed Code. 46

Components 51
Component Basics 51
Blocks 54
Standard Components 54
Code Components 55
User-defined Library Components 56
Extension Components. 57
Dimensioned Components 58

Component Data Table 60
Documentation/Configuration Tab 60
States Tab 65
Variables Tab 66
Version Tab 67
User-Comments Tab 67

Connecting Components 67
Rules for Connecting Components 68
Default Connections 68
Port Connections 70
Default Port Connection Points 71
Custom Connections 72
Making a Branch Connection 74
Connecting Incompatibly Vectorized Components. 75

Connection Lines 76

Moving Connection Line Endpoints 76
Moving Connection Line Segments 77
Changing an Anchored Connection Back to an Autoroute Connection 78
Customized Line Routing 78
Defining Connection Line Labels and Attributes 80
Connection Line Navigation. 83
Submodel Connection Labels 84
Connection Label Options 84
Moving Submodel Connection Nodes 85
Connection Line Color Dots. 86

Copying Components and Models 87
Copying Components within a Model 87
Copying Components From or To Another Model. 88
Copying Components With User-defined Names 88

Data Display 88

Data Types 92
States. 93
Variables 93
Parameters. 93
Tables 93

Debugging the Model and Analysis 94
Example of Using the Symbolic Debugger on Windows 94
Example of Using the Symbolic Debugger on a Linux Platform 97

Deleting Components and Connections. 100
Deleting Components 100
Deleting Connections 101

Discrete (Digital) System Analysis 102
Operating Point Considerations. 102
Linear Analysis Considerations 102
Integration Method Considerations 103

Discrete (Digital) System Modeling 103
Digital Models. 103
Hybrid Models 104
Discrete System Modeling Using Fortran, C and LIbrary Components 105
Matching TAU Method (obsolete) 106

Documenting and Printing the Model 109
Generating a Model Document File 109

Exporting an Easy5 Model as a MAT EMX Function 111
MAT function “ezmodel”. 111

Easy5 Window 111
Description Lines 112

Model Info 113
Menu Bar 113
Tool Bar 113
Dockable Add Component WIndow 113
Scroll Bars 113
Message Line 114
Schematic Window 114
Working with Easy5 Windows 114

Eigenvalue Sensitivity Analysis 116
Setting up an Eigenvalue Sensitivity Analysis 116

Eigenvalue Sensitivity Analysis Method 117

Executable Model 118
Create Executable 119
Link External Object 120
Solve Implicit Loops 120
Force Explicit Typing 120
Check for Duplicate Names 120
Debug Mode 120
Stop Create Executable 121

Executable Output Files 121
Create Executable Process 121
Model Generation Listing File 122
Executable Source File 122
Executable Error File 124

External (Environment) Variables 124

Fortran Component 129
Forced Explicit Typing 129
Using Integer or Logical Variables in Fortran Code 130
Adding Nonexecutable Fortran Statements 131
Reserved Fortran Unit Numbers 132
Adding Comments to Fortran Code 132
Easy5 Reserved Words 132
Calculating Initial Condition Values in a User-Code Component 134
Easy5 Matrix Operations 134
Sorting Fortran Component Code 135

Function Scan Analysis 135
Setting up a Function Scan Analysis 136
Function Scan with Two Independent Values 138

Function Scan Analysis Method 139

Graphic Files, EMFs, and PostScript 139
Generating the Schematic Block Diagram EMF Graphics File 139
Generating Plotter EMF Graphics File 141
Using EMF Graphics 141

Overriding Hard copy and EMF Plot Curve and Grid Widths 142
Exporting Plot Files 142
Importing a PostScript File Into a Document 144
Icon Editor. 144
Implicit Model. 145
Definition of an Implicit Model. 146
Example of an Implicit Model 146
How to Break Implicit Loops 149
Initial Condition Calculation. 151
Initialization Statement 152
Integration Methods 153
The Integration Method. 153
Integration Methods Available 154
Definition of Terms 155
The BCS Gear Method. 157
The Runge-Kutta Methods 157
The Huen Method 157
The Euler Method 157
The Adams Method. 158
The User-defined Method 158
Integration Method Selection Guidelines 160
Guidelines for Setting Error Controls 161
Interactive Simulation 163
Linear Model Generation Analysis 163
Types of Linear Model Generation Analysis 163
Setting up a Linear Model Generation Analysis 165
Controlling the Calculation 168
Saving the Linear Model System Matrices. 168
Linear Model Generation Method. 168
Continuous Systems 168
Stability Analysis for Sampled-Data Systems 175
Linking External Code 175
Linking Routines Using the Build Menu 176
Linking Routines Using the EASY5_OBJECT Variable. 177
Linking Routines Using an Object Library 178
Linking Library Component Routines. 179
Library Component Code 180
Using Variable Dimensions in Library Component Code. 180
Using Integer or Logical Variables in Library Component Code 181
Configurations 182
Component Libraries 182

Matrix Operations **183**
Matrix Equations 183
Matrix Operation Subroutines 186
Matrix Editor **188**
Creating a Data Table 188
Menus **190**
File Menu 190
Edit Menu 192
View Menu 194
Options Menu 195
Library Menu 196
Build Menu 197
Analysis Menu 198
Submodel Menu 200
Help Menu 200
Using Shortcut Keys to Open and Select Menus 202
Using Arrow Keys to Select Menu Items 202
Closing an Opened Menu 203
Menus: Shortcut Menu **203**
Model Explorer Window **209**
Pick Method 210
Model Files **211**
Modeling Fundamentals **213**
Example Problem 213
Block Diagram Modeling Method 214
Component “Systems Diagram” Modeling Method 215
User-defined Component Modeling Method 217
State Variable Modeling Method 218
Multiple Analysis **220**
Setting up Multiple Analysis 220
Understanding the Setup Form 222
Defining an Analysis Property 223
Open Model Dialog **225**
Bypassing the Open Model Dialog 225
Starting a New Model 226
Opening an Existing Model 226
Opening a Damaged Model 226
Checking Background Processes 226
Operating Point **228**
Creating an Operating Point 228
Operating Point Files 229
Editing the Operating Point File 230

Parameters - Defining Input Values **230**
Matrix Editor Window 231
Plot Tables Analysis **232**
Setting up a Plot Tables Analysis 232
Power Spectral Density Analysis **233**
The Power Spectral Density Analysis Data Form 234
The PSD Input Noise Distribution Parameters Form 236
Power Spectral Density Analysis Outputs 238
Power Spectral Density Analysis Method **238**
Print Options **241**
Windows Print Options 241
Linux Print Options 241
Reserved Words **242**
General Reserved Words 243
Physical and Mathematical Constants Reserved Words 246
Root Locus Analysis **246**
Setting up a Root Locus Analysis 247
Root Locus Analysis Method 250
Schematic Manipulation **252**
Moving Components 252
Moving the Window 253
Viewing the Entire Schematic 253
Returning to the Previous Schematic View 253
Locating Components 253
Simulation Analysis **254**
Setting up a Simulation 254
Setting Up Simulation Options 257
Setting up the Integration 258
Setting up Simulation Plots 260
Setting up Simulation Print Output 263
Specifying Secondary Plotting and Printing Rates 265
Saving a Final Simulation Operating Point 266
Executing the Simulation 266
Simulation Outputs Results 267
Simulation Monitor **268**
What is the Simulation Monitor? 268
Activating the Simulation Monitor 268
Simulation Monitor Plotter Features 268
Simulation Troubleshooting **269**
Type of Failure 269
Simulation Failure Error Messages 270

Write Your Own Diagnostics to the Output Listing File. 271
Monitoring CPU Time During a Simulation 272
Killing a Simulation Run 272
Integration Problems 272

Single Call Analysis 275

Sort Blocks 276
Model Sorting. 276
Defining Sort Blocks. 277
Fortran and Library Component Sort Blocks 278

Stability Margins Analysis. 278
Setting up a Stability Margins Analysis 279
Limitations 279

Stability Margin Analysis Method 280

States 282
Continuous States 283
Sample States 284
Delay States. 285

States: Defining Values and Controls 286
State Initial Conditions 286
Error Controls. 287
Freezing States 287

Steady-State Analysis 287
Setting up a Steady-State Analysis. 288
Steady-State Analysis Outputs 291

Steady-State Analysis Method 292
Steady-State Analysis Troubleshooting. 293
Steady-State May Fail to Converge. 294
Overcoming Steady-State Non-Convergence 294

Stop and Exit Flags 295
Terminating Using the ISTOP Flag 295

Submodels 296
Submodel Menu. 300
Defining a Submodel 300
Opening and Closing a Submodel. 301
Submodel Labels 302
Submodel Connection Lines 303
Editing Submodel Properties 303
Expanding a Submodel. 304
Navigating Submodels 304

Switch States 305
What Switch States Represent 306
The Advantages of Switch States 307

Using Switch-State Standard Components 309
Using Switch States in User Code and Library Components. 310
Example of Using Switch States 310

Temporary Settings File 313
Creating a New Temporary Settings File 313
Entering and Editing Data in the Temporary Settings Editor 314
Applying a Temporary Settings File to an Analysis. 317
Loading Temporary Settings Data Into the Model 318

Termination Commands 318

Text Editor. 320
User-Defined Text Editor. 321

TIME - Testing on the Value of Time 321

Transfer Function Analysis 322
The Transfer Function Data Form 323
Specifying the Transfer Function Input 323
Specifying the Transfer Function Output 324
Requesting a Frequency Response Plot. 324
Transfer Function Analysis Output Data 325

Transfer Function Analysis Methods 326
Frequency Response Method 326
Selecting Frequency Points. 327
Transfer Function Analysis Method Selection 327
Coordinate Transformation 328

Transfer Function Troubleshooting 329

Transfer Function of Sampled Data Systems 334
Stability Margins for Sampled Data Systems. 334
Single Rate Systems. 337
Multi-Rate Systems 338

User-Defined Names 341
Defining User-defined Names 342
User-defined Name Menu Options 343
Resolving User-Defined Name Conflicts 343
Changing User-Defined Names. 344

References 345
Suggested Reading. 346

A Summary of Analysis Commands

Overview 348

Data Input Commands 348

State Control Commands 348
Operating Point Commands 349
Eigenvalue Sensitivity Analysis 350
Function Scan Analysis Commands 350
Linear Model Analysis Commands 350
Root Locus Analysis Commands 351
Simulation Analysis Commands 352
Stability Margin Analysis Commands 353
Steady-State Analysis Commands 353
Transfer Function Analysis 354
Plot Commands 354
Print Commands 355

B Guide to Numerical Integration

Overview 358
Numerical Stability 358
Accuracy and Error Control 361
Integration Method Selection Guidelines 364

C Discrete Analysis Techniques

Overview 368
Linear Sampled-Data System Equations 368
Numerical Calculation of Continuous-System Transition Matrix 371
Discrete System Transition Matrix 371
Total Transition Matrix for a Single-Rate System 372
The Discrete Linear-Model Matrices for a Single-Rate System 372
A Single-Rate Example 374
Total Transition Matrix for a Multi-Rate System 376

D Batch Mode Commands

Overview 380
Batch Mode Command 380
Examples of Batch Commands 381
External Variables 382
Examples: 383

Extraneous Easy5 Batch Files 384
Command Definitions 385
Python Multiprocessing 385
Windows 387
Linux (or Windows) 387
Black Box (BBX) Export and Analysis 388

E Program Limits

Modeling Limitations 392
Analysis Limitations 393

Preface

■ Conventions Used in This Guide xviii

The *Easy5 Reference Manual* is a companion document to the *Easy5 User Guide* and contains reference material in greater detail than the user guide.

This preface explains how to use the mouse to select menus and perform other techniques in Easy5.

For ease-of-use, topics are presented in alphabetical order.

In addition to the Easy5 topics reference, the following Appendixes are also included:

- Ap. A: Summary of Analysis Commands
- Ap. B: Guide to Numerical Integration
- Ap. C: Discrete Analysis Techniques
- Ap. D: Batch Mode Commands
- Ap. E: Program Limits

Conventions Used in This Guide

This guide uses the following text conventions to identify functions and procedures:

- Text in Courier font indicates a file name, command line text, or programming text.

  `easy5x -help`
- Text in an italic font enclosed in < > indicates a variable file name that is specific to your computer.

  Select the <*model_name*>`.exe` file from the window.
- Most keyboard shortcuts require that you press the Control key together with another key simultaneously and are written as designated in the previous convention.

  Press Ctrl+S.
- Text in a bold font with right angle brackets between words indicates that you select a menu and subsequent submenus with your mouse.

  Go to **File** > **Print Preview** to view your document.
- Text in bold font indicates an action that you need to perform.

  Select the **Hydraulic_Act** model.

  Click **OK** to close the window.

Note: Notes are used to highlight certain important information.

Several icons are also used throughout this guide to bring your attention to certain items.

Items preceded by the Windows logo apply only to the Windows operating system.

1 Reference Manual Topics

- Overview 5
- Accelerator Keys 5
- Adding Components 6
- Analyses 11
- Analysis Data Form 12
- Auxiliary Input File 15
- Auxiliary Input File Data Format 21
- “Analysis Only” Mode 29
- Background Processes 32
- C Component 34
- Code Components 39
- Command Line Options 40
- Compiling External Code 43
- Components 51
- Component Data Table 60
- Connecting Components 67
- Connection Lines 76
- Copying Components and Models 87
- Data Display 88

- Data Types 92
- Debugging the Model and Analysis 94
- Deleting Components and Connections 100
- Discrete (Digital) System Analysis 102
- Discrete (Digital) System Modeling 103
- Documenting and Printing the Model 109
- Exporting an Easy5 Model as a MAT EMX Function 111
- Easy5 Window 111
- Eigenvalue Sensitivity Analysis 116
- Eigenvalue Sensitivity Analysis Method 117
- Executable Model 118
- Executable Output Files 121
- External (Environment) Variables 124
- Fortran Component 129
- Function Scan Analysis 135
- Function Scan Analysis Method 139
- Graphic Files, EMFs, and PostScript 139
- Icon Editor 144
- Implicit Model 145
- Initial Condition Calculation 151
- Initialization Statement 152
- Integration Methods 153
- Integration Method Selection Guidelines 160
- Interactive Simulation 163
- Linear Model Generation Analysis 163
- Linear Model Generation Method 168
- Linking External Code 175
- Library Component Code 180

- Matrix Operations 183
- Matrix Editor 188
- Menus 190
- Menus: Shortcut Menu 203
- Model Explorer Window 209
- Model Files 211
- Modeling Fundamentals 213
- Multiple Analysis 220
- Open Model Dialog 225
- Operating Point 228
- Parameters - Defining Input Values 230
- Plot Tables Analysis 232
- Power Spectral Density Analysis 233
- Power Spectral Density Analysis Method 238
- Print Options 241
- Reserved Words 242
- Root Locus Analysis 246
- Schematic Manipulation 252
- Simulation Analysis 254
- Simulation Monitor 268
- Simulation Troubleshooting 269
- Single Call Analysis 275
- Sort Blocks 276
- Stability Margins Analysis 278
- Stability Margin Analysis Method 280

- States 282
- States: Defining Values and Controls 286
- Steady-State Analysis 287
- Steady-State Analysis Method 292
- Stop and Exit Flags 295
- Steady-State Analysis Method 292
- Switch States 305
- Temporary Settings File 313
- Termination Commands 318
- Steady-State Analysis Method 292
- TIME - Testing on the Value of Time 321
- Transfer Function Analysis 322
- Transfer Function Analysis Methods 326
- Transfer Function Troubleshooting 329
- Transfer Function of Sampled Data Systems 334
- User-Defined Names 341
- References 345

Overview

This manual contains reference material that relates to the Easy5 Graphical User Interface (GUI), the Model Builder, and the analysis programs. This is a companion document to the Easy5 User Guide. The user guide contains tutorials and examples on how to use Easy5, whereas this document provides the user with additional reference material.

Easy5 topics in this manual are listed alphabetically for ease-of-use.

Accelerator Keys

Accelerators are typically keystroke combinations involving the "Ctrl" (control) key plus a character key or a function key that invoke a menu or menu item even when the menu is not displayed. Easy5 menus have accelerators and mnemonics associated with them. Mnemonics are keyboard equivalents that enable the user to access menus (and menu items) without having to use the mouse.

The mnemonics are indicated in the respective Easy5 menus. The accelerator and function keys are summarized below:

Table 1-1 Accelerator and Function Keys

Key(s)	Function
Ctrl+A	Add Component
Ctrl+B	Build Executable Model
Ctrl+Shift+C	Display C Component Source File
Ctrl+E	View Entire Schematic
Ctrl+F	Model Explorer
Ctrl+Shift+F	Display Executable Source File
Ctrl+C	Copy Selection
Ctrl+Shift+L	Display Analysis Output Listing
Ctrl+Shift+A	Display Analysis Log
Ctrl+Shift+B	Display Build Log
Ctrl+M	Move selected group (drag and drop)
Ctrl+N	File>New
Ctrl+O	File>Open
Ctrl+P	Print Current Schematic
Ctrl+Shift+P	Plot Current Analysis Results
Ctrl+S	Save Model
Ctrl+C	Copy Selection
Ctrl+V	Paste
Ctrl+-	Zoom Out
Ctrl++	Zoom In

Table 1-1 Accelerator and Function Keys (continued)

Key(s)	Function
<Space>	View Refresh
F6	Monitor Simulation
F7	Open Simulation Analysis Form
F8	Open Steady State Analysis Form
F9	Open Linear Model Generation Analysis Form
Ctrl+Click-Right	Go to Connection End

Adding Components

There are two ways of adding components to a schematic:

- Using the **Add Components** window to select a component from a list of libraries, groups, and components.
- By adding a component directly by referencing its name.

These two methods are described in the following sections. See also "Components".

Add Components Window

The dockable Add Components window lets you view and access a list of all libraries and components. Generally this window remains open until all model components have been added.

This window is divided into three scrollable areas: the type of library, the library group, and the components. The top windowpane lists the *available* libraries. The libraries listed in your window will depend on whether you are licensed to use a particular library.

Libraries contain components for a specific application. For example, in Figure 1, the Hydraulics (hc) library has been selected.

Some libraries contain a large number of components. To facilitate the management of these components, the libraries are further divided into groups. Groups within a selected library are listed in the second window pane. For Figure 1, the selected group is Pump/Motor Accessories.

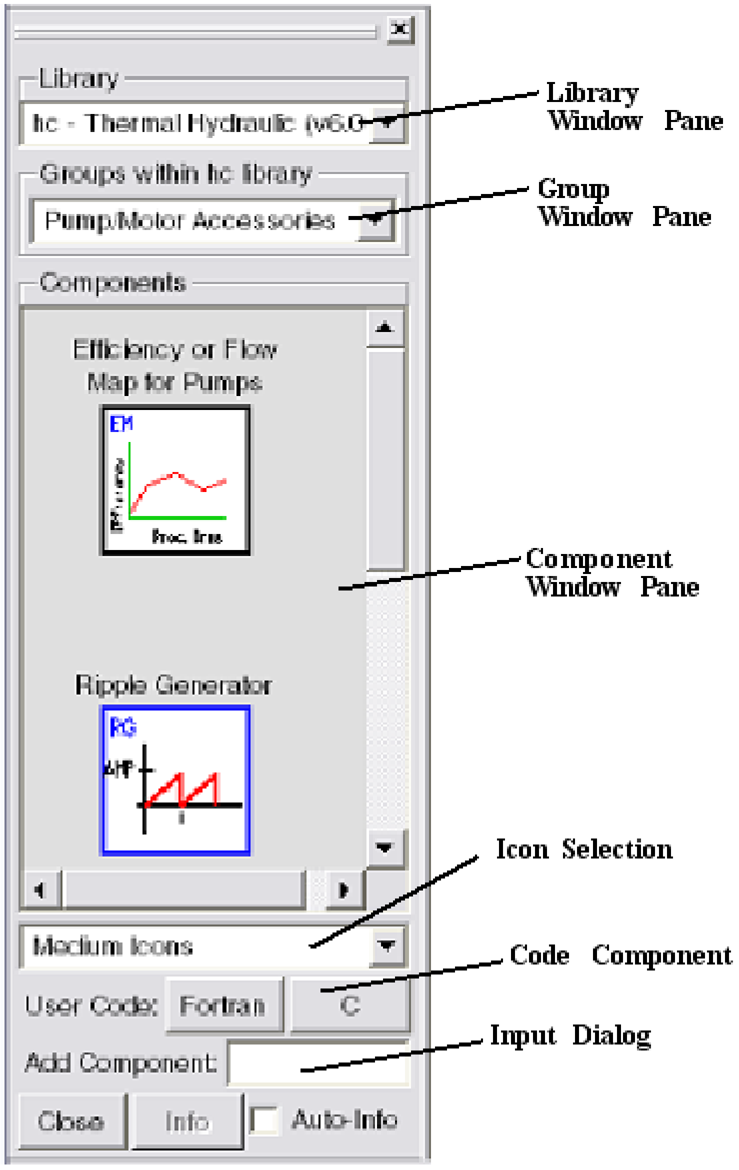


Figure 1 Add Components Window

The third window pane lists each component of the selected library. You select the manner in which these icons are displayed using the icon selection option. This option lets you select Text Only components or Small, Medium, or Large Icons. At the bottom of the list are the user code buttons for [`Fortran`] and [`C`] components. The code components are always listed because they do not belong to a specific library.

Whenever you select a library and group, all the window panes are updated with information for that particular library. When you select a component from the Components window pane, that component's library/name is automatically written into the "Add Component" input dialog at the bottom of the window. This identifies it as the next component to be added to the schematic block diagram.

Note:	You can only add components to the model from (licensed) libraries for which you have checked out the appropriate build license feature.

For example, if the Ripple Generator component is selected from the hc library, the component name RG is added to the text box.

A component name can have from two to four alphanumeric characters. The first two characters are taken from the component name given by the library. Two additional characters are used to assign a unique component name; these characters must be alphanumeric. If this is the first component being added to your model, Easy5 will name the component with those two characters only, such as RG. If a second RG component is to your model, Easy5 would automatically add an identifier to the component, such as RG2.

You can edit the assigned component qualifier by selecting and highlighting the component name in the Add Component text box. For example, if Easy5 automatically assigns the AC component to be named AC2, you may edit the name and change it to ACXX, or any other two alphanumeric characters, to provide a unique component name.

If the cursor is moved back into the Add Components window before releasing the mouse button and dropping the component into the schematic, the "add" function is deactivated, and that component is not added. You can then choose another component by selecting it's icon again. Press **Esc** to deselect the component.

With the Add Components window open, it is possible to quickly add many components to your model. When you have finished using the Add Components window, close the window by selecting the `[Close]` pushbutton.

Online Component Information

Online documentation can be obtained for any component. To obtain information about a component, first select the component, then select the `[Info]` button. An "Information" page opens describing the component's inputs/outputs, equations and other pertinent information. The info page can be one of four data formats: internal rich-text, HTML, PDF or Easy5 format.

Note:	Components do not necessarily have Info pages. Info pages are created by the library developer, and if not developed, the Info page will not appear when requested.

To automatically display Info pages, select the "Auto-Info" radio button. This automatically displays the info page for any component you select from the Add window. This feature allows you to view many Info pages without having to select the Info button for each component. Before adding components, turn off the Auto-Info radio button.

Adding Components to the Schematic

1. Select the component with the left mouse button.

   As the mouse is moved, the cursor changes shape. It becomes a square with the component's name inscribed.

2. Move the cursor to the location where the component is to be added

3. Drop it into place with a CLICK-L.

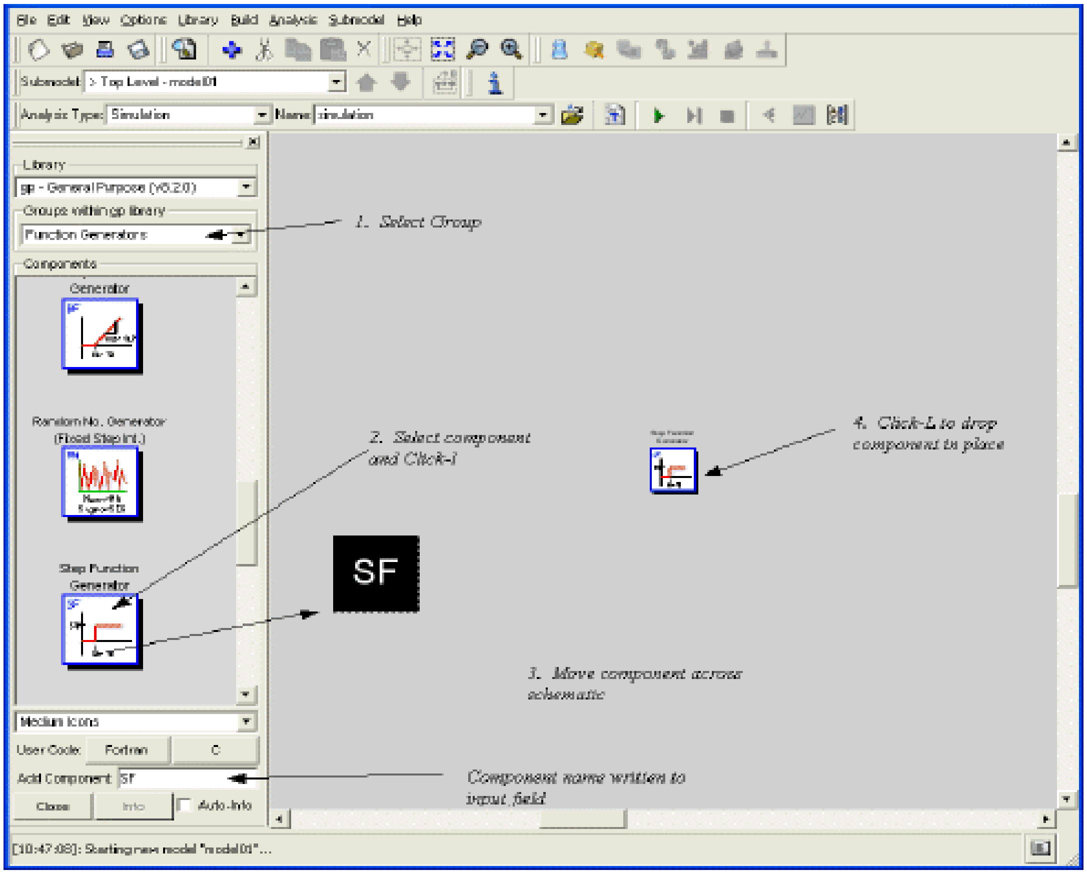
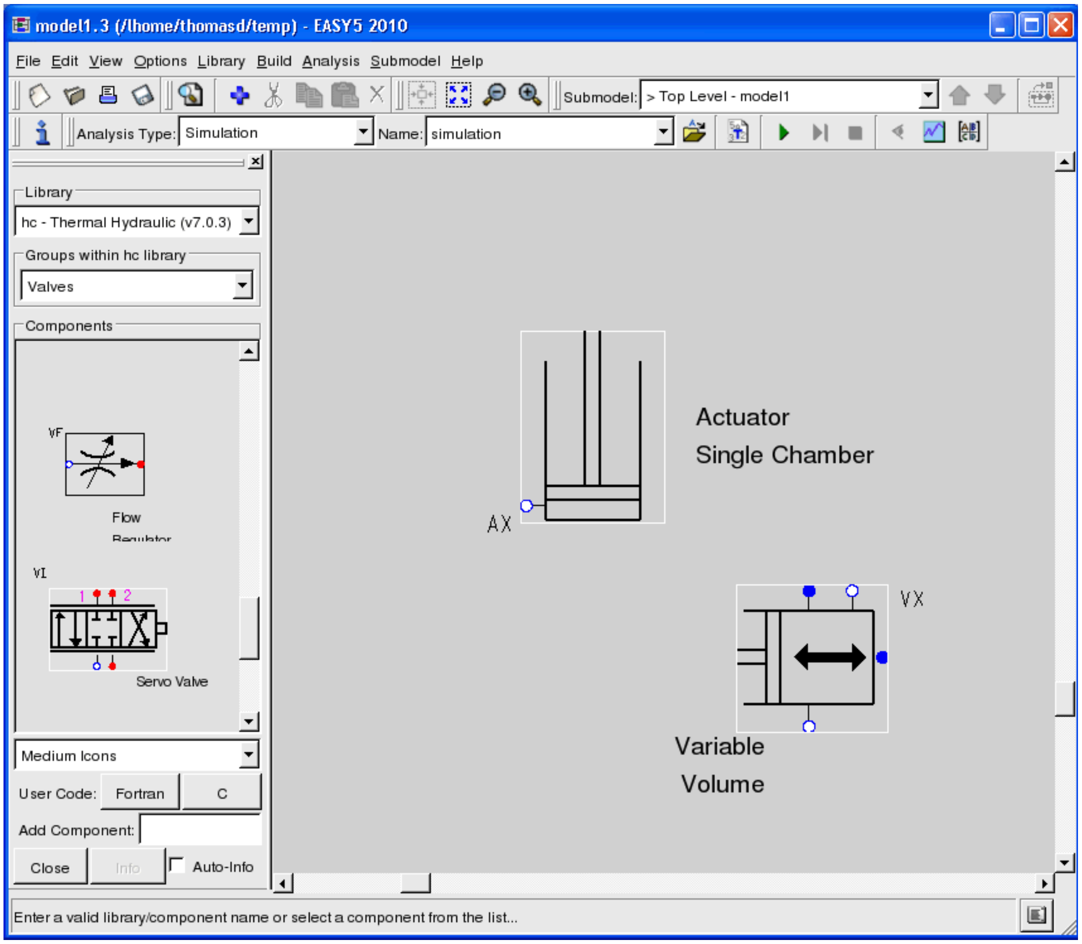

File Edit View Options Library Build Analysis Submodel Help
Submodel: > Top Level - model01
Analysis Type: Simulation
Name: simulation
Library
gp - General Purpose (v6.2.0)
Groups within gp library
Function Generators
Components
Generator
Random No. Generator (Fixed Step Int.)
Step Function Generator
Medium Icons
User Code: Fortran C
Add Component: SF
Close Info Auto-Info
SF
1. Select Group
2. Select component and Click-l
3. Move component across schematic
4. Click-L to drop component in place
Component name written to input field
[10:47:08]: Starting new model "model01"...

Figure 2 Steps to Add a Component

One advantage of using the icon-based Add Components menu with the Gas Dynamics Library (gd) or the Thermal Hydraulic Library (hc), is that you can quickly determine whether a specific component can be connected to your model, that is whether it is a *resistive* or *storage* port.

Example:

Using the icon-based Add Components menu with the Thermal Hydraulic Library (hc), you can determine if a component will connect to the current model by viewing the inlets, and using the one you need, be it a resistive or storage inlet.



Enter a valid library/component name or select a component from the list...

Figure 3 Icon-based Add Window

Add Components by Name Reference

You can also add components to the schematic pad by entering the component's two character name into the input data field at the bottom of the Add Window.

Use of this method assumes familiarity with the library and component names, since some libraries may have a component that has the same name as another component from a different library.

For example, the Integrator component in the gp library and the Ram Air Inlet component in the ec library are both named IN. To specify which component to add you must enter the library name, a forward slash ("/"), then the component name. The component name entered into this input dialog is not case sensitive. For example, you can either enter `ec/IN` or `EC/in`. Once entered, the Add Menu automatically updates the selected library and the Add Component text field.

Note:	If you wish to add a component from a library other than the "gp" library, and the component name is the same as one from the "gp" library, then you must specify the name of the library. If you do not, Easy5 will add the component from the currently selected library.

Adding a Code Component

Both Fortran and C code can be added to a model using the special User Code components as follows:

- Select the `[Fortran]` User Code button in the Add Components window to add a Fortran User Code component.
- Select the `[C]` User Code button in the Add Components window to add a C User Code component.

The component qualifier can be specified before adding the component to the model, in the same manner as any other library component. Easy5 will ensure that any component added to your model has a unique component name.

Analyses

The Easy5 analysis program provides you with the capability to perform a set of linear and nonlinear analyses on the same executable model. The following sections briefly describe these analyses and their features. For more information on these topics, refer to the analysis by name in this reference manual.

Note:	All analyses require checkout of an Easy5 Analysis license feature.

Nonlinear Analyses

Nonlinear analyses options include the capability to perform simulation and steady-state analyses on your system model. These analyses, listed below, are described in detail in this manual.

- "Steady-State Analysis"

Simulation provides a time history of the dynamic behavior of your system model. The Steady-State analysis locates values for model states that cause system rates (state derivatives) to equal zero. This is also referred to as the equilibrium condition.

Linear Analyses

Easy5 generates a linear approximation of a nonlinear model, and uses the approximation for linear analyses. The linear analyses provide insight into the stability and performance of the nonlinear system at various operating points. The advantage of using linear analyses is that model behavior can be predicted relatively quickly.

Easy5 provides a variety of linear analysis tools. These analyses, listed below, are described in detail in this manual.

- "Transfer Function Analysis"

  Transfer Function analysis permits you to calculate the poles, zeros, leading coefficient, and frequency response between any two points in your system model.
- "Root Locus Analysis"

  Root Locus analysis provides you with the capability to determine the locus of the system model eigenvalues as a function of *any system* variable.
- "Eigenvalue Sensitivity Analysis"

  Eigenvalue Sensitivity analysis measures the sensitivity of system eigenvalues to changes in a user specified system parameter.
- "Stability Margins Analysis"

  Stability Margins analysis calculates maximum and minimum values for user specified parameters which maintain system stability.
- "Linear Model Generation Analysis"

  The basic Linear Model Generation analysis includes the following:

  - Jacobian matrix calculation
  - Eigenvalues for the Jacobian

The Linear Model Generation analysis can also be used to calculate a complete linear model of the form:

$$\dot{\mathbf{x}} = \mathbf{A}\mathrm{x} + \mathbf{B}\mathrm{u}$$

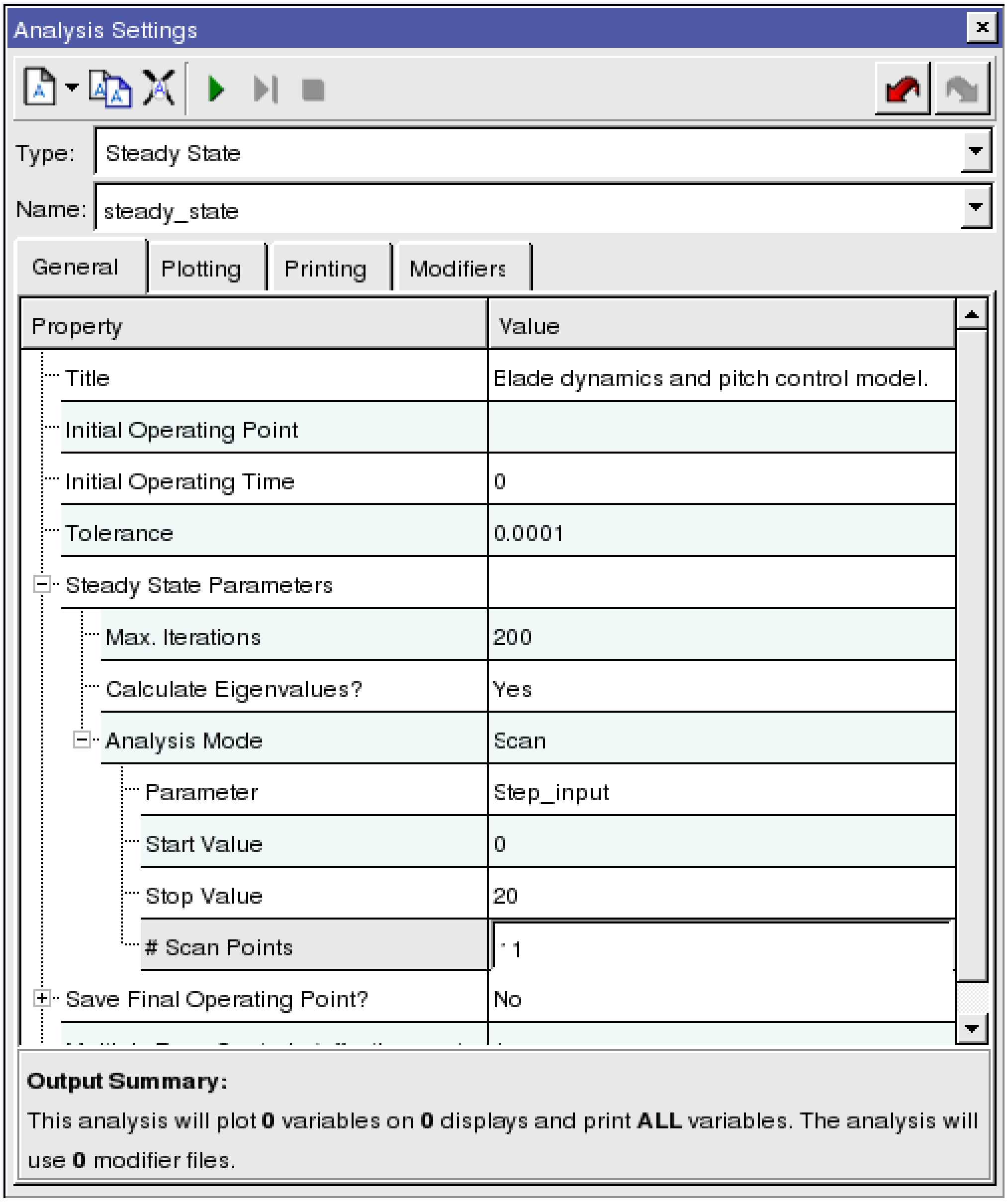
$$\mathrm{y} = \mathbf{C}\mathrm{x} + \mathbf{D}\mathrm{u}$$

This form of the linear model is generated when you specify an input vector **u** and an output vector **y** before the analysis is executed. This analysis also calculates the eigenvectors of the linear model in the form of a model matrix, which relate the modes of the model to model states. If you have a sampled data model, a discrete form of the linear model is generated.

Analysis Data Form

All Easy5 analyses are set up and executed using analysis data forms.

A typical analysis data form is shown in Figure 4. The data fields and options in these forms vary depending on the type of analysis requested.



Type: Steady State

Name: steady_state

Property	Value
Title	Elade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Tolerance	0.0001
Steady State Parameters	
Max. Iterations	200
Calculate Eigenvalues?	Yes
Analysis Mode	Scan
Parameter	Step_input
Start Value	0
Stop Value	20
# Scan Points	1
Save Final Operating Point?	No

Output Summary:

This analysis will plot **0** variables on **0** displays and print **ALL** variables. The analysis will use **0** modifier files.

Figure 4 Typical Analysis Data Form

Analysis Data Form Header

All analysis data forms contain a header appearing at the top of the data form. The header is a one line description of the data form, made up of two parts:

1. The level and title of the model opened, such as Top Level - blade_pitch_4, shown in Figure 5 under submodel.
2. The analysis type, such as Simulation, Steady State, Linear Model, Transfer Function, Root Locus, and so on as shown in Figure 5.

Each analysis file is given a default name corresponding to the type of analysis. For example, by default, the simulation analysis data form name is: simulation. This name may be changed by the user, as described in the next section.

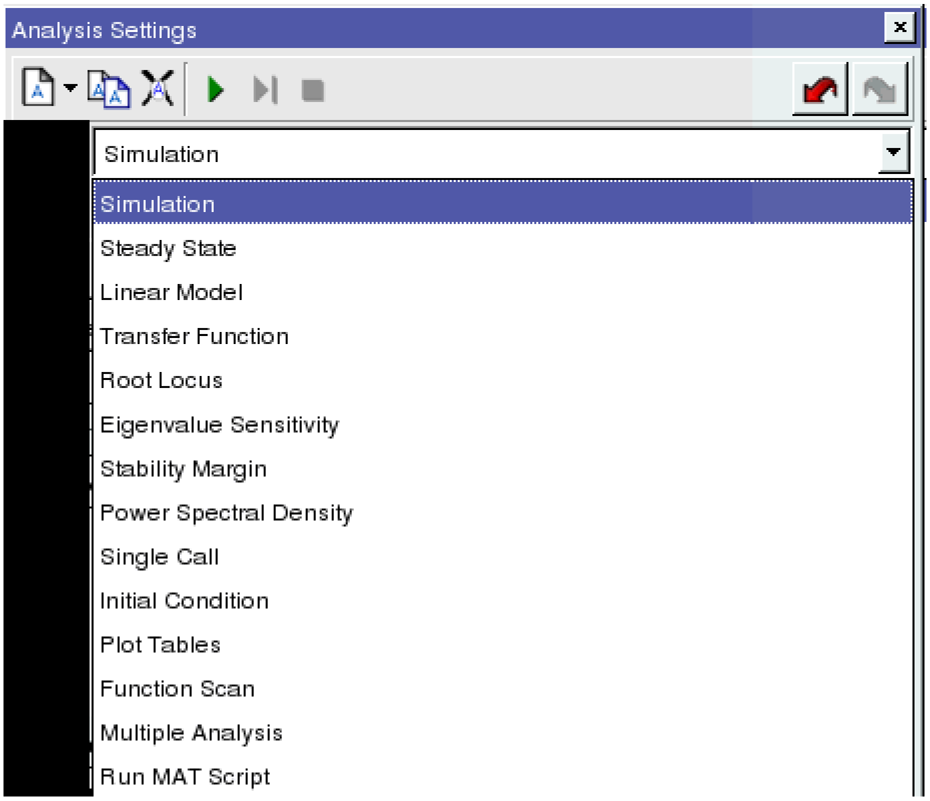


Figure 5 Analysis Data Form Options

Analysis Data Form

Analysis Settings File

All analysis data forms are given a standard filename. Every time you run a new analysis and change the analysis data form, the previous data is overwritten and lost. However, you can create and save multiple analyses by creating new analysis files.

Figure 5 shows an example of a copied analysis (simulation_copy) and a new analysis (simulation1) in addition to the original analysis settings file (simulation).

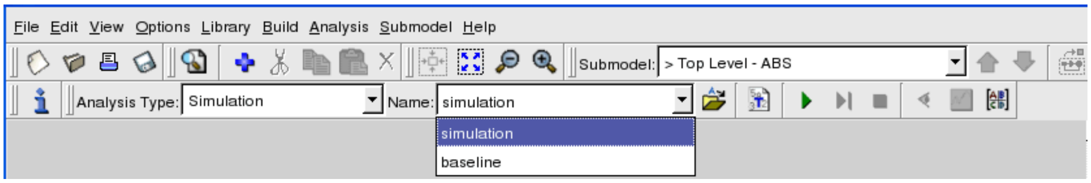

File Edit View Options Library Build Analysis Submodel Help
Submodel: > Top Level - ABS
Analysis Type: Simulation
Name: simulation
simulation
baseline

Figure 6 Analysis Settings Files Options

To save the current analysis data form, select the Execute button (green arrow button), and press Enter. To create a new analysis, select the New icon as shown in Figure 6; the new analysis is automiatically incremented by **1**. To copy the current analysis, press the Copy icon as shown in Figure 6 and press Enter. The default name automatically add a suffix of **_copy** to the original file name. This new analysis is added to the list of analysis files. Multiple analysis files can be created in this manner and saved in a database.

Settings files can be manipulated by using the scroll button to view the files, and then selecting the file and the desired action of either delete, copy, or rename.

Note: The analysis file name defined by the user is also used in the naming of the analysis output file (the "ezapl" file). The naming convention of the analysis output file is as follows: <*modelname*>.<*name*>.ezapl.

Temporary Settings File

The Temporary Settings File is an external data table containing a data base of component input parameters and state values. It is used to temporarily modify the data defined in the components' data table, and to apply this temporary modification to any analysis.

This option is used to create, copy, rename and delete Temporary Settings files. See the main topic "Temporary Settings File" for information on how to setup and use this option.

Auxiliary Input File

The Auxiliary Input File gives you an alternate means of entering data and analysis commands. It is a separate file that you create using standard Easy5 analysis "command line" commands. These files are commonly used to input large sets of data, such as matrices and tables. This menu option is used to create, copy, rename and delete Auxiliary Input files. See "Auxiliary Input File" for information on how to setup and use this option.

Analysis Title

You can change the analysis title by selecting the Title data field and entering a different title. The information in the "Title" field will appear at the top of the analysis output listing file that is created when you execute the next analysis. This title will also appear in any plots produced during the analysis. Easy5 automatically defaults the analysis title to whatever you entered in the model description line at the top of the main Easy5 schematic window.

Time of the Analysis

You define the time at which the analysis is executed with the “Time” value in the analysis data form. The default value of zero is usually appropriate. However, if you have time dependent functions in your model (e.g., table lookups as a function of time) and you wish to establish an equilibrium point at a time value other than zero, you should set this value appropriately. To set the “Time” value, select the data field following “Time =”, and type in a new value.

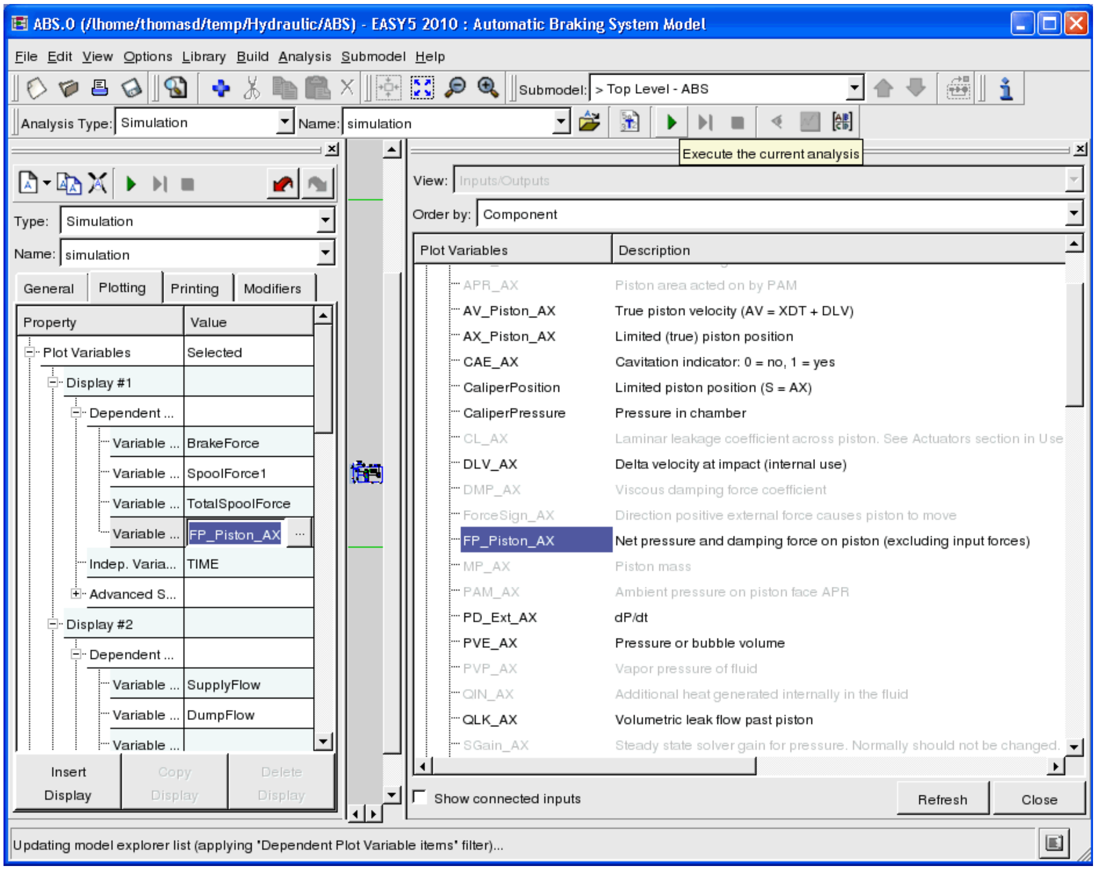
Initial Operating Point

An operating point that had been previously generated can be restored to perform the analysis at this operating point. For example, a previously generated steady-state operating point might be used as a starting operating point when performing a simulation.

This is done by selecting the “Initial Operating Point” input to brings up a menu that displays all available operating point files. These are files that were either created via the **Options > Save Operating Point** menu, or, from changing the default to Yes in the Save Final Operating Point? value in the analysis form. Further information on saving and restoring operating points is given in "States: Defining Values and Controls".

Model Explorer “Pickable” Fields

Text fields that cannot be "picked" are grayed out in the Model Explorer window as shown Figure 7. Such model names are inappropriate for the filter used.



ABS.0 (/home/thomasd/temp/Hydraulic/ABS) - EASY5 2010 : Automatic Braking System Model

File Edit View Options Library Build Analysis Submodel Help
Submodel: > Top Level - ABS
Analysis Type: Simulation
Name: simulation
Execute the current analysis
Type: Simulation
Name: simulation
General Plotting Printing Modifiers
Insert Display
Copy Display
Delete Display
View: Inputs/Outputs
Order by: Component
Show connected inputs
Refresh
Close

Property	Value
Plot Variables	Selected
Display #1	
Dependent ...	
Variable ...	BrakeForce
Variable ...	SpoolForce1
Variable ...	TotalSpoolForce
Variable ...	FP_Piston_AX
Indep. Varia...	TIME
Advanced S...	
Display #2	
Dependent ...	
Variable ...	SupplyFlow
Variable ...	DumpFlow
Variable ...	

Plot Variables	Description
APR_AX	Piston area acted on by PAM
AV_Piston_AX	True piston velocity (AV = XDT + DLV)
AX_Piston_AX	Limited (true) piston position
CAE_AX	Cavitation indicator: 0 = no, 1 = yes
CaliperPosition	Limited piston position (S = AX)
CaliperPressure	Pressure in chamber
CL_AX	Laminar leakage coefficient across piston. See Actuators section in Use
DLV_AX	Delta velocity at impact (internal use)
DMP_AX	Viscous damping force coefficient
ForceSign_AX	Direction positive external force causes piston to move
FP_Piston_AX	Net pressure and damping force on piston (excluding input forces)
MP_AX	Piston mass
PAM_AX	Ambient pressure on piston face APR
PD_Ext_AX	dP/dt
PVE_AX	Pressure or bubble volume
PVP_AX	Vapor pressure of fluid
QIN_AX	Additional heat generated internally in the fluid
QLK_AX	Volumetric leak flow past piston
SGain_AX	Steady state solver gain for pressure. Normally should not be changed.

Updating model explorer list (applying 'Dependent Plot Variable items' filter)...

Figure 7 Model Explorer Window

A Model Explorer window is automatically opened from an analysis form by clicking on the elipsis(“...”) of the data value that you want to enter. This puts you into “pick mode”, which allows you to simply select the appropriate model input/output name in the Model Explorer window that value is automatically entered into the data analysis form. When in “pick mode”, Easy5 automatically filters the model explorer view appropriate to the selection.

You can navigate your model using the Model Explorer “view” of your model, or the schematic itself -- both windows will track each other. However, selection is done using the model explorer window.

You can also enter a model name by selecting a component from your model schematic, and selecting the appropriate input/output name from the corresponding list. Once selected from the list, the name is automatically inserted into the text field. Both these methods are often faster and more accurate than manually typing in names.

Auxiliary Input File

Auxiliary input files give you an alternate means of entering Easy5 data and analysis commands. The auxiliary input file is a separate file that you create using standard Easy5 analysis commands (see Ap. A: Summary of Analysis Commands).

Auxiliary input files are most commonly used to input large sets of data, such as matrices and tabular data. During the execution of an analysis, Easy5 reads in data from the auxiliary input file, and uses this data to override the existing data defined in the Component Data Tables.

See Also:"Auxiliary Input File Data Format"

Ap. A: Summary of Analysis Commands

Creating an Auxiliary Input File

To create an auxiliary input file, select **File > Auxiliary Input File > New** to open an input dialog. The model_name is automatically inserted into the data input field. The format for an auxiliary input file is:

<*modelname*>.<*name*> or <*name*>

or as a label in an existing file as

<*modelname*>.<*name*> $<*label*> or <*name*>$<*label*>

All auxiliary input files are automatically saved with a suffix “ezax”. Therefore, the auxiliary input file is saved in your directory as:

<modelname>.<name>.ezax

Note: The inclusion of a *modelname* as part of the auxiliary file name is optional. However, using a *modelname* makes it easier to identify those auxiliary input files associated with a specific model.

After entering a file name, select OK; the Easy5 Text Editor window opens to allow you to enter and edit Easy5 analysis commands. An example of an auxiliary input file is shown in Figure 8. See “Steady-State Analysis Method” for more information on the Easy5 Text Editor.

```
PARAMETER VALUES
* Signal generator AF:
CODAF = 5,Step_input = 5,C2_AF = 0,C3_AF = 0,C4_AF = 0,C5_AF = 0
GKI_IN = 1.4    K_GF = 1,SLOPE = 0.031,MAXFLO = 1.55
* Define vector:  Servo_Data(1) -> (5)
Servo_Data= 23.456, 19.55, 12.3, 9., 5.005, 2.
* SERVO VALVE DATA TABULAR DATA
TABLE, FTA_FU,21
-50,-45,-40,-35,-30,-25,-20,-15,-10,-
5,0,5,10,15,20,25,30,35,40,45,50
-1.55,-1.5,-1.38,-1.19,-1,-0.74,-0.5,-0.26,-0.12,-
0.05,0,0.05,0.12,0.26,0.5
0.74,1,1.19,1.38,1.5,1.55
INITIAL CONDITIONS,Actuator_pos = 5,Pitch_angle = 5,X1_TF = 50
ERROR CONTROLS,Actuator_pos = 0.001,Pitch_angle = 0.001
INT CONTROL,Actuator_pos = 1,Pitch_angle = 1,X1_TF = 1,Current = 1
```

Figure 8 Example of an Auxiliary Input File

This auxiliary input file defines scalar data, vector data, and a table. It also is used to setup state initial conditions and error controls. Information on how to enter data into this file is given in "Auxiliary Input File Data Format".

This file is linked to the analysis data form in the same manner as the temporary settings file is linked.

Select **File >Auxiliary Input File** from the main menu to open a dialog that lets you perform other tasks, such as creating, opening, copying, renaming, or deleting an auxiliary file.

Inline Comments

Inline comments are supported as Easy5 analysis commands. Any character after an exclamation mark (“!”) is ignored during processing, allowing you to enter “inline” comments, as shown below:

```
PARAMETER VALUES
GKI_IN = 1.3  ! new value added based on recent test
```

Using an “*auxfile*” To Enter Blocks of Data

A special auxiliary file (identified by environment variable "*auxfile*") may be created to contain labeled groups (or blocks) of data and/or Easy5 analysis commands. The *auxfile* is generally used to manage large number of data sets and/or to initialize the model parameters. For example, hundreds of different flight conditions may be used to simulate an aircraft in different flight modes. Each flight condition may be defined by a different altitude and Mach number, and the corresponding aerodynamic derivative data.

The entire data base can be put into a single *auxfile*, and each flight condition defined by a label name.

The steps for creating an *auxfile* and inputting labeled data sets is as follows:

1. Create an *auxfile* (outside Easy5) and add data and Easy5 analysis commands as needed.

2. Define data blocks with label names. The label name should start with a “$” in column one, and the name should not exceed ten characters (e.g. $labelname).
3. Before invoking Easy5, at a command prompt enter:

```
export auxfile=<auxfile_name>          {Bourne shell}
setenv auxfile <auxfile_name>          {C shell}
set auxfile=<auxfile_name>             {Windows}
```

4. Run Easy5, and create an auxiliary input file to load in desired data blocks. In the auxiliary input file, you can request one or more blocks of data, by using the following command:

```
AUX INPUT = $label_name
```

More than one block label may be requested in an auxiliary input file. When the requested label is located, the commands that follow the label are executed until another label or end of file is reached.

In the following paragraphs, an example is used to show how an auxfile can be used to input different blocks of data. These blocks contain data that models a servo valve as a nonlinear function using the tabular data component “FU”. Assume that different servo valves need to be considered in the design, and that using a single FU component, you would like to perform a trade study to analyze the effect of different servo valves. You may create different temporary settings files, each containing a different
set of servo valve data. Or, you may create an *auxfile* containing a single data base with different servo valve data.

To do this, create an ASCII text file and add the data as shown in Figure 8. This file, named *servo_data*, contains three data blocks named: SERVO1, SERVO2, and SERVO3. Each block defines the tabular values required for the table named FTA_FU01.

Specifying a Label in an Auxiliary Input File

You may also specify a label directly when entering new auxiliary input file using the following syntax:

```
<modelname>.<name>$<label>
```

or

```
<name>$<label>
```

This specifies inclusion of $<label> data from an existing auxiliary input file <modelname>.<name>.ezax or <name>.ezax.

```
$SERVO1
* DATA FOR SERVO MODEL 1
 PARAMETER VALUES
 TABLE, FTA_FU01, 21
-50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0.  5, 10, 15, 20, 25, 30, 35, 40, 45, 50
-2.32, -2.25, -2.07, -1.78, -1.25, -.74, -.5, -.26, -.12, -.05, 0., .05, .12, .26, .5, .74, 1.25,
 1.78, 2.07, 2.25, 2.32
 $SERVO2
 * DATA FOR SERVO MODEL 2
 PARAMETER VALUES
 TABLE, FTA_FU01, 21
-50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0.  5, 10, 15, 20, 25, 30, 35, 40, 45, 50
-3.50, -3.25, -2.50, -1.90, -1.52, -.74, -.5, -.26, -.12, -.05, 0., .05, .12, .26, .5, .74, 1.52,
 1.90, 2.50, 3.25, 3.50
 $SERVO3
 * DATA FOR SERVO MODEL 3
 PARAMETER VALUES
 TABLE, FTA_FU01, 21
-50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0.  5, 10, 15, 20, 25, 30, 35, 40, 45, 50
-4.32, -4.25, -3.07, -2.78, -2., -1.74, -1.0, -.26, -.12, -.05, 0., .05, .12, .26, 1.0, 1.74,
 2., 2.78, 3.07, 4.25, 4.32
```

Figure 9 auxfile: servo_data

Before starting Easy5, the *auxfile* environment variable must be defined by entering the following command :

```
export auxfile=servo_data {Bourne shell}

set auxfile=servo_data    {Windows}
```

Run Easy5, and create an auxiliary input file. In this example, the auxiliary input file will be named *aux_servo*. This auxiliary input file will be used to load in the desired data set from the *servo_data* auxfile. For example, if a simulation required the servo valve data contained in the second labeled data ($SERVO2), the auxiliary input file (*aux_servo*) would contain the following line:

```
AUX INPUT = $SERVO2
```

Having created the auxiliary input file named *aux_servo*, you must request that the file be used in the simulation. Just select the first data field after the “Aux. Input Files:” data field and select the *aux_servo* file.

Auxiliary Input File Data Format

The Auxiliary Input file is an additional file that you can access running any analysis. This file may contain model data and/or analysis commands. When an analysis is executed, the data from this file overrides the default data defined in the model’s components, and is only applied to the analysis. As a result, the model’s data is not affected by the usage of an Auxiliary Input File.

The Auxiliary Input File is generally used to input external data, usually large sets of scalars, vectors, matrices and table data. This section describes only the command syntax used to load model data into an Auxiliary Input File. The analysis commands that may also be input, are described in Ap. A: Summary of Analysis Commands.

See Also:"Auxiliary Input File".

PARAMETER VALUES Command

The first line in the auxiliary input file should contain the following command:

```
PARAMETER VALUES
```

The **PARAMETER VALUES** command is followed by one or more parameter names, each followed by a numeric value containing up to 20 characters, or an expression that fits on the current line. Each name and its value are separated by one of the standard delimiters (comma, =, <Tab>, or 3 or more spaces). You use this command to specify the values of all parameters at the beginning of an analysis and, at any point between analyses, to modify the value of one or more model parameters.

A warning message will be printed at the beginning and the end of each Easy5 Analysis "run" if one or more model parameters have not been initialized. A default value of .99999 is provided for all parameters not specified. Parameters corresponding to certain standard components may default to other, more appropriate values, as noted in their component description. For example, the saturation limits on the (soft) limited integrator component, **IT**, will default to +/- 10^{36}. Special default values such as these are used only if you do not change the parameters from the standard default value of .99999, and if they have special initialization code in the particular component.

The following example shows the use of the **PARAMETER VALUES** command:

```
PARAMETER VALUES
MASS = 18000.95      C1_MA3 = 1.0E6,C2_MA3 = .0059
COMLEV = 25,MAXLEV = 30.7    MINLEV = 5
```

Scalar Parameter Data

The above example also shows how to input scalar parameter data. Just enter the parameter name followed by "=" and the data value. A parameter may be entered on a single line, or multiple names per line using a delimiter to separate the different parameters. A delimiter can be: three or more spaces, a <Tab>, or a comma.

If integer values are to be input, the decimal point may be omitted (however, all parameter quantities are stored as real Fortran variables). Exponential (or scientific) notation may also be used as shown by the **1.0E6** entry, which corresponds to 10^6. This notation consists of a decimal number and the letter **E** followed by a positive or negative integer.

Note:	The maximum auxiliary input file line width is 128 characters.

Array Parameter Data

Array parameters can be one or two dimensional arrays. The array input format must contain the array name, the input mode, and the appropriate array elements. In the following examples, a 0 is displayed in the arrays for undefined elements (or have no values loaded).

You can load parameter arrays following the **PARAMETER VALUES** command using one of several methods.

These methods inlcude:

Mode	Designation
Column	C (Default)
Row	R
Diagonal	D
Element	none
Zero	Z
Infinite	I

Note:	Modes are designated immediately after the array name is given. Therefore, arrays cannot be assigned the names **C**, **R**, **D**, **Z**, or **I**; this could cause mode designations to be misinterpreted. To guard against this, **these parameter names are Easy5 reserved words**. See "Reserved Words" for a complete list of reserved words.

The examples below show the various modes for loading parameter arrays (following a **PARAMETER VALUES** command).

Loading down a COLUMN

`ADATA,C(1,1)1,2,3,4,5`	Starts at element 1,1 and loads column one
`ADATA,C(1,2)6,7,8,9,10`	Starts at element 1,2 and loads column two

$$ADATA = \begin{bmatrix} 1 & 6 & 0 & 0 \\ 2 & 7 & 0 & 0 \\ 3 & 8 & 0 & 0 \\ 4 & 9 & 0 & 0 \\ 5 & 10 & 0 & 0 \end{bmatrix}$$

Loading across a ROW

`BDATA,R(2,3)7,8,9,10`	Starts at element 2,3 and loads row two
`BDATA,R(1,2)3,6,9,10`	Starts at element 1,2 and loads row one

$$BDATA = \begin{bmatrix} 0 & 3 & 6 & 9 & 10 & 0 \\ 0 & 0 & 7 & 8 & 9 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Loading down and across a DIAGONAL

`COEF,D(2,4).3,.4,.5`	Starts at element 2,4 and loads elements(2,4),(3,5) and (4,6)

$$COEF = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

Loading by ELEMENT

```
ADATA(1,2)= 12          ADATA(3,1)= 16     ADATA(2,4)= 21
```

This loads elements (1,2), (3,1) and (2,4).

Note: You must use "(" as the delimiter immediately following the array name when you use the element input mode.

ZERO the whole array

```
COEF,Z,R(2,2)1,2,3
```

$$ADATA = \begin{bmatrix} 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 21 \\ 16 & 0 & 0 & 0 \end{bmatrix}$$

The zero array command, Z, may be combined with any of the other modes, e.g., zero array and then load by row.

$$COEF = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Set array to INFINITE, “Infinite” = 10^{36}

```
COEF,I
```

$$COEF = \begin{bmatrix} 1.E36 & 1.E36 & 1.E36 & 1.E36 & \ldots & \ldots\cdot\ldots & 1.E36 \\ 1.E36 & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & 1.E36 \\ 1.E36 & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots \\ \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots \\ \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & 1.E36 \\ 1.E36 & 1.E36 & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & \ldots\cdot\ldots & 1.E36 \end{bmatrix}$$

Input by column starting at element 1,1 (Default)

```
     VECTOR = 1,2,3,4,5
```

$$VECTOR = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

If an input mode is not specified, data will be loaded by column starting at element (1,1). This default may be used for either "vector" (one dimensional) or "matrix" (two dimensional) arrays.

If the default mode is used for a matrix, the values are loaded in successive columns starting with column one. In all input modes, the dimensions of the array will be checked for each input value to ensure that all input elements are within the bounds of the array dimensions.

Expressions

Parameter values and initial conditions in analysis input and auxiliary input files also may be specified in terms of an expression delimited by braces "{" and "}". Between the braces the standard delimiters (comma, =, <Tab>, or 3 or more spaces) are ignored, and the expression is numerically evaluated during initialization. The following are allowed in expressions:

- Parameter value names
- State names
- Any operator allowed by Python: +, - , *, /, ** and so on.
- Any function contained in the Python math module (exp, log, sin, cos and so on). Refer to the Python math module documentation for a complete list of functions.

Here is a simple example of expressions in `PARAMETER VALUES` commands:

```
PARAMETER VALUES
MASS = {VOL * DENSITY)
MINLEV = 5    MAXLEV = {MINLEV + 32}   COMLEV = {(MINLEV + MAXLEV)/2}
VECTOR = 1,2,{VECTOR(1)+VECTOR(2)},4,{VECTOR(2)*2+VECTOR(1)}
```

In this example, all of the variables in the expressions are Easy5 parameters that appear in the Inputs tab of a component data table (CDT). State names could also be specified. In the expression containing MAXLEV on the right hand side, the value of MAXLEV used is the value calculated in the previous command where it appears of the left hand side.

Each expression, after numerical values have been substituted for the parameter or state names, is evaluated by Python, hence the expression must be valid Python. For each successfully evaluated expression, the numerical value will appear in the analysis output listing (.ezapl):

```
COMMAND -----> PARAMETER VALUES
COMMAND -----> TC_LA4 = 0.6623,{TC_LA2(2)*2*0.5},{asin(sin(TC_LA2(3)))}
EXPRESSION --> TC_LA2(2)*2*0.5 = 0.66206768824626
EXPRESSION --> asin(sin(TC_LA2(3))) = 0.66230000000000
```

If there is an error in evaluating the Python expression then that will also appear in the analysis output listing.

If your model contains parameter or state names that overload Python math module function names, then the Easy5 names will take precedence. For example, if exp is the name of an Easy5 input, but you need to take an exponential in your expression, then the results will likely not be as expected, or the evaluation of the resulting Python expression will result in an error.

Finally, please note that, in general, auxiliary input files have more capability than analysis conditions specified in GUI objects. That also applies here: expressions in auxiliary input files have more power and flexibility than expressions entered as CDT Input values in the Easy5 schematic. Schematic expressions do not allow for the inclusion of state names, and cannot be entered as initial conditions in the States tab.

Tabular Data

To load tabular data required by the system model, you use the following command:

```
TABLE
```

When you build your model, you specify all tables that require data to be loaded. These tables can have from one to nine independent variables. All data can be entered in a free field format.

The data items, which can each contain up to 20 characters, are separated by one or more standard delimiters. The data items required for each table are placed in one of the formats described below.

Arbitrarily Spaced Data

For arbitrarily spaced, independent variable tables, the table data format is as shown below.

The format for an nth-order independent variable table is:

```
TABLE, table_name, N1, N2, N3, ... , Nn
Nn values of nth independent variable table data
...
...
N2 values of 2nd independent variable table data
N1 values of 1st independent variable table data
Dependent variable table data (N1 x N2 x ... Nn values)
```

Note:	The information **TABLE, table_name, N1, N2, N3, ... , Nn** must be placed on a single line. For each of the other lines, the information may be placed on as many lines as required.

The total number, T_a of data values required is given by:

$$T_a = max(3, n+1) + \sum_{i=1}^{n} N_i + \prod_{i=1}^{n} N_i$$

The dependent variable table values are ordered by varying the first independent variable fastest, then scanning the second independent variable next fastest, then scanning the third independent variable next slowest and so on, with the nth independent variable being varied slowest.

For example, the dependent values of a three dimensional table with two values for each independent variable x,y, and z would be loaded in the following order:

f(x1,y1,z1), f(x2,y1,z1), f(x1,y2,z1), f(x2,y2,z1)

f(x1,y1,z2), f(x2,y1,z2), f(x1,y2,z2), f(x2,y2,z2)

A copy of all tabular input data is printed both as it was entered and in a formatted representation *only* if you have entered the **PRINTBACK TABLES** command.

The following example shows the data for a one and a two independent variable table printed back.

Line1	PRINTBACK TABLES	
Line2	TABLE, TAB_ONE, 10	
Line3	1,2,3,4,5,6,7,8,9,10	(First independent variable table)
Line4	11,12,13,14,15,16,17,18,19,110	(Dependent variable table)

Line5	TABLE, TAB_TWO,5,3	
Line6	10.3,20.4,30.5,	(Second independent variable table)
Line7	1,2,3,4,5	(First independent variable table)
Line8	11,12,13,14,15	(Dependent variable table, start....)
Line9	21,22,23,24,25	(Dependent variable table, cont. ...)
Line10	21,32,33,34,35	(Dependent variable table, cont. ...)

The printout of these tables would be:

```
          TABLE TAB_ONE
      TABLE OF INDEPENDENT VARIABLE NO. 1
      1.000 2.000  3.000  4.000  5.000  6.000  7.000  8.000  9.000 10.00
  DEPENDENT VARIABLE TABLE
      11.00 12.00 13.00 14.00 15.00 16.00 17.00 18.00 19.00 110.00
          TABLE TAB_TWO
      TABLE OF INDEPENDENT VARIABLE NO. 2
      10.30    20.40      30.50
      TABLE OF INDEPENDENT VARIABLE NO. 1
      1.000    2.000      3.000      4.000      5.000
      DEPENDENT VARIABLE TABLE
      11.00    12.00      13.00      14.00      15.00
      21.00    22.00      23.00      24.00      25.00
      31.00    32.00      33.00      34.00      35.00
```

Fixed Spaced Data

Because a fixed spaced table has all values of the independent variable at fixed increments, storage requirements are less than an arbitrarily spaced table.

For fixed spaced independent variable tables, you need only give the initial value and the spacing increment for each independent variable. To use this format, ***all*** of the independent variables for a given table must have fixed spacing.

The format for an nth-order independent variable table is:

```
TABLE, table_name, N1, N2, ... , Nn, FIXED INCREMENT
n  initial values for the independent variable table data
n  increment values for the independent variable table data
Dependent variable table data (N1 x N2 x N3 x ... Nn values)
```

Note: The information **TABLE, table_name, N1, N2, ... , Nn, FIXED INCREMENT** must be placed on a single line. For each of the other lines, the information may be placed on as many lines as required.

The total number, T_f of data values required on lines 2 to 4 is given by:

$$T_f = 2n + \prod_{i=1}^{n} N_i$$

The dependent variable table values are ordered in the same way as arbitrarily spaced data.

Refer to Ap. A: Summary of Analysis Commands for more information on how to use the fixed spaced data feature.

Setting Tables To Zero

You can set the dependent variable of a table to zero. This allows you to simplify a model by removing the effects of a table or providing a default value of zero when tabular data is not available for a model still in development. To set a table to zero, the command format is:

```
TABLE, table_name = ZERO
```

This command causes the table: **table_name** to be loaded with independent and dependent variable information that will result in zero as an output for all independent variable values. To remove the **ZERO** option, you must load new table data.

The OMIT TABLE PRINTBACK Command

To suppress the printback of table data, use the following command:

```
OMIT TABLE PRINTBACK
```

By default, this command is always set. This command is often used on "production runs" or models with large amounts of constant tabular data.

The PRINTBACK TABLES Command

If you wish to verify table data, you issue the following command to restore the table printback feature:

```
PRINTBACK TABLES
```

"Analysis Only" Mode

The *Analysis* license feature enables access to Easy5 analyses. It can be used to provide lower cost access to Easy5 customers that do not need to create component libraries or build their own models, but rather, that just want to apply Easy5's analysis set to one or more Easy5 models created by another user.

The Analysis license feature allows you to perform analyses on a given model. An "*Analysis Only*" mode of operation is defined by the situation *whenever you have checked out an Analysis license feature, but have no*

Model Building (or Library Developer) feature checked out. Please see Ap. A: License Management in the Easy5 User Guide for information on Easy5 license features.

Operations Allowed

When in "Analysis Only" mode, you are able to do the following:

- Open any existing model
- Change parameter and IC values
- Run any Easy5 analysis (assuming other library-specific license features are also checked out, if needed)
- Create and manage temporary setting and auxiliary input files.
- Create and manage analysis settings files
- Save new versions of a model (without affecting model topology)
- Save/restore initial conditions
- Use the Easy5 Plotter to display your results

Disabled Functionality

While in "Analysis Only" mode, you cannot:

- Create a new model
- Save an existing model under a new name using the "Save As..." option
- Perform any edit operations to a model, such as:
    - Copy individual or groups of components
    - Copy groups of components from other models
    - Add or delete components
    - Change names of any model quantities
    - Change any configuration parameter for any component
    - Change names of any components

- Access most Library menu items
- Access the Build Menu
- Access the Edit Menu
- Change table dimensions
- Edit User-Code (FORTRAN, C, or Extension) component
- View the executable source files
- Update a model with respect to its libraries

Creating a "Locked Configuration" Model for Distribution Purposes

By default, any Easy5 model can be opened in "Analysis Only" mode. However, if a model/library update (a model/library configuration resynchronization) is required, you will be unable to open that model, unless you (or the author) performs a Model Update -- requiring checkout of a Model Build license feature. In addition, models provided for "Analysis Only" mode distribution will only function for the platform created by the author. This is because the model is an executable file, which can only be run in a given platform/compiler environment.

To prevent model/update issues from occuring, you may want to more strictly enforce the configuration of a given model, in particular if it is to be used in an Analysis Only mode of operation.

Thus, if you want to distribute a "locked down" model configuration, you can do that using the following procedure:

1. Create a separate subdirectory for the model.
2. Include copies of any library dictionaries and object libraries in the local directory that you want "locked-in" for this model, overriding what normally gets used by Easy5. You can even include copies of Easy5's own (GP, IS, etc.) libraries, if you want to lock these down also. Recall that any model/library configuration difference would prevent the end-user from using a particular model in "Analysis Only" mode.
3. If any libraries have Info Pages (in the form of either .html, or .pdf files), these should be included in a subdirectory named *xx*/info (where xx=2-character library tag). Similarly, for alternate icons a directory named *xx*/icons would need to be included.
4. Build the model executable (for all platforms to be executed on).
5. Delete any generated source files in the directory, if desired.
6. Delete any unwanted listing or other output files (.ezlgl, .ezmgl, .ezapl, .log) from the directory.

7. Confirm that all required files are included in the model directory including:

<modelname>.<v>.ezmf	Easy5 modelfile
<modelname>.exe	model executable (for a given platform)
<xx>.ezdc, <xx>.ezdf	local dictionary files for each library <xx>
<xx>.a, or <xx>.lib	local object code libraries for each library <xx>

   In addition, you probably will want to include files for running analyses, operating point files, and temporary settings files.

8. It is always a good idea to include a <*modelname*>.info.html (.pdf, or .txt) file explaining what the model does and how it should be run. A provision for Model Info files named <*submodelname*>.info.html (.pdf, or .txt) is also provided, allowing Model Info files for each individual Submodel to be maintained and displayed (from the given submodel only). Please see "Modeling Fundamentals" for more details.

9. You can use WinZip (or a similar archiving tool) to compress the entire directory for distribution purposes. This makes it easy to transfer the whole set of files in a single archive. Password encryption can also be used at this step for security purposes.

Using a Locked Configuration Easy5 Model

Then, to use a locked configuration model, the ("Analysis Only" mode) Easy5 user should:

1. If appropriate, decompress the archived folder with all files as needed (using WinZip, for example).
2. Invoke Easy5 (and if needed, checkout the "Analysis" license feature).
3. Open the model in the given folder (one model per folder).
4. With a Analysis license feature checked out, the user can invoke Easy5 and run analyses on this model with all (desired) libraries and the model essentially pre-configured. You will also need to checkout any application library-specific runtime license features used in the model. The user will be unable to view generated source code or modify the model topology. Because a model executable has already been built, this installation of Easy5 will not require a compiler.

Background Processes

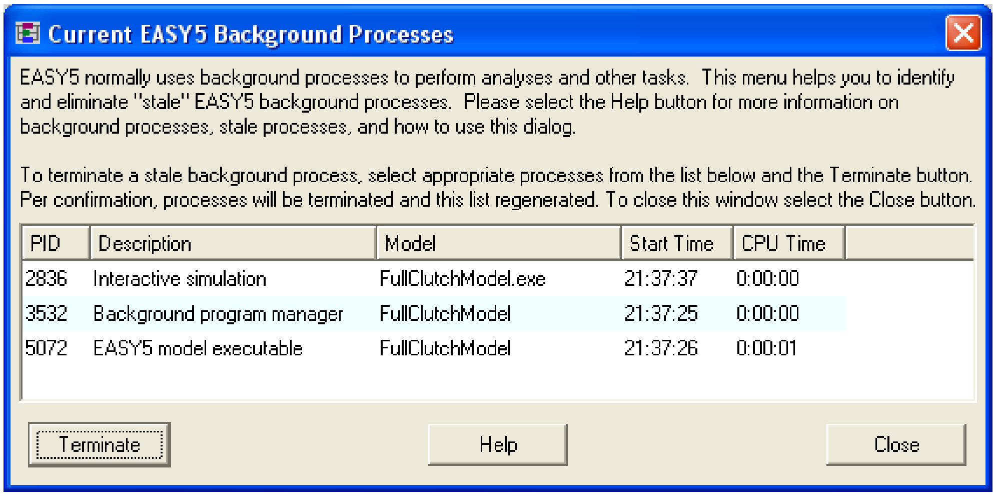
Easy5 normally uses so-called "*background*" processes to perform analyses and Build tasks, such as creating an executable, and updating a component library.

In some cases, several background processes may run simultaneously for a single analysis. This is because there are actually several background programs used to facilitate the many background operations being performed for the action in progress.

While Easy5 normally takes care of the management of these processes, it is possible that due to a system crash, or other unanticipated problem, one or more of these processes may continue to run in the background, often without actually performing any useful function.

Such processes, referred to as "*stale*" processes, continue to run even though you directed them to terminate using a Stop Analysis command, or other means. If you do not terminate stale background processes, they can sometimes run indefinitely and can potentially interfere with legitimate Easy5 background operations, such as analyses, as well as system hardware resources (for example. memory, CPU).

Since Easy5 cannot safely differentiate between stale and legitimate background processes, you can select **View > Easy5 Background Processes** from the main menu to identify and eliminate stale processes as shown below in Figure 10.



Current EASY5 Background Processes

EASY5 normally uses background processes to perform analyses and other tasks. This menu helps you to identify and eliminate "stale" EASY5 background processes. Please select the Help button for more information on background processes, stale processes, and how to use this dialog.

To terminate a stale background process, select appropriate processes from the list below and the Terminate button. Per confirmation, processes will be terminated and this list regenerated. To close this window select the Close button.

PID	Description	Model	Start Time	CPU Time
2836	Interactive simulation	FullClutchModel.exe	21:37:37	0:00:00
3532	Background program manager	FullClutchModel	21:37:25	0:00:00
5072	EASY5 model executable	FullClutchModel	21:37:26	0:00:01

Figure 10 Current Easy5 Background Processes Window

This window displays several columns of data designed to help you more readily identify stale processes. The first column is the Process ID, the second the (translated) process name, then the name of the associated Easy5 model, followed by the start time (or date), and the total CPU time expended.

Please note that some of the processes listed in this dialog window may actually **not** be stale; they could be legitimate background processes that are still running. While only processes identified with your <*username*> are listed in the process list, it is important that you verify that no other user is inadvertently running Easy5 on your computer with your username while you are checking these processes.

Also, please note that it is possible to launch Easy5 analysis jobs and then exit the graphical user-interface. These jobs will continue to run as background processes until they are manually terminated, or reach their conclusion normally. Only background processes launched from the Easy5 graphical user interface are shown in the Background Processes Window list; such background jobs are all managed using a Background program manager as described in the table below.

Table 1-2 Description of Background Processes in Dialog List

Process Name	Description
Matrix Algebra Tool	Program representing the Easy5 Matrix Algebra Tool
Interactive simulation	Program displaying active Interactive Simulation widgets running in an Easy5 simulation
Background program manager	Intermediate program ("remote_exec") used to communicate between background jobs and the Easy5 graphical user interface.
Model generator	Program used to construct an executable model, and update library components.
Editor	Easy5 text file editor used for text file display or editing
Docmod utility	Program used to document your model in either text or HTML modes
Icon Editor	Program used to edit and/or display Easy5 icon files.
Plotter	Program used to read, display, and customize Easy5 plot data files
compile command	Process used to compile model file
link command	Process used to link your model object file with Easy5
executable shell	Intermediate shell process used for launching background operations
model executable	Your Easy5 model executable (during an analysis)

Thus, all Easy5 "batch" mode jobs (see Ap. D: Batch Mode Commands) are excluded from this list.

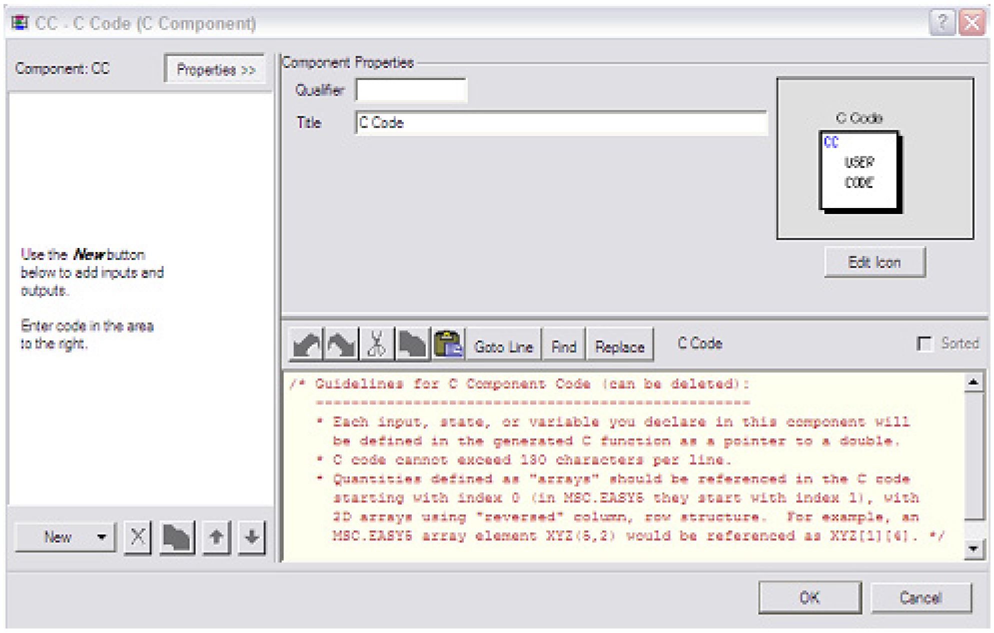
If you know that one or more processes in the list are stale, you can terminate them. However, please note that any process terminated in this way also lose any output data that was written.

To use the dialog to terminate one or more processes, select one or more (stale) processes from the dialog list, and choose Terminate. After a confirmation dialog, Easy5 terminates the selected process, and reopens the Background Processes window. Please note that the amount of time required to remove a process from the process list is system-dependent, so you may wish to close out the window and refresh it manually in some instances.

C Component

See also: "Compiling External Code"
"Fortran Component"
"Linking External Code"
User Guide, Chapter 5 - Code Components

C code is added to a model using the C code component. This component is accessed from the Add Window by selecting the C button at the bottom of the window, then dropping the component onto the schematic. It is named CC<xx>, where, <xx> is the component qualifier as shown in the C Code Component Editor in Figure 11.



Use the ***New*** button below to add inputs and outputs.

Enter code in the area to the right.

```
/* Guidelines for C Component Code (can be deleted):
   ================================================
   * Each input, state, or variable you declare in this component will
     be defined in the generated C function as a pointer to a double.
   * C code cannot exceed 130 characters per line.
   * Quantities defined as "arrays" should be referenced in the C code
     starting with index 0 (in MSC.EASY5 they start with index 1), with
     2D arrays using "reversed" column, row structure.  For example, an
     MSC.EASY5 array element XYZ(5,2) would be referenced as XYZ[1][4]. */
```

Figure 11 C Code Component Data Table

Adding C Code

C code is entered into to the C code editor. The coding requirements are easy to understand because Easy5 takes care of the code structure, compilation and linking. The following rules apply when adding C code to an Easy5 C Code component:

1. The C code entered is only the main body of the C code. Easy5 takes the C code and writes it to an external C code file as a separate function, with all the appropriate headers and structure, and automatically compiles this and links the code with the Easy5 executable model. All C components from a given model are written to a single file named <modelname>_c.c.
2. All Easy5 scalar variables (names displayed in the Component Data Table) must be defined in the C code as pointers. To do so, the names must be preceded with an “ * ”, such as: *current_ma. This only applies to all Easy5 scalar variables used in the C code.
3. All Easy5 array variables (names displayed in the Component Data Table that are arrays, also called vectors) are already referenced as double pointers. The names are entered as is, without referencing as a pointer.

   The arrays are listed in the data table using the Fortran convention, which indexes the array from 1 to “n”. However, when using the array in the C code, the C code convention is used, which indexes the array from 0 to “n”-1, where n is the array size.

   For two-dimensional arrays, row and colums are reversed between Easy5 and C code convention. This means that an Easy5 array XA(n,m) is referenced as XA[m-1][n-1] in the C code body.

4. Variables not added to the Component Data Table are "local variables", which take on the proper C code convention.
5. Global variables (reserved words) and functions from Easy5 can be accessed from C Components, and used in an analogous way as in Fortran Components.
6. #-defines and other non-executable code or C declarations can be added to a C component by using the DECLARATION keyword, and adding the appropriate C code. Such code will be sorted to the bottom of the non-executable portion of the resulting C source code file.

In general, such variables are defined by all UPPER case variables such as those shown in the table below. These variables and functions are defined in the C code file, *<model>_c.c*.

Table 1-3 Global Variables and Functions

Variable/Function	Definition
ITINC	ITINC flag
IEZSWS	IEZSWS flag
INCALL	initial call flag
ICCALC	Calc XIC flag
TIME	the value of TIME
IDELAY	IDelay flag
EZTBL1	1 D table-lookup
EZTBL2	2 D table-lookup
EZTBL3	3 D table-lookup
EZ_IS_CONNECTED(s)	to verify if an input name "s" is connected
INST	analysis type flag
ISTOP	termination flag
EZ5TOAPL(s)	to write string "s" to the EZAPL file

The following is a simple example of a C code component which contains both scalars and arrays, states, and local variables. The C code component models the dynamics of three mass spring dampers.

Example C Code Component

The C code component data inputs are:

- `Force_external  {scalar input, used to connect the external force}`
- `mass[3]  {array sized 3, to define the mass of three bodies}`
- `damping_coeff  {scalar parameter constant, the same damping coefficient is used on the 3 masses; it is initialized to 0.7 if needed using a test on INCALL}`

- `k_spring_constant[3]  {array sized 3, to define the spring constant for each spring}`
- `k_arr[4][3]  {a constant array sized in Easy5 as (3,4) -- used for example purposes only}`

The two states are:

- `position[3] {array sized 3, defines the position of each of the 3 masses}`
- `rate[3]  {array sized 3, defines the velocity of each of the 3 masses}`

The output variables are:

- `acceleration[3] {array sized 3, defines the acceleration of each mass}`
- `xarr[4][3]  {an output array defined in Easy5 as (3,4) for example purposes)`

In addition to the Easy5 variables, local variables used in the code are:

- `Force_spring  {scalar, defines the spring force}`
- `Force_damper  {scalar, defines the damper force}`
- `Force_on_mass  {scalar, defines the force on the masses}`

The C code body is as follows:

line# Code:

```
 1 double Force_on_mass, Force_spring, Force_damper;
 2 int i;
 3 if (INCALL == 2 && *damping_coeff == (double)0.99999)
 4        *damping_coeff = 0.7;
 4 for (i=0; i<3; i++) {
 5   Force_spring= k_spring_constant[i] * position[i];
 6   Force_damper= *damping_coeff * rate[i];
 7   Force_on_mass= *Force_external-Force_spring-Force_damper;
 8   acceleration[i]= Force_on_mass/mass[i];
 9   DERIVATIVE OF, rate[i]= acceleration[i];
10   DERIVATIVE OF, position[i]= rate[i];
11   for (j=0; j<4; j++)
12     xarr[j][i]= k_arr[j][i]*acceleration[i];
13  }
```

Examine the code. The first and second lines define the local variables, variables not defined in the Easy5 data table. On lines 3 and 11, the "for" loop indexes use the C standard, starting at 0, not 1.

Please note that the Easy5 names that display in the component data table use the FORTRAN convention which starts at index 1, whereas in the C code, the C convention is used, which starts at 0. On line 5, because `k_spring_constant` is an Easy5 array name it does not need to be referenced as a pointer.

On line 6, `damping_coeff` is an Easy5 scalar data type and therefore is referenced as a pointer by preceding the name with an *. The final lines use the Easy5 command "DERIVATIVE OF," to integrate the rates and calculate the states.

Adding C Declarations

Declarations can be easily added to your C code by using the Easy5 DECLARATIONS command, for example, to include C system include file stdio.h:

```
    DECLARATIONS, #include <stdio.h>
```

Such declarations will get placed near the top of the C source file that is generated by Easy5.

C Code Files and Structure

The process of creating, compiling and linking C code with Easy5 is shown in Figure 12 below.

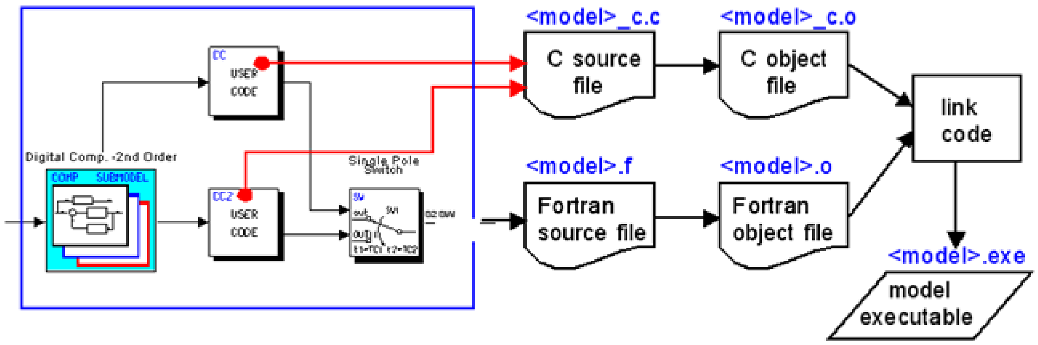


Figure 12 C Code Files, Compilation and Linking

Easy5 is a code generator; the GUI-based model is converted to Fortran source code, and the source file is compiled and linked with Easy5 object files. The underlying source code is always a Fortran source file, named *<model>.f*.

As a result, when you add C code to your model, the C code is not embedded in the Fortran source file as inline code. Instead, the C code from all C code components is written to a C source code file named, *<model>_c.c*. This file contains a separate function for each C code component, and each function is referenced as, *EZ__CC*<xx>, where, <xx> is the specific component designator.

Preceding these functions are various declarations allowing you access to most Easy5 global variables, and several functions, such as, table look-ups. Always check these declarations when using them in your C code.

An example of a C code source file containing two C functions is shown below:

```
void EZ___CC  ( double *acceleration, double *position, and so on )
   {
     <c code from component CC>
   }
void EZ__CC02  ( <list of variables> )
   {
     <c code from component CC02>
   }
```

When the executable model is created, the C code source file *<model>_c.c* is automatically compiled and linked with the Easy5 Fortran model code, as shown in Figure 12. The C code source file is written to the disk, and is accessed by selecting the menu, Build > Display C Component Source File. If an error occurs during the Fortran or C code compile, the error message is written to a file named *<model>.err*, and Easy5 automatically opens and displays the error file.

In the model source file *<model>.f*, each C code component is replaced with a call to the associated C code function, *call ez__CC<xx>* , where *<xx>* is the component qualifier. An example of such a call in a Fortran source file is shown below:

```
C             COMPONENT  CC02
Call ez___CC02 (acceleration,position,ez_cder_position,rate,
+ ez_cder_rate,Force_external,mass,damping_coeff,k_spring_constant)
```

Code Components

See also: "Fortran Component"
   "C Component"
   User Guide, Chapter 5 - Code Components

Both C and Fortran code is added to a model using a User Code component. Multiple code components may be added to a model, and the language may be mixed, that is, both C and Fortran code may be added to the same model.

There are two types of code components, a Fortran code component, and a C code component. The code component is added to a model by selecting the “User Code” selection from the bottom of the Add Window as shown in Figure 13. The Fortran code component is named FO<*xx*>, and the C code component is named CC<*xx*>, where, <*xx*> is the component designator.

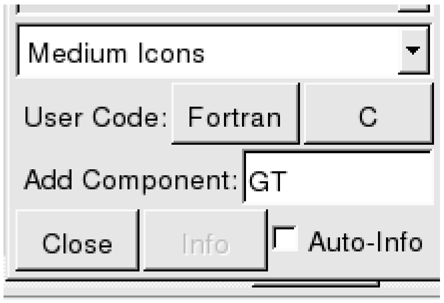


Figure 13 Code Component

Complete information about adding both C code and Fortran code is also given in the User Guide, Chapter 5 - Code Components.

Command Line Options

There are many command-line options available to setup the Easy5 environment and run special scripts. The options list is obtained by entering the following command at any Easy5 Command Shell window prompt:

```
easy5x -help
```

The Easy5 Command Line Options table shows a **partial** list of different Easy5 command line options. Your options list may vary depending on the Easy5 version you are running. These command options are entered from an *Easy5 Command Shell, not* from within Easy5. To open an Easy5 Command Shell window select the "**Open Command Shell...**" File menu item.

Note: **Windows:** Easy5 options can also be displayed by selecting **Start > Program Files > Easy5 2021.1 > Help > Command-Line Options**.

The command is entered as:

```
easy5x -<option>
```

Table 1-4 Easy5 Command Line Options

Option (case sensitive)	Description
-al *mm*	Convert your binary component library file '*mm*'.ezdc to ASCII format. This is used to transfer component libraries across platforms.
-AutoUpdate <*model*>	Convert or update an existing (v6) model
-bl *mm*.ezda	Convert ASCII component library file *mm*.ezda to Easy5 binary format.
-B [batch command]	Invoke Easy5 batch program directly.
-B -help	Displays batch help; lists batch commands and options.
-cc [-d] <*CcodeFiles*>	The "cc" option is for C compilation. Compiles the C code source files using the default compiler options used by Easy5. One or more file names can be entered, and wildcards can be used. See Compiling External Code for more information. Use the "-d" option to specify debug mode.

Table 1-4 Easy5 Command Line Options (continued)

Option (case sensitive)	Description
-clean [*modnam*] [-f]	Remove unnecessary [*modnam*] files from current directory.
-def	Displays internal variable settings.
-demo [options]	Copies Easy5 demo models into the current directory. Options: *nn* copies only demos for library nn *all* copies all available demos into a demo tree
-docmod <*mfile*> [*cc*] [-html]	Displays Easy5 component data descriptions and values for every component in the specified model. Optionally generates data only for component *cc*. The -html option generates an HTML version of the model description. Requires a runtime model building license.
-dp *mm* [*cc*] [-al] [-a] [-l] [-html]	Displays library component data descriptions and values for all components in component library *mm*.ezdc.0 Options: *cc* all data for library component *cc* -al all data for every component in library -a abbreviated data (no code) -l only component names, descriptions -html all data for all components <HTML> written to *mm* /info directory (requires a runtime Library Developer license) -hinf all data for all component in HTML for use as Info Pages - dp -help displays this list
-fc [-d] <Fortran code file(s)>	The "fc" option is for Fortran compile. Compiles the Fortran source code files using the optimum compiler options used by Easy5. One or more file names can be entered, and wildcard can be used. See Compiling External Code for more information.
-home	Displays directory where Easy5 is installed.
-hostid	Displays your workstation hostid (for License Management).

Main Index

Table 1-4 Easy5 Command Line Options (continued)

Option (case sensitive)	Description
-hotline	Displays phone numbers to call for Easy5 technical support.
-help_batch	Get help on using Easy5 (batch) mode.
-help_remote	Get help on using remote execution.
-iconedit	Edit icon. Opens, and saves data to file: icon_data.
-iconedit filename	Edit icon. Opens, and saves data to file: ‘filename’.
-iconedit ll cc	Edit icon. Opens icon for component ‘cc’ in library ’ll’. Saves icon data to file: icon_data.
-infoedit	Edit an Info Page. Data saved to file: info_data.
-infoedit filename	Edit info data. Opens, and saves data to file: ‘filename’.
-infoedit ll cc	Edit info data. Opens info for component ‘cc’ in library ‘ll’. Saves info data to file: info_data.
-libnotes	Displays where Application Library Release Notes are stored.
-LibConvert xx	Convert an existing v6 library xx to v7 format
-license	Displays info about your Easy5 license (expiration date, etc).
-lmstat	Displays out license manager status information.
-mat	Run the Easy5 Matrix Algebra Tool (MAT).
-matD [debugger]	Run the Matrix Algebra Tool (MAT) using the specified debugger.
-mat_nogui	Run the Easy5 Matrix Algebra Tool (MAT) w/o GUI.
-mat_demo	Copies demonstration MAT files to your current directory.
-notes	Displays the current Easy5 Release Notes.
-p [rpdfile | rplfile]	Runs the Easy5 plotter [auto-load ‘.ezrpd’ or ‘.ezrpl’ file].
-plot_demo	Copies demonstration plot files to your current directory.
-run_pdemo	Runs the Easy5 Plotter Demo.
-teach [-norun]	Copies all Easy5 tutorial models into your current directory. Also invokes Easy5 unless ‘-norun’ option is specified. Ref: Easy5 User’s Guide, Chapters 5,6 & 7.

Table 1-4 Easy5 Command Line Options (continued)

Option (case sensitive)	Description
-tutorial [-norun]	Copies Easy5 'quick' tutorial into your current directory. Also invokes Easy5, unless the '-norun' option is specified. Ref: Easy5 User's Guide, Chapter 3.
-v [*ll*]	Displays Easy5 version information [for library *ll*]
-vars	Displays the environment variables you can set to define various Easy5 options. See "External (Environment) Variables" for more information.
-varset	Lists all defined Easy5 environment vars. and their values.

Option Examples

The option command is proceeded by a "-" to modify the easy5x command. For example, to display the Release Notes, enter "-notes" at a command prompt as follows:

```
easy5x -notes
```

To generate an ASCII document of your model named *test_servo*, version 5, and have the components listed in alphabetical order, enter the *docmod* option with the -a modifier as follows:

```
easy5x -docmod test_servo.5.ezmf -a > test_servo.txt
```

The model document will be saved in a file named *test_servo.txt*

Compiling External Code

See also: "Fortran Component"
"C Component"
"Linking External Code"

External code that is called or referenced in your model must be compiled outside of Easy5. It is very important that you compile the code with compiler options that are compatible with Easy5.

The simplest method of assuring this is to use Easy5's compile commands:

For Fortran code: `easy5x -fc [-d]<`*`source file(s)`*`>`

For C code: `easy5x -cc [-d]<`*`source file(s)`*`>`

where, <*source file(s)*> are the names of the Fortran or C source files, and the "-d" option is used to request "debug" compiler options, allowing you to use a symbolic debugger with those files. The compile commands will compile the source files using the default compiler options required by Easy5, and are used from any Easy5 Command Shell window.

Default Compiler Options

The table shown below indicates the standard Fortran compilation options that Easy5 normally uses when compiling your model. These basic options are recommended when compiling your own user-defined Fortran routines that are to be linked to Easy5. However, it is easier to use Easy5 compile commands (see above), which automatically use the appropriate platform and compiler appropriate compile options for compatibility with Easy5.

Table 1-5 Easy5 Default Fortran Compilation

Platform	Fortran Compilation Options
Sun Solaris	f77 -c -O2 -Nl329 -Nn50000 -Ns6000 -Nx1500 -Nq4000
HP/UX	f77 -c -O +ppu
IBM/AIX	f77 -c -O -qextname
Windows Compaq Visual Fortran	df /c /MD /fpe:0 /Oxp

Caution:	The values in *Easy5 Default Fortran Compilation* table may not always be up-to-date with software updates. Therefore, always confirm these options by using the "`easy5x -def`" command as described below, or use the default Easy5's compile commands.

Obtaining Current Compiler Options

To obtain the current Easy5 default compiler options, enter the command:

```
easy5x -def
```

This command lists all internal Easy5 variables. For example, the Fortran compiler options are defined by the environment variable named *ezfflags*. To get a list of only the Fortran compilation options enter:

```
easy5x -def | grep ezfflags
```

To list the Fortran compilation **optimization** options enter:

```
easy5x -def | grep ezf77opt
```

And, to list the Fortran compilation **debug** options enter:

```
easy5x -def | grep ezdebugf77opts
```

The optimization option should **not** be used when compiling with the debugger. To compile with the debugger, set the *ezdebug* environment variable to *on, or use the "-d" option*. This disables all optimization.

The C default compiler options are defined by the variable named *ezcflags*, the C optimization options are defined by *ezcopt*, and the C debug options are defined by *ezdebugcopts*.

Setting Debug Compiler Options

To compile your external code with the debug compiler options turned on and the compiler optimization turned off, set the ezdebug variable as shown below. This variable must be set before compiling using the Easy5 Compile Command.

```
setenv ezdebug on        {C shell}
export ezdebug=on        {Korn/Bourne Shell}
set ezdebug=on           {Windows}
```

An alternate way is to add the "-d" option to specify debug mode with Easy5's "fc" or "-cc" compile commands.

User Specified Compiler Options

It is possible to customize the command line that Easy5 uses to compile the Fortran source file. This can be very useful in cases where a floating point accelerator is installed on a particular node and much of the execution time is spent in User Code Fortran or Library components.

To change the compilation options, set the environment variable `ftnopt` to the compile options you need, as shown below. This will override the Easy5 default compile options. This variable must be set before compiling using the Easy5 compile command:

```
setenv ftnopt <compile options>    {C shell}
export ftnopt=<compile options>    {Korn/Bourne Shell}
set ftnopt=<compile options>       {Windows}
```

Setting `ftnopt` will override the Easy5 default compile options.

Examples of Compiling External Code

1. To compile all Fortran or C source files in the current directory, use the wildcard “*” with the command:

```
easy5x -fc -d *.f         Fortran files
easy5x -cc -d *.c         C files
```

2. To compile C code source files named trim.c, filter.c and kad.c, enter the command:

```
easy5x -cc trim.c filter.c kad.c
```

3. To compile Fortran or C source code with the debug option on, the commands are:

```
easy5x -fc -d *.f         Fortran files
easy5x -cc -d *.c         C files
```

4. To override the Easy5 default Fortran compile options with your options, and then compile “test.f” using the easy5x command, the Windows command is:

```
set ftnopt=/c /MD          Windows method for setting ftnopt
easy5x -fc test.f
```

Compiling and Linking Mixed Code

This section contains a list of recommended compile and link commands that makes it easier to compile and call external C code from a Fortran routine, and to call Fortran routines from a C function. Sample source files are included at the end of this section to demonstrate how to call C from Fortran and Fortran from C.

C Code Considerations

There are special considerations that you must take when compiling C Code and linking it to a Fortran call. See "Compiling External Code" for information on Easy5 command-line options used to compile C code.

These are platform specific as follows:

Windows Notes

- Note that *all* function names should be in UPPERCASE.
- C functions called from Fortran should be declared __stdcall (for DVF6), and __cdecl (for others).
- Extra "string length" argument added for each character argument is put *immediately after the character argument*.

Linux Notes

- Add an underscore character to end of C functions called from Fortran.
- Extra "string length" argument added for each character argument is put *at the end of the argument list*.

HPUX Notes

- Use the Fortran compiler option `+ppu`

AIX Notes

- Use the Fortran compiler option `-qextname`

C Code Compile/Link Commands

The following tables show a list of compile and link commands for both C and Fortran code that should be used when calling C code from Fortran or vice versa. Note that these are general recommendations that depend on the compiler type and version and may not apply directly to your system.

See "Compiling External Code" for information on Easy5 command-line options used to compile C code.

Use these recommendations more as a general guideline. For example, you may wish to add and experiment with different compiler optimization options which are not included in this list.

Table 1-6 Windows Compaq Fortran Compiler

	Compile/Link Commands
C compile	cl -c /Zi -DWIN32 mixed_lang_c.c
C preprocessor	FPP -DWIN32 mixed_lang_f.F mixed_lang_f_cpp.f
Fortran compile	DF -c /Zi mixed_lang_f_cpp.f
Fortran link	DF /Zi mixed_lang_f_main.f mixed_lang_c.obj mixed_lang_f_cpp.obj

Table 1-7 Sun Solaris Compiler

	Compile/Link Commands
C compile	cc -c -g mixed_lang_c.c
C preprocessor	f77 -F mixed_lang_f.F
Fortran compile	f77 -c -g mixed_lang_f.f
Fortran link	f77 -g mixed_lang_f_main.f mixed_lang_c.o mixed_lang_f.o

Table 1-8 HP HPUX Compiler

	Compile/Link Commands
C compile	c89 -Aa -c -g mixed_lang_c.c
C preprocessor	f77 -F mixed_lang_f.F
Fortran compile	f77 -c -g +ppu mixed_lang_f.f
Fortran link	f77 -g +ppu mixed_lang_f_main.f mixed_lang_c.o mixed_lang_f.o

Table 1-9 SGI IRIX Compiler

	Compile/Link Commands
C compile and preprocessor	cc -c -g mixed_lang_c.c
Fortran compile	f77 -cpp -c -g mixed_lang_f.F
Fortran link	f77 -g mixed_lang_f_main.f mixed_lang_c.o mixed_lang_f.o

Table 1-10 IBM AIX Compiler

	Compile/Link Commands
C compile	xlc -c -g mixed_lang_c.
C preprocessor	/lib/cpp -P mixed_lang_f.F > mixed_lang_f.f
Fortran compile	xlf -qextname -c -g mixed_lang_f.f
Fortran link	xlf -qextname -g mixed_lang_f_main.f mixed_lang_c.o mixed_lang_f.o

Example of Calling Fortran from C

```
#
include <stdio.h>

#ifdef WIN32
void __stdcall CFUNCTION ( int *ival,
        char *str,
        int str_len,
        double *dval,
        double array[])
#else
void cfunction_ ( int *ival,
        char *str,
        double *dval,
        double array[],
        int str_len)
#endif
{
        int i;
        int local_ival;
        double local_dval;
        double local_array[3] = { 1.23, 2.34, 3.45 };
        char buf[1024];

        printf("Hello from cfunction, Values received from
Fortran:\n");

/* print out arguments received from Fortran main program */

        printf("ival = %d\n", *ival);
        printf("dval = %lf\n", *dval);
        for(i=0; i<3; i++)
        {
                printf("array[%d] = %lf\n", i, array[i]);
        }

        strncpy(buf, str, str_len);
        buf[str_len] = (char)0;
        printf("str = %s\n", buf);

/* call Fortran subroutine from C */

        printf("cfunction calling Fortran:\n");
        local_ival = *ival;
        local_dval = *dval;
#ifdef WIN32
        i = FFUNC(&local_ival, &local_dval, buf, strlen(buf),
local_array);
#else
        i = ffunc_(&local_ival, &local_dval, buf, local_array);
#endif

/* return new values to calling (Fortran) program */
        *ival = *ival * 100;
        *dval = *dval * 100.;
        strcpy(str, "new strng");
        for(i=0; i<3; i++)
        {
                array[i] = array[i] * 100;
        }
```

Fortran function called from C code

```
#ifdef WIN32
      integer function FFUNC ( ival, dval, string, array )
#else
      integer function ffunc ( ival, dval, string, array )
#endif
C
      integer ival
      real*8 dval
      character*(*) string
      real*8 array(3)

      write(6,5)
 5    format(1x,"hello from Fortran function: .")

          write(6,10)ival, dval, string
 10   format(1x,"(ffunction) Values received from c:",/,"ival = ",
     x  i3,", dval = ", f10.5, ", string = ", a10)
       do 100, i = 1, 3
         write(6,20)i, array(i)
 20      format(1x,"array(",i2,") = ", f10.5)
 100  continue

      ffunc = 10
      return
      end
```

Example of Calling C From Fortran

```
      integer ival
      integer i
      real*8 dval
      real*8 array(3)
      character*10 string

      ival = 3
      dval = 1.25
      array(1) = 1.1
      array(2) = 2.2
      array(3) = 3.3
      string = 'a string'

      write(6,5)
 5    format(1x,"hello from main Fortran program: calling c
function.")

      i = cfunction(ival, string, dval, array)

      write(6,10)ival, dval, string
10   format(1x,"(main Fortran) Values returned from c:",/, "ival = ",
     x  i3,", dval = ", f10.5, ", string = ", a)
       do 100, i = 1, 3
         write(6,20)i, array(i)
 20      format(1x,"array(",i2,") = ", f10.5)
 100  continue
      stop
      end
```

Components

See also: "Adding Components"
"Code Components"
"Component Data Table"
"Parameters - Defining Input Values"
"States"

Users of Easy5 represent their dynamic system model with interconnected modeling blocks known as components. Components, represented in the model as full color graphical icons, are actually mathematical models of physical system. Four types of components are used in Easy5:

- *Library* components
- User-defined *Library* components
- *User-Code* components used to enter Fortran or C code
- *Extension* components

These four types of components may be used together in a model. This modularized component approach to modeling is particularly well suited to very large system models characterized by hundreds of differential. difference and algebraic equations. Components may also be dimensioned, that is, the entire component can be made into a "vectorized" component, whereby the inputs and outputs are vectors and the underlying code is automatically vectorized. This is reviewed in "Dimensioned Components".
A review of each type of component is given in this section. But first, the fundamental component basics is presented.

Component Basics

Naming Convention

The Easy5 naming convention uses explicit names of up to 60 characters, separating the three parts of each name by underscores (_), no embedded blanks, and (for ported quantities) using port names of up to 12 characters. Results of this naming convention make it easy to identify any model name in various model name lists, the output listing.

This naming convention requires that no underscores be used except as these special name delimiters. This requirement is enforced during library component definition. In addition, port names are limited to only alphanumeric characters - no underscores or punctuation. The resulting *default names* often convey enough mnemonic meaning lessening the need for custom, user-defined names. Nevertheless, any default names can be overridden by simply typing over the default name with an up to 60 character *user-defined name*.

Example:

A model quantity representing the temperature at exit port "Exit" of the pipe (PI) component named L7 has a default Easy5 variable name of "**Temp_Exit_PIL7**". A user could override this default name with a user-defined name of "**TempExitPipeHFA7**", and still access the default name later, at any time.

Component Inputs

There are two categories of input quantities for components. These are: regular inputs, the values for which come from other components or are supplied by the user; and table inputs, which comprise a set of user-defined table lookup data.

All regular inputs left unconnected are called *parameters*. Parameters are quantities that are constants of your system. You must give parameters data values before the executable model is analyzed.

Component Outputs

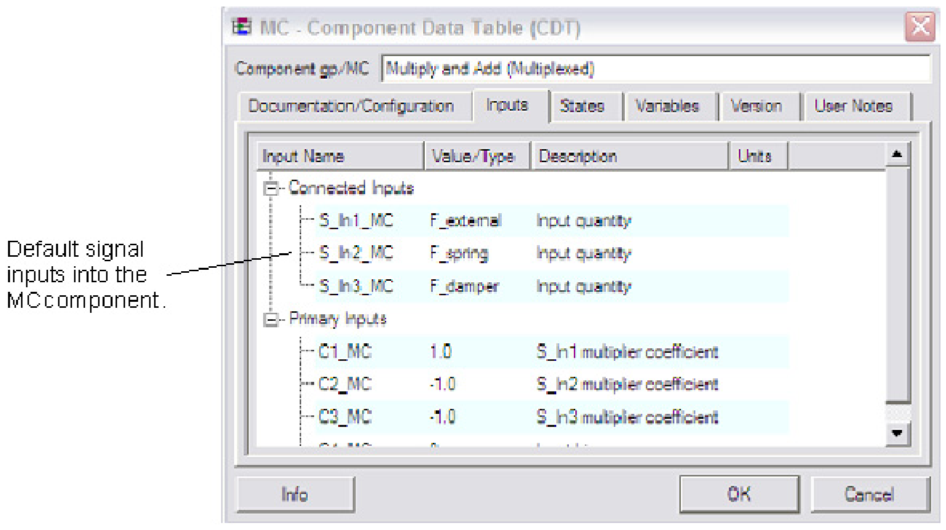
Two types of output quantities exist for components: *states* and *variables*. States are a function of first order differential equations (continuous states), difference equations (sample and delay states), or switching/memory equations. Variables are algebraic functions of states and other variables.

Component Input/Output Naming Convention

Easy5 uses a *default* naming convention for all inputs and outputs in a model. This convention allows you to recognize the source of any standard component input or output, and guarantees that every quantity name in your model is unique.

Any Easy5 defined *default* names can changed to a *user-defined* name. See "User-Defined Names" for information on how to redefine Easy5 names. Fortran and C components can use any legitimate Fortran or C name and are not subject to this unique variable naming convention. Therefore, Fortran and C component inputs and output names are by default, *user-defined*.

The default naming convention for all Library component inputs and outputs consists of up to 60 characters, separated into three parts by underscores, as shown in the example Component Data Table in Figure 14.


Default signal inputs into the MC component.
MC - Component Data Table (CDT)
Documentation/Configuration
Inputs
States
Variables
Version
User Notes
Info
OK
Cancel

Component gp/MC Multiply and Add (Multiplexed)

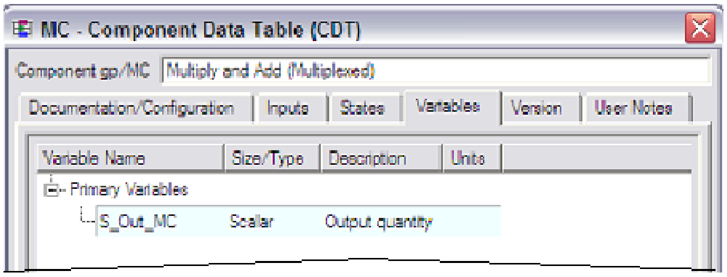
Input Name	Value/Type	Description	Units
Connected Inputs			
S_In1_MC	F_external	Input quantity	
S_In2_MC	F_spring	Input quantity	
S_In3_MC	F_damper	Input quantity	
Primary Inputs			
C1_MC	1.0	S_In1 multiplier coefficient	
C2_MC	-1.0	S_In2 multiplier coefficient	
C3_MC	-1.0	S_In3 multiplier coefficient	

Figure 14 Inputs Tab for MC Component

The example input, S_In2_MC, describes a signal input into an MC component port from an output Gain component. As you can see, there are three input ports. Easy5 automatically increments the port numbers if there is more than one to make each input unique.

The first part of the input or output name identifies the type of input or output. In this case, it is the signal input as designated by the letter S. The second part of the input in the example shown is designated as "In" meaning that this is an input. In this case, it is In1, In2, or In3. The last part of the input or output defines the component using the default component name, in this case, the MC component.

Figure 15 shows an example of a signal output from the MC component, S_Out_MC. Since there is just one output, there is no port number indicated.


MC - Component Data Table (CDT)
Component gp/MC
Multiply and Add (Multiplexed)
Documentation/Configuration
Inputs
States
Variables
Version
User Notes
Primary Variables

Variable Name	Size/Type	Description	Units
S_Out_MC	Scalar	Output quantity	

Figure 15 Output Variables Tab for MC Component

Note: Component input and output default names can be changed to a *user-defined* name of up to 60 alphanumeric characters. See "User-Defined Names", for information on how to define your own names.

Blocks

In order to simplify the process of model building, Easy5 contains a set of predefined library components referred to as *blocks*, which represent commonly used dynamic and mathematical effects. These blocks model many common dynamic effects such as integration, linear transfer function, and function generation, and specialized effects such as coulomb friction and hysteresis. Blocks are contained in the General Purpose (GP) Library, and the Interactive Simulation (IS) library, both of which are standard delivered libraries.

A block is considered to be a *primitive* component in that it only has the primitive attributes of a component. For example, blocks are *generally* single input/single output. An example is the TF (Transfer Function) block. This block is generally used with a single input (named S_In_TF) and a single output (named S_Out_TF). Blocks may be multi-input/multi-output, however the connections between blocks *are single data connections*.

For example, the MC (multiply and add) block has 3 inputs (S_In_MC, S_In2_MC, and S_In3_MC) and one output (S_Out_MC). However, connections between MC and other blocks only transfer one data variable.

The real advantage to using blocks in your Easy5 model is that these blocks, which are actually Fortran subroutines in compiled form, have been thoroughly verified and written by modeling professionals and are protected from modification. Not only do they contain a high degree of modeling accuracy, but they also have been carefully designed for speed. Thus, you can concentrate on the interconnection of blocks to build your system model, and be assured that the theoretical equations and the implementation of these equations are correct.

Standard Components

Standard components are library components that are different from blocks in that they may be multi-input/multi-output and the connections between these components can be *multi data connections*.Figure 16 is a good example of using standard components. These components are from the EC library. Even though only one connection line is shown between these components, they are bidirectional and multi-data connections. Each connection line shows the connection of five variables: T, W, SH, CO, and P, where P (pressure) connects in the negative (upstream) direction. Such attributes are not possible with blocks.

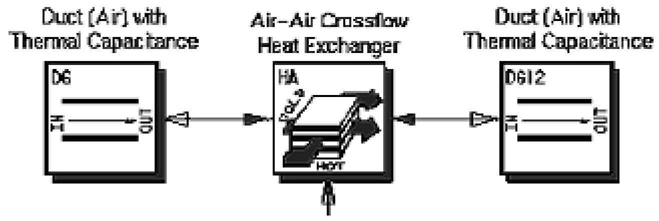

Duct (Air) with Thermal Capacitance
D6
IN
OUT
Air-Air Crossflow Heat Exchanger
HA
COLD
HOT
Duct (Air) with Thermal Capacitance
D612
IN
OUT

Figure 16 Example of Using Standard Components

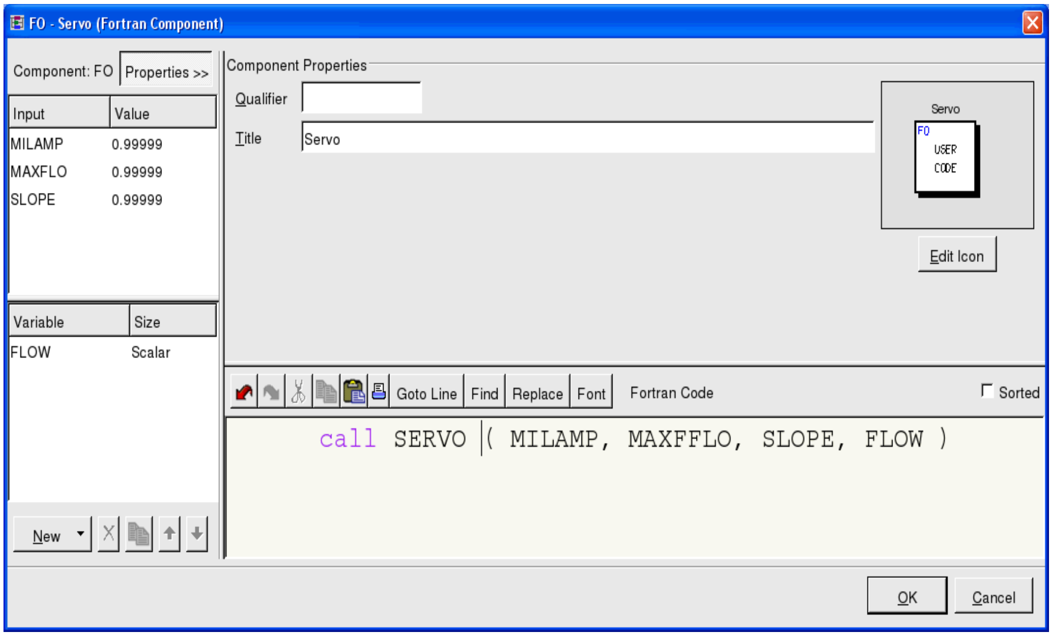
The GP and IS libraries are considered to be block libraries, because the components exhibit block like characteristics, whereas the HB & HC (Hydraulic), EC (Environment Control), VC (Multiphase Fluid) and the PT (Powertrain) libraries are called component libraries, and are purchased (or licensed) libraries.

Note: The terms *component* and *block* are sometimes used interchangeably without loss of meaning.

Code Components

You have the option of modeling all or part of your system in Fortran and or C code. Fortran and C code are added to an Easy5 model using a Fortran code component or a C code component. Once defined, Fortran and C components are connected directly into the rest of your Easy5 schematic just like Standard components. You can place as many code components in your model as you wish.

Fortran components are defined by placing an empty Fortran template in your model schematic and then placing Fortran code in it. A simple example of creating a Fortran component is shown below:.



Qualifier:
Title: Servo

Input	Value
MILAMP	0.99999
MAXFLO	0.99999
SLOPE	0.99999

Variable	Size
FLOW	Scalar

```
    call SERVO ( MILAMP, MAXFFLO, SLOPE, FLOW )
```

Figure 17 Example of Fortran Component Data Table

Fortran code is entered directly into the component using the full screen editor provided or, you can import existing source code into the component and then edit it as needed. Input and output variables referenced in the respective source code which will be connected to other components are then declared.

You also have the option of requesting that Easy5 sort the equations in your Fortran components during model generation. C components are very similar in usage and structure as the example of the Fortran component shown in Figure 17. For more information refer to "C Component".

Code components are useful because Easy5's Library components may not provide the exact modeling effect(s) you need. Code components also allow you to include routines from external libraries in your Easy5 model, to access external data sets or files during the course of an Easy5 analysis, and to link in external Fortran, C, or other code.

Complete information on how to create and use code components is also given in the User Guide, Chapter 5 - Code Components in the section "Creating a User Code Component".

User-defined Library Components

Library component code, representing the equations that define the input and output relationships represented by the Library Component, is written in stylized Fortran that contains various markers, allowing the Easy5 Model Generation program to create a unique copy of the code each time it is used in a given model.

You should write Library components to model specialized effects or subsystems when these features are to be included more than once in the same model, or used in other Easy5 models.

Library Component inputs and outputs are defined in the same way as inputs and outputs from User Code components. Input and output specification forms are used to define the inputs and outputs. You complete these forms in the same manner used to define User Code inputs and outputs, as described in the User Guide, Chapter 5 - Code Components. Library components provide you with all the flexibility of Fortran components, in addition to allowing you to generate a component library that is unique to your application. Once defined, Library Components are used exactly like any other Easy5 component. Users of Library components may be precluded from modifying the underlying code. This allows tight configuration control and maintenance of application specific component libraries developed for a given user community.

Library components are constructed almost exactly like Fortran components, except that Library Components are written in a pseudo-Fortran language that facilitates multiple component usage.

Figure 18 shows a section of code as it would appear both in a Library component and a Standard component.

Library Component Code

```
C   COMPONENT FI
C
  R FI= (V_Supply_FI - V_Load_FI)Inss_FI
  IF (R_FI .LT. 0) R_FI = 0
C      Rate of continuous state V_Load_FI
 IF(INX( 2).NE.0) XDOT( 2)=(I_Supply_FI - I_Supply_RE) / C_FI
C      Rate of continuous state I_Supply_FI
 IF(INX( 1).NE.0) XDOT( 1)=(V_Supply_FI - V_Load_FI-
 &           I_Supply_FI * R_FI)/L_FI
```

Standard Component Code

```
C   COMPONENT FI
C
 CALL EZCRFI(I_Supply_FI,XDOT(1),INX(1),V_Load_FI,XDOT(2),INX(2)
&   R_FI,Inss_FI,C_FI,L_FI,I_Load_FI,V_Supply_FI,'FI')
```

Figure 18 Comparison of Library Component and Standard Component Code

The resulting code is automatically filled in by the Easy5 code generator when the respective Library component is referenced in a model, to ensure that all variable names and statement labels are uniquely defined in the executable model. For more information, refer to the User Guide, Chapter 5 - Code Components.

Extension Components

Extensions are special components used to link other stand-alone applications to your Easy5 model. Although a powerful modeling feature, Extensions are not something the average user develops, but rather they are a mechanism for other application software developers to seamlessly link their modeling tools to Easy5.

Extensions that have been developed include links to MSC.NASTRAN, GSDS autocode generator, and the multibody dynamics modeling tools DADS® developed by LMS and Adams® (also by MSC.Software). Users of these Extensions will be able to access any mechanism generated in these packages as a component in the Easy5 *Add Components window.*

Extension components are added just like other components via the Add Components window. For more information, refer to the Technical Note on Easy5 Extensions.

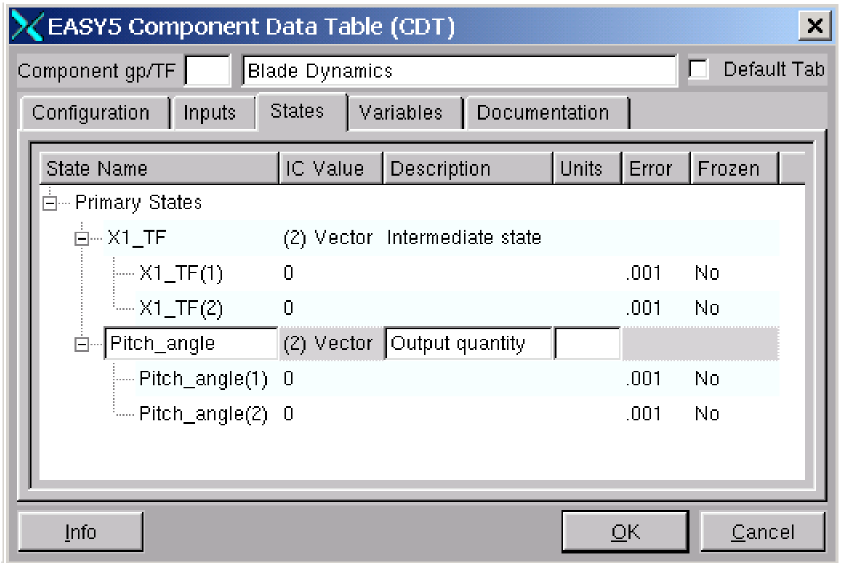
Dimensioned Components

The inputs and outputs of components can be either scalar or array quantities. Sometimes it is convenient to design a model with many identical scalar loops as one loop with variables that are arrays: i.e., a vectorized loop. This not only simplifies the model schematic, but speeds the execution of the model during analysis, by lowering the amount of program "overhead" for name storage and by utilizing higher speed vector processing algorithms on your computer.

By default, all component data are considered as scalars, and are given the default dimension of 0. However, you can dimension or vectorize many standard components by changing the values in the (Dimension parameter N): data field in the CDT Configuration Tab.

An example of a dimensioned component is shown in Figure 19. The transfer function component (TF) data table is shown (with the “States” tab visible) with the dimension parameter “Channels (N)” changed from a default value of 0 (scalar) to 2. Notice that all inputs and outputs are defined by two element vectors.

This component can be vectorized with only one dimension parameter, N -- as shown in the Configuration tab in for this component in Figure 20.

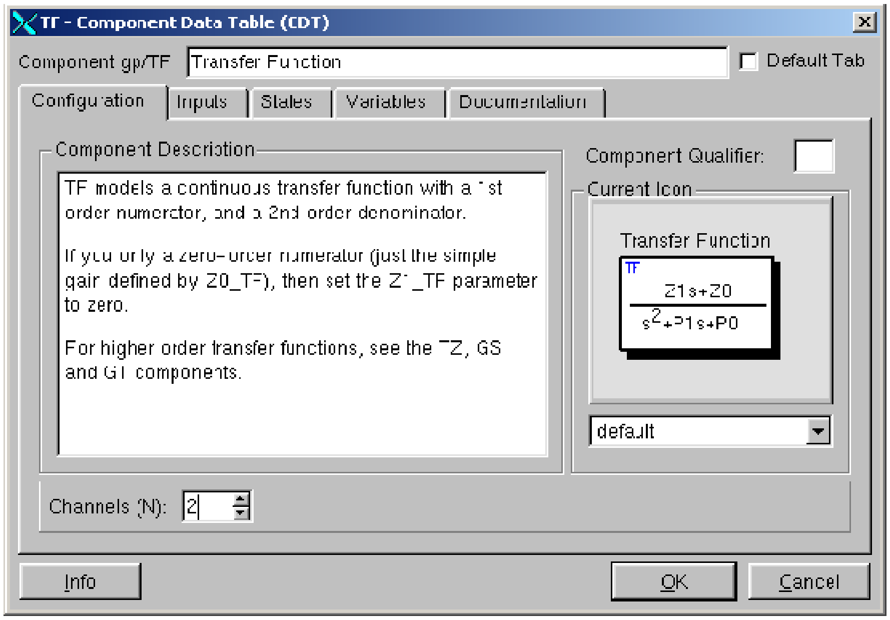

EASY5 Component Data Table (CDT)

Component gp/TF: Blade Dynamics — Default Tab

Tabs: Configuration | Inputs | States | Variables | Documentation

State Name	IC Value	Description	Units	Error	Frozen
Primary States					
X1_TF	(2) Vector	Intermediate state			
X1_TF(1)	0			.001	No
X1_TF(2)	0			.001	No
Pitch_angle	(2) Vector	Output quantity			
Pitch_angle(1)	0			.001	No
Pitch_angle(2)	0			.001	No

Figure 19 Sample Data Table of a Vectorized Component


TF - Component Data Table (CDT)

Component gp/TF: Transfer Function — Default Tab

Tabs: Configuration | Inputs | States | Variables | Documentation

Component Description:

TF models a continuous transfer function with a 1st order numerator, and a 2nd order denominator.

If you only a zero-order numerator (just the simple gain defined by Z0_TF), then set the Z1_TF parameter to zero.

For higher order transfer functions, see the TZ, GS and GT components.

Component Qualifier:

Current Icon: Transfer Function — default

Channels (N): 2

Figure 20 TF Component Vectorized with one dimension parameter

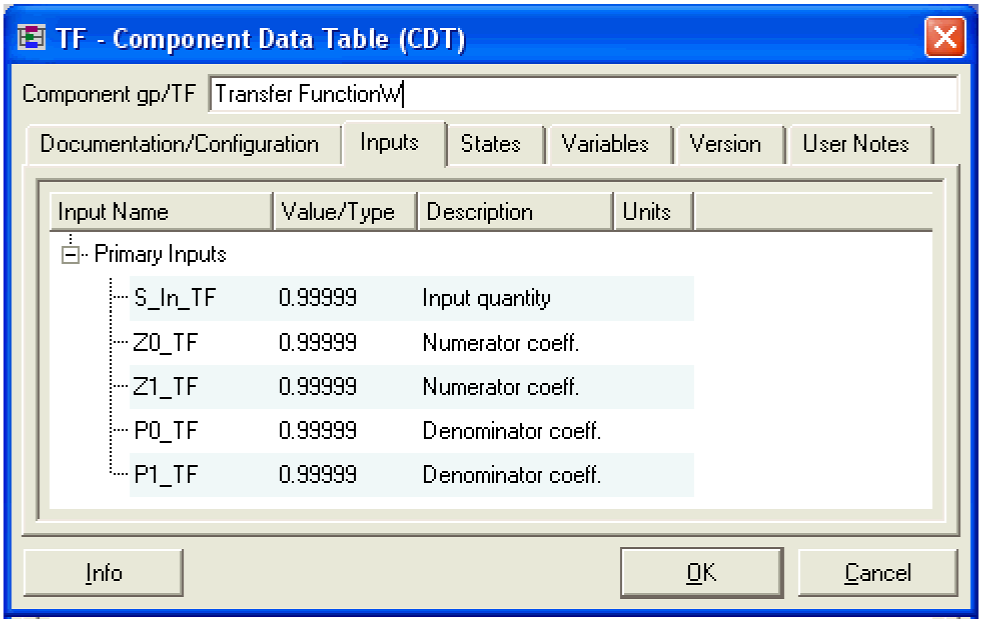
This is apparent from the dimension parameter N displayed in the “Channels (N)” data field. General Purpose (gp) Library components that can be vectorized with 2 dimensions typically have a (dimension parameter) N and a (dimension parameter) M data field in the Configuration tab for the component instance.

Component Data Table

See also: "Components"
"Parameters - Defining Input Values"
"States"

The component inputs and outputs are defined in a data table called the Component Data Table (CDT). The CDT is your view into the component.

To open or examine any component data table, select the component with the middle mouse button or double-click the left mouse button. Open the One-dimensional function generator (FU) component using this method. This opens the component data table as shown in Figure 21. The Component Data Table uses tabs to display and access all component data, including inputs, outputs, variables, states, and configuration data that are editable by the user.



Component gp/TF: Transfer FunctionW

Input Name	Value/Type	Description	Units
Primary Inputs			
S_In_TF	0.99999	Input quantity	
Z0_TF	0.99999	Numerator coeff.	
Z1_TF	0.99999	Numerator coeff.	
P0_TF	0.99999	Denominator coeff.	
P1_TF	0.99999	Denominator coeff.	

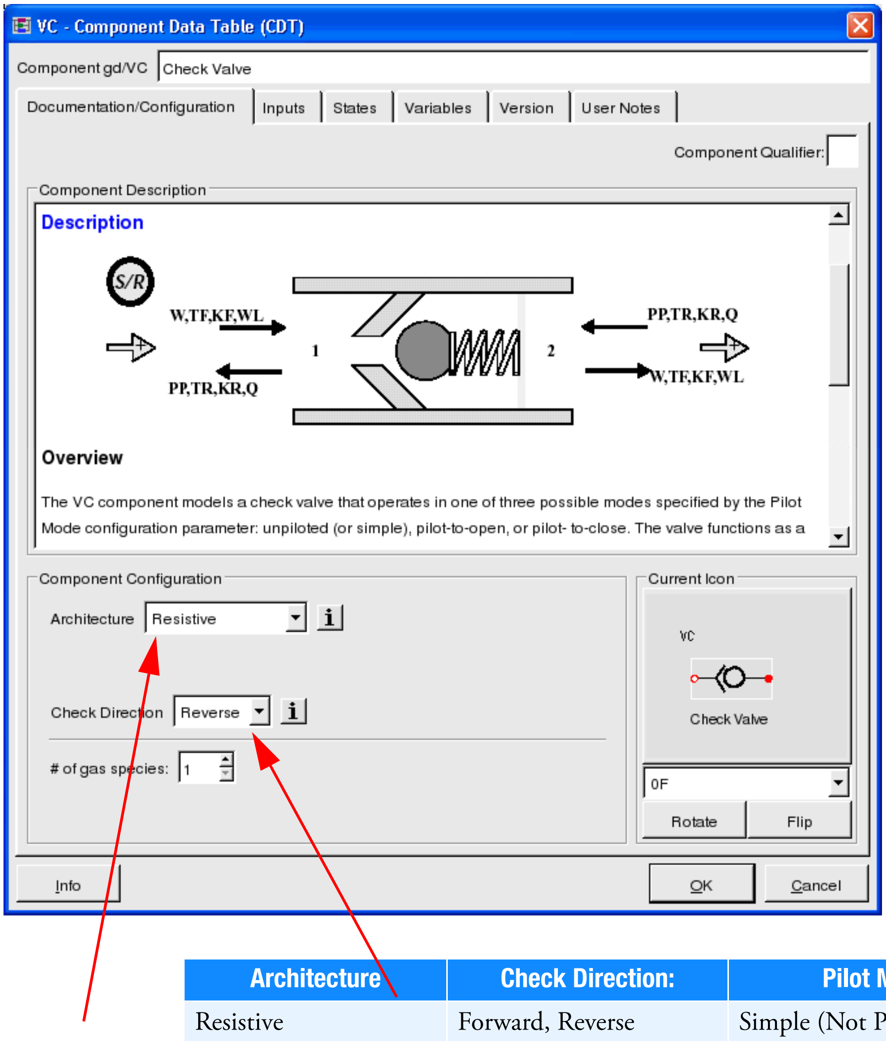
Figure 21 Example of a Component Data Table

Specific information on the component is accessed by selecting the appropriate Tab. To access additional important information about a component, click the [Info] button; this opens the online component file. When you have finished examining this information, close the dialog and continue to further define the component.

Documentation/Configuration Tab

A Component Description field describes the component, and if applicable, configuration settings are offered here to select the appropriate configuration. Since some components have many configurations each configuration provides different equations, data input/out and icons.

The GD Library Check Valve (VC) component has multiple configurations that can be selected via 3 feature parameters, and one dimension parameter:



VC - Component Data Table (CDT)

Component gd/VC: Check Valve

Documentation/Configuration | Inputs | States | Variables | Version | User Notes

Component Qualifier:

Component Description

Description

Overview

The VC component models a check valve that operates in one of three possible modes specified by the Pilot Mode configuration parameter: unpiloted (or simple), pilot-to-open, or pilot- to-close. The valve functions as a

Component Configuration

Architecture: Resistive

Check Direction: Reverse

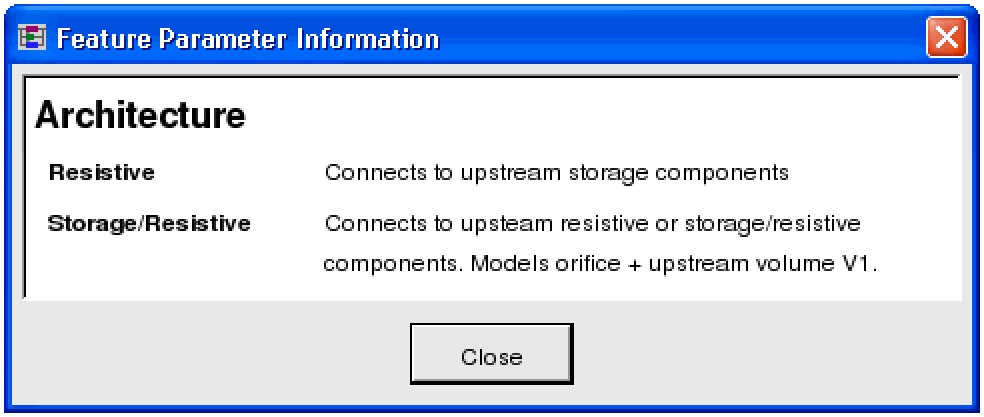
# of gas species: 1

Current Icon

Architecture	Check Direction:	Pilot Mode:
Resistive	Forward, Reverse	Simple (Not Piloted)
Resistive/Storage	Forward, Reverse	Pilot-to-Open
	Forward, Reverse	Pilot-to-Close

Figure 22 Example of Multiple Configurations

An “Information” (“i”) button is provided to obtain more detailed information on a specific feature parameter, as shown at left. In addition, the current feature parameter selection is written to the “Component Description” field.



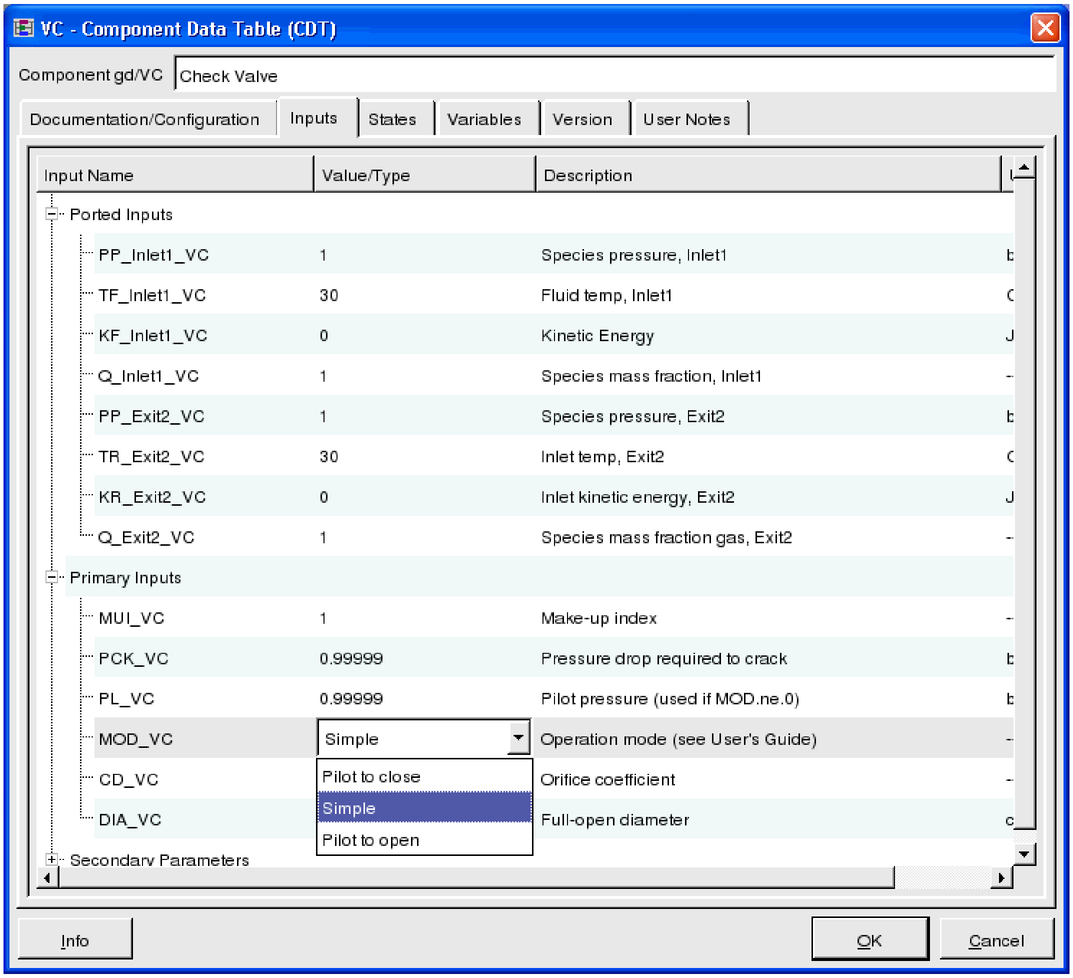
Architecture

Resistive	Connects to upstream storage components
Storage/Resistive	Connects to upsteam resistive or storage/resistive components. Models orifice + upstream volume V1.

Figure 23 Information Button

Inputs Tab

Editable fields including: Input Name with a display of primary and secondary inputs and an optional display of connected inputs, Value/Type, Description and Units.



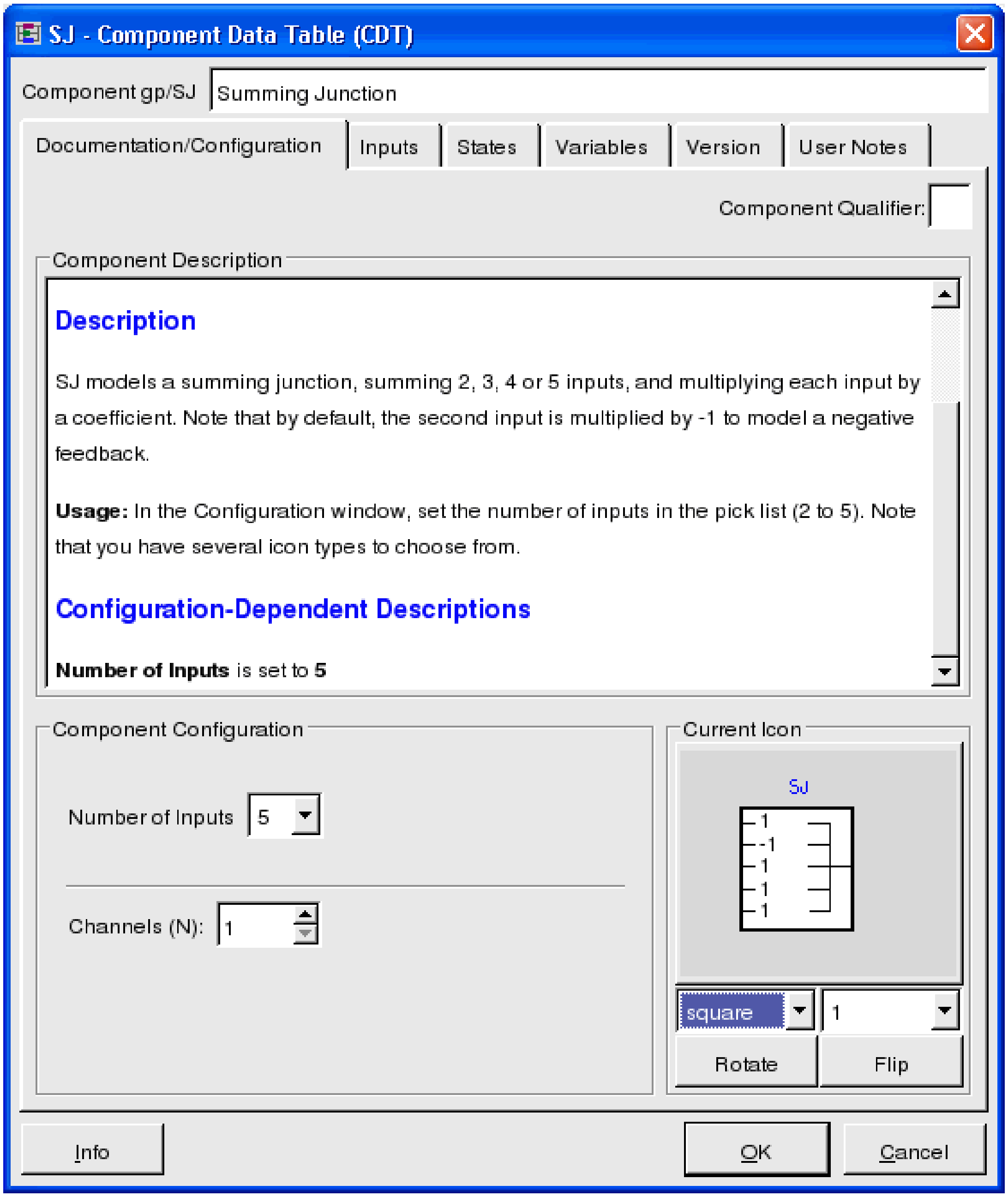
Component gd/VC: Check Valve

Input Name	Value/Type	Description
Ported Inputs		
PP_Inlet1_VC	1	Species pressure, Inlet1
TF_Inlet1_VC	30	Fluid temp, Inlet1
KF_Inlet1_VC	0	Kinetic Energy
Q_Inlet1_VC	1	Species mass fraction, Inlet1
PP_Exit2_VC	1	Species pressure, Exit2
TR_Exit2_VC	30	Inlet temp, Exit2
KR_Exit2_VC	0	Inlet kinetic energy, Exit2
Q_Exit2_VC	1	Species mass fraction gas, Exit2
Primary Inputs		
MUI_VC	1	Make-up index
PCK_VC	0.99999	Pressure drop required to crack
PL_VC	0.99999	Pilot pressure (used if MOD.ne.0)
MOD_VC	Simple	Operation mode (see User's Guide)
CD_VC		Orifice coefficient
DIA_VC		Full-open diameter
Secondary Parameters		

MOD_VC options: Pilot to close, Simple, Pilot to open

Figure 24 Example of Inputs and Other Parameters for VC Component

Dynamic Sizing of the Summing Junction - You can dynamically set the number of inputs from 2 to n.


SJ - Component Data Table (CDT)

Component gp/SJ Summing Junction

Component Qualifier:

Component Description

Description

SJ models a summing junction, summing 2, 3, 4 or 5 inputs, and multiplying each input by a coefficient. Note that by default, the second input is multiplied by -1 to model a negative feedback.

Usage: In the Configuration window, set the number of inputs in the pick list (2 to 5). Note that you have several icon types to choose from.

Configuration-Dependent Descriptions

Number of Inputs is set to **5**

Component Configuration

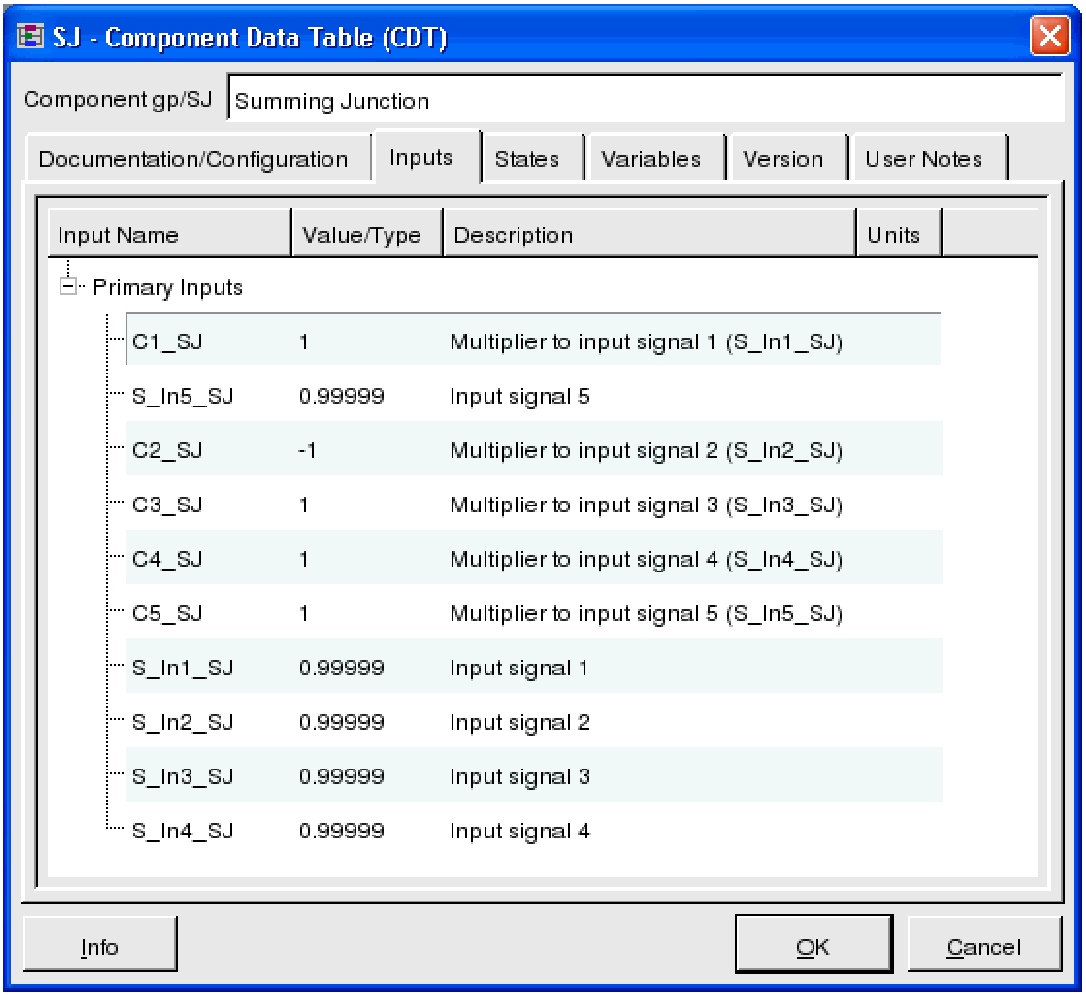
Number of Inputs: 5

Channels (N): 1

Current Icon

Figure 25 Dynamic Sizing of Summing Junction

In this example, below, the number of inputs has been set to 5:


SJ - Component Data Table (CDT)

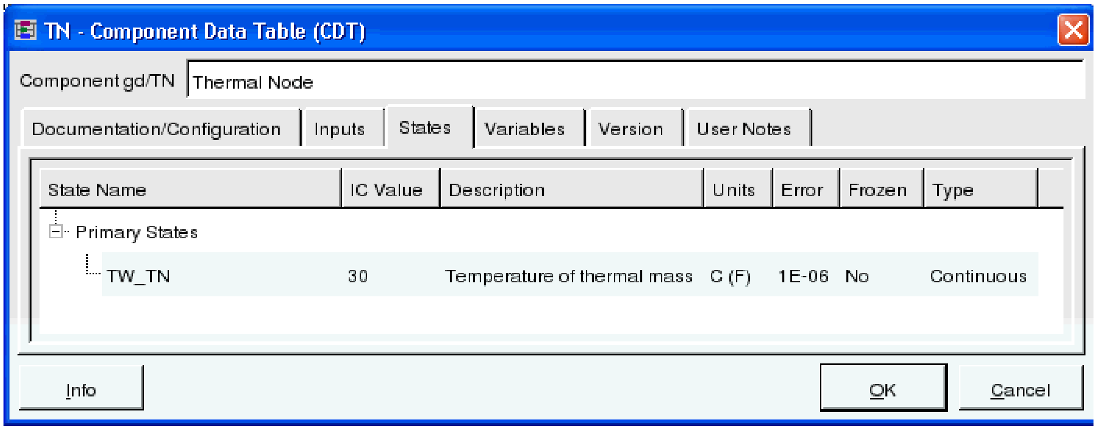
Component gp/SJ: Summing Junction

Input Name	Value/Type	Description	Units
C1_SJ	1	Multiplier to input signal 1 (S_In1_SJ)	
S_In5_SJ	0.99999	Input signal 5	
C2_SJ	-1	Multiplier to input signal 2 (S_In2_SJ)	
C3_SJ	1	Multiplier to input signal 3 (S_In3_SJ)	
C4_SJ	1	Multiplier to input signal 4 (S_In4_SJ)	
C5_SJ	1	Multiplier to input signal 5 (S_In5_SJ)	
S_In1_SJ	0.99999	Input signal 1	
S_In2_SJ	0.99999	Input signal 2	
S_In3_SJ	0.99999	Input signal 3	
S_In4_SJ	0.99999	Input signal 4	

Figure 26 Dynamic Sizing of Summing Junction

States Tab

Displays IC Value, Description, Units, percentage of Error, and Frozen flag. Secondary states (not shown here) are assigned to “internal” switch states.



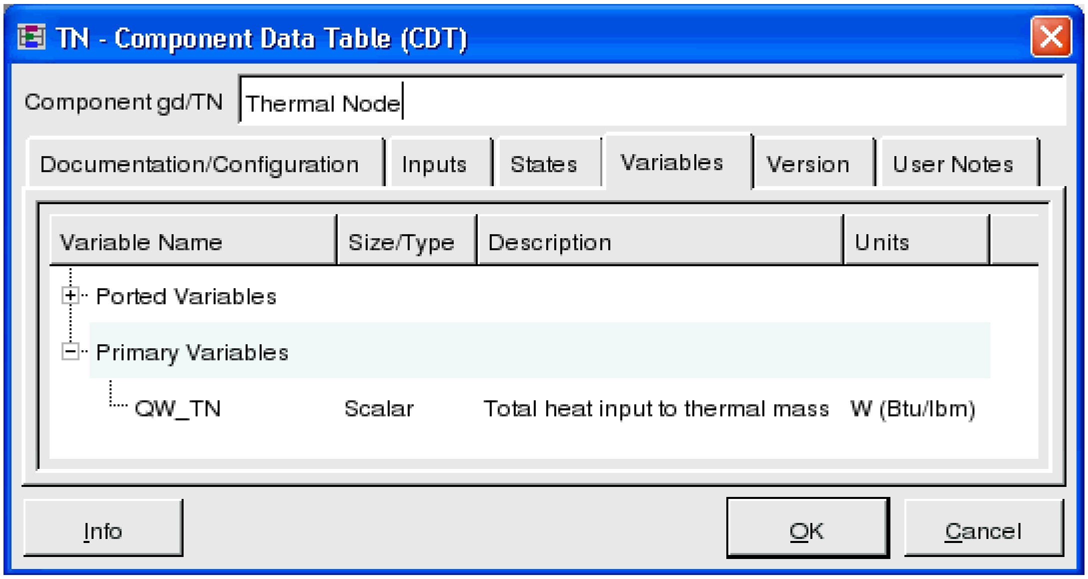
Component gd/TN: Thermal Node

State Name	IC Value	Description	Units	Error	Frozen	Type
Primary States						
TW_TN	30	Temperature of thermal mass	C (F)	1E-06	No	Continuous

Figure 27 Example of States Tab

Variables Tab

Displays complete information on outputs: Name, Size/Type, Description and Units.



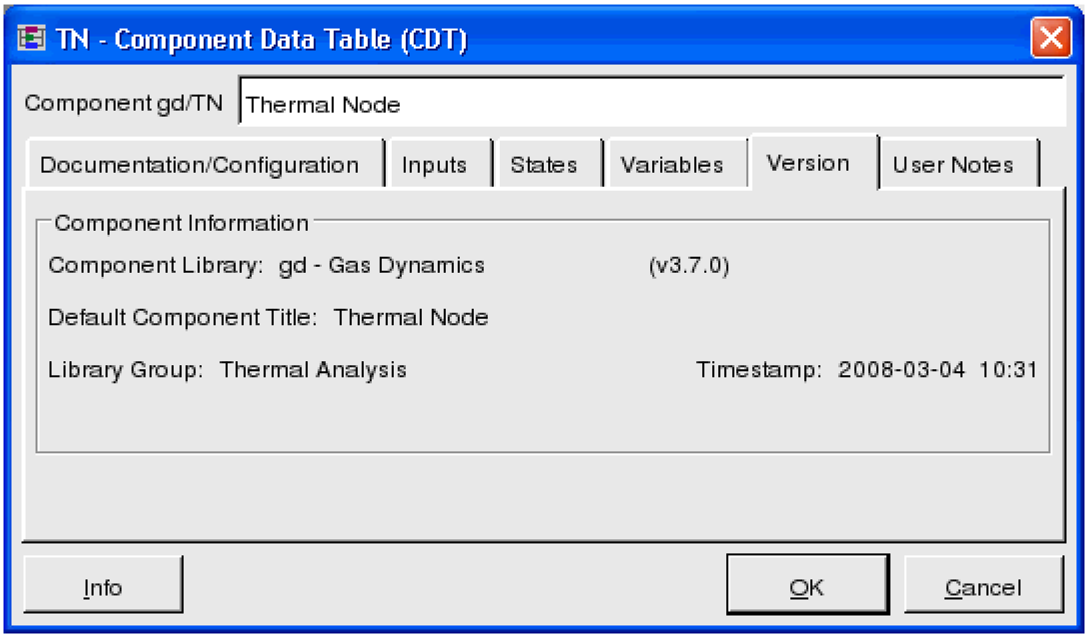
Component gd/TN: Thermal Node

Variable Name	Size/Type	Description	Units
Ported Variables			
Primary Variables			
QW_TN	Scalar	Total heat input to thermal mass	W (Btu/lbm)

Figure 28 Example of Variables Tab

Version Tab

This tab offers configuration control Information for the component including the library, default title, library group, and timestamp..



Component gd/TN Thermal Node

Component Information

Component Library: gd - Gas Dynamics (v3.7.0)

Default Component Title: Thermal Node

Library Group: Thermal Analysis Timestamp: 2008-03-04 10:31

Figure 29 Example of Version Tab

User-Comments Tab

This tab is used to provide the user the ability to enter text to document the particular instance of the component in the model. This field only accepts plain text. User Notes are included in the model documentation automatically.

Connecting Components

See also:"Connection Lines"

Components are connected using connection lines. Connections lines denote a transfer of data between components. The usage of connections in Easy5 sets it apart from other similar software tools. In Easy5, connections may be multi-input/output, and may be bidirectional (data flows both upstream and downstream). As a result, a single connection arrow indicates that one or more input/output data connections have been established between components that is, multiple redundant arrows are not drawn.

There are three types of connections:

- Default connections

- Port connections
- Custom connections

Each of these methods of connecting components will be described in the sections that follow.

Rules for Connecting Components

The following rules apply when connecting components:

- Any component input can be driven by, or connected to the output of another component.
- Each input signal can only be driven by one output signal.
- An output can be connected to one or more component inputs.
- Inputs and outputs don't have to be connected to anything.
- When an output is connected to an input, ***the name associated with the input is replaced with the output name***. In other words, only one variable name is associated with a connection between two components, and that name is always the output variable name.
- An element of a vector output can be connected directly to a scalar input, but a scalar output cannot be connected directly to an element of a vector input. The VX component can be used to connect scalar outputs to a vector input (VX is a MUX function).

Default Connections

You will find that many Easy5 standard components have similar types of inputs and outputs, as these components are often connected on a block diagram. To streamline the connection specifications for such components, default connections can be made.

If you make a default connection, the connection will occur provided that two conditions are met.

- First, the output name(s) of the *From* component must be the same as the input name(s) of the *To* component.
- Second, the output and input type(s) must match.

This means they must both be ported or nonported types. (“Port” is a designation, associated with certain inputs and outputs, that is used in the default connection logic.)

A default connection can be established by the following method:

1. Select the "from" component
2. Select the "to" component.

   When a connection is made, an arrow is drawn between the respective components on the Easy5 schematic. This arrow indicates that one or more input/output connections have been established (i.e., redundant arrows are not drawn).

The default connection is the easiest way of connecting compatible components.

Figure 30 shows some typical components with ported inputs and outputs.

A default connection can be made from TF to MD. Specifically, the output quantity S at port (Out) of TF will, by default, connect to the input quantity S at port (In) of MD, because the names match and they are ported quantities.

On the other hand, a default connection cannot be made from TF to XP, because even though they both are ported quantities, the names are not the same (i.e. S doesn't equal W).

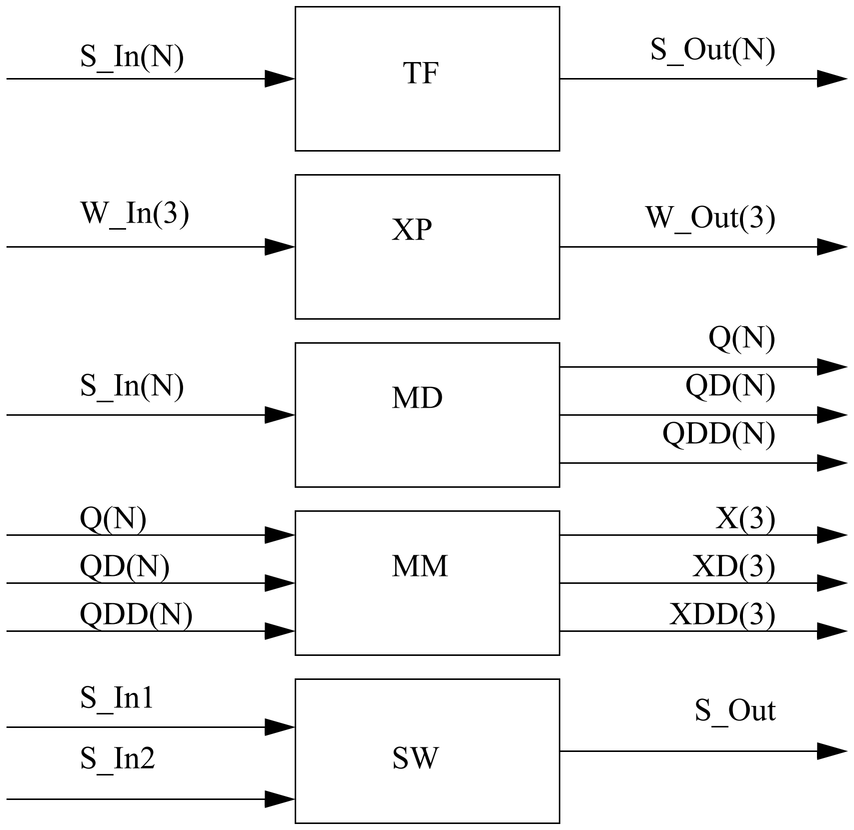


Figure 30 Typical Component Inputs and Outputs

As another example, note that a default connection can be made from components MD to MM. All the quantities are of the same type (nonported), and the output and input names are the same. Multiple inputs and outputs will connect to each other if the default connection criteria is satisfied.

Often, a component with only one ported output will be connected to a component with multiple ported inputs. As an example, consider a connection from TF to SW. In this case, a default connection would take place, but there is a choice: SW has two input ports with the same quantity names, S_In1 and S_In2. To resolve this matter, Easy5 would display a dialog allowing you to choose which connection to make.

When the default connection logic does not meet your connection requirements, you can create "Port" (ported input to ported output) or "Custom" connections to further modify a connection scheme between two components.

Port Connections

Port connections allow you to specify the connection of individual output ports to input ports. A port is simply an attribute, or tag, assigned to one or more inputs/outputs of a component and is used by Easy5 when making connections. Ports are identified with a unique port name and port number. To see how port connections work, try a connection from component AF to component SJ using the following procedure.

1. Specify the direction of the connection by first selecting the From component (AF) and then selecting the To component (SJ). A port connection table opens, as shown in Figure 31.

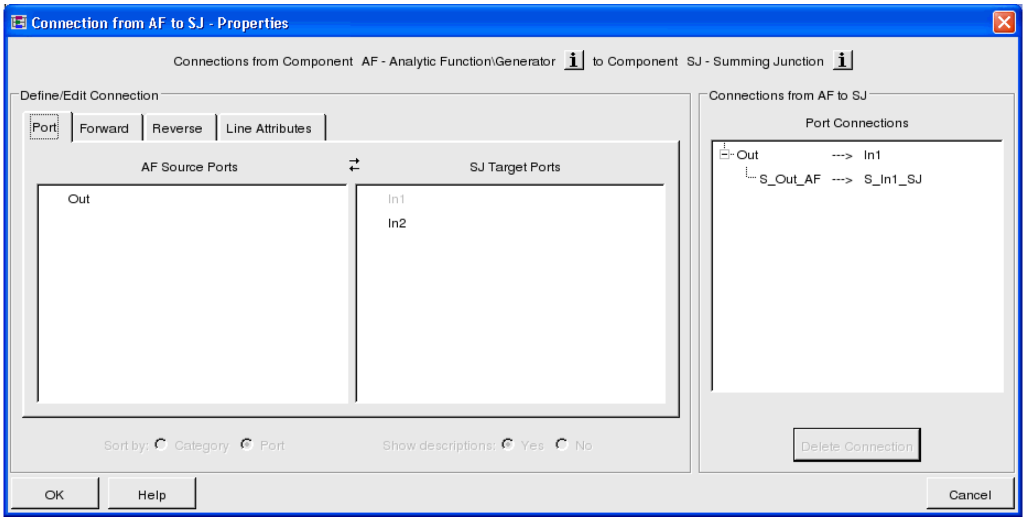


Figure 31 Connection Data Table for the AF -> SJ Connection

The connection table shows that the AF component has one output “Out”. The SJ component is a summation block; it has two ported inputs In1 and In2 (Feedback). Easy5 automatically opens the port connection table because it needs to know which input of SJ you want to connect to. The port numbers are really not important; the port name are the most useful because they describe the port.

2. Select **Out**, and then select **In1**. The connection is displayed in the Connection Specification column as Out -> In1.
3. Select **OK** to close the dialog. A connection line is made between the two components with a connection line label. This connection is examined in more detail in the next section.

Default Port Connection Points

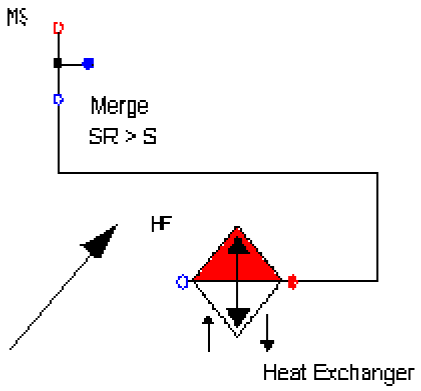
Component designers can designate a specific default connection point on a component icon that indicates where the end of the connection line is to be drawn when the connection is made. In addition, the default connection points are preserved:

- If components are moved
- If an alternate icon is selected.

Note: Users can override the default connection point, although this is NOT recommended.

Example 1:

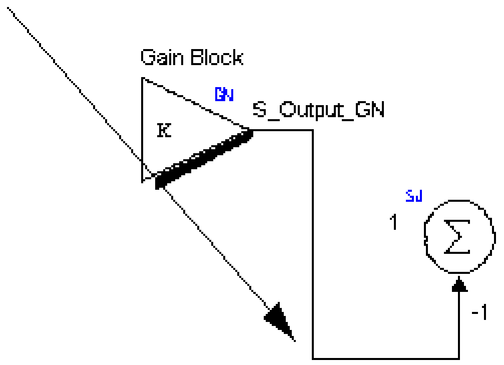
Autorouted connection from the Exit of a Heat Exchanger to Inlet2 of a Merge (regardless of their relative positions).



Example 2:

Autorouted connection from a Gain block to the feedback port of a summing junction.

The “Port” connection option gives you the flexibility to specify the connection of individual ported outputs to ported inputs.



This option is useful in connecting hydraulic and environmental control system components, since these components contain input/output ports defined with multiple variables.

On occasion, the default and port connection options will not meet your connection requirements. When this is the case, you can create "Custom" connections.

Custom Connections

Custom connections are used to connect components with incompatible port designations and input/output names. If this conditions occurs while connecting components, Easy5 recognizes that a default connection cannot be made. The message line warns you with the comment "cannot make a default connection", and automatically displays the connection table.

Specifying custom connections provides you with a great deal of flexibility by allowing you to connect any component output to any component input, regardless of name dissimilarity. You can even connect an element of a vector output to a scalar input. The only thing you cannot do with custom connections is connect a scalar output to an element of a vector input. This type of connection can only be made using the VX component or a code component (C or Fortran).

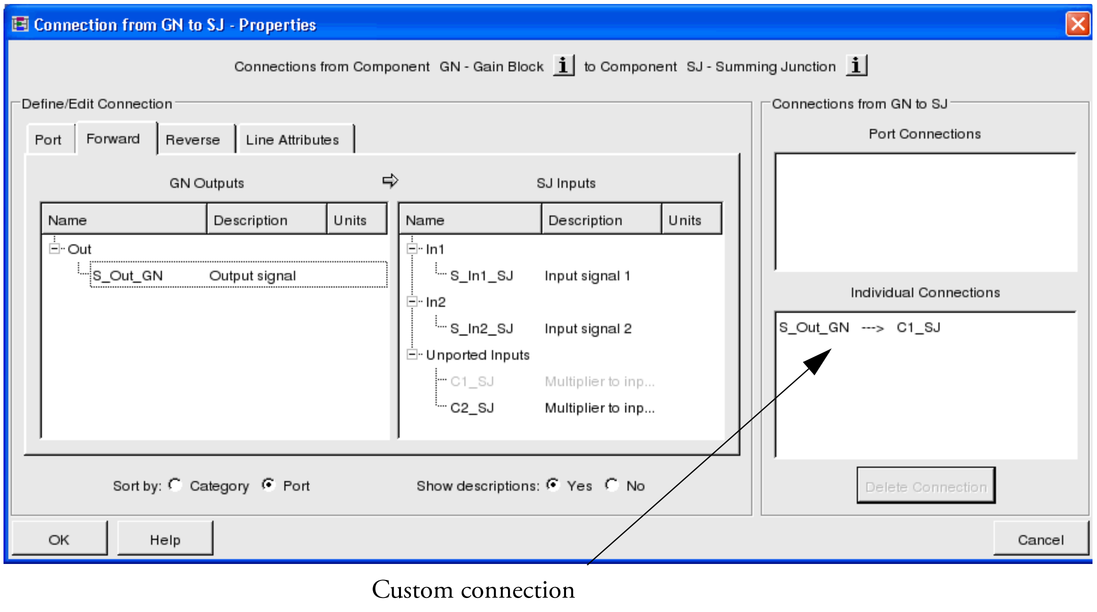
You may also force the connection to be a custom connection. A custom connection is established by the following method.

- Select the "from" component with a CLICK-L.
- Select the "to" component with a <Ctrl>CLICK-L (hold the Ctrl key while selecting the "to" component).

Note: Pressing the Ctrl key while selecting the "to" component forces a custom connection.

A Connection Data Table will appear similar to the table shown in Figure 31. To complete a custom connection, select the appropriate output name from the first column of the connection table. Then select the respective input name in the second column.

You will notice that as soon as an input is specified, a connection is noted in column three of the connection table, as shown in Figure 32.



Connection from GN to SJ - Properties

Connections from Component GN - Gain Block to Component SJ - Summing Junction

Define/Edit Connection

Port | Forward | Reverse | Line Attributes

GN Outputs

Name	Description	Units
Out		
S_Out_GN	Output signal	

SJ Inputs

Name	Description	Units
In1		
S_In1_SJ	Input signal 1	
In2		
S_In2_SJ	Input signal 2	
Unported Inputs		
C1_SJ	Multiplier to inp...	
C2_SJ	Multiplier to inp...	

Sort by: ( ) Category [✓ Port]

Show descriptions: [✓ Yes] ( ) No

Connections from GN to SJ

Port Connections

Individual Connections

S_Out_GN ---> C1_SJ

Delete Connection

OK | Help | Cancel

Figure 32 Making a Custom Connection GN2 -> SJ Connection

Also notice that the connected input is dimmed, indicating that this input has been used and that it cannot be connected to any other output.

You can make multiple output to input connections in a connection table as long as unconnected inputs are available. When you have finished making connections, close the connection table by selecting [OK].

If you close the connection table with “Cancel”, all the changes you made to the table will be lost; any existing connections in effect when the table was first opened will still be there.

Note: A custom connection is required when connecting to or from any Fortran component, because Fortran inputs/outputs cannot be defined with port numbers. As a result, a connection data table automatically displays forcing you to perform a custom connection to or from a Fortran component.

If any of the components in your model have been vectorized, their outputs will be appropriately dimensioned. You can make a custom connection between a vectorized output and a scalar input using the custom connection method described above.

You can make default, ported, or custom connections between vectorized components only when the dimensions of the two components are equal. If you want to make a connection between dissimilarly dimensioned components, you must insert a Fortran component between these components and make the connections manually with Fortran (i.e., "outname(#)" = "inname(#)"). A Fortran component must also be used if you want to change the order in which elements of vectorized inputs and outputs are connected. See the next section, "Connecting Incompatibly Vectorized Components", for more information.

In general, when you specify a connection between components, Easy5 eliminates the Fortran variable name associated with the input and replaces it with the respective output name. Thus, only one variable name exists for each connection between an input/output pair. If you were to connect a scalar output directly to a vector input, the name associated with one element of the vector would have to be replaced with the name of the scalar value. This name change violates the rules of Fortran.

Making a Branch Connection

A component output that “branches” to many components can be made into a “branch connection”. An example of a branch connection is shown in the right hand schematic in Figure 33. In this example, the output of the SF component named “command” connects to three other components. The schematic on the left shows the standard method of displaying three connection lines emanating from the SF component. The branch connection however has only one connection line exiting the SF component, and three branches emanating from the connection line to each component. Functionally, both schematics are identical, but graphically, the branched connection is easier to read and cleaner looking.

To create a branch connection, first select the connection line (instead of the component), then select the component you wish to connect to. Easy5 draws a connection line emanating from the connection line, and puts a solder joint at the junction where the branch connection initiates.

Note that branch connections become anchored.

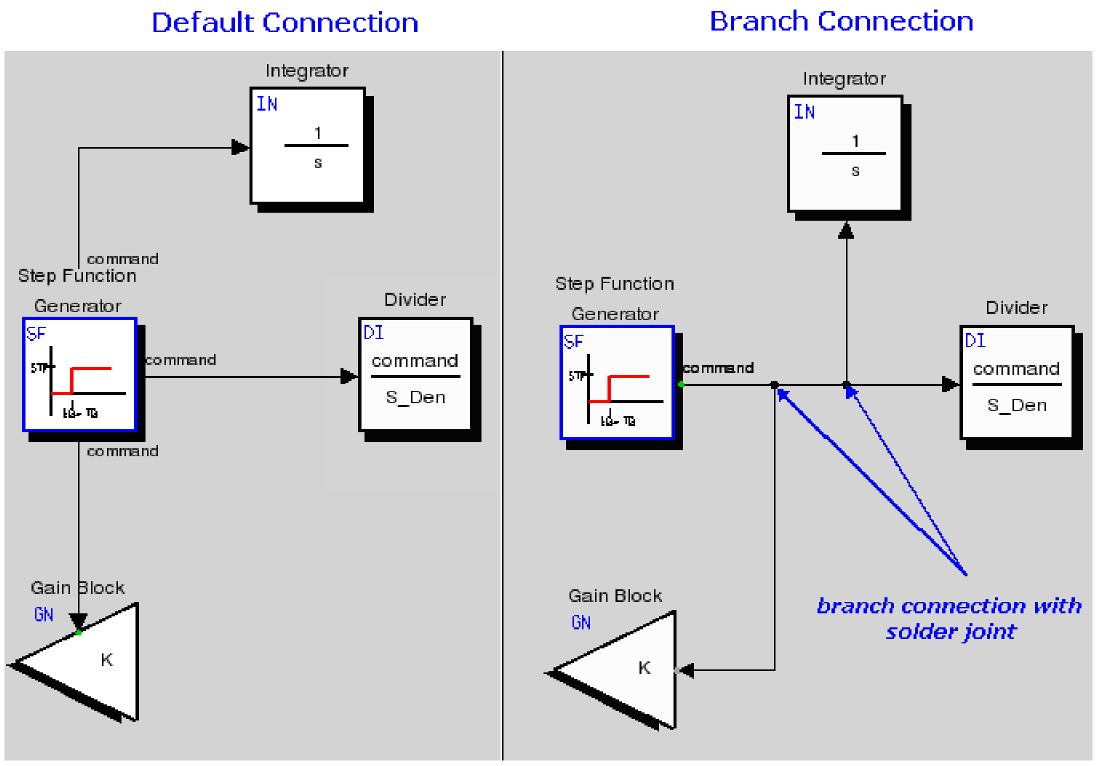


Figure 33 Example of a Branch Connection

Connecting Incompatibly Vectorized Components

There are two kinds of connections that cannot be made directly between the various types of Easy5 components, as follows:

1. You cannot connect a scalar output to an element of a vector or array input.
2. You cannot connect two vectorized components with different dimensions.

To connect a scalar output to an element of a vector input, insert the VX component between the components to be connected. This component connects up to five scalar outputs to a vectorized component of dimension five or less.

If a scalar to vector connection of more than 5 connections is required, then a Fortran component must be used as described in the following example.

To make a connection between a 6 or more scalar outputs to an array input, or connect two or more vectorized components with different dimensions, you must insert a Fortran component between the components to be connected. In effect, the Fortran component acts as a connecting bridge between mismatched inputs and outputs.

For example, to connect the output from six scalar components (six scalar outputs) to a component with a vectorized input of dimension six, the following Fortran component with the following code is needed:

```
C     Fortran COMPONENT INPUTS:  INPUT1, INPUT2, INPUT3, INPUT4,
C                                INPUT5, INPUT6
C     Fortran COMPONENT OUTPUTS: OUTPUT(6)
C
C Fortran CODE:
      OUTPUT(1) = INPUT1
      OUTPUT(2) = INPUT2
      OUTPUT(3) = INPUT3
      OUTPUT(4) = INPUT4
      OUTPUT(5) = INPUT5
      OUTPUT(6) = INPUT6
```

This Fortran component could then be connected between the six scalar and the one vectorized component, because the dimensions match for all input/output interfaces.

Connection Lines

When connections are drawn between components, Easy5 automatically routes the connection line to apply a best fit. The automatic routing of a connection line is called an autoroute connection. The autoroute connection is drawn such that the lines do not cross a component icon, and the shortest possible route is traced. However, you may manipulate connection lines to change the routing. Any time you move the autorouted connection line, it becomes an anchored connection.

Moving Connection Line Endpoints

Connection lines originate from and terminate at connection pins. Connection pins are attached to the component icon, and specify where the connection line endpoints are “pinned” or anchored.

Figure 34 shows the connection pins around the TF component as red carets (^). Every component has a total of 20 connection pins, usually five on each side.

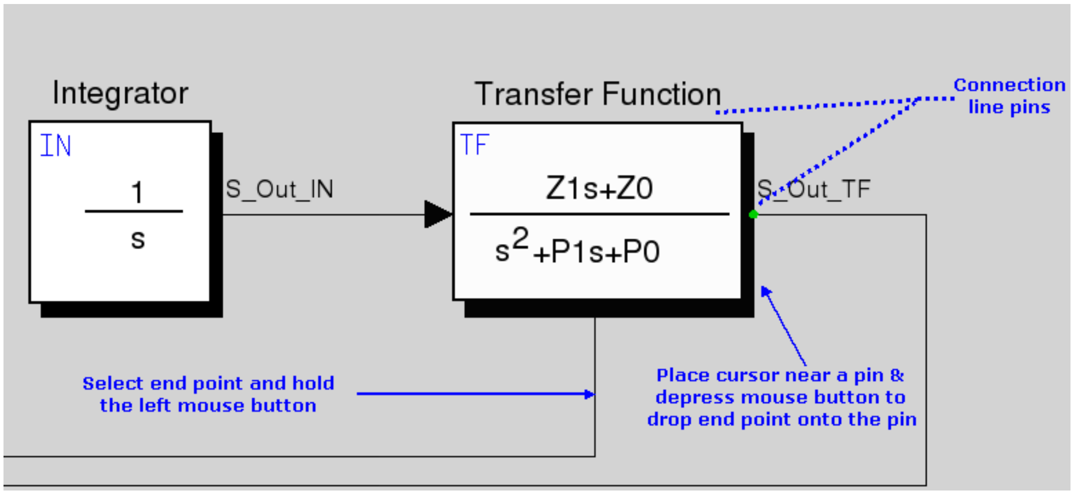

Integrator
IN
1
s
S_Out_IN
Transfer Function
TF
Z1s+Z0
s2+P1s+P0
S_Out_TF
Connection line pins
Select end point and hold the left mouse button
Place cursor near a pin & depress mouse button to drop end point onto the pin

Figure 34 Anchored Connection Line

Connection line endpoints are moved by selecting and moving an endpoint from one pin to another. Figure 34 shows an example of moving the TF component connection line endpoint from the bottom to the right side. Move this connection line endpoint on your model following the steps given here.

Procedure for moving endpoints

1. Select the connection line with HOLD-L from the TF component as shown in Figure 34. In this case, you want to move the output connection line endpoint, so select the connection line near the endpoint.

Note: When you select any connection line and hold the mouse button, the connection pins display to indicate where you can move the endpoint.

2. While holding down the left mouse button, drag the connection line over to the right side of the TF component, and place the cursor near the pin that you want to connect to.
3. CLICK-L to drop the connection endpoint onto the connection pin. The connection endpoint is not anchored, as indicated by the green dot on the endpoint that was moved.

Moving Connection Line Segments

You can also move the line segments of an anchored connection line. A vertical line segment can be moved left and right; a horizontal line segment, up and down as shown in Figure 35.

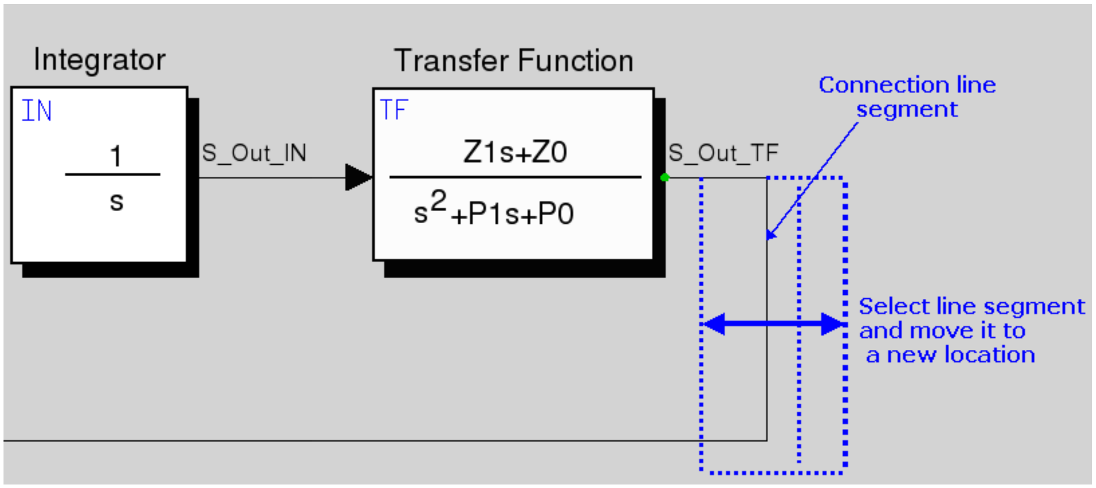


Figure 35 Moving a Connection Line Segment

To move a connection line segment:

1. Select the connection line segment with left-click hold from the TF component.
2. Move the line segment to the right and to the left and drop it where you feel it should be placed.

Changing an Anchored Connection Back to an Autoroute Connection

An anchored connection can be changed back to an autoroute connection by selecting it with SHIFT-CLICK-R, or using the right-click menu item "**Autoroute**".

Customized Line Routing

Default connection line routing uses straight line segments to "autoroute" a connection between the "from" and "to" components. To further customize your connection line routing, you can add a connection "point" or "jog". A connection *point* consists of an additional connection line node (used as an additional "corner" for a connection line segment), while a jog consists of a pair of nodes -- used to easily add a perpendicular line segment.

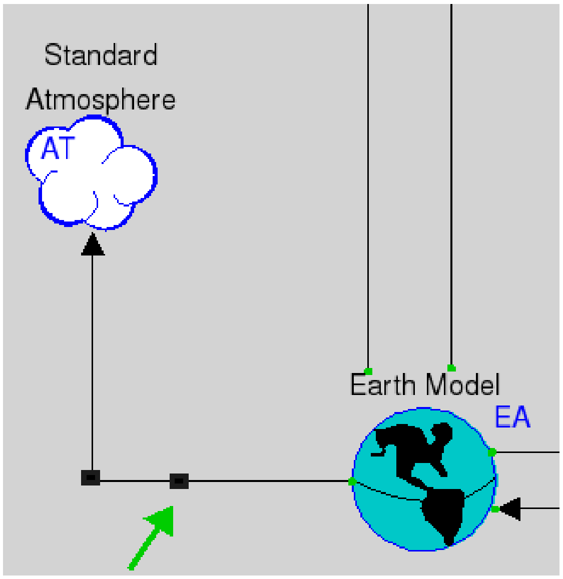


Figure 36 Inserting A Point

For example, to insert an additional connection line point, select a connection line, then right-click and select the menu item “**Insert Point**”. The resulting connection line will include an additional “point” in the connection line, as shown in Figure 36. Each point is identified by a small solid square on the connection line.

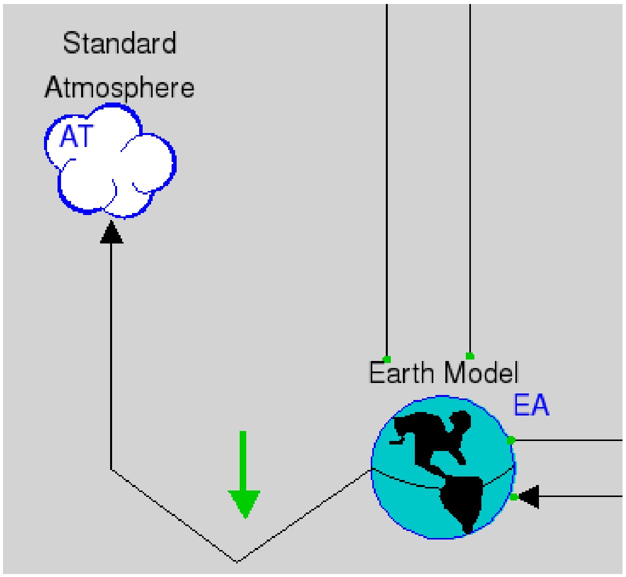


Figure 37 Adding a Corner

Then, simply move the connection line point, to introduce a new “corner”, as shown in Figure 37. In a similar manner, connection lines can be made to approximate contours consisting of multiple small line segments, as shown in Figure 38.

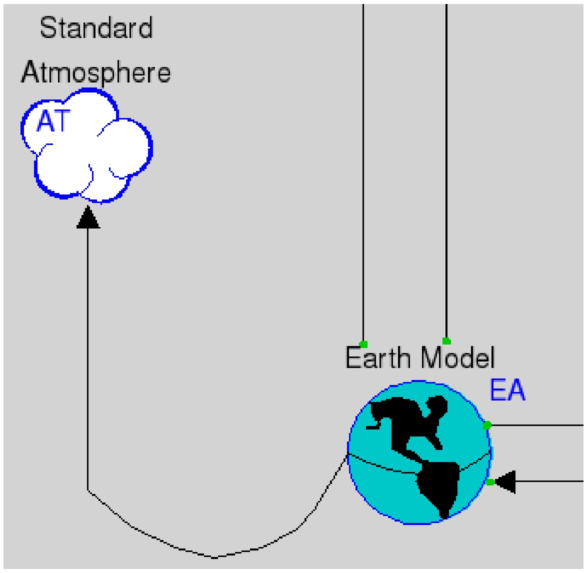


Figure 38 Contour

Defining Connection Line Labels and Attributes

You have seen how labels are automatically attached to connection lines. The label is the name of a single output that is connected. For example, the IN component output variable name is S_ Out_TF and is connected into the TF gain block.

Easy5 automatically applies the connection labels and determines the best attributes, such as position and orientation, for the given connection line configuration. However, you can override the default attributes. Connection attributes are defined and managed on an individual basis. Some example connection line labels are shown in Figure 39.

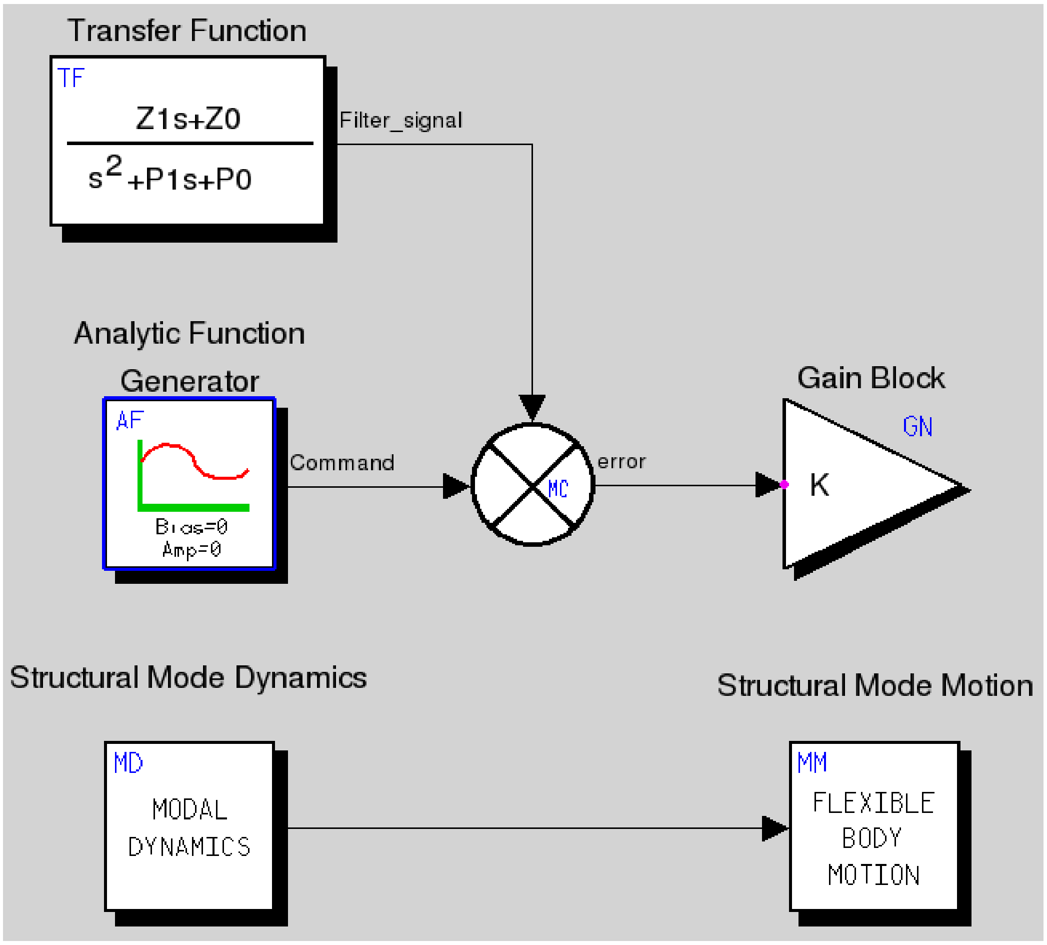


Figure 39 Example of Connection Line Labels

The output of the AF block is the variable named *Command*, and is connected to the summation block MC. This is a single data input/output connection and therefore the label *Command* is applied to the connection line. The TF component has two outputs (*X1* and *S_Out* renamed as *Filter_sgnl*). However, only one output, *Filter_sgnl* is connected to the MC component, and the label is applied to the connection.

Labels can only be applied to single data connections, and cannot be applied to multi-data connections. The connection between the MD and the MM component cannot be labeled. Even though only one connection line is drawn, it is actually a multi-data connection that connects the following three data values:

```
Q_MD  -> Q_MM
QD_MD  -> QD_MM
QDD_MD  -> QDD_MM
```

If you need labels for multi-data connections, you have three methods of adding labels:

1. The best method is to CLICK-R on the connection line, then select Line Attributes. Enter the desired label in the Label String field.
2. Use the icon editor to edit the component icon and add your own labels to the output or input of the component.
3. If possible, make the component connection a "port" connection. Then, port labels will automatically be applied when the connection is made. This is described in the next section.

To change the connection line attribute, CLICK-R to access the connection menu as follows.

1. Right click the IN -> TF connection line to display the Connection menu shown in Figure 40. While holding down the right mouse button, select Line Attributes....

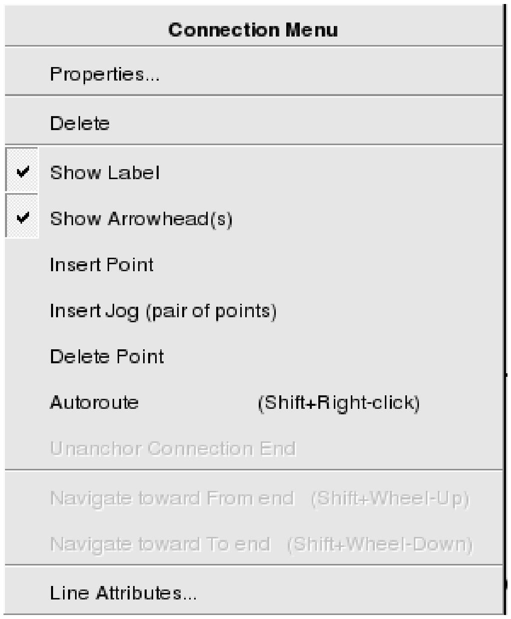
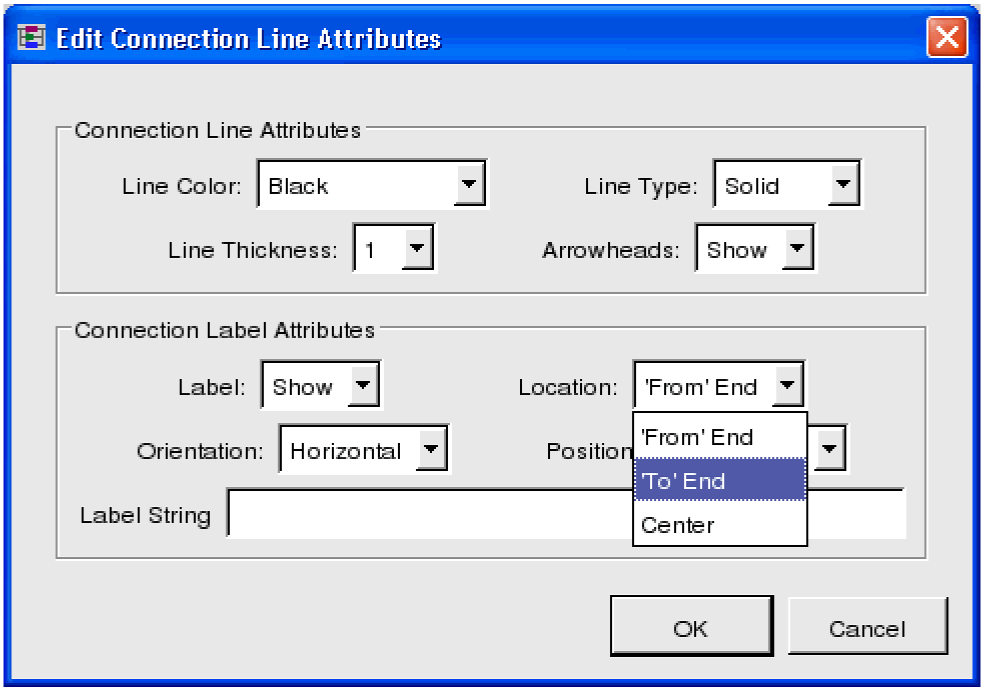

Connection Menu
Properties...
Delete
✓ Show Label
✓ Show Arrowhead(s)
Insert Point
Insert Jog (pair of points)
Delete Point
Autoroute (Shift+Right-click)
Unanchor Connection End
Navigate toward From end (Shift+Wheel-Up)
Navigate toward To end (Shift+Wheel-Down)
Line Attributes...

Figure 40 Connection Menu

2. This opens the window shown in that Figure 41 lets you select whether you want to display the connection line label, and how it is to be displayed, its position, orientation, arrowheads on/off, location, line type, and color.

   The color attribute is applied to both the label text and the connection line. To change the orientation from horizontal to vertical and move the label to the end, click the Vertical radio button in the Orientation box and click the Head End radio button in the Location box.



Edit Connection Line Attributes

Connection Line Attributes

Line Color: Black
Line Type: Solid
Line Thickness: 1
Arrowheads: Show

Connection Label Attributes

Label: Show
Location: 'From' End
Orientation: Horizontal
Position
Label String

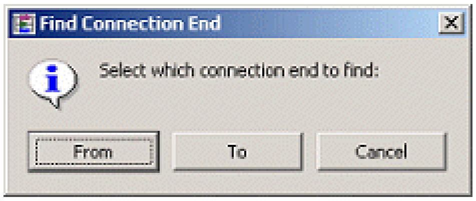
Figure 41 Edit Connection Attributes

3. Click **OK** to close the form. The label is then displayed vertically and at the connection's head end (near the arrow head).

Connection Line Navigation

Connection lines show the flow of data between components. It can be difficult to "navigate" connections lines in large models, especially connection lines that jump down/up several submodel hierarchical levels. To help you follow the path of a connection line, select the connection line with a Ctrl+right-click (or menu item "**Goto End**".

If the connection is into/out of a submodel, it will find and highlight the component that the connection goes to. If the connection is within the same hierarchical level, a dialog displays allowing you to "select which end to find", as shown in Figure 42.



Find Connection End

Select which connection end to find:

Figure 42 Find Connection End Dialog

You have the choice of selecting the [From] or [To] pushbuttons. *From* finds the component from which the connection initiates, *To* finds the component that terminates the connection.

Submodel Connection Labels

Connections into and out of submodels are labeled with submodel labels. The submodel label displays the component name that the submodel is connected to. An example of an opened submodel with connection labels is shown in Figure 43.

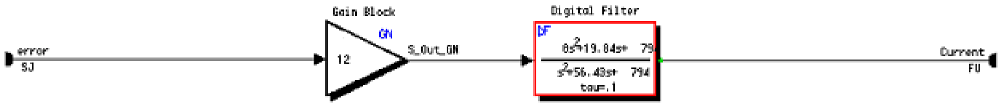


Figure 43 Example of Submodel Connection Labels

The submodel labels are displayed next to the filled in semicircles. For example, the output of the SJ component named *error* is connected into this submodel, to the input of the GN component. The submodel label **SJ** indicates that the connection comes from the SJ component.

Connection Label Options

By default, connection labels are applied when a model is being built. To turn off this options deselect the radio button in the following menu:

Options > Label New Connections

Connection labels can be applied globally to the entire model by selecting:

Edit > Label All Connections

Or, connection labels can be deleted globally by selecting:

Edit > Delete All Connection Labels

Each of the three types of labels can be turned on/off individually by deselecting the following check boxes from the **View** menu:

Show Output Labels

Show Port Labels

Show Submodel Labels

When the check box is NOT visible, the labels are NOT shown. When visible and checked, the labels are visible.

Moving Submodel Connection Nodes

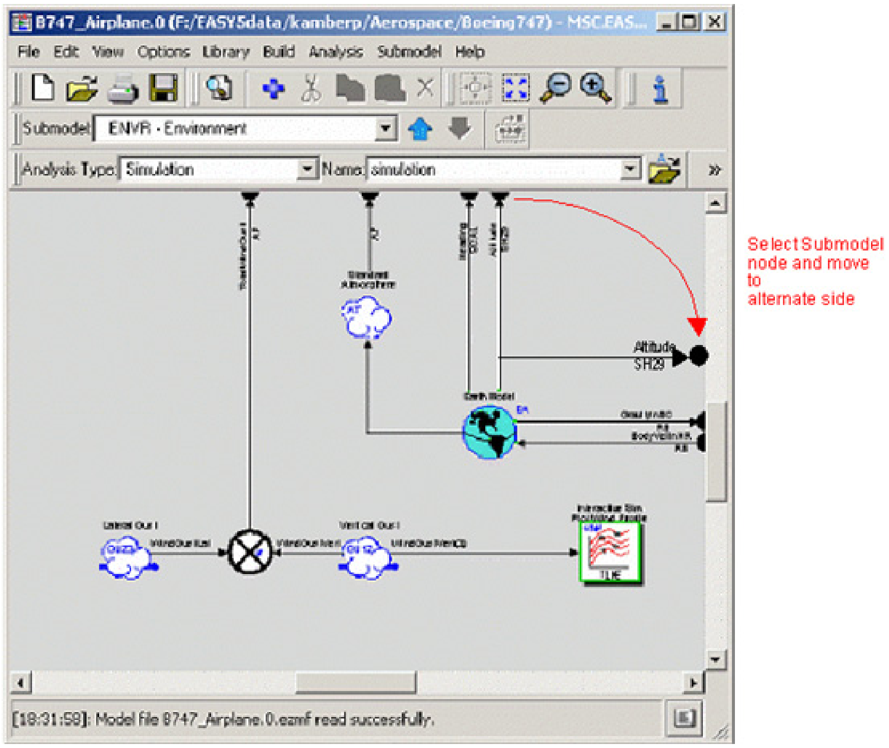
Connection lines that enter/exit submodels go through submodel nodes drawn as filled semicircles, as shown in Figure 45: Example of Moving Submodel Connection Lines. The nodes are used to show which components are connected to the submodel, and how the connection is formed. The component name is displayed with the associated node.

By default, the connection lines and the placement of the submodel nodes are made to correlate the connection at the submodel level with the connection at the higher block diagram level. The nodes are placed on the side of the submodel window to match the same side that the connection is made at the upper block diagram level. For example, if at the top schematic level, a connection enters the submodel **icon** from the left side, then, the submodel will have the node on the left side to force the connection line to enter from the left. This correlation between the submodel connection and the connection at the higher block diagram level is done to maintain a consistent display of the connection lines, making it easier to follow connections into and out of submodels.

You may break this default submodel connection scheme by moving the submodel node. To do this, just select the node with a HOLD-CLICK-L, and drag the node to another edge, as shown in Figure 44.

Note that nodes only reside on the submodel edges; therefore, you may only move the node to a different edge. A green dot is displayed on nodes that are moved.

If you move a submodel node to a new edge, the connection line at the higher block diagram hierarchy will also move to correlate with the same side.


B747_Airplane.0 (F:/EASY5data/kamberp/Aerospace/Boeing747) - MSC.EAS...
File Edit View Options Library Build Analysis Submodel Help
Submodel ENVR - Environment
Analysis Type Simulation
Name simulation
Altitude
SH29
Select Submodel node and move to alternate side

[18:31:58]: Model file B747_Airplane.0.ezmf read successfully.

Figure 44 Example of Moving Submodel Connection Lines

In Figure 44, the submodel node from the top edge is moved to the right edge. As a result, the connection out of the submodel icon will also emanate from the right side. A yellow dot is displayed on the submodel icon to indicate that the underlying submodel connection node was moved from its matching “current view” side of the component.

You can break the connection correlation by moving the connection line endpoint on the submodel icon. By doing this, there is no correlation between the connection into/out of the submodel icon, and the connection within the submodel. A red dot is displayed on the submodel icon to indicate that the submodel connection no longer correlates.

An example of overriding the default submodel connection is shown in Figure 45. In this submodel, connection into the submodel icon was moved from the top of the icon to the bottom.

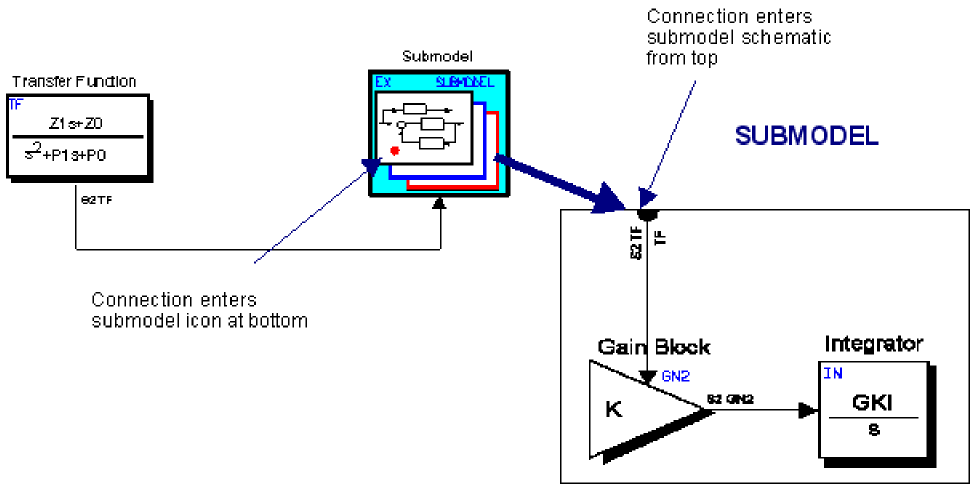


Figure 45 Example of Breaking Submodel Connection Lines

However, if you examine the submodel, the connection within the submodel still shows the connection emanating from the top side. As a result, there is no correlation between the submodel icon and the submodel schematic.

Connection Line Color Dots

Colored “dots” are added to the endpoints of connection lines to indicate the type of forced connection being used. These dots are used as a visual reference only, and are not printed when printing the schematic. A green dot represents a user initiated move, a yellow dot represents a cross submodel move where the connection on the other side of the submodel was the one moved by the user.

A red dot indicates that a connection across a submodel is not being drawn consistently, that is, the correlation between the connection into/out of the submodel icon does not match the connection into/out of the actual submodel boundary.

In summary, the connection line dot colors are used as follows:

1. If a connection has been made with default routing, no dots are used.
2. If a default connection is moved on the side of a component, the dot is green.
3. If a default connection is moved on the submodel icon, the dot on the submodel icon is drawn green and the dot inside the submodel connection boundary is drawn yellow.
4. If a default connection is moved on the inside of a submodel connection boundary, the dot on the inside is drawn green and the dot on the submodel icon is drawn yellow.
5. If a connection across a submodel is broken as a result of a user moving a yellow dot, the just moved yellow dot is changed to red and the corresponding green dot on the submodel icon side is changed to red.
6. If a connection ever becomes broken as a result of a user initiated move where it is not possible to keep the order of connections across a submodel consistent, both dots on either side of the submodel are drawn in red.
7. If a "pin-to-pin" type connection is established (to an icon with a defined port), a pink dot is used to indicate this.

Copying Components and Models

Components can be copied within a model, to another model, or from a different model. When a group of components is copied, all their characteristics, including connections and data contained in the component data tables, are copied with them. This saves time when constructing models that contain repeated component groups. The Model Building license feature is required for any model editing.

Copying Components within a Model

To make a copy of a single component, select it, then either choose the cut and paste icons from the toolbar, or enter the accelerator keys, **Ctrl+C**, then **Ctrl+V**.

The method for copying a group of components within a model is similar to moving groups of components. You copy a group of components using a selection box to capture the group. Then you move the selection box across the schematic to a desired location and enter **Ctrl+P** to copy the components into that location.

The component group can be copied to any part of the main schematic and to any submodel. Easy5 automatically assigns a 2-character component identifier to the new components, avoiding duplicate component names.

To copy a group of components:

1. Press the **Ctrl** key and select the components you want to copy by clicking on each component.
2. Select the **Copy** icon from the toolbar, or press **Ctrl+C**.
3. Place the pointer where you want to copy the component group, then select the **Paste** icon from the toolbar, or press **Ctrl+V**.

Note how the connection arrows between the copied components are also copied. Easy5 automatically renamed the components with unique component identifiers.

4. Before proceeding, delete the group of copied components by selecting them and pressing **Delete**.

Copying Components From or To Another Model

You can copy components from or to another model. This is done by first copying the components into a temporary model, and then opening the destination model and copying the components in from the temporary model.

To copy a group of components from one model into another:

1. Open the model that you want copy components *from*.
2. Press **Ctrl** and select the components you want to copy, or draw a selection box to encapsulate these components.
3. Select **Edit > Save Group As...** from the main menu.
4. In the file dialog, enter a new model name to copy the components into, such as **temp**.
5. Open the model that you want copy components *to*.
6. Select **Edit > Import Model From** from the main menu, and select the model you wish to copy (for example, temp).
7. A selection box with a crosshair in the lower left corner displays.
8. Move this box to any location on the schematic pad where you want to place the copied components, and then release the mouse button to drop the components into place.

Copying Components With User-defined Names

When components are copied, the two character component identifier is automatically changed. The two character identifier is used in the naming convention to define *default* input/output names. This naming convention guarantees that Easy5 defined *default* names are always unique.

However, if the user changes the *default* name(s) to a *user-defined* name(s), when copied, the *user-defined* names must be changed by the user. Fortran component input and output names are also considered to be user-defined names. When one or more components containing user-defined names are copied, Easy5 automatically renames all conflicting names, but pops up a “rename” window to assist the user in changing user-defined names. For information on how to use the rename feature, see “User-Defined Names”.

Data Display

Using schematic text annotation, operating point information can be displayed on the schematic and be tied to a specific component icon or placed anywhere on the schematic using special display icons.
This allows operating point data (states, parameters only) to be displayed on your schematic directly (Figure 46).

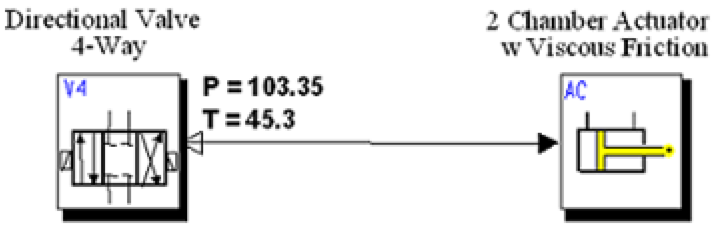


Figure 46 Operating Point Information Display

To create a text note on your schematic, right-click on any empty portion of the schematic to display the pop-up Schmatic Menu.

Select the **Add Text Note** menu item or press the letter T on your keyboard. The words Text Note display on your schematic, where you placed your pointer.

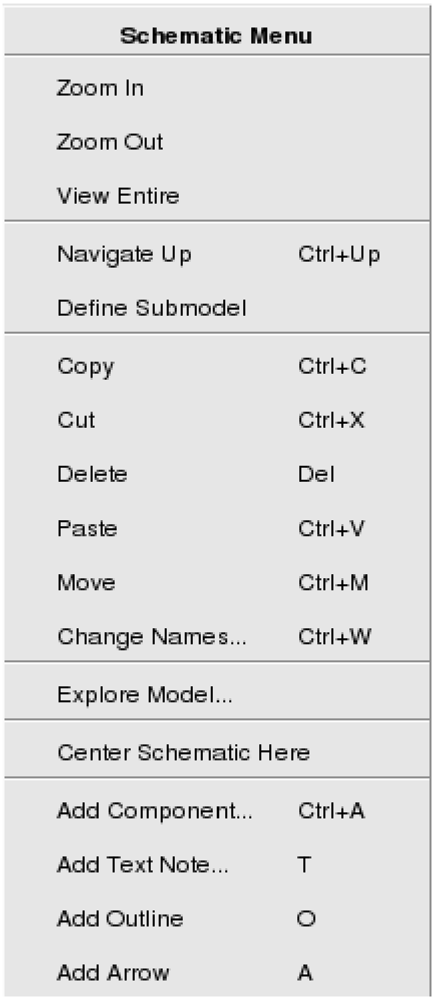
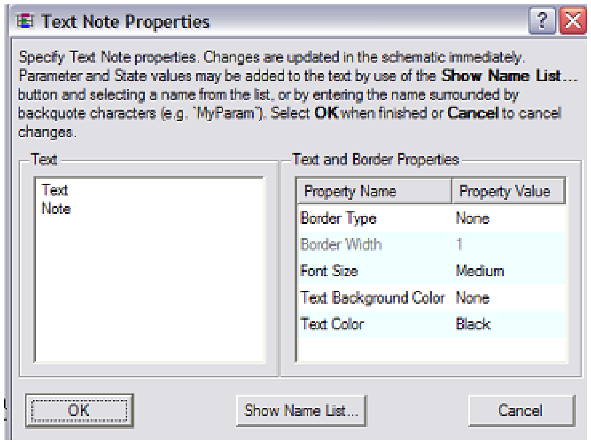


Figure 47 Schematic Menu

To open and define your text note, double left-click on the words Text Note to the Text Note Properties dialog shown in Figure 48.



Text Note Properties

Specify Text Note properties. Changes are updated in the schematic immediately. Parameter and State values may be added to the text by use of the **Show Name List**... button and selecting a name from the list, or by entering the name surrounded by backquote characters (e.g. `MyParam`). Select **OK** when finished or **Cancel** to cancel changes.

Text

Text
Note

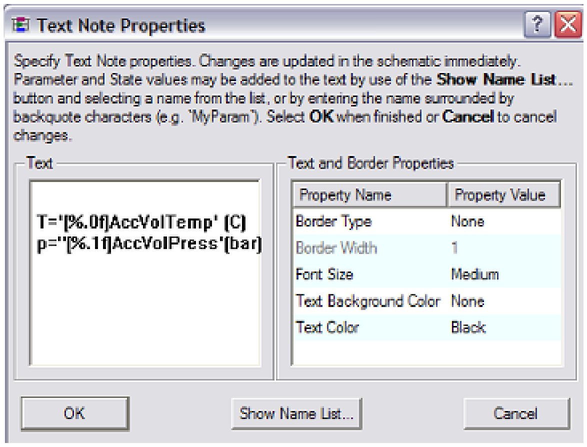
Text and Border Properties

Property Name	Property Value
Border Type	None
Border Width	1
Font Size	Medium
Text Background Color	None
Text Color	Black

Figure 48 Text Note Properties Dialog

Place your cursor in the left text box and write your note. To include actual data from your model, use the "**Show Name List**" button to select any model quantity available to include in a Text Note. Model data is represented by a model name enclosed by single quotes (as '<*name*>'). Additional text can be used to provide name, units, etc., such as "P='p_Out_PI23' (bar)".

Data formatting instructions can include any valid C-format string enclosed in square brackets, just after the first quote of the name, in the form: "'[%<*format*>]<*name*>'. Other formatting can also made to modify the text color, background, font size, etc. For example, consider the following text note dialog, used to list the values of some model quantities from a hydraulics model.



Text Note Properties

Specify Text Note properties. Changes are updated in the schematic immediately. Parameter and State values may be added to the text by use of the **Show Name List**... button and selecting a name from the list, or by entering the name surrounded by backquote characters (e.g. `MyParam`). Select **OK** when finished or **Cancel** to cancel changes.

Text

```
T='[%.0f]AccVolTemp' (C)
p=''[%.1f]AccVolPress'(bar)
```

Text and Border Properties

Property Name	Property Value
Border Type	None
Border Width	1
Font Size	Medium
Text Background Color	None
Text Color	Black

Figure 49 Text Note with Data Values

The resulting output on the schematic would appear similar to the values shown to the left of the icon shown in Figure 50.

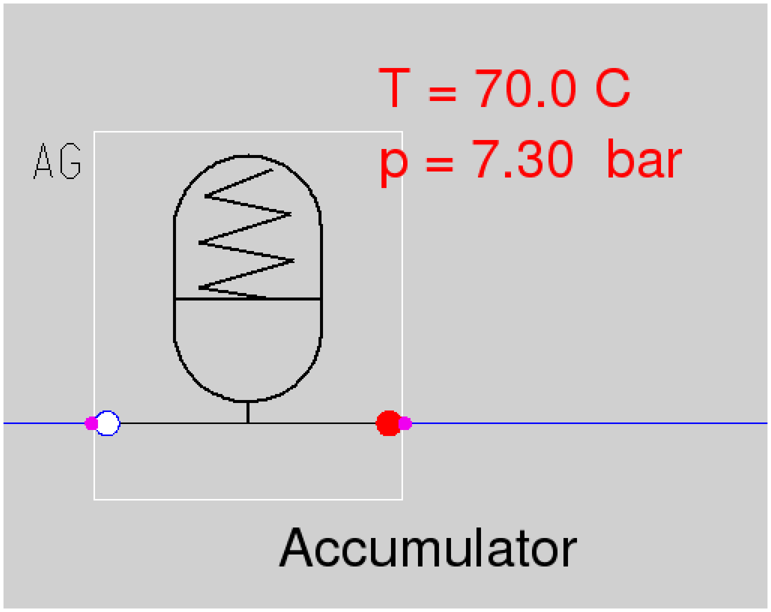


Figure 50 Output Values Displayed on Schematic

Note that any valid C formatting instructions can be used, but if an error occurs no data will appear. It is up to the user to correctly specify C-formatting instructions.

Some commonly used C-format constructs are:

%E	exponential notation
%.1f	floating point number, show one digit to the right of the decimal point
%.0f	floating point number, show no digit to the right of the decimal point

Note:	Any model quantity (at a given operating point) can be displayed using a text note. However, if the model quantity is a "variable", then the value can only be loaded by importing an operating point file generated either by the frontend, or as stored in the model file itself. It is possible to generate operating point data (.ezic) files that do not contain data for model "variables". When this happens, the value will be displayed as "unavailable".

Data Types

See also: "Parameters - Defining Input Values"

"States"

All data inputs and outputs of an Easy5 components fall into four distinct classifications: states, variables, parameters, and tables. The *PU Pump* component from the hydraulics library is a good example of a component that uses all four types of data. The following paragraphs describes these four types of data.

States

States are variable output quantities in the system model that are functions of first order differential or difference equations. For example, a state variable occurs when position is calculated by using an integrator to integrate velocity data. The output of the integrator (position) is a state. There are four types of state variables: continuous, delay, sample (also called sample and hold states), and switch states. These states are used in many standard components and can be added to any User Code and Library components you write. For a complete discussion of states, see the first level section "States".

Variables

Variables are output quantities in the system model that are a function of algebraic relationships. Simply put, variables are defined by algebraic equations, and may be a function of states, variables, parameters or table data.

Parameters

A parameter is an input to a component that is not connected to the output of another component. You determine what will and what will not be a parameter when connecting the components in your model. Parameters are assigned constant values before performing an analysis. These constants are derived from data associated with the system you are modeling.

For example, a component that models gain has in input parameter called "K", used to set the gain. You define the constant gain parameter before performing an analysis. For more information, see "Parameters - Defining Input Values".

Tables

A table is a set of constant tabular data used by any of several components that produce outputs using a table look up algorithm. In general, tables are used to represent algebraic functional relationships, containing from one to nine independent variables, and they allow you to include a set of "real world" data into your model. You can also design User Code and Library components which use table look up algorithms.

Table data is input and edited using the Matrix Editor. For more information on how to use tables, see the User Guide, Chapter 5 - Code Components, the section "Defining a User Code Variable".

Tabular data can be generated using Excel, and imported into an Easy5 table using Easy5's Matrix Editor. For more information on how to import Excel data into an Easy5 table, see the User Guide, Chapter 8 - Table and Matrix Editor, the section "Importing and Exporting Data".

Debugging the Model and Analysis

Debugging mostly makes sense for the following Easy5 analyses: *Simulation, Single Call, and Initial Condition Calculation*. To run the analysis in the debug mode, select the [Debug Mode] toggle in the Build menu. This will cause the model executable to be generated for debug purposes, allowing you to use an interactive symbolic debugger at analysis runtime.

When you run an analysis in debug mode, you can set a breakpoint in subroutine EQMO (the Fortran subroutine that represents your model), and then step through the code and monitor execution and print intermediate results.

You can also set variables to different values if the results of a calculation are incorrect. Since there is *significant* variation in the commands for the symbolic debugger between the different platforms, *please consult the reference manual (or man page) for your platform*. An example of using a Windows debugger is given in the following section, followed by a section describing a Sun/Solaris debugger.

Example of Using the Symbolic Debugger on Windows

The following example shows you how to use Compaq's Visual Fortran symbolic debugger running on Windows. This is example is not a tutorial and does not describe all the functions available. See the appropriate Fortran or Microsoft C++ compiler manual for more information.

1. Select the **Debug Mode** from the **Build** menu. This causes your model to be compiled and linked with debug options.
2. Select the “**Debug the current analysis**” button from the respective Easy5 analysis data form or toolbar. Invoking the debug option causes the MSDEV program to invoke your model executable in the debug environment. Selecting the “**Execute this analysis**” button will execute the analysis in non-debug mode.

   As the **Microsoft Visual Studio** debug environment loads you should see a related splash screen come up, followed by a window titled “Microsoft Visual Studio” followed by your model executable name.

The only source code files available for symbolic debugging are:

- **Your Fortran model executable** - this is the subroutine created by Easy5 which comprises all your components and submodels. This file is named <*modelname*>.f.
- **Any C-Component code** - if you are using any C-Components in your model, all code is combined into one source file named <*modelname*>_c.c
- **Your external routines or libraries** - these must be compiled with the debug option previously and linked in via The "**Link External Object**” Build menu item. For more information, refer to "Compiling External Code", and for linking, see "Linking External Code".

Enabling the Debugger Toolbar

If you do not already have the Debugger Toolbar visible, select the **View/Toolbars...** menu item and ensure that the “Debug” item is selected. This makes debugging operations (e.g. **Go, Step,** etc.) available via toolbar buttons.

Opening Source Files

Easy5 analysis and component libraries are not available for debugging purposes, that is, there is no source code available for these routines for symbolic debugging. You need to open your model's source file. Select the **File >Open** menus, and select the <*model name*>`.f` file. This will open the model's source file and display it in the debugger window. If you have additional external code that is called and linked into your model, and you would like to debug these files, then you must also open these source files. You are ready to create a breakpoint and begin to debug.

Creating a BreakPoint

To create a breakpoint, you must open the source file as instructed above. Scroll down in the file and simply select an executable line of your model code where a breakpoint should be placed. Use the "Insert Breakpoint" (debug) button to set a breakpoint. You can also toggle breakpoints on and off by repeatedly selecting this button.

A breakpoint is indicated by a filled (maroon) circle in the first column. You can place a breakpoint on any *executable* line of source code. Usually only a handful are used to start you stepping through a portion of your model executable source code.

Breakpoints can be centrally managed (e.g. selectively temporarily disabled) via the menu item **Edit/Breakpoints....**

Caution:	We recommend that you *not* enter the routine name using the Breakpoint menu item. Rather, open the file and select breakpoints with your mouse. Why? Selecting via your mouse is a more reliable method. There is less chance of specifying an incorrect breakpoint.

Activating Your Symbolic Debugging Session

To begin your symbolic debugging session you must first *activate* it. Either from the **Go** button, or via the **F5** function key. Once your debugging session is activated, a new menu called "Debug" displays on your menu bar. The "Debug" replaces the "Build" menu. You may also note that a background shell temporarily is displayed on your screen. This is simply your background job which executes your Easy5 model executable.

Stepping Through Your Code

To step through your code, use the buttons provided for this, the Debug menu items, or keyboard shortcuts. You can either **Step into, Step over, Run to cursor**, or **Step out**.

Starting, Restarting, Stopping

To initiate your model executable, you must select the **Go** button, or menu item. The **Restart** capability allows you to start from the beginning again, if you passed your area of interest already. To stop, use the **Stop Debugging** button or appropriate Debug menu item.

Examining a Variable

There are a variety of ways to display values of variables in your code. Perhaps the easiest is simply to ”hover” your mouse over a FORTRAN variable. Its value should be automatically displayed. Other display options are available. See the MS Developer Studio help for more information.

Determining Where the Model is Aborting

Other platforms may display termination information, called a “traceback”, directly in the Easy5 Analysis Program Listing (APL) file. A traceback provides vital information about which error occurred and, most importantly, at what line number in which routine. While this information cannot be provided during normal execution under Windows platforms, fortunately you can obtain similar information by running the debugger. However, you must first make a couple of changes to your debug environment.

Specifically, select the **Debug/Exceptions...** menu item to display a list of all possible exception conditions, and what action will be taken if such a condition occurs. In most cases, we are interested in floating point exceptions. Such exceptions begin with the word “Float”. By default, such exceptions are handled by Easy5’s error handler. This is something we don’t want to use during a debug session, so that we can see at which point the error occurs.

Therefore, we want to set all floating point exceptions to an Action of “Stop Always”. You will need to select each exception (you can hold down the Shift key to select more than one exception), select “Stop Always”, and then the “Change” button, followed by the OK button. Now, you are ready to let your model terminate.

Once, you have modified your exceptions, simply select the Go button. The debugger will stop at any breakpoints you have set. Please remember that your model may be called many times until the termination error you are trying to find occurs. For this reason, you may want to temporarily disable all breakpoints and let the debugger execute until any floating point error occurs.

When it does, the symbolic debugger should stop at the line of code where the error occurred. If this line of code is in your model (or an external source code routine you have provided and compiled in debug mode) you should be able to see the line where the exception occurred. Otherwise, a Disassembly window may appear (see below). Nevertheless, if you examine your **Call Stack** window you should see the line number at which the debugger has stopped. The Call Stack window should be available by default, otherwise; it can be activated via the Call Stack debugger toggle button.

Modifying a Variable

Something which can be quite useful in debugging your code is to change the value of a variable from the debugger (say, to force an alternate logic path). This is also easily done via the mouse: select the variable name by highlighting it, then drag and drop it to the **Watch** window. Here, you can simply change the value.

Disassembly View

Often, when you inadvertently step into a routine for which no debugging information exists, a so called “Disassembly” window automatically displays. This is a view of the assembly level code, which for most of us is pretty useless. If you want to see the “step by step” assembly code commands being processed you are in the right place. To return to the source window, select the “Step Out” function (to jump back out of the current routine) if needed, and select the “Disassembly” button to toggle the window off again. This should return to your source code window.

Debugger Help

We urge you to familiarize yourself with the symbolic debugging environment via the online help provided. Unfortunately, we cannot provide detailed information about using this symbolic debugger within the scope of this manual.

Example of Using the Symbolic Debugger on a Linux Platform

The following example shows you how to use the symbolic debugger on the Sun workstation. This is example is not a tutorial and does not describe all the functions available. See the "*Debugging Tools for the Sun Workstation*" manual for more information. This example is general enough to apply to other platforms. However, the commands and setup are different for each platform.

In this example, a floating point exception was purposely added to the model. Essentially, a divide-by-zero was added to the code. The model is first built using the debug option. Then, the Initial Condition Analysis data form was opened, and the analysis was executed selecting the [`Debug this analysis`] toolbar icon. The analysis is launched and is automatically put into the debug mode and the debugger window displays.

Note:	The symbolic debugger is optional software purchased with the compiler. It is **not** provided with Easy5, nor is it licensed by Easy5.

The Sun debugger has two window panes. The upper pane is the output window, which displays the code. The lower window pane is the input window. The debugger commands are input into this window.

When first opened, you may get warning messages in the lower window, such as:

Warning: skipping debugging info in "lm_checkout.o"

Warning: can't find source...

These warning messages are typical and can be ignored.

You want the debugger to run and put you into the model source code file (model_name.f), which contains the main routine EQMO. So, tell the debugger to stop in the routine EQMO, then run the debugger.

Enter:
```
stop in eqmo
run
```

This runs through the code until it comes to the subroutine EQMO where it stops.

This is shown in Figure 51 where the code "stopped' at the first executable line of the EQMO routine.

You can now scroll up and down, select different lines of code that you wish to stop at, and add a "STOP" flag by selecting the "Stop At" pushbutton. You can also step through the code line-by-line by entering the following commands:

`step <n>` performs a single step or n steps if specified, and steps into calls

`next <n>` same as the step command, but it skips over calls

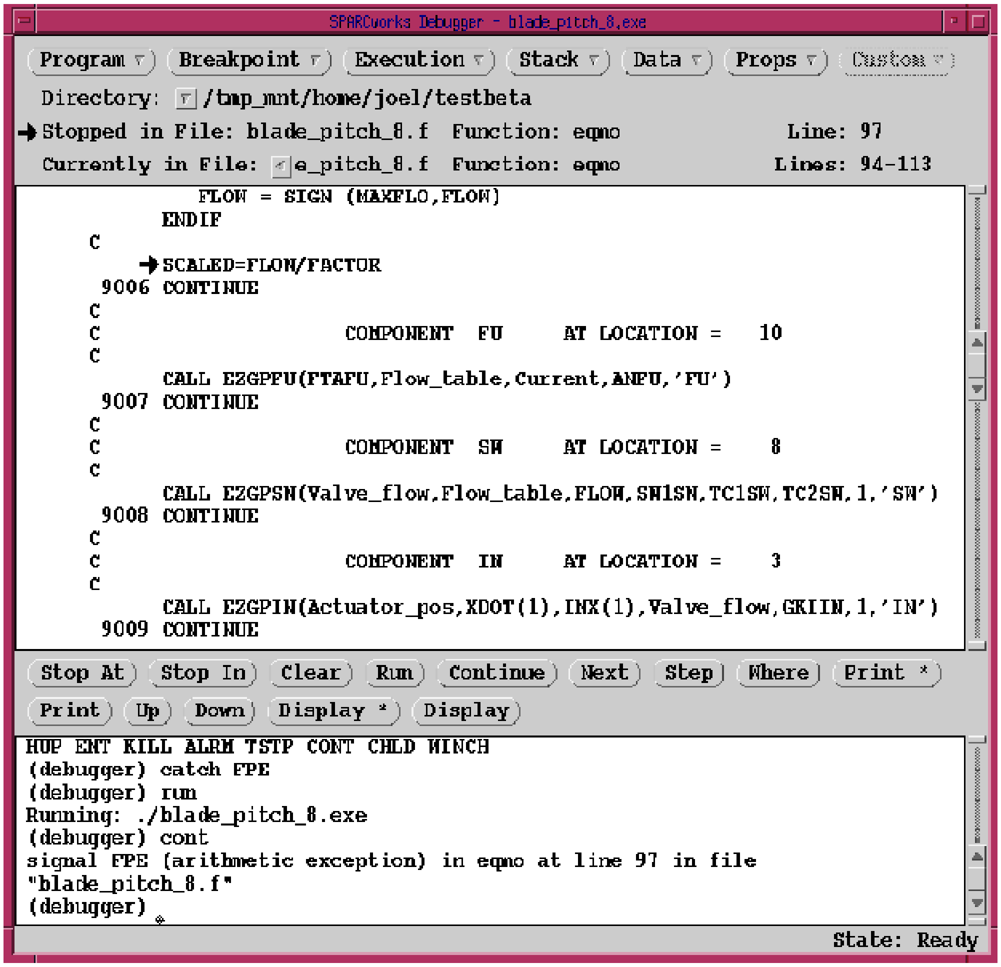
At any point, you may print out a variable value by entering the print command:

`print var`	where var is the variable name

If you wish to just run through the code without stepping line by line, enter the run command. In this example, the debugger encountered an abnormal exit, but displayed the message in hex code. The "floating point error" (FPE) condition was not set. So, the FPE "catch" condition was turned on by entering the command:

`catch FPE`	this turns on the floating point catch
`cont`	this continues the run

The FPE setting allows the debugger to catch the floating point error.



```
Directory: /tmp_mnt/home/joel/testbeta
Stopped in File: blade_pitch_8.f  Function: eqmo          Line: 97
Currently in File: e_pitch_8.f  Function: eqmo          Lines: 94-113
```

```
              FLOW = SIGN (MAXFLO,FLOW)
           ENDIF
      C
           SCALED=FLOW/FACTOR
 9006  CONTINUE
      C
      C                    COMPONENT  FU     AT LOCATION =    10
      C
           CALL EZGPFU(FTAFU,Flow_table,Current,ANFU,'FU')
 9007  CONTINUE
      C
      C                    COMPONENT  SW     AT LOCATION =     8
      C
           CALL EZGPSW(Valve_flow,Flow_table,FLOW,SW1SW,TC1SW,TC2SW,1,'SW')
 9008  CONTINUE
      C
      C                    COMPONENT  IN     AT LOCATION =     3
      C
           CALL EZGPIN(Actuator_pos,XDOT(1),INX(1),Valve_flow,GKIIN,1,'IN')
 9009  CONTINUE
```

```
HUP EMT KILL ALRM TSTP CONT CHLD WINCH
(debugger) catch FPE
(debugger) run
Running: ./blade_pitch_8.exe
(debugger) cont
signal FPE (arithmetic exception) in eqmo at line 97 in file
"blade_pitch_8.f"
(debugger)
```

State: Ready

Figure 51 Example of a Debugger Catching a Floating Point Error

In Figure 51, the debugger “points” to the line of code that triggers the FPE condition (SCALED=FLOW/FACTOR). The SCALED calculation fails because FACTOR has a value of zero which causes a floating point zero. The debugger also print the message:

```
signal FPE (arithmetic exception) in eqmo at line 97
```

This is a simple example that only applies to using the Sun debugger. Other debuggers are similar, but have different commands. For help, consult the appropriate documentation, or online, enter the `help` command in the debugger window.

Defining the Debugger Tool

There may be more than one symbolic debugger available on your platform. If you wish to use a different symbolic debugger, put a link in your current directory whose link name is that of the current Easy5 default and whose target is the location of the new symbolic debugger.

For example, if the Easy5 default is *dbx* and you want to use one called *codeview*, then define the variable WSDEBUGGERNAME as follows:

```
setenv WSDEBUGGERNAME codeview                  {C-shell}
set WSDEBUGGERNAME=codeview                     {Korn shell}
```

Now when you run Easy5 in the debug mode you will get *codeview* instead of *dbx*.

To find out which symbolic debugger you are using, at a command prompt, enter the command:

```
easy5x -varset
```

and look for the name WSDEBUGGERNAME. This is the symbolic debugger being used by Easy5.

A list of the symbolic debugger command names for different Linux platforms is as follows:

HPUX: xdb

Sun: “workshop -D”

IBM: dbx

Linux: gdb

Deleting Components and Connections

Easy5 allows you to delete components and connections individually or as a group. The methods for doing this are different, as described in the following sections.

See also: "Adding Components"
"Components"

Deleting Components

To delete a single component:

1. Select the **GN2** component and press **Delete**.
2. Click the **OK** button in the Confirm Deletion dialog to delete the component.
3. Before proceeding, add the GN2 component back into your model.

To delete a *group* of components:

1. Press **Shift**, and then press and hold the left mouse button and drag it to draw a box to encapsulate one or more components. Use this method to select the **GN** and **DF** components.
2. Release the mouse button. The components darken indicating that they have been selected as shown in Figure 52.

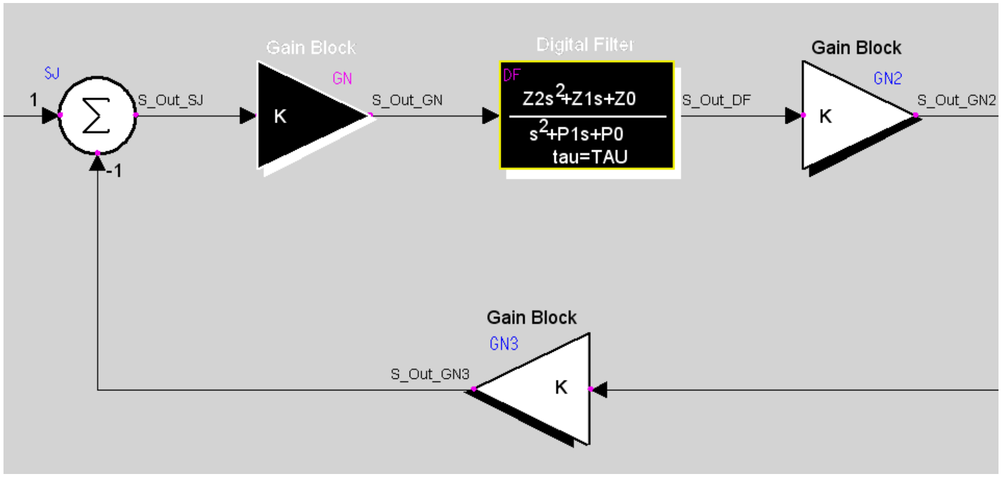


Figure 52 Example of Using a Selection Box to Capture Components

3. Press **Delete**.
4. Press **Cancel** in the Confirm Deletion dialog as you do *not* want to delete these components.

Deleting Connections

In addition to making connections between components in a schematic, you need to know how to delete unwanted connections. Connections established between two components can be deleted using two different methods:

- All connections between components can be deleted at once.
- Individual connections between two components can be deleted one at a time.

Deleting All Data Connections

A single connection line can contain one or more data connections. To delete all data connections between components, select the component connection line and press **Delete**.

When the connection is deleted, the IN to TF connection arrow in your schematic disappears. This indicates that all the data connections associated with the connection arrow between IN and TF have been deleted from the schematic. Before proceeding, reconnect the IN and TF components by selecting the IN component and then the TF component.

Deleting Individual Data Connections

A single connection line may contain multiple data connections. These individual data connections between components can be deleted by using the connection data table. For example, use the following steps to delete the connection between GN2 and SJ.

1. Double click on the connection arrow between GN2 and SJ to open the connection data table. The connection is shown in the right windowpane under Individual Connections.
2. Select **S_Out_GN2 -> C1_SJ**, and then click **Delete Connection**.
3. Click **OK** to close the table and apply the changes.

Discrete (Digital) System Analysis

See also:"Discrete (Digital) System Modeling"

"Transfer Function of Sampled Data Systems"

Ap. C: Discrete Analysis Techniques

For complete information about the analysis techniques Easy5 uses on discrete systems, refer to Appendix C. In general, when building discrete models and performing analyses on discrete systems, you do not have to be concerned with the complexities of how discrete systems are built and analyzed. Easy5 takes care of this for you.

The exception to this is with transfer function analyses of discrete systems. This is covered in detail in "Transfer Function of Sampled Data Systems". The following sections describe some general considerations for performing discrete system analysis.

Operating Point Considerations

Normally, an Easy5 operating point is defined purely by the value of all initial conditions for states in your model. For sampled-data systems, this can be slightly different, if the time at which the operating point is saved is a non-fundamental sampling rate interval of simulated time. In such a case, Easy5 additionally saves the values of the rates of all active digital states in the operating point file. This is done to allow you to seamlessly stop and restart sampled-data simulations using the familiar paradigm of an Easy5 operating point as a valid starting point.

Linear Analysis Considerations

Both time domain analyses (simulations) and the linearized stability analyses (z-plane) may be performed with sampled data discrete systems. Every discrete component has an input parameter named "*TAU*" that defines the sampling rate.

Multi-rate digital models can be build using up to ten sampling rates. For performing linear analyses, the sampling rates must be integer related; this is not required for performing simulations.

Easy5 does some pre-checking and management of digital and hybrid models. For example, if you set the sample times such that they are **not** integer related multiple sampling rates and you perform a linear analysis, the following error message will be given in the Analysis Listing File:

```
*** FATAL ERROR *** TO ANALYZE MULTI-RATE DISCRETE SYSTEMS, EACH LARGER
   SAMPLE PERIOD MUST BE AN INTEGER MULTIPLE OF ALL LOWER SAMPLE PERIODS.
   SAMPLE PERIOD <value> IS NOT AN INTEGER MULTIPLE OF SAMPLE PERIOD <value>
```

To resolve this fatal error, you must set the sample rates (TAU) such that they are integer related.

Integration Method Considerations

If you build a model that contains only discrete components, when performing a simulation, Easy5 will give you a "notice" that your model only contains discrete components and the integration method will be changed by Easy5 to the Euler integration method.

The Euler method is the most efficient integration method for purely discrete models. However, if you do not want the Euler method automatically applied by Easy5, then add an IN Integration component to your model, and freeze this state. This will trick Easy5 into thinking that the model contains a continuous state. Thus any integration method you specify will be used. In addition to this, for multi-rate sampling systems, Easy5 will change the integration "Time Increment" value to match the nearest multiple of the fastest sampling rate.

For example, if you have a multi-rate sampling system with .01, .05 and .20 sampling times and you set the "Time Increment" field to a value of .015 in the Simulation Analysis Data Form, Easy5 automatically resets this value to .01 and warns you of the Analysis Listing File change with the following message:

```
*** NOTICE ***   SIMULATION TIME INCREMENT TINC OR TINC2 IS INCOMPATIBLE WITH SAMPLE
PERIOD DIVISOR TAU0.
PRINT/PLOT PARAMETERS AND TINC/TINC2 HAVE BEEN REDEFINED FOR THIS SIMULATION AS
FOLLOWS:        {... list of new values ...}
```

The new simulation parameters will be listed. In this example, the TINC parameter will automatically be reset to.01.

Discrete (Digital) System Modeling

See also	"Discrete (Digital) System Analysis"
	Ap. C: Discrete Analysis Techniques

The Easy5 General Purpose Library contains several components used to model sampled data effects, including several forms of digital filters, a pure delay, and a zero order sample and hold. These components can be cascaded to form higher order digital filters, or used in conjunction with user-defined Fortran and LIbrary components to create a customized form of a digital filter.

In this document, the terms discrete and digital are used interchangeably. Discrete usually refers to a discrete states and mathematical algorithms, whereas digital usually refers to hardware systems such as filters and A-to-D converters. In this context, components may be referred to as either discrete or digital, whereas the state is referred to as a discrete state.

Digital Models

Discrete components can be connected into any part of your model. **All discrete components have a zero order sample and hold dynamic element placed on the output signal**, so that they can "communicate" with any other component in your model. When connecting one to another discrete component with the same sampling rate, this sample and hold will "drop out" without introducing any delay. Both time domain analyses and the linearized stability analyses (z-plane) may be performed with sampled data systems.

Every discrete component has an input parameter that defines the sampling rate. Micturate digital models can be built using up to ten sampling rates. Most important, *submodels can contain components sampled at* *different rates!* Unlike other similar software tools, there is no requirement for placing all digital components sampling at one rate into one submodel. For performing linear analyses, the sampling rates must be multiple integer related; this is not required for performing simulations.

Hybrid Models

A hybrid model is a model that contains both analog (continuous) and digital (discrete) systems. Discrete components can be connected to any part of your continuous plant model by simply placing the appropriate component into your schematic and making the connection. Also, any given submodel can contain components sampled at different rates.

Note: Easy5 automatically inserts a zero-order sample-and-hold between continuous and discrete components, at the sampling frequency set by the discrete component. For added clarity, the user can add a zero-order sample-and-hold component called “SH.”

Figure 52 gives an example of a hybrid system model using continuous and discrete components.

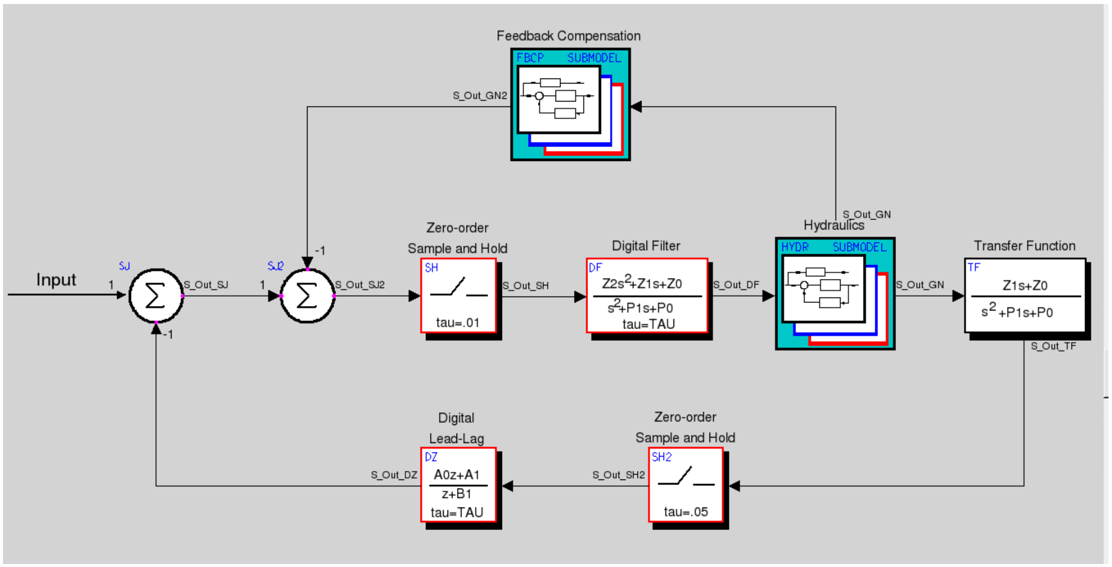


Figure 53 Example of a Hybrid System Model

A discrete component can also be vectorized, but can only have one sampling rate associated with it. Within the model, however, up to ten integer related sampling rates can be included. Sampling rates are always defined by the physical quantity name TAU.

Note: TAU is an Easy5 reserved word. You may not name any component input or output name starting with TAU.

Discrete System Modeling Using Fortran, C and LIbrary Components

When modeling digital systems using Fortran, C, or LIbrary components, additional concerns generally include the following:

1. Ensuring the component models the digital behavior correctly.
2. Maximizing the computational efficiency of the component.

Before continuing the explanation of discrete system modeling, it is important to review some of the terms which will be used in describing this method of modeling.

Delay State Declaration Statement

For every delay state that you define in a code component, you must also include a corresponding delay state declaration statement in the code body. This declaration statement has the following format:

```
NEXT VALUE OF, state name = mathematical expression
```

The state name is the name of the state as it is displayed in the data table.

In addition to the `NEXT VALUE OF` command, you must also define the sample period of the delay state. This is explained in "Setting the Sample Period".

Sample and Hold State Declaration Statement

For every sample state that you define in a code component output section, you must also include a corresponding sample state declaration statement in the code body. This declaration statement has the following format:

```
VALUE OF, state name = mathematical expression
```

The state name is the name of the state as it is displayed in the data table.

In addition to the `VALUE OF` command, you must also define the sample period of the sample and hold state. This is explained in "Setting the Sample Period".

Setting the Sample Period

There are two methods of setting the sample period. The first, and perhaps easiest method is as follows. In any Fortran, C, or Library component with a delay or sample state, you specify the sample period of the state using the `SAMPLE PERIOD OF` command.

For example, if a delay or a sample state is named “xxx”, and the sample period of that state is named “yyy”, to set the sample period, enter the following command before the occurrence of the “value of” or “next value of” statements for that state:

```
SAMPLE PERIOD OF, xxx = yyy
```

In this example, the name of the sample period yyy must be added to the Component Data Table as an input parameter, and the state xxx must be added as a state variable. The sample period input can be defined as a constant, or driven by the output of another component.

Matching TAU Method (obsolete)

An obsolete method (to explicitly defining a sample period using the `SAMPLE PERIOD OF` definition) is to simply define a "matching" TAU input to the code component to the corresponding state name.

Note: This obsolete method is provided only for the reason of backward compatibility.

The matching TAU input parameter must be named such that when you replace the first 3 characters of the state name with TAU, that is the "matching TAU input" name. As a parameter, the TAU parameter can be driven by the output of another component, or set to constant value.

For example, if a delay state is named `DelCont02`, then the associated matching TAU parameter should be named `TAUCont02`.

Note: More than one discrete state can be sampled at the same sample period `TAU`. To set this up using the matching name method, name the corresponding discrete states such that they have the same name expect for the first 3 characters. For example, if three discrete states are sampled at the same sample period `TAU1MIL`, then name all three states with `1MIL` as the last four characters (e.g., `STA1MIL, STB1MIL, STC1MIL`).

Discrete Code Example

Whereas an analog equation is executed continuously with time, a digital equation, by definition, is executed only at appropriate sampling times. Sampling times are defined as "official" calls to the model equations occurring at integer multiples of the "lowest common divisor of sampling periods", or smallest sampling period (which Easy5 calls TAU0).

If we define a User Code (or LIbrary) component which represents discrete, or sampled, behavior, we might conclude that all equations in that component have been automatically "marked" by Easy5 as being digital. Unfortunately, this is not the case.

The only equations that Easy5 considers to be digital, that is, executed only at appropriate sampling times, are those which have been declared as discrete states. These equations, called discrete state declarations, are those prefaced by the Easy5 commands NEXT VALUE OF or VALUE OF. All other equations are executed continually.

A typical digital component contains one or more digital state declarations, and additional code or logic used to define the derivatives for these declarations. However, to prevent any confusion, let us refer to any equations or logic used to define the rates of digital states as digital setup equations.

Thus, a typical digital Fortran component would have the following structure:

```
C   States names:
C          sample state: Position
C          delay state: varx
C   Parameter names for sample times (TAU):
C          sample state tau: Pos_tau
C          delay state tau: delay_tau
...
    (digital setup equations)
...
SAMPLE PERIOD OF, Position = Pos_tau
SAMPLE PERIOD OF, varx = delay_tau
VALUE OF, Position = <derivative_expression>
NEXT VALUE OF, varx = <derivative_expression>
```

The correctness of your digital components (and their interface with the rest of your model) is always preserved through the use of Easy5 delay and sample-and-hold states. These states, which are updated only at appropriate sampling times, ignore the digital setup equations (used to calculate their "rates") at all times other than sampling times.

This leads us to two important conclusions:

1. Only state variables may be used in implicit or recursive equations. That is, the only "saved" values (variables defined using values calculated during previous calls to the model) allowed on the right-hand side of an equals (=) sign are state variables. If this rule is not followed, you will most likely get incorrect results.
2. For purely digital User Code (or Library) components, calls to the model at non sampling times cause superfluous execution of the digital setup equations. However, because these equations are used only to define the rates of the digital states (and are thus not used at non sampling times), correctness is still guaranteed. Therefore, this is only a factor if you are interested in maximizing the efficiency of your code.

Skipping Execution of Digital Equations at Non Sampling Times

This section is devoted to methods of skipping execution of the digital setup equations at non sampling times. If you have a considerable block of these equations, the following information may be of interest to you. If, on the other hand, you only have a few lines of code, you can skip the remainder of this section.

First, you must weigh the impact of adding one or two IF tests to your component against the extraneous execution of digital setup code. Obviously, if you have only a handful of equations in your component, the additional IF tests required to jump around these digital setup equations for non sampling intervals would not result in a good execution time improvement. Let us assume that you have a large block of digital setup code and you want to jump around it for non sampling times. For single rate sampled data systems this simply involves adding a test on the Easy5 global variable IDELAY. It has values as indicated below:

IDELAY	Type of Model Call Indicated
0	For non sample times
1	For "official" sample times
-1, -2	For "test" sample times (used to calculate discrete linear model matrices)

Simply add a test using IDELAY which jumps around your digital setup code for non sampling times (IDELAY = 0). Please note that you must allow execution for “test” sample period model calls (IDELAY < 0), as these are used for discrete linear model calculations. This is in fact exactly what occurs internally for the Easy5 digital state declarations.

For example, you could use the following code in a Fortran component:

```
C   ADD PARAMETERS = TAUONE,...
C   ADD STATES = SAMONE/SAMPLE, DELONE/DELAY,...
C   FORTRAN STATEMENTS
              IF (IDELAY .NE. 0) THEN
          ...
          ...
          (digital setup equations)
          ...
          ...
          ENDIF
    SAMPLE PERIOD OF, SAMONE = TAUONE
    SAMPLE PERIOD OF, DELONE = TAUONE
    VALUE OF, SAMONE = ...
    NEXT VALUE OF, DELONE = ...
```

Note that both methods (explicit use of a `SAMPLE PERIOD OF` command, and the matching TAU method) were used in this example to define sampling periods for the two digital states.

For multi-rate sampled data systems, you have additional considerations to make. If you merely treated this case as a single rate model, you would have the digital setup equations executed at every (smallest) sampling period, TAU0. Thus, for sampling periods longer than this you would still be executing unused equations.

You can, however, add an additional test which will tell you whether or not the correct sampling time for a particular component has arrived. You must again weigh the computational expense of additional logic against unnecessary execution of the digital setup code at every sampling time.

Assuming you want this additional logic, you must add a test to determine if the current sampling period is the correct multiple of the smallest sampling period, TAU0, also known as the *fundamental sampling period*. This is most easily accomplished using the Easy5 function EZTAUS, which takes an argument (<tau>) and returns either 0 (false) or 1 (true) depending whether or not the current TIME is a sampling interval <tau>.

For example, consider the following Fortran component:

```
C   *** PARAMETERS = TAUONE,...
C   *** STATES = SAMONE/SAMPLE, DELONE/DELAY,...
C   FORTRAN STATEMENTS
                    IF ( EZTAUS(TAUONE) .EQ. 1)THEN
                    ...
                    ...
```

```
                                (digital rate update equations)
                                ...
                                ...
                                ENDIF
           NEXT VALUE OF, DELONE = ...
           VALUE OF, SAMONE = ...
```

In conclusion, while model correctness is always preserved through the use of Easy5 digital state declarations, extraneous model execution of large blocks of digital setup equations can be avoided through the use of additional logic.

Documenting and Printing the Model

See also: "Graphic Files, EMFs, and PostScript"

"Print Options"

Generating a Model Document File

You may generate a document of the model by selecting **File > Document Model** to display the menu shown in Figure 54.

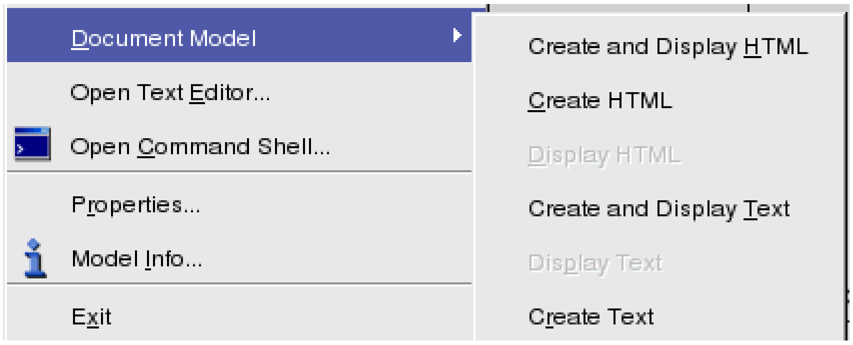


Figure 54 Generate Model Document File

There are several submenu options which allow you to generate either a text document, or an HTML document, and/or open the created documents. Note that this process requires a Model Building license.

The **Display** menu items are active only when a Model Document file already exists, otherwise they are grayed out.

Model HTML Document

To generate and display an HTML formatted document file, select **File > Document Model > Create & Display HTML**. This feature writes an HTML formatted file containing pertinent data on each component.

The component are listed alphabetically, and HTML tables are used to display the data contained in the Component Data Table. Tabular data is listed in tables, and Fortran and C component code is also listed.

When creating HTML model document files, Easy5 automatically creates a subdirectory named <*model_name*>.ezmd/documents to store these HTML files in. This keeps the potentially many HTML files together in one sub-directory. At least one HTML file is created for each component and submodel. The main Model Document file is named <*model_name*>.html. When requested, the HTML file is automatically opened using your default web browser.

You can also generate an HTML document of a model external to the Easy5 GUI by entering the following at an Easy5-enabled command prompt:

```
easy5x -docmod -html <model_file_or_path>
```

Note:	On Linux systems, the environment variable EASY5_BROWSER must be set to the command that launches a web browser.

Model Text Document

To generate and display an ASCII formatted text document file, select **File > Document Model > Create & Display Text**. This feature writes an ASCII formatted file containing pertinent data on each component.

The file alphabetically lists every component in the model, annotated with the data found in the Component Data Table. The data listing includes the component name and title, input and output names, user-defined names, and the corresponding values, description and units. Component connections are also listed. Tabular data is listed in columns, and Fortran and C component code is also listed.

When this option is selected, Easy5 writes to a local file named:

```
<model_name>.txt
```

When requested, the text file is automatically opened using the Easy5 Editor.

You may also document a model external to the Easy5 GUI by entering the following command at any Easy5-enabled command prompt:

```
easy5x -docmod <model_file>
```

This will write the output to the screen. To write the output to a file, use the “-o” option to specify the named *output_file* as follows:

```
easy5x -docmod  <model_file>  -o <output_file>
```

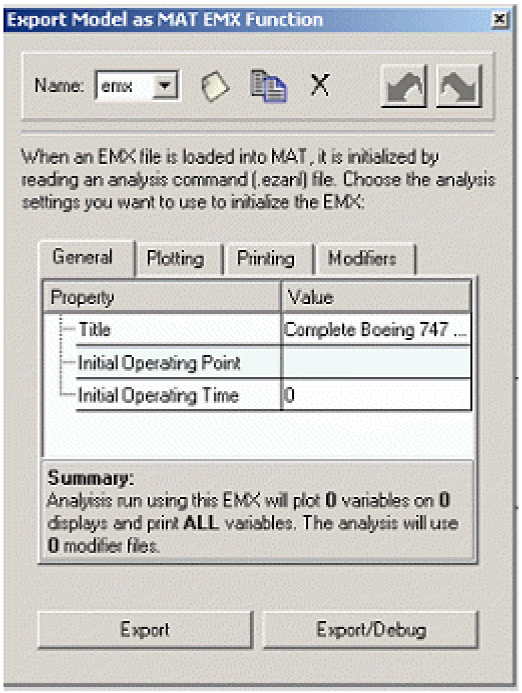
A single component may be documented by entering the following "docmod" option command:

```
easy5x -docmod <model_file>  cnxx
```

where, *cnxx* is the 2 to 4 character component name that you wish to document.

Exporting an Easy5 Model as a MAT EMX Function

One Build menu option in Easy5 lets the user export the model as a Matrix Algebra Tool (MAT) compatible function. Select **Build > Export Model as. . . > MAT EMX Function**.



When an EMX file is loaded into MAT, it is initialized by reading an analysis command (.ezanl) file. Choose the analysis settings you want to use to initialize the EMX:

Property	Value
Title	Complete Boeing 747 ...
Initial Operating Point	
Initial Operating Time	0

Summary:
Analysis run using this EMX will plot **0** variables on **0** displays and print **ALL** variables. The analysis will use **0** modifier files.

Figure 55 Export Model as MAT EMX File

MAT function “ezmodel”

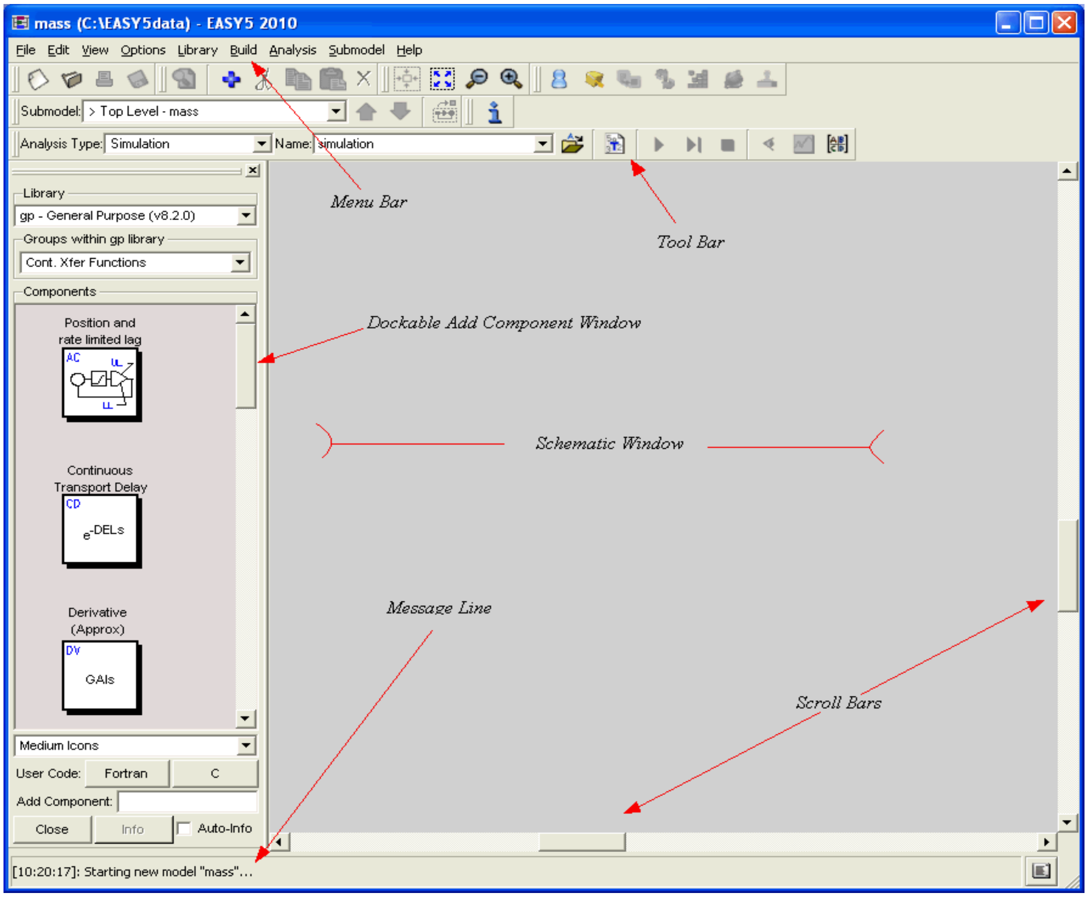
This command can be used in any MAT script: ezmodel(“Easy5 command”).

Examples of command syntax:

```
ezmodel(“simulate”) runs a simulation
fueltemp =ezmodel (“get value of, max_fuel_temperature”)
           returns the result to MAT
```

Easy5 Window

The Easy5 window is shown in Figure 56. This window is similar on both Linux and Windows systems. The window is made up of six main parts: the description lines, model info, menu bar, control panel, schematic window, scroll bars and the message line. Using your mouse, and to a limited extent your keyboard, you work in this window to construct your Easy5 model.


mass (C:\EASY5data) - EASY5 2010
File Edit View Options Library Build Analysis Submodel Help
Submodel: > Top Level - mass
Analysis Type: Simulation
Name: simulation
Library
gp - General Purpose (v8.2.0)
Groups within gp library
Cont. Xfer Functions
Components
Position and rate limited lag
Continuous Transport Delay
Derivative (Approx)
Medium Icons
User Code: Fortran C
Add Component:
Close Info Auto-Info
Menu Bar
Tool Bar
Dockable Add Component Window
Schematic Window
Message Line
Scroll Bars

[10:20:17]: Starting new model "mass"...

Figure 56 Easy5 Window

The following sections describe the parts and functions of the main window, and how to interact with it.

Description Lines

The title barof your Easy5 application window, as shown in Figure 57, contains the name of the model (mass), the directory where the model is located (that is, `C:\Easy5data\`*`username`*), and the version number of the application (`Easy5 2021.1`).

The model name is entered in the Select Model File to Open dialog when you create a new model. The model name you enter is used to name all files associated with that model. The first time you save the model, the version number appended to the new model is `0`, such as `mass.0`. In general, when you save any version of a multi-freshened model, Easy5 creates a new copy of the model with a version number that is one greater than the highest version of the model.

Model Info

A model can have an associated text file named <*model_name*>`.info.txt`, used to describe the model and provide information about the model. Selecting **File > Model Info**, or clicking on the "**i**" icon in the tool bar opens a rich-text editor, which allows you to add and store information about the model. Similarly, other formats are supported with the search order as: <*model_name*>`.info.htm` and <*model_name*>`.info.pdf`.

Menu Bar

The menu bar for Easy5 is located below the description lines and contains the following pull-down menus:

File Edit View Options Library Build Analysis Submodel Help

Tool Bar

The tool bar outlined in Figure 57 helps to orient you in your model. There are three drop down menus for you to identify the type of analysis, the name of the model, and an indicator of the drill-down level. Other icons in this tool bar use typical Windows interface icons, such as zoom and enzyme, open, save, and so on. To determine the use of an icon, place your mouse over the icon and a dialog will display indicating its function.

Dockable Add Component WIndow

This window lets you view and access a list of all libraries and their components. The first time you open Easy5 and begin to build your model, you will need to open this window using **Edit > Add Component**. Generally this window remains open on the schematic until all model components have been added. When you reopen the application, if the dockable window was open, it will display in the same manner as previously set.

This window is divided into three scrollable areas: the type of library, the library group, and the components. The top windowpane lists the *available* libraries. The libraries listed in your window will depend on whether you are licensed to use a particular library. See Ap. A: License Management for more information.

Libraries contain components for a specific application. For example, gas dynamics components are contained in the **gd** library. Some libraries contain a large number of components. To facilitate the management of these components, the libraries are further divided into groups. Groups within a selected library are listed in the second window pane.

Scroll Bars

You can two-dimensionally move the schematic window over the schematic pad to view different parts of the block diagram. One way of moving the schematic window over the schematic pad is by using the scroll bars located on the bottom and right side of the schematic window.

Message Line

The Easy5 message line is located at the bottom of the Easy5 window. Messages on this line inform you about actions and processes, as well as why a certain process may not be working. The arrow buttons at the end of the message line allow you to scroll up and down through messages that have been displayed during Easy5 session.

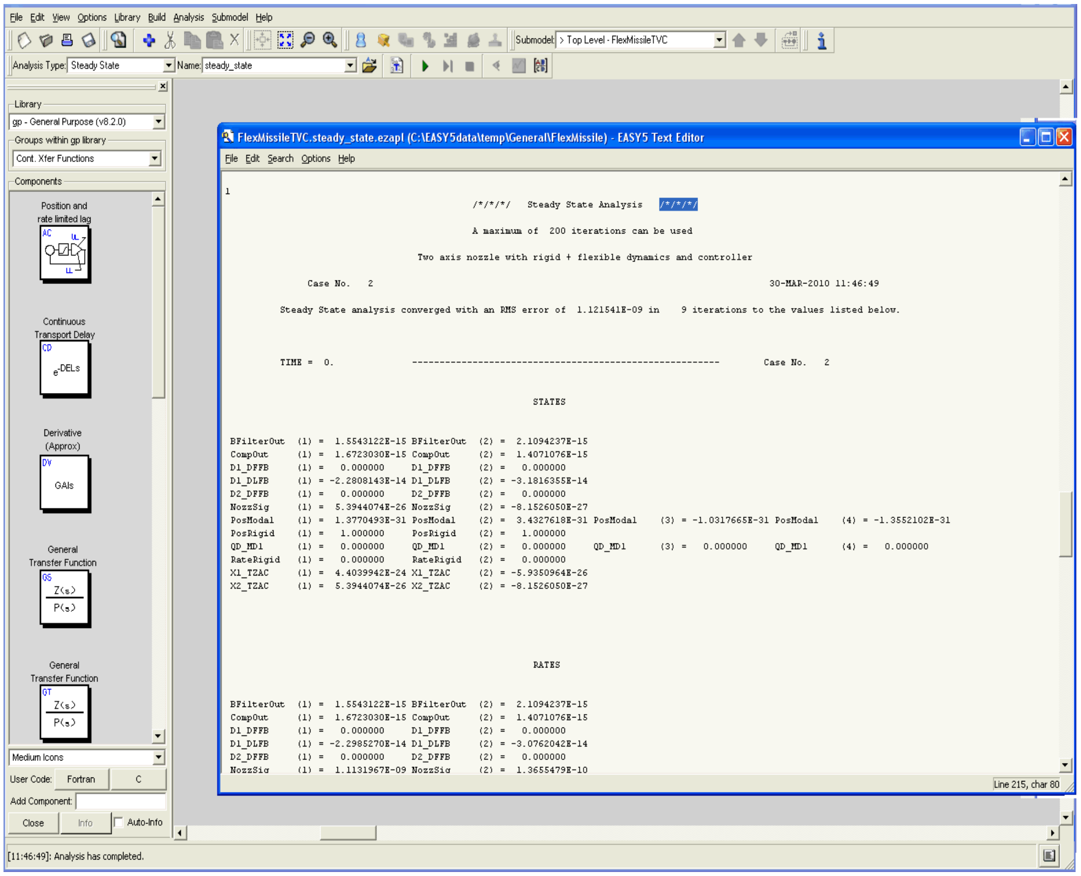
Schematic Window

The schematic window is the largest part of the Easy5 work area. It is the large rectangular area of the display that takes up about 80 percent of your screen.

The schematic pad is a two-dimensional area upon which a block diagram is constructed. You can move the window closer to or farther from the schematic pad, in effect zooming in and out on different parts of the block diagram by selecting the `[Zoom In]` and `[Zoom Out]` control panel buttons or icons.

Working with Easy5 Windows

When you work with Easy5, you often will have a number of windows open. Many commands open new windows allowing you to fill out data forms, view plots, or print output data. As a result, you may have many super imposed layers of windows, as shown in Figure 57.


File Edit View Options Library Build Analysis Submodel Help
Submodel: > Top Level - FlexMissileTVC
Analysis Type: Steady State
Name: steady_state
Library
gp - General Purpose (v8.2.0)
Groups within gp library
Cont. Xfer Functions
Components
Position and rate limited lag
Continuous Transport Delay
Derivative (Approx)
General Transfer Function
General Transfer Function
Medium Icons
User Code: Fortran C
Add Component:
Close Info Auto-Info
FlexMissileTVC.steady_state.ezapl (C:\EASY5data\temp\General\FlexMissile) - EASY5 Text Editor
File Edit Search Options Help
Line 215, char 80
[11:46:49]: Analysis has completed.

```
1
                                  /*/*/*/   Steady State Analysis   /*/*/*/

                                  A maximum of  200 iterations can be used

                             Two axis nozzle with rigid + flexible dynamics and controller

          Case No.   2                                                          30-MAR-2010 11:46:49

     Steady State analysis converged with an RMS error of  1.121541E-09 in    9 iterations to the values listed below.


     TIME =  0.            --------------------------------------------------------        Case No.   2


                                                  STATES


BFilterOut  (1) =  1.5543122E-15 BFilterOut  (2) =  2.1094237E-15
CompOut     (1) =  1.6723030E-15 CompOut     (2) =  1.4071076E-15
D1_DFFB     (1) =   0.000000     D1_DFFB     (2) =   0.000000
D1_DLFB     (1) = -2.2808143E-14 D1_DLFB     (2) = -3.1816355E-14
D2_DFFB     (1) =   0.000000     D2_DFFB     (2) =   0.000000
NozzSig     (1) =  5.3944074E-26 NozzSig     (2) = -8.1526050E-27
PosModal    (1) =  1.3770493E-31 PosModal    (2) =  3.4327618E-31 PosModal    (3) = -1.0317665E-31 PosModal    (4) = -1.3552102E-31
PosRigid    (1) =   1.000000     PosRigid    (2) =   1.000000
QD_MD1      (1) =   0.000000     QD_MD1      (2) =   0.000000     QD_MD1      (3) =   0.000000     QD_MD1      (4) =   0.000000
RateRigid   (1) =   0.000000     RateRigid   (2) =   0.000000
X1_TZAC     (1) =  4.4039942E-24 X1_TZAC     (2) = -5.9350964E-26
X2_TZAC     (1) =  5.3944074E-26 X2_TZAC     (2) = -8.1526050E-27



                                                  RATES


BFilterOut  (1) =  1.5543122E-15 BFilterOut  (2) =  2.1094237E-15
CompOut     (1) =  1.6723030E-15 CompOut     (2) =  1.4071076E-15
D1_DFFB     (1) =   0.000000     D1_DFFB     (2) =   0.000000
D1_DLFB     (1) = -2.2985270E-14 D1_DLFB     (2) = -3.0762042E-14
D2_DFFB     (1) =   0.000000     D2_DFFB     (2) =   0.000000
NozzSig     (1) =  1.1131967E-09 NozzSig     (2) =  1.3655479E-10
```

Figure 57 Easy5 Window With Multiple Windows

Note:	Windows: Select the ▁ button in the upper right corner of the window. This will "iconize" the window and place it in the taskbar.

Enlarging a Window

In most cases, you can enlarge an application window to fill the entire screen. This is a useful feature for enlarging the main Easy5 window. To enlarge a window to its maximum size, select the "maximize button" shown in Figure 56.

Note:	Windows: Select the ⬒ button in the upper right corner of the window. This will resize the window to the maximum size. Selecting it again will resize the window to it's previous size.

Closing a Window

One way to quit an application is to close the application window. All Easy5 windows have a menu item or push button that is used to close a window. Typical menu items used to close a window is "Exit". Typical pushbuttons are: OK, Cancel, or Done. Selecting any of these items will close the window.

Note:	On Windows, you can sometimes close a window by selecting the ☒ button in the upper right corner of the window.

Caution:	On **Linux** systems, do not close any Easy5 window using the window manager's control menu box. Window managers provide a control menu box, usually displayed in the left box of the title bar. Selecting this control menu box will drop down a menu that contains the "Close" command used to close the window. Do not use this method to close a window.

Eigenvalue Sensitivity Analysis

See also: "Analysis Data Form"

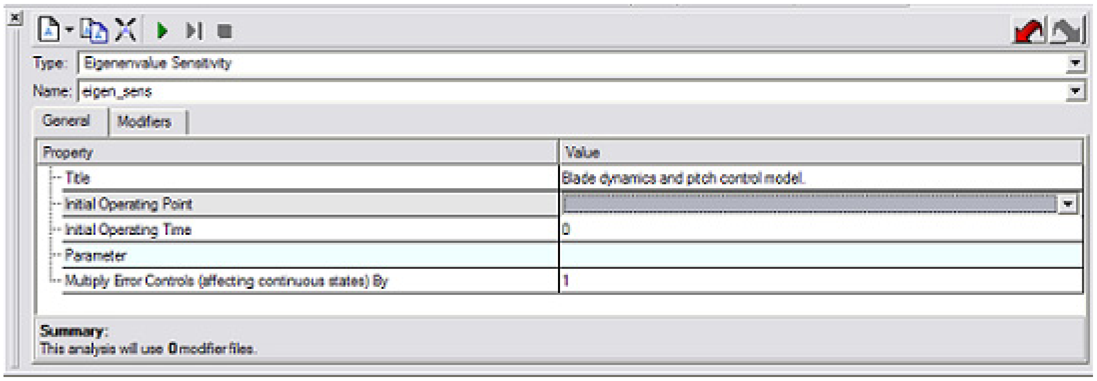
The Eigenvalue Sensitivity analysis measures the sensitivity of the system eigenvalues to a change in a specified system parameter. It is the ratio of the percentage change of the eigenvalue to the percentage change of the parameter for which the sensitivity is to be measured.

This type of analysis offers you an efficient method for assessing your model's sensitivity to parameters, since there are times when obtaining good values for parameters in your model is at best an approximation. This is fine, unless your model is overly sensitive to them, which is something that you should be aware of.

This analysis is also a more general way of showing how your system eigenvalues change with changes to one system parameter. This can be helpful when you are trying to "move" an undesirable eigenvalue. Of course, this requires an understanding of your model, at least to narrow down the list of "candidates" or possible parameters to a smaller, but meaningful list.

Setting up an Eigenvalue Sensitivity Analysis

The Eigenvalue Sensitivity analysis is setup and executed with the Eigenvalue Sensitivity Analysis data form. To access this data form, select **Analysis** from the main menu, then select **Eigenvalue Sensitivity...** from the **Linear** menu. Figure 58 shows the Eigenvalue Sensitivity data form with all data fields and options.


Type: Eigenenvalue Sensitivity
Name: eigen_sens

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Parameter	
Multiply Error Controls (affecting continuous states) By	1

Summary:
This analysis will use **0** modifier files.

Figure 58 Eigenvalue Sensitivity Analysis Data Form

To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to the section “Analysis Data Form”.

Note:	The Eigenvalue Sensitivity analysis does not work with models that contain discrete states, i.e., sampled data systems.

Defining the Eigenvalue Sensitivity Parameter

The first task in performing this analysis is to identify the parameter whose effect on the eigenvalues you want to measure. This parameter, called the "eigen parameter", must be an Easy5 parameter in your model that is defined in the "Eigen Parameter" data field. You define this parameter by typing a parameter name directly into the "Eigen Parameter" data field, or by using the "Pick" option to specify the eigen parameter.

Eigenvalue Sensitivity Analysis Method

The equations which describe the sensitivity measurements made during the Eigenvalue Sensitivity analysis are as follows:

$$S_{r_i} = S_{w_i} = \frac{1 - \frac{r'_i}{r_i}}{\left\| 1 - \frac{p'}{p} \right\|_2} \qquad S_{w_i} = \frac{1 - \frac{w'_i}{w_i}}{\left\| 1 - \frac{p'}{p} \right\|_2}$$

where:

S_{ri}	= Sensitivity measure of real part of ith eigenvalue to change in parameter P
S_{wi}	= Sensitivity measure of imaginary part of ith eigenvalue to change in parameter P
r_i	= Nominal value of real part of ith eigenvalue

w_i	= Nominal value of imaginary part of ith eigenvalue
p	= Nominal value of parameter for which sensitivity measure is being calculated
'	= Prime indicates perturbed values of parameters and eigenvalues
i	= 1,2,3, ... ,n and n = model order

Executable Model

See also: "Executable Output Files"

"Linking External Code"

Before you run any type of analysis, your graphical model must be converted into an executable program. The executable model, or "the executable", is a Fortran subroutine (called SUBROUTINE EQMO) or C function, representing your system that has been compiled and linked into the Easy5 analysis routines.

Executables are generated by the executable model builder program and are derived from the generic Easy5 block diagrams, and from any code that might be contained in User Code and/or Library components.

When you run an analysis, the executable is called by the main analysis program. Easy5 then accesses the model information in the component data tables and lookup data tables that contain the specific data you have defined for your model.

The **Build** menu is used to create the executable model. The **Build** menu options are shown in Figure 59. These options are reviewed in the following sections.

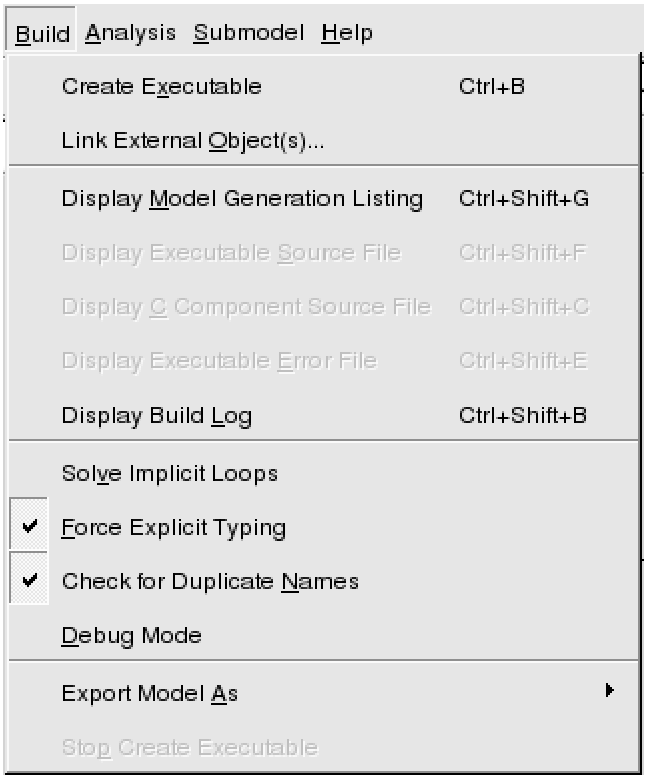

Build
Analysis
Submodel
Help
Create Executable Ctrl+B
Link External Object(s)...
Display Model Generation Listing Ctrl+Shift+G
Display Executable Source File Ctrl+Shift+F
Display C Component Source File Ctrl+Shift+C
Display Executable Error File Ctrl+Shift+E
Display Build Log Ctrl+Shift+B
Solve Implicit Loops
✓ Force Explicit Typing
✓ Check for Duplicate Names
Debug Mode
Export Model As
Stop Create Executable

Figure 59 Build Menu

Note:	The Build menu is only available when a Model Building license feature has been checked out.

Create Executable

You create an executable after you have assembled your model. To generate an executable model, select **Create Executable** (Ctrl+B) from the **Build** menu. Your model will automatically be saved for you as a first step in creating the executable. (If you have already saved a version of your model before selecting **Create Executable**, Easy5 will not save it again.) While the executable is being generated, the "Create Executable in progress..." message will appear in the message line.

Note: As the executable is being generated, you have control over the workstation and can continue to perform other operations. When the executable process is finished, the message “Executable has been created” will be displayed in the message line.

Only one executable model can exist for a given model regardless of the number of versions there are for that model. As long as components are not added to or deleted from your model or any connections changed, you do not have to recreate the executable. In general, you do not need to concern yourself with whether or not the executable is compatible with the version of the model you are working with. Easy5 always keeps track of executable and model version numbers, and warns you when incompatible models exist before an analysis is attempted.

Link External Object

Before creating the executable model, all external code called from within the model must first be compiled and linked. The **Link External Object....** menu item is used to setup and link all external code. For information on how to link external code, see "Linking External Code".

Solve Implicit Loops

If your model contains implicit loops, you may resolve these loops by setting the Solve Implicit Loops option. For information on solving implicit loops see “Implicit Model”.

Force Explicit Typing

You may force the compiler to trap all undefined variables by selecting the Force Explicit Typing option. This forces the user to explicitly type (define) all variables. This is a recommended programming practice.

Check for Duplicate Names

By default, this setting is on, and is used to verify that all input/output names are unique before a model executable is created.

Debug Mode

This option is used to generate the executable model with debug statements. Selecting this option allows you to run your system’s symbolic debugger from Easy5 (during an analysis). There are times when the standard print and plot output from Easy5 is not sufficient for isolating a problem that is occurring in your model. For these cases you should use the system symbolic debugger.

When you run with the symbolic debugger, you can set a break point in subroutine EQMO (the Fortran subroutine that represents your model), and then step through the code and monitor execution and print intermediate results. You can also set variables to different values if the results of a calculation are incorrect. Since there is *significant* variation in the commands for the symbolic debugger between the different platforms, *please consult the reference manual or man page for your platform*. For information on how to use the debugging feature, see "Debugging the Model and Analysis".

Stop Create Executable

Selecting this option will kill the Easy5 background process and stop the building of the executable model.

Executable Output Files

Easy5 produces three types of output files relating to executable model creation: the model generation listing file, the executable source file(s), and the executable error file. These files are discussed this section.

See also: "Executable Model"

Create Executable Process

The purpose of the "Create Executable" process is to take your graphical model and generate an executable file. In doing, Easy5 creates several files shown in Figure 60. First, intermediate files (<model>.*ezmod* and <model>.*ezmgl*) are built in the Easy5 language. The .ezmod file is seldom needed and may be deleted. Easy5 is a code generator. It takes the Easy5 model files and builds source code (<model>.*f* or <model>_c.*c*), and then links user-defined external object files and builds the executable file (<model>.*exe*).

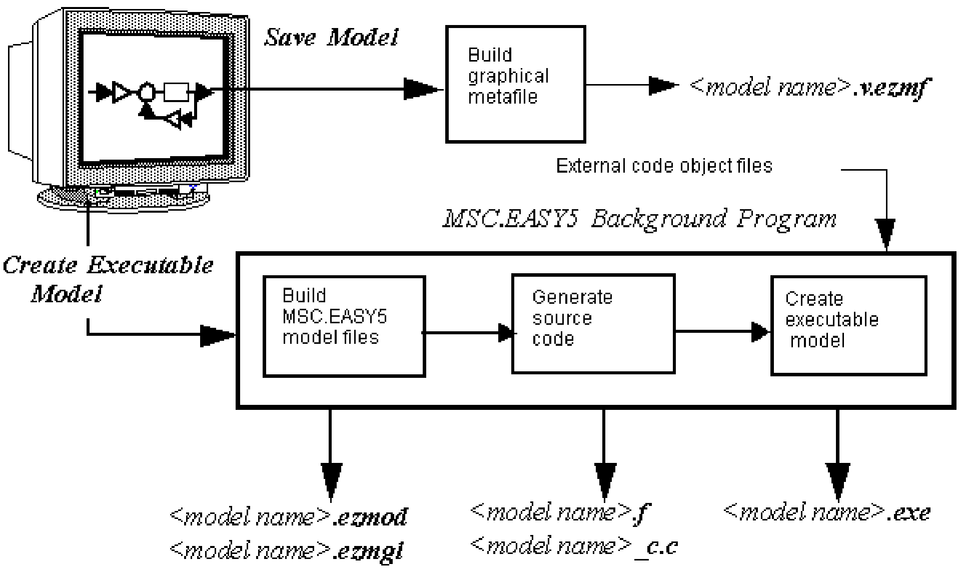


Figure 60 Model Files Created During Save and Create Executable

If the *Create Executable* process fails, an error file is generated (<model>.*ezerr*). You should make a habit of reviewing the executable source file. This is the source code that defines the graphical model in either Fortran or C code. These files are all explained in the sections that follow.

Model Generation Listing File

Every time you create an executable model, a model generation listing file for that model is updated. The model generation listing file is a primitive, nongraphic version of your model that is read by the Easy5 executable model builder program. (Easy5 creates this file for you automatically from your on screen graphical model.) There is only one model generation listing file for any given model in your account. The naming convention for this file is: <*model*>.ezmgl. Every time you recreate an executable this file is over written.

After you create an executable you can examine the model generation listing file. To examine the model generation listing file, open the **Build > Display Model Generation Listing**.

The primary purpose of generating this file is to warn you if an *explicit* model can **not** be built. If this occurs, the following warning will be written in the message line:

Error occurred during model generation phase

Any time you get this message, you should examine the model generation listing file. If for example, the model generator fails to build an explicit model because of one or more implicit loops, a `FATAL ERROR` is printed at the bottom of this file. An example of this error message is:

```
*** FATAL ERROR ***THE FOLLOWING CONNECTIONS FORM AN IMPLICIT LOOP. ALL CONNECTIONS
BETWEEN ANY TWO COMPONENTS IN THIS LOOP MUST BE REMOVED OR REPLACED BY STATE OUTPUTS.IF
OTHER IMPLICIT LOOPS EXIST, THEY CANNOT BE DETECTED UNTIL THIS LOOP IS REMOVED FROM THE
MODEL.

    S_Out_LE   FROM COMPONENT LE   IS AN INPUT TO THE FOLLOWING SORT BLOCK OF COMPONENT
SJ
```

Before proceeding, you must break the loop that is causing the implicit model. For more information, see "Implicit Model".

Executable Source File

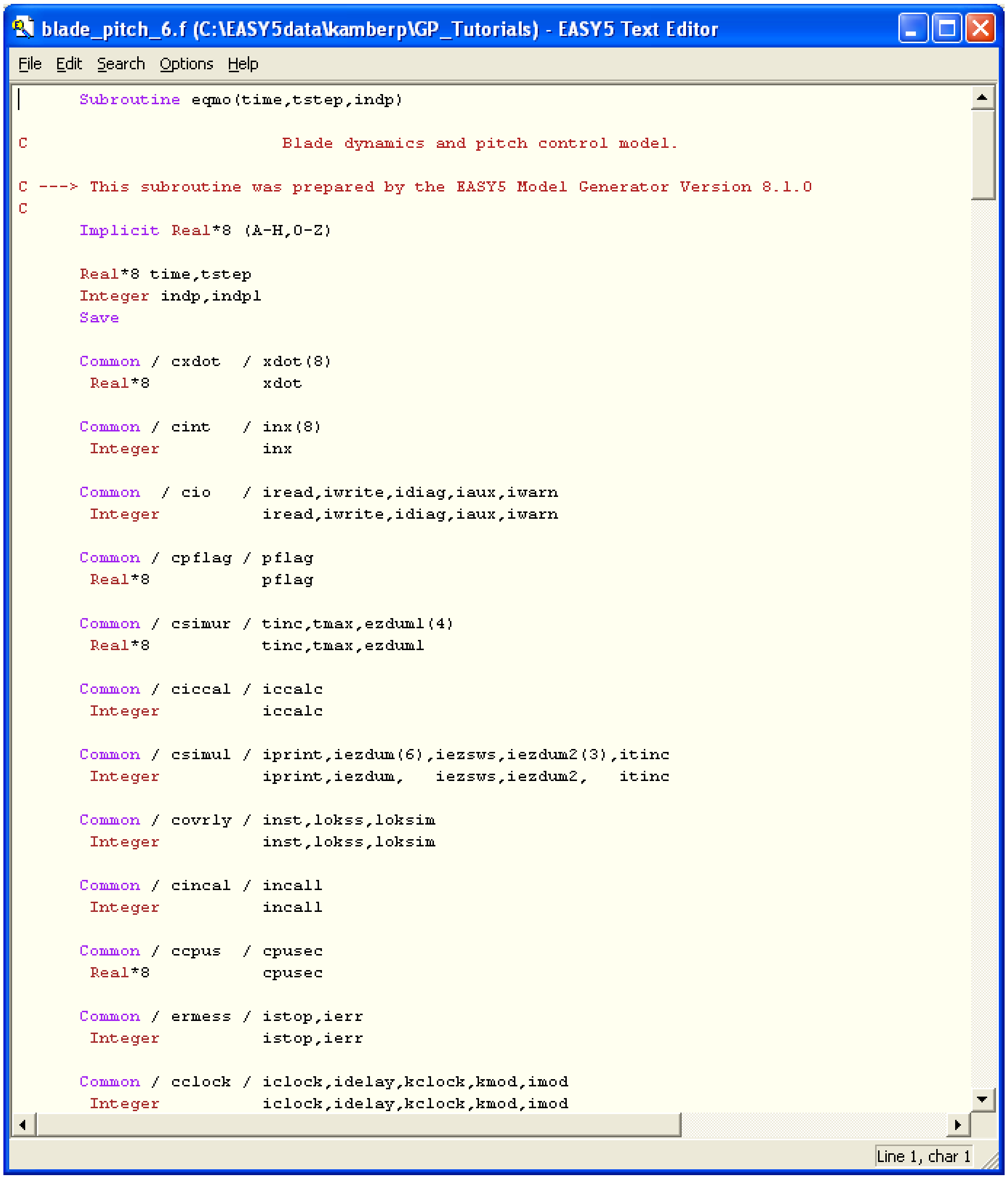
The executable source file contains the compilation listing for SUBROUTINE EQMO, the Fortran model of your system. If a compilation or run time error occurs, you will be provided with the EQMO source line number that caused the problem and you must look in this file to locate the problem. The naming convention for this file is: <*model*>.f.

After you create an executable, you should examine the executable source file. Warning messages that require your attention may appear in this file.

To examine the executable source file, select the **Build > Display Executable Source File**. It is important that you analyze the executable source file and become familiar with the structure of the subroutine EQMO. A portion of a sample executable source file is shown in Figure 61.

The first section is created by Easy5 to define all the common variables and data types. This is followed by the Easy5 components, which are called as Fortran subroutines. The code from Fortran components is loaded directly into the subroutine EQMO. The components and the Fortran component's code may be sorted and placed in an order required to obtain an explicit model.

If your model contains C code components, Easy5 takes the C code from each C code component, creates a C function, and places all of these functions into a single file called <*model*>_c.c. This file gets compiled and linked with other Easy5 files to create the final executable file. To examine the C code source file, select the menu **Build > Display C Source File.** See "Steady-State Analysis Method" for more information on the Easy5 Text Editor.


blade_pitch_6.f (C:\EASY5data\kamberp\GP_Tutorials) - EASY5 Text Editor
File Edit Search Options Help
Line 1, char 1

```
      Subroutine eqmo(time,tstep,indp)

C                         Blade dynamics and pitch control model.

C ---> This subroutine was prepared by the EASY5 Model Generator Version 8.1.0
C
      Implicit Real*8 (A-H,O-Z)

      Real*8 time,tstep
      Integer indp,indp1
      Save

      Common / cxdot  / xdot(8)
       Real*8           xdot

      Common / cint   / inx(8)
       Integer          inx

      Common  / cio   / iread,iwrite,idiag,iaux,iwarn
       Integer          iread,iwrite,idiag,iaux,iwarn

      Common / cpflag / pflag
       Real*8           pflag

      Common / csimur / tinc,tmax,ezdum1(4)
       Real*8           tinc,tmax,ezdum1

      Common / ciccal / iccalc
       Integer          iccalc

      Common / csimul / iprint,iezdum(6),iezsws,iezdum2(3),itinc
       Integer          iprint,iezdum,   iezsws,iezdum2,   itinc

      Common / covrly / inst,lokss,loksim
       Integer          inst,lokss,loksim

      Common / cincal / incall
       Integer          incall

      Common / ccpus  / cpusec
       Real*8           cpusec

      Common / ermess / istop,ierr
       Integer          istop,ierr

      Common / cclock / iclock,idelay,kclock,kmod,imod
       Integer          iclock,idelay,kclock,kmod,imod
```

Figure 61 Sample Listing of the Executable Source File Using the Text Editor

Executable Error File

One phase of executable creation involves the compilation of the Fortran subroutine (SUBROUTINE EQMO) which represents your system. Easy5 will warn you when errors occur during the compilation of this subroutine, and you must examine the executable error file to find out what went wrong. In most cases, compilation errors occur because you have made mistakes in the User Code and/or Library component code used in your model. The naming convention for this file is: <*model*>.ezerr.

Once your model has been successfully compiled, it is linked with the Easy5 analysis routines and any external object files referenced in any User Code and/or Library components to form an executable model. If errors occur during the link phase, Easy5 will notify you and you must examine the executable error file to find out what went wrong. In most cases, link errors occur due to undefined external references in User Code and/or Library components in your model.

Note: Further information about the compilation and linking of your model is available in the Build Log (file `easy5_build.log`). For analysis information, check the Analysis Log (file `easy5_analysis.log`). These log files can be displayed using the menu items **Build > Display Build Log...** or **Analysis > Display Analysis Log...** menu items, respectively

If errors occurred during the compilation of your executable file, you can examine the executable error file by selecting **Build > Display Executable Source File**, which opens a read only display area containing the source file that you view using the scroll bars.

Note: If compilation errors do not occur, this file will be empty.

External (Environment) Variables

External variables (commonly called *environment* variables) are used to setup special conditions prior to running Easy5. The syntax used to set environment variables depends on the type of shell you are using. The most common shells are the C, Bourne, and Korn shells, as well as the MS DOS Command Prompt in Windows. The three operations used to manage environment variables are: setting an environment variables, deleting an environment variable, and displaying the current values of an environment variable. The appropriate command syntax for each of these operations is given below for each type of shell.

Note: **Windows:** In place of commands entered at a command prompt, you can also access and modify global environment variables by selecting Control Panel > System > Environment.

To set an environment variable:

For shell type . . .	Use the command syntax . . .
C	setenv *<env_variable>* *<value>*
Bourne and Korn	export *<env_variable>*=*<value>*
MS DOS Command Prompt	`set` *<env_variable>*=*<value>*

Note: For Linux systems, if a `<value>` contains special characters, such as a slash (/) a blank space, or a period (.), the `<value>` should be enclosed by double quotes ("). When defining Windows environment variables, do not put quotes around the values, even if it contains multiple values with blank space separators.

To delete an environment variable:

For shell type . . .	Use the command syntax . . .
C	`unsetenv` *<env_variable>*
Bourne	`unset` *<env_variable>*
Korn	*<env_variable>*=
MS DOS Command Prompt	`set` *<env_variable>*=

To reference (display) the current value of an environment variable:

For shell type . . .	Use the command syntax . . .
C, Bourne, and Korn	`echo` $*<env_variable>*
MS DOS Command Prompt	`echo` %*<env_variable>*%

To display the value of all environment variables:

For shell type . . .	Use the command syntax . . .
C, Bourne, and Korn	`printenv` *or* `env`
MS DOS Command Prompt	`set`

A list of the Easy5 environment variables being used is obtained by the -varset option. To use this option enter the following at a command prompt:

```
    easy5x -varset
```

A list of the most commonly used Easy5 environment variables is shown in the following tables. To obtain a complete list enter the following command at a command shell prompt:

```
easy5x -vars
```

Table 1-11 Library search path control

Variable	Value	Description
EASY5_USER_LIB	*<directory_name>*	Editable library directory for personal use.
EASY5_GROUP_LIB	*<directory_name>*	Read-only library directory for group use.
EASY5_SITE_LIB	*<directory_name>*	Read-only library directory for site-wide access.
EASY5_FIRST_LIB	*<xx>*	Forces the *xx* library to be searched first, instead of the gp library, when adding a component.
EASY5_IGNORE_LIB	*<aa bb xx ...>*	If set, will not open the libraries in this string.
EASY5_SELECT_LIB	*<aa bb xx ...>*	If set, open only libraries in this string (ex: "ec hy"). Note: the gp library is always opened.
EASY5_OBJECT	*<filenames>*	Defines object code to be linked (space delimited).
WSLIBDIR	<library dir name>	Used to set an alternate Easy5 library directory.

Table 1-12 File Space Quota Control (be careful if you reduce these!)

Variable	Value	Description
WSANLSPACE	<#>	Minimum megabytes for analysis (default = 3).
WSEXESPACE	<#>	Min. megabytes to create executable (default = 6).
WSMFSPACE	<#>	Min. megabytes for model save (default = 2.5).

Table 1-13 Options for display control

Variable	Value	Description
WSNOAUTO_RESULTS	on	Set initial state of menu selection: "Options->Automatic Results Display" to off (output data will not be automatically displayed).
WSCONCOLOR	<*x*>	Sets connection line color, where *x* defines the color with an integer 0 - 7 as follows: 0 = black (def) 1 = red 2 = green 3 = blue 4 = cyan 5 = yellow 6 = magenta 7 = white
WSDEBUG	on	Activates debugging of the GUI and prints debug messages to the window from which Easy5 is run. This dramatically slows down Easy5, and should only be used for GUI debugging. When finished "unset" this variable.

Table 1-14 Connection options and label options

Variable	Value	Description
WSALTCONNECT	on	Allows for partial default connections
WSALTCONLIB	<*libs*>	For the given library or libraries <*libs*>, the port connection logic reverts back to Easy5's v5.x method -- that is, the port connection table does not appear when ambiguous port connections are being made.
CNT_LABEL_OFF	on	New connections will not be labeled.
CNT_LABEL_HEAD	on	Default labels appear at "head" end of connection.
CNT_LABEL_VERTICAL	on	Default labels drawn vertically.

Table 1-14 Connection options and label options (continued)

Variable	Value	Description
CNT_LABEL_BELOW_LEFT	on	Default labels drawn below and/or to the left of the connection.
HIDE_CNT_OUTPUT_LABELS	on	Do not show unported connection labels.
HIDE_CNT_SUBMODEL_LABELS	on	Do not show labels for off page connectors in a submodel.
SHOW_CNT_PORT_LABELS	on	Show ported connection labels.

Table 1-15 Background (Easy5) process options

Variable	Value	Description
auxfile	*<file_name>*	Name of file that contains Easy5 format auxiliary input data.
ezdebug	on	If set to anything, compile with debug options and run debugger.
copt	*<compile options>*	C compilation options -- overrides default setting
ftnopt	*<compile options>*	FORTRAN compilation options -- overrides default setting
bindopt	*<load options>*	options to loader (ld) -- overrides default setting
WSDEBUGGERNAME	*<debugger name>*	The name of an alternate debugger for source level debugging.
xtralib	<library or object file>	Adds this object or library last in the linking sequence

Table 1-16 Model generation input modification

Variable	Value	Description
WSMODAUX1	*<file_name>*	Will include named file before model definition in .mod file
WSMODAUX2	*<file_name>*	Will include named file after model definition in .mod file

Table 1-17 Miscellaneous options

Variable	Value	Description
EASY5_BROWSER	<*path*>	Set to the path of the HTML browser program (for Linux only)
EASY5_PDF_READER	<*path*>	Set to the path of the Adobe Acrobat Reader program (for Linux only)
EZ_NO_CONFIRM	true	Set to true to skip Easy5 shell-script prompts (for advanced users only)
WSPLOT_AUTO_SEL_LO	true	If set, plotter will automatically select layout file.
EASY5_XCOMP_DIR	<directory>	Override the default Easy5 "xcomp" directory path (the top-level extensions directory path which is "EZHOME"/xcomp)

For a more complete list of available environment variables, please use the shell command “`easy5x -vars`”.

Fortran Component

See also: "C Component"

"Compiling External Code"

"Linking External Code"

User Guide, Chapter 5 - Code Components

Fortran code is added to a model using the Fortran code component. The Fortran code component is accessed from the Add Components window, by selecting the “Fortran” button from the bottom of the window, then CLICK-L to drop the Fortran code component onto the schematic. The Fortran code component is named FO<xx>, where, <xx> is the component designator.

Complete information on how to use the Fortran code component to add Fortran code to your model is given in the User Guide, Chapter 5 - Code Components. The user guide describes how to add Fortran code variables to the data table, and how to add and edit the code in the code editor. This section is intended to give additional information specific to the Fortran code component.

Forced Explicit Typing

Explicit typing, or *strong typing*, requires all variable names to be explicitly typed, that is defined, which is considered a good programming practice. This is also the convention for most other modern programming languages such as C. This programming practice is very useful to help you avoid making typing errors.

To turn on this option, select Force Explicit Typing from the Build menu option. This forces Easy5 to write an `IMPLICIT NONE` statement to your model executable source file. This also forces you to explicitly declare all local variables (non-Easy5 variables) in your code using one or more Easy5 `DECLARATIONS` commands.

Otherwise, a Fortran compilation error occurs when building your model executable. *It is strongly recommended that you select this option to further reduce the opportunity for programming errors in your Easy5 user-defined code components.*

Note:	When using any "pre-release" version of Easy5 the "forced explicit typing" option is activated by default.

Example of Forced Explicit Typing

Assume that you have a Fortran component with one declared Easy5 input, `IN1`, one declared Easy5 output, `OUT`, and three local variables called `TVAL1` (real), `TVAL2` (real) and `IVAL` (integer).

Your Fortran component code might look something like this:

```
DECLARATIONS, REAL*8 TVAL1, TVAL2
DECLARATIONS, INTEGER IVAL
TVAL1 = MAX( IN1, ZERO)
TVAL2 = TVAL1*1.5
IVAL = INT(TVAL1)
OUT = TVAL1 + TVAL2 - REAL(IVAL)
```

Note that in this example both `IN1` and `OUT`, as declared input or output quantities, are already explicitly typed as `REAL*8` by Easy5 and should not be typed again with a `DECLARATIONS` command.

Using Integer or Logical Variables in Fortran Code

Easy5 declares each input or output name listed in the Fortran component data table as real, double precision. Therefore, if you want to use integer or logical variables in your Fortran code, they must not be declared as inputs or outputs.

If an input must be used as an integer (e.g., as the index on a DO loop) or an integer must be transmitted to other components as an output, use the following method. For example, suppose that II is an integer input to the Fortran component and that JJ, calculated in the Fortran code, is an integer output.

1. Define dummy names for communicating these variables to and from other components. Add XII as an input and XJJ as an output.
2. Assign the appropriate input value to a local variable in the Fortran code. For example, the real double precision input XII, is assigned to the integer II as follows:

   II = XII

   II is a local variable and is defined as integer because it starts with the letter I. This is also true for JJ used in step 3.
3. After the integer output (JJ) is calculated, assign its value to the appropriate output name. For example,

   ```
   XJJ = JJ
   ```

Adding Nonexecutable Fortran Statements

To include nonexecutable statements (DIMENSION, COMMON, DATA, INTEGER, and others) in a Fortran component, use this Easy5 command:

```
DECLARATIONS, nonexecutable statement
```

The comma following `DECLARATIONS` must appear.

Supply a separate declaration command for each type of nonexecutable statement. For example, suppose that DUMMY is to be dimensioned (3,12), XARRAY dimensioned to (10,10), and LOGI and ANSW are to be declared logical. The appropriate declaration statements are

```
DECLARATIONS, DIMENSION DUMMY (3,12), XARRAY(10,10)
DECLARATIONS, LOGICAL LOGI, ANSW
```

You can continue declaration comments on the following lines by the usual “characters in column-6” convention.

Note:	Easy5 declares any name displayed as an input or output to be double precision. You can only declare local variables (not defined in the Fortran component input/output table) to be otherwise. Easy5 includes all names that are displayed as an input or output in a COMMON block. You can include only local variables to your Fortran component in a COMMON block that you declare.

Reserved Fortran Unit Numbers

Easy5 reserves Fortran unit numbers for generating the model and executing the analyses. The Fortran units numbers available to the user are unit numbers 50 through 69.

Caution:	Unit 5 is used for Easy5 input, and Unit 6 is used for Easy5 output. Do not redefine these unit numbers.

Adding Comments to Fortran Code

You can add two types of comment lines to your Fortran code starting either with a `C` or an `*` in column one. The C comment line is a Fortran comment that is displayed in the source listing of your executable model. The `*` comment line is an Easy5 comment displayed only in the Fortran code area in the CDT; it is not displayed in the executable source file for your model. Use of other comments, such as inline comments prefaced with a “!”, are generally less reliable for use with Easy5.

Easy5 Reserved Words

Easy5 contains a set of reserved words that you cannot use in your User Code or Library components as inputs or outputs. In addition, these Fortran variable names should never be set in your code. (Exceptions to this are `ISTOP` and `PFLAG`, which can be set as described in the following sections). Figure 62 contains an alphabetical list of all Easy5 reserved words.

C	GRM	IPRINT	PI
CCLOCK	IAUX	IREAD	R
CIO	ICCALC	ISTOP	RENAME
CKLOCK	ICLOCK	ITINC	RPD
CP	IDELAY	IWARN	SDOT
CPOITR	IDIAG	IWRITE	STOP
CPUSEC	IDUMMY	IXOC	TABLE
CSIMUL	IERR	KCLOCK	TAU___*
CSIMUR	IEZ___*	KMOD	TAU0
CV	IFINAL	LOKSIM	TIME
CX	IMOD	LOKSS	TINC
CXDOT	INCALL	MJITER	TMAX
D	INDP	MNITER	TSTEP
*Any input/output name beginning with these letters are reserved words.			
EQMO	INX	NRCMAX	
ERMESS	IOC1	NS	VS
EZ___*	IOC2	NU	XDOT
FO___*	IOC3	ONE	Z
GRE	IOC4	PFLAG	ZERO

Figure 62 Easy5 Reserved Words

Caution:	If you declare any of these reserved words as inputs or outputs to User Code or Library components, Easy5 will detect them and issue a fatal error message to that effect. However, Easy5 cannot detect if you simply set (use it to the left of an equal sign) one of the reserved words, so exercise care when writing your code.

Several of these variables can be used in your User Code and library components to monitor an analysis. In example is the usage of the ICCALC reserved word, used to set initial conditions as described in the next section. In addition, two of the words, PFLAG and ISTOP, can actually be set to control the simulation analysis. Easy5 reserved words that you are likely to find useful are described in "Reserved Words".

Calculating Initial Condition Values in a User-Code Component

Initial conditions are often a function of one or more parameters or other states in your model. Your model can include one or more User-Code components containing code that sets initial values for these states.

For information on calculating Initial Conditions for library components describing the EZSETV and EZINIT subroutines review the topic "Initial Condition Calculation".

Prior to every analysis, Easy5 sets the global variable named ICCALC to a value of 1 and calls your model. (At all other times, ICCALC is set to 0.) By testing on this variable, you can include special code in a User Code component to calculate initial conditions for specific states. After this one time call to the model, the initial condition vector is updated with any new state values that were calculated; no integration is performed.

The following block of Fortran code shows the use of the ICCALC global variable:

```
if (ICCALC .eq. 1) then
   call ezsetv(S_Out_IN01, GKI_IN01 * 2.345d0)
   call ezsetv(S_Out_TF03, S_Out_IN01 + Z0_TF03 / 4.99d0)
endif
```

Analogous code could be defined in a C component also. In this example, initial condition values for two states, S_Out_IN01 and S_Out_TF03, are being calculated. S_Out_IN01 is a function of its gain parameter, GKI_IN01, and S_Out_TF03 is a function of both S_Out_IN01 and one of its zeros, Z0_TF03. If this code was included in a User Code component, any initial condition values set for S_Out_IN01 and S_Out_TF03 in their respective component data tables would be replaced by the values calculated in this code when any analysis was executed.

Note:	Initial condition values that are displayed in component data tables are not changed by Fortran components that use the ICCALC value. They are simply overwritten with new values before an analysis.

Easy5 Matrix Operations

You can perform matrix operations in Easy5 by using a set of shorthand matrix expressions, or you can call Easy5 subroutines that perform special matrix operations. Both methods are covered in detail in "Matrix Operations". An example of using matrix operations in a Fortran component follows.

Assume that you have a matrix named [B] and you want to calculate the eigenvalues (roots) of this matrix. The following matrix operation notation is used:

```
    /A/ = /B/E
```

where the resulting eigenvalues of [B] will be saved in the vector A.

There is matrix notation for performing all the common matrix operations: addition, subtraction, multiply, cross product, dot product, inverse (linear equation solution), transpose, eigenvalues, defining identity and null matrices, and defining an equality matrix. These are listed in "Matrix Operations".

Sorting Fortran Component Code

At your request, Easy5 sorts your Fortran code in the same way that it sorts your model and the code in any library components used in your model. By default, no sorting is done for Fortran (or C) components. The Fortran Component editor contains an **Sorted** checkbox in the upper right corner. To enable sorting, click the **Sorted** checkbox.

If the **Sorted** box is unchecked, Easy5 (specifically, the code generator) only sorts on inputs that are connected -- it does not look at the sort block code to confirm that quantities defined (anything on the left-hand-side of an equals "=" sign) are assigned by quantities already defined previously. A Fortran sort block with no connected inputs is thus always sorted "to the top" (but always after special "mandatory" library components that define global variables for a particular application library).

If the **Sorted** box is checked, the Easy5 code generator not only sorts on the inputs that are connected, but it also analyzes the (sort block's) code body to make sure that all quantities defined (anything on the left-hand-side of an equals sign) are dependent on quantities defined in a previous (or current) sort block.

You can also add `BEGIN SORT BLOCK` and `STOP SORT BLOCK` commands to turn Fortran sorting on and off, respectively. This is recommended to help minimize the number of sort blocks in your model. These commands are placed directly in your Fortran code on separate lines to delineate Easy5 sort blocks. For more information see the topic "Sort Blocks".

If you add one or more of these commands to your Fortran code, your Fortran component can be no longer that 512 lines, including Fortran comment lines and the `BEGIN SORT BLOCK` and `STOP SORT BLOCK` command lines. This 512 line limit only applies if you request sorting.

Function Scan Analysis

See also :"Analysis Data Form"

The Function Scan analysis is most commonly used as a model verification tool to check the algebraic relationship between two points in the model. You can specify one or two independent variables to be varied and a dependent variable of interest, and Easy5 will vary the independent variable(s) and produce a plot of each static functional relationship that exists.

Figure 63 shows examples of the types of plots generated with the Function Scan analysis. The items labeled in this figure correspond to data fields in the function scan data form discussed below.

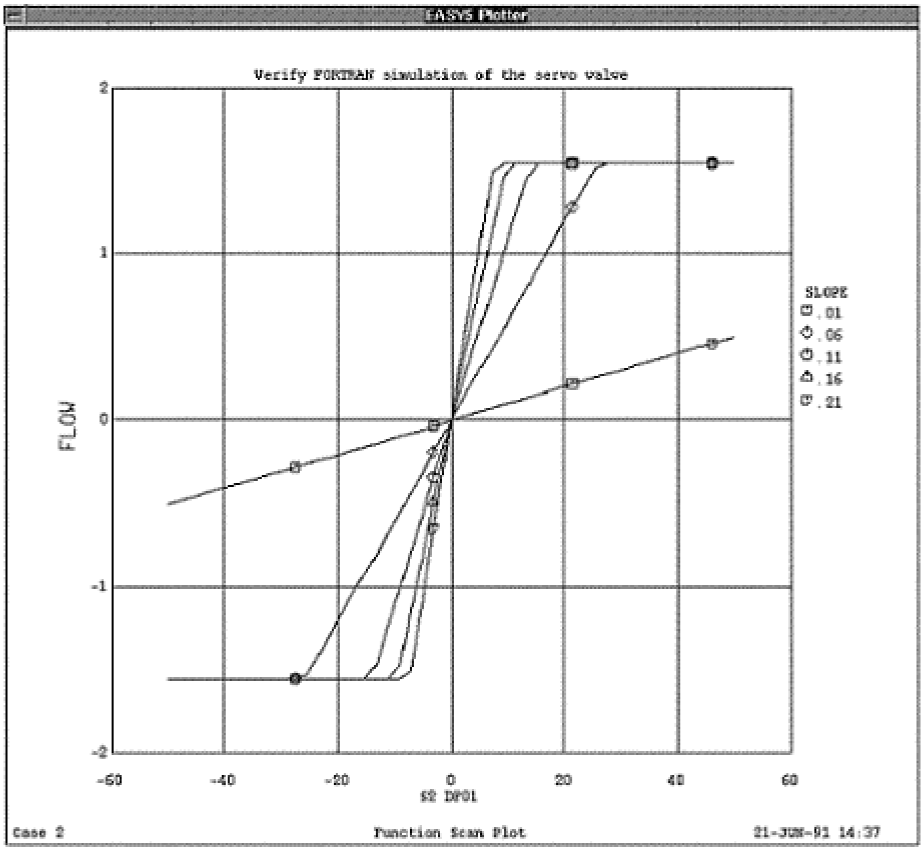
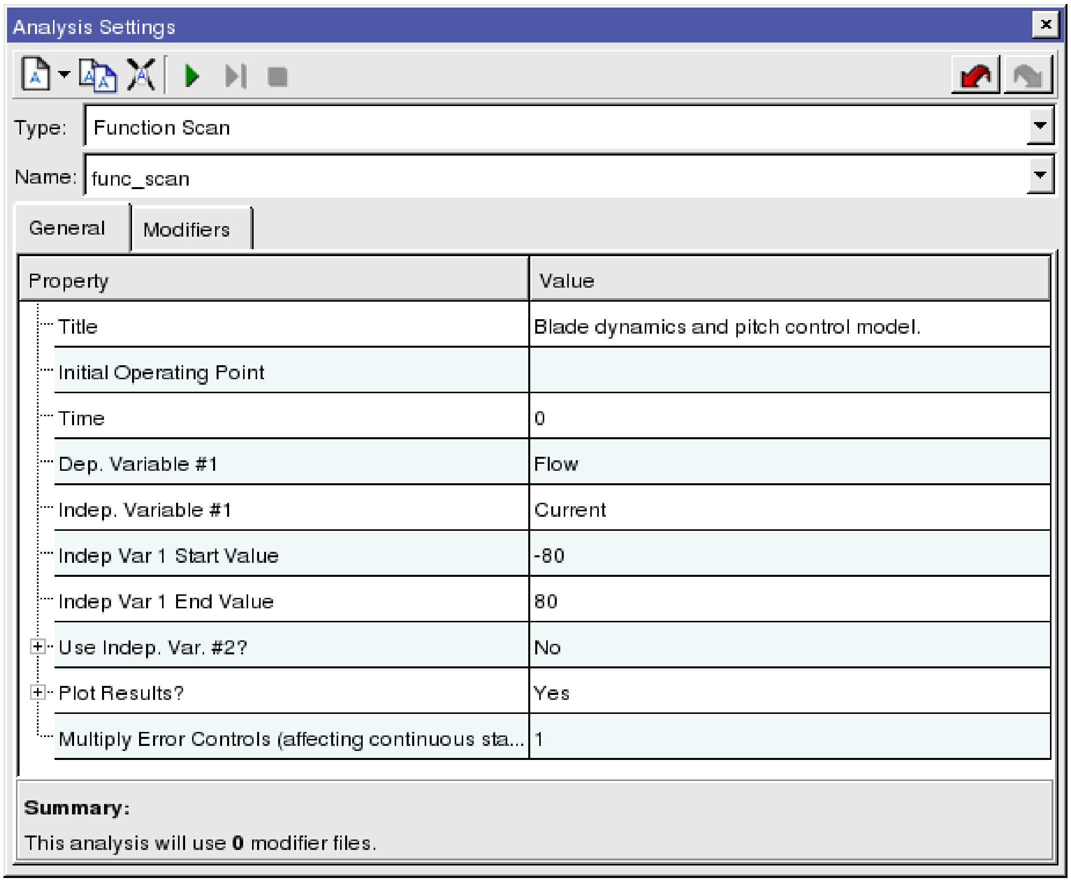


Figure 63 Function Scan Plots

Setting up a Function Scan Analysis

The Function Scan analysis is setup and executed using the Function Scan Analysis data form. This data form is accessed by first selecting **Analysis** from the main menu bar, then **Miscellaneous**, and **Function Scan...**

The function scan data form with all data fields and options is shown in Figure 64. To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to "Analysis Data Form".



Type: Function Scan

Name: func_scan

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Time	0
Dep. Variable #1	Flow
Indep. Variable #1	Current
Indep Var 1 Start Value	-80
Indep Var 1 End Value	80
Use Indep. Var. #2?	No
Plot Results?	Yes
Multiply Error Controls (affecting continuous sta...	1

Summary:
This analysis will use **0** modifier files.

Figure 64 Function Scan Analysis Data Form

Defining the Time of the Function Scan Analysis

You can define the time at which the Function Scan analysis will be performed with the “Time” value in the data form. Normally, the time value is zero, so the default value does not need to be changed. However, if you have time dependent functions in your model (e.g., table lookups as a function of time) and you wish to run this analysis at a time value other than zero, set this value appropriately. To set the “Time” value, select the data field following “Time =” and type in a new value.

Defining the First Independent Variable

The "1st Independent Var" data field is used to define the name of the independent variable that will be varied to produce the function scan plots. This quantity can be any state, variable, or parameter in your model. You can enter an independent variable name directly by typing it in the data field, or using the "Pick" option.

Defining the Dependent Variable

The "Dependent Variable" data field is used to define the dependent variable that will be plotted during the Function Scan analysis. The dependent variable must be the name of one of the variables in your model. You can enter a value by selecting the data field and typing in a dependent variable name. You can also use the "Pick" option to specify this quantity.

Defining the Independent Variable's Starting Value

The data field following “Start Value =” is used to define the initial value given the independent variable. To enter a start value, select the data field and enter a value.

Defining the Independent Variable's Final Value

The data field following “Stop Value =” is used to define the final value assigned to the independent variable during this analysis. To enter a final value, select the data field and enter a value.

The number of steps taken between "Start Value" and "Stop Value" is not under your control. Easy5 automatically takes 49 steps from "Start Value" to the "Stop Value", resulting in 50 points being calculated for the dependent variable.

Function Scan with Two Independent Values

You have the option of defining a second independent variable for the Function Scan analysis. If you do this, multiple function scan curves will be generated as previously shown in Figure 64. To define a second independent variable, select “Yes” following “Second Independent Variable:.” The data fields that appear in the data form are described below.

Defining the Second Independent Variable

Define the second independent variable using the “2nd Ind. Var:” data field. Type in the name for the variable directly into the data form, or use the “Pick” method described in the “Model Explorer Window”.

Defining the Initial Value for the Second Independent Variable

The starting value for the second independent variable is defined in the “2nd Start Val” data field. Type the secondary starting value directly in the data form, or use the “Pick” method described in the “Model Explorer Window”.

The final value for the second independent variable is defined using two parameters in the function scan data form, “2nd Var Step Size” and “# of Curves”, rather than a single parameter. These parameters are described in the next section.

Defining the Secondary Variable Step Size

The “2nd Var Step Size” parameter is used to define the step size applied to the secondary independent variable. This value, and the “# Of Curves” parameter described below, define the final value given to the second independent variable during this analysis. To define the step size, select the “2nd Var Step Size” data field, and type in a value.

Specifying Secondary Independent Variable Curves

This “# Of Curves” parameter value, minus one, defines the number of steps that will be taken from the “2nd Start Val” value for the secondary independent variable. This parameter and the “2nd Var Step Size” value define the final value given “2nd Ind. Var.” To enter a “# Of Curves” value, select the corresponding data field and type in a number.

Suppressing Function Scan Plots

When this analysis is executed, dependent variable values for all independent variables are printed to the analysis output listing file and a plot file is prepared. If you do not want to plot these data using the Easy5 on line plot program, select "No" following "Plot Results:" on the function scan data form.

Function Scan Analysis Method

Function scan calculations begin by setting the system states to the current operating point values, and all rates to zero. The system model equations are then evaluated. The specified independent variable, **INDEP1**, is set to its initial value, **START1**, and the model equations are reevaluated. If **INDEP1** is a state or parameter, the model equations are completely reevaluated. However, if **INDEP1** is a variable which would normally be calculated by the model equations, the reevaluation begins at the statement (in SUBROUTINE EQMO) immediately following the component that calculates the variable or rate. This process of reevaluation is repeated fifty times as the independent variable is scanned from **START1** to **STOP1**. After each reevaluation the value of **DEPEN** is recorded.

For a two dimensional function scan, **INDEP2** is set to its specified value before each scan of **INDEP1**.

Graphic Files, EMFs, and PostScript

The schematic block diagrams and plot data can be exported to a Window’s “enhanced meta file” (EMF) formatted file. EMF is a standard graphics format that can be directly imported into Word, PowerPoint and Excel.

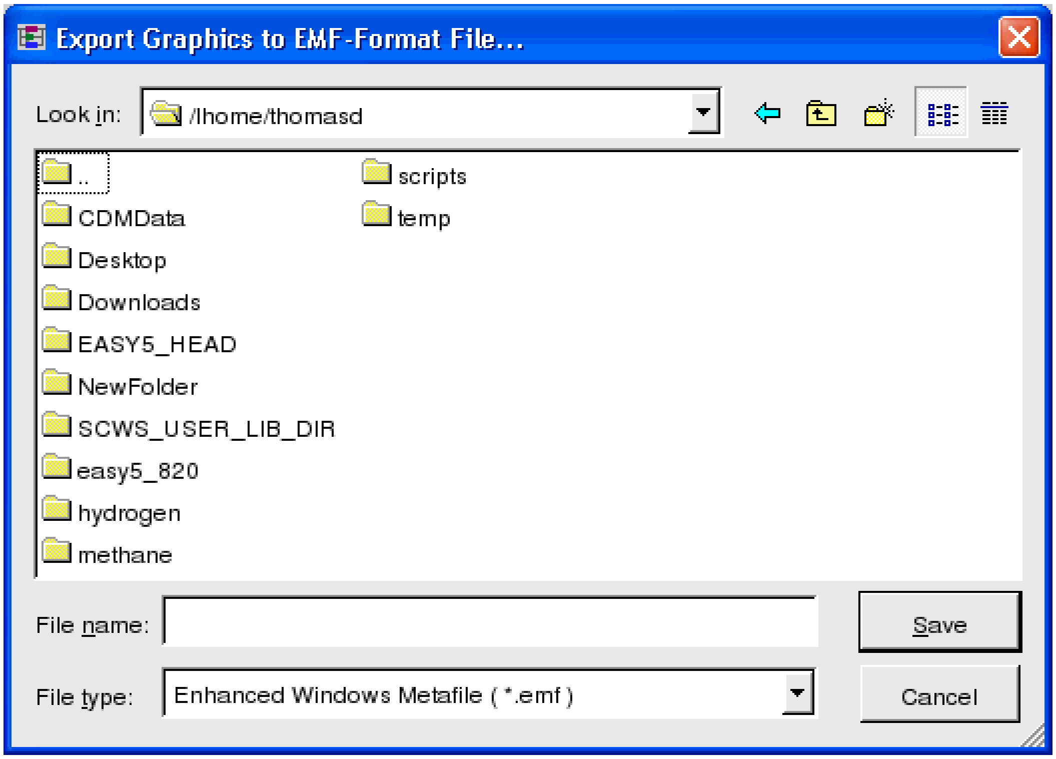
This allows you to easily output your Easy5 graphics into Word or PowerPoint, and most important, you can then edit your Easy5 graphics in these tools. You can annotate your Easy5 graphics, enlarge/decrease the graphic's size, change line colors, delete graphic elements, and edit text.

Generating the Schematic Block Diagram EMF Graphics File

The schematic block diagram is exported to external EMF graphic files using the **File > Export Schematic** menu options from the main menu bar. The following export options are available:

Export Menu Option	Action
Current View	Exports only the schematic block diagram currently displayed in the Easy5 window. You can zoom in/zoom out to get the desired view.
Current Schematic	Exports the current displayed schematic block diagram, but zooms in/out to fit the entire schematic in the window.
Entire Hierarchy	Exports the entire model and all submodels.

Selecting any of these options will open the dialog shown in Figure 65. In the bottom “Selection” input field, enter the root name of the output file. Easy5 exports the schematic with an “emf ” extension, named <file>.emf.


Export Graphics to EMF-Format File...

Look in: /home/thomasd

..
CDMData
Desktop
Downloads
EASY5_HEAD
NewFolder
SCWS_USER_LIB_DIR
easy5_820
hydrogen
methane
scripts
temp

File name:

File type: Enhanced Windows Metafile (*.emf)

Figure 65 Export to WMF Dialog

Selecting “Entire Hierarchy” exports a separate EMF file for each submodel. Easy5 automatically adds a file name extension for each submodel as: <file>.s<i>.emf, where "s<i>" designates a separate submodel, using i=1, 2, 3,...

If the model contains a top level schematic and two submodels, three EMF files are generated, named `schema_graphics.s1.emf, schema_graphics.s2.emf,` and `schema_graphics.s3.emf.`

Generating Plotter EMF Graphics File

Easy5 Plotter files are exported to external EMF graphic files using the **Export** menu options from the main plotter menu bar. The following export options are available:

Plotter Export Menu Options	Action
Export Selected Displays...	Exports only the displays selected in the “Displays” window pane. To select multiple displays, hold the <Ctrl> key and select one or more displays.
Export Entire Case...	Exports all the displays in the currently selected Case.
Export All Displays in File...	Exports the entire plot file and all the displays.

Easy5 exports the plotter display to a EMF file with "emf" extension, named <file>.emf. When you select Export Selected Displays, a dialog opens letting you specify the file that you want to export as EMF.

Selecting “Export Current Display” exports the plot from the current display to a single file. Selecting any of the other options exports more than one display to a separate EMF file for each display. In these multi-file exports, Easy5 automatically adds a file name extension for each display as: <file>`.d<i>.emf,` where "d<i>" designates a separate display, using i=1, 2, 3,...

In the example shown in Figure 65, if the plot file contains four displays and **Export All Displays** in File is selected, four EMF files are generated, with the following names: `schema_graphics.d1.emf, schema_graphics.d2.emf, schema_graphics.d3.emf` and `schema_graphics.d4.emf.`

Using EMF Graphics

EMF graphic files can be imported and edited in several word processing tools.

Importing Easy5 EMF Graphic Files

The EMF generated graphic files can be imported directly into Word, PowerPoint and Excel. To do this, select **Insert > Picture > From File...**, select the EMF file and select the `[Insert]` button. If you do not see the EMF file listed, make sure that the “Files of type” dialog includes "*.emf" as one of the options, or set this to “All Pictures”.

Editing Easy5 EMF Graphic Files

Once the file is imported into the document, you can edit the graphic. First, you may need to resize the image. To do this, select the image, and grab a corner and drag it to the desired size. The Easy5 EMF graphics is generated as a single “grouped image”. To edit the graphic image, you must first convert the image to an editable image. Double-click on the image. In PowerPoint, a dialog may appear, prompting the message, "This is an imported picture, not a group. Convert it to a Microsoft Office drawing?"; select "Yes". In Word, the graphic will be place in the Word graphics editor. Select the image, and then ungroup the image.

Every image will be converted as a separate vector graphic image. This allows you to edit every image. You can edit and change objects and text, change the color, width, delete, add, etc. Word's graphic editor is limited in features and functionality, and is not as good as PowerPoint's editor.

Overriding Hard copy and EMF Plot Curve and Grid Widths

By default, curve and grid widths will be 2 for printed and exported plots. You can override these defaults using the environmental variables EZ_HC_GRID_COLOR and EZ_HC-CURVE_WIDTH. For information on setting environmental variables, see "External (Environment) Variables".

Exporting Plot Files

You can also export plot data in various formats as described in the following table by selecting the Export Plot Data As.. from the **Plotter Export** menu.

Plot Export Menu Options	Action
ADAMS/PPT XRF File	Translates results into form readable by Adams/PPT.
Comma Delimited (CSV) File..	Translates results into generic data form.
ESA File...	Translates results into proprietary form (Boeing).

Easy5 files can be exported for use with ADAMS models. When you select this option, the dialog shown in Figure 66 appears.

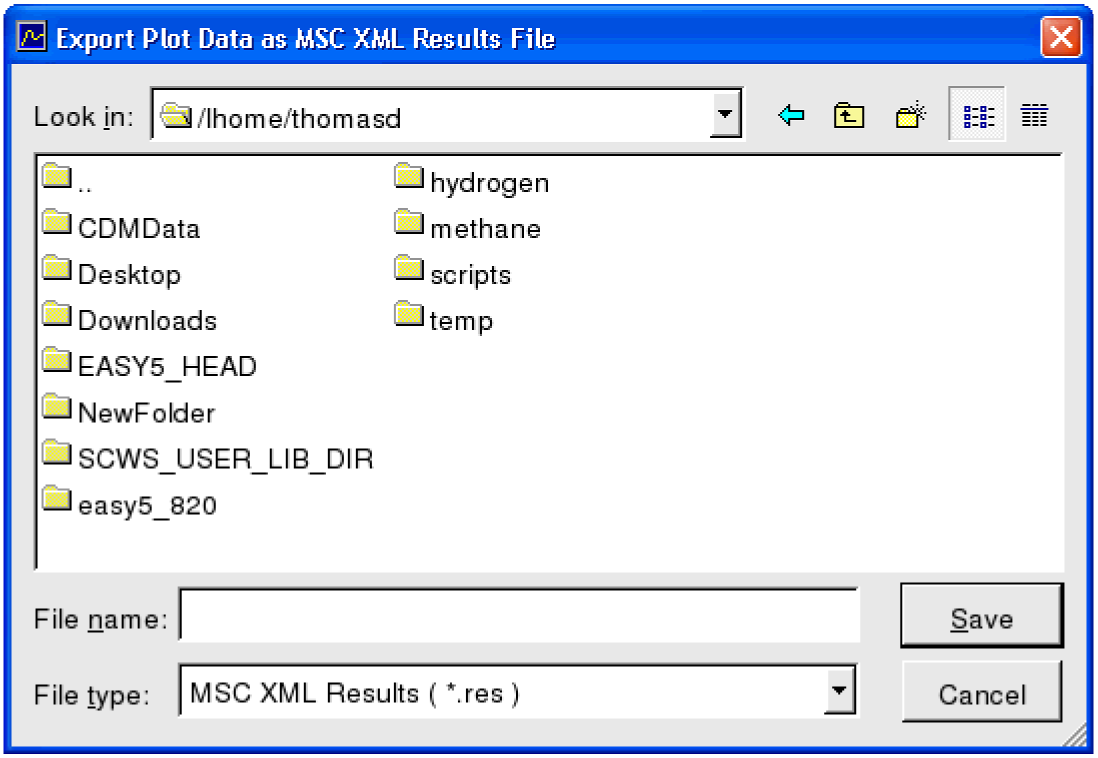


Figure 66 Export to ADAMS/PPT XRF File Dialog

EMF Graphic Editors

You may wish to edit an Easy5 generated EMF graphic before importing it into a document. There are many tools that can read EMF graphics, but do not edit the graphics and only allows you to save the graphics in a different format, such as GIF and TIF.

However, there are a few tools that may be used to edit the EMF graphic files. A list of a few tools are given. We do not promote these tools, but merely list them for your convenience.

Metafile Companion by Companion Software, Inc. - Metafile Companion (MC) is a Windows metafile editor. This is a full featured editor, that allows you to edit EMF files, and save the output in many different graphics formats. It costs approximately $29.00. For more information, visit their web site at: http://www.CompanionSoftware.com.

Paint Shop Pro by JASC Software, Inc. - Paint Shop is a good commercial tool used to read, edit and save EMF graphics. For more information, visit their web site at: http://www.jasc.com.

Importing a PostScript File Into a Document

The export of PostScript file to a document applies only to the *Linux* version of Easy5. The schematic block diagram can be exported to a PostScript file and imported into a document using any word processor that allows PostScript files to be imported. First, the schematic block diagram needs to be written into an appropriate PostScript formatted file.

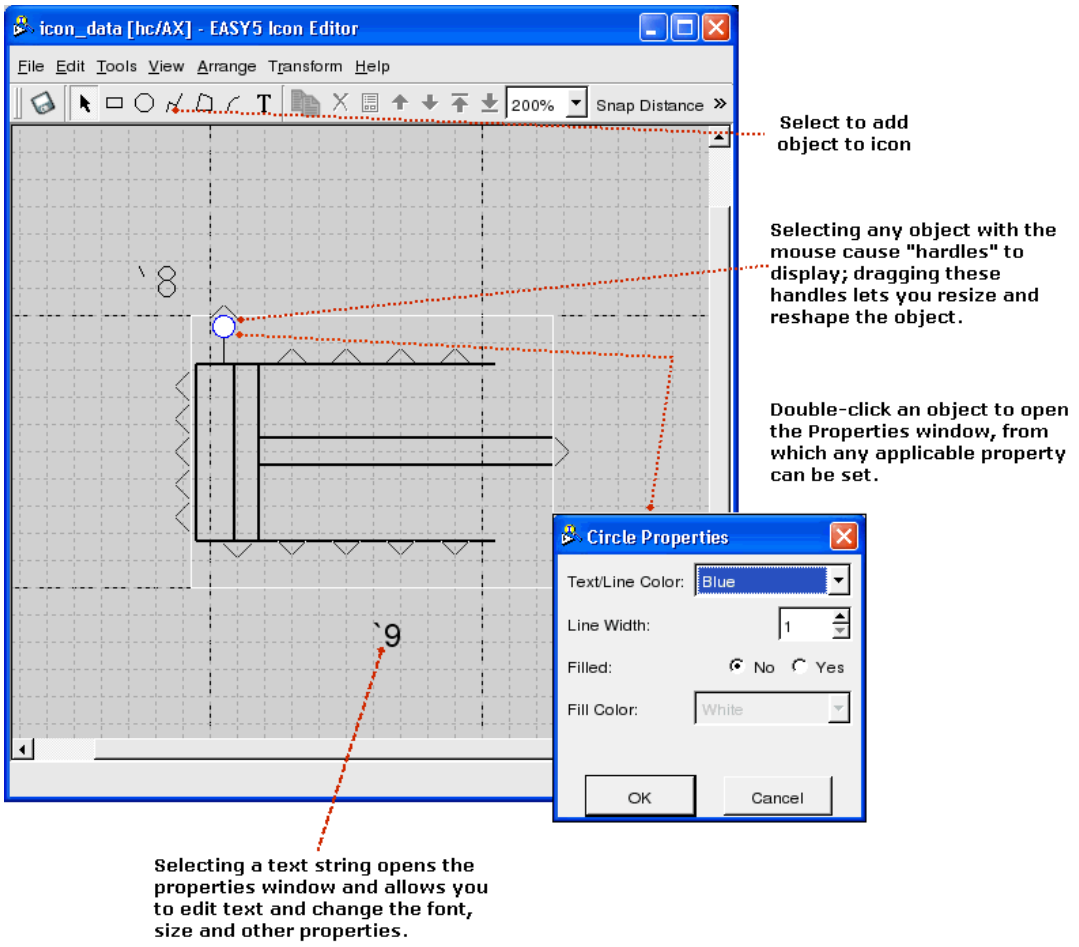
The following steps, which apply only to *Linux*, will setup the correct PostScript file:

1. Go to **File** > **Print**.
2. Select **Print to File** and enter the desired file name.

Note:	Easy5 writes the EPS file as a printable image, not as a bitmap image. As a result, when the PostScript file is imported into a document, the bitmap image cannot be seen; only a gray box is displayed. However, even though the image cannot be viewed, it does print correctly. If you have a PostScript previewer, you can view it.

Icon Editor

Please see the User Guide, Chapter 9 - Icon Editor for detailed information on how to use the Icon Editor.


icon_data [hc/AX] - EASY5 Icon Editor
File Edit Tools View Arrange Transform Help
200%
Snap Distance
Circle Properties
OK
Cancel

Text/Line Color: Blue
Line Width: 1
Filled: [✓ No] Yes
Fill Color: White

Select to add object to icon

Selecting any object with the mouse cause "hardles" to display; dragging these handles lets you resize and reshape the object.

Double-click an object to open the Properties window, from which any applicable property can be set.

Selecting a text string opens the properties window and allows you to edit text and change the font, size and other properties.

Figure 67 Easy5 Icon Editor

Implicit Model

See also: "Sort Blocks"

User Guide, Chapter 13 - Implicit Modeling

An implicit model is a model that contains an implicit loop. A model with an implicit loop cannot be built, and Easy5 will warn you that "errors occurred during model generation phase". You must resolve all implicit loops in your model before the model can be built. This section discusses what an implicit model is, and how to resolve implicit loops.

Definition of an Implicit Model

An implicit model is a model which contains an implicit mathematical relationship that results in an implicit loop. An implicit relationship occurs when variables are a function of each other. Figure 68 shows the most basic example of an implicit relationship: *x* is a function of *y*, and *y* is a function of *x*. In order to calculate *x*, you must know the value of *y*, but *y* requires that *x* be known.

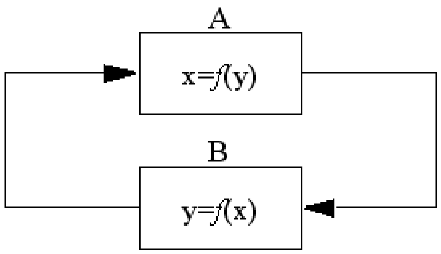


Figure 68 Example of an Implicit Relationship

Easy5 sorts components to build an "explicit" model. An explicit model is one for which all output variables are calculated before they are used as input to other components. Components are moved and put in a sequential order that results in an explicit model. When Easy5 detects an implicit loop, it attempts to break the loop by moving the components that cause the implicit loop. Easy5 will attempt to sort all components to form an explicit model.

However, there are cases in which implicit loops cannot be broken by sorting components. The example of Figure 68 is such a case. Moving component B before component A will not break the implicit loop. As a result, you must resolve this implicit loop. The sections that follow show several examples of implicit models and how to resolve the implicit loops.

Example of an Implicit Model

The most simple example of an implicit model is a model that is similar to the system shown in Figure 68. Figure 69 shows a similar implicit model built in Easy5. This models a simple controller. It has an command (AF component) and a feedback from the GF component, into a summation block (component SJ). The output of the SJ component is the controller error, which is input into the first order lead lag component LE. The output of LE is an *algebraic variable*, and is fed back into the summation component SJ.

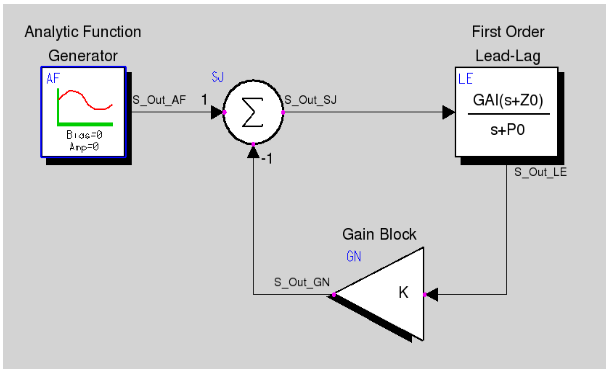


Figure 69 Example of an Implicit Model

To see why this results in an implicit model, write out the mathematical relationships as follows:

Equations:

S_Out_SJ = S_Out_AF - S_Out_GF

S_Out_LE = GAI_LE * S_Out_SJ + X1_LE

S_Out_GF = K_GF * S_Out_LE

Relationships:

SJ= *f*(AF,GF) but, GF= *f*(LE), and LE= *f*(SJ)

as a result:

SJ= *f*(AF,LE)

LE= *f*(SJ)

It is clear that the model in Figure 69 results in an implicit relationship that cannot be resolved. In general, an implicit model results from the lack of a state variable in a feed forward or feedback path. State variables are significant because their values are calculated by the integrator before the executable model (subroutine EQMO) is called, and as a result, they break the mathematical relationships in your system model.

Easy5 Model Generation of an Implicit Model

What happens when Easy5 attempts to resolve an implicit loop but fails? In the example shown in Figure 69, Easy5 will attempt to sort the components to break the implicit loop but will fail. During the "Create Executable" process, Easy5 will terminate the model building process and will warn you with the following message:

Errors occurred during model generation phase.

Whenever you get this message, you should open and analyze the model generation listing file (select the **Build > Display Model Generation Listing...** menu). If the Easy5 model builder failed to build an explicit model, an error message is given at the bottom of this file, telling you which connections form an implicit loop. For the model in Figure 69, the following message results:

```
*** FATAL ERROR ***   THE FOLLOWING CONNECTIONS FORM AN IMPLICIT LOOP. ALL CONNECTIONS
BETWEEN ANY TWO COMPONENTS IN THIS LOOP MUST BE REMOVED OR REPLACED BY STATE OUTPUTS.
IF OTHER IMPLICIT LOOPS EXIST, THEY CANNOT BE DETECTED UNTIL THIS LOOP IS REMOVED FROM
THE MODEL.
S_Out_SJ    FROM COMPONENT SJ    IS AN INPUT TO COMPONENT LE
S_Out_LE    FROM COMPONENT LE    IS AN INPUT TO COMPONENT GF
S_Out_GF    FROM COMPONENT GF    IS AN INPUT TO COMPONENT SJ
```

Example of an Explicit Model

Assume that the first order lead lag component LE is replaced by a first order lag component LA. This model is shown in Figure 70.

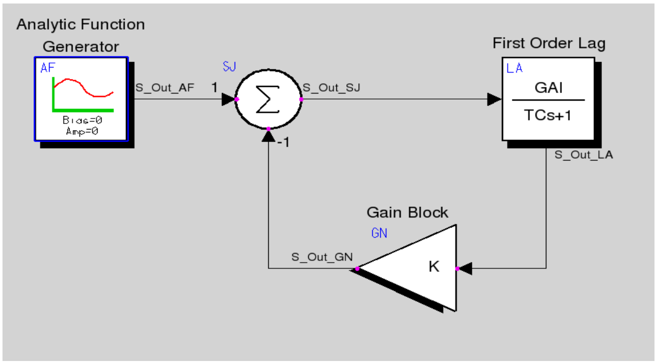


Figure 70 Example of an Explicit Model

The model generation program will detect an implicit loop, but in this case, it will be able to sort the components to form an explicit model. The equations and the mathematical relationships are similar to the previous example. *The main difference is that the LA lag component output, S_Out_LA, is a* ***state*** variable, *not an* ***algebraic*** variable!

States are the result of the integration of the rate vector, and therefore are always known quantities at all times. The state vector is passed to the model upon successful integration, and all state values are known at the beginning of the next iteration. As a result, the components that are a function of state variables can be sorted and moved anywhere in the model, and *can therefore break implicit loops*!

The model shown in Figure 70 will result in the model generator placing the components in the following order:

AF

GF{GF= *f*(S_Out_LA) where S_Out_LA= *state* breaking the implicit loop}

SJ{SJ= *f*(S_In_AF, S_Out_GF}

LA{LA= *f*(SJ}

The GF component can be moved to the top and placed before the SJ component. This is because the GF component is a function of a state variable, S_Out_LA.

Note:	You should make a habit of analyzing the executable source file. This file shows you how the components and equations are sorted to form an explicit model.

How to Break Implicit Loops

The following are general recommendation for breaking implicit loops. When you encounter an implicit loop, you should follow these recommendations in the order they are presented.

Correct Over-Simplified Model

In the physical world, there is no such thing as an implicit system; usually implicit system models arise during construction of simplified models of feedback control systems. Most control systems are made up of the following dynamics: plant, sensor, controller and actuator dynamics. During construction of a simplified model, one or more of these items may be omitted based on the assumption that they are too fast to be of any concern. The result is often an implicit model which assumes that information flows instantly around the system. This model is not only physically incorrect, but is also more difficult to analyze than a true model that includes all the dynamics. By restoring omitted system dynamic effects, implicit relationships can often be eliminated.

Modify Connections and Components to Break Implicit Loops

Easy5 will tell you which connections result in an implicit loop. Analyze your model and determine which components and connections cause the implicit loop. You may be able to use a different component to break the implicit loop. Components that typically cause implicit loops are as follows.

GT Component: This is a general transfer function component of order "n". The GT component output is a algebraic variable. If the GT component is causing the implicit loop, try using the GS component whose output is a state, or build the transfer function using a series of first and second order transfer function components.

SM Component: This component models a state variable model, with the output Y vector being algebraic variables. If the feed through matrix [D] is all zeroes, then, use the SR component in place of the SM component. This is a reduced format of the SM component which assumes that [D] is zero, and whose Y vector output is only a function of the state X vector. This will break the implicit loop.

Components with Hard Limits: Some components that model hard limits may cause implicit loops. An example is the *IH Integrator with Hard Limits* component. The clamped output signal (S_LimOut_IH) is an algebraic variable which outputs a signal with absolute accuracy during the hard limiting. However, if this causes an implicit loop, then delete the output connection and use the output signal (S_Out_IH) for connecting to other components. The S_Out_IH output is a continuous state which will break the implicit loop.

Use Sort Blocks to Break Implicit Loops

If a Fortran component or your own user-defined Library component is causing an implicit loop, you may be able to break this loop by breaking the code into multiple sort blocks. The components' entire body of code can be broken up into smaller blocks of code that can then be sorted by the model builder to break implicit loops. For information on how to use sort block, refer to "Sort Blocks".

Use Easy5's Implicit Modeling Feature

Easy5 has a built in implicit model solver which resolves an implicit loop by transforming the implicit relationship into a differential algebraic variable. If your model has an implicit loop which can not be resolved using the standard methods listed above, then use this feature to resolve the implicit loop. First you must turn on the implicit solver, before building the executable model. To do this, select: **Build > Solve Implicit Loops**. Then, build the executable model. When you are ready to perform a simulation, you must set the integration method to Radau54. This integration method is specific for solving the differential algebraic equations that results from solving the implicit loops. See User Guide, Chapter 13 - Implicit Modeling.

Transform Algebraic Variable into a State

If all else fails, you can resolve an implicit loop by transforming an algebraic variable into a pseudo state. To do this, insert a lag component (either the LA or LG component) in the implicit loop. This will transform the algebraic variable into a continuous state which can then be sorted and placed anywhere in the model.

For example, the model previously shown in Figure 69 is an implicit model which cannot be resolved using the recommendations given above. The only way to break this implicit loop is to insert a lag component after the lead lag component as shown in Figure 71.

The natural frequency of this lag should be made large compared to the other dynamic effects of your model, to minimize its effect on the model dynamics. As a rule of thumb, set the lag's natural frequency pole (=1/TC) 10 times higher than the highest dynamic pole.

For the example shown in Figure 71, if the lead lag pole (P0 LE) is set to 10 rps, then the lag pole should be set to 100 rps, or the time constant TC LA=.01.

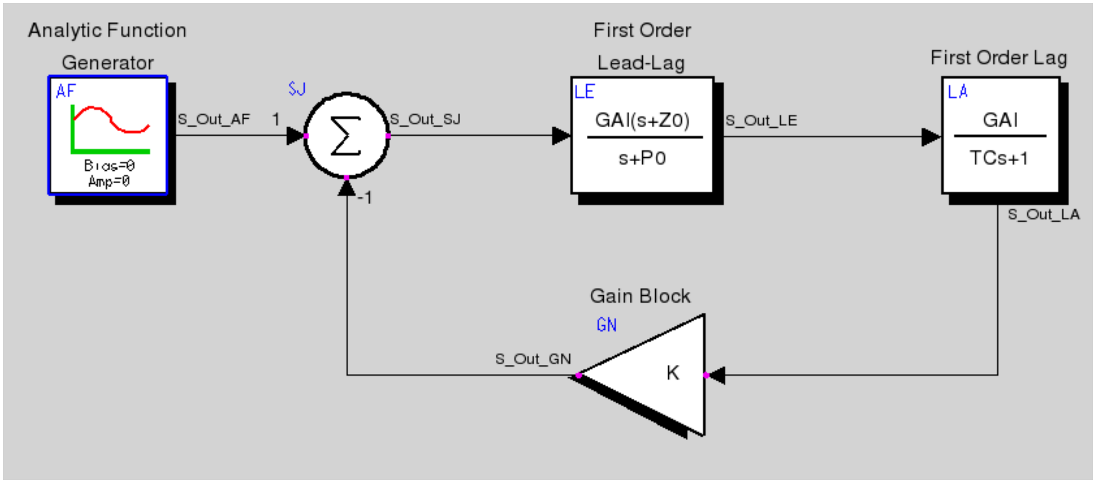


Figure 71 Example of using the LA Component to Break an Implicit Loop

Note:	Breaking an implicit loop using a lag component should only be done as a last resort. This method introduces a higher order dynamic mode and will result in longer simulation execution time. However, using a stiff integration method, such as BCS Gear, will make the effect of the added lag component less noticeable.

Initial Condition Calculation

See also: "Operating Point"

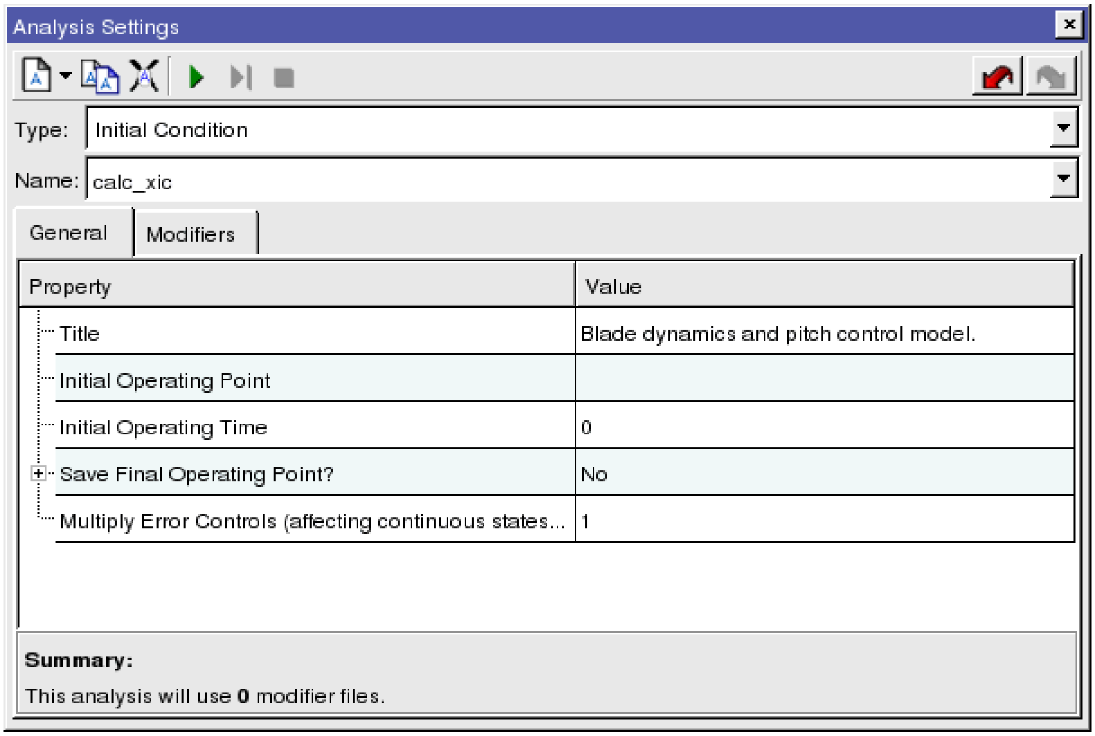
The Initial Condition Calculation analysis is used to calculate the model's initial condition. Initial conditions specify initial values of states for simulation and Steady-State analyses, and define the "operating point" for all other analyses.

The Initial Condition Calculation analysis executes the model equations one time to setup the initial conditions, and then updates the initial condition vector with the state values. With the state vector defined, Easy5 is ready to initiate any analysis.

The initial conditions should be calculated for all states in a model before any analysis is performed. As a result, Easy5 automatically calculates the initial conditions before performing a user requested nonlinear or linear analysis.

For example, when you submit a simulation, even though you do not request the calculation of initial conditions, Easy5 automatically calculates the initial condition vector, and then executes the simulation analysis. The exception to this rule is with Multiple Analysis. The initial conditions are not calculated at the start of the Multiple Analysis, and therefore, you should add "Calc. Initial Conditions" as the first entry in the Multiple Analysis data form.

You may wish to calculate the initial conditions without any other analysis, to obtain a snapshot of your model data. This can be performed by setting up and executing the Calculate Initial Condition data form as shown in Figure 72.



Type: Initial Condition

Name: calc_xic

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Save Final Operating Point?	No
Multiply Error Controls (affecting continuous states...	1

Summary:

This analysis will use **0** modifier files.

Figure 72 Calculate Initial Condition Data Form

To open this data form, select **Analysis > Miscellaneous > Initial Condition Calculation...**This data form is simple and contains the general data input fields as described in "Analysis Data Form".

This data form also contains the [Execute/Debug] pushbutton that allows you to run this analysis with the symbolic debugger. See "Debugging the Model and Analysis" for additional information on this feature.

The results of this analysis are contained in the output listing file. The initial condition values at the user specified time and operating point will be listed in this file.

Initialization Statement

Component initialization is generally done using tests on INCALL > 0. This works fine but causes many tests to be made on each call to the model. By using initialization blocks, all the initialization code is grouped together at the top of the model and is activated by a single test on INCALL > 0. The initialization statement will usually be more efficient both in amount of code in the generated model and the execution time.

Code initialization is defined using `BEGIN INITIALIZATION` and `END INITIALIZATION` statements, and inserting the initialization code between these statements. For example, suppose in a Fortran code component, you need to set the variable `xyz` to zero at the beginning of each analysis. You would do so using the following code:

```
Begin Initialization
        xyz = 0.0
End Initialization
```

The initialization code is only used in Fortran and Library components. The statement is an Easy5 statement, and therefore, can be in upper and/or lower case, and can begin in any column.

Note:	If you have initialization code which should be executed only once on the very first call of the model, you will still need to enclose that code in an "INCALL .EQ.2 if block, but it will still be more efficient to enclose the code in an initialization block.

Caution:	Do not include any inputs that are being "driven" by (i.e. connected from) other quantities in your model equations -- incorrect results would occur. Placing any code inside an Initialization block forces it to be sorted to the beginning of your model code, and does not allow it to be calculated by the model later.

Integration Methods

See also: "Integration Method Selection Guidelines"
"Simulation Troubleshooting"
Ap. B: Guide to Numerical Integration

Easy5 uses a central integrator to calculate the states. The executable model calculates the rate of all states and passes the rate vector to the central integrator. The integrator then integrates the rates to calculate the new state values. There are seven types of integration methods available to the user. These methods are discussed in this section.

The Integration Method

During a simulation, values for all states in your model are calculated by your model and the Easy5 integration algorithm you select, as shown in Figure 73.

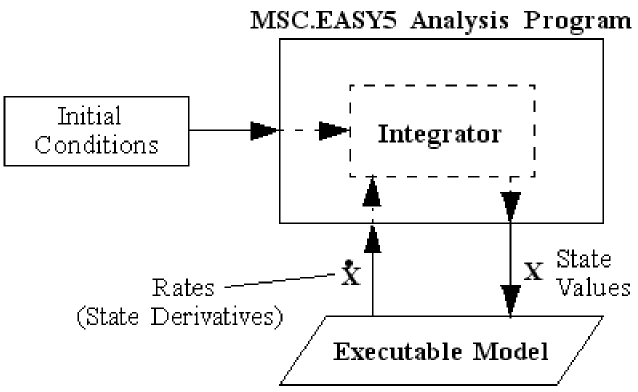


Figure 73 Integration Process During a Simulation

At the **INITIAL TIME** of the simulation, the set of initial conditions for all states is passed to the integrator for initialization purposes and then passed on to your executable model. The executable uses these initial state values to calculate the rates of all states at the **INITIAL TIME**, based on your model equations.

These rates are passed back to the integrators, and a new set of state values is calculated for the next time step. This process is repeated for every time step in the simulation to produce time history plots of the dynamics in your model.

Integration Methods Available

The Easy5 program offers you a choice of seven different integration algorithms, listed in the table shown below. All the methods can be used for sampled data systems. A brief discussion of each method and some guidelines for method selection follow. See Ap. B: Guide to Numerical Integration for more information on numerical integration.

Table 1-18 Integration Methods

Name	Method	Order	Type	INT MODE
BCS Gear	BCS Modified Stiff Gear	Variable	Variable Step/Implicit	1
Runge-Kutta	Variable-Step	4th	Variable Step/Explicit	2
Huen	Fixed-Step Huen Method	2nd	Fixed Step/Implicit	3
Euler	Fixed-Step Euler	1st	Fixed Step/Explicit	4
Adams	Adams-Bashforth Predictor Adams-Moulton Corrector	2nd-12th	Variable Step/Explicit	5

Table 1-18 Integration Methods

Name	Method	Order	Type	INT MODE
Stiff Gear	Hindmarsh version of Gear	Variable	Variable Step/Implicit	6
Radau54	Radau 5-4	5th	variable step three stage fifth order implicit Runge Kutta	7
Fixed-Step RK	Fixed-Step Runge-Kutta	4th	Fixed Step/Explicit	8
User-Defined	N/A	N/A	N/A	9

Definition of Terms

Several terms shown in the table above need defining so that the integration techniques can be fully understood. Terms discussed in the following subsections are: order, fixed and variable step types, and implicit/explicit methods. *INT MODE* refers to the internal analysis command value.

Order

The order of a particular method indicates the number of times your model equations must be executed for a successful integration time step. Thus, a second order method makes two calls to the model, while a fourth order method makes four calls per time step. In general, the higher the order, the better the stability and accuracy for a given time increment, but also the more model equation evaluations and the more computer time required per integration step.

Fixed and Variable Step Methods

Two kinds of step size selection methods exist: fixed and variable step. Fixed step methods use a time step that does not change during a run. In contrast, variable step methods continuously adjust their internal time step to fall within an error tolerance defined by the error control values in your model.

Fixed step methods, while requiring fewer calls to the model for a given time step, are restricted to the same time step (usually small to maintain accuracy and stability) during the entire simulation.

Thus, fixed step methods often require more total model equation evaluations (or model calls) over an entire transient than are required for variable step methods. Fixed step integration is depicted in Figure 74.

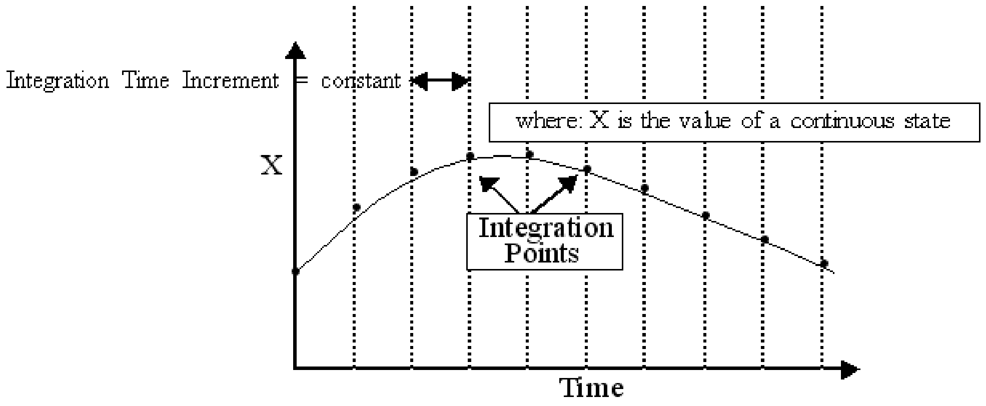

Integration Time Increment = constant
where: X is the value of a continuous state
X
Integration Points
Time

Figure 74 Fixed-Step Integration Method

Variable step methods, because they continuously adjust their time step, occasionally exhibit the problem of "grinding" the step size so small, to satisfy their error tolerance, that they take much longer to complete a transient. However, for most models variable step integrators will provide you with efficient and precise integration and are the methods of choice.

Figure 75 depicts variable step integration. Variable step methods are sensitive to model discontinuities as they try to reduce their step size to smoothly "cross" a discontinuity. Easy5's switch state components provide an efficient medium for handling model discontinuities of this kind.

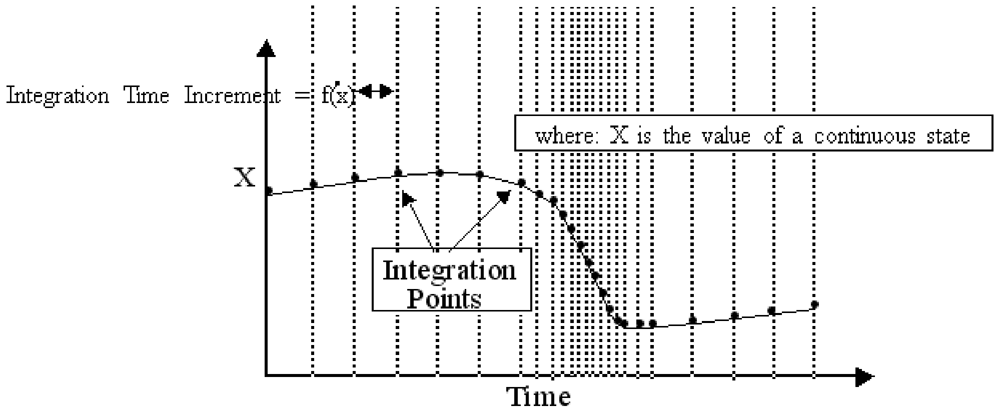

Integration Time Increment = f(x)
where: X is the value of a continuous state
X
Integration Points
Time

Figure 75 Variable-Step Integration Method

Implicit and Explicit Methods

Integration methods may be categorized as implicit and explicit. An explicit method uses an integration technique that uses the current rate to estimate the value of the state at the next time step. An implicit method, on the other hand, uses the rate at the current as well as the next time step, which is unknown, to estimate the same value.

Hence the name: the equation is an **implicit** one, implying an iterative solution. By their nature, implicit methods provide greater stability. However, for a given time step an implicit method requires more execution time than an explicit method because of the iterative nature of the solution required.

On the other hand, explicit methods cannot always ensure stability for a given time step. Depending on your model and your application, one method may be better than the other.

The BCS Gear Method

This variable step, variable order, implicit method is the default. Boeing Computer Services improved the Hindenhairs-Gear method to speed up its use of linear algebra. Its initial time step is set to **TINC INCREMENT**/100, and it will adjust its step size and move time forward and/or backward in its solution process.

This method was designed for so called "stiff" systems, in which there is a wide range of frequencies or eigenvalues, to effectively ignore high frequencies which would otherwise force you to use an extremely small step size with other methods. A real advantage to using BCS Gear is that it can take time steps up to 10 times larger than your value of **TINC** enabling it to make very fast progress through areas of your transient where system rates are relatively small. This method should be used with most types of models, especially for stiff systems. However, due to its stability region, *BCS Gear should be avoided for very lightly damped systems*. It can be used with sampled data systems where the internal time step will have its upper limit "clamped" at the fundamental sampling rate.

The Runge-Kutta Methods

This is a variable step, fourth order, explicit method. The integration step size is automatically adjusted to maintain error tolerances established by error controls. Accuracy is achieved by evaluating the derivative function, f(x,t), at values of time between a full time interval. The full step size is reduced until the difference between the states, as estimated by a full step and two successive half steps, is within the error tolerance. We suggest that this method be used after you first try the BCS-Gear or Adams methods. While it is the most robust method, you may pay for this with added execution time. A fixed step, fourth order, explicit Runge-Kutta integration algorithm is also available.

The Huen Method

This is a fixed step, second order, implicit method. It exhibits better stability and accuracy properties than the Euler method over a given time step. It has also been given other names: Second Order Runge-Kutta or Modified Euler. However, it requires two model evaluations per time step. In general, we recommend using this method over the Euler method.

The Euler Method

This is a fixed step, first order, explicit method. As the most simple integration technique, it is often used for debugging purposes or when a fixed step integrator is required.

The Adams Method

This is a variable step, variable order (2 through 12), explicit method and is well suited for systems exhibiting low damping.

The User-defined Method

You can use your own integration algorithm during a Simulation analysis. To do this, you must supply an integration subroutine for Easy5 to call when continuous states are to be advanced from the current point in time to the next point.

The user-defined integration subroutine, or the subroutine which calls your integration routine, must be named EZUSRI. The calling sequence of EZUSRI supplies the current time point and the time increment to the next time point. You must supply an integration algorithm that advances the state(s) through the time increment. Figure 76 shows a template for the user-defined subroutine, including the definitions of the calling sequence arguments.

The EZUSRI template is stored in the Easy5 scrlib folder, gp subfolders

<Easy5_home>/easy5/scrlib/gp

as the file `ezusri.f`.

Edit the `ezursi.f` to match the general format and content of Figure 76 and add your integration code to it. Then compile it and link it before creating the executable model.

```
        SUBROUTINE EZUSRI (TIME,TINC,CPUSEC,N,ISTOP)

      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 TIME,TINC,CPUSEC
      INTEGER N,ISTOP
      COMMON /CX    / X(1)
      COMMON /CXDOT / XDOT(1)
      COMMON /CWORK / A(1)
      COMMON /EZDEEF/ EZMDDF,HXNEXT,TEVNXT,IEZGAP,IEZDCS,ISWUPD
      COMMON /CORDER/ NOX,NOV,NOP,NOXN,NOVN,NOPN,NOD,NOS,NODN,NOSN,
     1                NOXT,NOXNT,IDSCRT,NOSW,NOSWN,NOXX,NOXXN
      SAVE CPU
      DATA CPU /-1.0/

C --- Save time at start of step
      STIME = TIME

C --- Initialization - Guess past state using current rate
      IF ( CPU .EQ. CPUSEC ) THEN
         DO 100 I = 1, N
100         A(I) = X(I) - TINC*XDOT(I)
      ENDIF
C --- Extrapolate to get predicted state and save current state
      Do 200 I = 1, N
         TEMP = X(I)
         X(I) = 1.5*X(I) - 0.5*A(I)
200      A(I) = TEMP

C --- Evaluate rate at end of step using predicted state
      TIME = TIME + TINC
      CALL EQMO(TIME,TINC,0)

C --- Correct the state vector using the new rates
      DO 400 I = 1, N
400      X(I) = X(I) + TINC*XDOT(I)

C --- Update the switch states if necessary
      If ( EZMDDF .GT. 0 ) THEN
         DO 500 I = NOXT+1, NOXX
500         X(I) = XDOT(I)
      Endif
C --- Restore saved time value
      TIME = STIME
      RETURN
      END
```

Figure 76 EZUSRI Template for User-Defined Integration Routine

Integration Method Selection Guidelines

See also: "Integration Methods"

"Simulation Troubleshooting"

Ap. B: Guide to Numerical Integration

In Easy5, the effect of integrator step size and integration method can be quickly evaluated. This is done by repeating a typical simulation run with different integration methods, error controls, or step sizes, and plotting the Easy5 reserved word `CPUSEC` versus `TIME`.

Such an evaluation should be made before expending a large amount of computer time on simulation studies of an unfamiliar model. Easy5 provides the BCS-Gear method, as a default. This method was chosen since it is very good with stiff systems, and since stiff systems are quite common in the areas to which Easy5 has been applied.

It is good practice to use the linearized analysis capabilities of Easy5 to investigate model stability and characteristics before attempting simulation. The following are some general integration method selection guidelines:

1. If no special knowledge is available about the system, try the variable step Runge-Kutta method. This method usually performs quite adequately on a broad range of problems. However, it will probably be worthwhile to consider switching to another method when more is known about the system, since Runge-Kutta is by no means the most efficient method.
2. If a large amount of output is desired with small time increments, the Adams or Stiff Gear methods should be considered. These methods use interpolation rather than generating smaller time steps when output points are smaller than current step sizes. However, these methods are adversely affected by discontinuities such as sampling, nonlinearities in the model, or external disturbances to the model.
3. If the model involves discrete dynamic elements (sampled data), small time steps will be required for the entire duration of the run. In these circumstances, it's more cost effective to use one of the fixed step methods because they require less computing per step. However, with these methods, the user must specify the size of the time step increment (TINC) and is responsible for insuring that the integration algorithm remains stable.
4. If the model contains non differentiable nonlinearities (e.g., coulomb friction, deadband, etc.) not implemented using switch states, the Runge-Kutta method is recommended. With the Runge-Kutta integrator, care should be taken if a large number of output data points at small time increments is desired. If the method is forced to reduce the step size in order to accommodate the data requirements, it could become significantly more costly than a fixed step method.
5. If the problem is stiff (i.e., there is a large spread in eigenvalues where the high frequency eigenvalues are well damped), either BCS-Gear or Stiff Gear is recommended. Both of these methods employ the same basic algorithm. However, BCS-Gear makes use of state of the art numerical techniques which increase its efficiency.
6. If the system has high frequency eigenvalues that are lightly damped (i.e., flexible modes), then the Adams method is recommended.

7. If the system is defined as an implicit model, then you must select the Radau54 integration method, and prior to building the executable model, you must turn on the implicit solver by selecting: **Build > Solve Implicit Loops**. For complete information on implicit modeling, see the User Guide, Chapter 13 - Implicit Modeling

It should be noted, however, that problems with large eigenvalues (with negative real parts) do not automatically indicate that one should use Stiff Gear. For example, consider the system:

$$\begin{aligned} \dot{X}_1 &= -X_1 \\ \dot{X}_2 &= -1000X_2 \end{aligned} \qquad \text{for time } 0 \le t \le \text{TMAX} \tag{1}$$

This is an uncoupled system (and might seem artificial), but coupled systems often display the behavior of rapidly damping components such as X_2. If one was integrating this model and the important variable was X_1 and TMAX was large, then a large step size could be used, provided the numerical integration of X_2 was damping to zero (i.e. stable). In such a case, the Stiff Gear method would be appropriate.

In contrast, if TMAX was small, such as TMAX = 0.0001, and X_2 was the component of interest (where relative accuracy is important), then an efficient integrator such as Adams, or perhaps Runge-Kutta, would be appropriate. Therefore, the decision to use Stiff Gear depends on both the user requirements for accuracy and the eigenvalues of the system.

Guidelines for Setting Error Controls

A quantity known as an "error control" is associated with every state in your model. Error controls are used in two ways by the Easy5 program: to determine an integration step size control for variable step integration during simulation, and as a perturbation step size during linearized analyses.

Integration Step Size Control

Variable step numerical integration techniques use the error controls in selecting a step size. In general, the smaller the error controls, the greater the accuracy of the numerical integration, but also the longer the execution time. Likewise, as the error controls become larger, less accuracy is achieved during numerical integration. Based on this generality, it would seem that you should set the error control as large as possible, while still maintaining a desired level of accuracy, in order to minimize the amount of CPU time expended. In practice, however, this is not always true.

By "easing off" on the error control values you may introduce "accepted" errors into the model which, while accepted for the integration of one state, may prevent another state from satisfying its error criteria. This will not only result in inaccurate integration, but the simulation will also exhibit very poor performance; it might not even make any progress at all! This type of poor performance often occurs in highly nonlinear systems where the sensitivity of perturbing certain states is very high with respect to the other states in the model. Usually, the error controls should be "tightened up" (made smaller) for such states. You should keep decreasing the error control until you reach a point of diminishing return, i.e., when no more gains in accuracy and execution time are noticed.

You may have to experiment with several short simulation runs to find the "optimal" values for the error control. Then, for longer transients, you can be assured that your simulation will run in an efficient manner.

As a general rule, the default value of 0.001 (for non-switch state and 1.E-6 for switch state) error control values should provide you with satisfactory performance during simulation. For further discussion of numerical integration techniques and error controls, see Ap. B: Guide to Numerical Integration.

Note:	Please exercise care when changing error control values, either individually or globally. These settings control the accuracy and performance of your simulation. Large changes in error controls may result in incorrect results and/or extremely long solution times.

Setting Error Controls

Error controls are altered from their current values when new values are entered in the appropriate component data tables. Values will be typed as double precision real numbers with a maximum of 7 significant digits. The default value is 0.001 (for non-switch states, and 1.e-6 for switch states). You may also enter the data in exponential format.

Setting Global Error Controls

The error controls are normally set on a state-by-state basis. However, you can easily adjust error controls affecting continuous states by setting the "**Multiply Error Controls**" analysis setting, as shown in Figure 77.

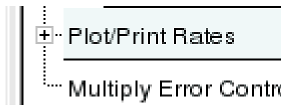


Figure 77 Multiply Error Controls Setting

Another method is to use appropriate analsyis commands to globally increase or decrease the size of affected error control values. To use this approach, create an auxiliary input file that contains the following command:

```
MULT INT ERROR BY = multiplier affects only continuous states
```

Likewise, there are additional analysis commands that allow you to globally control switch states, and/or all states, using the commands:

```
MULT SW ERROR BY = multiplier affects only switch states
MULT ERROR BY = multiplier affects all states
```

These analysis commands (or analysis form setting) causes each affected element of the error vector to be multiplied by the value "*multiplier*". For example, to decrease all continuous state error controls by a factor of 100 (*.01), use:

```
MULT INT ERROR BY =.01
```

The `MULT INT ERROR BY` analysis command can be very useful when adjusting the relationship between CPU time and numerical accuracy in your model. The **BCS GEAR**, **STIFF GEAR**, and **ADAMS** integration methods all use an "overall error control value", calculated as an "average" of all the **ERROR CONTROL** values. Thus, this command, which most easily changes this "average" value, is best suited for investigating the effect that changing error controls has on your simulation accuracy and performance (for these integration methods). For the **RUNGE-KUTTA** method, however, the individual **ERROR CONTROL** values are used for each state.

Interactive Simulation

See also: User Guide, Chapter 4 - Interactive Simulation

You can perform a simulation in an interactive mode during which model data can be changed, and plots displayed as the simulation progresses. Interactive simulation is useful because you can monitor results and change input parameters while the simulation is running, and stop it if the results are undesirable.

Components from the **is** (interactive simulation) library are used to setup the interactive simulation. This library contains interactive widgets for generating X-Y plots, strip charts, and input widgets that allow you to interactively vary different parameters and variables during a simulation. For example, while the simulation is running, you can vary a compensator gain using a “slider” widget, and then see the effect it has on your system with data plotted in a strip chart.

For complete information on how to setup and run an interactive simulation, refer to the User Guide, Chapter 4 - Interactive Simulation.

Linear Model Generation Analysis

See also: "Analysis Data Form"

"Linear Model Generation Method"

The generation of a linear model from a (generally) nonlinear model is the foundation for all of Easy5's linearized analysis tools. As such, this analysis can be very helpful in analyzing your model or the results from other Easy5 analyses. Because it is created automatically about any operating point, you need not maintain separate versions (linear and nonlinear) of your model. It is a very fast and inexpensive analysis to perform.

You should use this analysis as a complement to simulation. It provides a vitally needed picture of the system stability at a simulation starting point and is the basis for selecting integration methods and time increment values for fixed step integration methods.

Types of Linear Model Generation Analysis

Two types of linear models can be generated, depending on what you want. A simplified form linear model, used only to calculate the Jacobian and eigenvalues, is the fastest and most commonly used form of this analysis. The full form linear model generates the full set of systems matrices. This is described in the following section.

The Simplified Form Linear Model

The Linear Model Generation analysis calculates a linear approximation of your nonlinear model about any operating point, x_0. The linear model is in the simplified form:

$$\dot{x} = Ax\big|_{x = x_0}$$

where:

A = nxn stability matrix (or Jacobian)

x = state vector of length n

$\dot{x}$ = rate vector of length n

This model calculates the stability matrix, or Jacobian, of the linearized system model. This is used to calculate the eigenvalues of your system. This analysis can be performed on both continuous and sampled data models; it automatically selects the appropriate analysis method. This form requires no specification of any inputs or outputs. Eigenvalues are calculated as specified by the user in the Linear Model Generation data form.

The Full Form Linear Model

A full form linear model can be generated about any operating point, x_0, by specifying a set of inputs, **u**, and a set of outputs, **y**. A linear model will be created of the form:

$\dot{x} = Ax + Bu + K_x$

$y = Cx + Du + K_y$

where:

A = nxn system Jacobian (stability matrix), where n = the number of states

B = nxm system input (or control) matrix, where m = the number of inputs

C = pxn system output matrix, where p = the number of outputs

D = pxm direct transmission gain matrix

K_x = constant rate vector

K_y = constant output vector

u = system input vector of length m

y = system output vector of length p

x= system state vector of length n

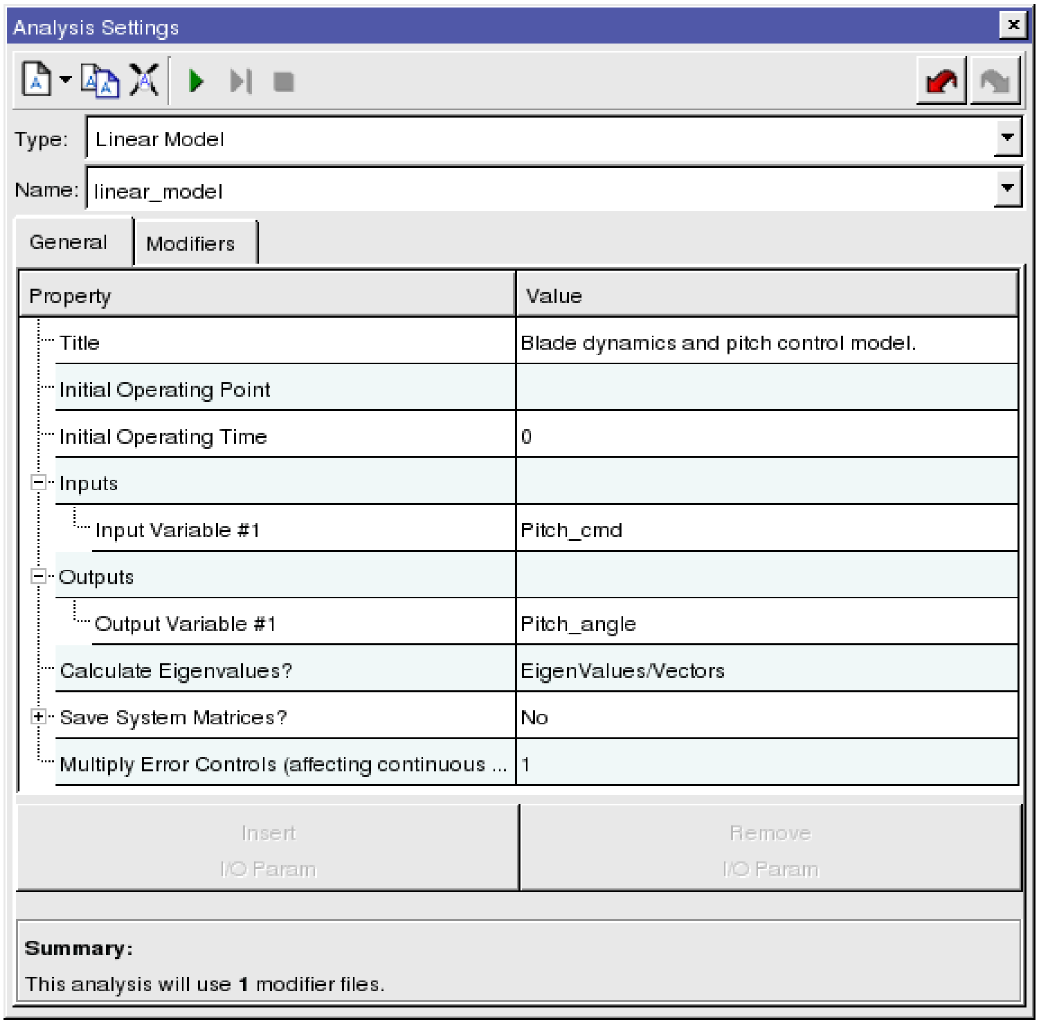
$\dot{x}$ = system rate vector of length n

The two constant vectors K_x and K_y are calculated so that the rate, x_0, and output, y_0, match the corresponding terms of the nonlinear model at the chosen operating point, x_0, and u_0.

If you choose to generate this form of the linear model, and choose to calculate eigenvectors, the eigenvectors, or modal matrix, of the system will also be calculated, along with the "observability" matrix (the B matrix in modal coordinates) and the "controllability" matrix (the C matrix in modal coordinates).

Setting up a Linear Model Generation Analysis

All specifications for performing a linear model generation analysis are defined in the Linear Model Generation Analysis Data Form. To access this data form, select **Analysis** from the main menu bar, then select **Linear**, and then **Linear Model Generation...** or press F9. Figure 78 shows the linear model generation data form.



Type: Linear Model

Name: linear_model

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Inputs	
Input Variable #1	Pitch_cmd
Outputs	
Output Variable #1	Pitch_angle
Calculate Eigenvalues?	EigenValues/Vectors
Save System Matrices?	No
Multiply Error Controls (affecting continuous ...	1

Summary:
This analysis will use **1** modifier files.

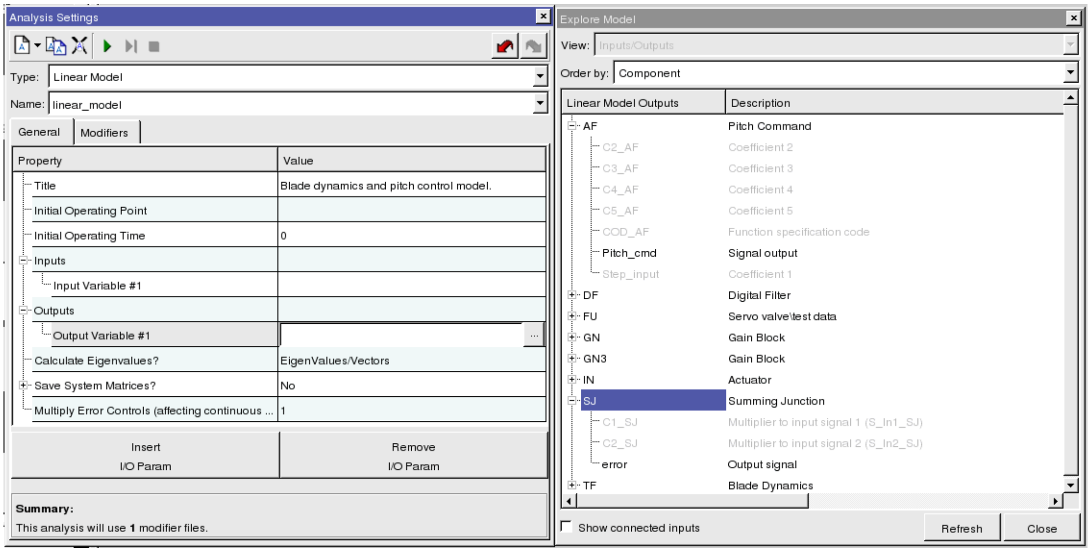
Figure 78 Linear Model Generation Analysis Form

This figure shows all the options and data fields that could possibly appear on this form. The sections that follow describe how to fill in this form. To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to "Analysis Data Form".

Defining Linear Model Inputs

The default setting for model Inputs and Outputs is blank (None). This results in the calculation of only the Jacobian A matrix. However, if you define a set of linear model inputs (and outputs) as part of the Linear Model Generation analysis data, the full form of the linear model (i.e., the A, B, C, and D matrices) will be calculated. Any set of parameters, variables, and/or states can be defined as inputs.

The linear model input variables data form is used to define the set of linear model inputs. You can enter Easy5 parameter, variable, or state names directly in this form by selecting a value field and typing in the data. Or you can use the Model Explorer "pick" feature. To do this, click on the input value field where you see an elipsis. This opens the Model Explorer window as shown in Figure 79, which displays a list of possible input names for the selected component. Select a name from this list and Easy5 copies it into the analysis form.


Analysis Settings

Type: Linear Model
Name: linear_model
General | Modifiers

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Inputs	
Input Variable #1	
Outputs	
Output Variable #1	...
Calculate Eigenvalues?	EigenValues/Vectors
Save System Matrices?	No
Multiply Error Controls (affecting continuous ...	1

Insert I/O Param | Remove I/O Param

Summary:
This analysis will use **1** modifier files.

Explore Model

View: Inputs/Outputs
Order by: Component

Linear Model Outputs	Description
AF	Pitch Command
C2_AF	Coefficient 2
C3_AF	Coefficient 3
C4_AF	Coefficient 4
C5_AF	Coefficient 5
COD_AF	Function specification code
Pitch_cmd	Signal output
Step_input	Coefficient 1
DF	Digital Filter
FU	Servo valve\test data
GN	Gain Block
GN3	Gain Block
IN	Actuator
SJ	Summing Junction
C1_SJ	Multiplier to input signal 1 (S_In1_SJ)
C2_SJ	Multiplier to input signal 2 (S_In2_SJ)
error	Output signal
TF	Blade Dynamics

Show connected inputs | Refresh | Close

Figure 79 Linear Model Generation Analysis Form with Model Explorer

Defining Linear Model Outputs

If you want to generate the full form of the linear model, you must define a set of inputs and a set of outputs. The outputs can be any set of states or variables in your model. Define linear model outputs in exactly the same way as the set of inputs. Use the Outputs field in the linear model data form located below the Inputs field.

Partial Form Linear Model

If you have not specified either the input or the output vector, the full form linear model cannot be created. However, a partial form linear model, depending on whether the input or output vector was omitted, can be created. That is, if you have omitted the input vector, only the **A** and the **C** matrices can be calculated (the **B** and **D** are undefined) as shown below:

$$\dot{x} = A$$
$$y = Cx$$

Likewise, if you have omitted the output vector, only the **A** and **B** matrices will be calculated as shown by the following form for the linear model:

$$\dot{x} = Ax + Bu$$

Controlling the Calculation

Controlling the linear model generation calculation may be useful especially for models with large numbers of states (n>500), as the linear model generation computational time increases approximately with n-cubed! The additional computational effort for eigenvalues and eigenvectors depends largely on how diagonal your A matrix is -- so it can be significant. You also may not be interested in the eigenvalue or eigenvector results at all.

Thus, you may control the linear model generation calculation by selecting the "`Calculation:`" options settings. Possible values are:

- **Eigenvalues/Eigenvectors** - both eigenvalues and eigenvectors are calculated (default)
- **Eigenvalues only** - no eigenvectors are calculated
- **Neither** - neither eigenvalues nor eigenvectors are calculated

Note: Controllability and observability matrices, per the appropriate linear model form, require calculation of eigenvectors.

Saving the Linear Model System Matrices

You may save the linear model system matrices and import the data into the *Easy5 Matrix Algebra Tool (MAT)*. If you wish to save the system matrices (A,B, C and D) in a special file, select "Yes" following the "Save System Matrices." A "Save As:" input field will appear requesting a filename. Select the input field and enter a filename.

This file is written using the auxiliary input file format, and only contains the system matrices **A, B, C, D, K_x** and/or **K_y** matrices, depending on the form of the model.

In the Matrix Algebra Tool, you can load the linear model data from this file by entering a simple command. The data from this file can also be loaded back into Easy5 models using the **SM** (System Matrix) or **SR** (System Matrix / Reduced Format) components.

Linear Model Generation Method

A Linear Model Generation analysis may be requested on either continuous or sampled data models. Easy5 detects the type of your model and selects the correct analysis technique. Whenever one or more discrete states (delay or sample state) is active anywhere in your model, the whole model is treated as a sampled data model. The following sections discuss the theory behind the Linear Model Generation analysis.

See also: "Linear Model Generation Analysis"

Continuous Systems

A nonlinear system in state space form can be defined by the equations:

$$\dot{x} = f(x,u,t)$$
$$y = g(x,u,t)$$

where:

$\dot{x}$ = n-dimensional state vector

u = m-dimensional input vector

y = k-dimensional output vector

A linear model of this nonlinear system can be expressed as:

$$\dot{x} = Ax + By + K_x$$
$$y = Cx + Du + K_y$$

where:

A = n x n system stability matrix

B = n x m system input matrix

C = k x n system output matrix

D = k x m system feed through matrix

K_x = rate bias vector at operating point

K_y = output bias vector at operating point

The elements of **A**, which make up the stability matrix, are related to the partial derivative of an element of the nonlinear function f, with respect to an element of the state vector **x** at the operating point $\mathbf{x}_0$:

$$a_{ij} = \left.\frac{\partial f_i(x, u, t)}{\partial_{x_j}}\right|_{\substack{x = x_o \\ u = u_o}}$$

This matrix shows you how each state in your model affects the rates (first derivatives) of the system and is the basis for determining your system eigenvalues. The **A** matrix is printed in the Analysis Output Listing file. The rates are the row designators and the states are the column designators. For example, for a given second order nonlinear system, the stability matrix may look like that shown below.

	S_Out_INR	S_Out_INP
S_Out_INR	-0.5000	-1.0000
S_Out_INP	1.0000	0.0000

The rate of state **S_Out_INR** can be related to both states **S_Out_INR** and **S_Out_INP** as:

$$\frac{d}{dt}(SoutINR) = -0.5 \cdot (SoutINR) - 1.0 \cdot (SoutINP)$$

and likewise:

$$\frac{d}{dt}(SoutINP) = 1.0 \cdot (SoutINR)$$

Similarly, the **B** matrix is the matrix of partial derivatives of the elements of the nonlinear function f with respect to the elements of input vector **u** at the operating point $\mathbf{x}_0$:

$$b_{ij} = \left.\frac{\partial f_i(x, u, t)}{\partial u_j}\right|_{\substack{x = x_o \\ u = u_o}}$$

The **C** matrix is the matrix of partial derivatives of the elements of the output vector y with respect to the elements of the state vector **x** at the operating point $\mathbf{x}_0$:

$$c_{ij} = \left.\frac{\partial y_i}{\partial x_j}\right|_{\substack{x = x_o \\ u = u_o}}$$

The **D** matrix is the matrix of partial derivatives of the elements of the output vector **y** with respect to the elements of the input vector **u**:

$$d_{ij} = \left.\frac{\partial y_i}{\partial u_j}\right|_{\substack{x = x_o \\ u = u_o}}$$

The bias vectors K_x and K_y are defined by:

$$K_x = f\ (x_o,\ u_o,\ t)\ -\ Ax_o\ -\ Bu_o$$
$$K_y = g\ (x_o,\ u_o,\ t)\ -\ Cx_o\ -\ Bu_o$$

and are used to match the rates and outputs of the linear system with the rates and outputs of the nonlinear system.

For more information on linearized analysis of control systems, see Reference 18 at the end of this manual.

The Numerical Method of Linearization

A perturbation method is used to calculate the linear model. To calculate the **A** and **C** matrices, each continuous state is perturbed, in turn, by an amount equal to the step size or error control associated with this state.

The effect on the system rates and output is then measured and used to calculate each column of the **A** and **C** matrices, as follows:

$$A_j = \frac{\dot{x}_j - \dot{x}_o}{e_j} \qquad C_j = \frac{y_j - y_o}{e_j}$$

where:

$\dot{x}_j$ = n-dimensional perturbed rate vector
$\dot{x}_o$ = n-dimensional rate vector at operating point $\dot{x}_o, u_o$
y_j = k-dimensional perturbed output vector
y_o = k-dimensional output at operating point $\dot{x}_o, u_o$
A_j = th column of system stability matrix
C_j = th column of system output matrix
e_j = th element of error control vector

This process is continued for all n columns of A.

To calculate the B and D matrices, each input is perturbed by 1% of its nominal value (or, 2^{-6} if its nominal value is smaller than 10^{-8}). The effect on the system rates and output is measured and is used to calculate each column of the **B** and **D** matrices as follows:

$$B_j = \frac{\dot{x}_j - \dot{x}_o}{u_j - u_o} \qquad D_j = \frac{y_j - y_o}{u_j - u_o}$$

where:

B_j = jth column of the **B** matrix
D_j = jth column of the **D** matrix
u_j = n-dimensional perturbed input vector
u_o = n-dimensional input vector at operating point

System Eigenvalues of the Linear Model

The eigenvalues of the linear system:

$\dot{x} = Ax$

are defined to be the solutions **x** of the equation:

$\det(A - \lambda I) = 0$

Their significance lies in the fact that these systems usually have solutions of the form:

$x(t) = Ue^{lt}U^{-1}x_o$

where:

λ = system eigenvalues
t = time
U = the system modal matrix (its columns are the system eigenvectors)
x_o = operating point

Consider an element of the solution for x(t), the function e^{lt}, where λ is a complex scalar. If the real part of λ is negative, this represents a function that eventually damps out as t increases. On the other hand, if the real part of λ is positive, the function will grow without bound as **t** increases. Finally, if the real part of λ is equal to zero, the function will oscillate with a frequency equal to the imaginary part of λ as **t** increases. The response of your system, its time domain behavior, x(t), is characterized by the values of λ, the eigenvalues of the system stability matrix.

Thus, the system eigenvalues are a set of n complex numbers that characterize the dynamic behavior of the system in a region about the chosen operating point. Existence of an eigenvalue with a positive real part indicates an unstable system about x_o.

For a linear system to be asymptotically stable (or, in other words, for a given step input to have all outputs eventually "damp out"), all system eigenvalues must have real parts less than zero.

Highly Nonlinear Systems

The stability predicted by the system eigenvalues is stability of a system at a Steady-State operating point (all rates equal zero) and subject to small disturbances. System eigenvalues do not necessarily predict the overall stability of nonlinear systems as shown in Figure 80.

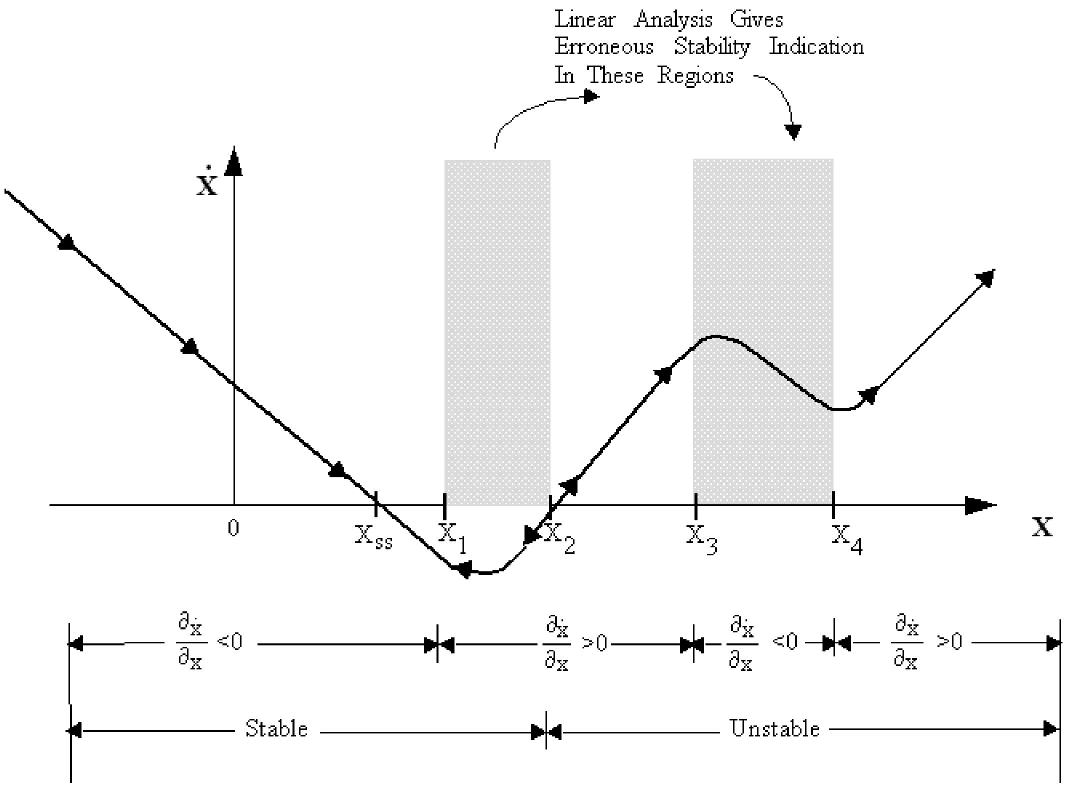


Figure 80 Nonlinear Stability Example

In this example the state derivative $\dot{x}$ is shown as a highly nonlinear function of the single state variable x. The arrows on the plot of the resulting function show the time domain trajectory that the state and state derivative (rate) would follow from any initial condition, x.

For the values of x shown, there is a region of asymptotic stability and a region of instability. Initial values of x in the stable region cause the system to reach the Steady-State operating point, x_{ss} Initial values of x greater than x_2 will result in divergence to large positive values.

The eigenvalue of this simple system is the partial derivative $\partial \dot{x} / \partial x$. The simple criterion of a negative real eigenvalue for asymptotic stability specifies that the system is unstable in the region x_1 to x_2. However, in this example the system will converge to the Steady-State point, x_{ss}. In the region x_3 to x_4 the eigenvalues criteria indicate that the system is stable, while in fact it will diverge from this region.

Importance of a Steady-State Operating Point

The example above also illustrates the risk of using eigenvalues alone as a measure of system stability at points other than Steady-State operating points. However, insights into system behavior and other useful information can still be obtained from such linear models, especially because they can always be easily verified by the nonlinear simulation capabilities of Easy5.

The method of model linearization used by Easy5 may result in erroneous predictions of stability for models linearized at "highly nonlinear" operating points. Because of this potential problem, Easy5 performs a linearity check as part of the Linear Model Generation analysis, and you are warned by the program if your model exhibits "highly nonlinear" behavior at the operating point specified. See the "Measure of Linearity" section below for more information on this linearity check.

Note: The current version of Easy5 performs the linearity check only when you do not specify input and output vectors as part of the analysis description.

The Measure of Linearity

As a measure of the validity or quality of the linear approximation of the nonlinear model, the stability matrix calculation described above is repeated using one half of the perturbations used in the initial calculation. The ratios of the derivatives are calculated using the two step sizes evaluated. Clearly, as shown in Figure 81, if the model is fairly linear in the region of the operating point, the ratios will be close to one.

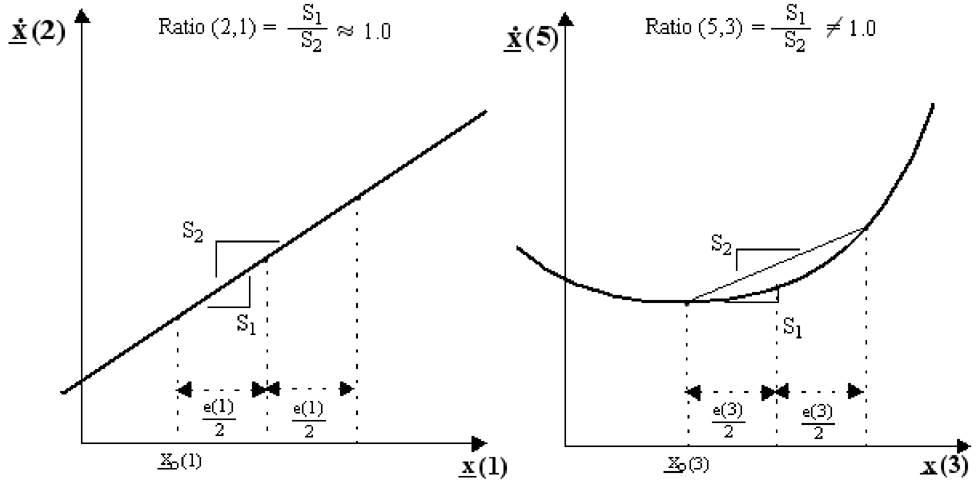


Figure 81 Linearity Measure of Nonlinear Model

When these ratios differ by more than ten percent, this is noted in your output listing. If one or more such elements are detected, the count of such elements is printed along with a list of ratio elements exceeding this tolerance.

If Easy5 warns you of poor linearity in your model, one or more of the following conditions may be present:

1. The perturbation step size is too large. You can make the perturbation step size smaller for a given state by reducing its **ERROR CONTROL** value.

2. The model is so nonlinear at this operating point that a meaningful linear approximation does not exist. In particular, there are two common values for the ratio which indicate certain behavior at operating points:

- If the **RATIO = 2.000**, this indicates that the state's perturbation has "stepped into" a region of saturation. To eliminate this effect, either temporarily remove the saturation effect or adjust the step size to avoid it.
- If the **RATIO = 0.000**, this indicates that the state's perturbation has stepped into a deadzone. In other words, no effect is felt by the system rates with respect to changes in this state's value while in the deadzone. Either temporarily remove the effect of the deadzone or adjust the step size to avoid it.

3. Your model may contain errors that are causing extreme nonlinearities.

Stability Analysis for Sampled-Data Systems

Easy5uses a state transition approach given by Kalman and Bertram for assessing the stability of sampled data systems. This approach determines a so called transition matrix for every sampling rate of the system. The autonomous system behavior between sample instants is given completely by the continuous system Jacobian matrix. At sampling instants, the system behavior is described by a combination of transition matrices for each sampling rate. See Appendix C for more information on the theory behind discrete components.

A complete discussion of the types of states defined in discrete components (delay and sample states) can be found in “States”. For a good background on analysis of sampled data systems, see Suggested Reading References 17 and 19 at the end of this manual.

Coordinate System Transformation

While s-plane results apply to continuous systems, the z-plane is the correct coordinate system for interpreting the stability of digital systems where time is, by definition, always at a sampling interval. For sampled data systems it is often convenient to be able to interpret stability in terms of the continuous system so that the results can be correlated with the known characteristics of the continuous plant dynamics.

For convenience, all Easy5 linearized analyses give results in both the z-plane and the s-plane for sampled data systems. Also, you can specify two of the discrete standard components, DF and DL, in terms of s-plane (Laplacian) coefficients if you are uncomfortable with working in the zplane. Easy5 uses a numerical transformation technique known as a Tustin transformation with prewarping to approximate the transformation of coordinates from the s-plane to the z-plane for filter coefficients.

Linking External Code

See also:"Compiling External Code"

"Executable Model"

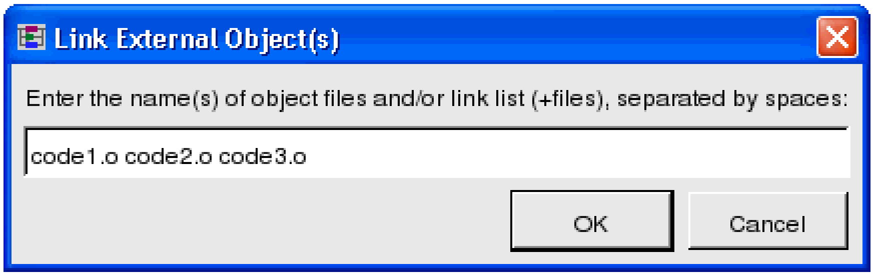
User Code and Library components may contain references to user-defined external subroutines or functions. These subroutines and functions are "external" in that they reside outside your model. The external user supplied routines must first be compiled before linking and creating an executable model. Recommended Fortran and C compilation options are listed in “Compiling External Code”.

During the link process, user object code will always be searched for global references before Easy5 standard routines. There are different methods for linking user-defined routines. Any one, or a combination of these methods may be used, depending on the circumstances. Each method is described in the following sections.

Linking Routines Using the Build Menu

To link user supplied routines, select **Link External Object** from the **Build** menu. A dialog box will appear requesting a file name or multiple file names separated by spaces. Object files or object libraries may be linked by typing the names into this dialog box.

Figure 82 shows an example of entering three object files. If several files need to be linked, you may use a "link list" to enter the multiple file names as described in the next section.



Enter the name(s) of object files and/or link list (+files), separated by spaces:

code1.o code2.o code3.o

Figure 82 Example of Using the Link External Object Dialog

Note:	The Link External Object dialog field width is 80 characters long. If more than 80 characters are needed to display the object files to be linked, then use a *link list* as described in the following paragraph.

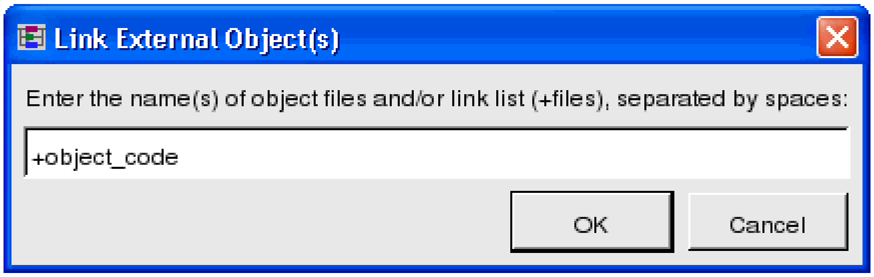
Using Link Lists to Link External Code

A *link list* is a file that contains a list of all files to be linked. The file names must be listed one per line in the link list file. The link list file should not exceed 256 characters per line. The link list file should not exceed 1024 characters, including a blank spaces used as delimiters between files. To avoid reaching this limit, minimize the usage of large pathnames and filenames, and use object libraries when possible. See "Linking Routines Using an Object Library".

If many object files need to be managed and linked, it is much better to store the object files in an object library. The object library can then be linked by simply entering its name in the Link External Object dialog, as described on the previous page.

Having created this external link list, the link list filename is entered the dialog box, preceding the first character with a "+" to distinguish the link list from a single object file. For example, assume that the three object code file names shown in Figure 82 are to be linked using a link list.

A link list file named "object_code" (or any file name) can be created, listing the object file names one per line. Figure 83 shows an example of entering the "object_code" link list file into the dialog. Easy5 will strip the "+" character, and open the file and read the list of filenames to be linked.



Enter the name(s) of object files and/or link list (+files), separated by spaces:

+object_code

Figure 83 Example of Using a Link List

Linking Routines Using the EASY5_OBJECT Variable

User supplied routines may be linked outside Easy5. The file names of the user supplied routines to be linked, are predefined before running Easy5 by defining the environment variable `EASY5_OBJECT`. This environment variable should be defined in the shell that Easy5 will be run from, and only needs to be defined one time. To do this, enter the following command in the shell from which Easy5 will be run:

```
setenv EASY5_OBJECT <my_object>   (C shell)
export EASY5_OBJECT <my_object>   (Bourne and Korn Shell)
set EASY5_OBJECT=<my_object>  (Windows Easy5 Command Shell)
```

where: *my_object* is the name(s) of the object code file(s).

For example, to use the EASY5_OBJECT variable to link three user supplied routines (Linux object files named code1.o, code2.o, and code3.o; or Windows files named code1.obj, code2.obj, and code3.obj), before running Easy5, you would perform the following steps:

1. Open a command shell window from which Easy5 will be run. For Windows, use an *Easy5 Command Shell* (or set the variable via your Control Panel).
2. In a command shell, enter:

```
setenv EASY5_OBJECT "code1.o code2.o code3.o"(C Shell)
export EASY5_OBJECT="code1.o code2.o code3.o" (Bourne, Korn
Shell)
set EASY5_OBJECT=code1.obj cod2.obj code3.obj (Windows Easy5
CommandShell)
```

3. Enter `easy5x` to open Easy5.
4. While running Easy5, you may verify that the object code has been linked by selecting **Build > Link External Object**. The Link External Object dialog box pops up, and the linked object file names appear in the dialog.

If a large number of user supplied routines need to be linked, a link list may be used to better manage the process. The `EASY5_OBJECT` variable may also be set to a *link list*.

For example, assume that the three object file names listed in step 2 are placed in a link list file named *object_code*.

The command to link this link list is:

```
setenv EASY5_OBJECT +object_code (C shell)
export EASY5_OBJECT=+object_code (Bourne, Korn shell)
set EASY5_OBJECT=+object_code (Windows Easy5 command shell)
```

Note: A maximum of 135 characters are allowed in a link list, including one blank space per filename. To avoid reaching this limit, minimize the usage of large path names and filenames. If many object files need to be managed and linked, it is better to archive the files into an archive library as described in the next section.

Linking Routines Using an Object Library

An object library is a file that contains one or more modules in object file format. Object files are created using a (Fortran or C) compiler. When a program references an object name not explicitly included in the program code, the linker attempts to resolve that reference externally. The linker searches any object libraries (specified in the link invocation) for the object reference and then, if found in an object library, extracts *only the particular object file*, not the entire object library. This method provides an efficient means of managing and linking large numbers of user-supplied routines.

Naming conventions differ slightly between Linux and Windows platforms with respect to default file tags used in creating and managing object library files. These differences are summarized below:.

Default File Type	Tag Name	Linux	Windows
Fortran source code file	*<fs_tag>*	`.f`	`.f`
C source code file	*<cs_tag>*	`.c`	`.c`
Object code file	*<obj_tag>*	`.o`	`.obj`
Library object file	*<lib_tag>*	`.a`	`.lib`

For a component library, the object library name will always be the library two-character name (*nn*) with the appropriate platform-appropriate object library file tag extension (.a or .lib):

<nn>.<lib_tag>	Object library for component libary *nn*

To create or manage an object library, perform the following steps as appropriate:

1. Open an Easy5 Command Shell window by selecting File > Open Command Shell... from the Easy5 main menu.
2. Move (or copy) all source files (`.f` or `.c` files) to be compiled and archived to a separate directory (*<compile_dir>*). This is not necessary but is recommended to make it easier to execute global compile and library management commands.
3. Navigate to this directory using the command:

cd <*compile_dir*>

4. From the <*compile_dir*> directory, perform a global compile by entering the appropriate compile command:

   `easy5x -fc *.f` *Fortran code*

   `easy5x -cc *.c` *C code*

   This ensures that all object files are compiled in such a way that they are compatible with Easy5.
5. To **create** an object library containing all object files use the command:

   `easy5x -lc` <*nn*> *.<*obj_tag*>

   For example, under Windows to create an object library named `xy.lib` using all locally available object files, use the command:

   ```
   easy5x -lc xy *.obj
   ```
6. To **update** an object library nn with new object files or to replace a current object file with updated code from <*filename*>.<*obj_tag*>, use the following command

   `easy5x -lu` <*nn*> <*filename*>.<*obj_tag*>
7. To **remove** an object file from an existing object library *nn*.<*lib_tag*>, use the following command:

   `easy5x -lx` <*nn*> <*filename*>.<*obj_tag*>
8. To **list the contents** of an existing object library *nn*.<*lib_tag*>, use the following command:

   `easy5x -ll` <*nn*>

Note: When creating an Easy5 executable model containing components from the *nn* library, the component library's object library is automatically linked; there is no need for you to explicitly reference it.

Linking Library Component Routines

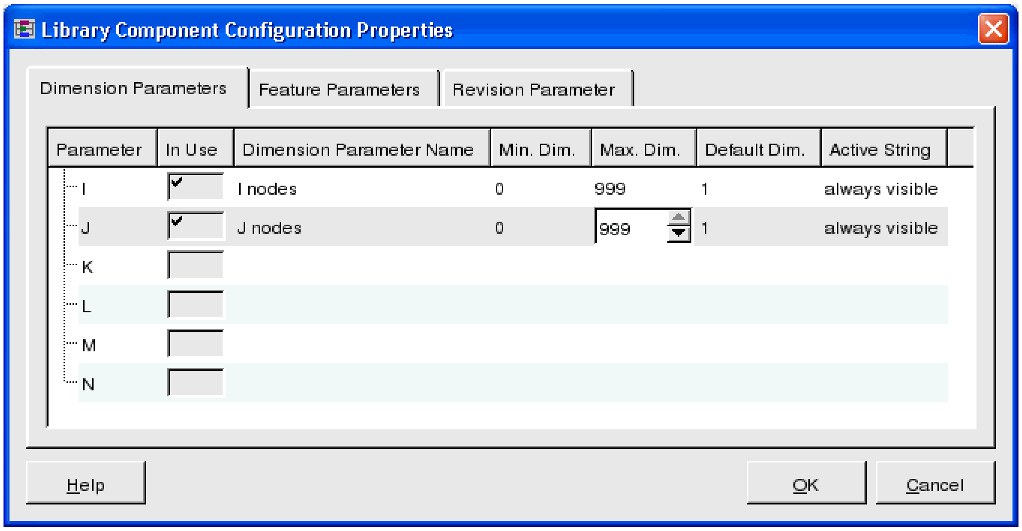
External code called from within Library components must also be linked before creating the executable. The external object code file name(s) can be entered directly into the Link External Object Dialog. Alternatively, the external routines can be automatically linked if all object code is put into an object library named <*nn*>.<*lib_tag*>, where <*nn*> is the two character tag name of the Component library. This is the easiest and the recommended method of linking external Library Component routines. Please see "Linking Routines Using an Object Library" for more details.

Library Component Code

Using Variable Dimensions in Library Component Code

If you specify one or more library component inputs using variable dimensions, you need to reference the value that is later assigned to the dimension parameter in your library component code. To reference a dimension parameter, enclose the dimension parameter in quotes (") whenever you use it in your library component code. For example, suppose that you specify a Library component input `XV` as being dimensioned (`I,J`). The sequence of commands is described as follows:

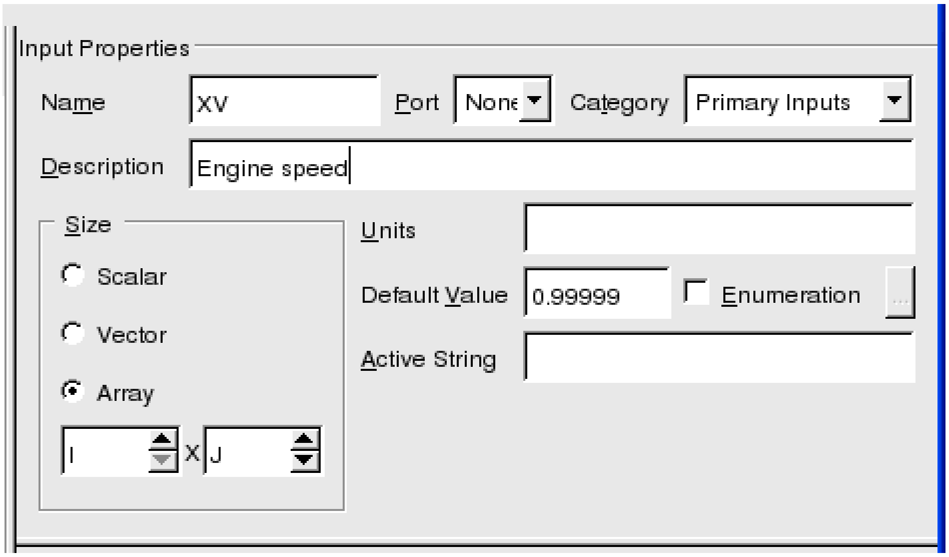
1. It is assumed that for this library component, you have already defined dimension parameters I and J using the Configuration button for the main component properties. If you have not done this already, select the Component Properties button at the top left of the Input/Output Definition panel, and select the Configuration button, resulting in a Configuration dialog as shown in Figure 85. In this case, we have assigned dimension parameters I and J, appropriately.



Parameter	In Use	Dimension Parameter Name	Min. Dim.	Max. Dim.	Default Dim.	Active String
I	✓	I nodes	0	999	1	always visible
J	✓	J nodes	0	999	1	always visible
K						
L						
M						
N						

Figure 84 User Code Editor Array Example

2. Select **New > Input** from the Library Component Editor, and specify the correct input name ("XV") in the Name field (of the properties panel).
3. Select **Array**, and then **enter "I"** and **"J" (no quotes) in the corresponding Size fields for input XV.**
4. Enter other input properties., as appropriate. The resulting Input Properties form should look similar to that shown in Figure 85.


Input Properties

Name: XV
Port: None
Category: Primary Inputs
Description: Engine speed
Size: Scalar [ ] Vector [ ] Array [✓]
I x J
Units:
Default Value: 0.99999
Enumeration [ ]
Active String:

Figure 85 User Code Editor Array Example

5. Use a similar approach to define other variable-dimension input and outputs.

Then, to use the dimension parameters I and J in your code as indexes of a nested DO loop, enclose the mode names in quotes:

```
DO L_AA = 1, "I"
  DO M_AA = 1, "J"
    <output> = ... XV_AA(L_AA,M_AA)
     enddo
enddo
```

The executable model builder program replaces the quoted dimension parameters with the values that you assign in the respective component data table when the executable model is created.

Using Integer or Logical Variables in Library Component Code

Easy5 declares each name defined as a Library Component input or output as `Real*8`. Therefore, if you want to use integer or logical variables in your library component code, they must not be declared as inputs or outputs. If an input must be used as an integer, such as the index on a `DO` loop, or an integer must be transmitted to other components as an output, use the following method. For example, `II` is an input to the component and that `JJ`, calculated in the component, is an output.

1. Define dummy names for communicating these variables to and from other components such as XII and XJJ, and specify these names as an input and output.

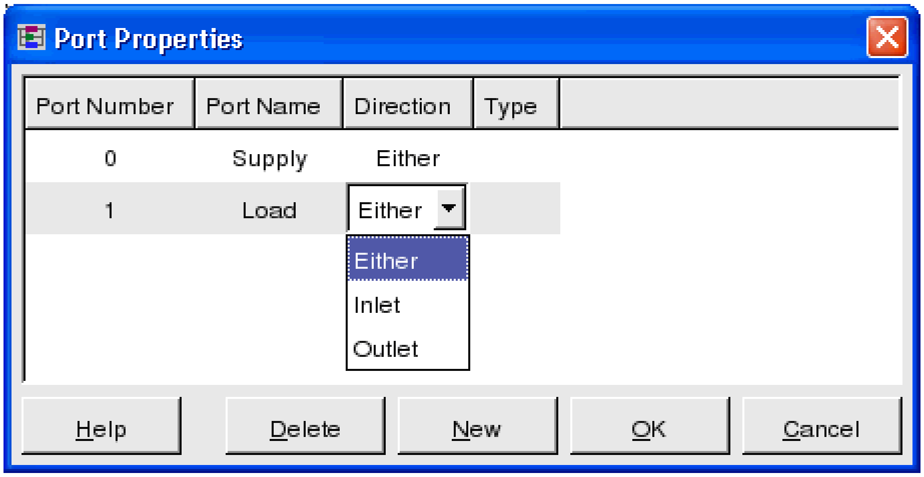
2. Assign the appropriate input value to a local variable in the component code. For example, `II_TT = XII_TT`

Configurations

The Library Component allows for multiple configurations in a single component with a different icon for each configuration, plus the ability to setup different classes of data, define ports with unique names, force connections to specific ports, and enter conditional code for different configurations. Please see User Guide, Chapter 12 - Library Components, for details.

Port Name Specification

Click on New to add a port. The Port Number field is only used internally, but the Port Name field is what your users see, so choose it carefully. You can use up to 12 alphanumeric characters (no spaces or underscores) in a port name.



Port Number	Port Name	Direction	Type
0	Supply	Either	
1	Load	Either	

Figure 86 Library Component Port Properties Specificaiton

Component Libraries

Select **Library** > **New**. A selection box, as shown in Figure 87, shows available editable libraries. In this example, the directory contains only one editable library; each library contains one or more Library components. This dialog lists all edit-enabled component libraries in your working directory, or as assigned per library environment variables. You may also change directories to access other Component libraries. Select the desired Component library from this dialog list. The Component library appears in the Library Menu. See the User Guide, Chapter 11 - Library Developer Toolkit, the section "Library Search Order" for more information.

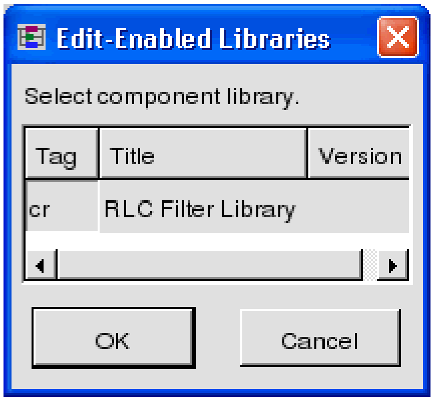


Figure 87 Edit Library Dialog

Matrix Operations

Matrix operations in Easy5 may be performed using a set of shorthand matrix expressions, or, you may call Easy5 subroutines that perform special matrix operations. Both methods will be covered in the following sections.

Matrix Equations

Easy5 includes a set of shorthand matrix equations that you can use in your Fortran code.

Caution:	These matrix equations cannot be used in Library components; instead, use the equivalent matrix subroutines described in the next section.

The following notations are used in these tables:

- Lowercase letters represent the elements of matrices, e.g., A_{ij} = i, j element of matrix A.
- N and M represent the row and column dimensions of matrices, respectively. For example, matrix A has dimensions $A(N_A,M_A)$.

As shown below, most common matrix arithmetic calculations can be placed in a Fortran component using a single line command.

Table 1-19 Easy5 Matrix Equations

Matrix Arithmetic Format	Mathematical Description of Operation	Dimension Requirements
Addition /A/=/B/+/C/	$a_{ij} = b_{ij} + c_{ij}$	$N_A = N_B = N_C$ $M_A = M_B = M_C$
Subtraction /A/=/B/-/C/	$a_{ij} = b_{ij} - c_{ij}$	$N_A = N_B = N_C$ $M_A = M_B = N_C$
Multiply /A/=/B/*/C/	M_B $a_{ij} = E \quad b_{ik} c_{kj}$ $k = 1$	$N_A = N_B$ $M_B = N_C$ $M_A = M_C$
Cross-Product /A/=/B/X/C/	$a_1 = b_2c_3 - b_3c_2$ $a_2 = b_3c_1 - b_1c_3$ $a_3 = b_1c_2 - b_2c_1$	$N_A = N_B = N_C = 3$ $M_A = M_B = M_C = 1$
Dot Product /A/=/B/./C/	3 $\underline{A} = a1 = E \quad b_i c_i$ $i=1$	$N_A = M_A = M_B = M_C = 1$ $M_B = N_C = 3$
Linear Equation Solution /A/=/B/-1/C/	Note: The Matrix B is destroyed by this operation. $\underline{B}\,\underline{A} = \underline{C}$ $\underline{A} = \underline{B}^{-1}\ \underline{C}$	$N_B = M_B = N_A = N_C$ $M_A = M_C$
Transpose /A/=/B/T	$a_{ij} = b_{ji}$	$N_A = M_B$ $M_A = N_B$

Table 1-19 Easy5 Matrix Equations

Matrix Arithmetic Format	Mathematical Description of Operation	Dimension Requirements
Eigenvalues /A/=/B/E	a_{i1} - Real (e_i) a_{i2} - Imag (e_i) e_i = Complex eigenvalues of B	$N_A = N_B = M_B$ $M_A = 2$
Identity Matrix /B/ = /1/	b_{ij} = 1 if i = j b_{ij} -= 0 if i = j	Dimensions of /B/ must be qualified by another statement.
Null Matrix /B/ = /0/	b_{ij} = 0 for all i,j	Dimensions of /B/ must be qualified by another statement.
Equality /A/ = /B/	$a_{ij} = b_{ij}$	$N_A = N_B$ $M_A = M_B$

The following rules apply to matrix expressions:

1. Each matrix expression must begin on a new line in your code and have a slash, "/", as its first nonblank character.
2. All matrix expressions must have a slash before and after each matrix quantity name. An equal sign, "=", must follow the first matrix name in the expression.
3. Only one matrix operation can be used on each line.

Matrix arithmetic statements may have one or two input matrices, designated /B/ and /C/ as shown in the figures. You may use your own names in place of /B/ and /C/. The dimensions of these matrices must be defined in the inputs section at the top of the Fortran component data form. If /B/ or /C/ are state or parameter matrices, they may appear at any point in your Fortran component. If they are variable matrices, they must be defined before the matrix arithmetic expression which uses them occurs in the model.

Caution:	The /B/ and /C/ matrices must be defined as "Inputs" in the Fortran component data form. (You may use your own names in place of /B/ and /C/).

The output quantity from the matrix expressions is designated by /A/. The dimensions of /A/ may be defined by a previous matrix expression (i.e., /A/ is a local dummy matrix and not an output of the component), by defining the /A/ matrix to be an output of the component, or by setting /A/ equal to a vector or array Fortran variable. If the dimensions of /A/ have not been defined, /A/ will be dimensioned based on the dimensions of /B/ and /C/.

Caution:	The matrix inverse expression (linear equation solution: /A/=/B/-1/C/) may only be defined one time in your model. If you need to use this more than once, make a “call” to the matrix operation Subroutine LUEQS as defined in the next section.

Matrix Operation Subroutines

The following Fortran subroutines can be used within your Easy5 Fortran and Library Component code to perform a variety of matrix operations. These operations use the following conventions:

Matrix A is defined as A (r_A,c_A)

where: r_A,c_A= row, column dimensions of A, entered as *Integer* values

Load Entire Matrix with a Scalar Value

Syntax:	CALL MATLOD (A, VALUE,r_A,c_A)
Operation:	/A/ = / VALUE/ where r_A,c_A are row, column dimensions of A

Matrix Addition

Syntax:	CALL MATADD (A, B, C, rB,cB)
Operation:	/A/ = /B/ + /C/
Restrictions:	r_B = r_C, c_B = c_C

Matrix Subtraction

Syntax:	CALL MATSUB (A, B, C, r_B, c_B)
Operation:	/A/ = /B/ - /C/
Restrictions:	r_B = r_C, c_B = cC

Matrix Multiply

Syntax:	CALL MATMPY (A, B, C, r_B, c_B, c_C)
Operation:	/A/ = /B/ * /C/
Restrictions:	c_B = r_C

Matrix Transpose

Syntax:	CALL TRANS (A, B, r_B, c_B,)
Operation:	/A/ = /B/ * /C/
Restrictions:	$r_B = c_A$, $c_B = r_A$

Matrix Cross-Product

Syntax:	CALL CRSPRD (A, B, C)
Operation:	/A/ = /B/ X /C/
Restrictions:	$r_B = r_C$, $c_B = c_C = 1$ or 0

Matrix Dot Product

Syntax:	CALL DOTPRD (A, B, C, N)
Operation:	/A/ = /B/ . /C/
Restrictions:	$N = r_B = r_C$

Matrix Equality

Syntax:	CALL XFR (B, A, N)
Operation:	/A/ = /B/ where N is length of A,B
Restrictions:	$N = r_A = r_B$

Matrix Inverse

Syntax:	CALL LUEQS (A, B, C, IA, NA, NB, MA, MB, MC, FPZ, IEZERR)
Operation:	/A/ * /B/ = /C/ where B is the solution or: /B/ = /A/ -1 /C/ (Easy5 shorthand notation and matrix A is destroyed!)

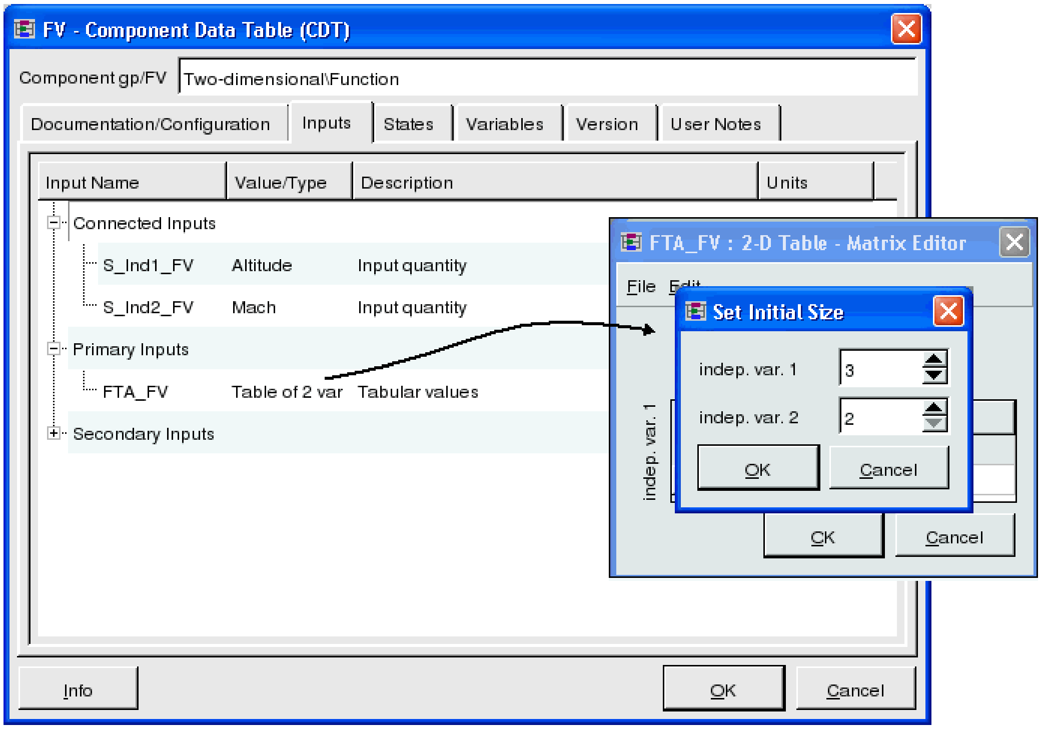
Inputs:	A	Multiplier matrix
	C	Right-hand side (identity)
	NA	Order of matrix A
	NB	No. of columns of B and C
	MA	Row dimension of A
	MB	Row dimension of B
	MC	Row dimension of C
	FPZ	Floating point precision
		(Usually set to 1.E-14 or 1.D-14)
Outputs:	B	Solution matrix, if IEZERR=0
	A	The LU decomposition of A
	IA	Integer work array of length NA
	IEZERR	Success flag =0 Successful =*n* Error, A(*n*,*n*) is singular

Matrix Editor

Creating a Data Table

The first step is to use a component that matches the requirements. Easy5 provides a 2-dimensional table component, the FV - Two-dimensional Function. Add this to your model, connect the inputs, and edit/open the data table as shown in Figure 88. Select the **Table of 2 Var** input field; this changes to **Edit Table**. Select **Edit Table** to open the editor.

To size the table, select **Edit > Resize**, The resize dialog will appear as shown in Figure 88. Set the independent variable 1 to 3 to accomodate the 3 Mach numbers, and set the independent variable 2 to 2 to match the number of altitude inputs.


FV - Component Data Table (CDT)

Component gp/FV: Two-dimensional\Function

Tabs: Documentation/Configuration, Inputs, States, Variables, Version, User Notes

Input Name	Value/Type	Description	Units
Connected Inputs			
S_Ind1_FV	Altitude	Input quantity	
S_Ind2_FV	Mach	Input quantity	
Primary Inputs			
FTA_FV	Table of 2 var	Tabular values	
Secondary Inputs			

FTA_FV : 2-D Table - Matrix Editor

Set Initial Size
indep. var. 1: 3
indep. var. 2: 2

Figure 88 Two-Dimensional Table Component

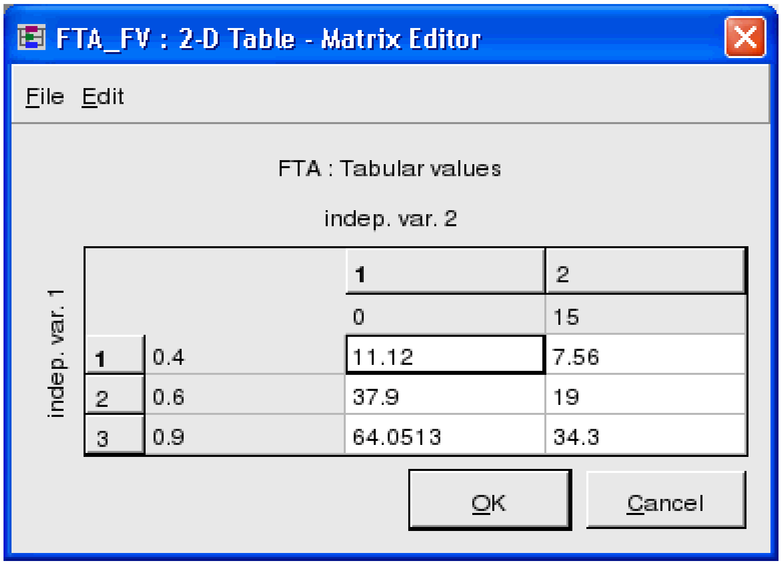
Having created the data table and sized it, you can enter data directly into the data cells using the Easy5 Matrix Editor.

After you have created the table, you can enter data, or copy it from an external spreadsheet program, or, create the data in an external file and import the data into the table.

The tabular data used in this example is small, and therefore it is easy to manually enter the data into the data cells.

1. Just select the first independent variable field, and type in the Mach numbers (0.4, 0.6, 0.9).
2. Do the same for the second independent variable, altitude.
3. Enter the dependent data *Mdelta–e*.

The finished data table is shown in Figure 89. Note that this spreadsheet matches the layout of the original data table.



FTA : Tabular values

indep. var. 2

indep. var. 1

		1	2
		0	15
1	0.4	11.12	7.56
2	0.6	37.9	19
3	0.9	64.0513	34.3

Figure 89 Completed 2-D Data Table

Menus

The menu bar, located at the top of the Easy5 window below the description lines, contains the following menus:

File Edit View Options Library Build Analysis Submodel Help

These menus are pulldown menus. The selection of these menus, using the mouse or the keyboard is discussed in "Using Shortcut Keys to Open and Select Menus". The following sections briefly describe each menu and the items that they contain.

File Menu

The **File** menu allows you to perform standard file operations. It contains the following items:

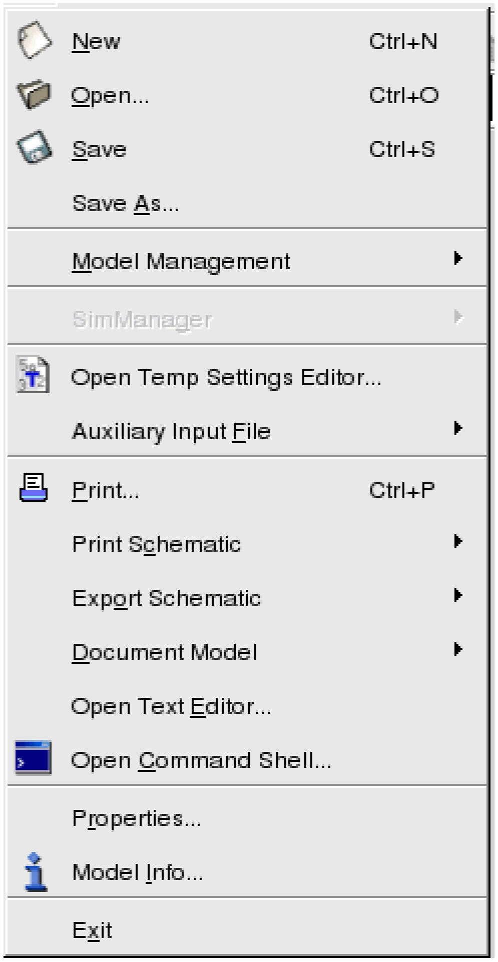


Figure 90 File Menu

New starts a new model. (The Model Building license feature must be checked out to use this item.)

Open is used to open a new or existing model, and change directory.

Save allows you to save changes to your model.

Save As... saves your model to a different name than the current one. (The Model Building license feature must be checked out to use this item.)

Model Management is used to manage your model files to purge, delete, rename files.

Open Temporary Settings Editor allows you to create, edit, delete, copy, or rename a temporary settings file for your current model.

Auxiliary Input File allows you to create, edit, delete, copy, or rename an auxiliary input file.

Print is used to get hardcopy printouts of your model schematic.

Print Schematic is used to print schematic diagrams to a Windows Meta File formatted graphics file to be used in Word or PowerPoint. The following print options are available: Current View, Current Schematic, Entire Hierarchy.

Export Schematic is used to save schematic diagrams to an enhanced Windows Meta File formatted graphics (.emf) file to be used in Word or PowerPoint. The following export options are available: Current View, Current Schematic, and Entire Hierarchy.

Document Model create and/or display an ASCII or HTML formatted file that documents the model, listing all component input/output names, values, description and units. (The Model Building license feature must be checked out to use this item.)

Open Text Editor... opens the Easy5 Text Editor (or another you designate). This allows you to view and edit any text file. See "Steady-State Analysis Method" for more information on the Easy5 Text Editor.

Open Command Shell... opens a command shell in the current model path allowing you to enter Easy5 command-line operations, any Korn-shell and/or DOS commands on Windows, or any Linux shell commands on Linux platforms.

Properties... describes the model by name, version, modifications, and time stamp.

Model Info gives a text-based description of the model.

Exit allows you to exit Easy5.

Edit Menu

(The Model Building license feature must be checked out to use this menu.)

The **Edit** menu is used to perform standard edit operations. It contains the following items:

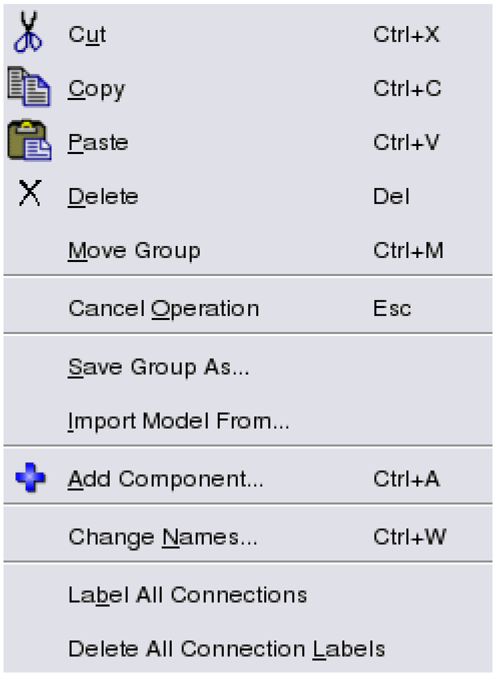


Figure 91 Edit Menu

Cut lets you remove a component from the schematic.

Copy lets you copy a component group to another location on your schematic.

Paste lets you move a group of components into a new model.

Delete lets you delete any connections or components that are currently selected.

Move Group is used to move a group of components to another location on your schematic.

Cancel Operation of an analyis, or similar action that has not yet run.

Import Model From... allows you to import an existing model into your current model.

Add Component... is used to add components to your model and opens the dockable Add Components window on your schematic pad.

Change Names... allows you to change user-defined names and Fortran names of selected components. (See "User-Defined Names").

Label All Connections labels all connection lines.

Delete All Connection Labels removes all connection line labels.

View Menu

The **View** menu is to view different aspects of your model. It contains the following items:

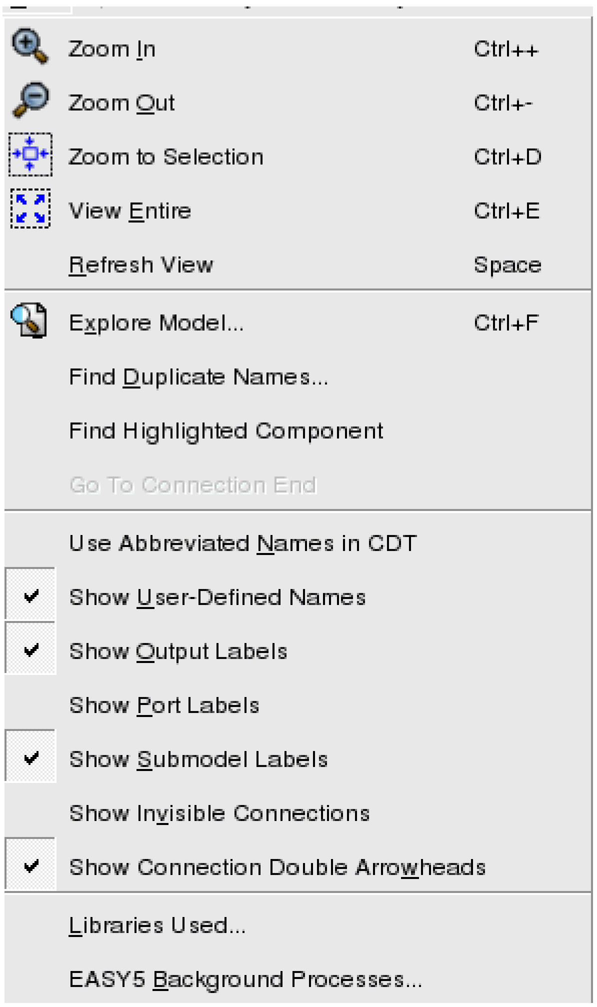


Figure 92 View Menu

Zoom In/ Out makes your view of the schematic diagram larger/smaller.

Zoom to Selection makes a selected area larger.

View Entire resizes the schematic block diagram to fit in the window.

Refresh View redraws the schematic to refresh the view.

Explore Model opens a listing of all inputs and ouputs in a Model Explorer window.

Find Duplicate Names finds and names them and where the are, so you can easily correct them.

Find Highlighted Component locates the currently highlighted component.

Go To Connection End lets you locate a connection's "to" or "from" component.

Use Abbreviated Names in CDT causes "shortened" names (without their component qualifier and underscore) to be displayed in any CDT. An Easy5 environment variable, `EZ5_VIEW_SHORT_CDT_NAMES`, can also be used to define this as the default behavior.

Show User-Defined Names displays the user-defined names used throughout the model when turned on, or displays the Easy5 default names when turned off.

Show Output Labels shows all output labels on connections lines (On/Off toggle check box).

Show Port Labels labels all connection lines with port names (On/Off toggle check box).

ShowSubmodel Labels opens a list of the submodels of your model.

Show Invisible Connections makes invisible connection lines visible.

Show Connection Double Arrowheads allows you to search for any duplicate input/output names.

Libraries Used displays a listing of all libraries used in your model.

Easy5 Background Processes... displays running Easy5 background processes.

Options Menu

The **Options** menu is used to setup options. It contains the following items:

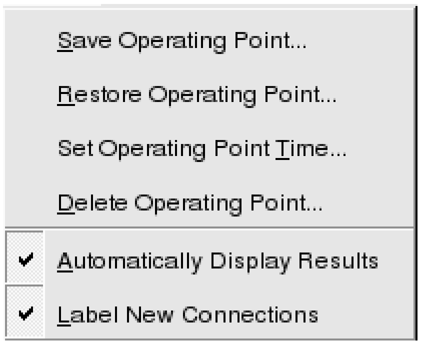

Save Operating Point...
Restore Operating Point...
Set Operating Point Time...
Delete Operating Point...
✓ Automatically Display Results
✓ Label New Connections

Figure 93 Options Menu

Save Operating Point... saves the operating point calculated during an analysis.

Restore Operating Point... restores an operating point that was saved using the Save Operating Point function.

Set Operating Point Time... defines the operating point value of TIME.

Automatic Display Results will automatically plot/print analysis output data after analysis completes (On/Off toggle check box).

Label New Connections when on (default), automatically labels new connections, when off, doesn't apply labels to connection line (On/Off toggle check box).

Library Menu

The **Library** menu is used to create and manage Library Components. It equires checkout of the Library Developer license feature, except for certain menu items. For more information see the User Guide, Chapter 12 - Library Components. This menu contains the following items:

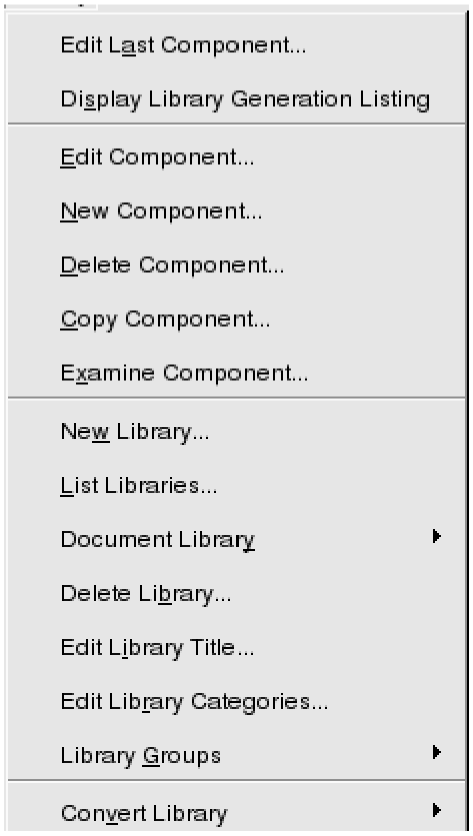


Figure 94 Library Menu

Edit Last Component... opens the last Library Component edited (in a given session).

Display Library Generation Listing opens and displays the Library Generation Listing file that was created during the generation of your Library component using the Easy5 Text Editor. See "Steady-State Analysis Method" for more information on the Easy5 Text Editor.

Edit Component... lets you open and edit an existing Library Component.

New Component... creates a new Library Component.

Delete Component... deletes an existing component.

Copy Component... lets you copy an existing component from one library to another.

Examine Component... allows you examine, but not modify, any Library component. This menu requires checkout of the Library Developer license feature, except for certain menu items.

New Library... create a new component library.

List Libraries... lists all libraries that are available to you.

Delete Library... used to delete only the libraries that you own.

Edit Library Title... allows you to change the title of an existing library.

Edit Library Categories... allows you to define library categories, used to group inputs and outputs in

Library Groups sub-menu is used to define groups within a library.

Convert Library from or to ASCII text, and from a V6 library to the newest library verison.

Build Menu

The **Build** menu is used to build the executable model. It contains the following items:

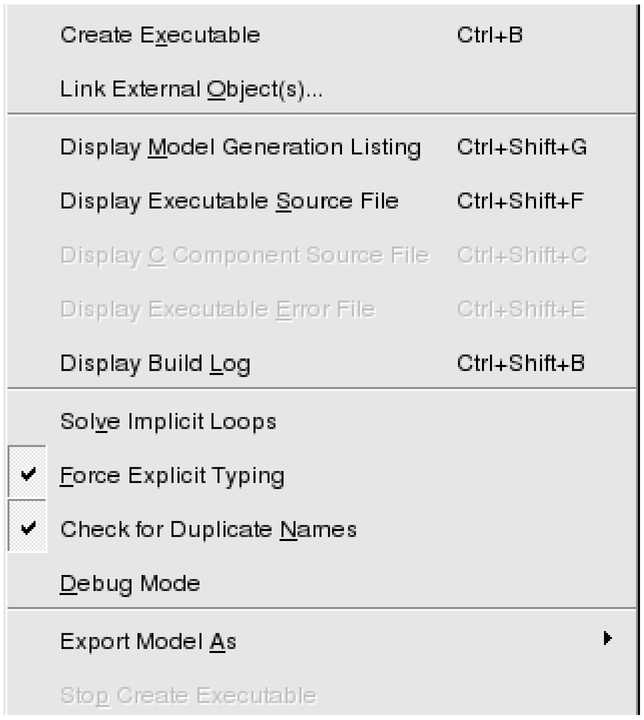

Create Executable Ctrl+B
Link External Object(s)...
Display Model Generation Listing Ctrl+Shift+G
Display Executable Source File Ctrl+Shift+F
Display C Component Source File Ctrl+Shift+C
Display Executable Error File Ctrl+Shift+E
Display Build Log Ctrl+Shift+B
Solve Implicit Loops
✓ Force Explicit Typing
✓ Check for Duplicate Names
Debug Mode
Export Model As
Stop Create Executable

Figure 95 Build Menu

Create Executable builds an executable version of the model before analysis.

Link External Object... links external object code when creating an executable version of your model.

Display Model Generation Listing... displays the Model Generation Listing file created during the Create Executable using the Easy5 Text Editor. See "Steady-State Analysis Method" for more information on the Easy5 Text Editor.

Display Executable Source File... displays the Executable Source File created during the Create Executable using the Easy5 Text Editor.

Display C Component Source File... opens and displays the C code source file containing all the code from all the C components (using the Easy5 Text Editor).

Display Executable Error File... opens the executable error file indicating errors occurring during compilation (using the Easy5 Text Editor).

Display Build Log... displays the log data created during the create executable process, including errors which occurred during the compilation process.

Solve Implicit Loops is used to solve implicit model loops.

Force Explicit Typing forces the compiler to check for all undeclared names.

Check for Duplicate Names checks for duplicate input/output names.

Debug Mode is a toggle switch enabling debug mode for generation of executables.

Export Model As... lets you select either a MAT EMX file format, an ADAMS External System Library format, or a MATLAB/Simulink S-Function.

Stop Create Executable terminates the "create executable" background process. (Linux only).

Analysis Menu

The **Analysis** menu is used to run analyses, and contains the following items:

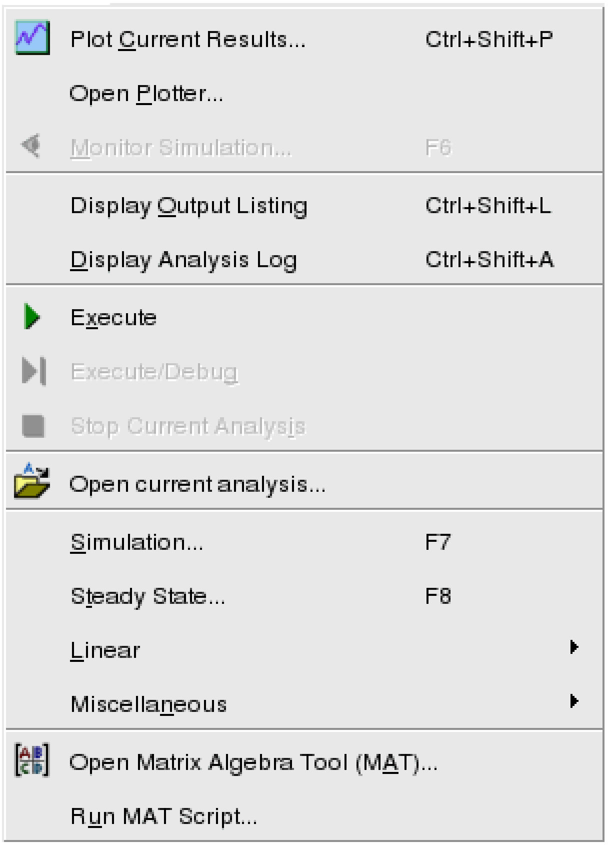


Figure 96 Analysis Menu

Plot Current Results... brings up the Easy5 Plotter and automatically displays plot data results from the most current analysis.

Open Plotter... brings up the Easy5 Plotter and allows you to plot data from any Easy5 plot file.

Monitor Simulation...(F6) opens the plotter in the monitor mode to display simulation data while the simulation is running.

Display Output Listing... opens and displays the Analysis Output Listing file that was created during an analysis (using the Easy5 Text Editor). See "Steady-State Analysis Method" for more information on the Easy5 Text Editor.

Display Analysis Log... opens and displays the analysis log file.

Execute performs the analysis.

Execute/Debug performs the analysis with the debugger turned on.

Stop Current Analysis terminates the execution of the analysis that is currently running.

Open current Analysis... opens the last analysis data form that was used.

Simulation... opens the Simulation Analysis Data Form which allows you to setup and launch a nonlinear simulation analysis.

Steady State... opens the Steady State Analysis Data Form which allows you to setup and launch a nonlinear steady state analysis.

Linear allows you to perform Linear analyses such as the Transfer Function analysis.

Miscellaneous allows you to perform analyses that do not fall under Linear, such as Plot Tables, Function Scan, and Multiple Analyses.

Open Matrix Algebra Tool spawns the Matrix Algebra Tool (*MAT*) program.

Run MAT Script opens a Run MAT Script analysis form.

Submodel Menu

The **Submodel** menu is used to create and manage submodels. It includes the following items:

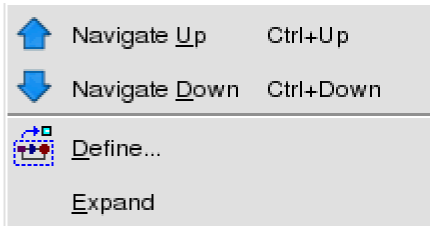


Figure 97 Submodel Menu

Navigate Up (**Ctrl+Up**) displays a hierarchical listing of all submodels in your model.

Naviage Down (**Ctrl+Down**) closes the submodel that you are currently viewing and moves the schematic up one hierarchical level.

Define... lets you to create a submodel for a group of selected components.

Expand is used to expand the submodel back to its original configuration, thereby eliminating the submodel.

Help Menu

The **Help** menu gives you access to online documentation. It contains the following items:

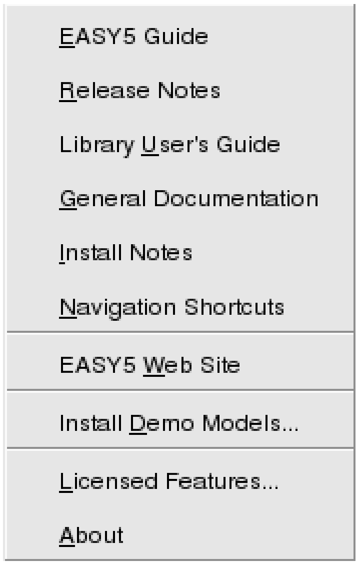


Figure 98 Help Menu

Easy5 Guide containing a guide to all the online documentation. This is the main entry point used to get help on various Easy5 topics.

Release Notes displays the Release Notes for the installed version.

Library User Guides gives you access to all library user guides.

General Documentation gives you access to general Easy5 documents including Users Guide, Reference Manual, and Matrix Algebra Tool.

Install Notes displays the Installation Notes for the installed version of Easy5.

Navigation Shortcuts offers information about schematic navigation shortcuts.

Easy5 Web Site... launches a web browser and opens the Easy5 home page.

Install Demo Models... provided a dialog allowing you to install Easy5 demo models to your current directory path.

Licensed Features... provides information about your license, including a complete list of all licensed Easy5 features, license checkout status, and the ability to manage individual license features.

About... displays information about the installed version, including version number, library variant selected, and build date.

Using Shortcut Keys to Open and Select Menus

Note: Standard use of shortcut keys is offered for access to all menu items. If unfamiliar with this, please read on.

Shortcut keys provide an additional means of opening menus, by typing in an assigned mnemonic character. All menu items have a character underlined to indicate the mnemonic key. To activate the menu item, just type in the underlined letter. However, to open a main menu from the menu bar, you must also enter the "Metakey" with the menu character. Once the main menu is opened, the Metakey is not needed. The Metakey is platform-dependent, but is commonly located to the left of the <space bar>. The Metakey is defined for different platforms as described below:.

Platform	Metakey
HPUX	Extend Character
IBM	Alt
Sun	Alt
Windows	Alt

Practice using the shortcut keys to activate the **Purge** menu function previously mentioned in the Model and Dat "File Menu".

1. Open the **File** menu by typing in <Metakey> + **f**.
2. Select the **Model Management** sub-menu by typing **m**.
3. Finally select the **Purge** function by typing **p**.

Using Arrow Keys to Select Menu Items

Note: Standard keyboard navigation is offered with all menus. If unfamiliar, read on.

In addition to the mouse and character keys, you can scroll through the menus using the keyboard arrow keys. A main menu from the menu bar must first be opened to use this feature. Perform the following steps to see how this feature works.

1. First open the **File** menu by either selecting it with a CLICK-L, or typing the **<Metakey> + f**.
2. Enter the right arrow key a few times; notice how the menu selection scrolls to the right.
3. Enter the left arrow key a few times; notice how the menu selection scrolls to the left.
4. Enter the down arrow key; notice how the selection moves down the menu.
5. Enter the up arrow key; notice how the selection moves up the menu.

Closing an Opened Menu

To close an opened menu without selecting a menu item, move the pointer outside the opened menu and CLICK-L.

Menus: Shortcut Menu

The shortcut menu is a context sensitive menu that displays when you select certain objects with the right mouse button. These menus gives you quick access to important options available through the main menu. The following section describes these menus.

Schematic Shortcut Menu

Point the mouse cursor in an open area on the Easy5 schematic pad and press and hold the right mouse button to open the schematic shortcut menu shown in Figure 99.

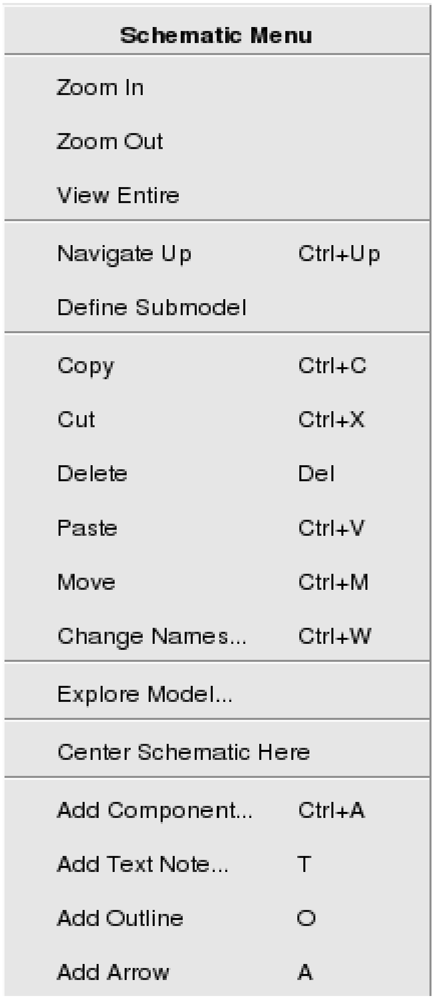


Figure 99 Schematic Menu

This is the only way to access this menu, which gives you menu options that pertain to the schematic.

You can easily copy, paste, or cut components from the schematic, as well as open the Model Explorer window to view all the inputs and outputs of the model.

An important feature of this menu is to use it to create a text note for component data that you want to place directly on the schematic.

Component Shortcut Menu

Selecting a component with the pointer, then holding down the right mouse button opens the component connect shortcut menu.

The component shortcut menu pops up as shown in Figure 100. This context sensitive menu gives you options to quickly access important component features, such as editing the title and descriptor, copying and deleting the component, and opening the icon editor.

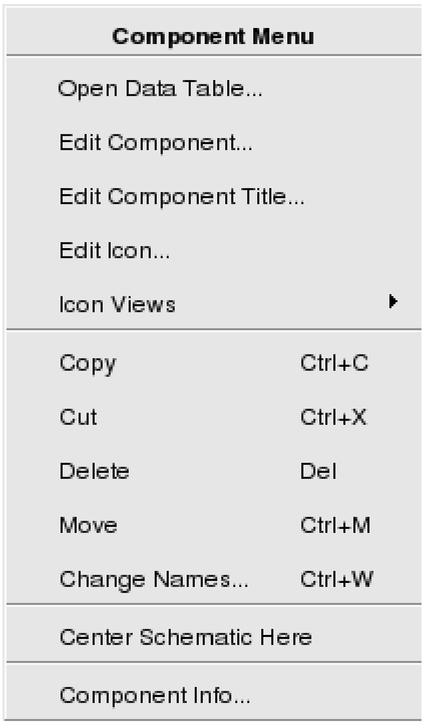


Figure 100 Component Menu

For example, selecting Edit Component Title opens a dialog used to edit the component title.

Component Connect Menu

If you right click on a connected component wihtout the component being activated (hightlighted), you get a component connection menu that lets you quickly customize components connections.

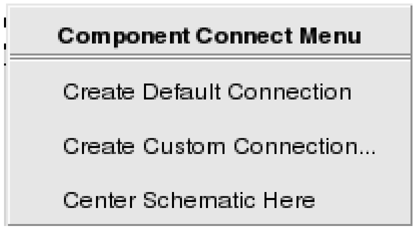


Figure 101 Component Connect Menu

Connection Line Shortcut Menu

Selecting any connection with the hold right mouse button brings up the connection shortcut menu as shown in Figure 102.

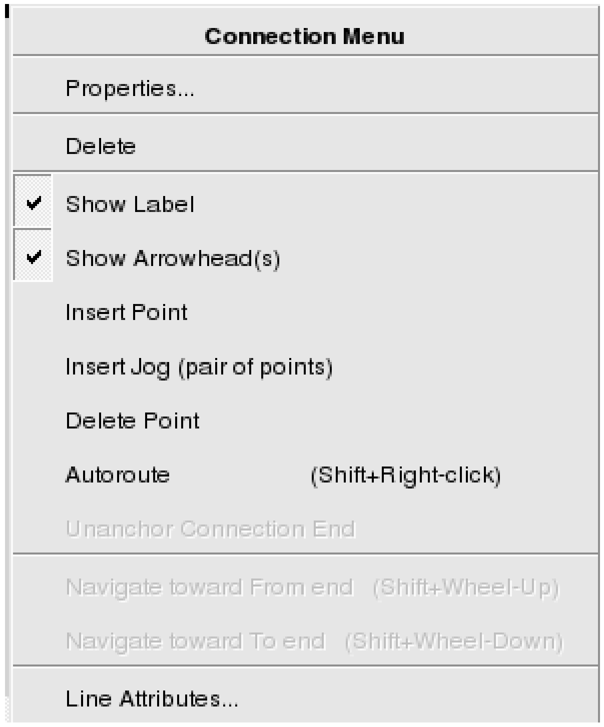

Connection Menu
Properties...
Delete
✓ Show Label
✓ Show Arrowhead(s)
Insert Point
Insert Jog (pair of points)
Delete Point
Autoroute (Shift+Right-click)
Unanchor Connection End
Navigate toward From end (Shift+Wheel-Up)
Navigate toward To end (Shift+Wheel-Down)
Line Attributes...

Figure 102 Connection Menu

The connection shortcut menu is very useful for opening the connection data table to view connection information, editing connection attributes, turning labels on or off, and following a connection to its endpoint.

Submodel Shortcut Menu

Selecting any submodel with the hold right mouse button brings up the submodel component shortcut menu as shown in Figure 103.

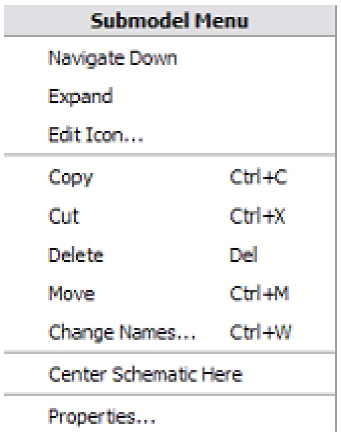


Figure 103 Submodel Menu

This context sensitive menu gives you options to quickly access important submodel features, such as editing the descriptor, copying and deleting the submodel and editing the icon.

Submodel Stub Shortcut Menu

A submodel “stub” is the semicircle along the submodel window boarder that defines the connection into/out of the submodel.

Selecting a submodel stub with a hold right mouse button opens the submodel stub shortcut menu shown in Figure 104. This context sensitive menu gives you options to quickly access important submodel connection features.

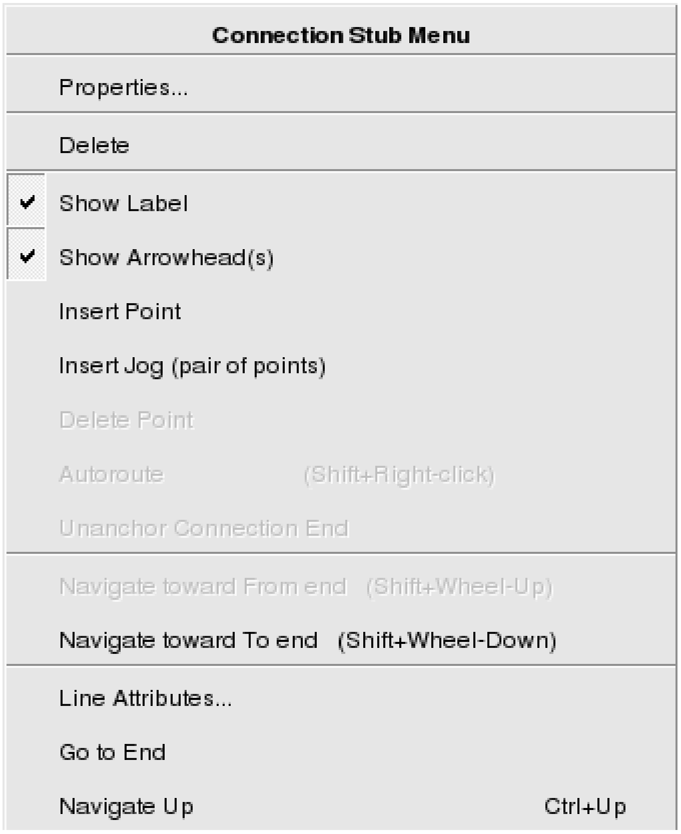


Figure 104 Submodel Menu

Group Shortcut Menu

The Group shortcut menu applies to grouped components. First, define a group by drawing a selection box around a group of components. To do this, select the upper left corner of the box with a CLICK-L; hold the left mouse button and move the mouse down and to the right to expand the box until all the components you wish to select are enclosed by the box. Then, to open the group shortcut menu, hold the right mouse button; this brings up the shortcut menu as shown in Figure 105.

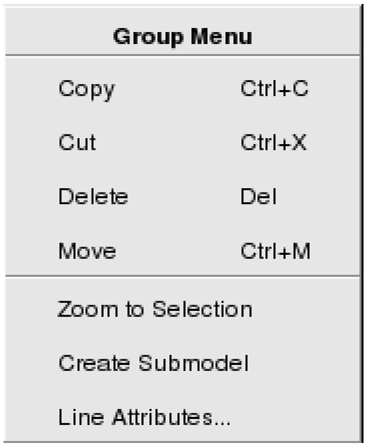


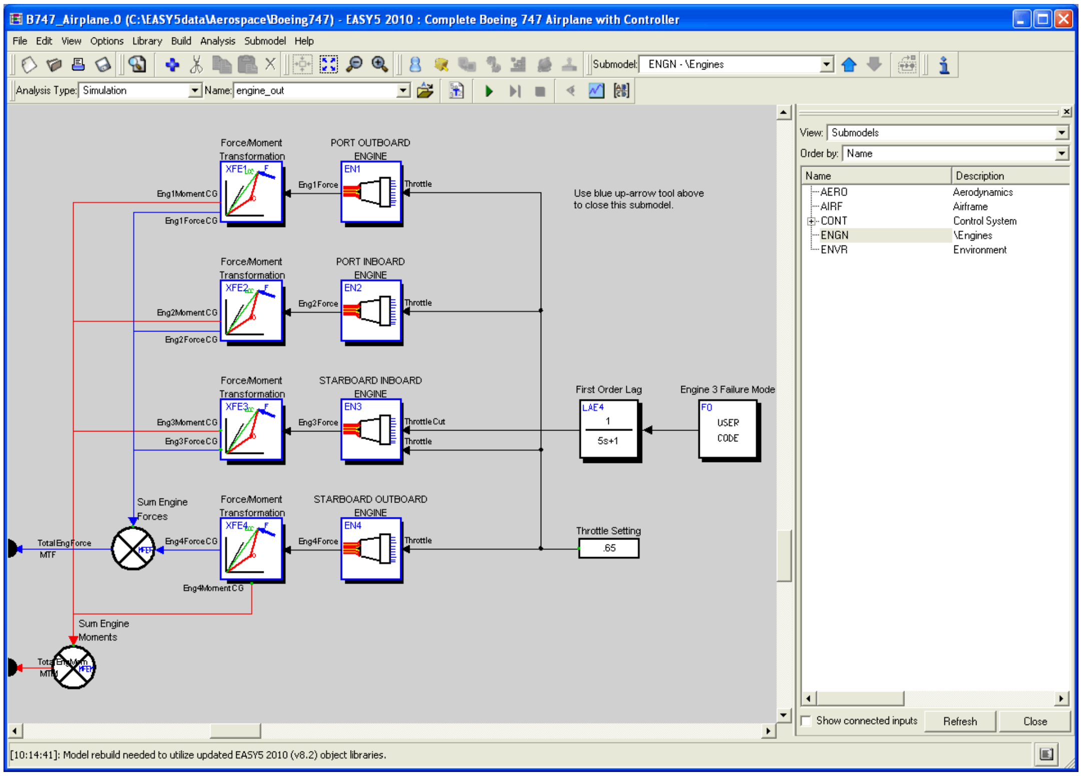
Figure 105 Group Menu

This shortcut menu gives you a convenient method to create a submodel, copy/delete a group of component and move a group of components.

Model Explorer Window

In general, the Model Explorer offers you another way of interacting with your model. When invoked from the Edit Tools toolbar (or **View > Explore Model** menu item), it pops up in a dockable window. As with any dockable window, Easy5 remembers the state of the window, and restores that state. The default position of the Model Explorer window is docked on the right-hand side of the schematic.

The Model Explorer window offers various "filters" for examining and navigating within the structure of your model. The schematic and Model Explorer are maintained in synch with one another. So, when you navigate in one window, the other will stay in-synch with the other. This allows you to interchangeably navigate using the schematic, or the Model Explorer, as shown in Figure 106, with the "filter" View selected to visualize the model by **Submodel**.



B747_Airplane.0 (C:\EASY5data\Aerospace\Boeing747) - EASY5 2010 : Complete Boeing 747 Airplane with Controller

Submodel: ENGN - \Engines

Analysis Type: Simulation Name: engine_out

Use blue up-arrow tool above to close this submodel.

View: Submodels
Order by: Name

Name	Description
AERO	Aerodynamics
AIRF	Airframe
CONT	Control System
ENGN	\Engines
ENVR	Environment

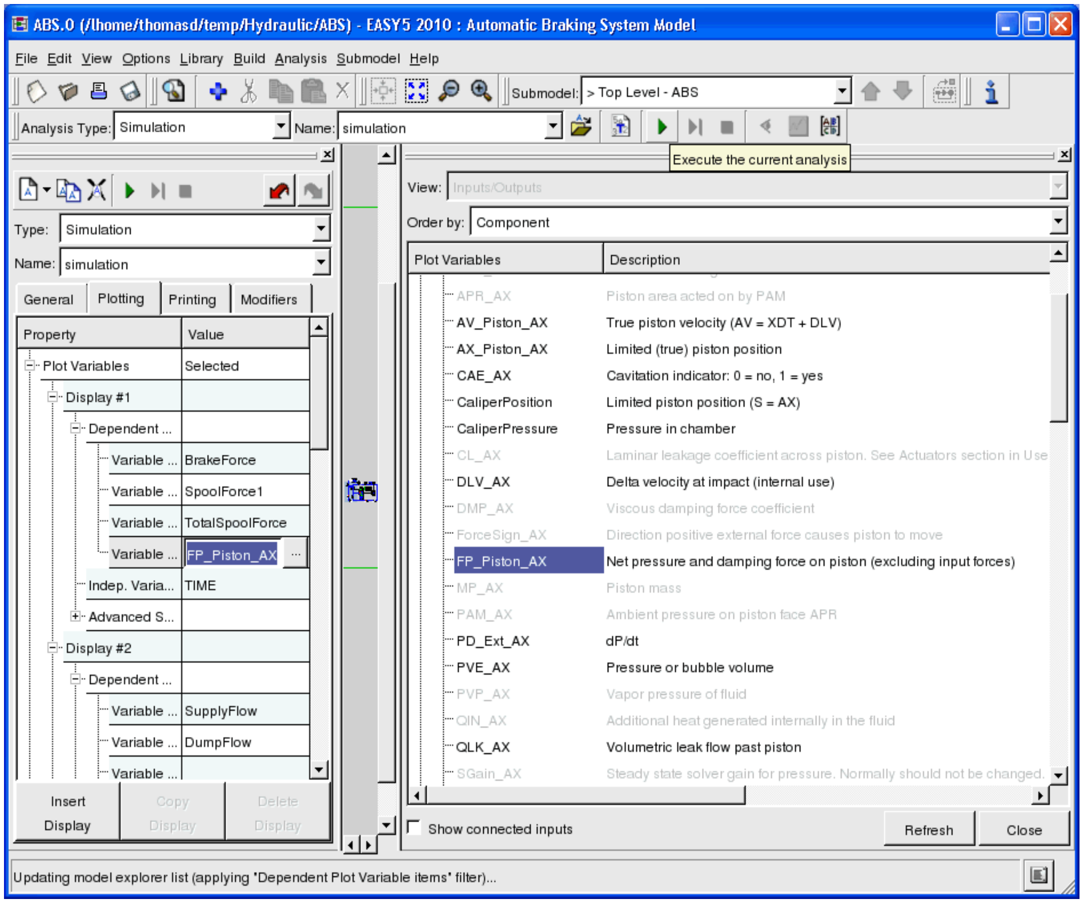
[10:14:41]: Model rebuild needed to utilize updated EASY5 2010 (v8.2) object libraries.

Figure 106 Schematic and Model Explorer Windows

Pick Method

In addition, to its use as a general navigation tool, the Model Explorer window is also used for any “picking” operation. Such operations are used whenever a model name is needed for some data entry field. This is commonly used to specify both Temporary Settings files, and many analysis settings forms. In this case, once you have selected a “pickable” field (with an ellipsis button offered next to the field), the Model Explorer is utilized by applying a specific filter to your model. Only names that follow the filter will be shown in the Model Explorer (other names will appear dimmed out).

For example, when specifying output variables for plotting, selecting either the pickable field, or its ellipsis (“...”) button, automatically opens a Model Explorer window (if needed), with a specific filter applied resulting in a navigable list of all model output variables available for plotting. If you click on an output variable from this list, it is inserted into the value column of a simulation form as shown in the example for Variable #4 (`Actuator_pos`) in Figure 107.


ABS.0 (/home/thomasd/temp/Hydraulic/ABS) - EASY5 2010 : Automatic Braking System Model

Property	Value
Plot Variables	Selected
Display #1	
Dependent ...	
Variable ...	BrakeForce
Variable ...	SpoolForce1
Variable ...	TotalSpoolForce
Variable ...	FP_Piston_AX
Indep. Varia...	TIME
Advanced S...	
Display #2	
Dependent ...	
Variable ...	SupplyFlow
Variable ...	DumpFlow
Variable ...	

Plot Variables	Description
APR_AX	Piston area acted on by PAM
AV_Piston_AX	True piston velocity (AV = XDT + DLV)
AX_Piston_AX	Limited (true) piston position
CAE_AX	Cavitation indicator: 0 = no, 1 = yes
CaliperPosition	Limited piston position (S = AX)
CaliperPressure	Pressure in chamber
CL_AX	Laminar leakage coefficient across piston. See Actuators section in Use
DLV_AX	Delta velocity at impact (internal use)
DMP_AX	Viscous damping force coefficient
ForceSign_AX	Direction positive external force causes piston to move
FP_Piston_AX	Net pressure and damping force on piston (excluding input forces)
MP_AX	Piston mass
PAM_AX	Ambient pressure on piston face APR
PD_Ext_AX	dP/dt
PVE_AX	Pressure or bubble volume
PVP_AX	Vapor pressure of fluid
QIN_AX	Additional heat generated internally in the fluid
QLK_AX	Volumetric leak flow past piston
SGain_AX	Steady state solver gain for pressure. Normally should not be changed.

Updating model explorer list (applying 'Dependent Plot Variable items' filter)...

Figure 107 Model Explorer Window

Model Files

A summary of the model files created and saved in your working directory is given in the table below. This describes each file and also indicates which files can be safely deleted without compromising your model. The deleted files can always be regenerated.

Table 1-20 Data Displayed in General and Exponential Formats

File name	Description	Can delete?
<*model name*>`.v.ezmf`	This file contains the version of the model schematic, including the components, connections, data, Fortran component code, and connection data tables and data forms. <model name> is the name of your model, and v is the version number. A new version of this file is created whenever you save model data or create executable.	No
<*model name*>`.info.doc`	This file documents the <*model name*>*.info.xxx* model file, and can be either a *.info.txt*, *.info.pdf*, or *info.htm* file	Yes
<*model name*>`.v.ezadb`	This version-specific file contains the analysis settings database, matching the model file with the same version, v.	Yes
<*model name*>`.ezmod`	This file contains the model defined in Easy5 language and is created each time you create an executable. This file is overwritten during successive create executable requests for the same model.	Yes
<*model name*>`.ezmgl`	This file contains the model generation listing written by Easy5 each time you create an executable. This file is overwritten during successive create executable requests for the same model.	Yes

Table 1-20 Data Displayed in General and Exponential Formats

File name	Description	Can delete?
<model name>`.ezerr`	This file contains the error messages describing problems that occurred when you requested Easy5 to create executable. This file exists only if errors were found, and it is overwritten during successive create executable requests for the same model.*	Yes
<model name>`.f`	This file contains the source code of the Fortran program written by Easy5 each time you create an executable. This file is overwritten during successive create executable requests for the same model.	Yes
<model name>`_c.c`	This is the C source code file generated only if the model has one or more C components.	Yes
<model name>`.exe`	This file contains the model's current executable. This large binary file is built when you request Easy5 to create executable. This file is overwritten during successive create executable requests for the same model.	Yes

Modeling Fundamentals

There are many different methods of modeling a system. This section will present four different methods commonly used in Easy5 to model systems. The four methods are:

- block diagram method
- component diagram method
- user-defined component method
- state variable method

Example Problem

In the sections that follow, a simple mass spring damper system will be used as an example of how to build Easy5 models. The mass spring damper system is shown in Figure 108. The free body diagram of this system is also drawn showing the external forces acting on the mass.

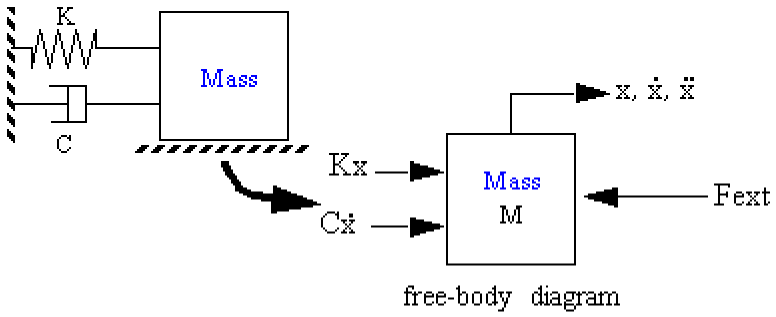


Figure 108 Mass Spring Damper System

Summing all forces acting on the body, results in the following differential equation:

$$M\ddot{x} + C\dot{x} + Kx = Fext$$ Equation 1

Block Diagram Modeling Method

The mass spring damper model can be built by using many “blocks” to model the equation. This is called the “block diagram” method because it models the system using primitive blocks. Each block models a simple equation. In Easy5, the term “component” is used. A block is just a simple “primitive” Easy5 component. A component that models a simple equation and has simple inputs and outputs. To use blocks to model the spring mass damper system, just solve for acceleration in the previous equation.

accel=(1/mass)*(Fext - C*velocity -K*position)

Then, integrate acceleration (accel) and velocity to get the resulting position. Simple summation, gain and integration blocks can be used to model this as shown in Figure 109. Generally, this block diagram method is not recommended. Six components are needed to model the simple mass spring damper system. In general, the component modeling method is preferred, as described in the next section.

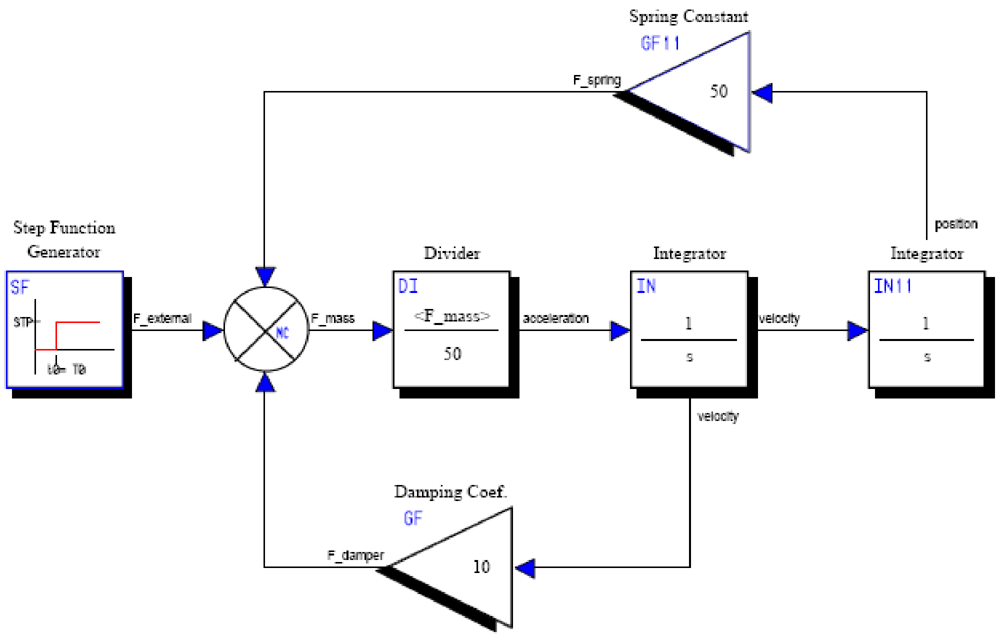


Figure 109 Block Diagram Method of Modeling Spring Mass Damper System

Component “Systems Diagram” Modeling Method

The component method of modeling uses Easy5’s unique “systems diagram” approach to modeling systems. Unlike blocks, Easy5’s components allow you to model entire systems using a single component. These components may contain many inputs and outputs, and connections between components may be multi-input/output connections, and bidirectional. Many Easy5 components model complex systems, such as heat exchangers, hydraulic valves and aircraft dynamics.

To apply the component modeling method, you should first identify prebuilt components that match the functionality you require. The most general components are contained in the “GP” (general purpose) library. If the GP library does not contain a matching component, you may find that a different Easy5 library may have the required component. For example, if you are modeling an air to air heat exchanger, the “ec” (environmental control) library contains several different models of heat exchangers.

If you cannot find a component that exactly matches your application, you may need to combine several components or alter them. Complicated engineering blocks must often be broken down into simpler blocks in order to match them up with the functionality provided by standard components. Figure 110 shows how a block diagram can be simplified.

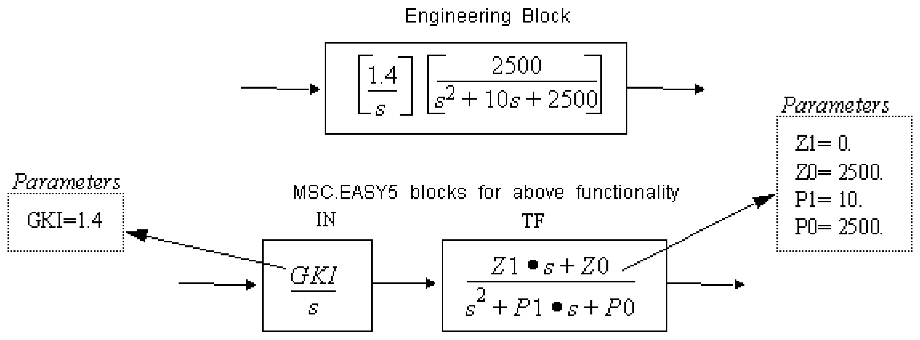

Engineering Block
$\left[\frac{1.4}{s}\right]\left[\frac{2500}{s^2+10s+2500}\right]$
Parameters
GKI=1.4
MSC.EASY5 blocks for above functionality
IN
TF
$\frac{GKI}{s}$
$\frac{Z1 \bullet s + Z0}{s^2 + P1 \bullet s + P0}$
Parameters
Z1= 0.
Z0= 2500.
P1= 10.
P0= 2500.

Figure 110 Simplifying an Engineering Block Diagram

Easy5 does not have a prebuilt component that specifically models a mass spring damper. However, the system defined by the differential equation of Equation 1 can be transformed to a LaPlace equation and then to a transfer function as follows:

Differential Equation: $M\frac{d^2x}{dt^2} + C\frac{dx}{dt} + Kx = f(t)$

Apply LaPlace Transform: $Ms^2X(s)+CsX(s)+KX(s)=F(s)$

$X(s)[Ms^2+Cs+K]=F(s)$

Transfer Function:

$$\boxed{\frac{X(s)}{F(s)} = \frac{1/M}{s^2+(C/M)s+K/M}}$$

Equation 2

Easy5 contains many components that model first order, second order, and nth order transfer functions. Several components model the transfer function defined by Equation 2.

One of these is the MD (Modal Dynamics) component, as shown in Figure 111.

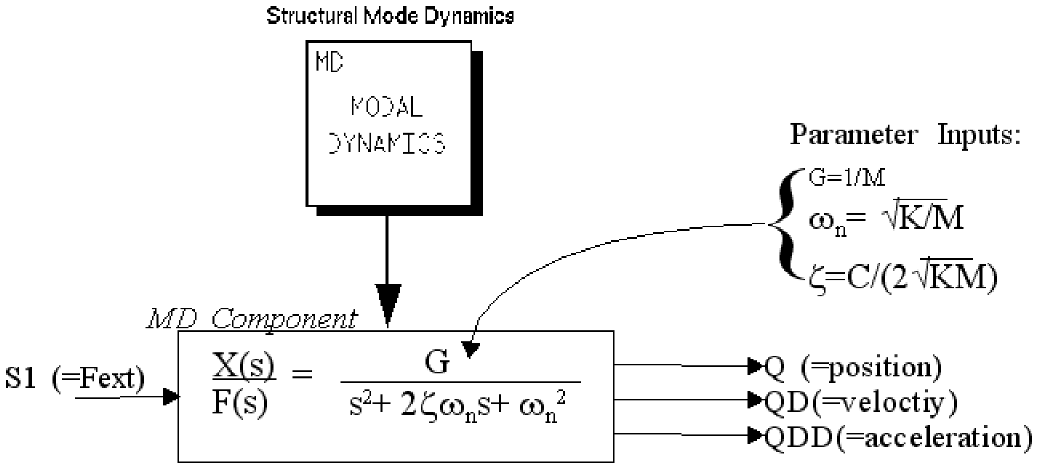


Figure 111 Component Method of Modeling Spring Mass Damper System

Notice that the MD component was not created to specifically model a mass spring damper system. However, the mathematical structure of the MD component matches the transfer function defined by Equation 3. The component's input parameters can be used to enter the mass (M), spring coefficient (K) and damper coefficient (C) as shown. This component has one input and three outputs This example illustrates the difference between the systems diagram approach of *component* modeling versus *block* modeling. One component is needed using the systems diagram approach compared to the 6 components used in the block diagram approach.

The MD is a standard component from the GP library. The limitation in using this is that the parameter inputs and names do not match the mass spring damper parameters. You can easily create your own user-defined component that specifically models the mass spring damper. This can either be a Fortran, C, or a Library component, as described in the next section.

User-defined Component Modeling Method

You can create your own component to model a system using either Fortran, C, or Library components. Using a Fortran or C component, you can code the differential equations, and define input/output names and values. A library component is similar to a Fortran or C component, but allows you to customize the component by adding it to a library, creating your own icon and info page, and building in connection specifications. From the users point of view, a Library component is just like any other standard component in that it is managed in a library and may be added multiple times to a model.

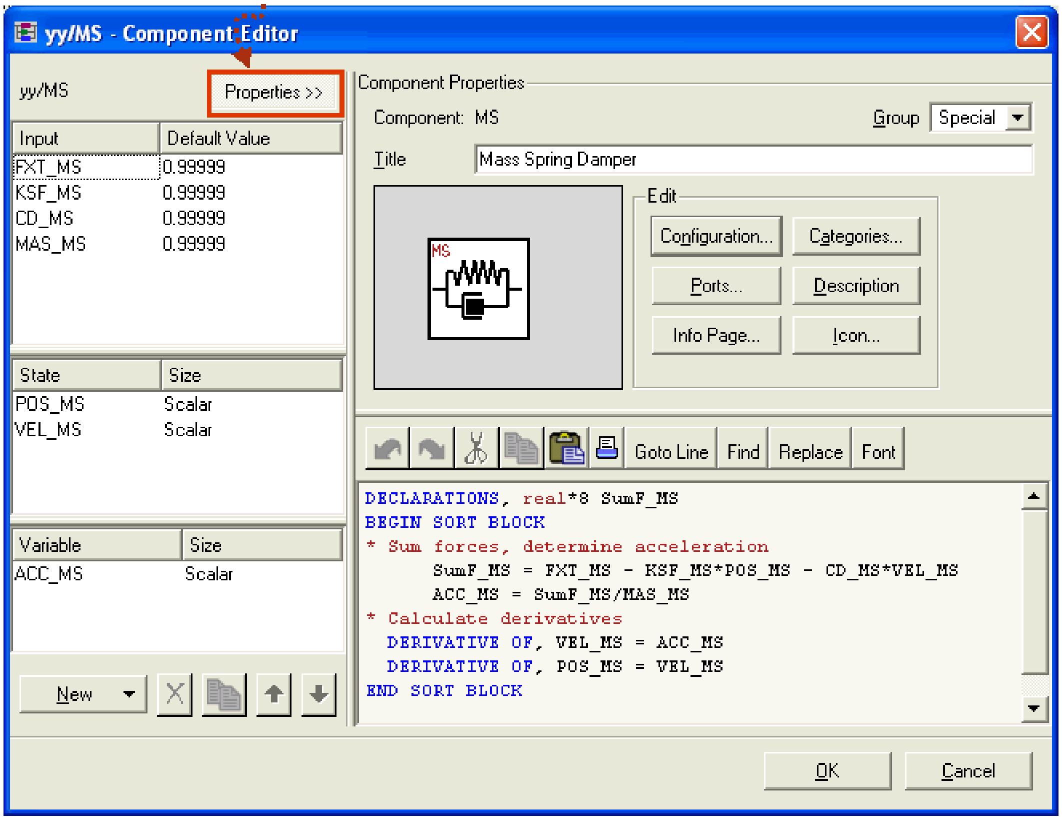
The details of creating a Library Component is given in the User Guide, Chapter 11 - Library Developer Toolkit This section will only show the end result of creating a library component.

Figure 112 shows an example of a Library Component. This figure shows the Library Component Editor with the input/output names and code editor panel with the code used to define the dynamics. With minimal experience, you should be able to build this user-defined component in a matter of a few minutes. Notice in the code that, not including the comment lines, only four lines of code are needed to define the dynamics.

First, you calculate the summation of all forces (SumF), then calculate acceleration (ACC), then integrate this once to get velocity (VEL), and integrate again to get position (POS).

Having built the Library Component, you may create your own icon and online documentation (info page). This component can then be added to any model and can be shared with other users. This modeling feature is unique to Easy5, and sets Easy5 apart from similar software.

The "Properties >>" button is used to switch from input, State or Variable properties to Component Properties



Input	Default Value
FXT_MS	0.99999
KSF_MS	0.99999
CD_MS	0.99999
MAS_MS	0.99999

State	Size
POS_MS	Scalar
VEL_MS	Scalar

Variable	Size
ACC_MS	Scalar

Component: MS   Group: Special

Title: Mass Spring Damper

```
DECLARATIONS, real*8 SumF_MS
BEGIN SORT BLOCK
* Sum forces, determine acceleration
      SumF_MS = FXT_MS - KSF_MS*POS_MS - CD_MS*VEL_MS
      ACC_MS = SumF_MS/MAS_MS
* Calculate derivatives
  DERIVATIVE OF, VEL_MS = ACC_MS
  DERIVATIVE OF, POS_MS = VEL_MS
END SORT BLOCK
```

Figure 112 Library Component Editor for Example Mass-Damper-Spring Component

State Variable Modeling Method

The final method is to create a state variable linear model of your system. This method assumes that you are building a linear model defined by the linear equations:

$$\dot{x} = [A]x + [B]u$$

$$y = [C]x + [D]u$$

where

x = state vector

u = input vector

y = output vector

[A] = system (plant) matrix

[B] = input matrix

[C] = output matrix

[D] = feed through matrix

This requires that you build the system [A, B, C, and D] matrices. To do this, you must first break the nth order differential equations into “n” 1st order differential equations. In this example, the dynamics are defined by a 2nd order differential equation. Break this into 2 first order equations as follows:

Differential Equation: $$M\frac{d^2x}{dt^2} + C\frac{dx}{dt} + Kx = f(t)$$

$$\frac{d^2x}{dt^2} = \frac{Fext}{M} - \left(\frac{C}{M}\right)\frac{dx}{dt} - \left(\frac{K}{M}\right)x$$

Define x_1 and x_2: $\{$

$$x_1 = \frac{dx}{dt} \quad \therefore \quad \dot{x}_1 = \frac{d^2x}{dt^2}$$

$$x_2 = x \quad \therefore \quad \dot{x}_2 = \frac{dx}{dt}$$

u=Fext

State Equations: $$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-C/M & -K/M\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1/M\\ 0\end{bmatrix}\begin{bmatrix}u\end{bmatrix}$$ Equation 3

The final state variable equation is defined by Equation 3. This can be modeled using the *SR - State Variable Model Reduced Format* component.

This component is used to model linear models using the state variable format. Just add SR to your model, and define the [A, B, C] matrices. This component is shown in Figure 113.

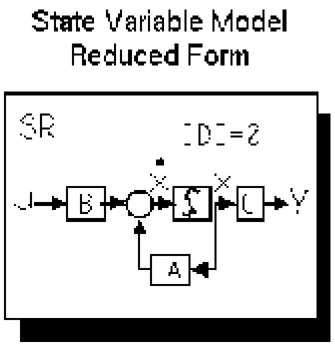


Figure 113 Example of the State Variable Modeling Method

There are many ways to build models in Easy5. Four of the most common methods were presented in the previous sections. The component systems method and the user-defined component method are the recommended methods for building models in Easy5. In the long run, building your own user-defined components (Library Components) is by far the best method. This allows you to document the component with your own icon and info page, and most important, you can manage the component in a library, which allows you to share your components with others.

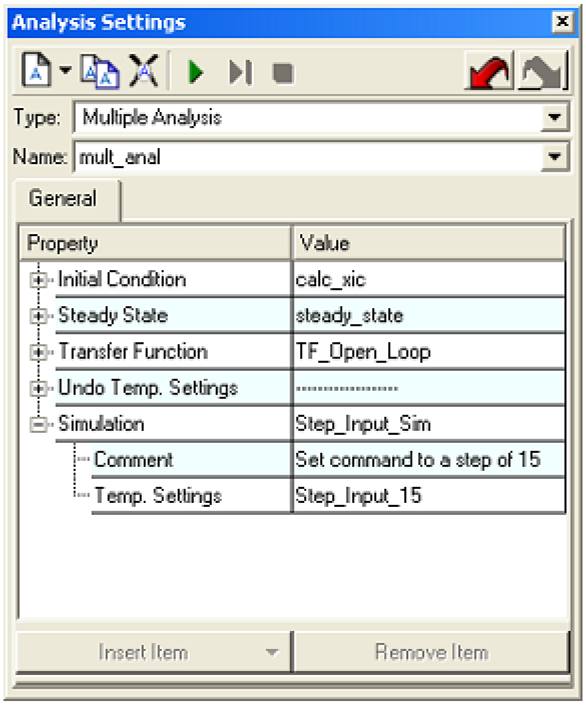
Multiple Analysis

Easy5 allows you to perform multiple analyses. This feature allows you to set up a large number of analyses in one data form and then execute them as one large job. With this feature, you have the option of defining and executing a series of analyses all at once, rather than defining and executing them individually as you have done in the previous sections.

The Multiple Analysis feature is very useful in many ways. First, it allows you to utilize your time and the computer resources more efficiently. For example, you can submit a large multiple analyses job overnight or over a weekend. The execution will occur during low computer usage time, and the results will be available to you at your convenience. Multiple Analysis also allows you to define a standard set of analyses that can subsequently be repeated as a group.

Setting up Multiple Analysis

Multiple analyses are setup and executed via the multiple analysis setup form. A completed example of this data form is shown in Figure 114.



Type: Multiple Analysis

Name: mult_anal

Property	Value
Initial Condition	calc_xic
Steady State	steady_state
Transfer Function	TF_Open_Loop
Undo Temp. Settings	....
Simulation	Step_Input_Sim
Comment	Set command to a step of 15
Temp. Settings	Step_Input_15

Figure 114 Multiple Analysis Data Form

This form contains fields used to define the type of analyses to run (property), the name of the analysis file associated with those analyses, temporary settings file names, if any, and an additional analysis titles.

When the Multiple Analysis Data Form first displays, no analysis is included. As you add inputs, additional lines are automatically added to the form.

To setup and execute a multiple analysis you need to do the following:

1. Select the analysis type (Simulation, Steady-State, etc.) and execution order for the analyses you wish to run by selecting the Insert Item button to open the pop up menu (Figure 115).

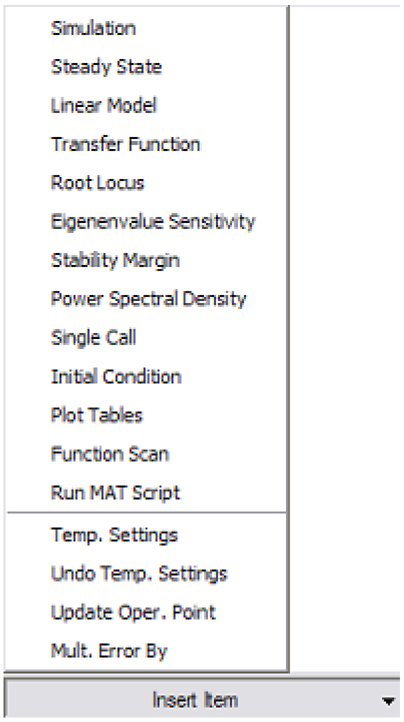


Figure 115 Multiple Analysis Pop-up

2. For each analysis inserted, enter the name of the analysis settings file that contains the respective analysis setup instructions.
3. (Optional) For each analysis inserted, enter the name of a temporary settings file if you want to over ride any parameter, initial condition, or error control values in the model.
4. (Optional) For each analysis inserted, enter an additional analysis title that supplements the title in the respective analysis data form.
5. Execute the set of analyses.

The following sections provide detailed descriptions on how to complete these steps using the fields in the multiple analysis setup form. Prior to performing a multiple analyses, you should first have Easy5 calculate the initial conditions. When you submit a single analysis data form, Easy5 automatically calculates the initial conditions prior to performing the analysis.

Understanding the Setup Form

The Multiple Analysis Data Form allows you to enter an unlimited number of analyses. Analyses are executed in the order defined. The push buttons located at the bottom of this form are the same as the other analysis data forms in addition to the "Insert" and "Delete" buttons. To insert or delete an analysis, first select the analysis number (the corresponding number in the first column) and then select the insert or delete push button. Multiple analyses may be deleted at one time by selecting more than one analysis number.

Defining an Analysis Property

The Property column is used to define the type of analyses you want to execute. To enter an analysis type, point to the open field in this column and select it to display the Easy5 Analysis Types pop-up shown in Figure 115. Select the analysis you want from this menu and it will be copied into the setup form for you. For every analysis type you specify, you must also specify a respective analysis file name.

The section that follows describes this process in detail.

In the **Multiple Analyses...** menu there are four additional options available as follows:

1. **Calc. initial conditions** - In the case of single analyses (those executed directly from analysis data forms) Easy5 always sets a variable named ICCALC to 1 and calls your model once. By testing on ICCALC in your user-code you can define initial condition values and/or perform any initialization tasks.

   If you use this option you can set up the initial condition calculation using the data form as described in "Initial Condition Calculation".

   By default, this analysis is provided as the first analysis of a multiple analysis run.
2. **Update I.C./Operating Point** - This option loads the current set of state values into the initial condition vector and allows to you use the new operating point for all analyses that follow. (This option does not over-write the initial conditions in the model's component data tables, only the "Restore operating point" option in the Options menu can do this.)

   This option is often used after a Steady-State analysis or a simulation to load a steady-state operating point into the initial condition vector. Initial conditions defined in this manner remain in effect until another "Update initial condition" command is issued or the multiple analysis session ends.

Note:	Prior to performing multiple analyses, you should first have Easy5 calculate the initial conditions. When you submit a single analysis data form, Easy5 automatically calculates the initial conditions prior to performing the analysis. However, an initial conditions calculation is not implicitly performed when submitting analyses via the multiple analysis data form. Instead, by default the first analysis for a multiple analysis run is explicitly provided as **Calc. initial conditions**.

3. **Mult. error by** - This option is a convenient way to multiply all the error control values for all of the states in your model by a constant value. See the subtopic under Integration "Guidelines for Setting Error Controls" for more information about error controls. For example, if the error control values in your model are too large you could use this option to multiply them by.1.

When this option is selected, you will be prompted to enter a multiplier. After you enter a number and press RETURN, Easy5 will copy the line "Mult. error by xx" in the setup form, where xx is the value you entered.

This option does not change any of the error control values in any of the model's data tables. Therefore, changes made to the error control values remain in effect only during the multiple analysis session or until they are changed by another "Multiply error by..." request.

4. **Undo temp settings** - This option allows you to change all parameter, initial condition and error control values back to the values in the model's component data tables. This option is used when you have been using temporary settings files and want to return to default values.

Caution: Temporary Settings Files are additive! Once a Temporary Settings File has been applied, the change remains in effect throughout subsequent analyses in a Multiple Analysis. To remove the effect of all accumulated Temporary Settings File, use the **Undo temp settings** feature.

Specifying an Analysis Settings File

For every analysis listed in the "Type" column you must also specify an analysis file describing the analysis to be performed. Analysis files are defined in the column labeled "Analysis File." Analysis files contain analysis information and are created when you perform a save operation in any of the analysis data forms.

After selecting the type of analysis for the "Type" column, Easy5 displays a list of analysis files that currently exist for the analysis type specified. Select the analysis file you want from this menu with your left mouse button and it is copied into the data form. You can also create a new analysis file by selecting the "New" push button located at the bottom of the menu. A dialog box prompts you to enter a new Auxiliary Input File name. Selecting "OK" transfers the name into the data form.

Once you have entered an analysis file you can open up the respective data form by selecting the analysis name with the center mouse button. When the data form opens you can change the analysis definition just as if it were a "single execution" analysis. This feature allows you to enter a new analysis file name and set up an analysis from scratch directly from the multiple analysis data form.

Defining a Temporary Settings File

For every analysis specified, you have the option of defining a set of temporary parameter and initial condition values that will over ride the default values (those contained in the model's component data tables) during the analysis. Temporary settings are defined in the respective analysis data form as described in the "Temporary Settings File".

When setting up a multiple analysis you can define another set of over riding temporary values for any of the analyses listed. Additional sets of temporary values are defined in the multiple analysis data form in the column labeled "Temp Settings File." If a temporary settings file is already being used, and you specify another temporary settings file, both sets of values will be applied to the model, with the original set applied first, and the multiple analysis set applied second.

To specify a temporary settings file, highlight the field and select it to display a list of temporary settings files that already exist for the respective analysis file. From this list select a file name or use the "New" option to define a new file.

Once a temporary settings file name has been entered, you can open up the respective file by highlighting the name and selecting it with you center mouse button. See "Temporary Settings File" for more information about placing values in temporary settings files.

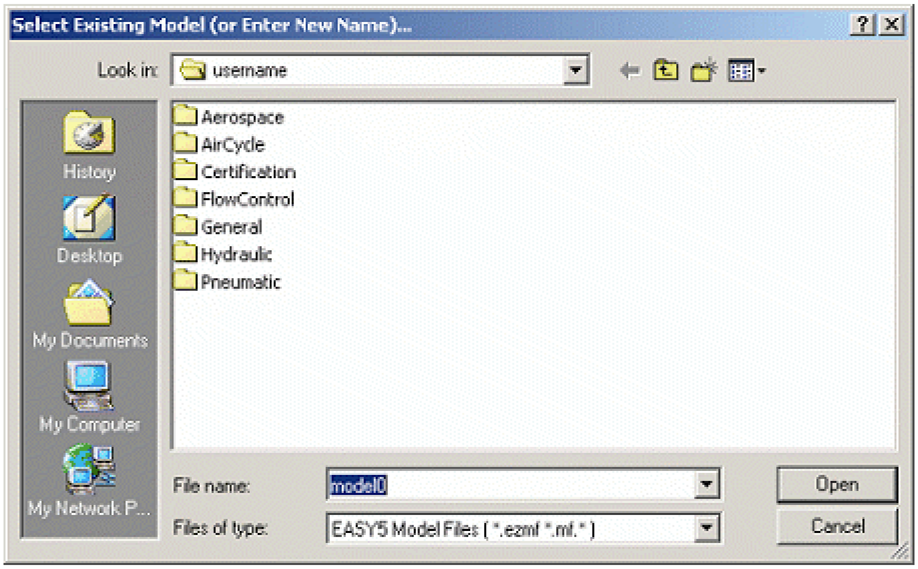
Specifying Multiple Analysis Titles

You can specify a title for every analysis listed in the multiple analysis setup form. This title will be combined with the title in the respective analysis data form to label analysis output. Specifically, the combined title is a combination of the multiple analysis title followed by a dash.

Open Model Dialog

After Easy5 starts, the *Open Model* dialog displays.

This dialog allows you to change directories, define a new model and open an existing model. A default model name is always pre-assigned, so if you don't care what it should be named, simply select the [OK] button. This dialog utilizes a native File dialog, with an example for Windows shown below in Figure 116.



Aerospace
AirCycle
Certification
FlowControl
General
Hydraulic
Pneumatic

File name: model0
Files of type: EASY5 Model Files (*.ezmf *.mf.*)

Figure 116 Open Model Filter Dialog

Bypassing the Open Model Dialog

If you are resuming work on a previously built model *from a command shell*, you start Easy5 with a particular model by entering the following command at the prompt:

```
easy5x <model_name>
```

where: *model_name* is the name of a previously built model, file, or path. If you enter only the model name, the highest version will be opened; you may include the *v.ezmf* extension to open a specific version of the model. Of course, this means you must start Easy5 from a command shell. If you include a full path, Easy5 will open the specified model in that path.

When a *model_name* argument is provided Easy5 will start up and bypass the Open Model dialog shown in Figure 116, and load the model specified.

Note:	**Windows:** You can open any Easy5 model via an Explorer Shell by double-clicking on the appropriate Easy5 model file.

Starting a New Model

To start a new model, select **File** > **New** and enter a model name in the appropriate field. A new, empty model will be created and named accordingly.

The model name must conform to the following conventions:

- Model name must use alphanumeric characters, with 32 characters or less.
- do not use embedded blanks, instead, use an underscore (__).
- do not use periods "." in the name.

Opening an Existing Model

If you wish to open an existing model, select File > Open, navigate to the appropriate folder, and select the existing model name from list provided. Either select it with a double-left-click which automatically opens the model, or with a single mouse left-click and then select the `[OK]` button.

Opening a Damaged Model

If an Easy5 model is damaged, you may encounter a problem loading it. When this occurs, Easy5 will display a warning message informing you of the fact that the model is damaged. After you acknowledge the warning, Easy5 will attempt to correct the problem by rereading the model in a so-called "corrective-read" mode. If this fails, please contact Easy5 Support, and provide an attached copy of the damaged model file.

Checking Background Processes

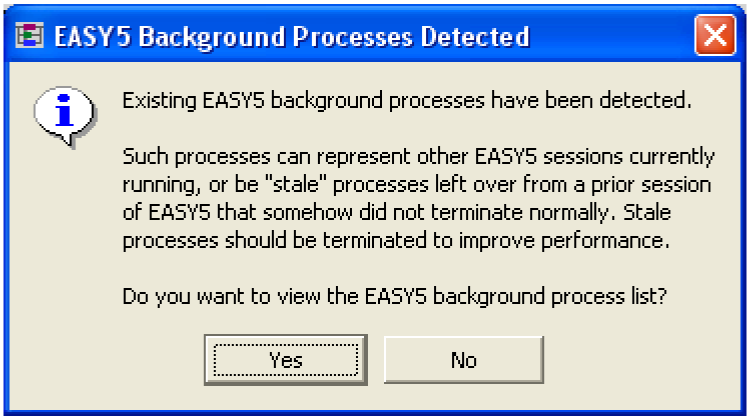
After an initial model has been opened following the launch of the Easy5 graphical user interface, a check is made on any background processes identified with your username that are currently running. Normally, you would not expect that Easy5 background processes are already running during initialization.

However, if background processes are running, it could mean one of three things:

1. You previously launched some Easy5 analyses and exited the graphical user interface before they completed (which is OK).

2. You are running other Easy5 jobs on this computer, which is OK as long as you are running them in a different directory.
3. Jobs launched from Easy5 are no longer working -- they may have halted for some reason -- and related processes are still running, although it is doubtful that they are doing anything. These are called “stale” processes.

This background processes check is done primarily to alert you of the latter case: potential stale background processes that may be running. Such processes can potentially interfere with Easy5’s ability to run analyses, so Easy5 notifies you with the following dialog as shown in Figure 117:


EASY5 Background Processes Detected

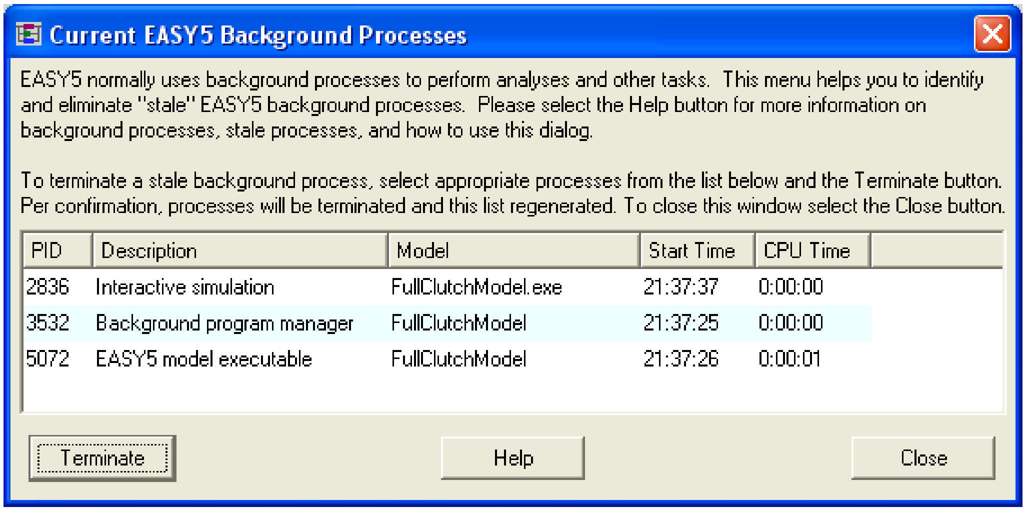
Existing EASY5 background processes have been detected.

Such processes can represent other EASY5 sessions currently running, or be "stale" processes left over from a prior session of EASY5 that somehow did not terminate normally. Stale processes should be terminated to improve performance.

Do you want to view the EASY5 background process list?

Figure 117 Background Process Detection Dialog

If you select **Yes**, a Background Process dialog is displayed, allowing you to view and terminate (stale) background processes.


Current EASY5 Background Processes

EASY5 normally uses background processes to perform analyses and other tasks. This menu helps you to identify and eliminate "stale" EASY5 background processes. Please select the Help button for more information on background processes, stale processes, and how to use this dialog.

To terminate a stale background process, select appropriate processes from the list below and the Terminate button. Per confirmation, processes will be terminated and this list regenerated. To close this window select the Close button.

PID	Description	Model	Start Time	CPU Time
2836	Interactive simulation	FullClutchModel.exe	21:37:37	0:00:00
3532	Background program manager	FullClutchModel	21:37:25	0:00:00
5072	EASY5 model executable	FullClutchModel	21:37:26	0:00:01

Figure 118 Background Processes Check

Please see "Background Processes" for further details on Easy5 background processes, and using the Background Process list dialog.

Operating Point

The starting point, or operating point, for all Easy5 analyses is defined by the set of initial conditions you define. Initial condition values for the respective states are defined in the component data tables containing these states. Initial conditions for a simulation define the point from which Easy5's integration algorithms begin. For linear analyses, the initial conditions define the point about which linearization occurs. When you use one of the Easy5 linear analyses to evaluate model stability, it is usually assumed that the operating point is at equilibrium. An equilibrium point (also referred to as steady-state) is a point where the first derivatives of all states are zero.

Note: For operating points saved by the main application, the values of model *variables* are also included in an operating point. This allows such values to be displayed in the schematic, and permits a consistent restart point. Such operating points are only valid for a given set of inputs and initial conditions.

Creating an Operating Point

In general, specifying an equilibrium point for your model is a primary task of model characterization, and this can be accomplished using any of the methods described below.

- Load initial conditions manually
- Calculate initial conditions in a Fortran component
- Have equilibrium initial conditions automatically calculated and loaded for you by Easy5 (i.e., use the Steady-State analysis)
- Copy the final state vector from a simulation into the initial condition vector for use in the next analysis

Manually Load Operating Point Data

You can load initial conditions manually by "examining" (opening) the appropriate component data tables (i.e., those associated with components that calculate state variables), and entering initial condition values in the "(IC) Value" column. Only nonzero values require loading, since all initial conditions are initialized to zero.

Define the Operating Point in a User Code Component

You can calculate values for one or more initial conditions with a User Code component. Before Easy5 executes any of its analyses, it sets a common variable, an Easy5 "reserved word," named ICCALC to one, and calls your model once. By testing on the value of this variable, you can execute an appropriate section of code and set values for any states you wish. After the model is called, the state vector is simply copied into the initial condition vector; no integration occurs. See “Reserved Words” for information on the ICCALC variable.

Define the Equilibrium (Steady-State) Operating Point

You can have Easy5 calculate the equilibrium (steady-state) operating point for you by running a Steady-State analysis. To do this, in the Steady State Analysis Data Form, save the operating point by setting the “Save Final Operating Point” button to “Yes”. This will open a data field that prompts you for a file name. Enter a file name used to save the operating point. The Steady-State analysis is discussed in “Steady-State Analysiss”.

Define an Operating Point from a Simulation

Easy5 can save the operating point at the end of a simulation analysis to a specified file. This is done by setting the “Save Final Operating Point” button to “Yes” in the Simulation Analysis Data Form, which opens a data field prompting you enter a file name.

Note: Normally, an Easy5 operating point is defined purely by the value of all initial conditions for states in your model. For sampled-data systems, this can be slightly different, if the time at which the operating point is saved is a non-fundamental sampling rate interval of simulated time. In such a case, Easy5 additionally saves the values of the rates of all active digital states in the operating point file. This is done to allow you seamless stop and restart for sampled-data simulations using the familiar paradigm of an Easy5 operating point as a valid starting point.

Operating Point Files

Operating Point files can be created and then used either to update the model values, or as a starting point for any analysis.

Creating an operating point file from the state initial condition values defined in the components:

1. Select the **Options** > **Save Operating Point** menu. This will open a dialog requesting an operating point file name.
2. Enter a name.

This feature will copy all of the state initial condition values currently defined in the model’s components, into a operating point file. This kind of operating point also saves values for all model *variables*. You may create as many operating point files you wish.

Generating an operating point file from a Steady State analysis or Simulation Analysis:

1. From the analysis form, set the “**Save Final Operating Point**” setting to “**Yes**”.
2. Enter a file name used to save the operating point in the appropriate field.

This kind of operating point does not contain values for model *variables*.

Loading operating point data directly back into the model, or using it as a starting point for an analysis:

1. Select the **Options > Restore Operating Point...** menu to select an operating point file and load the data from that operating point file.
2. Select a file to load the operating point data directly into your model and update all state initial condition data.

To use the operating point from an operating point file as a starting point for any analysis:

1. Select the "**Initial Operating Point**" data field. Every analysis has this data field at the top of the analysis data form. A pull-down list offers all operating point files associated with your model.

If you select a file, the operating point data will be loaded in only for the single analysis and used an the initial condition vector; this does not modify the model, it only updates the state data used as a starting point for the analysis.

2. If TIME is tagged with the operating point, you will be prompted to decide whether to update and use this time. If your model has time dependent functions then this is important; you may need to use the time tagged with the operating point data. Otherwise, you may select "No".

For example, if you generated the operating point file by performing a simulation to 20 seconds and saving the final operating point, when you attempt to restore this operating point in a subsequent analysis, it will tell you the "Time=20" is defined with the operating point. If you have tabular functions of time or sine/cosine functions of time, select "Yes" to restore TIME with the operating point.

Editing the Operating Point File

When you save an operating point to a file, the file is saved to your directory and named:

<*model_name*>.<*op_ID*>`.ezic`

This is a simple text file that you can view and edit. Use the Easy5 Text Editor to look at the data by selecting **File > Open Text Editor...** . Please see "Steady-State Analysis Method" for more information on the Easy5 Text Editor.

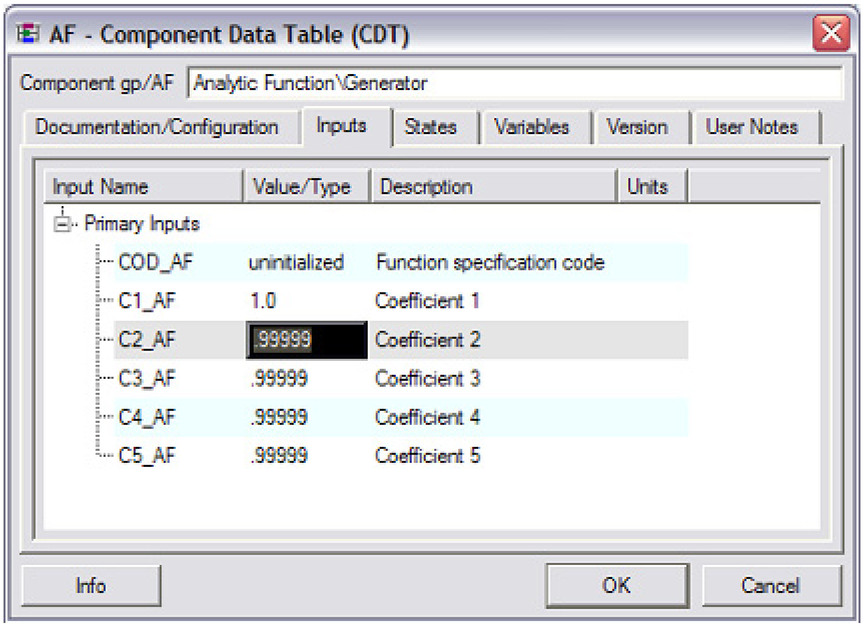
Parameters - Defining Input Values

See Also: "Component Data Table"

Parameters are defined in the CDT's Inputs category. Each unconnected input must be given a constant value, or it will be assigned a default value of .99999 by Easy5. Parameters are scalar, vector, or array quantities, depending upon whether or not the respective component has been vectorized.

To specify scalar parameter values, open the Component Data Table. In the Values/Type column, next to the corresponding component input name, point to the ".99999" value. When the value you selected is highlighted, type in the appropriate scalar value and then press Enter.

Pressing Enter moves the highlight to the next data field in the Values column Figure 119.



AF - Component Data Table (CDT)

Component gp/AF: Analytic Function\Generator

Tabs: Documentation/Configuration | Inputs | States | Variables | Version | User Notes

Input Name	Value/Type	Description	Units
Primary Inputs			
COD_AF	uninitialized	Function specification code	
C1_AF	1.0	Coefficient 1	
C2_AF	.99999	Coefficient 2	
C3_AF	.99999	Coefficient 3	
C4_AF	.99999	Coefficient 4	
C5_AF	.99999	Coefficient 5	

Figure 119 CDT Inputs

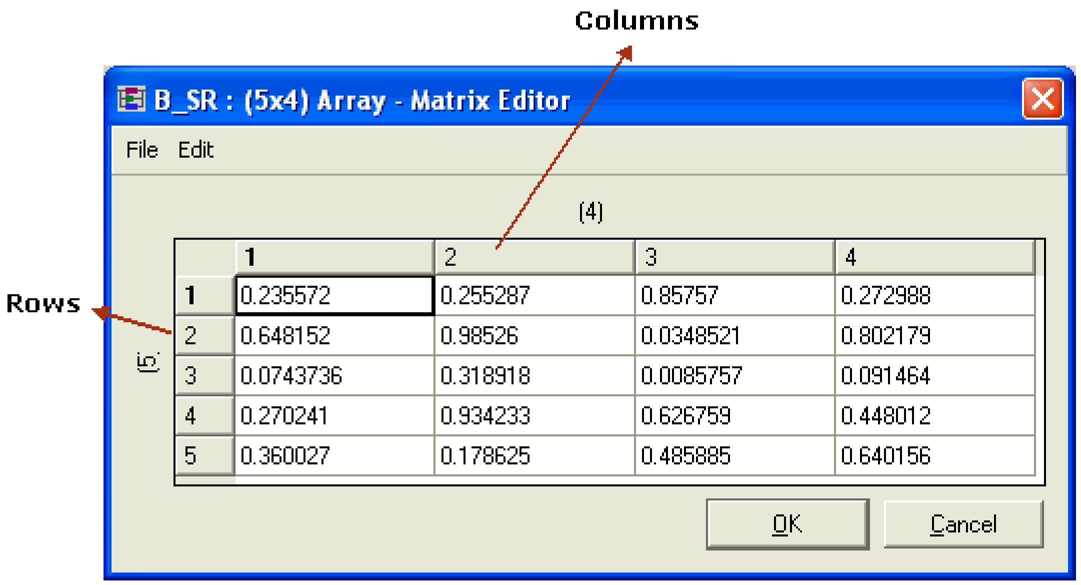
You can continue to enter scalar parameter values, or you can close the component data table and save the new parameters by selecting OK.

Note: All component input values in Easy5 are typed as double-precision real numbers. Even if you enter an integer value, Easy5 converts it to a real number.

If a component makes use of a vector parameter that has three or fewer elements, individual entries for every element of the vector appear directly in the component data table, and values are specified for them exactly as they are for scalar quantities. When vectors have four or more elements, or when arrays (matrices) are being used, you must assign parameter values by filling in the respective data table.

Matrix Editor Window

The Matrix Editor window is shown in Figure 120, displaying a `5x4` matrix. The format is similar to other spreadsheet programs. This is similar to the Table Editor in functionality, but the display differs in that it does not display the additional data fields for the independent variables.



	1	2	3	4
1	0.235572	0.255287	0.85757	0.272988
2	0.648152	0.98526	0.0348521	0.802179
3	0.0743736	0.318918	0.0085757	0.091464
4	0.270241	0.934233	0.626759	0.448012
5	0.360027	0.178625	0.485885	0.640156

Figure 120 Matrix Editor Window

A complete description of how to use the Matrix Editor is provided in the User Guide, Chapter 8 - Table and Matrix Editor. This chapter describes the Matrix Editor features and functionality.

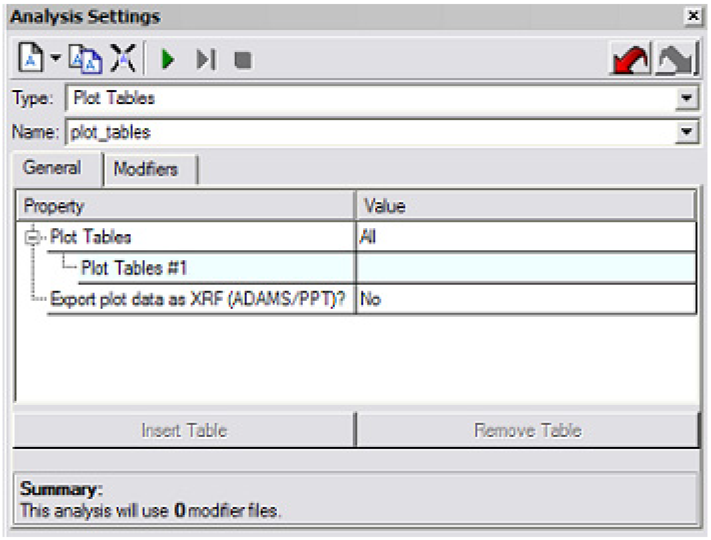
Select he OK push button located at the bottom of the table to save any changes you have made and return to the component data table. Selecting the Cancel button will return you to the component data table without saving any of your changes.

Plot Tables Analysis

The Plot Tables analysis generates plots of table data used in your model. This analysis is normally used during the initial stages of model development to verify that all tabular data in your model are free of gross errors, since it is often very difficult to detect typing mistakes just by looking at rows and columns of numbers.

Setting up a Plot Tables Analysis

The Plot Tables analysis is set up using the Plot Tables Data form. Select **Miscellaneous** > **Analysis** > **Plot Tables** from the main menu to view the form shown in Figure 121.


Analysis Settings

Type: Plot Tables

Name: plot_tables

General | Modifiers

Property	Value
Plot Tables	All
Plot Tables #1	
Export plot data as XRF (ADAMS/PPT)?	No

Insert Table | Remove Table

Summary:
This analysis will use **0** modifier files.

Figure 121 Plot Tables Data Form

Setting up this analysis involves entering the names of the tables you want plotted or using the Model Explorer “Pick” method. Either use the **Show Tables** button, or select a component from the on-screen schematic to display a list of table names for that component. Select a name in this list and it is copied into the data form for you.

To delete a table name, select the name with your left mouse button and select the “Delete” push button at the bottom of the data form.

You can generate plots of all table data in your model by selecting “All” from the “Plot Tables:" option. If you do this, all the data fields in this form disappear and Easy5 creates plots of all table data in your model when the analysis is executed.

Power Spectral Density Analysis

The Power Spectral Density analysis is used to examine the energy content of an input and output signal of your linearized system model. This is a useful tool in designing a control system because it gives you an indication of the system frequencies at which the signal energy is concentrated. These results are helpful in determining the effect that combinations of random signals and noise disturbances have on your control-system design.

See also:	"Analysis Data Form"
	"Power Spectral Density Analysis Method"

The Easy5 Power Spectral Density analysis uses an input power spectral density and a specified transfer function to calculate an output power spectral density. The transfer function is calculated from a model that is linearized about the defined operating point. You have several options for the input power spectral density, including two where you supply the form or the equations. Power Spectral Density Analysis can be performed with both continuous and sampled-data models.

The Power Spectral Density Analysis Data Form

All the specifications for performing a power spectral density analysis are given with the power spectral density data analysis form via your mouse and keyboard. To open this analysis data form, select , **Analysis > Linear > Power Spectral Density...** from the main menu. Figure 122 is an example of the power spectral density analysis data form with all analysis options selected.

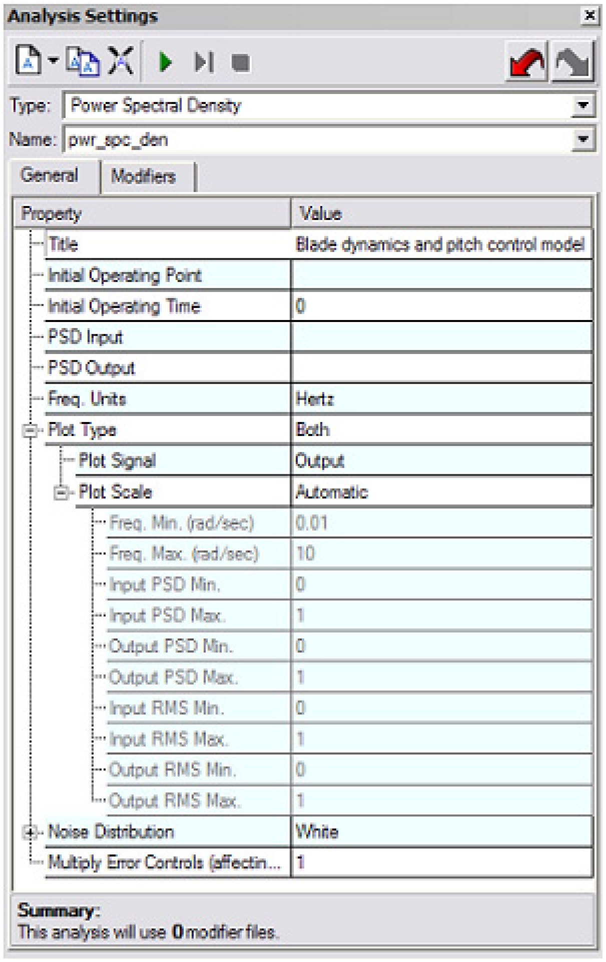
Specifying the Power Spectral Density Input

"PSD Input" can be the name of any state, variable, or parameter in the model. It represents the injection point for the input power spectral density, and it is the input part of the transfer function specification. To enter an input name, point to the respective data field, and enter the name. You can also use the "Pick" option to enter the PSD input name, described in "Model Explorer Window".

Caution:	If "PSD Input" is set to a state name, *that state must be frozen*. Otherwise, signal paths around the state will remain active, and will become part of the calculated transfer function. This will yield incorrect results. In general, better numerical accuracy will be obtained if all model states not included in the specified power spectral density are frozen. Otherwise, zeros will be calculated for the transfer function to cancel the poles corresponding to extraneous states.

Specifying the Power Spectral Density Output

"PSD Output" can be the name of any state or variable in your model. This quantity represents the measurement point for the output power spectral density, and it is the output part of the transfer function specification. To enter an output name, point to the respective data field, and enter the name. You can also use the "Pick" option to enter the PSD output name, as described in "Model Explorer Window".


Analysis Settings

Type: Power Spectral Density

Name: pwr_spc_den

General | Modifiers

Property	Value
Title	Blade dynamics and pitch control model
Initial Operating Point	
Initial Operating Time	0
PSD Input	
PSD Output	
Freq. Units	Hertz
Plot Type	Both
Plot Signal	Output
Plot Scale	Automatic
Freq. Min. (rad/sec)	0.01
Freq. Max. (rad/sec)	10
Input PSD Min.	0
Input PSD Max.	1
Output PSD Min.	0
Output PSD Max.	1
Input RMS Min.	0
Input RMS Max.	1
Output RMS Min.	0
Output RMS Max.	1
Noise Distribution	White
Multiply Error Controls (affectin...	1

Summary:
This analysis will use **0** modifier files.

Figure 122 Power Spectral Density Data Form

Selecting the Input Noise Distribution

The default setting for the input noise distribution is white noise with a 1-sigma amplitude of unity. To specify a different 1-sigma amplitude or a different noise distribution, select “Show/Edit Input Noise Distribution Parameters.” A PSD “Input Noise Distribution Parameters Form” will appear on your screen. For a detailed explanation of this form see "The PSD Input Noise Distribution Parameters Form".

Setting the Frequency Units Option

“Frequency Units:” is used to specify the frequency units associated with the input power spectral density. This is especially important when any of the user-supplied input noise distributions has been chosen. To select a value, point to the desired setting and click the left mouse button.

Setting the Plot Data Option

“Plot Data:” tells Easy5 whether or not you wish to plot the results from the Power Spectral Density analysis. Point to the desired setting and click the left mouse button.

Setting the Plot Type and Plot Signal Options

“Plot Type:” and “Plot Signal:” tell the Easy5 plotter which power spectral density results you wish to see displayed first. “Plot Signal:” selects either input quantities or output quantities or both input quantities and output quantities for display. “Plot Type:” is used to tell whether the power spectral density or the accumulated RMS or both the power spectral density and the accumulated RMS will be displayed. Point to the desired settings and click the left mouse button. The default settings have the input and output power spectral density displayed on the same plot.

Setting the Plot Scales Option

“Plot Scales:” is used to specify manual scales or automatic scales to the plotter. Point to the desired setting and click the left mouse button. When automatic is selected Easy5 automatically finds the maximum and minimum of all quantities to be plotted. For the frequency axis, the minimum frequency is 1/10 of the lowest non-zero frequency in the model, and the maximum frequency is 10 times the highest frequency in the model.

Selecting manual scales expands the Power Spectral Density Analysis Data Form to include maximum and minimum settings for all variables that can be plotted. Specify the scales by entering the appropriate values in the data fields. If the maximum and minimum values in a given data field are the same, Easy5 uses automatic scaling for that variable. This is a way to turn off a previously set manual scale.

The PSD Input Noise Distribution Parameters Form

Right-click to select the desired distribution and click the left mouse button. All distributions have an “Intensity” data field, which corresponds to the 1-sigma amplitude of the distribution; the default value is 1.00. Depending on the distribution selected, additional options and data fields will appear on the form. These additional options and fields are defined in the following sections.

Defining the Air Speed Parameter

The “Air Speed Parameter” is the name of the quantity in your model that corresponds to true airspeed. You enter the name of this quantity in the data field following “Air Speed Parameter:” on the data form. To enter a parameter name, point to the data field and enter a name. You can also use the “Pick” option to enter the airspeed parameter name.

Specifying the Scale Length Parameter

“Scale Length:” is used to specify the spatial frequency of the selected noise distribution. Information about its use is discussed in “Power Spectral Density Analysis Method”. To enter a value, point to the “Scale Length:” data field and enter a number.

Selecting the Axis

For a description of how the noise distributions differ between axes, see "Power Spectral Density Analysis Method". To select an axis, point to the desired axis, u, v, or w, and click the left mouse button.

Specifying the Peak Amplitude Parameter

"Gust Amplitude:" is used in the discrete (1 - cos) noise distribution to specify the peak amplitude of the (1 - cos) function which occurs at the "Gust length" value specified in The PSD Input Noise Distribution Parameters Form. For an indication of its use, see the noise distribution equations in "Power Spectral Density Analysis Method". To enter a value, point to the "Gust Amplitude:" data field and enter a number.

Specifying the Gust Length Parameter

"Gust Length:" Is used in the discrete (1 - cos) noise distribution to specify the spatial distance to the gust of the (1 - cos) function. For an indication of its use, see the noise distribution equations in "Power Spectral Density Analysis Method". To enter a value, point to the "Gust Length" data field and enter a number.

Using The User Subroutine Option

If you select "User Subroutine" you must provide a subroutine that calculates the input power spectral density from a given frequency input.

The calling sequence for this subroutine is:

```
SUBROUTINE EZPSDU (FRQRPS, FRQHZ, PSDVAL)
```

where:

FRQRPS = evaluation frequency in radians/second (input)

FRQHZ = evaluation of frequency in Hertz (input)

PSDVAL = value of Power Spectral Density at evaluation frequency (output)

The evaluation frequency is provided to routine `EZPSDU` in both units of radians/second and hertz for convenience; use the one which is most suited for your calculation.

Note:	You must specify `FRQRPS`, `FRQHZ`, and `PSDVAL` as REAL*8 variables (double precision values).

Using the User Table Option

If you select "User Table", you must add a table named `PSDINP` in a Fortran component. If there are no Fortran components in the model, you must add one which specifies `PSDINP`. This component does not need to have any other inputs or outputs, but it must have at least one line of code in it, for example, a Fortran comment line.You specify data for the table in the normal manner using frequency as the independent variable.

Power Spectral Density Analysis Outputs

Power Spectral Density analysis outputs can be in plotted and/or printed format. These output formats are described in the following sections.

Power Spectral Density Plots

If power spectral density plots were generated during your analysis, you can look at them by selecting **Plot Analysis Results** (Ctrl+P) from the **Analysis** menu.

The Power Spectral Density Analysis Output Listing

The Power Spectral Density analysis output listing may be examined by selecting **Display Analysis Output Listing** (Ctrl+L) from the **Analysis** menu. Once this file has been opened, search for the string "POWER SPECTRAL DENSITY ANALYSIS" to point you to the start of the Power Spectral Density analysis output printout.

You will see the linearization data and the operating point information for the transfer function associated with the selected input power spectral density. This is followed by the zeros, the leading coefficient, and the poles of the transfer function. Next is a listing of the input and output power spectral densities and their associated accumulated RMS values, as a function of frequency - both Hertz and radians/second. The selection specified by the "FREQUENCY UNITS" option will be printed in the first column.

Power Spectral Density Analysis Method

Power Spectral Density analysis is based on passing an input power spectral density through a transfer function to yield the output power spectral density. The related equation is:

$$P_{out}(\omega) = |H(j\omega)|^2 P_{in}(\omega)$$

where:

ω	is the frequency in radians/second
P_{in}	is the input power spectral density
H(jw)	is the transfer function
P_{out}	is the output power spectral density

The accumulated RMS for either the input or the output power spectral density is also printed. The actual mean square value of the given PSD is:

$$V^2 = \frac{1}{2\pi}\left(\int_0^{\infty} P(\omega)\,d\omega\right)$$

The accumulated RMS curve is the square root of this quantity; the upper limit is the maximum frequency and not infinity. As long as the integration range is “infinite”, the final value of the accumulated RMS will be close to the actual RMS value of either the input or the output.

For more information on Power Spectral Density analysis, see Reference 20, at the end of this manual.

Input Noise Distributions

The gust distributions are either Von Karman, Dryden, or Discrete. The von Karman distribution is defined by:

$$\Phi(\Omega) = \frac{L\sigma^2}{\pi}\frac{1 + (8/3)(1.339L\Omega)^2}{[1 + (1.339L\Omega)^2]^{11/6}} \qquad \textit{v,w-axis}$$

$$\Phi(\Omega) = \frac{2L\sigma^2}{\pi}\frac{1}{[1 + (1.339L\Omega)^2]^{5/6}} \qquad \textit{u-axis}$$

The Dryden distribution is defined by:

$$\Phi(\Omega) = \frac{2L\sigma^2}{\pi}\frac{1}{1+(L\Omega)^2} \qquad \textit{u-axis}$$

$$\Phi(\Omega) = \frac{L\sigma^2}{\pi}\frac{1+3(L\Omega)^2}{[1+(L\Omega)^2]^2} \qquad \textit{v,w-axis}$$

where:

L	is the gust scale length - units, length
σ	is 1-sigma amplitude - units, length/time
Ω	is the spatial frequency - rad/length

The spatial power spectral densities are related to temporal power spectral densities through the airspeed. The conversion equation is:

$$\Phi(\omega) = \frac{1}{V}\phi(\Omega = \omega/V)$$

where:

V	is the true airspeed - units, length/time
ϕ	is the spatial power spectral density
Ω	is the spatial frequency - rad/length
Φ	is the temporal power spectral density
ω	is the temporal frequency - rad/second

The 1-sigma amplitudes of these distributions are related to the accumulated RMS calculation as follows. The 1-sigma amplitudes are defined as:

$$\sigma^2 = \left(\int_0^\infty \Phi(\omega)d\omega\right)$$

The equation that relates the final value of the accumulated RMS to the 1-sigma amplitude is:

$$\bar{V} = \frac{\sigma}{(2\pi)^{0.5}}$$

Note:	Easy5 uses trapezoidal integration to calculate the accumulated RMS values. If the range of integration is not wide enough, then the integral will be in error. The range of integration must span the transfer function frequency range (Easy5 will ensure this) and the power-spectral-density frequency range. Therefore, you may wish to use manual scales during this analysis.

The Discrete (1 - cos) distribution is defined by:

$$\Phi(\omega) = \left(\frac{\pi V}{d}\right)^4 \left(\frac{A}{2}\right)^2 \left[\frac{2\left(1 - cos\frac{2\omega d}{V}\right)}{\omega^2\left(\omega^2 - \left(\frac{\pi V}{d}\right)^2\right)^2}\right]$$

where:

d	is the distance to gust peak, units - length
V	is the true airspeed, units - length/time
A	is gust amplitude, units - length/time

This gust is deterministic, but its parameters can be related to the statistical form of the preceding distributions.

Print Options

See also: "Documenting and Printing the Model"

Windows Print Options

Print options on the Windows version of Easy5 are managed using the standard Windows print features. When a print is requested, the Print dialog opens. This dialog is self-explanatory. Selecting the [Properties] button opens a dialog that allows you to setup additional print options. The print settings last applied are saved and used for subsequent print jobs.

Linux Print Options

Print options for Linux platforms are managed using a native Linux **Print** dialog. When a print is requested, a standard Print dialog opens. Again, print options are generally self-explanatory, and offer the ability to print the schematic page in a variety of layouts, different printers, and to printed to a file (using Postscript).

Reserved Words

Easy5 contains a set of reserved words that you must never use in your User Code or Library components as inputs and/or outputs. In addition, these Fortran variable names should never be set in your code. (Exceptions to this are ISTOP and PFLAG, which can be set as described below).

Easy5 Reserved Words

C	ICLOCK	IWRITE	TAU0
CCLOCK	IDELAY	IXOC	TIME
CIO	IDIAG	KCLOCK	TINC
CKLOCK	IDUMMY	KMOD	TMAX
CP	IERR	LOKSIM	TSTEP
CPOITR	IEZ___*	LOKSS	UPRINT
CPUSEC	IFINAL	MJITER	VARSET
CSIMUL	IMOD	MNITER	VS
CSIMUR	INCALL	NRC	XDOT
CV	INDP	NRCMAX	Z
CX	INST	NS	ZERO
CXDOT	INDP	NU	
D	INX	ONE	
DPR	IOC1	PFLAG	
EQMO	IOC2	PI	
ERMESS	IOC3	R	
EZ____*	IOC4	RENAME	
FO____*	IPRINT	RPD	
GRE	IREAD	SDOT	
GRM	ISTOP	STOP	
IAUX	ITINC	TABLE	
ICCALC	IWARN	TAU___*	

* Any input/output name beginning with these letters are reserved words.

Caution:	If you declare any of these reserved words as inputs or outputs to User Code or Library components, Easy5 detects them and issues a fatal error message to that effect. However, Easy5 cannot detect if you simply set (use it to the left of an equal sign) one of the reserved words, so exercise care when writing your code.

General Reserved Words

Several of these variables can be used in your User Code and Library components to monitor an analysis. In addition, two of the words, PFLAG and ISTOP, can actually be set to control the Simulation analysis. Several Easy5 reserved words that you are likely to find useful are described in the following tables:

Reserved Word	Description
CPUSEC	A variable set to the number of computer CPU seconds that have elapsed.
ICCALC	Flag used to calculate the initial conditions. ICCALC=1 only once during the calculation of initial conditions; ICCALC=0 at all other times.

INCALL	A flag that indicates the initial call to EQMO for each analysis. The value of INCALL is set by the Analysis program as follows:
	
	INCALL = 2 for the first call to EQMO during the first analysis of a run, that is, the first analysis defined in the Analysis Description file.
	
	INCALL = 1 for the first call to EQMO during the second and all subsequent analyses.
	
	INCALL = 0 for the second and all subsequent calls to EQMO during all analyses.
	
	INCALL can be used to cause initialization calculations to be performed only on the first call to the system model for each analysis. By testing on INCALL=2, default parameters can be set only at the beginning of an analysis run. The following code shows how this feature is used in a Fortran component:
	
	`      IF (INCALL.NE.2) GO TO 20`
	`      .....`
	`C     (DEFAULT CALCULATIONS AT BEGINNING OF RUN)`
	`      .....`
	`20    IF (INCALL.EQ.0) GO TO 100`
	`      .....`
	`C     (INITIAL CALCULATIONS BEFORE EACH ANALYSIS)`
	`100    CONTINUE`

IDELAY	A flag used in discrete models. IDELAY is set to 1 during model calls at sample periods. For all other calls to the model, IDELAY is set to 0.

Reserved Word	Description

IERR	A flag used primarily for steady-state analysis validity checking. IERR is set to 1 after a steady-state solution has been found, but prior to the "official" model call. In this way, you can perform validity checks to see if the solution point is a reasonable one. For example, the following code is used to check if system pressure makes sense at the equilibrium point:
	`IF (INST.EQ.31 .AND. IERR.EQ.1 .AND. PRESSUR.LT.ZERO)`
	`+  WRITE (IWRITE,'(//5X,''INVALID-PRESSURE NEGATIVE''//)')`
	This flag is also used by the Runge-Kutta variable-step integration method to indicate calls to the model where the error controls are satisfied.

INST	An analysis indicator whose value depends on the type of analysis being executed. All the values for INST are listed below: .

Analysis Type	INST Flag Value
CALC XIC	61
LINEAR MODEL GENERATION (no INPUTS/OUTPUTS)	27
LINEAR MODEL GENERATION (with INPUTS/OUTPUTS)	63
PRINT	60
ROOT LOCUS	32
FUNCTION SCAN (one indep. var.)	13
FUNCTION SCAN (two indep. var.)	14
SIMULATION	26
STABILITY MARGINS	29
STEADY- STATE	31
TRANSFER FUNCTION	30
EIGENVALUE SENSITIVITY	28
POWER SPECTRAL DENSITY	106

INDP	Simulation termination mode flag; see "Termination Commands".
ISTOP	A special reserved word that you can set to 1 or 2 to stop a simulation before the "STOP TIME" value has been reached. See "Stop and Exit Flags" for details.
ITINC	A flag used during a simulation. ITINC is set to 1 at every "TIME INCREMENT", and set to 0 at all other times. ITINC is typically used to ensure that data is output by one of your components at a valid reporting interval.

Reserved Word	Description
IEZSWS	A flag used to signal whenever a switch state event search is taking place (=1) or (=0 otherwise).
IWRITE	A COMMON variable that is set equal to the Fortran unit number which corresponds to the Analysis Output Listing file. You can "write" to this unit number in your Fortran components and the output will be sent to this listing file. Theoretically, you could change the value of IWRITE during a simulation, and all output would be redirected to another file, assuming it has been properly opened.

LINEAR	A logical flag that indicates whether an analysis is nonlinear or linear. In some circumstances, you may wish to provide alternate code depending on the value of this flag. The table for the reserved word INST gives the values of LINEAR for all analysis types. For example, consider a component modeling a deadzone. For linearized analyses it would be convenient to ignore this nonlinearity. This could be done with the following code:
	`C - - - IN NULLZONE`
	`  IF (.NOT. LINEAR) THEN`
	`      OUTPUT = 0.0`
	`  ELSE`
	`      OUTPUT = GAIN * INPUT`
	`  ENDIF`
	Several General Purpose components utilize this flag to ignore certain non-linearities for linearized analyses.

PFLAG	A reserved word that can be set by your code to control printing and plotting during a simulation. If you set PFLAG to 1, any secondary print and plot options you have specified will become active. If PFLAG is set to 0, printing and plotting will return to the primary mode. This variable is also set by Easy5 according to the information you have specified with the "FROM TIME" and "TO TIME" data fields.
TAUMAX	A variable that is set equal to the least common multiple of all sample periods.
TAU0	A variable equal to the smallest or lowest common divisor of sample periods.
TIME	TIME is always set to the current time of the analysis. In the case of a Simulation analysis, TIME changes as the simulation progresses. For all the other analyses, TIME is equal to the "TIME" value you enter in the respective data form.
TINC	For a Simulation analysis, this variable is set equal to the "TIME INCREMENT" value in the simulation data form. For all other analyses, TINC is equal to 0.1.
TMAX	For a Simulation analysis, this variable is set equal to the "STOP TIME" value in the simulation data form. For all other analyses, TMAX is given a default value of 1.0.
TSTEP	TSTEP is set equal to the current integrator step size. For the fixed-step integrator, TSTEP is always equal to TINC. For variable-step integrators, TSTEP changes depending upon the integrator's step-size selection algorithm.

Physical and Mathematical Constants Reserved Words

The following Easy5 reserved words are available to use in your model as constants. They represent floating point numbers or values for physical and mathematical constants used frequently by Easy5 users. They have up to 14-point accuracy.

Reserved Word	Description
ONE	The value of one. Also, this is often used to multiply another floating-point number located in an argument list. This helps to minimize "conversions" that may be necessary when transporting code between different machines. For example, the following code on a 64-bit machine would work well (because X is typed REAL): X = MAX(X,11.57) However, this code would not be the same if run on a 32-bit machine. In this case, you would have to add D0 to the number because the variable X is typed REAL*8, not REAL. To avoid this, simply multiply the value 11.57 by ONE and the resident FORTRAN compiler will convert it to the correct type automatically. The machine-independent form of this code would be: X = MAX(X,11.57*ONE) Thus, the suggested form for using floating-point literals in an argument list is as follows: literal*ONE where literal is a literal floating-point number.
ZERO	A constant in COMMON set equal to a double precision real value of 0.0.
PI	A constant in COMMON set equal to a double precision real value for Pi (=3.14159265358979).
RPD	Conversion constant: radians/degree (0.017453293 radians/degrees).
DPR	Conversion constant: degrees/radian (57.29577951 degrees/radian).
GRM	Gravity constant: SI units (9.80621 m/sec2).
GRE	Gravity constant: English units (32.1725 ft/sec2).
EZXLG	A very large number: 1.0E+36.
EZXSM	A very small number: 1.0E-14.
FPZ	Floating point precision of your machine. This is usually set to a value of 1.0E-14.

Root Locus Analysis

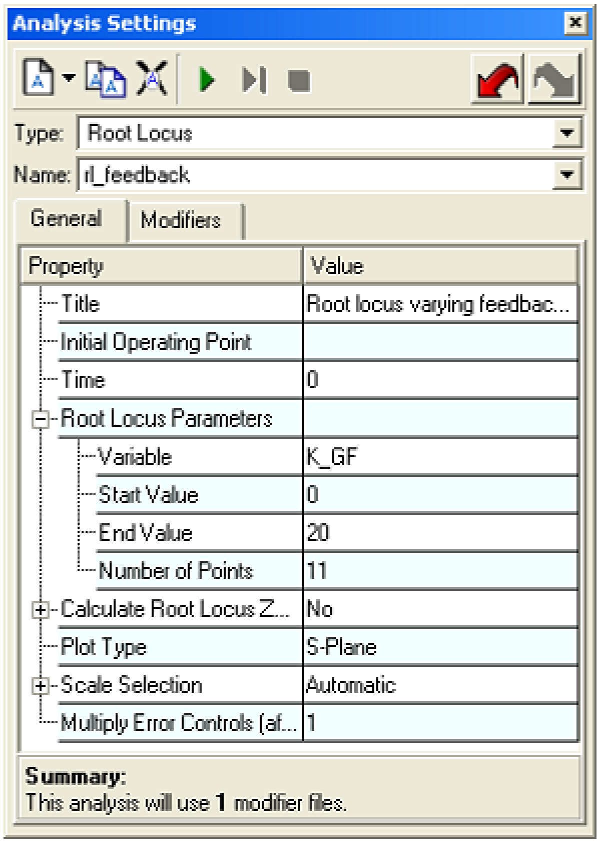
The Root Locus analysis calculates the loci of the system eigenvalues as a function of a specified parameter. The Easy5 program allows a Root Locus analysis to be performed at any operating point value, as well as a function of any system parameter. This analysis can be performed on both continuous and sampled data systems and, therefore, printed and/or plotted results are given in either the s-plane or the z-plane.

Before execution of this analysis, you must specify which "quantity" of your model should be varied, through what range, and with what resolution. The parameter to be changed, which is called the root locus parameter, may be any system parameter or frozen state and is entered via the root locus data form. Calculation of the zeros of the root locus may also be requested.

Setting up a Root Locus Analysis

The root locus analysis is setup and executed via the Root Locus Analysis Data form. This analysis form is accessed by first selecting **Analysis > Linear and Root Locus...** from the main menu bar.

Figure 123 shows the root locus data form with all data fields displayed. To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to "Analysis Data Form"



Type: Root Locus

Name: rl_feedback

Property	Value
Title	Root locus varying feedbac...
Initial Operating Point	
Time	0
Root Locus Parameters	
Variable	K_GF
Start Value	0
End Value	20
Number of Points	11
Calculate Root Locus Z...	No
Plot Type	S-Plane
Scale Selection	Automatic
Multiply Error Controls (af...	1

Summary:
This analysis will use **1** modifier files.

Figure 123 Root Locus Analysis Data Form

Specifying the Root Locus Parameter

The root locus parameter is the name of the "quantity" that will be varied from the "Start Value" to the "End Value" (described below) to produce the root locus. The root locus parameter can be either a parameter name or a state name. A state may be used as a root locus parameter only if the specified state variable has been frozen. To enter a root locus parameter, select the "Parameter" data input field (the cursor will blink) and enter an appropriate name. You can also use the "Pick" option described in the "Model Explorer Window", to define the root locus parameter.

Setting the Starting Value

"Start Value" is the initial value the defined root locus parameter will be given for the root locus analysis. Select the respective data field and enter a starting value.

Setting the Ending Value

“End Value” is the final value the root locus parameter field will be given for the root locus analysis. Select the respective data field and enter an ending value.

Setting the Number of Points

The “# of Points” value is used to set the number of points you want on the root locus plot. Select the respective data field and enter a value. This value minus one equals the number of steps taken from “Start Value” to “End Value.” For example, if you wish to vary a gain from 0 to 10 in increments of 1 (i.e. 0, 1, 2,... 9, 10), then set Start Value=0, End Value=10, # of Points=11.

Specifying Linear or Geometric Progression

By default, if the value for **# of Points** is positive, the step size between root locus evaluations is calculated as a **linear progression** of steps as indicated by:

```
    step size = (End Value - Start Value)/(# of Points)
```

Alternatively, if the # of Points value is negative, the step size is calculated as a geometric progression of steps, where each point is calculated as a factor of the previous step. This factor is calculated by:

```
    factor = (End Value/Start Value)^(1/(# of Points -1))
```

An obvious requirement for specifying a geometric progression is that *the value for* **Start Value** *must be a nonzero number having the same sign as the* **End Value**. Otherwise, the point selection will revert to a linear one.

Most often, for a geometric progression, the hardest quantity to calculate is the value for **# of Points** (because you usually know what you want the value of the factor (or base) to be).

As a convenience, solving for **# of Points** yields an equation of the form:

# of Points_value = (log(End Value/Start Value)/log(factor)) + 1

For example, a root locus could be generated having a geometric progression with a factor of 10; this is analogous to a logarithmic progression. If the **Start Value** is set to `1` and you wish to go up by factors of 10 to 1000, then set **End Value** to `1000`, and **# of Points** to `-4` .

Calculating Root Locus Zeros

If the zeros of the root locus are to be calculated, select "Yes" following "Calculate Root Locus Zeros:" on the root locus data form. When you do this, two new fields will appear on your form. These are described in the following paragraphs.

The root locus zeros will be the zeros of the transfer function between the points specified with “RL Input” and “RL Output.” Using the textbook definition of root locus, the branches of the locus terminate on the zeros if the root locus gain is set to an arbitrarily high value. This occurs only if the system is linear with respect to the root locus parameter, and if the root locus parameter appears as a separable gain in the feed forward loop. Easy5 does not impose either of these restrictions on root locus calculations. However, for the zeros to have their traditional textbook meaning, the root locus “Parameter” you define should be a simple gain parameter such that RL Input = Parameter * RL Output.

To specify the root locus input, select the respective data field and enter an appropriate input value. You can also use the “Pick” option to define a root locus input.

To specify a root locus output, select the respective data field, enter an appropriate output value, and press the RETURN key. You can also use the “Pick” option to define a root locus output.

Requesting Root Locus Plots

If you want a root locus plot, select “Yes” following “Plot Results:” and fill in the appropriate data input fields as they appear.

Specifying Root Locus Output

The root locus can be plotted in either the s-plane or the z-plane. This is defined by selecting the “Coordinates” options in the data form. The s-plane plot is the default option.

The “Scales” option in the data form allows you to specify either automatic or manual scales. Automatic scales is the default option, and is the recommended scale option. If manual scales is chosen, then you must specify the maximum and minimum values for both the real and imaginary axes.

Root Locus Analysis Method

The root loci are calculated by forming the Jacobian or stability matrix of the system for each value of a root locus parameter. In other words, your model is re-linearized at each root locus point to accurately track non linearities in your model. The eigenvalues of each stability matrix are calculated to give the root loci.

This approach is a nonlinear generalization of the linear technique developed by Evans. For nonlinear systems, this analysis provides an accurate picture of the loci of the system eigenvalues when a system parameter is varied.

Since this technique does not utilize a single, fixed linear model, the zeros corresponding to the selected root locus parameter cannot be calculated directly. They can only be approximated by setting the root locus parameter to a very large value or by requesting calculation of root locus zeros, described later in this section. See “Root Locus Analysis” for additional information.

Simplifications Made During Linearization

Simplifications are made to improve the efficiency of the Root Locus Analysis. To begin with, a linearity test, which is usually performed for calculations of the stability matrix and requires that the stability matrix be calculated twice, is omitted.

Another simplification relates to the columns of the stability matrix. Each column of the stability matrix requires one call to your model equations. Specifically, static elements are identified by calculating the complete stability matrix for the nominal system and again for the first value of the root locus parameter. The elements of these stability matrices are compared to determine which columns are affected by changes in the root locus parameter. Subsequent stability matrix calculations only reevaluate those columns containing elements which have been identified as functions of the root locus parameter.

For sampled data systems, the selection of the **RL PARAMETER** can profoundly affect the execution speed of the analysis. The most time consuming part of the root locus analysis is the calculation of the e^{At} matrix (where A is the continuous system stability matrix). It is repeated for every value of the **RL PARAMETER**, if the stability matrix is found to be affected by the **RL PARAMETER**. Thus, if the **RL PARAMETER** does not affect the continuous system stability matrix, this calculation is skipped resulting in much faster executions.

Calculation of Root Locus Zeros

If the zeros of the Root Locus analysis are to be calculated, you must specify root locus input and output parameters. The root locus zeros are the zeros of the transfer function between the points specified. The sections that follow detail the difference between the "textbook" definition of root locus zero calculations, and Easy5's approach.

Under the textbook definition of root locus, the branches of the locus will terminate on the zeros if the root locus parameter is set to an infinitely high value. This occurs only if the system is linear with respect to the root locus parameter and if the root locus parameter appears as a separable gain in the feed forward loop. If these criteria are not satisfied, the root locus zeros will not be terminators of the loci.

The Easy5 Approach to Root Locus Analysis

Easy5 allows a root locus to be performed on any constant of your system. This will always yield the system poles at each value of the root locus parameters. To obtain zeros, you must also specify a root locus input and output. For the zeros to have their traditional meaning (that is they are the terminators of locus when the root locus parameter is set to an infinitely high value), the root locus parameter must be a simple gain connection such that:

```
RL INPUT = RL PARAMETER * RL OUTPUT
```

The Root Locus analysis verifies that this relationship exists; a warning will be printed if it is not satisfied. This warning tells you that, while the Easy5 Root Locus analysis will give you an excellent picture of the behavior of your system, your intuitive expectations concerning the shape of the loci will not be met.

Note: Exercise care in verifying this relationship between input and output. The first step the root locus zero analysis takes is to set the root locus parameter to a value of zero, thus breaking the loop. Therefore, if the root locus parameter, input and output are not as shown in Figure 124, the resulting transfer function will have no connection between input and output and Easy5 will print a warning message to this effect in the Analysis Output Listing file.

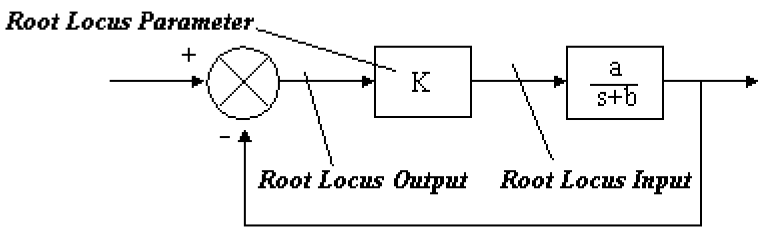


Figure 124 Correct Relationship between Root Locus Parameter, Input and Output for Calculating Root Locus Zeros

Schematic Manipulation

The Easy5 model schematic can be manipulated in a variety of ways. You can move components, either individually or in groups. You can use Easy5 to locate a single component in a schematic. You can also manipulate the schematic window by "zooming" in or out over a particular location, or by "panning" to any part of the schematic you want to view. In addition, Easy5 permits you to reduce the size of the schematic view so that the entire block diagram is displayed within your schematic window. Then you can return the schematic view to its previous size.

These capabilities are described in the following sections.

Moving Components

In the course of constructing an Easy5 block diagram, you will often want to reposition a component. Components can be moved around a schematic using the following methods:

1. Moving components individually.
2. Moving components in groups.

Easy5 uses a method called "drag and drop" to move individual components. This method allows the user to select a component and "drag" the component anywhere in the schematic, including up and down hierarchical layers, and then "drop" the component at the desired location.

Moving Individual Components

To move an individual component, perform the following steps:

1. Left click on the component to activate it.
2. Drag the component across the schematic.
3. Left click again to "drop" the component.

Moving Groups of Components

1. Draw a selection box, first define a corner of the box by pointing to the desired location and HOLD-L. Next, drag your mouse diagonally away from the location where you defined your corner until everything you wish to be highlighted is inside the box, then release the mouse button.
2. Either move the selected group (by dragging it), or select the *Ctrl+M* keyboard shortcut. The latter is useful, if you want to move the group inside a submodel.
3. Move the selection box to where you want to position the components on the schematic and select that location by clicking the left mouse button.

   Selecting a location for the captured components causes Easy5 to redraw your schematic. You can select as many or as few components in a group as you desire.

Moving the Window

You look at the schematic pad through the schematic window. Easy5 permits you to move this window over the pad in order to view different parts of your model. This technique is called “panning.” You can also position the schematic window farther away from or closer to the schematic pad, in effect “zooming” in or out on your schematic. The following section describes the methods of moving the schematic window.

You can “pan” (move) the schematic window over any part of the Easy5 block diagram by using the scroll bars on the right hand side and bottom of the window. To move your schematic window to the left or right, use the scroll bar at the bottom of your window. In addition, use the **Shift+Hold-Left** keyboard/mouse navigation shortcut to pan your model. When in “pan” mode, you will notice a large crossed set of double arrows that you can move around the schematic.

The “zooming” feature permits you to expand the schematic view (zoom out) or contract the schematic view (zoom in).

Zoom in on your schematic to increase the level of detail by selecting **Zoom In** icon from the Edit toolbar (or Ctrl++). You will see the view of your schematic contract, revealing less of the overall schematic, but more (increasing the size) of a particular point.

You can continue zooming in by performing the step described previously until the component (or components) reaches the maximum size possible. Then you can zoom out, reducing the apparent size of the schematic until it eventually returns to the original view or until it becomes the smallest size allowed by the software.

Zooming out is performed by selecting **Zoom Out** icon from the Edit toolbar (or use the Ctrl+- keyboard shortcut). You will see the view of your schematic expand, revealing more of the overall schematic, but less (decreasing the size) of a particular point.

Viewing the Entire Schematic

To have Easy5 fill the window with the entire schematic, select **View Entire** from the **View** menu, the “View Entire” Edit toolbar icon, or using the Ctrl+E keyboard shortcut. Easy5 places the schematic window over the schematic pad to give you the best overall view of your entire block diagram.

Returning to the Previous Schematic View

If, after working in the View Entire schematic mode, you decide to return the view to the detail level that preceded the selection of that mode, Easy5 has another short-cut feature that performs this in one step. This feature is called **Refresh View**, and is selected from the **View** menu, or by pressing the Space bar.

Locating Components

Your block diagram can become so complex and include so many components that it can be difficult for you to quickly locate a specific component. However, if you know its exact name (the ID you gave it when you added it to the schematic), Easy5 can find it for you.

To direct Easy5 to locate a specific component in your schematic, do the following:

1. Select **Explore Model** from the **View** menu, rom the Edit toolbar, or using the Ctrl+F keyboard shortcut.

   A (dockable) Model Explorer window that lists all the components along with their inputs and outputs in your model will be displayed.
2. Double click on any component that you want to open and that component is activated and it's properties are displayed.

Note: You can also enter the first character of the component name to help navigate the component list.

Simulation Analysis

See also: "Analysis Data Form"

"Integration Methods"

"Interactive Simulation"

"Root Locus Analysis"

"Simulation Monitor"

User Guide, Chapter 4 - Interactive Simulation

Simulation is the process of numerical integration of your model's equation set through time. It is used to generate time history plots of system variables to evaluate transient or time domain behavior. While simulation offers you the most accurate view of your model, it also exercises all the non linearities in your model, and is much more time consuming than the linearized analyses.

Several numerical integration techniques are available with Easy5. It is necessary to provide several methods, as no single method is appropriate for the wide range of differential equation characteristics that are encountered in dynamic models. The choice of the “best” integration method depends on a number of important considerations.

User specified requirements for accuracy, resolution, and duration of transients, and model characteristics such as natural frequencies, discontinuities, external disturbances, and sampling effects all must be considered. See **Appendix B**, for a further discussion of selecting integration techniques.

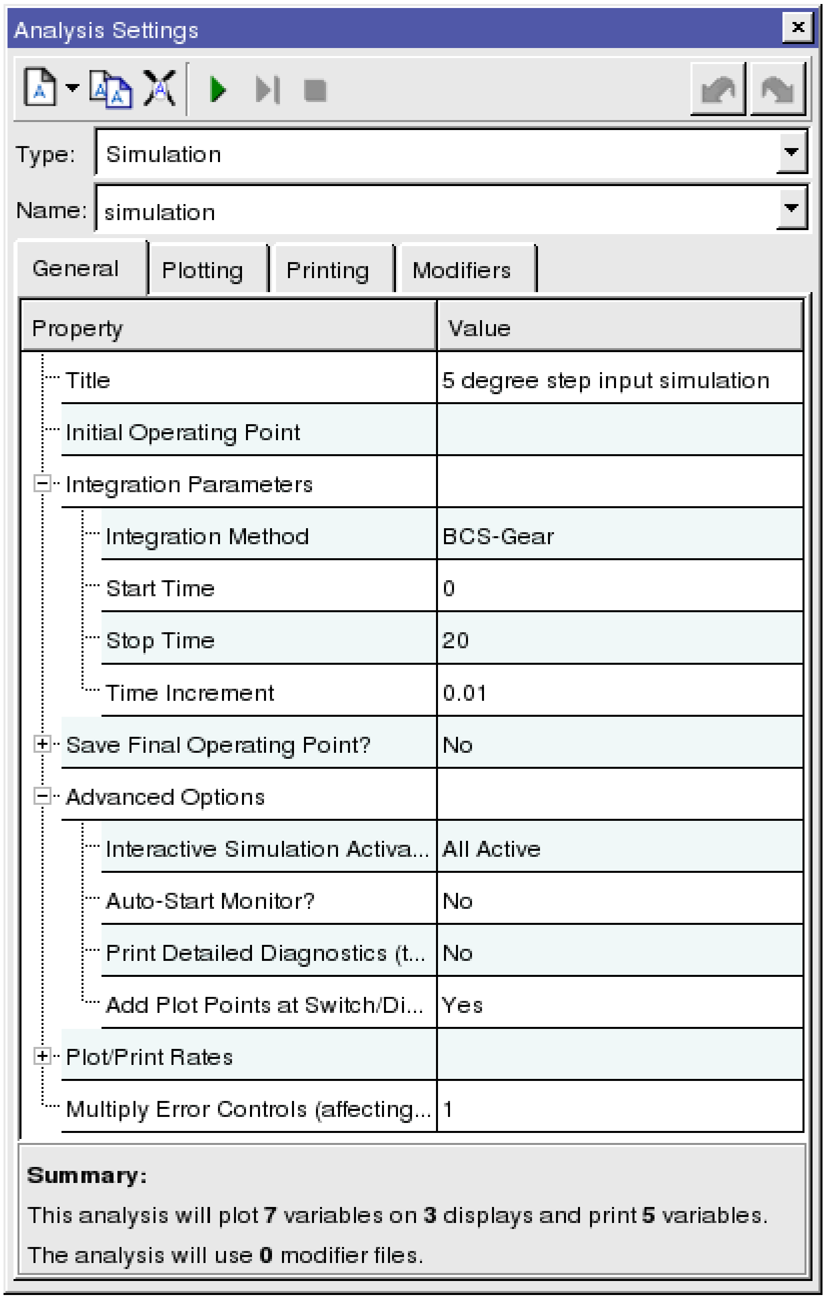
Setting up a Simulation

All data for setting up a simulation is entered in the simulation data form, which is shown in Figure 125, which shows all the simulation options and associated data fields.

The sections that follow describe how to fill in the form.

Specifying Initial Operating Point

An operation point that has been previously generated may be restored to define the simulation’s operating point. This is done by selecting the “Initial Operating Point” input field. A menu will appear displaying all available operating point files. These are files that were either created via the **Options > Save Operating Point** menu, or from an analysis defined by the “Save Final Operating Point” function. Further information on saving and restoring operating points is given in "Saving a Final Simulation Operating Point", and "Parameters - Defining Input Values".



Analysis Settings

Type: Simulation

Name: simulation

Property	Value
Title	5 degree step input simulation
Initial Operating Point	
Integration Parameters	
Integration Method	BCS-Gear
Start Time	0
Stop Time	20
Time Increment	0.01
Save Final Operating Point?	No
Advanced Options	
Interactive Simulation Activa...	All Active
Auto-Start Monitor?	No
Print Detailed Diagnostics (t...	No
Add Plot Points at Switch/Di...	Yes
Plot/Print Rates	
Multiply Error Controls (affecting...	1

Summary:

This analysis will plot **7** variables on **3** displays and print **5** variables.

The analysis will use **0** modifier files.

Figure 125 Simulation Analysis Data Form

Specifying Simulation Times

You can use the simulation data form to identify the time related data for your simulation analysis.You do this by specifying the following three items:

1. The time at which the simulation should start; the "Start Time."
2. The time at which the simulation should stop; the "Stop Time."
3. Depending upon the type of integration method you are using, the integration time step and/or the data recording rate; the "Time Increment."

The "Start Time" field specifies the time at which the simulation will start. To change the "Start Time" from its default value of zero, select the "Start Time" data field and enter a new value.

"Stop Time" specifies the time at which the simulation will stop. The total number of simulated seconds that a given simulation will run is calculated by subtracting the "Start Time" from the "Stop Time." To define the "Stop Time", enter a value in the "Stop Time" data field.

Note: The simulation time specified above is not the actual clock time the simulation will run on your computer. The amount of real time the simulation will require is a function of the speed of your machine.

The "Time Increment" simulation value is used in three different ways depending upon the type of integrator you have chosen. First and foremost, "Time Increment" always defines how often simulation data will be made available for printing or plotting, for both variable and fixed step integrators. In the case of a 10.0 second simulation run, when "Time Increment" is 0.01, 1000 sets of simulation data would be gathered and made available for printing or plotting.

Secondly, if you are using a variable step integration method, the "Time Increment" value is used to calculate the initial time step that will be taken at the beginning of the simulation.

Finally, when a fixed step integration method has been selected, the "Time Increment" value defines the integration step size for the fixed step integration algorithm chosen, in addition to the data recording rate.

To enter a "Time Increment" value, select the "Time Increment" data field and enter a value.

Setting Up Simulation Options

There are some noteworthy options used to setup plot and output formats.

Add Plot Points at Switch/Discrete Events By default, Easy5 plots out data at fixed time intervals. This advanced option allows you to save plot data at switch and discrete events that occur outside the plot time interval. "Yes" is the default mode, which turns on this option, and provides precise plot data (with the ability to track "sharp corners"). For simulations with a large occurrences of switch and discrete events you may wish to select "No" to turn off this option to decrease the amount of plot data.

Auto-Start Monitor The simulation Monitor is a special tool used to monitor simulation results while the simulation is running. By default, this advanced setting is set to "No" to turn off this feature. "Yes" launches the monitor. For more information, refer to "Simulation Monitor".

Export Plot Data as CSV This plot option exports the plot data as an CSV (comma separated value) filenamed <*modelname*>.<*runid*>`.csv`. "No" is the default setting which turns off the feature. To turn on this feature, select "Yes".

Export Plot Data as XRF (ADAMS/PPT) This plot option exports the plot data to an ADAMS XRF (ADAMS results) file, named <*modelname*>.<*runid*>`.res`. This file can then be plotted using the ADAMS/PPT Plotter. "No" is the default setting which turns off the feature. To turn on this feature, select "Yes".

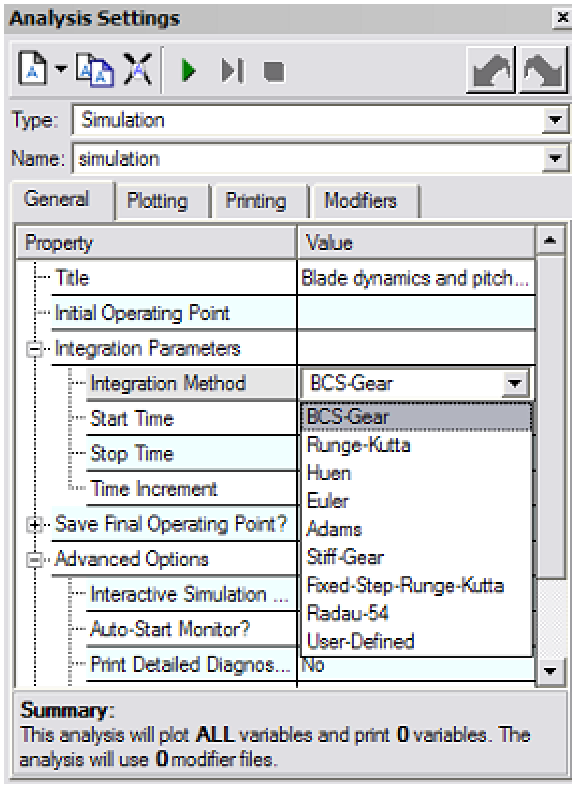
Print Detailed Diagnostics This advanced option provides a diagnostic tool to monitor the performance of the integrator. By default, the option is set to "No". To turn this on, select "Yes". The diagnostic data is written to the analysis file. Note that this tools generates a large amount of data and should be used with caution. For more information, refer to "Simulation Troubleshooting".

Multiply Error Control (affecting continuous states) This general option is used to globally change the error controls affecting continous states. This is used to affect the integration performance of your simulation. Please refer to "Setting Global Error Controls" for more details.

Setting up the Integration

To setup the integration, first, specify which integration method you wish to use. The "Int. Method" defines the integration mode or method that will be used during simulation.

The default method is BCS Gear. If you want to select a different method, CLICK-L- and hold, and a menu of choices will appear as shown in Figure 126. Point to the desired method and release the mouse button.


Analysis Settings

Type: Simulation

Name: simulation

General | Plotting | Printing | Modifiers

Property	Value
Title	Blade dynamics and pitch...
Initial Operating Point	
Integration Parameters	
Integration Method	BCS-Gear
Start Time	
Stop Time	
Time Increment	
Save Final Operating Point?	
Advanced Options	
Interactive Simulation ...	
Auto-Start Monitor?	
Print Detailed Diagnos...	No

Summary:
This analysis will plot **ALL** variables and print **0** variables. The analysis will use **0** modifier files.

Figure 126 Integration Methods Menu

You also have the option of setting the integration diagnostic flag to print out the integration steps. To do this select the "**Print Detailed Diagnostics** (**this run only**)?" = "**Yes**" advanced setting.

The "Yes" setting is only active during a single run, and always defaults back to "No". As a result, you must set this flag each time you submit the analysis.

This feature prints out information about the how the integrator is working, including such details as the current Time, step size, integration order, and number of calls to the Jacobian. This diagnostic feature is generally useful only when using a variable step integration method. The diagnostic messages are printed to the Analysis Output Listing file.

Note: The Print Detailed Diagnostic setting may generate a *large* amount of data. You should only use this when you need detailed information about how the integrator is working. It is very useful for isolating integration problems.

Setting up Simulation Plots

The values of up to 8000 variables can be plotted during a given simulation (4 variables per display, 2000 displays). Outputs can be plotted as a function of time, or as a function of another variable. You can plot outputs on individual sets of axes, or you can "over-plot" up to four outputs on one set of axes.

To plot data you do three things:

1. Specify that you want plots generated. See the **Plotting** tab.
2. Define how often you want the values plotted, i.e., the resolution of the plots. These settings are found in the **General** tab, under **Plot/Print Rates**.
3. Specify the output variables to plot.

In the Plotting tab, specify that you want plots by selecting **"Yes"** following **"Plot Results:"** in the simulation data form. Notice that when you select "Yes", other relevant settings appear in the data form.

You use these fields, to specify the specific model names to be plotted, and whether or not you want the simulation monitor to auto-start (see "Simulation Monitor" for information on using the Monitor Simulation feature).

"Primary Plot Rate" is an integer that Easy5 multiplies by the "Time Increment" value to determine how often it should record data. For example, if you wanted data plotted every 0.1 seconds, and "Time Increment" was 0.01 seconds, you would set "Primary Plot Rate" to 10.

Figure 127 shows the relationship between "Time Increment" and "Primary Plot Rate".

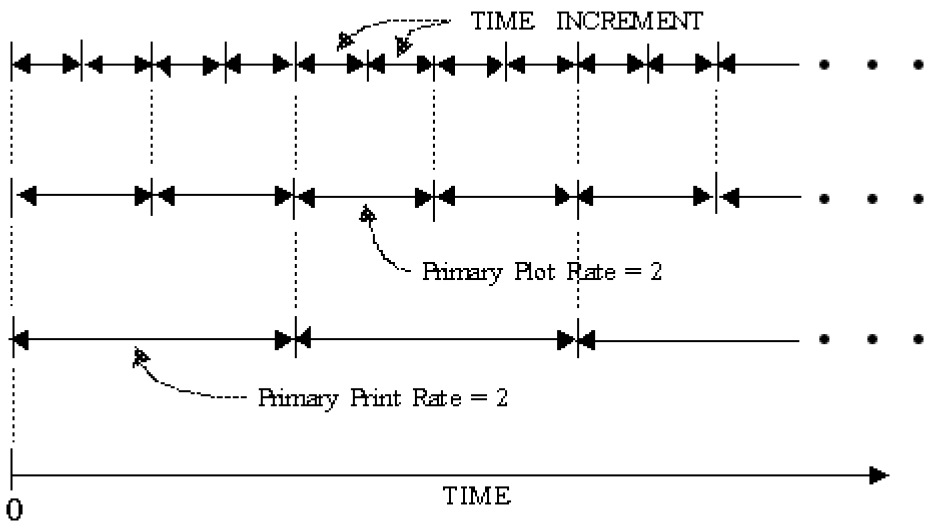
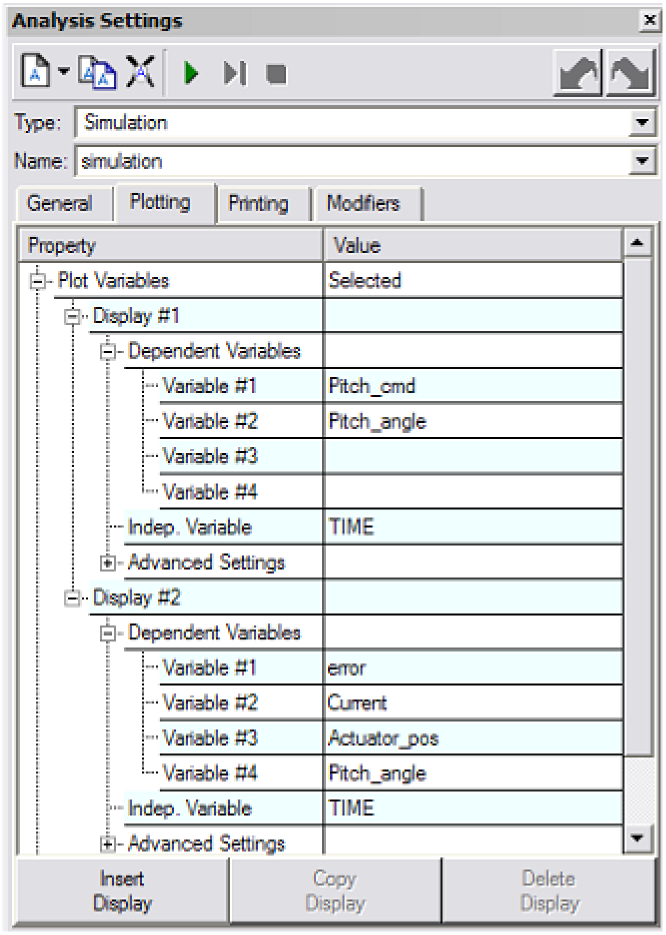


Figure 127 Relationship between Time Increment, Plot Rate, and Print Rate and TIME

Note: The “Time Increment” value specifies how often data is made available for printing and/or plotting, not how often it actually will be printed and/or plotted. Be sure you understand the difference between “Time Increment” and “Primary Plot Rate”.”

You specify plot variables by selecting the **Plot Variables** = **“Selected”** on the **Plotting** tab of the simulation form. The simulation plot specification form as shown in Figure 128 appears.

There are 2000 lines in the simulation plot specification form. Each line defines one of up to 2000 separate screen displays. (You look at displays one at a time with the Easy5 Plotter.) Up to four dependent variables can be plotted in each display. You can plot up to 8000 variables during one simulation run.


Analysis Settings

Type: Simulation

Name: simulation

General | Plotting | Printing | Modifiers

Property	Value
Plot Variables	Selected
Display #1	
Dependent Variables	
Variable #1	Pitch_cmd
Variable #2	Pitch_angle
Variable #3	
Variable #4	
Indep. Variable	TIME
Advanced Settings	
Display #2	
Dependent Variables	
Variable #1	error
Variable #2	Current
Variable #3	Actuator_pos
Variable #4	Pitch_angle
Indep. Variable	TIME
Advanced Settings	

Figure 128 Plot Specification Form

To input data into this form, you must first select the desired input field.

There are three methods to input data into the selected input field:

1. The name of any variable can be typed directly into the boxed input field. If you enter a name that does not exist in your model, an error message will be given.
2. Any name can be "picked" from the schematic block diagram (or Model Explorer window). Simply select a component in the schematic (or in the Model Explorer window) and a filtered list of available names will be displayed to select from. Please see "Model Explorer Window" for further information. See Figure 128 for an example of using selected plot variables.
3. Another method is to plot all output data. To do this, use the (default) setting for "**Plot Variables**" = "**All**". Every output name automatically be included in the resulting plot file. Please note that this is the default setting, but as your model grows, this setting should be replaced with manually specified names per the "**Selected**" setting.

Caution:	The *Plot All Data* method will plot every output in the model, to a maximum of 8000 variables. This can potentially generate a huge plot file. Make sure you have sufficient disk space when creating large plot data files!

Dependent variables can be plotted as a function of “time” (the default) or any other output variable. By default, dependent variables are plotted on individual sets of axes. However, you can over plot all variables for one display on a single set of axes using the “Overplot?” option.

Easy5 automatically scales plot axes for you, unless you want to set up your own scales with the “Manual” scaling option. These advanced plotting options are described in the following paragraphs.

Overplot? To request overplotting, select “Yes” in the “Overplot?” advanced Plotting tab setting for each display you want over plotted. All dependent variables defined on the respective display line will be plotted on one set of axes.

Auto-Scale? When “Yes.” is selected, scaling will be done automatically for the respective display. If you don't want automatic scaling, use the “No” setting, and specify the appropriate values for Manual scales. Please note that manual scaling can also be done using the Easy5 Plotter later.

Setting up Simulation Print Output

Simulation data can be written to the output listing file or can be totally suppressed. In general, printing is independent of plotting, so you can print and/or plot any combination of simulation outputs.

The steps for printing results are similar to those for setting up plots. Specifically:

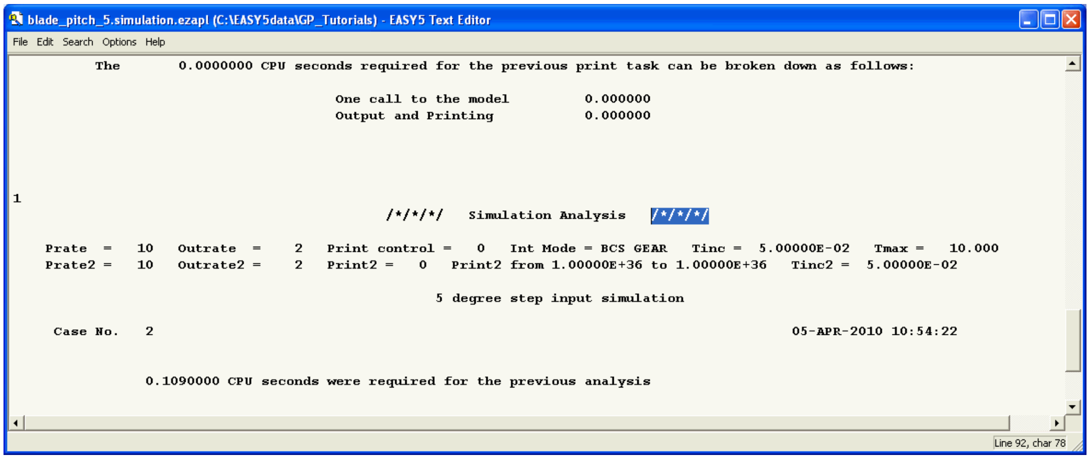
1. You tell Easy5 that you want to print data.
2. You specify how often output variables are to be printed.
3. You identify the output variables to be printed.

There are three choices for indicating that you want to print data: “No”, “All” and “Selected.” “No” is the default option resulting in no data being printed. If “All” is used, Easy5 will print all simulation output variables to the analysis output listing file at the interval you specify.

The “Selected” print option allows you to specify which output variables will be printed. If you choose “Selected”, an additional settings are displayed in the data form to allow you specify a list of names. Use these fields in addition to the appropriate “Primary Print Rate” (General Tab) field to specify the variables to be printed and the rate at which they will be printed, respectively.

Figure 129 is a sample of a “Selected” output listings. If ten or less variables are requested for output, the data will be listed in columns as shown in the figure. Otherwise, the data is displayed in a paragraph format.

The “Primary Print Rate” (General tab) data field defines how often you want output data printed. “Primary Print Rate” is an integer that Easy5 multiplies by the “Primary Plot Rate” value, to determine how often it should print data.


blade_pitch_5.simulation.ezapl (C:\EASY5data\GP_Tutorials) - EASY5 Text Editor
File Edit Search Options Help
Line 92, char 78

```
          The       0.0000000 CPU seconds required for the previous print task can be broken down as follows:

                                       One call to the model          0.000000
                                       Output and Printing            0.000000



1
                                           /*/*/*/   Simulation Analysis   /*/*/*/

    Prate  =   10   Outrate  =    2   Print control =   0   Int Mode = BCS GEAR   Tinc =  5.00000E-02   Tmax =   10.000
    Prate2 =   10   Outrate2 =    2   Print2 =   0   Print2 from 1.00000E+36 to 1.00000E+36   Tinc2 =  5.00000E-02

                                                5 degree step input simulation

     Case No.   2                                                                     05-APR-2010 10:54:22

                0.1090000 CPU seconds were required for the previous analysis
```

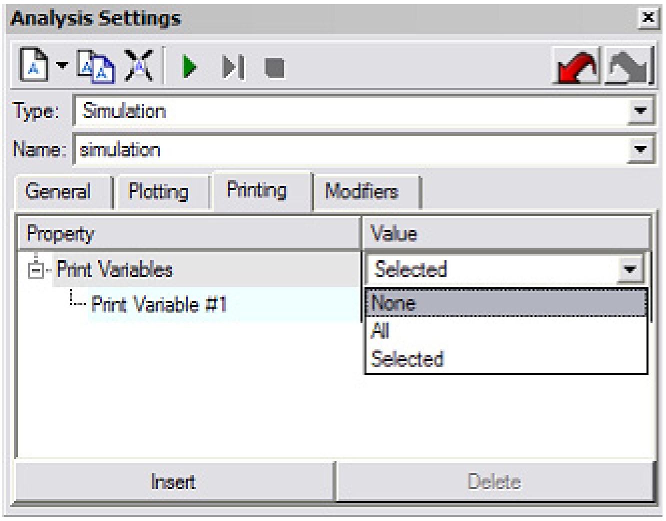
Figure 129 Sample of Printed Simulation Analysis Output

For example, if you wanted data printed every 1.0 seconds, and "Time Increment" was 0.01 seconds and "Primary Plot Rate" was 10 (i.e., the plot rate is 0.1), you would set "Primary Print Rate" value to 10. Figure 127 previously showed the relationship between "Time Increment", "Primary Plot Rate", and "Primary Print Rate."

Note:	The "Time Increment" value specifies how often data is available for printing and/or plotting, not how often it actually will be printed and/or plotted. Be sure you understand the relationship between "Time Increment", "Primary Plot Rate" and "Primary Print Rate."

You specify the print variables by selecting the Print tab on the simulation data form. The print specification form shown in Figure 130 appears on your screen.

Use this form to specify up to 40 output variable names. Variable values will be printed as a function of time in columnar form in the order specified when reading from left to right.


Analysis Settings

Type: Simulation

Name: simulation

General | Plotting | Printing | Modifiers

Property	Value
Print Variables	Selected
Print Variable #1	

Figure 130 Print Specification Form

You insert variable names in the simulation print specification form using the same method to insert names in the plot specification form, as described in the previous section.

Note: There is no need to specify TIME as a print variable. TIME is automatically printed as the first variable in the output listing file.

Specifying Secondary Plotting and Printing Rates

During a simulation it is possible to print and/or plot data at rates different from those established with the “Time Increment”, “Primary Plot Rate”, and “Primary Print Rate” parameters, by specifying secondary print or plot rate parameters. For example, at the beginning of a simulation you may want a high plot rate of 1000 samples per second to get a close look at a start up transient, yet after a few seconds need a relatively slower rate of 100 samples per second for the rest of the run.

To set up secondary output rates, select “Yes” after “Use Secondary Print/Plot Rates?” in the Advanced portion of the General tab. Additional data fields will appear on your screen.

These fields are described below:

- The “Secondary StartTime” field is used to define the simulation time at which the secondary time period should begin.
- The “Secondary Stop Time” field is used to define the simulation time at which the secondary time period should end.
- The specified value for “Secondary Time Incremenet” will be used in place of the “Time Increment” value during the secondary time period.

- The specified value for “Secondary Plot Rate” will be used as the “Time Increment” or “2nd Time Incr” (whichever is active) multiplier during the secondary time period.
- “Secondary Print Rate” is an optional value. If it is specified, it will be used as the “Plot Rate Multiplier” or “Secondary Print Rate” (whichever is active) during the secondary time period.

Saving a Final Simulation Operating Point

To save the operating point calculated during the last time step of the simulation, select “**Yes**” following “**Save Final Operating Point?**”. A data field, “**Final Operating Point Name**” will become active on your form. The name you enter in this field is used to name the “saved” operating point. To subsequently restore the saved operating point for another analysis, this name will appear in the list of operating point names that Easy5 offers you during the “**Restore operating point**” function.

Note:	When you save the operating point at the end of a simulation, the final simulation time (the “Stop Time”) is also saved as part of the operating point data. When the operating point is restored, the correct “Time” value is also restored.

Executing the Simulation

There are two methods of executing (submitting) the simulation: execute *without* the symbolic debugger (default method), and execute *with* the symbolic debugger.

Nominal Execute

Selecting the **Execute the current analysis** analysis toolbar icon will start the simulation job as a background process. If the model executable needs to be rebuilt, it will do that automatically.

Execute With Debug

There are times when the standard print and plot output from Easy5 is not sufficient for isolating a problem that is occurring in your model. For these cases you should use the system symbolic debugger. First, select the **Build > Debug Mode** setting. Then, make sure that the model executable is rebuilt, by selecting the **Build > Create Executable** menu item. This generates a debuggable version of the model executable. Then, to execute with debug select the **Debug the current analysis** toolbar icon (which should now be active). For more information, see “Debugging the Model and Analysis”.

Centralized Activation of Interactive Simulation Widgets

In the **Advanced** section in the **General** Tab is a setting called **Interactive Simulation Activation**. This option provides you with a way to centrally control the activation of potentially active Interactive Simulation components in your model by including the following 3 choices: **None Active, TI Only, All Active.**

The Interactive Simulation components are special components belonging to the Easy5 Interactive Simulation (IS) library that once placed in your model and activated, provide the means to communicate interactively with the background simulation. Normally, they are controlled by the ACT parameter for each individual IS component.

However, to provide a more centralized method for controlling these components, Easy5 provides these options as described below:

Value	Description
None Active	All otherwise active IS components in your model will be deactivated.
TI Only	All otherwise active IS components, except for IS/TI components will be deactivated. The TI block provides the minimum Interactive Simulation overhead and feedback with your model simulation, and is most commonly used.
All Active	All otherwise active IS components are activated (default)

Please see User Guide, Chapter 4 - Interactive Simulation for more details on Interactive Simulation components.

Simulation Outputs Results

Simulation outputs can be plotted and/or printed. These outputs are described in the following sections.

Plots

If simulation plots were generated during your analysis, you can look at them with the Easy5 Plotter. When the analysis is complete, the plotter will automatically displays and plot the data from the analysis. See User Guide, Chapter 7 - Easy5 Plotter for information on how to use the plotter.

The Simulation Output Listing

To examine the output listing generated by a simulation, you first need to select **Display Analysis Output Listing** (`Ctrl+L`) from the **Analysis** menu. When the output listing file displays on your screen, search for the string “SIMULATION ANALYSIS” to position yourself at the start of the simulation output. The output listing can be broken down into two basic parts: an input data summary, and the simulation output.

Information is listed at the top of the form which should reflect the simulation options you specified earlier in the simulation data form. The “PRATE” (Primary Print Rate), “OUTRATE” (Primary Plot Rate), “PRINT CONTROL” (the flag which indicates output format), “INT MODE” (Integration Method Number), “TINC” (Time Increment), “TMAX” (Max Time), “PRATE2” (Secondary Print Rate), “OUTRATE2” (Secondary Plot Rate), “PLOT EVENTS” (Add Plot Points at Switch/Discrete Events) and “PRINT2” from, to (Secondary Start and Stop Times) values are listed in the header above the title you assigned.

Below the input data summary are the results from the simulation run itself. Depending on the selection made for PRINT RESULTS in the simulation data form one of the following displays:

- A listing of all states, rates, and variables at every print output interval.
- A listing of only selected quantities.
- No listing.

After the last time point, the total number of CPU seconds expended during the simulation is listed.

Simulation Monitor

The Simulation Monitor is a special tool used to monitor the simulation results while the simulation is running. This is generally used to monitor long simulation runs. The monitor takes the “live” simulation results as it is being generated, and inserts the data into the Easy5 plotter while the simulation is running.

What is the Simulation Monitor?

The Simulation Monitor is a special simulation tool used to display the simulation data in the plotter window while the simulation is running. The data generated by the simulation is written to memory, and is read in by the plotter and displayed in the plotter window.

The plotter is the basic Easy5 Plotter, but is run in the “monitor” mode. It reads in data generated from a simulation and updates the plot data at a given time interval set by the user. Because it uses the basic Easy5 plotter, you can choose different displays, apply a layout template, and generally use the basic features of the Easy5 plotter.

Activating the Simulation Monitor

There are two ways to activate the Simulation Monitor feature, First, in the simulation analysis data form, set the Monitor Simulation radio button to “Yes”. When the simulation is executed, the plotter will automatically displays in the monitor mode and display the simulation data.

Another method of activating the Simulation Monitor is while the simulation is running. When a simulation is running, the **Analysis > Monitor Simulation** menu will be active. Select this menu option (or the F6 shortcut key) and the plotter displays in the monitor mode, and displays the simulation data.

Simulation Monitor Plotter Features

In the monitor mode, the plotter is identical to the basic Easy5 plotter, but contains an additional pulldown menu called Monitor with the following options:

- **Get Current Data**

  Gets the current data from memory and updates the plot.
- **Automatic Data Update**

  This option is toggled O*n* or *Off*. When *On*, the plotter automatically updates the plot at an update interval set by the menu item Set Data Update Interval.
- **Set Data Update Interval**

  Used to set how often the data is updated. The data interval is in wall clock time, in seconds.

Running the Simulation Monitor

The simulation writes the data to a temporary binary file named EZ5_PLOT.HDR. The plotter loads in the data from this file and displays the data. When the plotter first opens, it takes a few seconds before the data is available.

At first, no data is displayed. It takes a few seconds for the simulation data to become active and available. When ready, the data will automatically be plotted in the plot window and will update the data at a user-defined interval. To set how often you wish the data to be updated (in seconds), select **Monitor > Set Data Update Interval**. Enter the time in seconds. This sets how often the plotter updates in wall clock seconds (not simulation seconds). This makes sense because, slow simulation runs that take a long time, should be updated in real time, not simulation time.

By default, the plot data is automatically updated. To turn the automatic update of plot data on or off, select the plotter menu **Monitor > Automatic Data Update**.

While the simulation is running, you may zoom in/out, view data, and perform the standard plot functions. You can even apply a layout format. However, this format is not saved.

When the simulation is finished, the temporary plot file is closed, and the final Easy5 plot file is automatically opened and displayed.

Note:	You cannot perform all Plotter operations, such as exporting plot data, editing plot files, while in the Monitor mode.

Simulation Troubleshooting

See Also:	"Debugging the Model and Analysis"
	"Integration Methods"

Several types of problems commonly occur during simulation. Problems manifest themselves by either causing an abnormal termination, or by taking an inordinate amount of CPU time to complete a simulation. Solving these kinds of problems or trying to improve the performance of your simulation is not always a simple task and, unfortunately, there is no prescribed formula for success. However, the following guidelines and techniques are useful.

Type of Failure

When the simulation fails, the following message is displayed in the message line:

```
Analysis has stopped before completion - errors may have occurred.
```

When this occurs, you must determine what type of failure occurred. To do this, examine the analysis output listing file (also called the ezapl file). This file is usually opened for you automatically when a failure occurs. However, if this is not the case, then select **Analysis > Display Analysis Output Listing...** to open the file:

Analysis Output Listing

The Analysis Output Listing file displays warning and error messages that are important. Search for the following warning message and data:

```
                ******** WARNING ***********
     THE Easy5 ANALYSIS PROGRAM HAS TERMINATED ABNORMALLY
     MODEL VALUES AT TERMINATION WERE AS FOLLOWS:
     STATES:
         .......... list of states and values ......
     RATES:
     .......... list of the rate of the states and values ......
     VARIABLES:
     .......... list of variables and values ......
     PARAMETERS:
     .......... list of parameters and values ......
```

This warning message tells you that the simulation has failed, and it also lists the TIME, and all model data at the time of failure (states, rates, variables and parameters).

You can examine the warning message and data to determine the following:

- At what TIME did the simulation fail? If TIME=0, the failure occurred during calculation of initial conditions. Otherwise, it may have failed into the simulation.
- Perform a general check on all data values - do they fall within a reasonable range?
- Look at the states, rates and variables at the time of failure. Do the state values make sense? If some state values are abnormally large, (>1E15), then this may indicate an integration failure. Are any of the rates very large? If so, the integrators may have failed.

Scroll to bottom of this file and you'll see the following message and data:

```
-----------------------------------------------------------------
--------------   VARIABLES IN COMPUTATIONAL ORDER     ------------
-----------------------------------------------------------------
....... list of variables.......
                ********** WARNING *********
THE ABOVE VALUES REPRESENT MODEL STATUS AT ABNORMAL TERMINATION OF THE
Easy5 ANALYSIS PROGRAM
```

The variables are listed in the order in which they are computed in the model source file (model_name.f or.c). This can be useful information. Scan through the list (from left to right) and gauge the values. If a variable has an abnormal value, then the failure took place either when the variable was calculated or just before. For example, if a variable has a value of "-I", this indicates an arithmetic indefinite, meaning that an improper calculation occurred, such as a divide by zero. This will help locate where in the code the failure occurred.

The very last lines in the analysis output listing file will usually display important error messages that will tell you what type of failure occurred. Some of these messages are summarized below.

Simulation Failure Error Messages

When the simulation fails, the error message is printed at the bottom of the analysis output listing file. When you get any of these messages, you must debug your model. See "Debugging the Model and Analysis" for information how to debug your model and resolve these errors. The most common error messages are summarized as follows.

Divide by Zero

If your code has a division equation and the denominator has a value of zero, you will get a floating point error message as follows:

```
    FLOATING POINT ERROR: DIVISION BY ZERO
USE DBX (OR DBXTOOL) AND THE DBX WHERE COMMAND TO FIND THE LOCATION OF
THE ERROR
```

This is one of the most common bugs. Use the debugger to locate where the division by zero occurs.

Invalid Operation

An invalid operation occurs when an invalid mathematical operation occurs, such as the square root of a negative number. The following error message is displayed:

```
   FLOATING POINT ERROR: INVALID OPERATION
     USE DBX (OR DBXTOOL) AND THE DBX WHERE COMMAND TO FIND THE
LOCATION OF         THE ERROR
```

Overflow

A mathematical overflow condition occurs when the numeric value exceed the numeric limit of your machine. The following error message is displayed:

```
   FLOATING POINT ERROR: OVERFLOW
     USE DBX (OR DBXTOOL) AND THE DBX WHERE COMMAND TO FIND THE
LOCATION OF  THE ERROR
```

This can happen if the system is unstable (poles in the right hand plane). The state value will grow until an overflow occurs. Use the debugger to determine which data value is overflowing. Also check your model for incorrect connections and possibly a positive feedback that causes the system to be unstable. Perform a Linear Model Generation to determine the eigenvalues at the time of the simulation failure.

Write Your Own Diagnostics to the Output Listing File

You can write your own diagnostic messages and print out data to help you debug your code. Just write to the unit number called IWRITE. IWRITE is an Easy5 reserved word that writes the output to the Analysis Output Listing file.

For example, the following lines of code can be added to a Fortran component. This is just an example of how you can use IWRITE to write any data to the output listing file.

```
C If DIAGNST flag is on (=1) then print out diagnostic message & data
     IF ((DIAGNST.EQ. ONE) .AND. (VALVE .GT. ZERO)) THEN
        WRITE(IWRITE, *) '*** Pressure and Flow Data; Valve= OPEN ***'
        WRITE(IWRITE,*) 'Time=', TIME, 'Pressure= ',PRES, 'Flow=',FLOW
     ENDIF
```

Monitoring CPU Time During a Simulation

The cost of performing a particular analysis, as measured in CPU seconds, is given at the end of each simulation. The efficiency of the various integration techniques can be assessed by this measure. The amount of CPU time consumed during a simulation run can be monitored with the **CPUSEC** variable. This is an Easy5 variable representing the number of CPU seconds used for the current analysis. By plotting this variable versus **TIME**, you can determine if any portion of a simulation run is consuming a disproportionate amount of the total CPU time. By comparing runs using different integration methods, you can determine a measure of relative efficiency between methods. You will have to weigh this measure against the required accuracy, as the accuracy may change between methods.

Killing a Simulation Run

If the simulation performance slows and almost grinds to a halt it could be due to the failure of the variable step integration method. The integration algorithm will repeatedly decrease the integrator step size until it finally reaches the machine's epsilon value (smallest numeric value). This can consume a large amount of CPU time. If this occurs, you should kill the job and analyze what's going on. Either select the "**Stop the currently running analysis**" analysis toolbar icon, the "`[Terminate Simulation]`" interactive simulation button (active when running with interactive simulation widgets), or select the following menu items to kill the job:

Analysis > Stop Current Analysis

The following messages will be displayed in the status message area:

```
Trying to stop background process...
Signalling background process...
Process successfully stopped
```

Plot the data (if there is any) and open an examine the Analysis Output Listing file and look at the data. Also, try repeating the simulation, and turn on the integration diagnostics as described in the next section.

Integration Problems

For fixed step methods, the selection of **TINC** is critical and should be about ten times smaller than the smallest natural frequency in your model. In some cases, it should even be smaller (some standard component require a fixed step integration method). If the value for **TINC** is too large, the simulation will be numerically unstable and will appear to "blow up" when, in fact, its eigenvalues appear to make it look stable.

For variable step methods, in particular the **BCS GEAR**, **STIFF GEAR** and **ADAMS** methods, a simulation may get bogged down for several reasons. One common reason is that there are discontinuities that occur during the transient. Discontinuities are defined as derivatives that exhibit discontinuous behavior. When this occurs, the integrator will try to find the exact point in time where the discontinuity occurred; this may take a lot of time, and it may eventually fail. Sometimes, the values of the **ERROR CONTROLS** may also need tuning. Experience has shown that it is often better to reduce error controls for states where accuracy is critical.

Important: Default error controls are usually sufficient for most problems. However, they can be adjusted manually. Error controls should typically fall in the range of 1.E-8 > x > 0.001, but certainly not be set smaller than 1E-10. Certain application, such as hydraulics, work better with tighter error controls (and so defaults are smaller for such states), but please consult the application library notes for specific advice.

II Component For Integration Information

A useful diagnostic component for these methods (**BCS GEAR, STIFF GEAR, ADAMS,** or **RADAU**) is to use the *II - Integration Information* component from the General Purpose library. This component outputs information about how the central integrator is performing, including the step size, order, number of completed steps, number of calls to EQMO, and the number of Jacobian evaluations. To use this, add the *II* component to your model (this component is not connected to any other component), then, plot and/or print the outputs of the *II* component.

Print Integration Diagnostic Function

Another useful diagnostic tool for these methods (**BCS GEAR, STIFF GEAR, ADAMS,** or **RADAU**) is the "print integration diagnostics" function. In the Simulation Analysis Data Form, turn on the "*Print Detailed Diagnostics*" (General tab, advanced) setting by selecting the "*Yes*" option.

A large amount of data will be printed to the Analysis Output Listing file, so try to keep the simulation time down to a reasonable value (set *Stop Time* to a small value). You want to get a detailed picture of the portion of the transient where you suspect a problem is occurring. This diagnostic printout, which includes a definition of terms, can be very helpful in showing which Easy5 states are posing problems to these integration methods.

You can also use the II component to turn the diagnostic print on and off at a specified time interval. This allows you to print the diagnostic only during a time of interest.

An example of this output obtained by turning on the integration diagnostic is shown below. First, the output from few successful steps is shown, then the diagnostic print of a failed integration step is given.

Diagnostic Output of Successful Integration Steps:

```
  NRKVS COMPLETED STEP NO.    1 AT TIME=   3.98551E-03 USING H=   9.05797E-05
WITH MAX H=   1.00000E+37 ERROR=   1.07686E-08
  NRKVS COMPLETED STEP NO.    1 AT TIME=   4.07609E-03 USING H=   9.05797E-05
WITH MAX H=   1.00000E+37 ERROR=   9.90447E-09
 NRKVS DETECTED DISCONTINUITY AT TIME=    4.1666666666667D-03
  NRKVS COMPLETED STEP NO.    1 AT TIME=   4.16667E-03 USING H=   9.05797E-05
WITH MAX H=   1.00000E+37 ERROR=   9.10190E-09
 NRKVS DETECTED DISCONTINUITY AT TIME=    4.2572463768116D-03
  NRKVS LOCATED DISCONTINUITY AT TIME=    4.1666666666667D-03
  NRKVS COMPLETED STEP NO.    1 AT TIME=   4.16767E-03 USING H=   1.00000E-06
WITH MAX H=   1.00000E+37 ERROR=   1.17505E-03
  NRKVS COMPLETED STEP NO.    2 AT TIME=   4.16967E-03 USING H=   2.00000E-06
WITH MAX H=   1.00000E+37 ERROR=   1.26049E-15
  NRKVS COMPLETED STEP NO.    3 AT TIME=   4.17367E-03 USING H=   4.00000E-06
WITH MAX H=   1.00000E+37 ERROR=   4.02591E-14
```

```
  NRKVS COMPLETED STEP NO.   4 AT TIME=   4.18167E-03 USING H=   8.00000E-06
WITH MAX H=   1.00000E+37 ERROR=   1.28376E-12
  NRKVS COMPLETED STEP NO.   5 AT TIME=   4.19767E-03 USING H=   1.60000E-05
WITH MAX H=   1.00000E+37 ERROR=   4.07892E-11
  NRKVS COMPLETED STEP NO.   6 AT TIME=   4.22967E-03 USING H=   3.20000E-05
WITH MAX...
```

Diagnostic Output of a Failed Integration Step

```
*********************************************************************
DIAGNOSTIC MESSAGE INTERPRETATION
    IF PREDICTOR AND CORRECTOR STEPS DISAGREE OR CORRECTOR CONVERGENCE FAILS,
THE NAME, VALUE, AND RATE FOR THOSE STATES WITH AN
    ERROR RATIO GREATER THAN ONE ARE PRINTED.  THESE ARE PRECEEDED BY, -P- OR
-C- , AND FOLLOWED BY:
     T = THE VALUE OF TIME AT THE LAST SUCCESSFUL STEP
     H = THE CURRENT INTEGRATION STEP SIZE
     ORD = THE CURRENT ALGORITHM ORDER
     ER = THE ERROR RATIO FOR EACH STATE
     NS = THE INTEGRATION STEP NUMBER
     JE = THE NUMBER OF JACOBIAN EVALUATIONS
     CPU = THE CPU SECONDS USED SINCE START OF RUN

*****************************************************************************
*-*-*-*-*-*  DRIVE2 COMPLETED STEP **** AT TIME=   15.399045 CPUTIME=   71.000
HUSED= 0.614286E-03 ORDER=5 NFE= 31600 NJE=  439
 *-*-*-*-*-*  DRIVE2 COMPLETED STEP **** AT TIME=   15.399660 CPUTIME=
71.000 HUSED= 0.614286E-03 ORDER=5 NFE= 31601 NJE=  439
 *-*-*-*-*-*  DRIVE2 COMPLETED STEP **** AT TIME=   15.400000 CPUTIME=
71.000 HUSED= 0.340308E-03 ORDER=5 NFE= 31602 NJE=  439
 -P- Actuator_po=  614090.     RATE= 4.3538E+05 FAILED AT T=   15.4000 H=
3.4031E-04 ORD= 5 ER= 4.4E+06 NS= 28188 JE=439 CPU=   71.00
 -P- X1 TF       = 1.766675E+07 RATE=-5.9791E+08 FAILED AT T=   15.4000 H=
3.4031E-04 ORD= 5 ER= 3.8E+04 NS= 28188 JE=439 CPU=   71.00
 -P- Pitch_angle=  855224.     RATE= 9.1152E+06 FAILED AT T=   15.4000 H=
3.4031E-04 ORD= 5 ER= 2.0E+02 NS= 28188 JE=439 CPU=   71.00
```

Controlling Print Diagnostic Using the IDIAG Flag

The simulation data form provides a method for printing integration diagnostics as described in the previous section. However, this will print the diagnostics for the entire simulation, which may result in excessive print data.

You may use the IDIAG flag in a code block (User Code or Library component) to turn the integration print diagnostic on or off at will. Setting IDIAG=1 turns on the integration diagnostics, and IDIAG=0 turns it off.

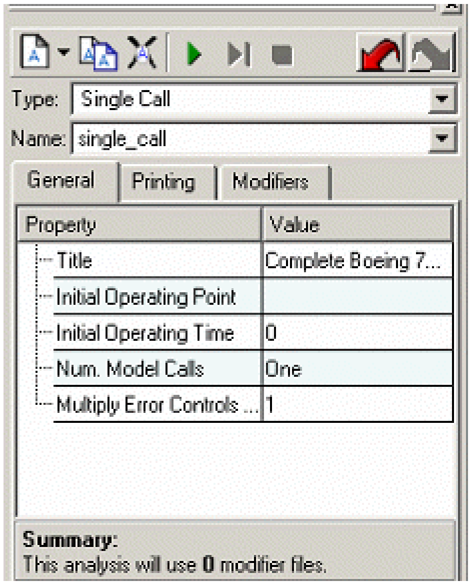
An example of code using IDIAG is shown below. In this example, the input parameter DIAG_ON sets the time at which to initiate printing the integration diagnostic, and the input parameter DIAGOFF sets the time at which to terminate the print.

```
* DIAG_ON= Time at which to turn diagnostic on
* DIAGOFF= Time at which to turn diagnostic off
    if (TIME .ge. DIAG_ON .AND. ITINC .eq. 1) IDIAG= 1
    if (TIME .gt. DIAGOFF .AND. ITINC .eq. 1) IDIAG= 0
```

Single Call Analysis

During model checkout, and prior to some analyses, you will want to make a "single call" to your model equations and print the results of the model calculations (states, rates and variables). When you request a single call analysis, the states are set to their initial condition values, time is set equal to the “Time=” value defined in the data form, and the model is executed once. The Single Call Analysis issues a PRINT command to the analysis input file. In effect, this is a snapshot of your model at the current operating point.

The Single Call Analysis is setup and executed using data form is shown in Figure 131. To open this data form, select **Analysis > Miscellaneous > Single Call...**The general usage of this data form is described in “Analysis Data Form”. This data form also contains the “Debug this analysis” toolbar icon that allows you to run this analysis with the symbolic debugger. See “Debugging the Model and Analysis” for information on this feature. The “Num. Model Calls” settings is unique to this data form and is described in the following paragraphs.



Type: Single Call
Name: single_call

Property	Value
Title	Complete Boeing 7...
Initial Operating Point	
Initial Operating Time	0
Num. Model Calls	One
Multiply Error Controls ...	1

Summary:
This analysis will use **0** modifier files.

Figure 131 Single Call Analysis Data Form

The Single Call Analysis allows you to specify the number of model calls as either one or two. One model call executes the single call one time through your model, whereas, two model calls executes the single call two times.

The one model call is the most commonly used option. This is used to give you a quick snapshot of your model. This analysis can also be used to locate arithmetic errors in your model equations by selecting the **Build > Debug Mode** option from the main menu bar. Print statements generated by this debug option execute only during the single call analysis.

For more information on using the debug option, refer to "Debugging the Model and Analysis". The one call Single Call Analysis can also be used in conjunction with the Linear Model Generation Analysis, to obtain a linearized picture of the operating point.

The two model call option is used to determine if there is “internal algebraic memory” being set in your model equations (your User Code or Library Component code). Internal algebraic memory occurs if a state or variable is defined in such a manner that the value changes even though time remains fixed (i.e. no integration step is taken). Internal algebraic memory will cause any variable step integration method to fail.

The two model call option is used to check for the existence of internal algebraic memory. Execute the Single Call Analysis with the two model call option. When the analysis has completed, analyze the output listing file. The output data from the first call of the single call can be visually compared with the output data of the second single call. Since time has not changed, the output data should be identical. If an output variable or state has changed, then you have determined that the model contains internal algebraic memory.

For models containing large amount of data, you may find it helpful to edit the output listing file and create two files; the first file containing the output of the first single call, the second file containing output of the second single call. Then perform a “difference” between these two files to detect changes in data values.

Caution:	Algebraic memory causes variable step integrators to fail! If you detect the occurrence of internal algebraic memory, analyze your user-defined code (contained in User Code or Library components) and remove the algebraic relationship that causes the internal memory. If this is not possible, you will be forced to use a fixed step integrator.

Sort Blocks

See Also: User Guide, Chapter 5 - Code Components

Sort blocks allow you to break up your user-defined Fortran or User Code code into one or more blocks of code that can then be sorted by Easy5 to build an explicit model. This powerful feature gives you some control over the model sorting process and allows you to break implicit loops in your model. This section review model sorting, and how to define sort blocks in Fortran and User Code components.

Model Sorting

Before Easy5 builds the executable source file (the *model_name*.f file which contains the Subroutine EQMO), the model generation program sorts the components to define an explicit model. An explicit model is one for which all output variables are calculated before they are used as input to other components. Essentially, Easy5 looks at the input/output relationships of all components in your model, and reorders the logical sequence of the component to form an explicit model.

You can build your graphical on screen model and place the components in any order and Easy5 will automatically sort the components to create an explicit model. Since many standard components are represented by subroutine calls in EQMO, this reordering process consists of rearranging the order in which the component subroutine calls appear in the model. For other components, such as Fortran and User Code components which contain in line code as opposed to subroutine calls, the input/output relationship is used to determine the order in which these components will appear in EQMO. Therefore, if your model has been reordered, you may notice that the structure of EQMO differs from that specified in your graphical model.

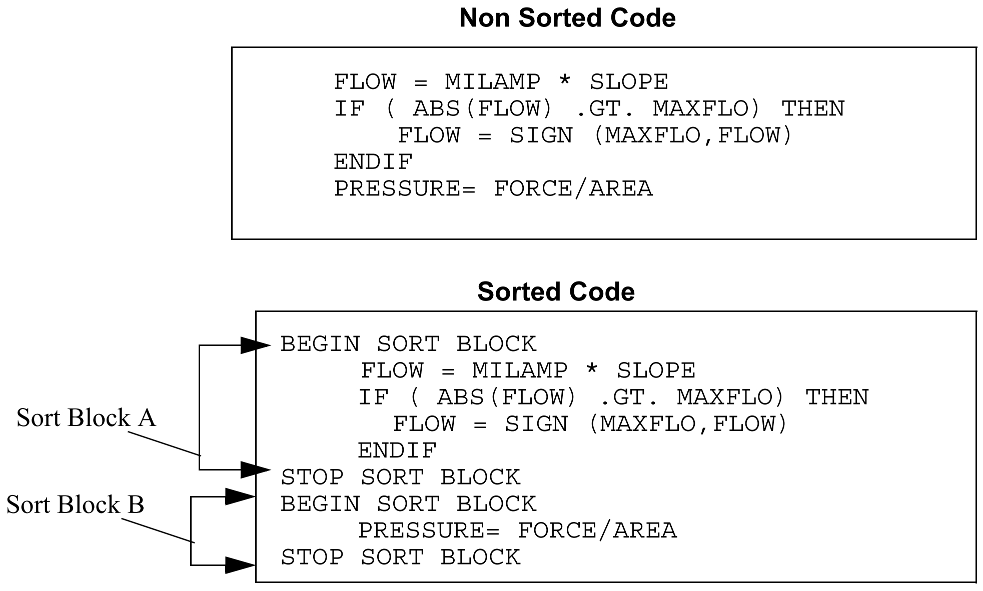
Note: You should make a habit of analyzing the executable source file. This file shows you how the components and equations are sorted to form an explicit model.

However, there are times when a system model cannot be placed into an explicit from because it is an implicit model. When this occurs, Easy5 warns you and describes the algebraic loops that cause the implicit model (see "Implicit Model"). If the implicit loop is caused by a Fortran or a User Code component, you may be able to break this loop by using "sort blocks" as described in the sections that follow.

Defining Sort Blocks

Sort blocks can only be define in a Fortran or a User Code component. Sort blocks are defined by enclosing code in special Easy5 commands called `BEGIN SORT BLOCK` and `STOP SORT BLOCK`. The `BEGIN SORT BLOCK` command tells Easy5 to suspend sorting. This command will preclude further sorting until the end of the component or until you include the `STOP SORT BLOCK` command in your code. You may use `BEGIN SORT BLOCK` and `STOP SORT BLOCK` to divide your Fortran and Library code into as many sort blocks as needed. Each sort block is then "seen" by the model builder as a separate component, and can then be moved (sorted) into a location that forms an explicit model.

A simple example of defining sort blocks is shown in Figure 132.



```
FLOW = MILAMP * SLOPE
IF ( ABS(FLOW) .GT. MAXFLO) THEN
    FLOW = SIGN (MAXFLO,FLOW)
ENDIF
PRESSURE= FORCE/AREA
```

```
BEGIN SORT BLOCK
     FLOW = MILAMP * SLOPE
     IF ( ABS(FLOW) .GT. MAXFLO) THEN
       FLOW = SIGN (MAXFLO,FLOW)
     ENDIF
STOP SORT BLOCK
BEGIN SORT BLOCK
     PRESSURE= FORCE/AREA
STOP SORT BLOCK
```

Figure 132 Example of Defining Sort Blocks

This example shows how a body of Fortran or Library Component code can be broken into two sort blocks. The code in the upper box is one block of unsorted code. This has been broken into two sort blocks in the lower box, designated as sort block A and B. The sorted code will allow the model sorting routine to move the two sort blocks anywhere in the model as required.

For example, if the calculation of PRESSURE in sort block B is needed “upstream” in the model, then this line of code will be extracted and moved before the placement of the component that requires the variable PRESSURE. This will break the implicit loop that results when using the unsorted code.

Notice that in Figure 132, the user could have used two Fortran (or Library) components to input the code; the first component would contain the code in sort block A, the second component the code from sort block B.

This puts the burden on the user to know a prior how the equations and mathematical relationships are defined when building a model, and it also requires the usage of a large number of components. Sort blocks are a powerful feature that takes the burden off the developer. With sort blocks, you can put the burden of sorting code and blocks of code on the Easy5 model builder.

Fortran and Library Component Sort Blocks

By default, Fortran component code is not sorted. That is, the entire code in a Fortran component is considered to be a single sort block. To break a Fortran component code into multiple sort blocks, you must first select the [**Sorted**] checkbox in the code editor toolbar. Then, you can use the `BEGIN SORT BLOCK` and `STOP SORT BLOCK` commands to define individual sort blocks. Please note that without these sorting commands, each line of code will be treated as a separate sort block!

If the Sorted box is unchecked, Easy5 (specifically, the code generator) only sorts on inputs that are connected -- it does not look at the sort block code to confirm that quantities defined (anything on the left-hand-side of an equals “=” sign) are assigned by variables already defined previously. A Fortran sort block with no connected inputs is thus always sorted “to the top” (but always after special “mandatory” library components that define global variables for a particular application library).

If the Sorted box is checked, the Easy5 code generator not only sorts on the inputs that are connected, but it also analyzes the (sort block’s) code body to make sure that all quantities defined (anything on the left-hand-side of an equals sign) are defined in a previous (or current) sort block.

There is no specific option for sorting for Library components. All library components have sorting activated. However, by default, Library component code is also not sorted, through the default inclusion of a `BEGIN SORT BLOCK` command at the top of the library component code body. Therefore, to define sort blocks, you need to use alternating `STOP SORT BLOCK` and `BEGIN SORT BLOCK` commands, as appropriate. For more information on sorting Library Component code, refer to "Using Library Component Sorting Options" in the User Guide, Chapter 12 - Library Components. This chapter also contains a good example showing an implicit loop and how the implicit loop is broken using sort blocks.

Stability Margins Analysis

See Also: "Analysis Data Form"

"Stability Margin Analysis Method"

The Stability Margins analysis calculates maximum and minimum values for user specified parameters that result in stable system operation. This analysis also determines the oscillation frequencies that result if either boundary is exceeded. Stability margins for up to ten system parameters can be calculated in a single analysis.

The Stability Margins analysis does not work with systems that contain discrete states, that is, sampled data systems. The Stability Margins analysis assumes that the system being analyzed is linear with respect to the parameters selected. If this assumption is not a good approximation, then the answers produced by stability margins may be misleading. Root Locus analysis will calculate accurate gain margins for situations in which the assumption of linearity does not hold.

Setting up a Stability Margins Analysis

The stability margins analysis is setup and executed using the Stability Margins Analysis Data Form. This data form is accessed by first selecting **Analysis** from the main menu, then selecting **Stability Margins...** from the **Linear** menu. Figure 133 shows the stability margin data form with all data fields and options displayed.

To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to the topic, "Analysis Data Form".

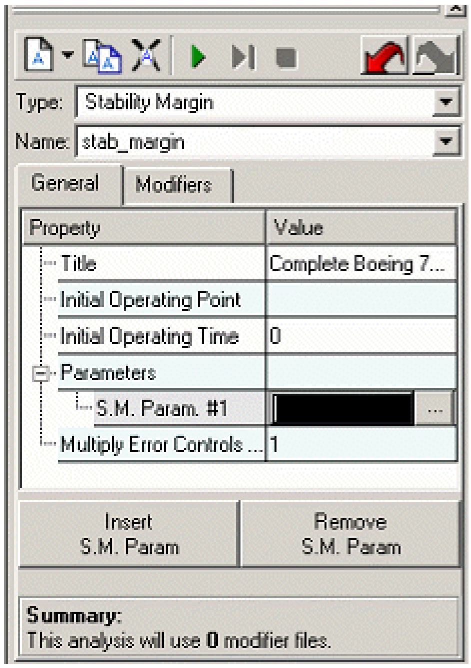
Defining Stability Margins Parameters

You can specify up to ten parameters for which stability margins will be calculated. Stability margins parameters are entered in the ten data fields following "Parameters" in the stability margins data form. The only way to place parameter names in the data form is to select components in your on screen schematic using the "Pick" option. When you do this, Easy5 will display a list of parameter names associated with the component selected. Select a name from this list and Easy5 will copy that name into the data form for you.

To delete parameters from this section, select the name with your left mouse button (use a triple CLICK-L to highlight the entire field), and enter the space bar to clear the field.

Limitations

The stability margin search is limited to parameter values of the same algebraic sign as the nominal value. For example, zero is the lowest magnitude that will be considered for the lower stability boundary of a parameter with a positive nominal value.


Type: Stability Margin

Name: stab_margin

General | Modifiers

Property	Value
Title	Complete Boeing 7...
Initial Operating Point	
Initial Operating Time	0
Parameters	
S.M. Param. #1	
Multiply Error Controls ...	1

Insert S.M. Param | Remove S.M. Param

Summary:
This analysis will use **0** modifier files.

Figure 133 Stability Margins Analysis Data Form

Stability Margin Analysis Method

The method used to determine stability margins is a frequency domain technique of Bode. This technique is numerically superior to approaches such as the Routh array approach, and is much faster than the direct approach of repeated eigenvalue determination.

As shown in Figure 134 Equivalent Stability Margin System, the parameter K for which the stability margin is to be calculated can be thought of as providing a single loop feedback around the system model.

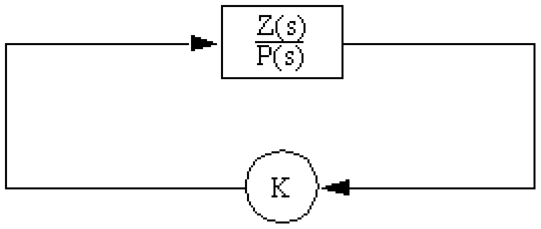


Figure 134 Equivalent Stability Margin System

Solving Equation 1 for the open loop transfer function in terms of K_n, N(s) and P(s), and substituting jω for s, yields:

$$K_0 = \frac{K_n}{R(j\omega)} \qquad \text{Equation 1}$$

where

$$R(j\omega) = 1 - \frac{N(j\omega)}{P(j\omega)} \qquad \text{Equation 2}$$

Thus, the method simplifies to a search for the frequencies that cause the phase of R(jω) to be 0^o regardless of the sign of K_n.

A range of $0 < \omega < \omega_{max}$ is searched to find those values of ω at which the phase of R(jω) is zero. At this frequency, ω0, the limiting value of K, K_0, can be calculated by:

$$K_0 = \frac{K_n}{\|R(j\omega_0)\|_2} \qquad \text{Equation 3}$$

Magnitudes of R(jω) > 1 result in lower K limits. Magnitudes of R(jω) < 1 determine upper K_o limits. The usual definition of the stability margin is the ratio of maximum K, to nominal K_n, and is obtained from Equation 3 to be:

$$\frac{K_0}{K_n} = \frac{K_n}{\|R(j\omega_0)\|_2} \qquad \text{Equation 4}$$

Search for Zero Phase

A range of ω from 0 to ω_{max} is searched for zero crossings of R(jω). ω_{max} is arbitrarily established as two times the magnitude of the largest eigenvalue of the nominal system. Zero frequency is included, since a real divergence is indicated by a zero phase of R(0). After ω = 0 has been checked, the search begins at some low frequency, ω_{min}. Since only phase angles near 0 are of interest, small angle approximations may be used for the phase of R(jω). This approach avoids time consuming trigonometric calculations.

Several different values of w may be found that cause the phase of R(jω) to go to zero. The lower stability limit is determined by the maximum value of $1/R(j\omega_0)$ which is less than 1. The upper stability limit is determined by the minimum value of $1/R(j\omega_0)$ which is greater than 1. Any remaining values of w which cause the phase of R(jω) to go to zero correspond to divergence frequencies which occur beyond the critical stability limits. Oscillations will occur at these frequencies if the parameter is increased beyond the critical stability bounds. If such values exist in the searched region, they will be printed out by the program as noncritical stability limits.

The characteristic equation of this system with the stability margin parameter equal to its nominal value, K_n, is

$$N(s) = P(s) - K_n Z(s) \qquad \text{Equation 5}$$

Note: The sign of the feedback is determined by the sign of K and is not assumed to be negative, as is often the case in textbooks.

The roots of N(s) are the eigenvalues of the nominal system, and the roots of P(s) are the eigenvalues of the system with K = 0.

To concentrate the analysis on the stability boundary of the complex plane, (the imaginary axis), set $s = j\omega$ in Equation 5.

Those real values of K, K_o which will cause $N(j\omega) = 0$, will produce roots of the characteristic equation on the imaginary axis of the complex plane.

Solving the above Equation 5 for such values of K results in:

$$0 = P(jw) - K_0 Z(jw) \qquad \text{Equation 6}$$

$$K_0 = P(jw)/(Z(jw)) \qquad \text{Equation 7}$$

Since you are interested in only real values of K_0 of , you need consider only those values of ω which cause the phase of $P(j\omega)/Z(j\omega)$ to equal 0^o or 180^o. Further, if the nominal parameter $K_n < 0$, only values of 180^o need be considered, and if the nominal parameter $K_n > 0$, only the values of w that produce 0o phase need be considered.

The approach to determining K_0 is as follows: the roots of N(s) and P(s) of Equation 5 are the eigenvalues of the nominal system and the eigenvalues with K=0, respectively, and are designated as:

```
    Ni i=1,2,3, ... ,n       eigenvalues for K=Kn
    Pi i=1,2,3, ... ,n       eigenvalues for K=0
```

Thus, N(s) and P(s) can be stated in terms of their roots as:

$$P(s) = \left(\prod_{i=1}^{n} (s - P_i)\right) \qquad \text{Equation 8}$$

$$N(s) = \left(\prod_{i=1}^{n} (s - N_i)\right) \qquad \text{Equation 9}$$

States

See Also: "Integration Methods"

"Switch States"

"States: Defining Values and Controls"

There are four types of state variables used in Easy5: continuous states, delay states, sample states (also called sample-hold states), and switch states. These states are used in many standard components and can be added to any User Code and Library components you write. The following sections describe each type of state.

Continuous States

Continuous states are model variables that result from integration of first-order, ordinary differential equations. Any time you model a continuous (non digital) part of a system, you will use continuous states. You can define continuous states in User Code and Library components.

Although continuous states are not truly continuous (that is, their behavior is being approximated on a digital computer), you can think of them as having values at all points in time, as shown in Figure 135. This example shows the value for continuous state X as a function of time.

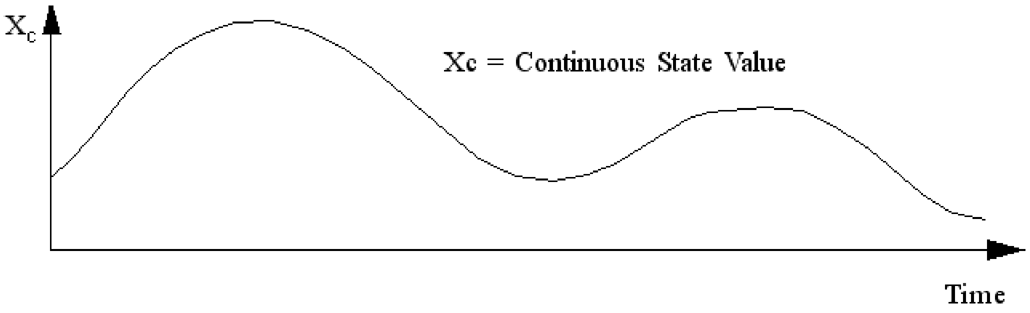


Figure 135 Continuous State as a Function of Time

Resetting Continuous States

Since the values of continuous states are determined by differential equations, they usually vary smoothly as time advances. There are rare situations in which the value of a continuous state needs to change instantaneously, usually when the actual physics being modeled has been radically simplified, or the continuous state is an approximation of a discrete process such as a digital filter.

Resetting continuous states in User Code components (and library components as well) is accomplished using the "`RESET STATE`" command, which has the following syntax:

```
    RESET STATE, continuous_state_name = value
```

Where continuous state name is the name of a continuous state (scalar, vector, or matrix) and value is either a Real*8 literal if the state is a scalar or an array of the same size as the state which has been loaded with the new values of the state components.

This command causes the Easy5 variable step integration algorithms to restart every time the "`RESET STATE`" command is executed and hence may cause performance problems if executed often for large models.

An example of a model using `RESET STATE` is a bouncing ball modeled by assuming that when the ball impacts the ground its velocity reverses sign and is reduced by a factor K_R called the coefficient of restitution. A Fortran component implementing this model would have continuous states height (initial condition = initial height), velocity (initial condition = 0), switch state netAcceleration (initial value = -32 for acceleration of gravity), and input parameters coef_restitution (a positive number less than one), and smallv (a small positive number such that if the rebound velocity is less than smallv, the ball stays at rest on the ground). The code for the model is as follows:

```
Derivative of, velocity = netAcceleration
Derivative of, height = velocity
If (height .LE. 0 .and. velocity .LT. 0 ) Then
     If ( Abs(velocity) .GT. smallv ) Then
        Reset State, velocity = -coef_restitution*velocity
     Else
        Reset State, velocity = ZERO
        Set Switch, netAcceleration = ZERO
     End If
        End If
```

Sample States

Sample states are one of the two types of discrete states used to model sampled-data systems; the other is delay states. When you use any Easy5 discrete standard components to model a digital part of a system, you will be using a sample state. You can define sample states in User Code and Library components.

A sample state is the output of a function of the form:

$$X_{t^+} = f(X_{t^-}, u, t)$$

where:

u is one or more input;

f is an algebraic function defined on the domain

t_o, t_o+t, t_o+2t, ..., t_o+nt and t^+ and t^- represent values

just after and just before sampling intervals.

The value of a sample state changes only at sample times. Easy5 monitors the passing of simulated time and updates the value of the sampling state only at the appropriate sampling interval, τ.

Consider the simplest case:

$$X_{t^+} = I_{t^-}$$

where I is some input signal. The value of the sample state will change only at sample times and will be the value of the input at that time as shown in Figure 136.

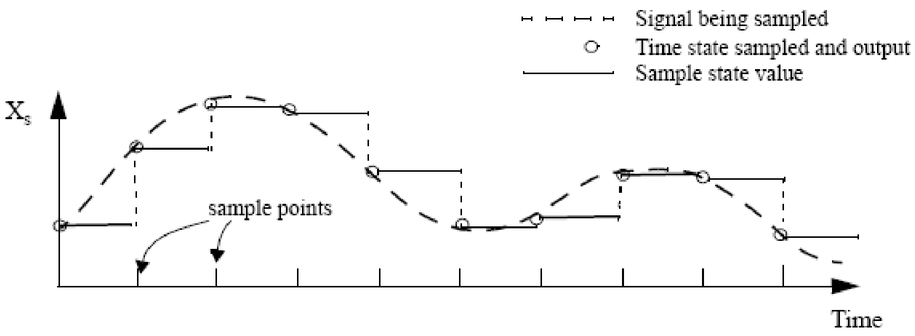


Figure 136 Sample State Values Held Constant for a Sample Period

Delay States

Delay states are one of the two types of discrete states used to model sampled-data systems; the other is sample states. When you use one of the Easy5 discrete standard components (except SH) to model a digital part of a system, you will be using a delay state. You can define delay states in User Code and Library components if you use discrete standard components.

The value of a delay state is the result of evaluating first-order difference equations of the form:

$$X_{t+\tau} = f(X_t, u, t)$$

where t is the sampling interval associated with the state and f is an algebraic function.

The domain of the function f is the sequence of discrete intervals of time:

$$t_o, t_o{+}t, t_o{+}2\tau, \ldots, t_o{+}n\tau$$

So the value of the delay state will change only at these discrete intervals. Easy5 monitors the passing of simulated time and updates your delay states only at the appropriate sample times.

Consider the simplest first-order difference equation:

$$X_{t+\tau} = I_t$$

where I_t is some input signal.

The value of the delay state will change only at sampling intervals and its value will be the value of the input signal t seconds before, as shown in Figure 137.

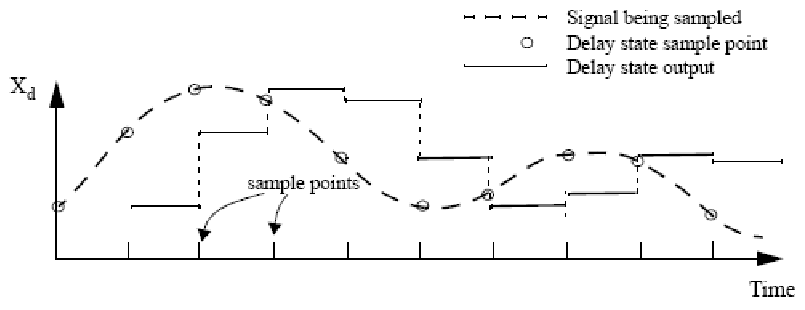


Figure 137 Delay States Change Only at Discrete Points

Switch States

Switch states allow you to efficiently use any of Easy5's variable-step integrators to model systems containing extremely discontinuous phenomena. In the past, discontinuous effects like hard limiting, coulomb/static friction, and hysteresis could be only approximated in your model and integrated with fixed-step integrators. For more information, see the major topic "Switch States"

States: Defining Values and Controls

All the states in your Easy5 model are stored internally in what is called the "state vector." This vector is as long as the total number of continuous, delay, sample, and switch states in your model.

The values given to states, or the elements of the state vector, at any point in time are determined by your model equations and are calculated by Easy5. For continuous states, your model calculates the time derivatives, which are then used by the numerical integration algorithms to calculate a new state value.

See Also:	"Data Types"
	"Operating Point"
	"States"

State Initial Conditions

Every state in the model has a corresponding initial condition value, which is defined in the data table of the component that produces the state. These values are used as the initial operating point values for all analyses, and define the point about which the linearization is performed for the set of linearized analyses. There are three methods for defining the initial operating point:

1. Loading the initial conditions manually.
2. Using the final set of state values from a simulation or steady-state analysis.
3. Calculating a set of initial conditions in a User Code component.

Initial conditions need only be set if they have nonzero values. You can assign new values by simply entering the new values under the “Value” column in the component data table. Values can be entered with a maximum of 9 significant digits and will be interpreted as double precision floating point numbers. The newly entered values will be saved as initial conditions after the “OK” push button is selected.

The topic “Operating Point” describes different methods of generating initial conditions and operating points.

Error Controls

A quantity known as an “error control” is associated with every state in your model. Error controls are used in two ways by the Easy5 program: to determine an integration step size control for variable step integration during simulation, and as a perturbation step size during linearized analyses. See the topic, “Guidelines for Setting Error Controls” for important information on error controls that can be used with simulation analysis.

Perturbation Step Size

During linearized analyses the error control is used to set the perturbation step size for each state. This step size is used when calculating the partial derivatives required to generate linear models of your nonlinear model about an operating point. During linearized analyses, the ideal scaled value for an error control (or step size) for a nonlinear system should be about one 10th of one percent of the state's nominal value. In most circumstances, the default value of 0.001 (and 1.e-6 for switch states) should provide you with satisfactory results.

Freezing States

Any state in your model can be rendered inactive or “frozen.” A *frozen* state is held at its initial condition value regardless of the value of its calculated rate in the model. This feature provides a convenient method for simplifying your model. You may wish to do this to temporarily isolate certain “loops” of your system, eliminate certain high frequency modes of your system, isolate the cause of instabilities, eliminate certain degrees of freedom of your system, or open feedback paths.

In order to determine if a state is frozen, examine the component's data table. Either “YES” or “NO” will be displayed under the column marked “Frozen.” To change this value, simply place the cursor over the appropriate value and press the left mouse button. The default value for all states is unfrozen (“NO”).

Steady-State Analysis

See Also:	"Analysis Data Form"
	"Temporary Settings File"

The Easy5 steady-state analysis is used to algebraically find an equilibrium point for your model. An equilibrium point is defined as a point where the root mean square value of all active system rates is near zero. Within Easy5 this is termed a “steady-state” operating point.

Achieving a steady-state operating point is a very important step in initializing your nonlinear model for two reasons: it assures you that results from linearized analyses can be used to predict stability, and it eliminates “start up” transients during simulation.

The method used to find a steady-state operating point is an algebraic one, and therefore is much less time consuming than performing a simulation to achieve the same result, a so-called “null transient.”

Guiding the Steady-State Analysis

A nonlinear model may contain several valid equilibrium points. When you execute a steady-state analysis, Easy5 searches for a steady-state point starting from the operating point defined by the current set of initial condition values.

By default, all initial conditions are given a value of zero. If you do not specify any initial condition values, Easy5 will start its search from this zero-IC point. For many physical quantities, a value of zero does not make much sense (for example, absolute temperature, pressure). Therefore, you should assist the steady-state finder by defining an initial operating point that is as “reasonable” as possible.

Model Validity Bounds

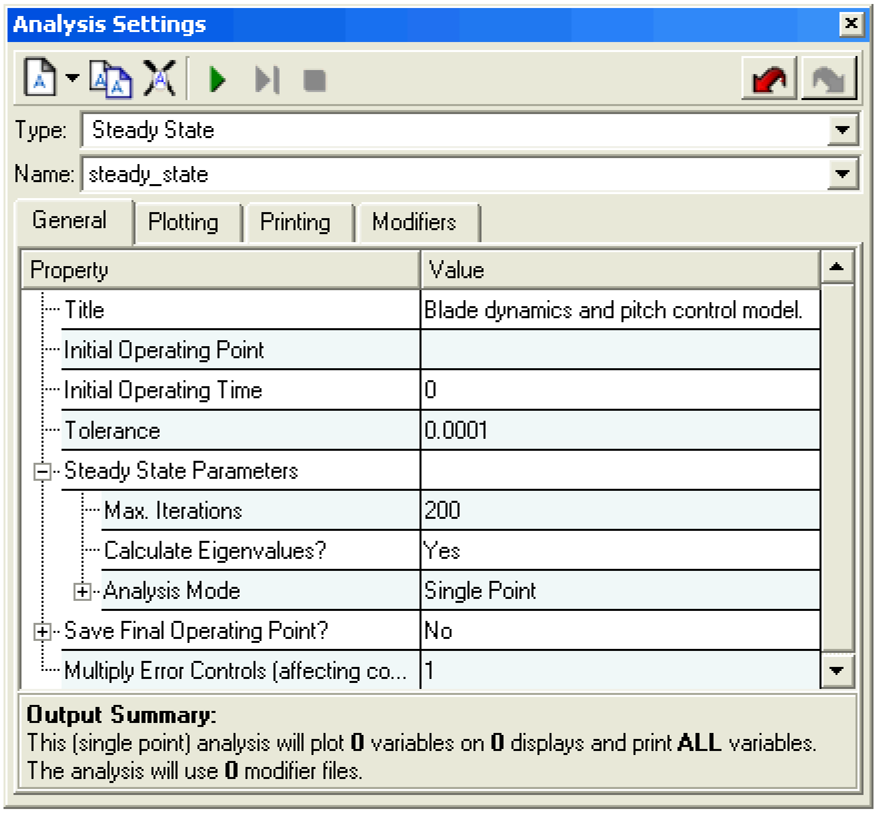
For highly nonlinear systems, finding a reasonable operating point can be especially important, but extra care should be exercised because the valid operating regime may be limited. For instance, a system containing states that describe pressures and energy terms may utilize a series of table look up data to acquire thermodynamic data. For such a system it is important to stay within the bounds of the table data to maintain model validity.

Types of Steady-State Analyses

Two types of steady-state analysis may be performed: a single point steady-state analysis, or a steady-state scan. A single point analysis is simply that it finds only one steady-state point. A scan, on the other hand, allows you to find a series of steady-state points while one of your system parameters, called the steady-state scan parameter, is changed through a specified range of values. Both types use your model's initial conditions as a starting point.

Setting up a Steady-State Analysis

All data for setting up a steady-state analysis is entered in the steady-state data form, shown in Figure 138. This figure shows all the options and data fields that could possibly appear on this form. The sections that follow describe how to fill in this form. To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to "Analysis Data Form".



Type: Steady State

Name: steady_state

Property	Value
Title	Blade dynamics and pitch control model.
Initial Operating Point	
Initial Operating Time	0
Tolerance	0.0001
Steady State Parameters	
Max. Iterations	200
Calculate Eigenvalues?	Yes
Analysis Mode	Single Point
Save Final Operating Point?	No
Multiply Error Controls (affecting co...	1

Output Summary:
This (single point) analysis will plot **0** variables on **0** displays and print **ALL** variables.
The analysis will use **0** modifier files.

Figure 138 Steady-State Data Form

Specifying the Maximum Number of Iterations

Easy5 uses an algebraic/iterative approach to locate a steady-state point. Since it is possible that there are no steady-state points in your model, or that Easy5 may be unable to locate one, a maximum number of iterations value must be specified so that the program knows when to stop searching. A default value of 200 “Maximum Number of Iterations” is provided. If you want to specify some other number, just select the data field following “Maximum Number of Iterations” and enter a new value.

Specifying the Tolerance

The tolerance is the root-mean-square (RMS) error convergence criteria. The default RMS error of 0.0001 is usually adequate, but sometimes a smaller error is required. You can set this field to a smaller error to increase the accuracy of the result. However, you should not set this field to a value larger than the deafult. Also, if you set this field to a very small value (that is, less than 1e-8) you may get a message that the analysis did not converge in your output listing. This may mean that the steady-state solver could not satisfy the requested RMS error.

Finding a Single Steady-State Point

To find a single steady-state point, select the value "**Single-Point**" for "**Analysis Mode**" on the data form with your left mouse button. Easy5 will find a single steady-state point when you execute the analysis.

Using the Steady-State Scan Option

If you want to activate the steady-state scan option, select the "**Scan**" value for the "**Analysis Mode**" on the data form with your left mouse button. Notice when you do this that new options and data fields appear on the data form. These additional options and fields are used to set up a steady-state Scan analysis, and are defined in the following paragraphs.

During a Steady-State Scan analysis, each incremental "starting point" (after the first one) will begin with the previously calculated steady-state operating point, thus minimizing the amount of work required for each "step" of the scan. Also, if you request that the steady-state operating point be saved, only the final point will be saved for a scan.

The steady-state scan parameter is the name of the quantity in your model that will be varied during the analysis. You enter the name of this quantity in the data field following "Steady State Parameter " on the data form. To specify the quantity, select the data field and enter the name, or use the "pick" method. Valid quantities include all system parameters, states that have been frozen, and the system variable, Time.

Another option is available for entering a scan parameter name. After selecting the parameter data field, select a component in the schematic. When you select a component with your left mouse button, Easy5 will display a list of all of the possible scan parameters in the Model Explorer window. If you select one of the names in this list, Easy5 will copy it for you into the scan parameter data field. Using this option will help you to avoid mistakes when entering scan parameter names.

Start Value is used to specify the starting value for the steady-state scan parameter. To enter a value, select the "Start Value" data field and enter a number.

Stop Value is used to specify the ending value for the steady-state scan parameter. To enter a value, select the "Stop Value" data field and enter a number.

The # of Scan Points data field defines the number of equally spaced values from the starting value to the ending value that the steady-state scan parameter will be given during the Steady-State Scan analysis. If you choose to create steady-state scan plots, this value will determine the number of data points that will appear on the plots.

The values of up to 8000 outputs can be plotted during a Steady-State Scan analysis. Outputs are usually plotted as a function of the steady-state scan parameter. You can plot (up to 2000) outputs on individual sets of axes, or you can "over-plot" up to four plots on one set of axes. When you want to plot steady-state scan data, you have to do two things: 1) tell Easy5 that you want plots generated and 2) specify the output variables to plot.

To tell Easy5 that you want steady-state scan plots, select "**Selected**" or "**All**" following "**Plot Variables**" in the Plotting tab of this form. To specify the output variables that you want to plot, use the "**Selected**" value. Then, simply add names to the list of Dependent Variables offered. This form is used similar to definition of simulation plotted variables as described in "Setting up Simulation Plots".

Printing Steady-State Results

By default, the values for all the states, rates, and variables in your model at the steady-state point(s) will be printed to the analysis output listing file. If you don't want any printed output, select “**None**” following “**Print Variables**” on the Printing tab of this data form.

If you just want to print selected variables during the analysis, you should select “Selected” following “Print Variables” on the Printing tab. When you do this, you will be able to specify Dependent Variables in a list below, in a similar fashion as plotted variables.

Saving a Steady-State Operating Point

If you want to save the operating point calculated during a single analysis, or the last value calculated during a Scan analysis, select “**Yes**” for the “**Save Final Operating Point?**” field in the General tab for this the data form. A new data field named “**Final Operating Point Name**” will appear on your form.

The name you enter in this field will be used to name the “saved” operating point. A list of these names will be presented to you when you use the **Options** > **Restore Operating Point...** menu option.

Note:	When you save the final operating point at the end of this analysis, the “Time” value specified in the data form will also be saved as part of the operating point. This means that if you restore an operating point that was saved at the end of a Steady-State analysis, each “Time” data field in every analysis data form will be updated with the saved “Time” value.

Steady-State Analysis Outputs

Steady-State analysis outputs can be in plotted and/or printed formats. These output formats are described in the following sections.

Steady-State Plots

If steady-state scan plots were generated during your analysis, you can look at them by selecting **Analysis** > **Plot Current Results** (Ctrl+Shift+P). For information on using the Easy5 Plotter, see the User Guide, Chapter 7 - Easy5 Plotter

The Steady-State analysis output listing may be examined by selecting **Analysis** > **Display Analysis Output Listing** (Ctrl+Shift+L). Once this file has been opened, search for the string “STEADY-STATE ANALYSIS” to point you to the start of the Steady-State analysis output printout.

Aside from the usual header information, the first item to notice is whether convergence was achieved. Easy5 will also print out the maximum number of iterations and the final root-mean-square value for all the active rates of your system.

Thereafter, depending on what you specified on the print specification data form, you will see the values that were obtained for all states in order to achieve convergence the solution. Values for output “VARIABLES” and “PARAMETER VALUES” are also included.

As a check of the asymptotic stability at the steady-state operating point, a linearization made about this point yields the system eigenvalues. In order to be asymptotically stable, no positive real parts should be present.

For a more complete linear analysis output, you should run a Linear Model Generation analysis at this new operating point.

Steady-State Analysis Method

See Also: "Steady-State Analysis"

The algorithm used by the Steady-State analysis is an algebraic manipulation of the system states based on a Newton-Raphson iterative approach using knowledge of the system Jacobian in converging to a solution. This method involves a solution of a set of nonlinear algebraic equations obtained by setting your model differential equations to zero. The solution process is repeated until either the norm of the rates becomes less than 0.0001, or the maximum number of iterations has been reached. Evaluation of model rates is limited to those corresponding to continuous states, and to continuous and delay states for sampled-data systems.

Delay states, however, are treated as continuous first-order lags during steady-state analysis. Sample states and switch states both drop out of the evaluation, and their rates are simply set to the value of their corresponding states during the analysis.

This process continuously checks whether any state in your model causes the Jacobian matrix to become singular. A singular matrix is a matrix with a zero determinant, which would manifest itself, for example, as a zero row or column in a given matrix. When such a matrix is inverted, as is required during the steady-state solution, the calculation fails. To prevent this, if a singular Jacobian is detected, the corresponding state will be frozen at its current value and the solution process will continue.

Finally, eigenvalues are calculated at the steady-state (solution) operating point to give you a measure of the asymptotic stability about this point.

A Singular Jacobian

A singular Jacobian matrix indicates either that a state in your model has no effect on the other states in your model (it is behaving like a “free” integrator), or that a state in your model is unaffected by all other states in your model (it is behaving like it is not part of your model). Sometimes, due to numerical inaccuracies, an ill-conditioned problem can also cause a singular Jacobian.

For example, if a state in your model described the position of a piston, and during the steady-state solution the position of the piston hit its upper limit, it would cause a singularity in your system Jacobian matrix. Any positive perturbation of the other states in your model would not affect the position, which is limited.

Also, if using tables which do not allow linear extrapolation, the steady-state finder may exceed a table boundary, get “clamped”, and cause a singular Jacobian.

The Determination of System Eigenvalues

To provide you with information about the asymptotic stability at the steady-state operating point, a linearization is performed about this point. This analysis only occurs if you have selected the “Calculate Eigenvalues” steady state analysis option. Easy5 automatically selects the correct form of linearization for either a continuous or a sampled-data system. From this linearization, the system eigenvalues (in either the s-plane or the z-plane) are calculated.

Steady-State Analysis Troubleshooting

Certain approaches have proven beneficial to achieving a good steady_state operating point in a timely manner. These approaches include: a piece-wise solution, convergence, and reasonableness.

A Piece-Wise Solution

For relatively small models (with fewer than 25 states) exhibiting no enormous non linearities, it is usually possible to find a steady-state operating point with just one steady-state analysis. For larger models it is often better to approach the problem in stages: first finding a steady-state point for one part of your model by freezing states associated with the rest of your model, saving that operating point, and unfreezing the states; then, starting with the saved operating point, performing another steady-state analysis for the whole model. Often, this piece-wise approach is much more successful for large nonlinear models than a "brute-force" approach.

Poor Convergence

If a model does not achieve convergence within the maximum number of iterations, it is usually an indication of a modeling inconsistency. For example, it could be an indication that some conservation law is being violated (such as Conservation of Mass, Energy, or Kirchhoff's laws). You can increase the maximum number of allowable iterations, but if the model has not converged within, say, 200 iterations (twice the default value of 100), it probably will not help to increase the maximum number of iterations further. Instead, you should check your model equations carefully.

Check for Reasonableness

Carefully check the values of all variables and states after a steady-state analysis to ensure that you are within your valid operating regime. For example, it may be feasible to end up with an operating point where the temperature or pressure is negative, or the steady-state angle-of-attack (for airplane simulations) is too large. If an operating point is determined invalid, you may have to obtain a better starting point and try again.

Figure 139 illustrates the importance of having a reasonable starting point for finding a valid steady-state.

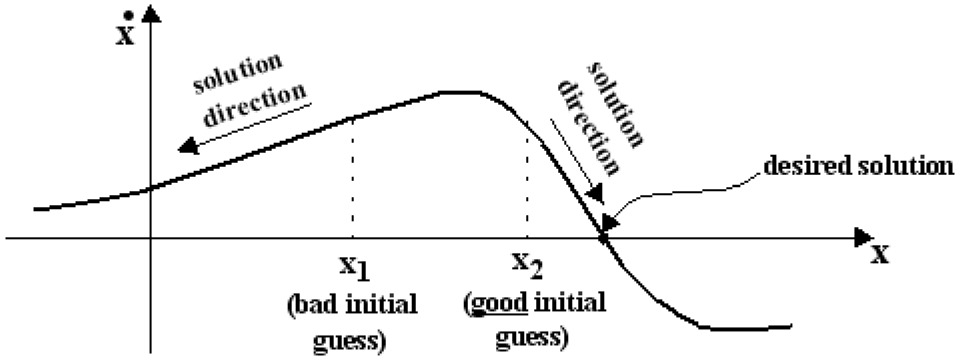


Figure 139 Steady State Analysis Requires a Reasonable Starting Point

Steady-State May Fail to Converge

Steady-State analyses may fail to converge to a solution for many reasons including the following:

- A steady-state point is too far away from the initial operating point and the algorithm fails
- No steady-state point(s) exists
- Nonlinear models have not been properly modeled with switch states and the algorithm fails
- The “maximum number of iterations” variable is exceeded
- The problem is too complex and the algorithm simply fails

Steady-State analyses may fail to locate the correct (the one you want) steady-state point because the algorithms converges to a steady-state point closer to its starting point. In addition to this, the steady-state analysis may locate a steady-state point, but the steady-state point is unstable. To guard against this, Easy5 also calculates the eigenvalues at the respective point and they should be checked to verify stability.

Overcoming Steady-State Non-Convergence

If the steady-state fails to converge due to any of the reasons cited above, try the following suggestions.

Increase Number of Iterations

By default, the number of steady-state iterations is set to 200. Increase this to 300, and if it still fails try 400. In general, if the steady-state doesn’t converge within 400 iterations, then increasing the number further will likely not help.

Try a New Initial Condition Operation Point

A poorly defined initial condition operating point is one of the primary reasons for steady-state failure. *You are responsible for setting a proper initial condition!* By default, all state initial conditions are zero. It is obvious that a state that calculates absolute temperature should not have zero as an initial condition. A temperature of absolute zero is not a valid operating point and in most cases will cause the steady-state algorithm to fail. Try setting different state initial condition values. For information on this, see “Operating Point”.

Try performing a short simulation and save the final operating point. This may allow the transients to start decreasing. Then re-load the saved operating point and try the steady-state again.

Generate the Jacobian Matrix and Eigenvalues

Execute a Linear Model Generation to obtain the Jacobian matrix and the eigenvalues. Please ensure that eigenvalues have been selected in the **“Calculate Eigenvalues?”** setting. (There is no need to define linear model inputs/outputs). Examine the output of the linear model analysis. Jacobian large numeric values indicates poor scaling or an ill posed system. Re-do the model units. Large positive eigenvalues indicates incorrect data or bad connections.

Decrease Error Controls

The state’s error control is used as a perturbation step size during the steady-state search. Decreasing the error control may help some convergence difficulties. The error controls may be decreased on a state-by-state basis.

However, it is easier to perform a global change. To do this, simply set the "**Multiply Error Controls (affecting continuous states)**" setting to a value less than 1, say in "decades", or subsequent values of powers of 10. For example, to reduce the global error control by a factor of 10, set the value to 0.1.

Print Output at Each Iteration

You can print out the steady-state data at each iteration point during the steady-state search. This should only be used for diagnostic purposes to help determine the sources of steady-state convergence problems. To do this, create an auxiliary input file and enter the following single command:

```
PRINT CONTROL = x
```

where x is the print control setting number:

x = 6	prints all states, rates, and variables at each iteration
x = 7	prints the Jacobian matrix and all states, rates, and variables

Stop and Exit Flags

See Also: "Termination Commands"
"Fortran Component"
"C Component"

There are several ways to stop an analysis and exit Easy5. Depending on how you need to stop an analysis, there are several flags and calls to routines you can use.

Terminating Using the ISTOP Flag

A global variable named **ISTOP** is provided for you to control termination of a particular analysis, usually a simulation. You should do this in the form of a conditional block of code in a Fortran, C, or library component. This block of code should detect the occurrence of a terminating event (at a valid checkpoint such as ITINC) and then set the variable **ISTOP** to the appropriate value as described in the table below. This is often useful when validity checks are placed in your model.

Value	Description
ISTOP =1	User-requested termination of a simulation, where the termination should be considered a "normal" termination. This method of controlling a simulation is typically used when testing on a certain event occurring. Subsequent analyses, if defined, will be executed.
ISTOP =2	User-requested termination of a simulation, with indication of an error condition to the GUI. This will cause the simulation to stop, and the Analysis Program Listing to automatically pop up. Typically, this occurs from validity checks from the application libraries - flagged via a "FATAL ERROR" message. Subsequent analyses, if defined, will be executed.
ISTOP =3	Easy5-generated termination due to a failure in the numerical integration (matrix?) Users should not set this value for ISTOP.
ISTOP=4	Immediate termination of a particular analysis. No subsequent analyses will be executed. This is useful when an invalid model condition is detected, and further execution of any analysis would be meaningless.

Caution:	Do not attempt to stop the program by inserting a **RETURN**, **STOP**, or **END** statement in your Fortran or library component code. This will result in loss of all or most of your printed and plotted data!

Submodels

A submodel is a group of components that when grouped together, form a subset of the model. Easy5 uses submodels to create hierarchical layers of the block diagram, in essence, creating a three-dimensional display of the model's block diagram. The ability to create hierarchical layers of a system block diagram is useful in two ways. First, it allows the user to divide the system model into submodels that logically model subsystems. And second, submodels reduce the complexity of large block diagram schematics.

Submodel features are best shown using an example. Figure 140 shows an example of how submodels are used in Easy5. The upper schematic shows the Easy5 schematic block diagram developed to model a two axis rocket nozzle control system. This block diagram is annotated for instructional purpose, to better describe the subsystems.

For example, the components defining the rigid body dynamics are encapsulated with a cross-hatched lines and labeled "Rigid body dynamics." Two additional subsystems are defined as "Flexible body dynamics", and "Control system". This model has been converted into an equivalent schematic block diagram using submodels as shown in the lower schematic of Figure 140. Notice how each subsystem has been made into a submodel.

Hierarchical Layers

Submodels are used to create hierarchical layers. This feature is best described by examining the "control system" submodel. Notice in the upper schematic block diagram of Figure 140, that the control system box contains two digital compensator components that are also encapsulated with a box defined as “digital compensator.” These two components can be defined as a second submodel within the “control system” submodel. This results in multiple hierarchical submodel layers as shown in Figure 141.

The top layer contains the main schematic block diagram. This can be thought of as the “root” layer from which all other submodels branch-out into lower layers. Opening the “control system” submodel moves the user down one layer in the block diagram hierarchy. Opening the “digital compensator” submodel from this first layer will move the user down one additional layer into the lowest hierarchical layer.

Submodels are merely a graphical feature used to facilitate in displaying the model’s block diagram. The use of submodels does not alter the nature of the Easy5 components; that is, component connections and data are not affected when using submodels. Unlimited layers of submodels may be created and manipulated.

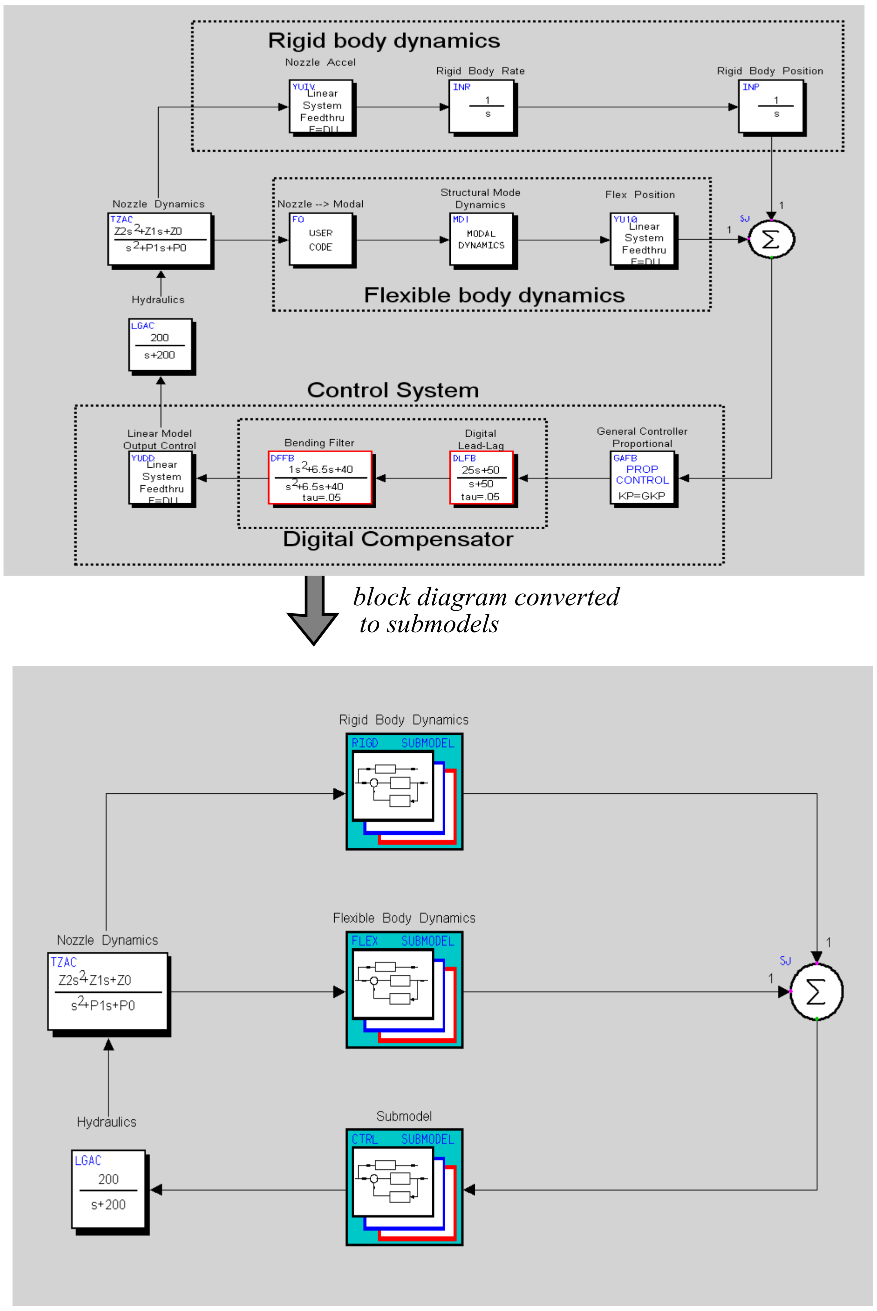


Figure 140 Example of Using Easy5 Submodels

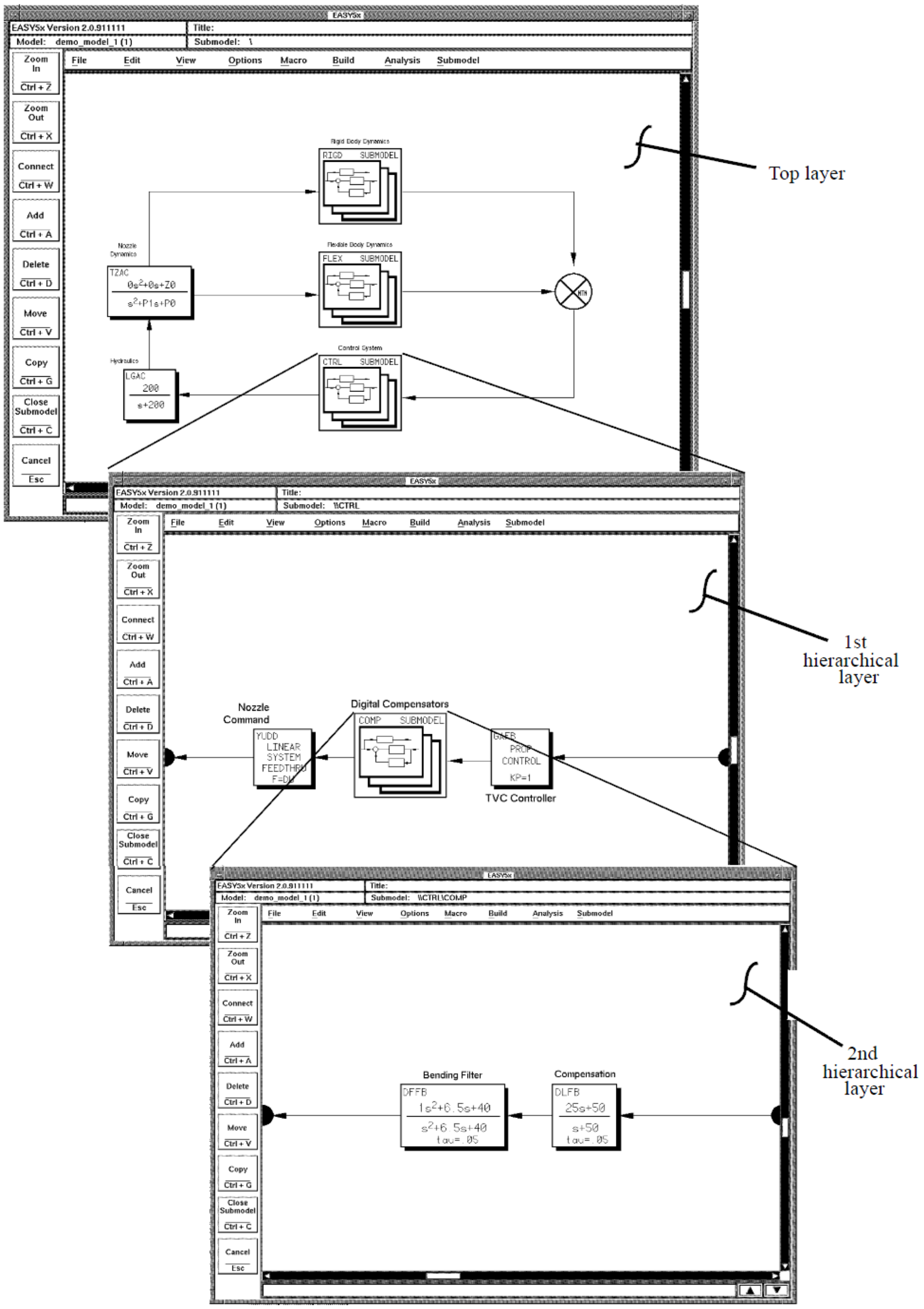


Figure 141 Example showing 2 Hierarchical Submodel Layers

In the next section, the submodel menus and features will be described showing how the user can easily move up, down, and across the hierarchical layers by opening and closing submodels.

Submodel Menu

The Submodel menu is used to create, edit, and in general, explore and manipulate submodels.

To open this menu, select the **Submodel** menu from the window menu bar. The Submodel menu is shown in Figure 142.

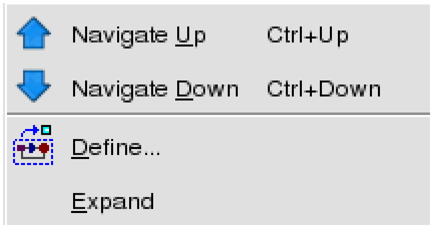


Figure 142 Submodel Menu

The options given in the submodel menu are explained in greater detail in the following sections.

Defining a Submodel

To create (or define) a new submodel, select **Define..**(Ctrl+H) from the **Submodel** menu. One or more components must be defined as a submodel. The user defines which components are to be included in the submodel by using a selection box to capture the desired component(s). Easy5 provides instructions in the message line telling the user which steps to take.

The selected submodel components become highlighted and a dialog opens to enter a submodel name. Submodel components have the same naming requirements as other components - that is, the name must be defined with four or less characters. Enter a submodel name that appropriately describes the submodel, and select the OK button to close the dialog.

Note:	The selected submodel components are highlighted. If the submodel components are incorrectly chosen, then select the "CANCEL" push button from the dialog box, and re-define the submodel.

The selected components will "collapse" into a single submodel icon. The default submodel icon is shown in Figure 143. The submodel descriptor displays above the icon and is given the default description of "Submodel." The editing of the submodel name and descriptor is described in "Submodel Labels".

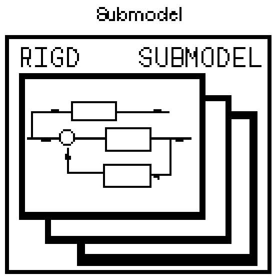


Figure 143 Default Submodel Icon

Opening and Closing a Submodel

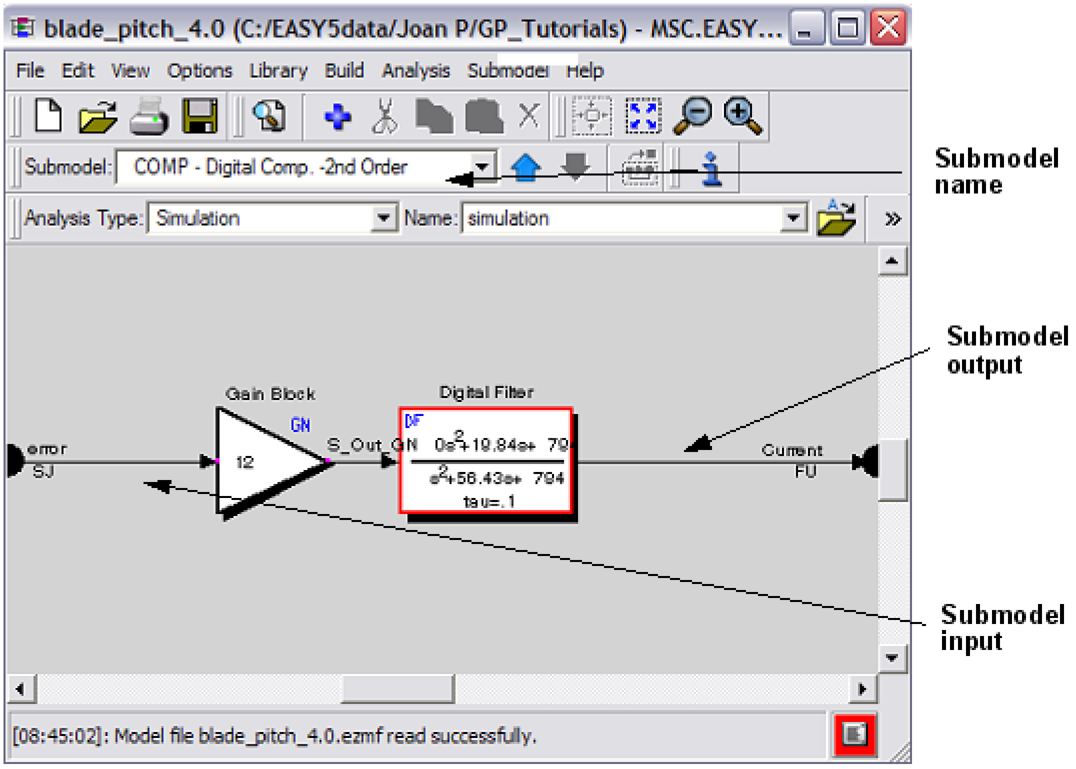
The submodel must be “opened” to examine the contents of the submodel. Opening a submodel is similar to examining any Easy5 component. Just select the submodel with the examine mouse key (DOUBLE-CLICK-L), and the components within the submodel will appear. This action jumps the user down one hierarchical layer into the next submodel layer.

Note:	Opening a submodel with a CLICK-C only moves the user down one hierarchical layer. The user can open any submodel from any hierarchical layer by selecting Open from the Submodel menu. This feature is described in Submodels, "Opening and Closing a Submodel".

An example of opening a submodel is shown, in Figure 144.

This submodel was opened by selecting the RIGD submodel icon as previously shown in Figure 143: Default Submodel Icon with a CLICK-C. The description area defines this to be a submodel schematic with the submodel name: "RIGD".

This submodel is defined with the three components shown in Figure 144.


blade_pitch_4.0 (C:/EASY5data/Joan P/GP_Tutorials) - MSC.EASY...
File Edit View Options Library Build Analysis Submodel Help
Submodel: COMP - Digital Comp. -2nd Order
Analysis Type: Simulation
Name: simulation
Gain Block
GN
12
Digital Filter
DF
error
SJ
S_Out_GN
Current
FU
tau=.1
Submodel name
Submodel output
Submodel input

[08:45:02]: Model file blade_pitch_4.0.ezmf read successfully.

Figure 144 Example Submodel Schematic

The connections into and out of the submodel are shown with the filled-in half circles. The semi-circles show how the submodel is connected to the model. Each semi-circle represents a component, and as such, is labeled with the component name. Components may be added to this submodel and all Easy5 model building features may be used. Additional submodels may also be created, spawning lower hierarchical levels.

Closing a submodel is performed by any one of the following three methods. The **Close...** (Ctrl+C) option may be selected from the **Submodel** menu, or the accelerator keys Ctrl+C may be used. In addition, the *Close Submodel* push button can be selected from the control panel. When the submodel is closed, Easy5 automatically moves the user up one hierarchical layer.

Submodel Labels

Semi-circles are used in the submodel schematic diagram to represent connections into/out-of the submodel. These semi-circles are automatically labeled with the component name that the semi-circle represents. For example in Figure 144, the semi-circle on the left border is labeled with the component name TZAC. This shows that the connection into the submodel's YUIV component is from the TZAC component that resides outside this submodel.

The output of the submodel in this example is represented by the semi-circle on the right-hand border. The INP component output is connected to the semi-circles labeled as MTM, which shows that INP is connected to the MTM component which resides outside the submodel.

You can turn the submodel label feature on or off by toggling the following menu

View > Hide Submodel Labels

Submodel Connection Lines

Submodel connection lines and connection nodes may be manually routed. Information on submodel connections is given in "Moving Connection Line Endpoints".

Editing Submodel Properties

The submodel name, descriptor and icon attributes may be modified. To edit a submodel's attributes, CLICK-RIGHT-HOLD on the particular submodel component in the model schematic, and select **Properties...** from the **Submodel Component Menu**.

In the following example, the "Rigid Body Dynamics" submodel, component RIGD, will be selected using the CLICK-RIGHT-HOLD (Submodel Component) menu. The "<name> Submodel Attributes" dialog will pop up for the selected submodel. This dialog box is used to edit the name, description and the submodel icon.

The submodel name is defined when the submodel is first created. The name may be changed by editing the "**Name**" input field. The name is used to define the submodel, and is used when copying and deleting submodels. This name also displays in the submodel's schematic title.

The submodel descriptor is similar to the component descriptor, and is used to place a title over the submodel icon. The default descriptor is "Submodel", as shown in Figure 145.

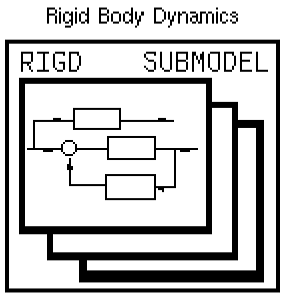


Figure 145 Edited Submodel Descriptor

The descriptor may be changed by editing the "**Description**" input field.

In this example, the description is changed to "Rigid Body Dynamics." The resulting submodel icon will appear with the description above the icon as shown in Figure 145.

You are allowed to edit the default submodel icon by selecting the Edit Icon push button, which initiates the Icon Editor. For instructions on using the icon editor, see the User Guide, Chapter 9 - Icon Editor. When finished with the "Edit Submodel Attributes" dialog box, select "OK" to apply the changes.

Expanding a Submodel

A user can take a submodel that has been created, and "expand" this submodel back to its original configuration. This is called "expanding" a submodel. This function expands the submodel and places the components of the submodel back into the block diagram from which the submodel was created.

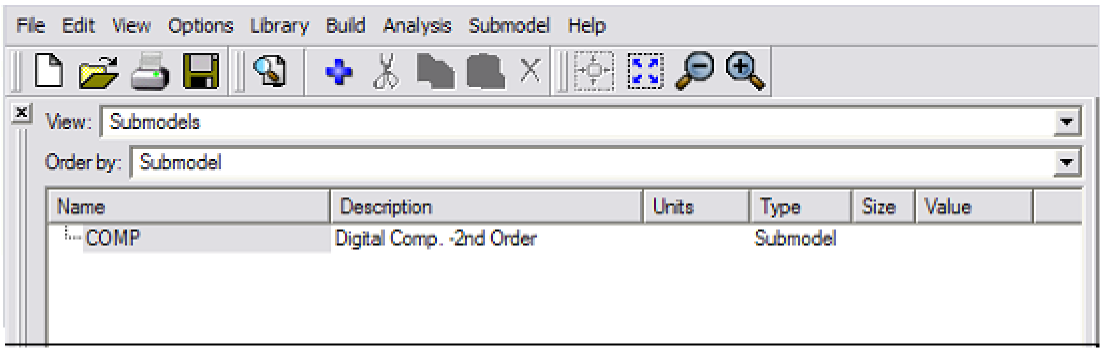
Caution:	Expanding a submodel completely eliminates the submodel from the schematic block diagram. The submodel expansion cannot be undone. If, after expanding a submodel it is needed, you will need to re-create that submodel.

Expanding a submodel back to its original configuration is performed by selecting the **Expand...** option from the **Submodel** menu, then selecting the submodel that is to be expanded. Alternatively, you can select the submodel component to expand via a CLICK-RIGHT-HOLD, and select the **Expand Submodel** menu item from the **Submodel Component Menu**. Easy5 will prompt the user with a dialog box to confirm that the selection is correct. The submodel will be expanded and brought up one hierarchical level. The schematic diagram should now appear as the original block diagram prior to building the submodel.

Navigating Submodels

Large models may contain many hierarchical layers of submodels. These layers may be connected across any layer. For example, the output from a component on the top layer schematic may be input to a submodel that is buried five layers deep; or an output from a submodel eight layers down can be connected to the input of a submodel only one layer down. As a result, opening and closing submodels one layer at a time can become cumbersome.

The most efficient way to display all submodels and jump to any submodel is to use the submodel explore function. To do this, select **Explore Model** from the View menu (or the "Explore Model" edit toolbar icon), and then select Submodels in the View: field to display a list of current submodels as shown in Figure 146.



Name	Description	Units	Type	Size	Value
COMP	Digital Comp. -2nd Order		Submodel		

Figure 146 Example of Model Explorer Submodel Hierarchy

This list of all submodels first gives the submodel name followed by its description. The submodel level currently displayed in the model schematic is also highlighted.

The Model Explorer can also be used to navigate among the various hierarchical submodel layers. Double left-click the submodel name to open and view all the components and inputs and outputs in this submodel. This feature lets you easily move in and out of the submodels without having to step through the hierarchical layers, one at a time.

To locate a particular submodel, simply select it in the list, and the schematic will be navigated to show the highlighted submodel component. This works in either direction. In this manner, the Model Explorer window (or schematic window) can be used to track where you are in the model schematic at any point.

Switch States

See Also: "States"

"States: Defining Values and Controls"

"Editing Submodel Properties"

Switch states allow you to efficiently use any of Easy5's variable-step integrators to model systems containing extremely discontinuous phenomena. In the past, discontinuous effects like hard limiting, coulomb/static friction, and hysteresis could be only approximated in your model and integrated with fixed-step integrators. The switch state facilitates simulation that is both more accurate and more efficient. For example, a model containing a mechanical actuator with coulomb friction was converted to a switch-state model and integrated in one-tenth the time previously required.

Several Easy5 components in the General Purpose Component Library that model common discontinuous effects with switch states. A partial list of these includes the following:

- AC - Position and Rate Limited Lag
- IH - Integrator with Hard Limits
- IL - Position and Rate Limited Integrator
- I2 - Double Integrator, Acc- Rate- Pos Limited

- CF - Velocity of Object with Coulomb + Static Friction
- DB - Deadband (Bang-Bang) Controller
- F2 - Two Bodies with Friction
- HY - Hysteresis
- NZ - Deadzone

In addition, you may add switch states to your own User Code and Library components.

What Switch States Represent

In some cases, a switch state represents a simplified model of a device that toggles from one setting to another and has some reluctance to switch back. A good example is a deadband controller like a thermostat. A furnace thermostat turns on when the sensed temperature falls below the set-point temperature but does not turn off again until the temperature is somewhat above the set point.

A second category of effects requires switch states to be modeled accurately. These effects use several sets of equations, and the selection of the appropriate set is based on some condition in the model.

For example, there are situations in which the set of equations to be used in determining the derivative of a continuous state depends on the value of that same state. The following are the equations for an integrator with hard limits:

dx/dt = input if x is within limits or input has opposite sign of limit

dx/dt = 0 otherwise

A switch state can indicate which set of equations should be used to determine the derivative. The switch-state value will be changed by the Easy5 integration algorithm when the algorithm has determined that the equation set to be used should change. The integrators are told when to change by the executable, which sets the rate of the switch state. The switch-state rate is the value the switch state should have during the next call to the model.

The changing of equation sets based on the value of one or more states in the model is often described as a "state-related event". Notice the importance of changing equation sets at the correct instant: if you delay, x will go beyond the limit. In this usage, the switch state represents a nonphysical quantity used to control some logic in your model.

A third category of effects that can be modeling with switch state are discrete events that occur at irregular but predetermined times. Switch states are used for these effects primarily to ensure that the Easy5 variable step integration algorithms do not simply "jump over" these events as the integration algorithms moves ahead in time in discrete steps. While such "timed events" can be modeled using normal switch states and logic based on the Easy5 variable TIME, doing so is somewhat inefficient because the normal switch state processing involves searching for the time of an event. To improve efficiency, Easy5 provides a special type of switch state called "timed event switch states which are described later in this article.

The Advantages of Switch States

Switch states are convenient and efficient for applications like deadband controllers, but are necessities for models involving state related events. This is because Easy5 simulation uses numerical integration and is subject to all the limitations thereof.

The most significant of these limitations is that numerical integration can only evaluate the model equations at a finite number of instants of time among the infinite number that occur in any time interval. The solution between those times is obtained by some form of interpolation and/or extrapolation.

To illustrate, consider the equations for the acceleration of an object subject to coulomb friction:

accel = 0 if velocity = 0 and external force < coulomb friction force Equation 1

Otherwise,

*accel = [external force - coulomb force * sign(velocity)]/mass* Equation 2

where sign(velocity) is +1 if velocity/ 0 and -1 if velocity <0.

Figure 147 shows the correct transient response of such an object under the initial condition that at time=0, the velocity is .951.

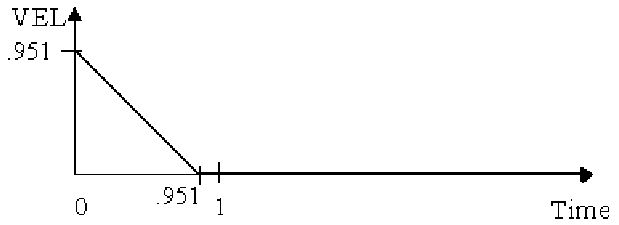


Figure 147 Transient Response of Moving Object with Friction

In the above example, there is no external force, and the mass and coulomb friction force are unity. Note that the object decelerates at a constant rate according to Equation 2 until the velocity is zero at time=.951. After this, the velocity stays at zero under Equation 1.

Figure 148 :Effect of Euler Integration of Transient shows the effect of integrating this transient using the simple Euler integration algorithm without a switch state.

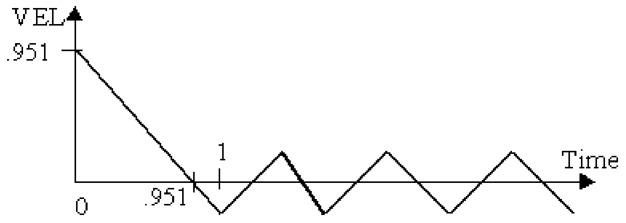


Figure 148 Effect of Euler Integration of Transient

The Euler integration algorithm can be expressed as follows:

$$x(t+\Delta t) = x(t) + dx(t)/dt * \Delta t \qquad \text{Equation 3}$$

This can be written in terms of velocity and acceleration as:

$$vel(t+\Delta t) = vel(t) + accel(t) * \Delta t \qquad \text{Equation 4}$$

where $\Delta t = .25$.

Using Equation 4 and the fact that accel(t)=-1 as long as the velocity is positive, vel(.75)=.201. Repeating this calculation for the next time step, accel(.75)=-1 since vel(.75)>0, so vel(1.0)=vel(.75)+(-1) * .25 = -.049. Note that the equations were never evaluated at time=.951, which was the point at which a switch should have been made to the other equation set.

To further complicate matters, continuing the integration another step gives accel(1.)=+1, since vel(1.)<0, so vel (1.25)=vel(1.)+(+1) * .25 = .201. This is exactly vel(.75); hence the cycle repeats as shown. Notice that if the fourth step had been of length $\Delta t = .201$, the integrator would have switched to Equation 1 and obtained the correct solution.

Using the Euler algorithm, a microscopic step size Δt would have to be used to integrate this transient with acceptable accuracy for an arbitrary initial condition. Unfortunately this difficulty cannot be avoided by switching to a different integration algorithm. Variable step integration algorithms assume that the derivative equations can be written as continuous functions of the state variable. They require continuity because they need to be able to probe back and forth in time to test and, if necessary, improve the accuracy of the solution.

In Figure 149, the acceleration of the coulomb friction example is plotted as a function of the velocity and is clearly discontinuous.

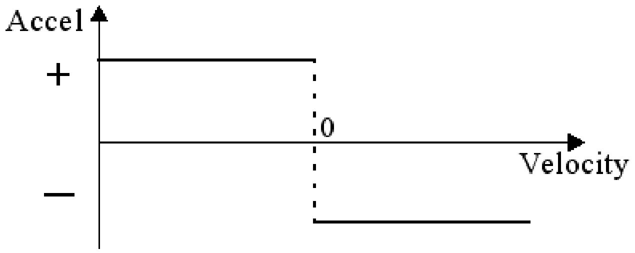


Figure 149 Acceleration as a Function of Velocity for Coulomb Friction

The method Easy5 uses for integrating this type of problem can be summarized as follows:

1. Easy5 has the model set a flag, the switch state's rate, when a state related event has occurred.
2. The integration algorithm searches for the time t (within tolerance) when the event occurred.
3. The integration algorithm completes the abbreviated integration step up to just before, while remembering which equation set to use after.
4. The integration algorithm tells the model to switch to the new equation set by setting the new state value, and then takes a new step of length W.
5. The integration algorithm continues with the transient.

There are two observations to make about the previous technique.

- First, notice that in step 3 there is a need for some memory; the rate of the switch state is used for this.
- Second, a transition step size needs to be specified; the **ERROR CONTROL** supplied with every Easy5 state, but not needed for the usual functions of switch states, is used for this purpose. You must set switch state **ERROR CONTROLS** on the basis of model requirements. A deadband controller will probably operate with sufficient accuracy using the default error control of .001, but other switch states will require a much tighter error control for accurate results.

Using Switch-State Standard Components

For the most part, you can ignore the fact that the component contains switch states. Switch states usually are internal to the component and not connected to other components. They can be printed and plotted just like any state or variable. Like all states, switch states are given a default initial value of zero. The initial value may be set just like any state initial condition.

There following sections describe a few instances when switch states will need attention:

Poor Performance

If simulation results indicate that the switch state is not switching close enough to the correct time, performance can be improved by decreasing the transition interval duration. This can be set by decreasing the error-control value for the switch state.

Reducing the error control will usually increase run time, so you should use the largest value that gives acceptable results. Contrary to other types of states, the default value of the error control for a switch state is 1.0E-6.

Freezing Switch States

For diagnostic or other reasons, you may wish to force a switch state to stay at its value. This can be done in the component data table, by changing the state's "Frozen" flag indicator form "No" to "Yes".

When Switch States are Active

At the present time, switch states are only active during nonlinear simulation and steady-state analysis. During other analyses, they are essentially frozen. This may change in future versions of Easy5.

Understanding the Rate of a Switch State

Since a switch state does not have a derivative, the "rate" of a switch state is used as "temporary memory" and represents the next value of the switch state (the value it will have as soon as the integration algorithm is ready to switch it). During an analysis, you can determine if the model would like to switch by checking whether the state and rate values of each switch state match.

Using Switch States in User Code and Library Components

An Easy5 model generation command, `SET SWITCH`, allows User Code and Library components to contain switch states. The syntax of this command is as follows:

```
SET SWITCH, switch state name = new value
```

where ***new value*** is the name of a User Code or Library Component variable whose value is the desired value of the switch state. Notice that, like every other state variable, the model (the component equations) sets the **rate** value of a switch state, but never the value of the switch state itself.

Example of Using Switch States

As an example of a switch state, consider modeling a deadband controller as a Fortran component that is functionally equivalent to the Easy5 standard component DB. Before writing the code that uses switch state logic, you should always develop a state transition diagram. These diagrams are ideal for laying out the logic necessary to make switch states work. Once the state transition diagram is complete, it is a simple matter to code the logic.

A state transition diagram for the deadband controller is shown in Figure 150.

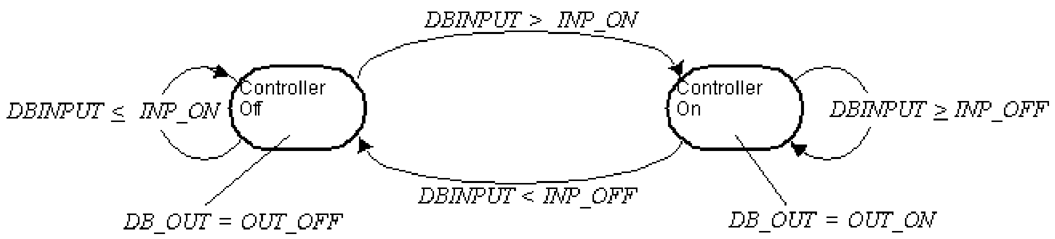


Figure 150 State Transition Diagram For A Deadband Controller

Figure 151 shows the code that would be used to code a deadband controller using a Fortran component that is functionally equivalent to the Easy5 standard component DB.

```
**** Fortran Code of Dead-Band Controller Using Switch States
   IF   (SW .EQ. ZERO) THEN
        DB_OUT= OUT_OFF
        SWDOT= 0
        DB_Current_Value= (DBINPUT -INP_ON) * (INP_ON -INP_OFF)
        IF (DB_Current_Value .GT. ZERO) SWDOT=1
   ELSE
        DB_OUT= OUT_ON
        SWDOT=1
        DB_Current_Value= (DBINPUT -INP_OFF) * (INP_OFF -INP_ON)
        IF (DB_Current_Value .GT. ZERO) SWDOT=0
   ENDIF
*
SET SWITCH, SW =SWDOT
```

Figure 151 Sample Fortran Code For A Deadband Controller

In this Fortran component, the inputs used are:	
DBINPUT	- the input signal
INP_ON	- the value of the input DBINPUT at which the controller should turn on
INP_OFF	- the value of the input DBINPUT at which the controller should turn off
OUT_ON	- the value of the output signal when the controller is on
OUT_OFF	- the value of the output signal when the controller is off
The outputs are:	
DB_OUT	- the output signal
SW	- the controller switch state (0=off, 1=on)

Note that the switch-state rate **SWDOT** is assigned a value on every pass through the code even if the setting of the switch state is not changing. This is absolutely necessary for the integrator to accurately locate the time at which the setting of a switch state should be changed.

Figure 152 shows an example of a non physical usage of a switch state. An integrator with hard limits is modeled as in Easy5 component IH. The code for component IH seems complex (N=0 configuration shown only). This complexity is introduced to avoid ambiguities in the switching logic.

```
     BEGIN SORT BLOCK
           If ( linear .and. lf_ih  .NE. ZERO) Then
              s_LimOut_ih  = s_out_ih
              If (incall .GT. 0) write(iwarn,+++10) '_IH'
     +++10    Format(/5x,'*** NOTICE ***   Component ',A,' is ignoring ',
          &     'its inherent nonlinear behavior for this analysis'/T24,
          &     '(per user-request via non-zero parameter LF).'/)
           Else
             If ( sw_ih  .EQ. ZERO ) Then
                s_LimOut_ih  =
          &           Max( Min(s_out_ih ,max_ih ),min_ih )
             Else If ( sw_ih  .EQ. ONE ) Then
                s_LimOut_ih  = max_ih
             Else
                s_LimOut_ih  = min_ih
             End If
           End If
```

Figure 152 Library Component Implementation of IH Component

Switching logic ambiguities occur when the logic switches back and forth between two values of a switch state. These ambiguities are complicated by the fact that, even with switch states, we are still dealing with numerical approximation. For example, using an IH component with a tight error control will result in extremely close approximation to perfect limiting, but not exact limiting, of the continuous state S_Out.

A setting of 25.0 for the upper limit, with an error control of 1.0E-8, may indicate perfect limiting on the printout, while internally the limit may actually be 25.000001. Suppose the Component IH had been built using the following logic:

If SW = 0 and S_Out Š max, then next SW = 1

If SW = 1 and S_In < 0, then next SW = 0 (assume GKI=1)

This is identical to the IH logic except that the first condition is missing the requirement that S_In Š 0. Now suppose that you have simulated the modified component to a point where SW=0 and S_Out is just slightly larger than the maximum (say S_Out = max+1.0E-10).

The first condition will force the integration algorithm to change the state to 1. S_Out will stay limited until the input turns negative. Suppose that S_In turns very slightly negative (say S_In=-1.0E-20). This is enough to cause the integrator to switch back to SW=0. Each of these steps will be taken at a step size equal to the switch-state error control. Since this is generally a very small number, you are effectively in an infinite loop.

Timed Event Switch States

Discontinuous events which occur at irregular but predetermined times can be modeled most efficiently using a special syntax which tells the Easy5 variable step integration algorithms to take an integration step exactly up to the event time rather than stepping past the event and then performing a search for the event time.

This special syntax is:

```
AT TIME = event_time, SET SWITCH, switch_state_name=new_value
```

Here event_time should be the name of a variable (either an output variable or a local variable) which has be set to the value of the event time, switch_state_name is the name of the switch state, and new_value is the value that the switch state should be set to when TIME = *event_time*.

For an example of a library component using switch states, examine the PT component of the Easy5 gp library using the "**Examine Component…**" in the Library menu.

Perfect Limiting

Certain applications become physically inconsistent if the limits are exceeded even by an extremely small amount. For this reason, an alternate output, S_LimOut, is provided for standard component IH, which assumes exactly the limit value when a limit is reached. However, S_LimOut is a variable, and may result in an implicit-loop problem in a model.

Additional Information on Switch States

Further information on switch states is given in the *Easy5 Technical Notes - Switch States*. This technical note is provided as a PDF file in the Easy5 Guide. To view the file, select **Help** > **Easy5 Guide** > **Technical Notes** > **Switch States**.

Temporary Settings File

There are two methods to change the parameter input data that characterizes your model. First, you may directly modify the parameter values in the component data tables. This alters the parameters that characterize the "nominal" model. A second method is to define a temporary settings file that contains a set of special data that temporarily modifies the "nominal" parameter values. A temporary settings file is used for this purpose. This is the preferred method since it does not destroy the nominal parameters defined in the component's data table. However, the temporary setting file data can be loaded back into the model.

The temporary settings file is an external file containing a data base of parameter values. This data base is linked into the analysis data form by selecting the "Parameter Settings File" input field at the bottom of the analysis forms. Prior to the execution of an analysis, these values are read in, and override the nominal parameter values obtained from the component data tables.

Creating a New Temporary Settings File

First you need to create the temporary settings file. To do this, select **File** > **Open Temporary Settings Editor**.

Note:	This file may be created at any time. You do not need to edit an analysis data form to edit both temporary settings and auxiliary input files. You can access these files by selecting **File** from the main window menu bar.

Input a filename in the **Settings** field that uniquely defines your temporary settings file. For example, if you are setting up a temporary settings file to alter the model to simulate a step input of 10 degrees, call the file `Step_input_15`.

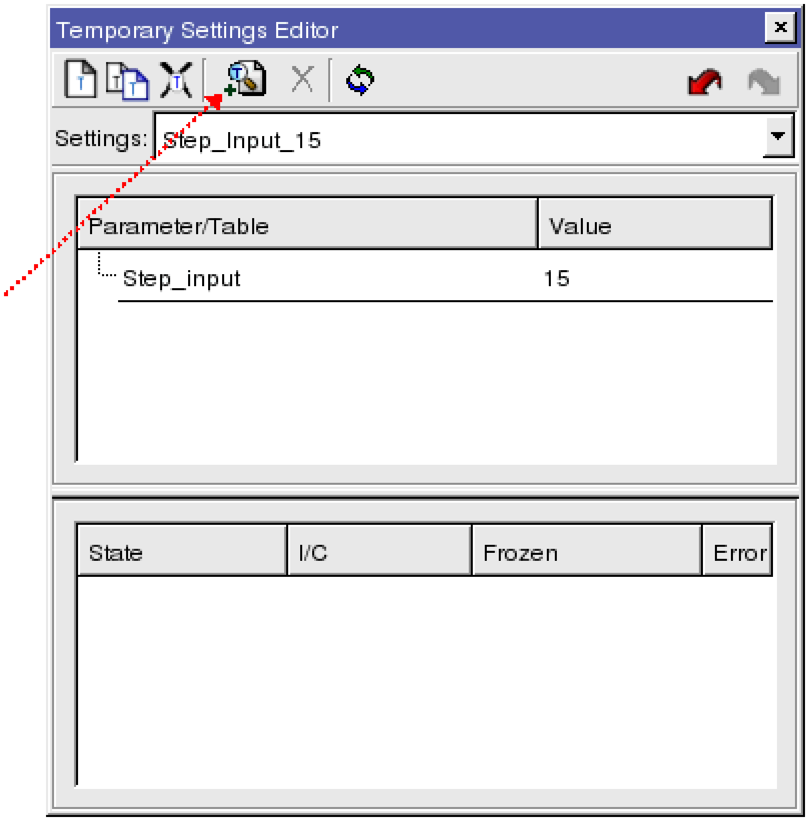
Figure 153 shows an example of an empty temporary settings editor. This editor is used to add and define temporary settings files, including parameter and state names and values that make up the temporary settings file.

To add a temporary settings name, you first need to select the "**Add a model parameter, table, or state from a pick list**" toolbar icon. This automatically opens a Model Explorer window, which is then used for all "picking" of model quantities.

Entering and Editing Data in the Temporary Settings Editor

The Model Explorer window is used to specify model names to your temporary settings file. Just like any other "pickable" name, temporary settings parameters, tables, or states are selected in the same manner.

First, you need to select the "**Add**" button, as indicated by the red arrow in Figure 153. Then, if you select any parameter, table, or state name in the Model Explorer window, that name and all associated data is automatically copied into the temporary settings data table.



Settings: Step_Input_15

Parameter/Table	Value
Step_input	15

State	I/C	Frozen	Error

Figure 153 Temporary Settings Editor

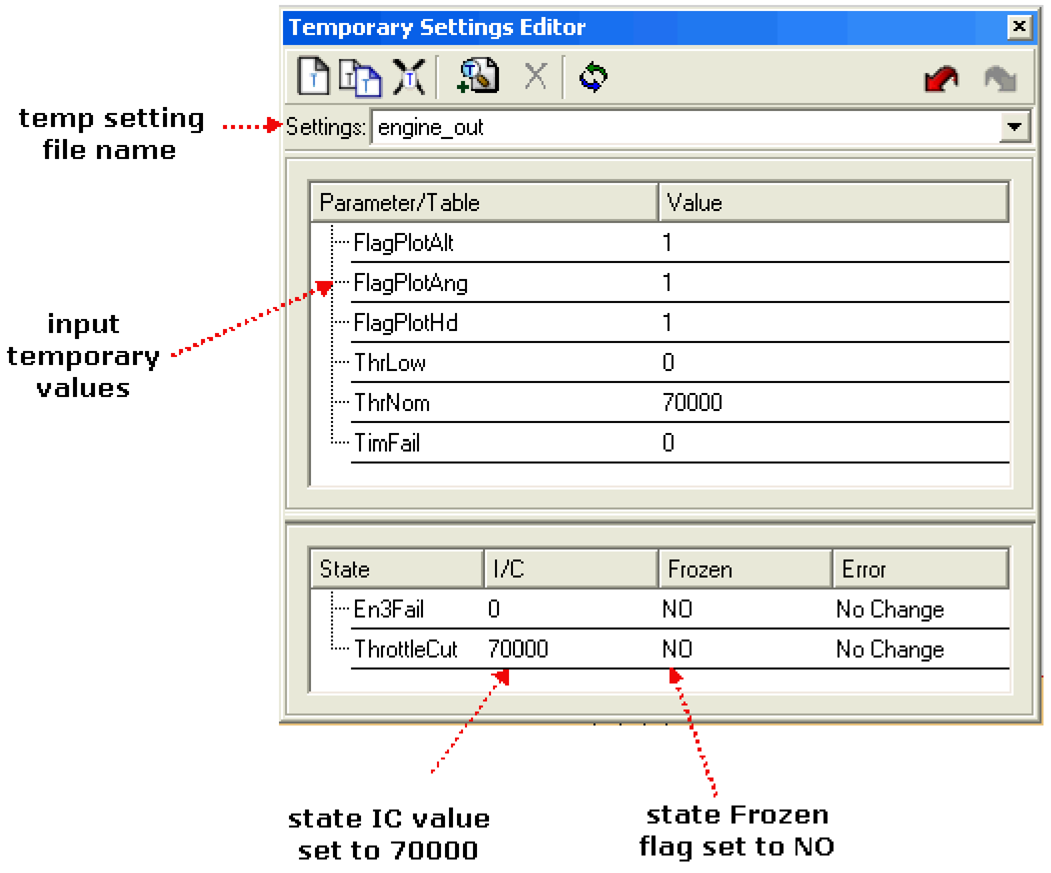
Please note that the Model Explorer window indicates the current model value for all parameters and states.

Figure 154 shows an example of a temporary settings file automatically filled in with parameter inputs and state conditions. To change any of the parameter values in the temporary settings data table, simply select the data value field and enter a new value.

When a state is added to a temporary settings file, the “Frozen” and “Error” fields are set to the default of “No Change”. In Figure 154, the state named *S_out_DF* has both the Frozen and Error fields set to “No Change”.

The “No Change” designation means that the user does not want to change the setting in the temporary settings file, and instead, wishes to use the current model setting.

For example, if a state error control is set in the component data table as. 00005, then, this default value will be used in the temporary settings file and will not be changed. This allows the user to update and modify the values in the component data table, and have these values automatically applied to the temporary setting file.


Temporary Settings Editor
temp setting file name
input temporary values
state IC value set to 70000
state Frozen flag set to NO

Settings: engine_out

Parameter/Table	Value
FlagPlotAlt	1
FlagPlotAng	1
FlagPlotHd	1
ThrLow	0
ThrNom	70000
TimFail	0

State	I/C	Frozen	Error
En3Fail	0	NO	No Change
ThrottleCut	70000	NO	No Change

Figure 154 New Temporary Settings File

You can change the “No Change” values by selecting them with the left mouse button. The Frozen field will toggle between YES (to freeze the state), NO (to unfreeze the state), and “No Change”.

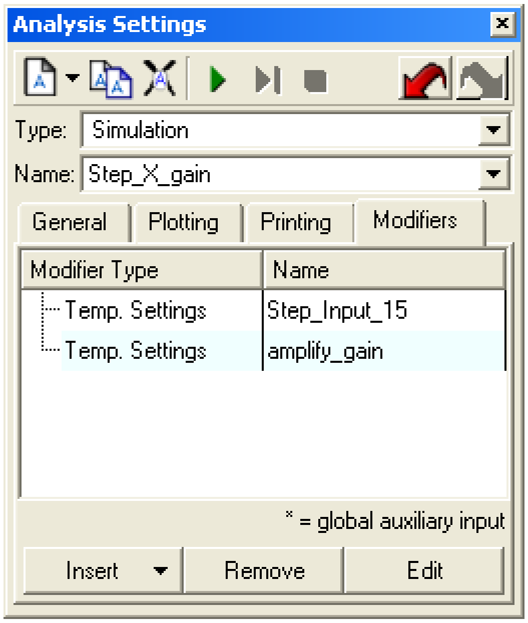
In Figure 154, the state named *EneFail_FO* has the Frozen flag changed to *NO*, and *the ThrottleCut IC value set to 70000*. To change the Error field from “No Change” to a numeric value, just left-click the “No Change” value and enter a numeric value.

To delete parameters and states from the temporary settings data table, select the appropriate name(s) in the data table, and select the Delete icon from the toolbar. Easy5 will remove the selected quantities and all associated data from the table.

To close the temporary settings editor select the small “x” in the upper corner. Once a temporary settings file has been created, it can be edited, deleted, copied or renamed using the Temporary Settings Editor. Finally, all changes applied in the Temporary Settings Editor have an undo/redo capability.

Applying a Temporary Settings File to an Analysis

Having created the temporary settings file, you need to apply (add) this file to the analysis data form. This is done in the analysis data form by selecting the **Modifiers** tab, followed by the **Insert** button, and selecting "**Temporary Settings File**". A new entry will be added, with a pulldown menu containing all available temporary settings files will appear on your screen as shown in Figure 155.



Type: Simulation
Name: Step_X_gain

Modifier Type	Name
Temp. Settings	Step_Input_15
Temp. Settings	amplify_gain

* = global auxiliary input

Figure 155 Adding Temporary Settings to an Analysis as a Modifier

Select the appropriate file from this pull-down list, and Easy5 will automatically insert the name into the analysis data form.

To delete a temporary settings file name from the data form, select the file name, and then select **Remove**. Similarly, select **Edit** to open a specific temporary settings file.

An example of linking an analysis in a temporary settings file is also shown in Figure 155. This figure shows the temporary setting file named `Step_Input_15` linked to a specific simulation analysis. If you have more than one temporary settings file that you want to use, select the next "Temporary settings file" input field and follow the same procedure you used to enter the first file.

Loading Temporary Settings Data Into the Model

A Temporary Settings "file" is generally used to test different data settings and conditions when running analyses. As such, it is ideal for parametric studies. However, once the data settings have been determined and tested, you may want to load them back into the model. This can be done manually by opening every component and typing in the data from the desired Temporary Setting File. Or, the data contained in a Temporary Settings File can be loaded back into the model by selecting the "**Update the model with the temp settings data**" toolbar icon in the Temp Settings Editor. This will load all the "temporary settings" data into the model.

Termination Commands

See Also: "Stop and Exit Flags"

Special commands can be used to execute code which is called at the end of an Easy5 analysis or when an Easy5 analysis execution terminates (either normally or abnormally). This capability is provided for users who have tasks they wish to be done "on the last valid call to the model". This capability is more flexible than the "last call" concept in the following ways:

1. If a simulation has become unstable and "blows up", it is not generally possible to call the model again.
2. There are times when the end simulation cannot be detected until it has happened so you would have to make an extra complete call to the model to execute the termination code.

Termination Commands

The commands, `BEGIN TERMINATION CODE` and `END TERMINATION CODE`, delineate the termination code. These termination commands are entered in Fortran and Library components, but are not available for C components. The commands should be in pairs, and entered on separate lines before and after the termination code. Because these are Easy5 commands, they are case insensitive, and can start in any character field (that is, you can enter these commands in the first character field).

Note: The termination commands can only be entered in Fortran User-Code or Library components. Currently, termination code is only implemented for simulation and steady-state analyses.

Termination Mode Flag

The integer*4 variable `INDP` is available within the termination code and has the following meaning:

1. **INDP = 0** means Easy5 has completed everything and is terminating the run normally (this means the termination code may be called more than once; e.g. when the analysis is over and when the run is over).
2. **INDP = INST** means Easy5 has completed normally an analysis with type INST (e.g. situation has a type code of INST = 26. See "Reserved Words" for other codes). Currently, termination code is only implemented for simulation and steady-state analyses.
3. **INDP = -INST** means and error has occurred during an analysis of type INST and Easy5 is exiting.

Example

Suppose you are designing an Easy5 component which is computing variables x and y and you wish the final values of those variables to be written out to a file named xyz at the end of a simulation even if that simulation fails to complete. You could do this with the following code:

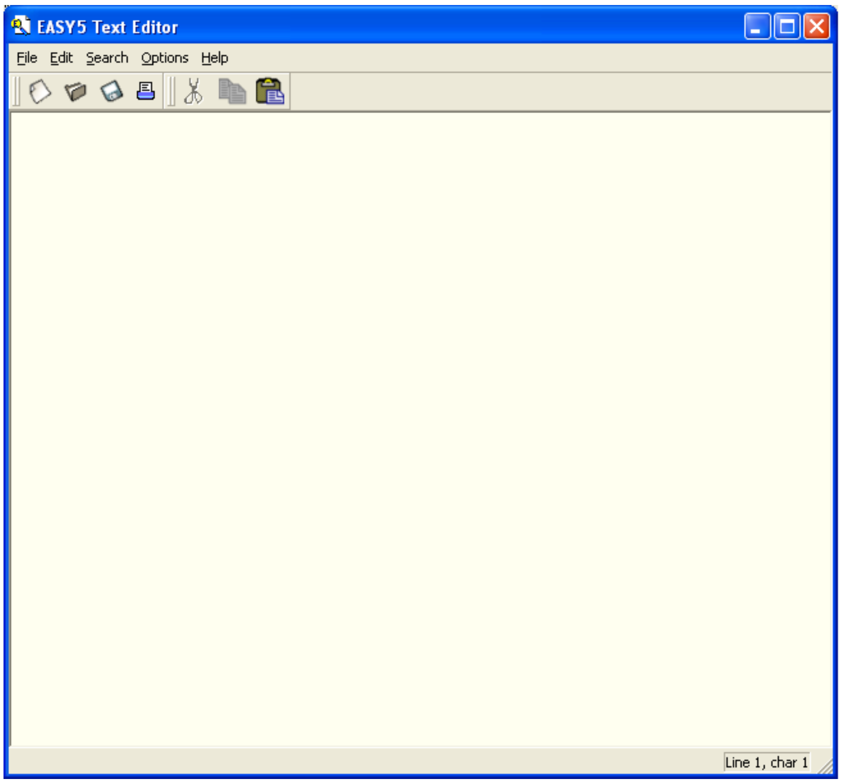
```
  < Code to calculate x and y >
Begin Termination Code
  If ( INDP .EQ. 26 ) Then
     Open(76, file = 'xyz')
     Write(76,'(2G20.13)')'Successful simulation: ' x, y
     Close(76)
  ELSE IF ( INDP .EQ. -26 ) Then
     Open(76, file = 'xyz')
```

```
        Write(76,'(2G20.13)')'Simulation FAILED: ' x, y
        Close(76)
   End If
End Termination Code
```

Text Editor

The Easy5 Text Editor, is a basic text editor offering the ability to view (and edit) output files generated by Easy5. It also offers basic cut/copy/paste operations, search/replace, and goto a specific line number.

An example of the application window is shown below:



This is used to display the following types of Easy5-specific text files:

- Auxiliary input files
- Model generation listing
- Analysis output listing
- Log files

In addition, it is invoked via the **File > Open Text Editor...** menu item. Individualized settings are stored on a per-user basis (i.e. font selections, etc.).

User-Defined Text Editor

The Easy5 Text Editor is not designed as a “full-service” text editor, nor can it handle large files very well (i.e. it loads the entire file into memory at once). Recognizing the potential need for using other text editors, and knowing that users have their individual preferences, we allow users the ability to users the ability to use their own text editor for such text files. This situation can occur if output files get very large. To offer an alternative text editor for these situations, Easy5 offers the following approach using environment variables.

This is done via two environment variables:

`EZ5_TEXT_EDITOR=`<*absolute path to text editor*>

`EZ5_USE_TEXT_EDITOR_SIZE_MB=`<*threshold size in MB of file*>

For example, to specify use of the “gvim” editor for all cases, one could set the following (global) environment variables (before invoking Easy5):

```
set EZ5_TEXT_EDITOR=c:\progra~1\vim\vim70\gvim.exe
set EZ5_USE_TEXT_EDITOR_SIZE_MB=0
```

Setting a value of 0 for the latter, means always use this text editor. Setting a value greater than 0 means, only use the text editor when the file size is larger than that value (in MB).

Note:	Use of a user-defined text editor does not apply to the Easy5 Code Editors (User-Code components, or Library components). These always use the embedded code editor.

TIME - Testing on the Value of Time

You may want to test on the value of TIME or an event in a conditional block in your model equations. In most circumstances there is no problem in doing this. There are certain conditions. however, that you should avoid. In general, while you can always extract data from your model, when setting values in your model you must be very careful.

By intuition, time appears to flow continuously forward during a simulation. With numerical integration, however, this is not so. In fact, time actually "jerks" forward in discrete steps, where the step size is governed according to the method used. For variable step integration methods, time may "back up" as well.

If you want to test on TIME to generate additional printout or data, you should add a condition that tests if the point in time is a "valid" call to the model. Otherwise, the call to the model may just be a "test" point whose data would otherwise be discarded. Usually, you want the additional data generated to coincide with other data generated during the simulation. If so, you will want to add a test on the variable ITINC. This additional test is necessary for all integration methods except the first order Euler method. Even the second order, fixed step Huen integration method makes two calls to your model for every value of TIME, which might cause confusion. The suggested form for conditional tests (for generating additional data) on the variable TIME (or some other event), where condition is the test, is as follows:

```
IF (TIME condition .AND. ITINC .EQ. 1) THEN
. . .
. . .
(conditional block of Fortran or Library component code)
```

```
      .  .  .
      .  .  .
      ENDIF
```

Note: If you require better accuracy and are using the RUNGE KUTTA integrator, you can instead test on the variable IERR equal to one. IERR is a flag that indicates that ERROR CONTROLS were satisfied on a given time step, a so called “valid call”. This will, however, not guarantee that time goes forward or that points in time coincide with other simulation data.

You must be careful that the conditional block does not set the values of any quantities of your model into memory. If something is set into memory, it will use this value during the next call to your model equations. If the next call to the model occurs at a value of time less than or equal to the last call, the saved value will probably be invalid.

If you need to set a value into memory, you should use a switch state instead. The switch state should be used to trigger the event that occurs at some time, tevent. Then, you simply test on the first call of your model, for which the value of the switch state has been set. This is guaranteed to be a valid call to the model. This is the most accurate method for determining the point in time when the event occurs. For example, consider a model where you want to set some values at some point in time, say, during a rocket trajectory. You could use the following form of code in a Fortran or Library component:

```
C
C ----- DETERMINE VALUE OF SWITCH STATE
C
      IF(TIME.LT. tevent) THEN
         switch_state_rate = Voff
      ELSE
         switch_state_rate = Von
      ENDIF
SET SWITCH, switch_state = switch_state_rate
C
C ----- TEST ON VALUE OF SWITCH STATE TO SET VALUES IN MODEL
C
      IF(switch_state .EQ. Von) THEN
      .  .  .                .
      (set values here)
      .  .  .                .
      ENDIF
```

In the previous example, *tevent* is the time at which the event occurs, and *Voff, Von*, are the values used for the switch state, indicating whether the event has occurred or not.

Transfer Function Analysis

See also: "Transfer Function Analysis Methods"

"Transfer Function Troubleshooting"

"Transfer Function of Sampled Data Systems"

Easy5 performs a frequency response analysis by calculating a transfer function between any two points in your model. Transfer Function analysis can be performed on both continuous and/or sampled data models. For systems containing both continuous and digital elements, a z-plane transfer function is calculated.

Transfer Function analysis in Easy5 consists of two parts. First, a linearization automatically occurs about the defined operating point, followed by calculation of the poles, the zeros, and the leading coefficient of the system. You only need to define the Transfer Function input and output in the Transfer Function Data Form as explained in the following section.

The Transfer Function Data Form

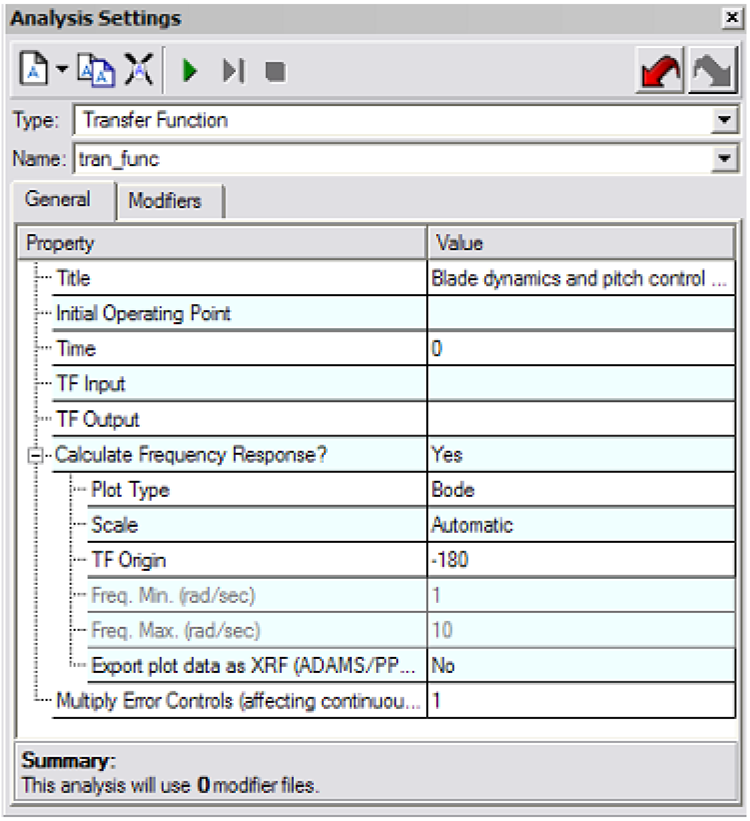
All specifications for performing a Transfer Function analysis are defined with the Transfer Function Data Form via mouse and keyboard inputs. To access this data form, select **Analysis** > **Linear** > **Transfer Function...** from the main menu bar. Instructions for entering the required data are explained in the following sections.

Figure 156 is an example of the transfer function data form with all options shown. To save the settings in this data form, and to fill in the title, time and initial operating point data fields, refer to “Analysis Data Form”.

Specifying the Transfer Function Input

To define an input, select the respective data field, and enter the name of any parameter, state or variable in your model. You can also define an input using the “Pick” method as discussed in “Model Explorer Window”.

There are special requirements that you must consider when using variables and states as Transfer Function inputs. See “Transfer Function Troubleshooting” for more information.


Analysis Settings

Type: Transfer Function

Name: tran_func

Property	Value
Title	Blade dynamics and pitch control ...
Initial Operating Point	
Time	0
TF Input	
TF Output	
Calculate Frequency Response?	Yes
Plot Type	Bode
Scale	Automatic
TF Origin	-180
Freq. Min. (rad/sec)	1
Freq. Max. (rad/sec)	10
Export plot data as XRF (ADAMS/PP...	No
Multiply Error Controls (affecting continuou...	1

Summary:
This analysis will use **0** modifier files.

Figure 156 Transfer Function Data Form

Specifying the Transfer Function Output

“TF Output” may be the name of any state or variable in your model. To enter an output name, select the respective data field and enter the name. You can also define an output using the “Pick” method. See "Model Explorer Window".

Requesting a Frequency Response Plot

If you want a frequency response plot as part of the Transfer Function analysis, select “Yes” following “Frequency Response” on the transfer function data form. If you do this, additional data fields and options will appear on the form. Use the additional fields and options to specify a frequency response plot type and scaling information. The usage of “Plot Type”, “Scales” and “TF origin” is self explanatory.

Scales

When automatic scales is selected, Easy5 automatically finds the maximum and minimum of all quantities to be plotted. For the frequency axis, the minimum frequency is 1/10 of the lowest non zero frequency in the model, and the maximum frequency is 10 times the highest frequency in the model.

If “Manual” scales are selected, then you must define two additional fields: “Freq Max:” and "Freq Min:". These frequencies define the range of frequencies to be *plotted*. Enter the maximum and minimum frequencies, in the units of radians/sec. If you enter “0” in any of these fields, Easy5 will use automatic scales.

For example, if you wished to use manual scales to define the minimum frequency to be 1 rad/sec, but not define an upper frequency limit, then enter:

Freq Max: 0

Freq Min: 1.

TF Origin

The Transfer Function phase is always displayed with a total 360^o. The “Tf Origin” defines the phase plot scale midpoint value. By default this is set to -180^o, which results in the Bode phase plot being displayed with -180^o as the midpoint, and therefore the scale is from 0^o to -360^o. You can set this TF Origin to any value. For example, if you set this to -90^o, the phase plot scale will be from +90^o to -270^o.

Transfer Function Analysis Output Data

Transfer Function analysis results can be in plotted and/or printed formats. These are described in the following sections.

Transfer Function Analysis Plots

If frequency response plots were generated during your analysis, you can view them with the Easy5 online plot program. To start the Easy5 Plotter, select **Analysis > Plot Current Results** (Ctrl+Shift+P). For more information see the User Guide, Chapter 7 - Easy5 Plotter.

The Transfer Function Analysis Output Listing

After the transfer function analysis has completed, you may analyze the output by selecting **Analysis > Display Analysis Output Listing** (Ctrl+Shift+L). Once you are looking at the Easy5 analysis output listing file, search for the string "TRANSFER FUNCTION ANALYSIS" to position yourself at the top of the transfer function output listing. Listed first are the “TF Input” and 'TF Output” selections as well as the title. These are followed by information describing the operating point. Following the operating point information is a printout showing the zeros, the leading coefficient, and the poles of the system. For a sampled data system this would include both s-plane and z-plane results.

The frequency response is listed in tabular form and includes frequency (in rad/sec and Hz), gain (in both magnitude and db), and phase (in degrees). For sampled data systems a column for the w-plane frequency will also be included. Finally, the total amount of CPU seconds expended during the analysis is shown at the bottom.

Note: You should be very careful about extending the results of multirate discrete transfer functions to situations other than simple gain connections between the transfer function output and input.

Transfer Function Analysis Methods

See also:	"Transfer Function Analysis"
	"Transfer Function Troubleshooting"
	"Transfer Function of Sampled Data Systems"

This section describes the theory used in calculating a transfer function used by the Transfer Function analysis. It describes both the Frequency Response and Transfer Function methods, and shows the difference between a continuous and sampled data system transfer function.

Frequency Response Method

A linearized model, described by your input, output, and nonlinear model, is calculated about the operating point. The linearized model is represented by system poles, zeros, and a leading coefficient.

In effect, you replace your nonlinear model with a linear model accompanied by one input and one output, as illustrated in Figure 157.

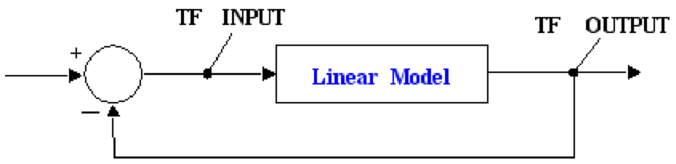


Figure 157 Linearized Model, Transfer Function Analysis

This linear model is used in the frequency response analysis discussed below.

Computationally, a frequency response analysis, as shown in Figure 158, involves injecting a sinuosity input signal at a frequency, ω, an amplitude, Ain, and phase angle, Φin, and measuring a corresponding steady-state output signal also having an amplitude, Aout, and phase angle, Φout.

The transfer function gain (or simply, Gain), G, and transfer function phase (or simply, Phase), Φ, are given by:

transfer function gain: $G=| A_{out}/A_{in} |$

transfer function phase: $\Phi= \Phi_{out} - \Phi_{in}$

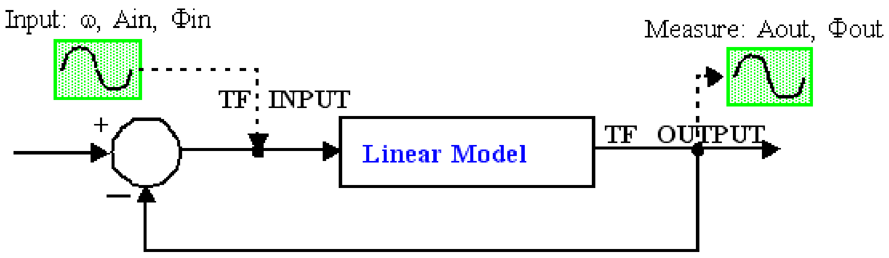


Figure 158 Linearized Model, Transfer Function Analysis

A frequency response is created by calculating the gain and phase over a selected range of frequencies. For convenience, the gain is expressed in both magnitude and decibels (dB). Units for frequency are in radians per second (rad/sec) or Hertz (Hz), and units for phase are in degrees. The selection of frequency points can be an important factor in the accuracy and resolution of the results, as described in the next section.

Selecting Frequency Points

Easy5 automatically selects frequency values for the frequency response to be evaluated; there is no need for you to provide such values. The frequency point selection scheme is designed to minimize discontinuities and provide you with a smooth and accurate “curve”. Thus, for systems with lightly damped poles, you are assured of good resolution in such areas of rapidly changing frequency response.

The frequencies Easy5 selects will automatically span your system eigenvalues unless you limit the selection range with manual frequency bounds. For sampled data systems, the upper frequency bound using automatic scaling is the Nyquist frequency (of the fundamental sampling rate). Also, sophisticated search algorithms find both the 0 dB gain crossover and the -180 degree crossover, which are critical stability reference points.

Transfer Function Analysis Method Selection

Depending on your type of model, either a continuous (s-plane) or a sampled data (z-plane) analysis will be performed. Easy5 automatically selects the appropriate method for your model.

Continuous Systems

For linear systems, Easy5 creates a linear model of the form:

$$\dot{x} = Ax + Bu + Kx$$

$$y = Cx + Du + Ky$$

A = nxn system Jacobian (stability matrix); n= number of states

B = nxm system input (control) matrix; m= number of inputs

u = scalar system input vector

x = system state vector of n length

y = scalar system output vector

C = pxn system output matrix, where p = the number of outputs

D = pxm direct transmission gain matrix

Kx = constant rate vector

Ky = constant output vector

The linear model is formed by Easy5 about your operating point by perturbing your system states and the system input (specified as **TF INPUT**).

The transfer function poles are the eigenvalues of your system Jacobian matrix, **A**. The zeros and lead coefficient of the transfer function are calculated using the method of Patel. The zeros are obtained as the eigenvalues of a matrix whose order equals the number of zeros. This matrix is determined from the **A, B, C,** and **D** matrices and uses a series of coordinate transformations.

Having determined the n poles, Pi, the m zeros, Zi, and the lead coefficient, L, the transfer function can be stated as:

$$G(s) = \frac{L\Pi|_{i=1}^{m}(s - Zi)}{\Pi|_{i=1}^{n}(s - ZPi)}$$

If m equals zero, the numerator is equal to L.

Sampled Data Systems

For sample data systems, where the input and output are digital signals, the linearized model assumes the form:

$$zx_z = A_z x_z + B_z U_z$$

$$Y_z = C_z x_z + D_z U_z$$

where the z subscripted quantities are the z-domain counterparts of the s-domain quantities. For multi-rate systems, the z-domain sampling period used, called the fundamental sampling rate, is the slowest or least common multiple of your system’s sampling periods. The **Az** matrix is called the system transition matrix, and is calculated using the method described in Ap. C: Discrete Analysis Techniques.

Given these matrices, the Patel method is used to calculate the poles, zeros, and lead coefficient of the transfer function.

Coordinate Transformation

All transfer function results for a sampled data system are calculated in the z-domain with a sample period equal to the fundamental sampling period. System singularities, the poles and zeros, are given in both the z-plane and the s-plane. The definition of z ($z=e^{Ts}$) is used to transform from the z-plane to the s-plane.

These s-plane singularities include both the poles and zeros of the transformed continuous plant and digital compensation, and give a correct representation of the aliased plant dynamics. Compensation specified in the s-plane (as is done with the DF and DL standard components), and transformed to the z-plane, may be different from the original specification when transformed back to the s-plane. This change is due to differences between the z-transform and the bilinear transform.

Frequency values on the unit circle in the z-plane are related to their corresponding values on the imaginary axis in the s-plane by the relation:

$$z = cos(w \cdot \mathrm{Tmax}) + j\,sin(w \cdot \mathrm{Tmax})$$

where:

Tmax = fundamental sample period, seconds

ω = frequency, rad/sec

z = z-plane complex variable

Caution:	You should be very careful about extending the results of multi-rate, sampled data Transfer Function analyses to situations other than simple gain connections between the **TF OUTPUT** and **TF INPUT**.

Transfer Function Troubleshooting

See Also: "Transfer Function Analysis"
"Transfer Function Analysis Methods"
"Transfer Function of Sampled Data Systems"

The Transfer Function analysis can fail for various reasons. There may be no connection between the specified input and output. If this is the case, a warning is given, and you must correct this as explained in the next section. Also, the "TF Input" can be the name of any state, variable, or parameter. However, two situations must be avoided when specifying the input as discussed in the following two sections. For sampled data systems, additional requirements exist.

No Connection Exists Between Tf Input and Tf Output

In order to run the Transfer Function Analysis, you must specify the TF INPUT and TF OUTPUT. If there is no connection between the input and output, the analysis will fail and the following error message is printed in the analysis output listing file:

NO CONNECTION EXISTS BETWEEN TF INPUT AND TF OUTPUT

An example of this condition is clearly shown in Figure 159.

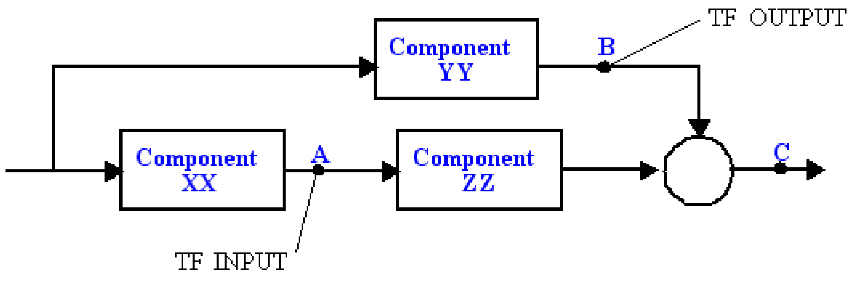


Figure 159 Example of Incorrect TF INPUT and TF OUTPUT

This simple model has the TF INPUT defined at point A, the output of Component XX. The TF OUTPUT is defined at point B, the output of Component YY. It is obvious that there is no connection between these two points. If you inject a signal at Point A, you will not be able to measure the result of this signal at point B.

If this were a closed loop system where the output of the summation were fed back into Component‘s XX and YY, then a connection would exist between the input and output.

There is no way to solve this problem, it is physically constrained. However, if you can redefine either the input or output, a transfer function can be solved. For example, define the TF OUTPUT at Point C, and you will have a connection between the input and output.

A less obvious problem occurs when hard limits, switches, and switch states are used. Assume one or more of the components is in the transfer function loop (between TF INPUT and TF OUTPUT). If the component is at a hard limit, or the switch is opened, then the Transfer Function Analysis will fail and print the "*NO CONNECTION EXISTS BETWEEN TF INPUT AND TF OUTPUT*" error message.

An example of such a system is shown in Figure 160. Assume that the output of the Summing Junction component SJ is used as the TF INPUT, and the output of the Transfer Function component TF is the TF OUTPUT. Note that the transfer function loop contains the Integrator with Hard Limits component (IH) and the Switch component (SW), and that the IH component has hard limits at + 2.0.

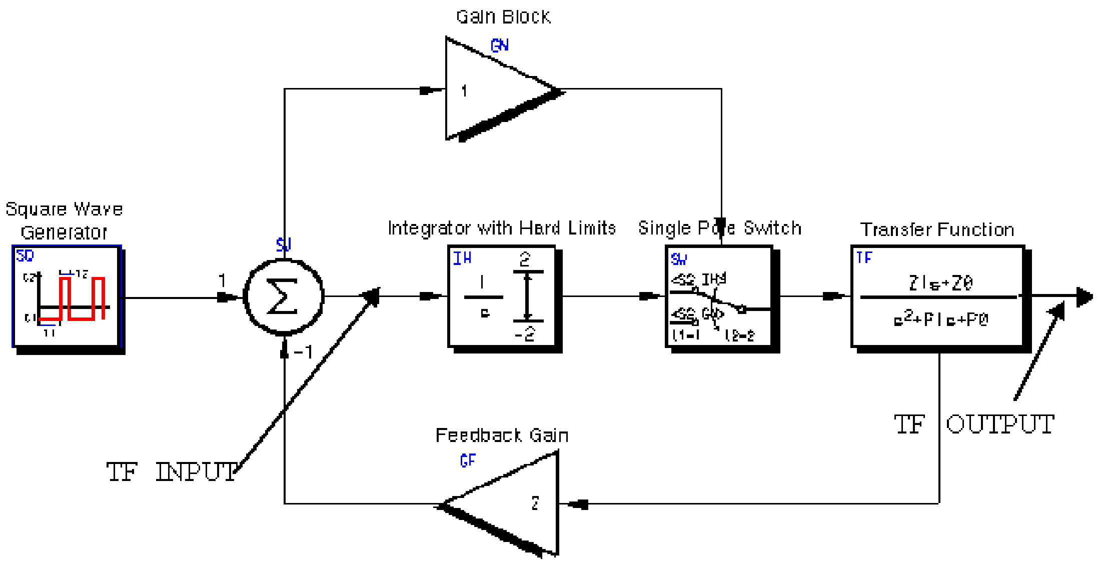


Figure 160 Example of a Broken Transfer Function Connection

First assume that the output of IH is defined at the hard limit of 2.0. *The Transfer Function Analysis will fail with these conditions!* This is because the sinusoidal input signal injected at the TF INPUT will not be seen at the TF OUTPUT. The sine wave is "broken" by the hard limit of the IH component, and you will get the error message: "*NO CONNECTION EXISTS BETWEEN TF INPUT AND TF OUTPUT*".

The transfer function loop will also be broken if the switch is opened such that it inputs the signal from the GN component. This “breaks” the loop between the IH and the SW component and will result in the same error message.

Variables as Transfer Function Inputs

When you use an Easy5 variable name as the **TF INPUT**, you must *make sure that it is not used to calculate other variables within the same standard component (or sort block) that are passed as outputs to other components*. Easy5 perturbs the **TF INPUT** by starting at the statement in SUBROUTINE EQMO just below the (sort block) definition of the variable, and uses a perturbed value for the variable (which would otherwise be calculated).

If other variables defined in the same component (or sort block) are a function of the **TF INPUT**, they will not see any effect from the perturbation. This will result in an incorrect transfer function calculation if the **TF OUTPUT** is connected to one of these "blind" output variables. If this is the case, you may see the following error message is printed in the analysis output listing, even though it seems a connection should have been found:

NO CONNECTION EXISTS BETWEEN TF INPUT AND TF OUTPUT

This problem is perhaps best illustrated by the example shown in Figure 161.

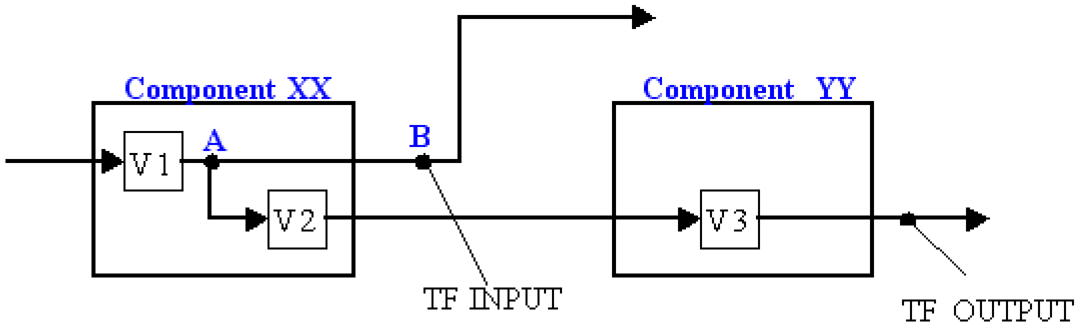


Figure 161 Example of Incorrect TF INPUT as a Variable

Here, the **TF INPUT** is a variable, **V1**, and the **TF OUTPUT** a variable, **V3**. Both variables **V1** and **V2** are calculated in component **XX**, where **V2** is a function of **V1**. Variable **V3** is defined in component **YY** as a function of **V2**. Note that the variable **V1** is perturbed not at location **A** but at location **B** in the block diagram.

Point **B** represents the first statement following the (sort block) definition of variable **V1** in SUBROUTINE EQMO (remember, standard components are accessed as subroutine calls). Thus, no connection exists between the **TF INPUT** and **TF OUTPUT**, and the transfer function cannot be calculated. In contrast, if the variable **V2** had been selected as the **TF INPUT**, there would have been no problem.

States as Transfer Function Inputs

Exactly the opposite situation should be avoided if the **TF INPUT** is set to a state name. Here, **you should ensure that no feedback of the state occurs within the component** (or, if a sorted library component, within the sort block) in which the input is defined. Otherwise, perturbations made to the **TF INPUT**, now made internal to the component, will be fed back (inside the component or sort block) and will become a part of the calculated transfer function. The resulting incorrect dynamic effects will be included in the calculated transfer function.

The lag component LA is a good example of a component that contains internal feedback. The lag component equation is:

d(S2)/dt=(Gain•S1 - S2)/Tau {where S1=input; S2=output (state)}

In this component, the derivative of the output **S2** is a function of **S2**. That is, the derivative of the state is a function of the state. This causes internal feedback. Many components have this relationship that results in internal feedback. Figure 162 illustrates these effects.

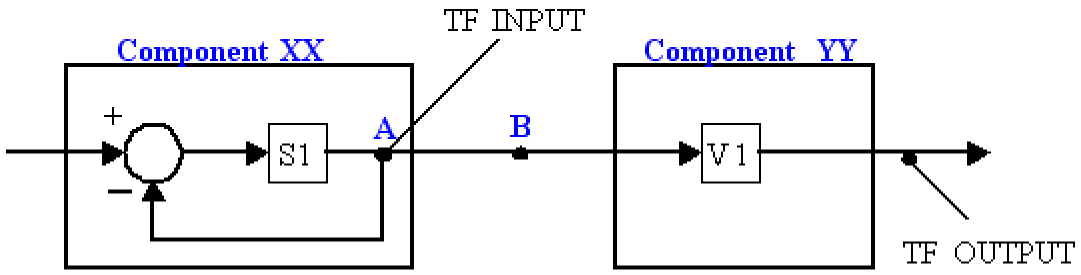


Figure 162 Example of Incorrect TF INPUT as a State

Here, a component **XX** is connected to another component **YY**. The **TF INPUT** is the state **S1** of the **XX** component and the **TF OUTPUT** is the variable **V1** of the **YY** component. Note that the state **S1** is fed back within the component **XX**. Contrary to the situation that occurs for variables as inputs, **TF INPUT S1** is perturbed at point **A**, not at point **B**. This is because it is a state variable.

Thus, the resulting transfer function is incorrect. Freezing the **S1** state will rectify this situation.

Note:	As a general rule, *always freeze the state variable used as a "TF Input"*. Otherwise, signal paths around the state will remain active, and will become a part of the calculated transfer function.

In general, you will obtain better numerical accuracy if you freeze all model states not included in the specified transfer function. If you do not freeze them, zeros will be calculated for the transfer function to cancel the poles corresponding to the extraneous states.

If the previous example shown in Figure 162: Example of Incorrect TF INPUT as a State were a closed loop system such that the TF Output were fed back to the input to Component XX, as shown below in Figure 163: Example of a Transfer Function with Feedback, then setting the TF INPUT at point A, and freezing the "upstream" state S1 would result in a broken loop.

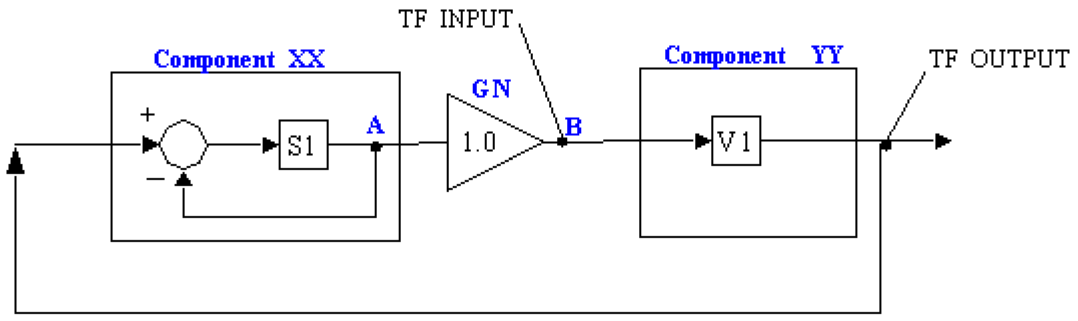


Figure 163 Example of a Transfer Function with Feedback

This will open the loop and would give you incorrect results for a "closed loop" transfer function.

To avoid this, insert a gain block (GN) between components XX and YY, set the gain to “1.0”, get the TF INPUT at point B, and do ***not*** freeze states in the loop. This is the easiest way to guarantee correct results without having to worry about freezing states.

Transfer Function of Sampled Data Systems

See Also:	"Transfer Function Analysis"
	"Transfer Function Analysis Methods"
	"Transfer Function Troubleshooting"

As mentioned in "Transfer Function Analysis Methods", a sampled data transfer function is calculated in the z-domain by forming the discrete time linear model description of your system in the form:

$$x_{n+1} = Ax_n + Bu_n$$

$$y_n = Cx_n + Du_n$$

This form assumes that the input and output are only changing at the fundamental (or slowest) sample time!. If this is not the case for your selection of **TF INPUT** and **TF OUTPUT**, the Transfer Function analysis will be invalid and will not be calculated. If you add a sample and hold (**SH**) component at either the input or the output, Easy5 will perform the requested calculation. You must judge the validity of this change to your model, however.

This invalid condition often occurs when you are trying to perform a transfer function across a continuous portion of a sampled data model. If this is the case, you must first “freeze” all digital states so that the analysis reverts to a continuous time (s-plane) calculation. In other cases, you may find that by requesting a transfer function between a different input and output you can avoid this problem. For an open loop, for instance, you may wish to “break” the loop at a different point in your model.

The following sections describe how to break (or open) the loop of a sampled data system, to calculate the stability margins, and also examines the problems associated with obtaining transfer functions of single rate and multirate sampled data systems.

Stability Margins for Sampled Data Systems

The stability margins of a system can be calculated by applying the Nyquist Stability Criterion to an "open loop" transfer function obtained by "breaking" a feedback path and taking the input and output points respectively just after and just before the break. For continuous systems the breakpoint can be anywhere in the feedback path.

However, for sampled data systems, since the Easy5 Transfer Function analysis computes a discrete system transfer function (see Appendix C for details) and then computes the frequency response by setting $z=e^{ts}$, some configurations present problems. As an example, consider the system shown in Figure 164: Sample Multi Rate Discrete System.

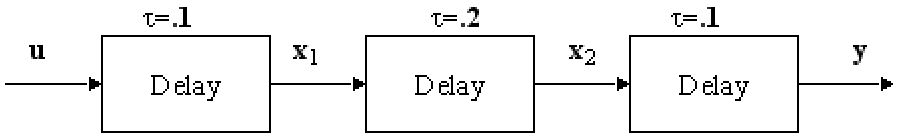


Figure 164 Sample Multi Rate Discrete System

This is a discrete system consisting of three delays in series. The first and third elements sample every.1 sec, and output that value.1 sec later. The middle element samples every 0.2 sec and outputs that value 0.2 sec later. Notice that an element interior to the system is sampling slower than either the input, u or the output, y. This is referred to as an fast slow fast (FSF). To determine a discrete transfer function for this system requires that we express the output y at each sample time in terms of the input u at the current and previous sample times at y at previous sample times. But this cannot be done for this system. To see this, note that:

$$y(.5) = x_2(.4) = x_1(.2) = u(.1)$$

where y(.5) represents the value of y at time=.5, and so on.

This calculation would lead one to expect the transfer function should be:

$$y/u = z^{-4}$$

But the calculation:

$$y(.6) = x_2(.5) = x_1(.2) = u(.1)$$

would lead one to believe that the transfer function should be:

$$y/u = z^{-5}$$

Thus, no single input output relation (transfer function) exists.

This section describes a number of typical situations in which the discrete transfer function cannot be used to compute stability margins and provides an alternate root locus based method which can be used in those situations. We also suggest an approximate method which can be applied in many cases.

If we call components containing dynamics sampled at the fundamental rate "slow" components, and all other components containing dynamics (those having continuous or discrete states) "fast" components the requirements for calculating an Easy5 Transfer Function analysis for a sampled data system can perhaps be better understood by studying the table below. This table shows the consequences for all types of combinations of **TF INPUT**, **TF OUTPUT**, and dynamics occurring elsewhere in the model referred to as "Internal" in the table.

TF INPUT	Internal	TF OUTPUT	Result	Type
Fast	All fast	Fast	No message	FFF
Slow	All slow	Slow	No message	SSS
Slow	Mixed fast and slow	Slow	No message	SFS
Slow	Mixed fast and slow	Fast	Message	SFF
Fast	Any slow	Fast	Fatal error	FSF
Fast	Mixed fast and slow	Slow	Message	FFS

To understand these limitations it is useful to examine what would happen if the Easy5 Transfer Function analysis were allowed to proceed with the computation of the transfer function of a fast slow fast (FSF) system. The first step in the discrete transfer function calculation is the generation of matrices A, B, C, and D such that

$$x_{n+1}= Ax_n + Bu_n$$

$$y_n= Cx_n + Du_n$$

where: x_n= state vector at the nth sample time (time= $n\tau$)

u_n= input vector

y_n= output vector

For a multi-rate system ‘t’ must be taken be taken as the "fundamental" (or slowest) sample period. This representation implicitly assumes that u and y only change at (fundamental) sample times. But if we are computing an open loop transfer function in order to find stability margins of the nominal "closed loop" system, this format assumes that the feedback u_n=y_n is changing only at fundamental sample times.

This is equivalent to adding a fundamental rate sampler at either the input or the output of the systems. Let us call the original system plus this new sampler the augmented system. The augmented system is generally less stable than the original system but there are cases when the stability margins of the original system are greatly exaggerated by using the augmented system. In any case, the stability margins are almost always wrong. For this reason we have not allowed the calculation to proceed.

Note that the validity of computing the transfer function of an FSF system concerns the passage of information through the feedback path at times other than the fundamental sample times. This is not a problem if there is a fundamental sample hold device at either the input or the output. Therefore, the calculation is allowed to proceed for FSS and SSF systems.

There are situations in which combining transfer functions of FSS and SSF system can produce incorrect results. Again, the problem is that the discrete transfer function formulation assumes that the system input and output are only changing at fundamental sample times.

Suppose we have two systems, and SSF system S_1 and a FSS systems S_2. Let S_1S_2 be the system obtained by connecting S_1 and S_2 in series. Then, in almost all cases, the discrete transfer function across S_1S_2 is not the product of the discrete transfer function across S_1 and the discrete transfer function across S_2. For this reason, the discrete transfer function computation is allowed to proceed for FSS, FFS, SSF, and SFF systems but a message is printed suggesting caution in the use of those results.

Single Rate Systems

A typical continuous model is shown in Figure 165.

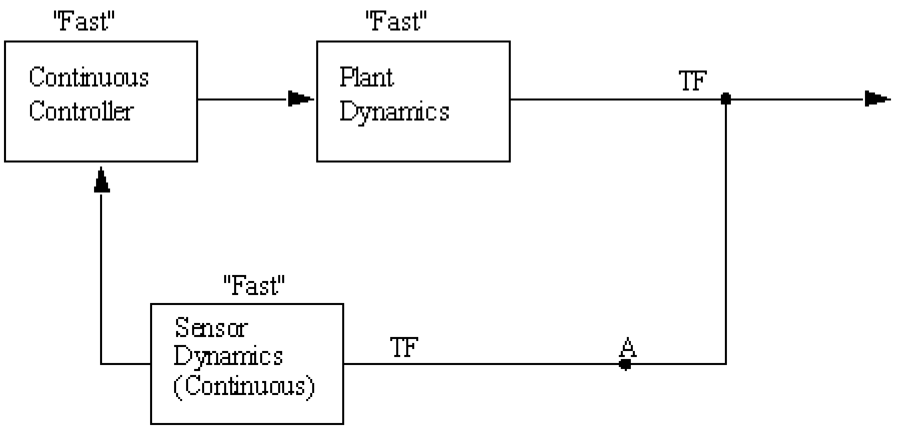


Figure 165 Typical Continuous System Model

If the feedback loop is broken at point A and the **TF INPUT** A and **TF OUTPUT** are chosen as shown, this configuration corresponds to a Fast Fast Fast (FFF) type and the desired margins will be obtained.

Now suppose the continuous controller is replaced with a digital controller, creating a sampled data system. The same selection for **TF INPUT** and **TF OUTPUT** results in a FSF system and the transfer function calculation will not be performed.

This is because both input and output are continuous signals, essentially being "sampled" at an infinitely high rate, while the digital controller represents an embedded "slow" element.

If instead, the breakpoint were moved to point B, giving an SFF configuration as shown in Figure 166, the transfer function calculation would be performed. Since the output is changing at a rate faster than the fundamental rate, a contrary message will be printed.

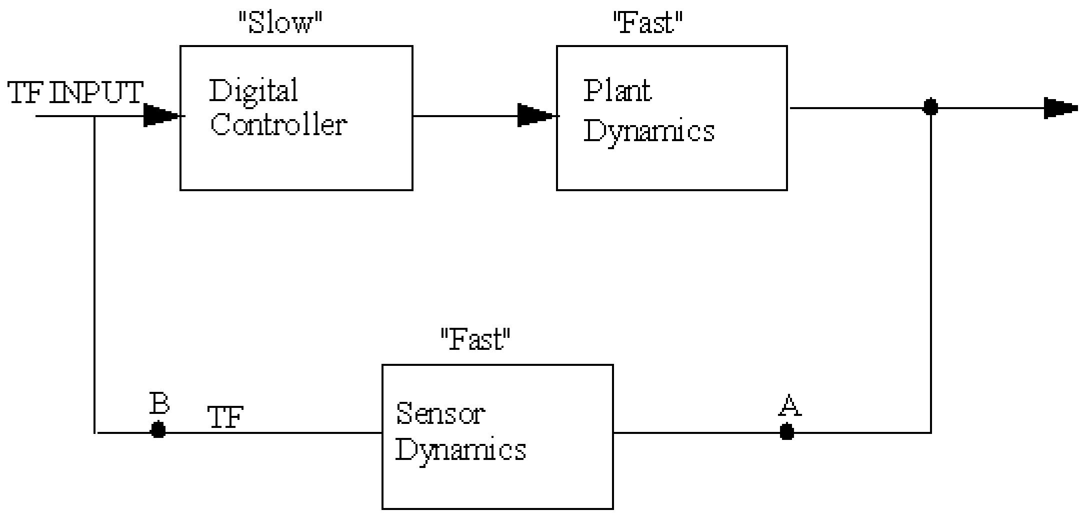


Figure 166 Appropriate Break Point Selection for Sampled Data Model

Multi-Rate Systems

For multi-rate sampled data systems you must consider additional implications of your model topology. The "loop sampling rate" is the rate at which the feedback signal is being sampled. Usually, in "real world" systems sampling occurs within the digital filters as well as in each feedback signal just before it is fed back. For Easy5 models, however, usually a sampler is only located in the discrete component itself, prior to the zero order hold, and the feedback sampler is omitted from the model (it can be included if you want though) as it superfluous.

A frequency response is calculated by perturbing the **TF INPUT** and observing the result at the **TF OUTPUT**. Everything else in your model is considered to be "inside" the loop even if it does not actually lie on a path between the **TF INPUT** and **TF OUTPUT**. In particular, all active samplers in your model other than those associated with the loop being tested are considered to be "inside" the loop. This is the crux of the problem for multi-rate systems because this may constitute an FSF type system. Easy5 will trigger a fatal error if it detects this.

Therefore, two approaches are necessary for calculating stability margins for these types of systems. One approach should be used for loops which contain no embedded slower samplers and another approach for loops containing embedded slower samplers. These are discussed with the use of examples in the following sections.

Approach for Loops Having no Embedded "Slow" Samplers.

Figure 167 shows a typical multi-rate sampled data model, having two loops: an outer position loop, having a sampling rate equal to the fundamental rate (the "slow" loop); and an inner rate loop, having a sampling rate faster than the fundamental rate (the "fast" loop). Note that a zero order hold (ZOH) is appended to each Easy5 discrete component automatically.

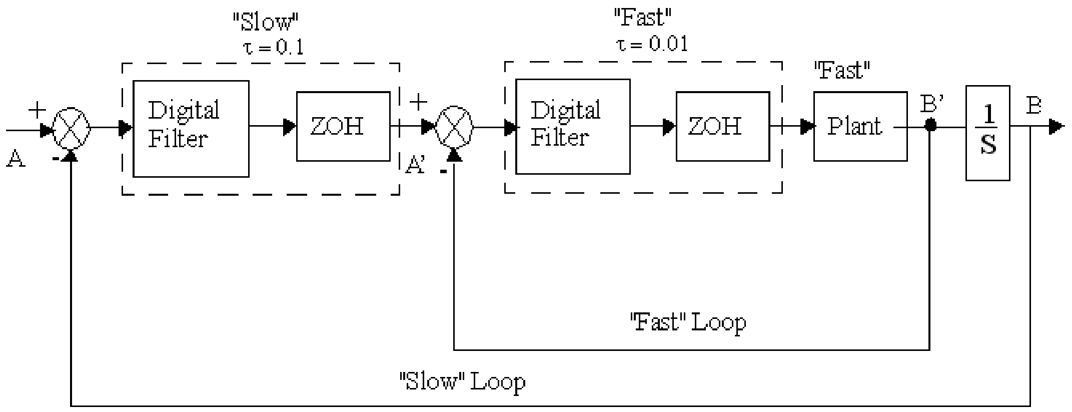


Figure 167 Typical Multi Rate Model

To calculate (SISO) stability margins for this system you can use the discrete transfer function approach only for the outer (slow) loop. This loop is defined by specifying the **TF INPUT** at point A and the **TF OUTPUT** at point B. The inner "fast" loop causes no problems because from the view of the outer loop the system is time invariant. This transfer function calculation has an SFS type configuration. Because an Easy5 Transfer Function analysis is valid for such loops, you can readily obtain gain and phase margins by examining the frequency response data.

Approach for Loops Containing Embedded "Slow" Samplers

This technique requires a little bit of planning as minor changes to your model as necessary. A separate method is required for calculating a gain margin and for calculating a phase margin for a particular loop where the loop contains an embedded slower sampler.

This would occur if you wanted to calculate a transfer function calculation between A' and B' to obtain stability margins for the inner rate loop. Unfortunately, you cannot do this because the resulting transfer function calculation "contains" an embedded "slow" sampler, due to the slow sampler in the outer loop.

Figure 168 shows how the transfer function model actually looks with **TF INPUT** at A' and **TF OUTPUT** at B'. This is an FSF type system and is not allowed.

You can prove that the slow sampler is "inside" the fast loop by imagining that you could freeze (deactivate) the slow sampler. If it were not inside the fast loop, it would not affect the transfer function calculation, but in fact it does.

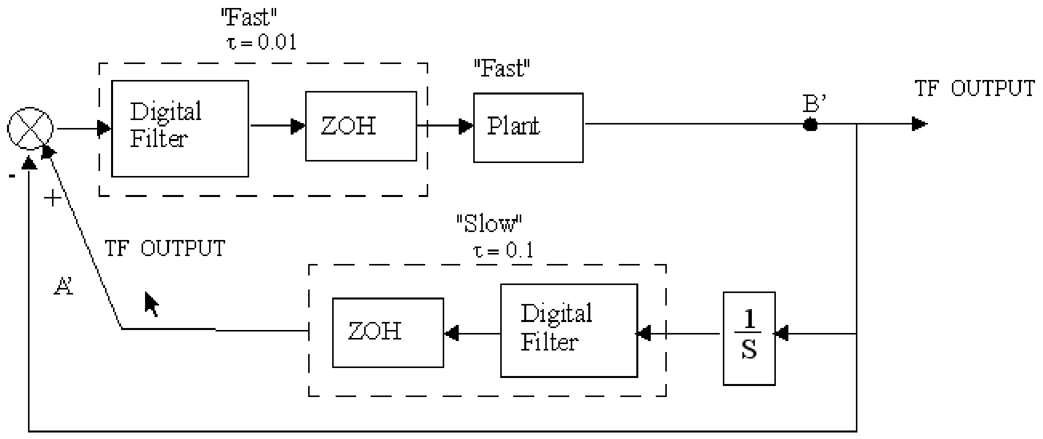


Figure 168 Resulting Invalid Transfer Function Between Points A' and B'

While Easy5's Transfer Function analysis cannot be used for these types of configurations there is another approach to calculating stability margins for these systems. This approach, using Easy5's Root Locus Analysis, is described in the following section.

Root Locus Analysis Technique

To calculate a gain margin add an MA Easy5 standard component in the feedback loop of the fast inner loop, as indicated in Figure 169.

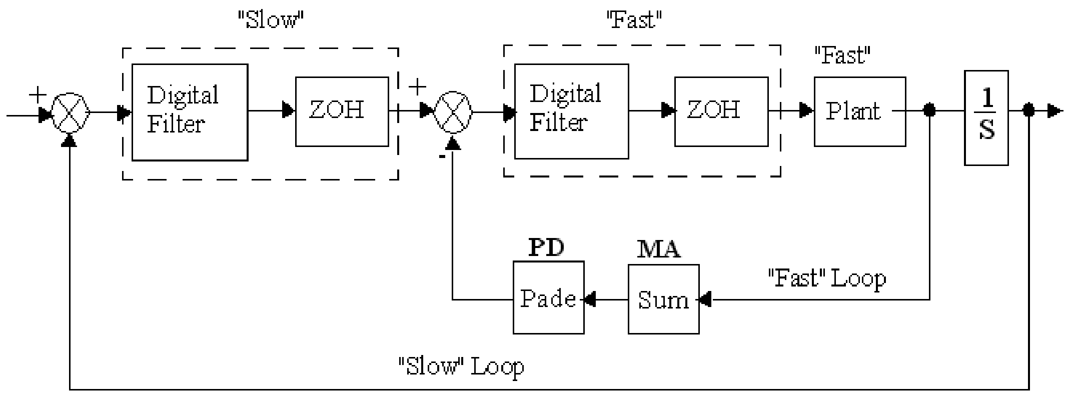


Figure 169 Multi Rate System Showing Placement of MA, PD Components in Inner Loop

Parameter values for this component should be set as follows: **C1_MA** to 1, and **C2_MA** to 0. Then request a Root Locus analysis with the **RL PARAMETER** = **C1_MA**. The gain margin is then determined from the smallest value of **C1_MA**, converted to units of dB, at which a system pole crosses the imaginary (jω) axis.

Phase margins are calculated in a similar manner. However, here the addition of a PD standard component (Pure Time Delay - m_{th} order Pade Approximation) is required as shown in Figure 169. Then, again, with the outer (slow) loop "broken", a Root Locus analysis should be performed on the time delay parameter, **DEL_PD**, of the PD component. The PD component, uses an m_{th}-order Pade approximation to represent a pure time delay. Higher values of n increase accuracy, but also increase the complexity and execution time of your model.

The phase margin is then calculated as follows:

1. Obtain the value of DEL_PD, Δ, at which a system pole crosses the imaginary ($j\omega$) axis.
2. Find the natural frequency, ω_n, of the pole at this point.
3. Multiply ω_n by Δ to get the phase margin in radians.
4. Convert radians to degrees by multiplying by $180/\Pi$.

A Useful Approximation

The Root Locus analysis method described in the previous section yields accurate stability margin results, but requires making more than one Easy5 execution and a number of alternations to the model. In many systems the stability margins are determined by modes whose natural frequencies lie far below the Nyquist frequency of the fundamental samplers.

Adding a fundamental rate sample and hold in a fast loop of such a system will have little effect on the stability margins. Results obtained by using this approximation should be used with caution however. Final results should be validated using the root locus method or by performing nonlinear simulation.

User-Defined Names

Standard components contain Easy5 defined “default” names for the parameter inputs, output states and output variables. The *default* names follow a strict naming convention. The full names are restricted to 60 alphanumeric characters, and some input/output names use special characters and port numbers.

This default naming convention is described in detail in “Component Input/Output Naming Convention” in the section on Components. This default naming convention is a useful feature that saves you time. You do not have to define every input/output name, it insures that all Easy5 names are unique, and the naming convention is used to perform automatic data connections between components

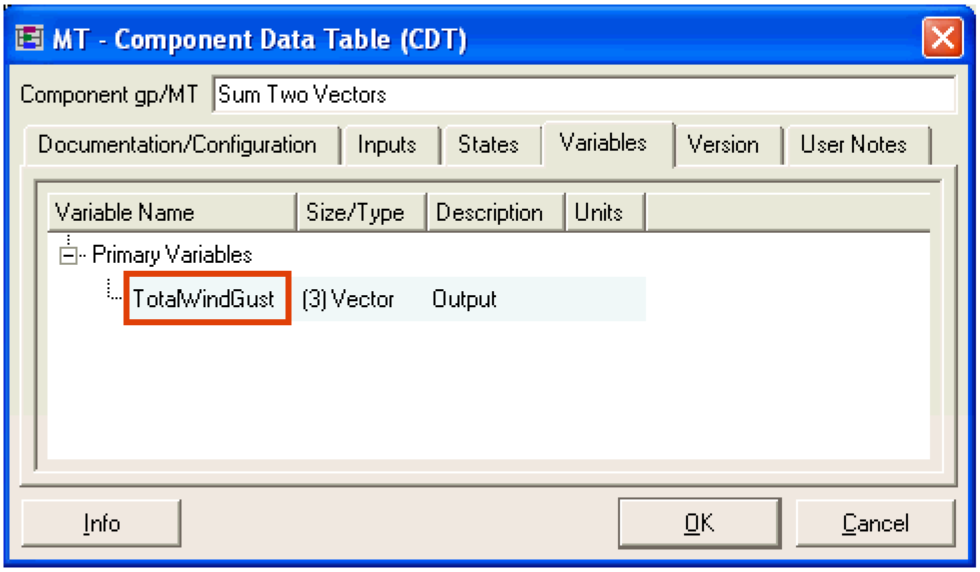
However, the default names may be renamed by the user to a *user-defined* name. User-defined names do not conform to the default naming convention, and the uniqueness of the names must therefore be maintained by the user. This section describes the methods used to define user-defined names, and how to use these names.

Note:	Fortran and C code component input and output names are always defined by the user, and are also considered to be user-defined names.

Defining User-defined Names

The component input/output names are renamed in the component data table. You can assign your own *user-defined* name to *all* component inputs and outputs. User-defined names "substitute" the Easy5 default names. Names can be renamed at anytime. The *user-defined* name automatically gets applied to every occurrence of the name in the model and data forms.

To change a name, just select the name and type in your own name, using up to 60alphanumeric characters. An example of a component data table with user-defined names is shown in Figure 170: User-Defined Name.


MT - Component Data Table (CDT)

Component gp/MT: Sum Two Vectors

Variable Name	Size/Type	Description	Units
Primary Variables			
TotalWindGust	(3) Vector	Output	

Figure 170 User-Defined Name

Use the following guidelines when defining user-defined names:

- Use mnemonic names (names that have meaning)
- Use up to 60 alphanumeric characters; however, very long names do not display well in the data tables and when printing and plotting out data. It is recommended that you define user define names with 20 characters or less characters.
- Do *not* use blank spaces; instead, use underscores
- Always start with an alpha character
- User-defined names are not case sensitive; that is, Pitch_cmd is the same as Pitch_CMD

The name you define in the component data table is automatically applied throughout the model and all data forms. The new name gets applied to connections, and is used in the executable source file.

Once a name has been user-defined, you may need to know the original Easy5 default name. To do this, hover over the user-defined name with your selector -- the original default name will appear in a "hover box" along with the user-defined name.

A user-defined name can be changed back to the original Easy5 default name by selecting the user-defined name, and deleting the name. Deleting a user-defined name automatically inserts the default name.

Caution:	Changing an input/output name alters the model. Therefore, you must always rebuild the executable model any time you change a name!

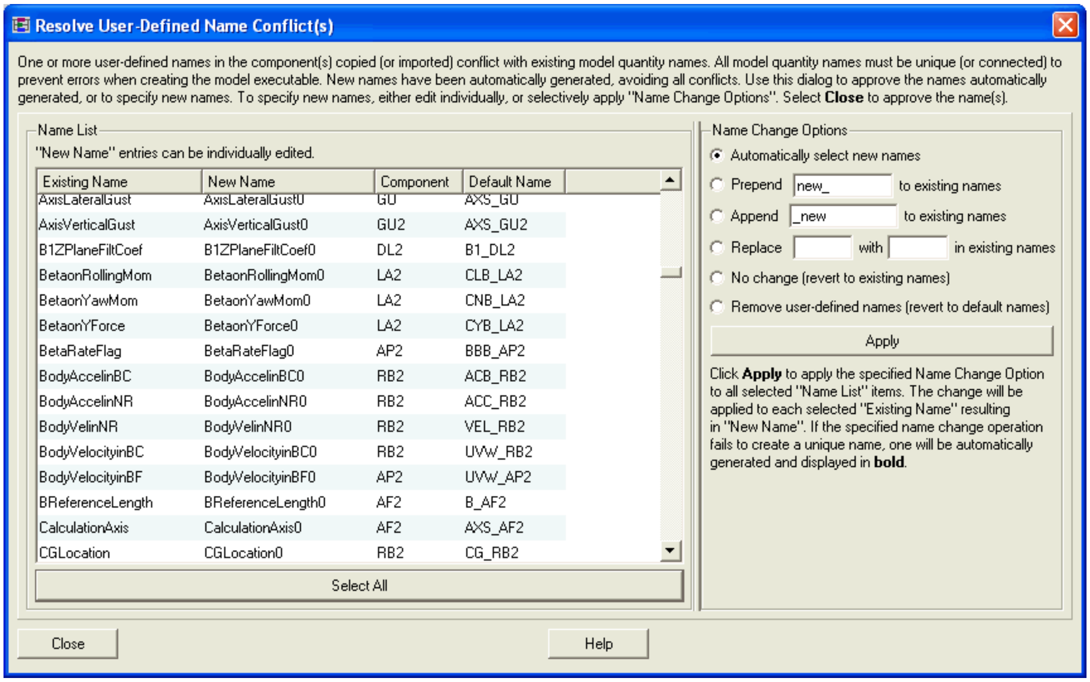
User-defined Name Menu Options

The **View** menu contains the **Show User-Defined Names** toggle button box. When this button is selected "on", the user-defined names are shown throughout the model. By default, this toggle button is on. If you wish to only show the Easy5 default names, check this button off. The Easy5 default names will be shown in all the data forms. This option can be checked on or off without effecting the model. In general, you want this option on at all times, but you may wish to temporarily check this option off to quickly view the Easy5 default names.

Resolving User-Defined Name Conflicts

When components are copied, the two character component identifier is automatically changed, thereby avoiding duplicate component input/output names. However, if the component contains user-defined names, when copied, the user-defined names must be changed to avoid a naming conflict. Easy5 automatically corrects such naming conflicts, but allows you to choose alternate names using a **"Resolve User-Defined Name Conflict(s)"** dialog window.

Figure 171 shows an example of such a dialog. The left side contains a table of names, with columns as “Existing Name”, “New Name”, “Component”, and “Default Nmae”. As this dialog is popped up, Easy5 has already resolved all conflicting names by automatically renaming them as shown.



One or more user-defined names in the component(s) copied (or imported) conflict with existing model quantity names. All model quantity names must be unique (or connected) to prevent errors when creating the model executable. New names have been automatically generated, avoiding all conflicts. Use this dialog to approve the names automatically generated, or to specify new names. To specify new names, either edit individually, or selectively apply "Name Change Options". Select **Close** to approve the name(s).

Name List

"New Name" entries can be individually edited.

Existing Name	New Name	Component	Default Name
AxisLateralGust	AxisLateralGust0	GU	AXS_GU
AxisVerticalGust	AxisVerticalGust0	GU2	AXS_GU2
B1ZPlaneFiltCoef	B1ZPlaneFiltCoef0	DL2	B1_DL2
BetaonRollingMom	BetaonRollingMom0	LA2	CLB_LA2
BetaonYawMom	BetaonYawMom0	LA2	CNB_LA2
BetaonYForce	BetaonYForce0	LA2	CYB_LA2
BetaRateFlag	BetaRateFlag0	AP2	BBB_AP2
BodyAccelinBC	BodyAccelinBC0	RB2	ACB_RB2
BodyAccelinNR	BodyAccelinNR0	RB2	ACC_RB2
BodyVelinNR	BodyVelinNR0	RB2	VEL_RB2
BodyVelocityinBC	BodyVelocityinBC0	RB2	UVW_RB2
BodyVelocityinBF	BodyVelocityinBF0	AP2	UVW_AP2
BReferenceLength	BReferenceLength0	AF2	B_AF2
CalculationAxis	CalculationAxis0	AF2	AXS_AF2
CGLocation	CGLocation0	RB2	CG_RB2

Select All

Name Change Options

[✓ Automatically select new names]

Prepend new_ to existing names

Append _new to existing names

Replace with in existing names

No change (revert to existing names)

Remove user-defined names (revert to default names)

Apply

Click **Apply** to apply the specified Name Change Option to all selected "Name List" items. The change will be applied to each selected "Existing Name" resulting in "New Name". If the specified name change operation fails to create a unique name, one will be automatically generated and displayed in **bold**.

Close Help

Figure 171 Resolve User-Defined Name Conflict(s) Dialog

You can offer alternate names by selectively selecting individual names (by row), and operations from the right-hand side (labeled "**Name Change Options**"). To apply a given operation to the selected names, use the **Apply** button. Continue to do this until you are satisfied with the list of names under the column "**New Names**" and select the **Close** button. A **Select All** button is provided to select all names in the table.

Changing User-Defined Names

You can change the user-defined names in one or more components by using the **Edit > Change Names** menu option. First select one or more components by drawing a selection box around the components. Then, select the **Edit > Change Names** menu item, or use the **Ctrl+W** keyboard shortcut. A window displays allowing you to change one or more of the user-defined names. An alternate method is, of course, to open each individual component, one at a time, and change the user-defined names. However, this method is much easier.

User Code component input and outputs output names are always defined by the user, and are also considered to be user-defined names. An example of a Change Names dialog resulting from selecting two components, one a User Code component, the other containing some user-defined names, is shown in Figure 172.

Let us look at the **Name Change Options** for this example.

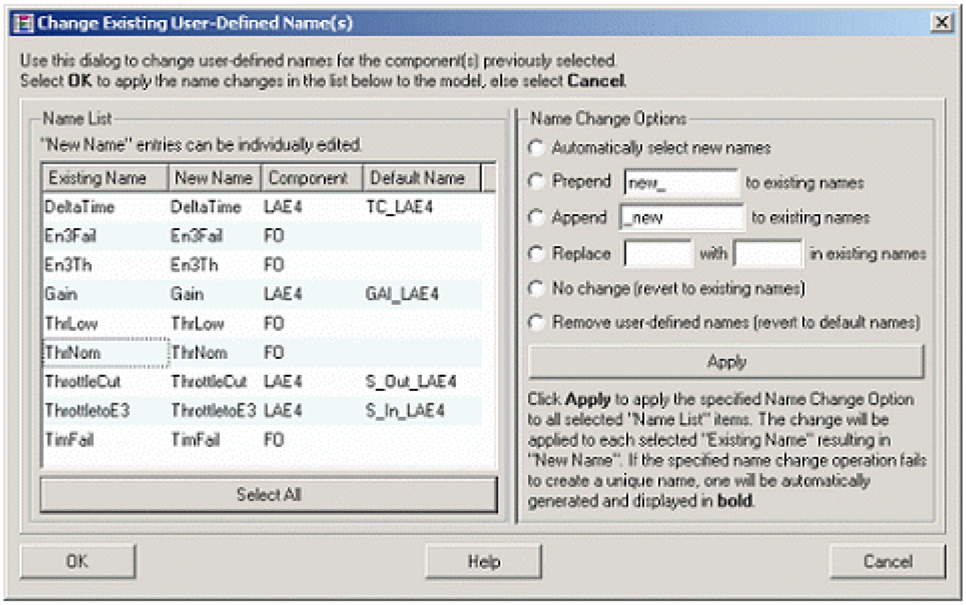
Automatically select new names: selecting this option will force Easy5 to automatically rename the user-defined names that appear in the window.

Prepend <name> to existing names: Selecting this option prepends the prefix entered in the "Prefix" data field to all selected names.

Append Suffix to existing names: Selecting this option appends the suffix entered in the "Suffix" data field to all selected names.

Replace <name> with <name> in existing names: performs a replacement operation on selected names.

Remove user-defined names (**revert to default names):** removes user-defined names for selected names, reverting them to their default Easy5 names.



Change Existing User-Defined Name(s)

Use this dialog to change user-defined names for the component(s) previously selected.
Select **OK** to apply the name changes in the list below to the model, else select **Cancel**.

Name List

"New Name" entries can be individually edited.

Existing Name	New Name	Component	Default Name
DeltaTime	DeltaTime	LAE4	TC_LAE4
En3Fail	En3Fail	FO	
En3Th	En3Th	FO	
Gain	Gain	LAE4	GAI_LAE4
ThrLow	ThrLow	FO	
ThrNom	ThrNom	FO	
ThrottleCut	ThrottleCut	LAE4	S_Out_LAE4
ThrottletoE3	ThrottletoE3	LAE4	S_In_LAE4
TimFail	TimFail	FO	

Name Change Options

- Automatically select new names
- Prepend new_ to existing names
- Append _new to existing names
- Replace with in existing names
- No change (revert to existing names)
- Remove user-defined names (revert to default names)

Click **Apply** to apply the specified Name Change Option to all selected "Name List" items. The change will be applied to each selected "Existing Name" resulting in "New Name". If the specified name change operation fails to create a unique name, one will be automatically generated and displayed in **bold**.

Figure 172 Change Names Dialog Example

References

The following references are referred to in this manual.

1. Patel, R.V., Sunswat V., and Fallside, F.; "A Method for Computing the Zeros of Transfer Functions in Linear Systems", Int. J. Systems Science Vol 8, No. 3, 1977
2. Athans F. and Falb, P.; Optimal Control, McGraw-Hill, 1966
3. O'Donnel, J.J.; "Asymptotic Solution of the Matrix Ricatti Equation of Optimal Control", Proceedings from the 4th Annual Allerton Conference on Circuit and System Theory, pp. 577-586, 1966
4. Meditch, J.S.; Stochastic Optimal Linear Estimation and Control, McGraw-Hill, 1969
5. Potter, J.E.; "Matrix Quadratic Solutions", SIAM Journal on Applied Math, Vol 14, No. 3, pp. 496-501, May 1966

6. Kalman, R.E., and Bertram, J.E.; "A Unified Approach to the Theory of Sampling Systems", Journal of Franklin Institute, p. 405, May 1959
7. Forsythe, Malcolm, and Moler; Computer Methods for Mathematical Computations, PrenticeHall, New Jersey 1977
8. Wilkinsons, J.H.; "The Algebraic Eigenvalue Problem"
9. Wilkinson, J.H., and Reinsch, C.; Handbook for Automatic Computation, Volume II: Linear Algebra, SpringerVerlag, Heidelberg 1971
10. Smith, B.T., et al.; Matrix Eigensystem Routines - EISPACK Guide, 2nd Edition, SpringerVerlag, Heidelberg 1976.
11. Gaed, J., Redish, K.A., and Brebner, M.A.; "Calculation of Eigenvalues of Real Matrices by the QR Method using Double QR Steps", Computer Journal, Volume II, pp. 112-115, 1968
12. Osborne, E.E.; "On Preconditioning of Matrices", J.A.C.M., Volume 7, pp. 338345, 1960
13. Gear, C.W.; Numerical Initial Value Problems in Ordinary Differential Equations, PrenticeHall, 1971
14. Doyle, J.C. and Stein, G.;"Multivariable feedback design concepts for a classical/modern synthesis", IEEE Transactions on Automatic Control, Vol. AC-26, Feb 1981
15. Dongarra, J.J. et al.; LINPACK User's Guide, SIAM, Philadelphia, 1979

Suggested Reading

16. Anderson, B.D.O., and Moore, J.B.; Linear Optimal Control, PrenticeHall, New Jersey 1971
17. Kuo, B.C.; Digital Control Systems, Saunders College, 2nd edition, 1992.
18. D'azzo, J.J. and Houpis, C.H.; Linear Control System Analysis and Design, McGraw-Hill, 1988.
19. Franklin, G.F., Powell, J.D. and Workman, W.L.; Digital Control of Dynamic Systems, Addison-Wesley, 2nd edition, 1990.
20. Bower, J.L. and Shultheiss, P.M.; Introduction to the Design of Servo Mechanisms, Chapter 10, John-Wiley & Sons, New York, 1958.
21. Franklin, G.F., Powell, J.D. and Emami-Naeini, A..; Feedback Control of Dynamic Systems, Addison-Wesley, 2nd edition, 1991.
22. Kuo, B.C.; Automatic Control Systems, Prentice Hall, 6th edition, 1991.
23. Shearer, J. L. and Kulakowski, B. T.; Dynamic Modeling and Control of Engineering Systems, Macmillan Publishing Company, 1990.

A Summary of Analysis Commands

- Overview 348
- Data Input Commands 348
- State Control Commands 348
- Operating Point Commands 349
- Eigenvalue Sensitivity Analysis 350
- Function Scan Analysis Commands 350
- Linear Model Analysis Commands 350
- Root Locus Analysis Commands 351
- Simulation Analysis Commands 352
- Stability Margin Analysis Commands 353
- Steady-State Analysis Commands 353
- Transfer Function Analysis 354
- Print Commands 355

Overview

This appendix summarizes the Easy5 language used to setup analyses. When you fill out any analysis data form using the graphical user interface, and launches the analysis, a special Easy5 analysis command file is created, and is referred to as the "anl" file. The file is named with the *.ezanl* extension as: *model_name.analysis_name.ezanl*. This file contains the analysis commands that are summarized in this appendix.

You will seldom need to "manually" create this file since it is done via the Easy5 graphical-user-interface. However, these analysis commands may be put into an auxiliary input file and used to modify an analysis. The auxiliary input file is most commonly used to input external data, as summarized in the first section "Data Input Commands".

This appendix contains only a brief summary of all commands used for loading model data, establishing operating points, executing analyses, and specifying plots. A complete description of analysis commands is published as an Easy5 Technical Note. This technical note is provided as a PDF file in the Easy5 Guide. To view the file, select **Help** > **Easy5 Guide** > **Technical Notes** > **Analysis Commands**.

Note:	For clarity, some of the commands are used more than once in a particular section. Square brackets indicate optional entries.

Data Input Commands

You load numeric values into the system model using the following command:

`PARAMETER VALUES`, **parameter=value, parameter=value,...** (Default=0.99999)

`INITIAL CONDITIONS`, **state=value, state=value,...** (Default=0.0)

`INITIAL VALUES`, **variable=value, variable=value**

Expressions containing parameter or state names, delimited by { }, also can be incorporated to PARAMETER VALUES and INITIAL CONDITIONS statements. See the Auxiliary Input File Data Format section for additional details.

For loading external auxiliary input data:

```
AUX INPUT = $label
```

To have your model calculate initial conditions:

```
CALC XIC
```

To signal that a “multiple analysis” is being executed, use the command:

```
MULT_ANAL
```

State Control Commands

To specifiy values for error controls:

ERROR CONTROLS, *state_name* = *value*(Default=0.001)

MULT INT ERROR BY, **factor**	*only affects continous states*
MULT SW ERROR BY, **factor**	*only affects switch states*
MULT ERROR BY, **factor**	*affects all states*

To "freeze" or activate specific states:

INT CONTROLS, **state_name = integer_value**	(0:frozen)
	(1:active=Default)
ALL STATES	(All states active)
NO STATES	(No states active)

To control the step size used by the Gear integration algorithm during its

Jacobian calculations:

GEAR PERTURBATION = value	(Default=0)

A value of zero means use the new method. To use the old method, use a value of 1. Any other value will use the old method, but adjust by that factor.

Operating Point Commands

To specify the value of time:

INITIAL TIME = **time** (Default=0.0)

To update and transfer operating points:

XIC-X	(Store state vector in XIC)
XIC1-XIC	(Store initial condition vector in XIC1)
XIC2-XIC	(Store initial condition vector in XIC2)

XIC-XIC1	(Retrieve XIC1 as initial condition vector)
XIC-XIC2	(Retrieve XIC2 as initial condition vector)

To save the current operating point:

SAVE XIC	(Save initial condition vector to an external file)
IC SAVEFILE=<file name>	(Names the external operating point file to *filename*)

Eigenvalue Sensitivity Analysis

To perform an Eigenvalue Sensitivity analysis:

EIGEN PARAMETER = **parameter_name**

EIGEN SENSITIVITY

Function Scan Analysis Commands

DEPEN = **dependent_var_name**
INDEP1 = **independent_var_name**

START1 = **starting_value**	(Default=-1.)
STOP1 = **final_value**	(Default=1.)

SCAN1

To perform a two-dimensional function scan:

DEPEN = **dependent_var_name**
INDEP1 = **first_independent_var_name**

START1 = **starting_value_for_INDEP1**	(Default=-1.)
STOP1 = **final_value_for_INDEP1**	(Default=1.)

INDEP2 = **second_independent_var_name**

START2 = **starting_value_for_INDEP2**	(Default=-1.)
DELTA2 = **increment_for_INDEP2**	(Default=0.)
CURVES2 = **number_of_curves**	(Default=1)

SCAN2

Linear Model Analysis Commands

For a simplified-form linear model:

LINEAR MODEL

For a partial or full-form linear model:

INPUTS = **input_names**

OUTPUTS = **output_names**

LINEAR MODEL

To calculate the eigenvectors (the columns of which constitute the modal matrix), you must specify the following command in your Model Description file:

THREE WORK ARRAYS

For reduced-order linear models (which require calculation of the eigenvectors):

INPUTS = **input_names**

OUTPUTS = **output_names**

MODEL ORDER = **order**

REMOVE MODEL MODES = **mode1,...**

REAL = **mode,...**

COMPLEX = **mode1,mode2**

LINEAR MODEL

To save linear model output data:

LM SAVEFILE=<file name>	(Names the linear model output file to *file name*)	
LM EV FLAG=<n>	(Eigenvalue/Eigenvector calculation flag, where n is:	
	0 = Eigenvalues & Eigenvectors	1=Eigenvalues
		2=Neither)

Root Locus Analysis Commands

To perform a Root Locus Analysis:

RL PARAMETER = **root_locus_ parameter_name**

RL START = **start_value**	(Default=0.)
RL STOP = **stop_value**	(Default=1.)
RL POINTS = **integer_value**	(Default=6)

ROOT LOCUS

To calculate root locus zeros as well:

RL PARAMETER = **root_locus_ parameter_name**

RL INPUT = **input_name**

RL OUTPUT = **output_name**

RL START = **start_value**	(Default=0.)
RL STOP = **stop_value**	(Default=1.)
RL POINTS = **integer_value**	(Default=6)
ROOT LOCUS	

To specify formatting of root locus data:

RL AUTO SCALES	(Default)
RL MANUAL SCALES	
REAL MIN = **min_value**	(Default=-10.)
REAL MAX = **max_value**	(Default=0.)
IMAG MIN = **min_value**	(Default=0.)
IMAG MAX = **max_value**	(Default=10.)
S PLANE	(Default)
Z PLANE	

Simulation Analysis Commands

An optional command used to specify a particular "runid":

RUN IDENTIFIER = **name**

Before requesting a simulation to be executed you must specify the following:

TMAX = **max_simulated_time**	(Default=1.)
TINC = **value**	(Default=1.)
INT MODE = **integration_mode**	(Default=1 or BCS GEAR)
INITIAL TIME = **time**	(Default=0.)

To specify the simulation output data:

OUTRATE = **plot_rate_multiplier**	(Default=1)
PRATE = print_rate_multiplier	(Default=1)
PRINT CONTROL = integer_value	(Default=3)
PRINT VARIABLES = variable(s)	
SI AUTO SCALES	(Default)
SI MANUAL SCALES	(w XRANGE, YRANGE commands)

To control activation of interactive simulation (IS) widgets:

DEACTIVATE IS = value	((0: all, 1:TI only, or 2:none)

To specify what the plots will look like:

DISPLAY[**i**],[OVERPLOT,]**yname**,[VS,**xname**][,XRANGE=**min,max**,YRANGE=**min,max**]

where: **i**=1,2,3,...,2000 (Default **xname** is TIME)

To specify secondary print or plot rates:

PRINT2 FROM,**t1**,TO,**t2**	(Default: **t1**,**t2**=10^{35})
OUTRATE2 = **secondary_ plot_rate_multiplier**	(Default=OUTRATE)
PRATE2 = **secondary_ print_rate_multiplier**	(Default-PRATE)
PRINT2 = **integer_value**	(Default=PRINT CONTROL)
TINC2 = **secondary_time_increment_value**	(Default-TINC)

To specify how switch state transitions are plotted:

PLOT EVENT = 0	(No data saved at switch state transitions)
PLOT EVENT = 1	(Data saved at all switch state transitions)
PLOT EVENT = 2,**begin**,TO,**end**	(Data saved only for **begin** < time < **end**)

To execute a Simulation analysis:

SIMULATE

To update the current operating point with the end point from a simulation analysis:

XIC-X

Stability Margin Analysis Commands

To perform a Stability Margin analysis:

SM PARAMETERS = **parameter(s)**

STABILITY MARGINS

Steady-State Analysis Commands

To perform a Steady-State analysis:

SS ITERATIONS = **max_number_of_iterations**	(Default=100)

STEADY STATE

To update the current operating point with the results from a SteadyState analysis:

XIC-X

To perform a Steady-State Scan analysis:

SS PARAMETER = **parameter _ name**

SS START = **starting_value**	(Default=-1.)
SS STOP = **final_value**	(Default=1.)
SS POINTS = **number_of_ points**	(Default=5)
SS ITERATIONS = **max_number_of_iterations_ per_scan**	(Default=30)

STEADY STATE

To specify scaling modes:

SS AUTO SCALES	(Default)
SS MANUAL SCALES	(w XRANGE, YRANGE commands)

To specify Steady-State Scan analysis plots:

DISPLAY[**i**],[OVERPLOT,]**yname**,[VS,xname][,XRANGE=**min,max**,YRANGE=**min,max**]

where: **i**=1,2,3,...,2000 (Default **xname** = SS PARAMETER)

Transfer Function Analysis

To perform a Transfer Function analysis:

TF INPUT = **input**

TF OUTPUT = **output**

To specify scaling:

TF AUTO SCALES	(Default)
TF MANUAL SCALES	(w FREQ MIN, FREQ MAX commands)

FREQ MIN = **min_freq**

FREQ MAX = **max_freq**

To specify plot format:

BODE	(Default)
NICHOLS	
NYQUIST	

Plot Commands

To specify plot generation:

PLOT ON	[SINGLE | DOUBLE]	(Default = DOUBLE)
PLOT OFF	(Default)	

PRINTER PLOTS

CSV PLOTS

ESA PLOTS

Plots are generated using double-precision data by default. To generate single precision data, use PLOT ON SINGLE.

To specify use of symbols during plotting:

OMIT PLOT POINTS

PLOT POINTS

To specify plotting of tabular data:

PLOT ALL TABLES

PLOT TABLES = **table_name**(s)

To specify simulation or steady-state scan displays:

DISPLAY[**i**],[OVERPLOT,]**yname**,[VS,**xname**][,XRANGE=**min,max**,YRANGE=**min,max**]

where: **i**=1,2,3,...,2000 (Default **xname** = TIME (SIMULATION)

SS PARAMETER" (STEADY STATE))

Note:	XRANGE, YRANGE commands require use of the SI MANUAL SCALES (or SS MANUAL SCALES for STEADY STATE) command.

Print Commands

To specify print generation, first set the print control flag:

PRINT CONTROL = *value*

where: *value* =

0 None
1 All states, rates, and time
2 All states, rates, variables, and time
3 All states, rates, variables, and time (and parameters at time = 0) (Default)
4 All states, rates, variables, and parameters
5 Time and the quantities specified via PRINT VARIABLES command (see below.)
6 All states, rates, variables, and parameters at each STEADY STATE iteration
7 All states, rates, variables, parameters, and system Jacobian matrix at each STEADY STATE iteration
8 User-furnished PRINT STATEMENTS

To specify which variables get printed:

```
PRINT VARIABLES = name1, name2, name3, ..., name40
```

To generate a printout of current values:

```
        PRINT
```

B Guide to Numerical Integration

- Overview 358
- Numerical Stability 358
- Accuracy and Error Control 361
- Integration Method Selection Guidelines 364

Overview

Easy5 provides several different numerical integration methods. These integrators are listed below, where INT MODE is the Easy5 analysis command name assigned to the integrator.

- BCS-Gear (default integrator): an improved version of Stiff Gear (INT MODE=1)
- Runge-Kutta: a fourth order, variable-step Runge-Kutta class method (INT MODE=2)
- Huen: a second order fixed-step implicit method (INT MODE=3)
- Euler: a first order fixed-step explicit method (INT MODE=4)
- Adams: an automatic step-size/order-selection method, using AdamsBashforth predictor/Adams-Moulton corrector pairs - 2nd through 12th order (non stiff option of Gear) (INT MODE=5)
- Stiff Gear: an improved version of Gear's original method (INT MODE=6)
- Radau54: variable step three stage fifth order implicit Runge Kutta used to solve implicit models (INT MODE=7)

Fixed-Step Runge-Kutta: fixed-step fourth order method (INT MODE=8)

Several integration methods are provided, as no single method is appropriate for the wide range of differential-equation characteristics that are encountered in dynamic analysis models. The choice of the best integration method depends on a number of considerations.

User requirements for accuracy, resolution and duration of transients, and model characteristics, such as natural frequencies, discontinuities, disturbances, and sampling, all must be considered. Many excellent texts exist on the numerical integration of ordinary differential equations. This appendix helps the user in selecting the integration options provided by Easy5.

The following two sections provide some background on the problems of numerical stability, accuracy, and error control. These sections discuss the topics from the view of how they apply to the integration options available in Easy5. The third section provides some general guidelines to the selection of an integration option based on the user's requirements and the characteristics of the model. The last section describes the linear simulation option in greater detail.

Note:	In some cases an equation reference number (e.g., Equation B-1) appears in parentheses along the right margin, next to the equation. This reference number is cited in text when it is necessary for the reader to refer back to an equation for additional insight.

Numerical Stability

The numerical integration of the equations of motion for a dynamic system is a discrete dynamic process. The numerical integration process has the potential of being unstable regardless of the true behavior of the system being modeled. A necessary condition for a satisfactory simulation of a dynamic system is that the numerical integration process be stable.

However, just as with control system design, several other conditions must be met for a satisfactory dynamic response. These other conditions are discussed in the following sections: Accuracy and Error Control and Integration Method Selection Guidelines.

All of the integration methods available in Easy5 are "conditionally stable." That is, if the integration step size, h, is made small enough, each method will produce stable approximations to the true solution. In practice, the step size, h, required by some of the methods for stability may be so small that a very large number of model evaluations are required to calculate a transient response.

The stability of numerical integration methods can be conveniently analyzed for most dynamic systems, by considering a linear approximation of the model. If the model is approximated by the linear equation:

$$\dot{X} = \lambda X \qquad \text{(B-1)}$$

where λ is the eigenvalue of the model, the stability of the various integration methods can be related to the product of the step size and the system eigenvalue, hλ.

Figure 1 shows the stable regions in the hl plane for the fixed-step size integration methods available in Easy5. For real eigenvalues, the stability limits of the two fixed-step integration methods are the same.

For complex eigenvalues, the second order Huen method is stable for slightly larger step sizes than the first order Euler method. For very lightly damped systems with eigenvalues near the imaginary axis, either integration method requires very small values of *h* to maintain stability.

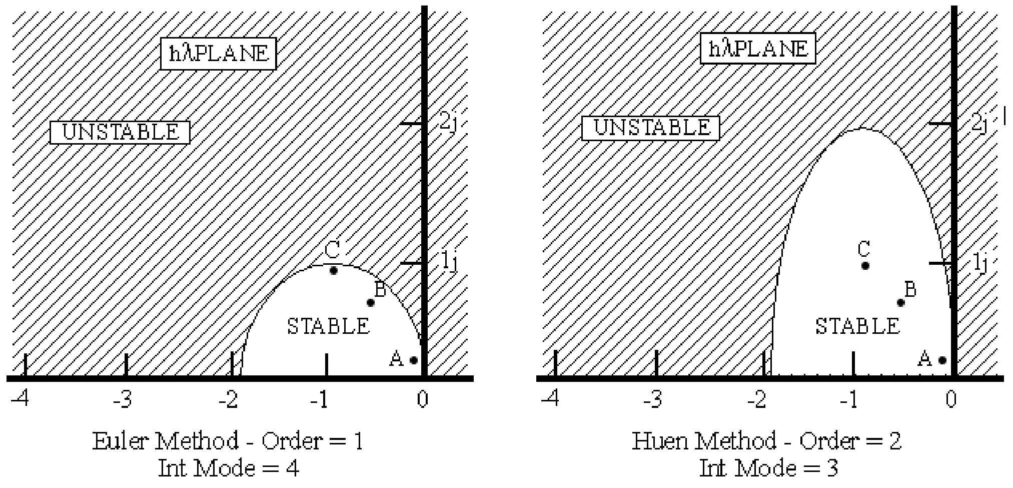


Figure 1 Stability Regions (Fixed-Step Methods)

A demonstration of the effect of step size is shown in the following figure, Figure 2.

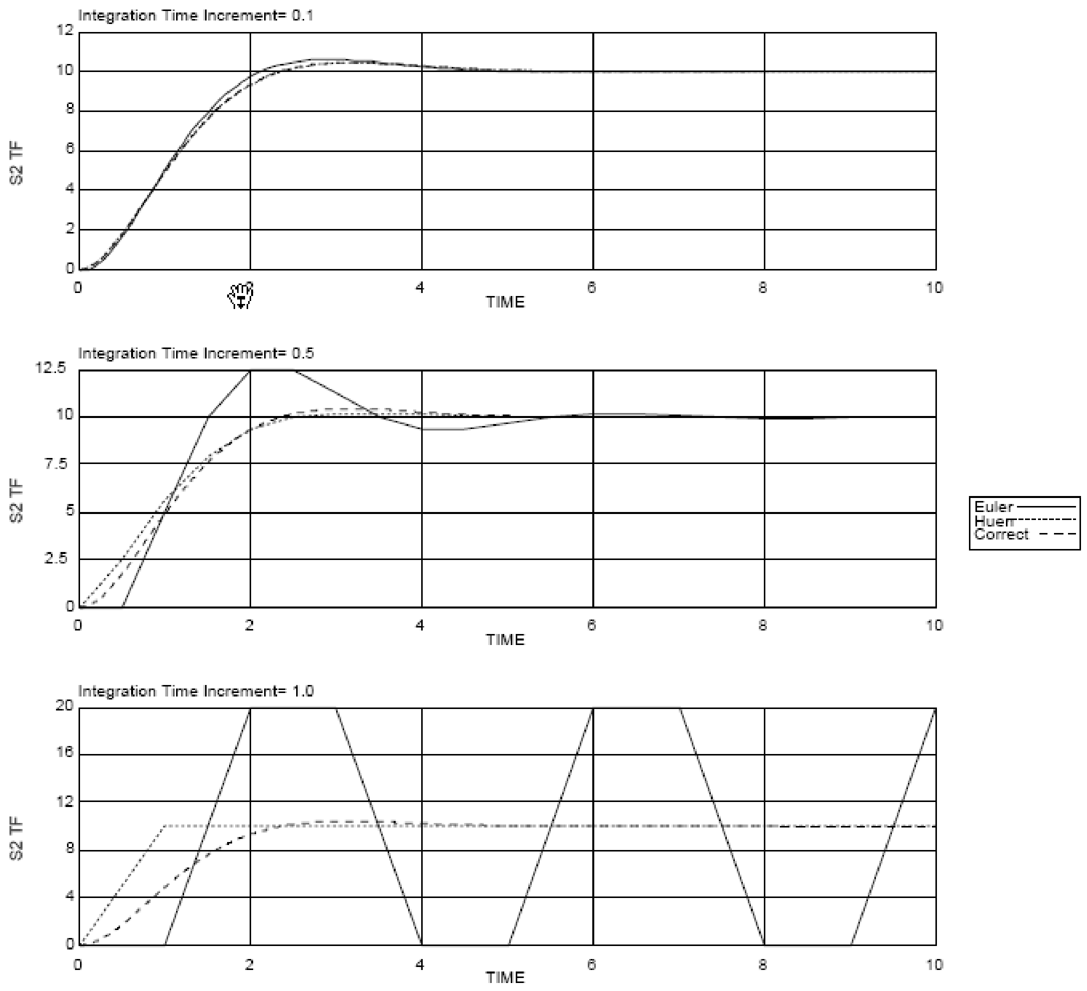


Figure 2 Comparison of Euler and Huen Integrator Response

In this case, the response of a second order linear system to a step of amplitude 10 is calculated with each of the fixed-step integration methods for different step sizes.

The system transfer function is:

$$T(s)= \frac{2}{s^2 + 2s + 2}$$

which has eigenvalues at -1 $\pm$ j. Step sizes, *h*, of .1, .5, and 1 are used.

These result in points A, B, and C in the stability regions. At a step size of .1, both Euler and Huen methods provide a reasonably accurate solution.

At a step size of .5, which is well within the stability region of each method, the Euler result has departed considerably from the correct solution and the Huen method has a slight error. With a step size of 1, the Euler method is on the stability boundary and produces an oscillating result, while the Huen method is accurate only in its final value.

Figure 3 shows the stable regions for two of the variable-step integration methods in Easy5. The stability regions for fixed orders are shown for both methods, although the Stiff Gear method automatically varies the order as well as the step size.

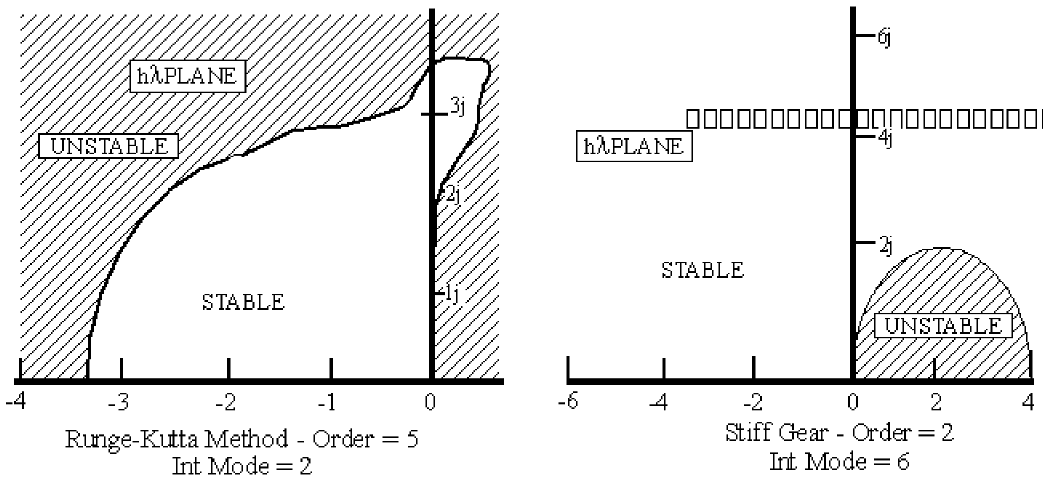


Figure 3 Stability Regions (Variable Step Methods)

The significant difference between the stability regions of these methods and the fixed-step methods is the enlargement of the stable region, especially near the imaginary axis. However, the fact that the time step, *h*, of these methods automatically adjusts to maintain a specified accuracy is of much more practical significance.

The system eigenvalues of many practical problems change as the system states vary. With a fixed-step integration method, the step size must be selected to keep the highest value of the system eigenvalues well within the stability region. The small step size required to accomplish this must be used for the entire transient. This small step size holds true even though the highest eigenvalues may occur only for a brief period, such as when some limit or other nonlinearity is encountered in the model.

The fixed-step integration methods are used when the required resolution of model features, such as sampling, forces the use of very small time steps. If the time steps are small enough to place the product of the step size and the model's eigenvalues well within the stability region of a fixed-step method, then the fixed-step methods will usually require the least amount of computer time.

Accuracy and Error Control

The step size demonstration shown in Figure 3 illustrates that fixed step integrators with a step size well within their stability region may produce stable results that are rather poor approximations of the correct solution.

For some models, the step size required for acceptable accuracy can be orders of magnitude less than that required for stability. The problem of selecting a fixed-step size can be avoided by using one of the variable-step size integration methods available in Easy5.The step sizes for these methods are selected by the integration algorithm as the integration proceeds.

The step size is selected to meet a given error criterion and maintain numerical stability. These "adaptive" methods estimate the local truncation error at each step of the integration, accept or reject the approximation, and predict the next step size to be tried. Local truncation error can be loosely thought of as the error incurred during one step of the integration process, given that all previous approximations are exact. The order of a method is a crude measure of accuracy. A method is said to be of order p if it is exact for p*th* order polynomials.

The adaptive Easy5 integrators (BCS-Gear, Runge-Kutta, Adams, and Stiff Gear) strive to keep the step size small enough to insure reasonable local error, which in turn should produce a small global error. Whether or not the global error is small depends both on the model and the stability of the method.

The adaptive integrators measure the local truncation error by comparing two estimates of the solution that theoretically differ only in high order terms from the Taylor's expansion of the solution over the current step. The details of how this is done in each method are not important here, except as to how it relates to the Easy5 integration controls. The array, ERROR(I), is a measure of significance of the corresponding I*th* state of the system. To be precise, ERROR(I) is a value below which the I*th* state is in some sense considered negligible by the BCS-Gear, Runge Kutta, Adams, and Stiff Gear integrators. There are two techniques of error control employed by these four integration methods. Runge-Kutta is described first. The error control in BCS-Gear, Adams, and Stiff Gear is basically the same and will be discussed second.

In Runge-Kutta the initial step size, H_0, is chosen as a function of TINC.

$$H0=.01 * TINC \qquad \text{(B-2)}$$

Subsequent step sizes are selected on the basis of local error-control estimates. There are a number of refinements in Runge-Kutta that will not be discussed.

The basic error control is governed by the following quantity, Q:

$$Q = \overset{\text{MAX}}{(I)} \left[\frac{\text{LTE (I)}}{\text{ERROR (I) + X(I) *ERROR(I)}} \right] \qquad \text{(B-3)}$$

where LTE(I) is the local truncation error estimate for the I*th* state of the solution as calculated by comparing a 4th order solution to a 5th order solution, X(I) is a recent history size measure of the I*th* state (initially set to the initial value), and ERROR(I) is the user input error control. The integrator strives to make Q = 1. If Q < 1, the step size on the next integration step is increased. If Q > 10, the current step is rejected and a new smaller step size is calculated for another attempt. To interpret the effect of the input error controls, ERROR(I), set Q = 1 (the desired value for Q) in (B-3) and solve for LTE(J) for the maximal choice of I. That is, for some I, if Q=1, then

$$\text{LTE (I)} = \text{ERROR (I)} + \left| \overline{X}(I) \right| * \text{ERROR (I)} \qquad \text{(B-4)}$$

i.e., the LTE is close to the ERROR + X*ERROR. If X(I) has been small, ERROR(I) dominates the right hand side of (B-4), and the error control is essentially absolute error. On the other hand, if X(I) is very large, X(I)*ERROR(I) will dominate, and thus relative error is controlled. If the solution gets large, then log10(ERROR) will roughly give the number of significant digits of accuracy (locally).

The use of error control differs for the BCS-Gear, Adams, and Stiff Gear integrators. A local truncation error, LTE, is computed by the integrator. The Euclidean error is controlled, i.e.,

$$\sum_{I=1}^{N} \left[\frac{\text{LTE (I)}}{\text{XMAX (I)}} \right]^2 \qquad \text{(B-5)}$$

is required to be less than $(\text{EPS})^2$ where N is the number of states, XMAX(I) is the maximum of the I*th* component of X over the course of the integration. The user impacts this control by effecting the initialization of XMAX(I) and the choice of EPS. EPS is chosen as follows:

$$\text{MIN}_{(I)} \ \text{(ERROR (I))} \qquad \text{(B-6)}$$

with the constraint that EPS£.01. If ERROR(I)<1.E-12 for all I, then EPS is set to 1.E-4. The initialization of XMAX(I) is given by

XMAX(I) = ERROR(I) /EPS (B-5)

IF (XMAX(I).EQ.0) XMAX(I) = 1.

The net effect of this initialization for the EPS and the XMAX array result in the ERROR array being used in a manner similar to its use in Runge-Kutta. For example, if EPS = .001 and ERROR = .001, then XMAX = 1.0 and error control is essentially absolute error until the solution X(I) exceeds 1. If X(I) grows, the error processing will gradually become relative error since XMAX is set equal to X whenever X exceeds it. If the solution grows to a maximum value, and then decays, the error control will be relative to that maximum. The user must remember that EPS is set by the smallest ERROR(I).

Thus, in a two-component system, if ERROR(1) = .001 and ERROR(2) = 1.0, the resulting controls will be as follows:

EPS=.001; XMAX(1)=1; XMAX(2)=100.(B-6)

If X(1) = X(2) = 0 initially, the integrator considers values less than 0.001 negligible for X(1) and values less than 1.0 negligible for X(2). This is quite similar to what Runge-Kutta would do with these same inputs for ERROR(1) and ERROR(2).

To summarize, when using any of the variable-step size integration methods, the user should provide reasonable values of error controls for each state in the system model. The default values of.001 provided by Easy5 may be much too large or much too small, depending on the units of measure used in the system model.

Too small an error control, especially for states that represent accelerations or other higher order derivatives, may result in excessive computer time. Too large an error control may result in inaccurate and unstable results. For state values much greater than one, the ERROR values control the relative error in the results. For state variables much less than one, the ERROR values control the absolute error.

Integration Method Selection Guidelines

In Easy5, the effect of integrator step size and integration method can be quickly evaluated. This is done by repeating a typical simulation run with different integration methods, error controls, or step sizes. Such an evaluation should be made before expending a large amount of computer time on simulation studies of an unfamiliar model. Easy5 provides the BCS-Gear method, as a default. This method was chosen since it is very good with stiff systems, and since stiff systems are quite common in the areas to which Easy5 has been applied.

It is good practice to use the linearized analysis capabilities of Easy5 to investigate model stability and characteristics before attempting simulation.

The following are some general integration method selection guidelines:

1. If no special knowledge is available about the system, try the variable step Runge-Kutta method. This method usually performs quite adequately on a broad range of problems. However, it will probably be worthwhile to consider switching to another method when more is known about the system, since Runge-Kutta is by no means the most efficient method.
2. If a large amount of output is desired with small time increments, the Adams or Stiff Gear methods should be considered. These methods use interpolation rather than generating smaller time steps when output points are smaller than current step sizes. However, these methods are adversely affected by discontinuities such as sampling, nonlinearities in the model, or external disturbances to the model.
3. If the model involves discrete dynamic elements (sampled data), small time steps will be required for the entire duration of the run. In these circumstances, it's more cost-effective to use one of the fixed-step methods because they require less computing per step. However, with these methods, the user must specify the size of the time step increment (TINC) and is responsible for insuring that the integration algorithm remains stable.
4. If the model contains non differentiable nonlinearities (e.g., coulomb friction, deadband, etc.) not implemented using switch states, the Runge-Kutta method is recommended. With the Runge Kutta integrator, care should be taken if a large number of output data points at small time increments is desired. If the method is forced to reduce the step size in order to accommodate the data requirements, it could become significantly more costly than a fixed-step method.
5. If the problem is stiff (i.e., there is a large spread in eigenvalues where the high frequency eigenvalues are well damped), either BCS-Gear or Stiff Gear is recommended. Both of these methods employ the same basic algorithm. However, BCS-Gear makes use of state-of-the-art numerical techniques which increase its efficiency.
6. If the system has high frequency eigenvalues that are lightly damped (i.e., flexible modes), then the Adams method is recommended.

7. If the system is defined as an implicit model, then you must select the Radau54 integration method, and prior to building the executable model, you must turn on the implicit solver by selecting: **Build > Solve Implicit Loops**. For complete information on implicit modeling, see the User's Guide, Chapter 13: Implicit Modeling

It should be noted, however, that problems with large eigenvalues (with negative real parts) do not automatically indicate that one should use Stiff Gear. For example, consider the following system:

$$\begin{aligned} \dot{X}_1 &= -X_1 \\ \dot{X}_2 &= -1000X_2 \end{aligned} \qquad \text{for time } 0 \le t \le \text{TMAX} \qquad \text{(B-7)}$$

This is an uncoupled system (and might seem artificial), but coupled systems often display the behavior of rapidly damping components such as X_2. If one was integrating this model and the important variable was X_1 and TMAX was large, then a large step size could be used provided the numerical integration of X_2 was damping to zero (i.e. stable). In such a case, the Stiff Gear method would be appropriate. On the other hand, if TMAX was small (e.g., TMAX = 0.0001) and X_2 was the component of interest (where relative accuracy is important), then an efficient integrator such as Adams, or perhaps Runge-Kutta, would be appropriate. Thus, the decision to use Stiff Gear depends on both the user requirements for accuracy and the eigenvalues of the system.

C

Discrete Analysis Techniques

- Overview 368
- Linear Sampled-Data System Equations 368
- Discrete System Transition Matrix 371
- Total Transition Matrix for a Single-Rate System 372
- A Single-Rate Example 374
- Total Transition Matrix for a Multi-Rate System 376

Main Index

Overview

Easy5 can perform nonlinear simulation and linear analyses on sampled data systems containing up to ten integer-related sampling rates. The linear sampled-data analysis options are based on the state-space approach described by Kalman and Bertram. This approach requires the non-linear model be linearized and then transformed into the following discrete state space form:

$$\mathbf{x}_{i+1} = \mathbf{A}_z\mathbf{x}_i + \mathbf{B}_z\mathbf{u}_i$$

$$\mathbf{y}_i = \mathbf{C}_z\,\mathbf{x}_i + \mathbf{D}_z\mathbf{u}_i$$

where:

$\mathbf{x}_{i+1}$ = the vector of the continuous and delay state values at the i^{th} slowest sample time

$\mathbf{u}_i$ = the vector of inputs at the i^{th} slowest sample time

$\mathbf{y}_i$ = the vector of outputs of the i^{th} slowest sample

$\mathbf{A}_z$, $\mathbf{B}_z$, $\mathbf{C}_z$ and $\mathbf{D}_z$ = discrete stability, input, output and feed through matrices

The method of calculation of the $\mathbf{A}_z$, $\mathbf{B}_z$, $\mathbf{C}_z$, and $\mathbf{D}_z$ matrices is the subject of this appendix.

Linear Sampled-Data System Equations

A discrete system may be described by the following three types of states:

- Continuous states
- Delay states
- Sample-and-hold states

State Type . . .	Are Defined By . . .	State Value is Set . . .
Continuous states	First-order ordinary differential equations and vary continuously as a function of time.	N/A
Delay states	First order difference equations, which are evaluated only at appropriate sample times. Between sample times the values of these states are constant.	Equal to the result of the difference equation evaluation after a delay of one sample period.
Sample-and-hold states	First order difference equations, which are evaluated only at appropriate sample times. Between sample times the values of these states are constant.	Equal to the result of the difference equation evaluation as soon as its made.

Figure 1: Examples of Continuous, Delay, and Sample States shows an example of each state type

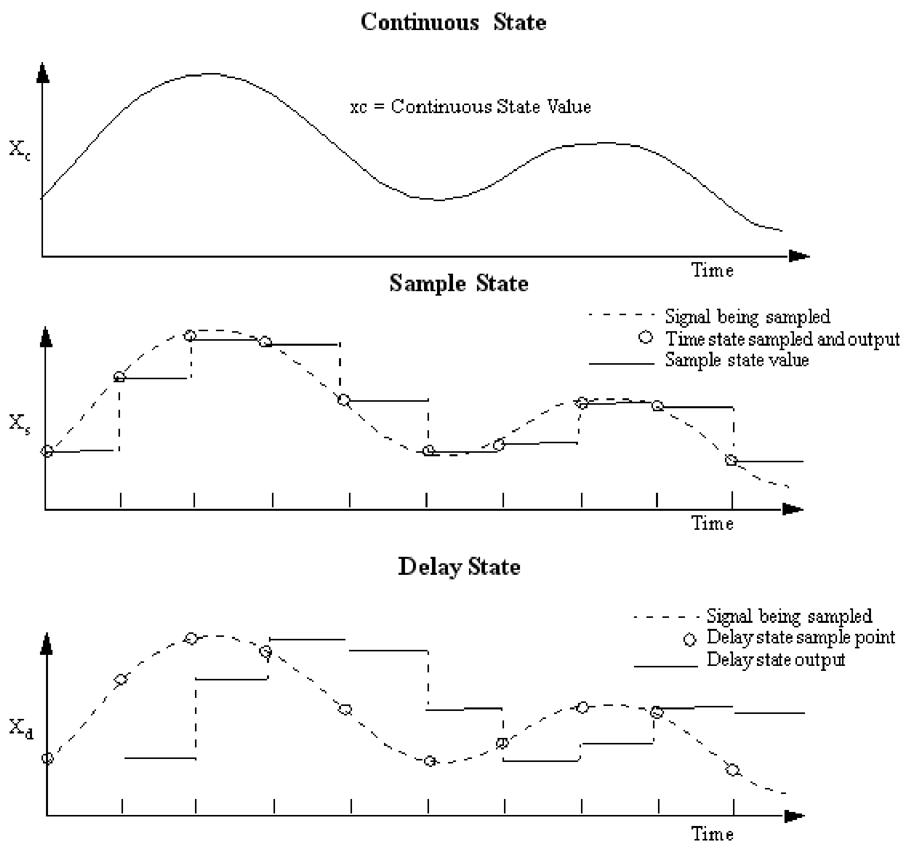


Figure 1 Examples of Continuous, Delay, and Sample States

The Kalman-Bertram method of analysis requires that the sampled-data system be "transitioned" over one "fundamental" (slowest) sample period. This is done by transitioning the continuous part of the model between sample times and transitioning the discrete part of the model across sample times. The continuous system transitioning is accomplished by a closed form integration of the linearized differential equations. The discrete system transition across a sample time is determined by the difference equations governing the delay and sample-hold states.

In order to guarantee this separation of the continuous and discrete transitions, Easy5 requires that a sample-hold state be inserted between delay states and continuous states. These "output" hold states are used to form the discrete system output equations.

The final total system transition matrix is the coefficient matrix for the set of linear equations which defines the values of all states just after a fundamental rate sample time in terms of the states just after the previous fundamental rate sample time. Since all samplers are assumed to sample at a fundamental sample time, the values of all states just after a fundamental sample time can be expressed completely in terms of the continuous states and the delay states.

Thus the sample-hold state rows and columns can be eliminated from the final transition equations. The discrete stability of the system is completely determined by the continuous and delay state transition equations.

Let the continuous, delay, and sample-and-hold states be grouped together as three state vectors:

x_c = n_c vector of continuous state values

x_d = n_d vector of delay state values

x_s = n_s vector of sample and hold state values

The total system state vector of dimension $n_c + n_d + n_s$ is formed into the single partitioned vector:

$$x = \begin{bmatrix} x_c \\ x_d \\ x_s \end{bmatrix}$$

The changes that occur between sample times affect only the continuous states. During this time, the derivatives of the delay states, sample-hold states and inputs are zero and the system differential equations can be written:

$$\dot{\mathbf{x}}_c = Accxc + 0\ xd + Acs\ \ xs\ + Bcu \qquad \text{(Equation C-1)}$$

$$\dot{\mathbf{x}}_d = 0$$

$$\dot{\mathbf{x}}_s = 0$$

$$\dot{u} = 0$$

If we define an augmented state vector by:

$$x_t = \begin{bmatrix} x_c \\ x_d \\ x_s \\ u \end{bmatrix} \qquad \text{(Equation C-2)}$$

then we can write (C-1) as the matrix equation:

$$\dot{x}_t = A_t x_t \qquad \text{(Equation C-3)}$$

where:

$$A_t = \begin{bmatrix} A_{cc} & 0 & A_{cs} & B_c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(Equation C-4)

The solution of (C-3) is:

$x_t(t^-_n) = e^{A_t \tau_o} x_t(t^+_{n-1})$

where:

$x_t(t^+_{n-1})$ = the augmented state vector values just after the (n-1)st sample time

$x_t(t^-_n)$ = the augmented state vector values just before the nth sample time

$\tau_o = t_n - t_{n-1}$ = time between successive samples

If $\mathbf{\Phi} = e^{Atto}$, then the structure of $\mathbf{A}_t$ implies that $\mathbf{\Phi}$ has the form:

$$e^{A_t t_0} = \begin{bmatrix} e^{A_{cc}\tau_o} & 0 & Ecs & Eu \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

(Equation C-5)

Numerical Calculation of Continuous-System Transition Matrix

The matrix exponential $\mathbf{\Phi}$ is calculated using the 3rd order Padé approximation:

$$\Phi = e^{A_t \tau_o} \approx \left[I - \frac{\tau_o}{2}A_t + \frac{\tau^2_o}{10}A^2_t - \frac{\tau^3_o}{120}A^3_t\right]^{-1} \bullet \left[I + \frac{\tau_o}{2}A_t + \frac{\tau^2_o}{10}A^2_t + \frac{\tau^3_o}{120}A^3_t\right]$$

(Equation C-6)

Since powers of $\mathbf{A}_t$ have the same block structure as $\mathbf{A}_t$ only the blocks of $\mathbf{\Phi}$ corresponding to non-zero blocks of $\mathbf{A}_t$ are computed.

Discrete System Transition Matrix

At sampling instants, the system behavior is described by a system of 1st-order difference equations. Since the continuous states and the inputs are unaffected by the sampling, these equations may be written in matrix form as:

$x_t(\tau+) = Tx_t(\tau-)$

where:

$x(\tau+)$ = the augmented state vector after the sample

T = the transition matrix

$x_t(\tau-)$ = the augmented state vector before the sample

The matrix **T** will have the following block structure:

$$T = \begin{bmatrix} I & 0 & 0 & 0 \\ T_{dc} & T_{dd} & T_{ds} & T_{du} \\ T_{sc} & T_{sd} & T_{ss} & T_{su} \\ 0 & 0 & 0 & I \end{bmatrix} \quad \text{(Equation C-7)}$$

The elements of T are calculated by perturbing each state or input and forming the quotient of the change in rates to the size of perturbation.

Note that at the initial sampling time, all samplers sample; this implies that the state values after the sample can all be written in terms of the continuous states, delay states and inputs. Thus, for the initial sample-time transition matrix, the blocks $\mathbf{T}_{ds}$ and $\mathbf{T}_{ss}$ are zero.

Total Transition Matrix for a Single-Rate System

For a single-rate system, to transition the system through a complete cycle requires that we transition through a sample and then transition the continuous part up to the next sample time. This transition is expressed in matrix form as:

$$\psi = \Phi T_O = \begin{bmatrix} e^{A_{cc}\tau_o} & 0 & E_{cs} & E_u \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix} \bullet \begin{bmatrix} I & 0 & 0 & 0 \\ T_{dc} & T_{dd} & 0 & T_{du} \\ T_{sc} & T_{sd} & 0 & T_{su} \\ 0 & 0 & 0 & I \end{bmatrix}$$

$$= \left[\begin{array}{c:c:c:c} e^{A_{cc}\tau_o} + E_{cs}T_{sc} & E_{cs}T_{sd} & 0 & E_{cs}T_{su} + E_u \\ T_{dc} & T_{dd} & 0 & T_{du} \\ T_{sc} & T_{sd} & 0 & T_{su} \\ 0 & 0 & 0 & I \end{array}\right] \quad \text{(Equation C-8)}$$

The Discrete Linear-Model Matrices for a Single-Rate System

If we set:

$$A_z = \left[\begin{array}{c|c} e^{A_{cc}\tau_o} + E_{cs}T_{sc} & E_{cs}T_{sd} \\ \hline T_{dc} & T_{dd} \end{array}\right] \qquad \text{(Equation C-9)}$$

$$B_z = \begin{bmatrix} E_{cs}T_{su} + E_u \\ T_{du} \end{bmatrix} \qquad \text{(Equation C-10)}$$

then we have

$$x_n = A_z x_{n-1} + B_z u_n + K_{xn} \qquad \text{(Equation C-11)}$$

where

$$x_k = \begin{bmatrix} x_{c(t_{n^-})} \\ x_{d(t_{n^-})} \end{bmatrix}$$

$\mathbf{x}_n$ = vector of continuous and delay state values at the n^{th} sample time

$\mathbf{u}_n$ = vector of input values at the n^{th} sample time

$\mathbf{K}_{xn}$ = vector of bias values for x_n of operating point (x_0, u_0)

The discrete output and feed through matrices are obtained by perturbing each state or input and forming the quotient of the change in each output by the amount of the perturbation. The resulting linear system has the form:

$$\mathbf{y}_n = \mathbf{C}_c\mathbf{x}_{cn} + \mathbf{C}_d\,\mathbf{x}_{dn} + \mathbf{C}_s\,\mathbf{x}_{sn} + \mathbf{D}_u\mathbf{u}_n + \mathbf{K}_{yn} \qquad \text{(Equation C-12)}$$

We can eliminate the sample-hold state term $\mathbf{C}_s\mathbf{x}_s$ from this equation by obtaining an expression for $\mathbf{x}_{sn}$ from the third row of blocks in (C-8) as:

$$\mathbf{x}_{sn} = \mathbf{T}_{sc}\mathbf{x}_{cn} + \mathbf{T}_{sd}\,\mathbf{x}_{dn} + \mathbf{T}_{su}\mathbf{u}_n$$

Substituting this in (C-12) gives:

$$\mathbf{y}_n = (\mathbf{C}_c + \mathbf{C}_s\mathbf{T}_{sc})\,\mathbf{x}_{cn} + (\mathbf{C}_d + \mathbf{C}_s\mathbf{T}_{sd})\,\mathbf{x}_{dn} + (\mathbf{D}_u + \mathbf{C}_s\mathbf{T}_{su})\,\mathbf{u}_n \qquad \text{(Equation C-13)}$$

Thus the final form of the output and feed forward matrices is:

$$\mathbf{C}_z = [\mathbf{C}_c + \mathbf{C}_s\mathbf{T}_{sc} \;\; \mathbf{C}_d + \mathbf{C}_s\mathbf{T}_{sd}] \qquad \text{(Equation C-14)}$$

$$\mathbf{D}_z = \mathbf{D}_u + \mathbf{C}_s\mathbf{T}_{su} \qquad \text{(Equation C-15)}$$

$\mathbf{K}_{yn}$ = vector of bias values for y_n at operating point (x_0, u_0)

The matrices $\mathbf{A}_z$, $\mathbf{B}_z$, $\mathbf{C}_z$ and $\mathbf{D}_z$, given by C-9, C-10, C-14 and C-15, form the desired linear- model representation. See the section, "Continuous Systems", for a more complete description of $\mathbf{K}_{xn}$ ($\mathbf{K}_x$) and $\mathbf{K}_{yn}$ ($\mathbf{K}_y$) vectors.

A Single-Rate Example

A simple single-rate sampled-data system is shown in Figure 2.

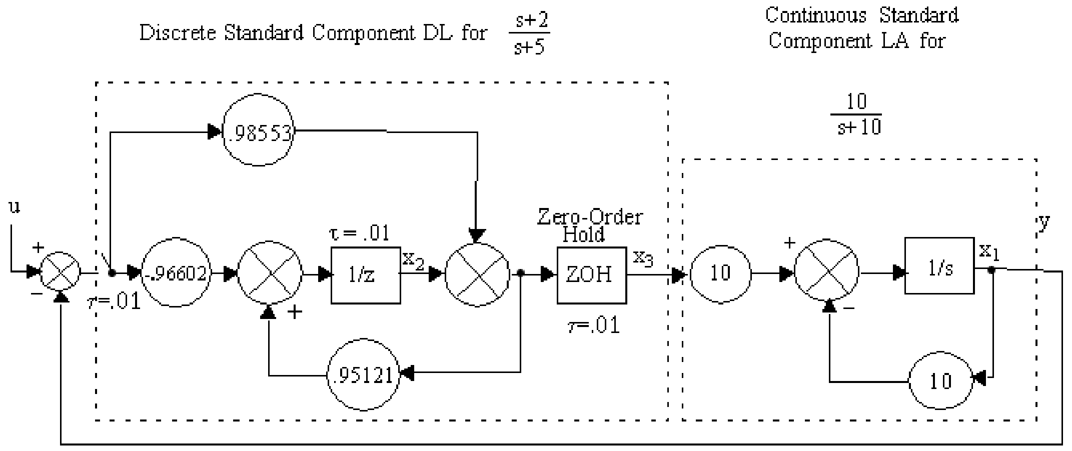

Discrete Standard Component DL for $\frac{s+2}{s+5}$
Continuous Standard Component LA for $\frac{10}{s+10}$
u
+
−
τ=.01
.98553
.96602
τ = .01
1/z
x_2
+
.95121
Zero-Order Hold
ZOH
x_3
τ=.01
10
+
−
1/s
x_1
10
y

Figure 2 Single Sampling Rate Example

The discrete portion of this system approximates the continuous transfer function:

$$\frac{s+2}{s+5} \qquad \text{(Equation C-16)}$$

The continuous-system stability matrix for this system is:

$$A = \begin{bmatrix} -10 & 0 & 10 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{(Equation C-17)}$$

The continuous-system transition matrix for this system is:

$$\Phi = e^{0.01A} = \begin{bmatrix} 0.904837 & 0 & 0.0951625 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ (Equation C-18)

The discrete-system transition matrix for this system is:

$$T = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0.02857 & 0.95121 & 0 & -0.02857 \\ -0.98553 & 1.0 & 0 & 0.98553 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ (Equation C-19)

A complete cycle of this system occurs after one sample period as shown in Figure 3. The system transition is given by:

$x(.01-) = \psi x(0) = \phi T x(0)$ (Equation C-20)

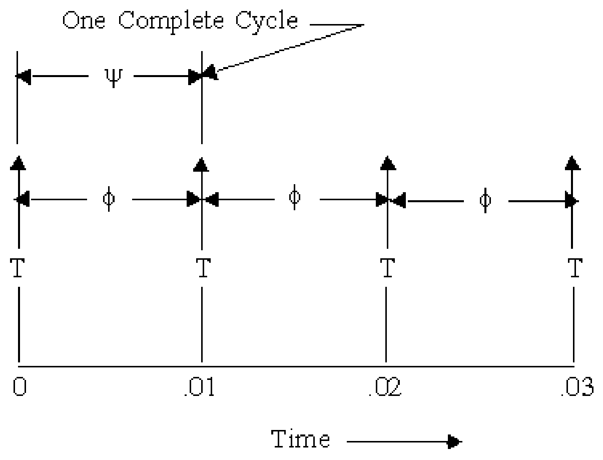


Figure 3 Pictorial representation of Single Sampling-Rate Transition Matrices

The output equation is:

$y = x_1$

Thus, the final linear-model representation is:

$$\begin{bmatrix} x_{1n+1} \\ x_{2n+1} \end{bmatrix} = \begin{bmatrix} .8111 & .09516 \\ .02857 & .95121 \end{bmatrix} \begin{bmatrix} x_{1n} \\ x_{2n} \end{bmatrix} + \begin{bmatrix} .0937855 \\ -.02857 \end{bmatrix} u_n \quad \text{(Equation C-21)}$$

$$y_n = \begin{bmatrix} 1.0 & 0 \end{bmatrix} \begin{bmatrix} x_{1n} \\ x_{2n} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u_n \quad \text{(Equation C-22)}$$

Total Transition Matrix for a Multi-Rate System

Suppose we have a sample-data system with k + 1 different sample periods τ_o, τ_1,... τ_k,

where $\tau_i/\tau_{i-1} = m_i$ is an integer for i = 1, ...,k.

Moreover, we assume that all the samplers in the system take their initial sample at time = 0.

A typical situation is shown in Figure 4 with $\tau_o = .01$, $\tau_1 = .04$ and $\tau_2 = .08$.

If ϕ is the matrix which transitions the continuous part of the system between consecutive τ_o samples, T_i is the matrix which transitions the discrete part of the system at a sample time which is a multiple of t_i (but not a multiple of τ_{i+1}), and we call E_i the matrix which transitions the total system between successive τ_i sample times, then:

$$Eo = \phi$$

$$E_j = (E_{j-1}\ T_{j-1})^{mj-1}\ E_{j-1} \quad j = 1,...,k$$

So the total system-transition matrix over one complete cycle is:

$$\psi = E_k\ \ T_k$$

The linear-model matrices are derived from the total system-transition matrix and output equation exactly as in the single-rate case.

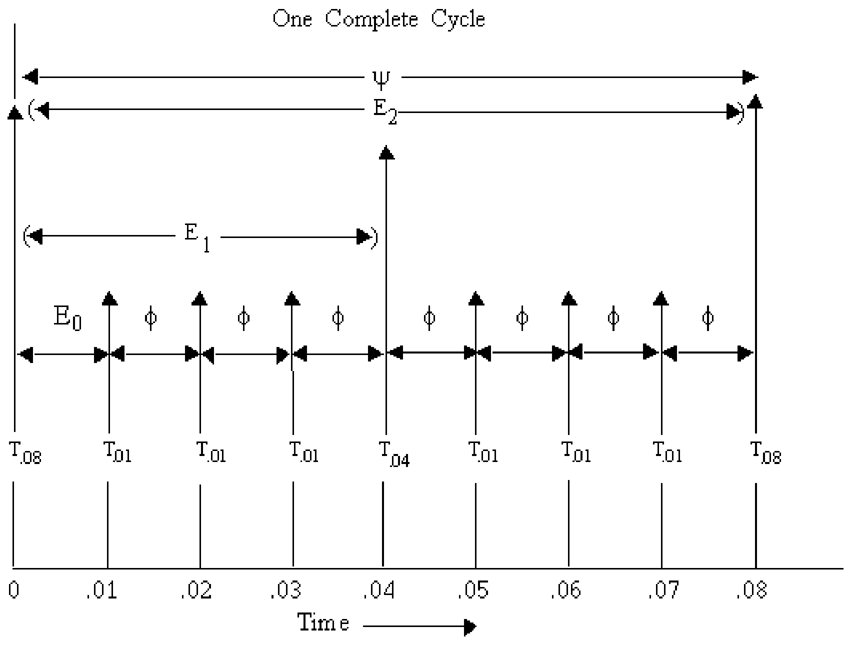


Figure 4 Pictorial Representation of Multi-Sampling-Rate Transition Matrices - Integer Multiple Rates

D Batch Mode Commands

- Overview 380
- External Variables 382
- Command Definitions 385
- Python Multiprocessing 385
- Black Box (BBX) Export and Analysis 388

Overview

Easy5 is generally run using the graphical interface to build models and to setup and execute analyses. In addition, you can also launch batch jobs by entering commands at a command prompt. In Windows systems, the command can be entered in any command shell (Easy5 Command Shell or Easy5 Korn Shell).

This section contains a summary of the batch mode commands to run Easy5 as a batch job, plus several examples.

Before running Easy5 in the batch mode, you must have two files:

- model file (*model_name*.ezmod), also referred to as the "mod" file
- analysis file (*model_name*.ezanl), also referred to as the "anl" file

The *mod* file defines the model in the Easy5 language and is created each time you select "Create Executable" in Easy5. You must first build the graphical model within Easy5, then create the executable model.

The anl file is the analysis input file defined in the Easy5 language, and is created each time you setup and execute an analysis.

To create this file:

Setup and execute an analysis from within Easy5.

OR

Manually create this file using the commands given in Appendix A.

Note: We recommend you use the Easy5 program to setup and run an analysis to create this file, and then kill the job.

Note: Batch mode on-line help is accessed by entering the following command: `easy5x -help_batch`

Batch Mode Command

Batch jobs are submitted by entering the "easy5x -B" command at a command prompt. The command has the following format:

```
$ easy5x -B[root_name|options][model_file|options][analysis_file]
                (Position 1)              (Position 2)          (Position 3)
```

The command is followed by one or more arguments, entered in up to three different "positions". A summary of the position arguments is given in the following tables.

Argument	Purpose
root_name	Base name for all Easy5 output files (.ezmgl, .f, .exe, .ezapl, etc.)
-help	On-line help.
-def	Displays the settings of all Easy5 internal variables
-vars	Displays the settings of all Easy5 external variables
-version	Displays the current version of the Easy5 engine
-home	Displays the home directory path for Easy5 engine

Argument	Purpose
model_file	Full name of model input file (*model_name.mod)*
+[exeid]	Easy5 bypasses model generation and goes directly to analysis execution. File *root_name.exe* is used, unless *exeid* is specified; then *exeid.exe* is used.

Argument	Purpose
model_file	Full name of model input file (*model_name.mod*)
+[exeid]	Easy5 bypasses model generation and goes directly to analysis execution. File *root_name.exe* is used, unless *exeid* is specified; then *exeid.exe* is used.
-l	Relinks using the compiled model from file root_name.o. If an analysis file has been included (in position 3), analysis follows the link.
-es	Generates an input file for ESA using plot data from file root_name.ezrpd

Argument	Purpose
analysis_file	Full name of analysis command file (*model_name.anl*) A blank bypasses analysis execution.

Examples of Batch Commands

1. To create a model executable only, using model file named my_model.mod, enter:

```
easy5x -B run1 my_model.ezmod
```

   This will create an executable file named: *run1.exe*. If you want the executable file to have the same core name as the model name, then use the model name as the root name as follows:

```
easy5x -B my_model my_model.ezmod
```

This will create an executable file named: *my_model.exe*.

2. To create a model executable from a model file named *test_bed*, and follow this with running an analysis using an *anl* file named *test_bed.simulation.ezanl*, enter:

   ```
   easy5x -B test_bed test_bed.ezmod test_bed.simulation.ezanl
   ```

   This will create an executable file named *test_bed.exe*, and an output listing file named: *test_bed.ezapl*, and if plots are generated, a plot file called: *test_bed.ezrpd*.

3. If you have already created the executable model and need only to run the analysis, then using the same filenames used in the previous example, enter the following command:

   ```
   easy5x -B test_bed + test_bed.simulation.ezanl
   ```

   The "+" indicates that the model executable has already been created, and that it is named using the root_name (i.e. in the above example the file named *test_bed.exe* exists). If the executable model uses a different name, then include the name with the "+" sign. For example, if the executable is named *my_model.exe* instead of *test_bed.exe*, then the above command would be:

   ```
   easy5x -B test_bed +my_mod test_bed.simulation.ezanl
   ```

Note: Additional examples are given in the last section of this appendix.

External Variables

External variables are set prior to the Easy5 execution command and can be used to customize Easy5 execution.. For complete information on how to setup and use external variables, see the section "External (Environment) Variables".

Note: For a list of all Easy5 external variables, enter the command:

```
    easy5x -vars
```

To display the current values of Easy5 external variables that are set, enter the command:

```
    easy5x -varset
```

Variable Name	Purpose
appli*bn*	The pathname for an Easy5 application library or for an existing component library, where *n*= 1,2,3, ... ,18. Easy5 will automatically search for application libraries on its default component library directory. If not found in that directory, it will search the current working directory. You specify the entire pathname (minus the file identifier) to explicitly assign a component library.
auxfile	The name of the auxiliary input file to be used during analysis.
complib	The name of the default component library to be used during the link phase. Default is the Easy5 General Purpose block library
dmpfile	The name of the DUMP DATA input file to be used during analysis.
ezdebug	If set to yes, will use the -g FORTRAN compilation option and will execute the model executable using the debugger. For complete information about the debugger and its commands, consult the appropriate debugger manual or invoke the debugger's own help command for online assistance.
ftnopt	compilation of model is performed using options listed with this variable. To list Easy5 compiler options enter: `easy5x -B -def | grep ezfflags`
ftnsave	model is saved as root_name.f if set to yes.
macrofile	Name of the operative library to be used by the model generation program, if one is needed.
object	Name of the user object code file to be included during the link phase
source	Name of the user source code file to be appended to your Fortran model
userlib	Name of the user library to be included during the link step.

Examples:

The following examples show the various batch commands and Easy5 variable settings. These are for the Linux Korn shell and any *Easy5 Command Shell* only.

1. To create a model executable, using model file my_model.mod; and execute the analysis program using analysis file my_analysis.ezanl:

```
$ easy5x -B run1 my_model.ezmod my_analysis.ezanl
```

2. To just build a model executable with file my_model.mod and include additional binary code in the link sequence (from pathname */blat/my_object.o*):

```
$ export object=/blat/my_object.o
$ easy5x -B run2 my_model.ezmod
```

3. To execute the analysis program using analysis file *my_analysis.ezanl* using a saved model executable on file *run2.exe*:

```
$ easy5x -B run2 + my_analysis.ezanl
```

4. To access an auxiliary input file from file *saved.dat*; request a copy of your Fortran model; create a new model; and execute the analysis program:

```
$ export auxfile=saved.dat

$ export ftnsave=yes

$ easy5x -B run3 my_model.ezmod my_analysis.ezanl
```

5. To access an Easy5 application library, *xx.ezdf*; update your own component library on pathname */blat/macro/my_comp_lib.ezdf*; and create a model executable using model file *my_model.ezmod*:

```
$ export applib1=xx

$ export macrofile=/blat/macro/my_comp_lib.ezdf

$ easy5x -B run4 my_model.mod
```

6. To execute Easy5's model generation program again and request use of the Fortran compiler options *-O -Nx 400*, enter:

```
$ export ftnopt="-O -Nx 400"

$ easy5x -B run5 my_model.ezmod
```

Note: When defining Windows environment variables, do not put quotes around the values, even if it contains multiple values with blank space separators.

Extraneous Easy5 Batch Files

Special "link files" are created when running the Easy5 batch mode, and are generally deleted by Easy5 when the execution is finished. However, if these files are not deleted by the Easy5 program, you should delete these files:

- APPDFn (where n=1,2,3,...18)
- AUXINP
- DATAA
- DATAM
- DMPDAT
- EZAX
- EZGPDF
- PLOTS

- PLOTINPF

Command Definitions

To submit a batch job enter the "easy5x -B" command at a command prompt using the following format:

```
$ easy5x -B[root_name|options][model_file|options][analysis_file]
```

(Position 1) (Position 2) (Position 3)

The command is followed by one or more arguments, entered in up to three different "positions". A summary of the position arguments is given in the following tables.

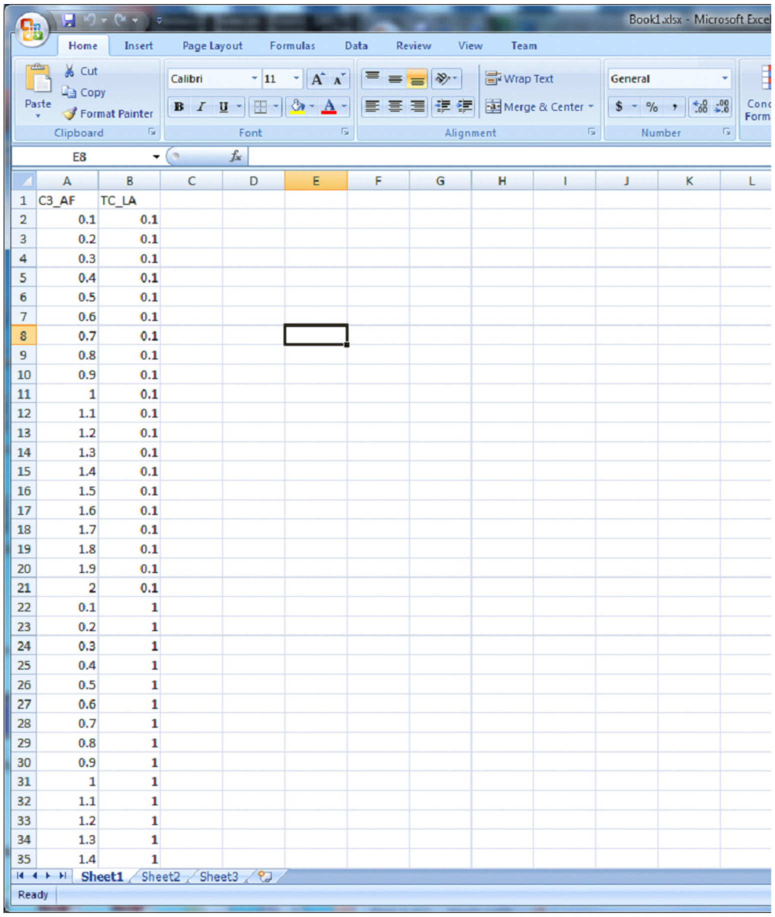
Argument	Purpose
root_name	Base name for all Easy5 output files (.ezmgl, .f, .exe, .ezapl, etc.)
-help	On-line help.
-def	Displays the settings of all Easy5 internal variables
-vars	Displays the settings of all Easy5 external variables
-version	Displays the current version of the Easy5 engine
-home	Displays the home directory path for Easy5 engine

Argument	Purpose
model_file	Full name of model input file (*model_name.ezmod*)
+[exeid]	Easy5 bypasses model generation and goes directly to analysis execution. File *root_name.exe* is used, unless *exeid* is specified; then *exeid.exe* is used.
-l	Relinks using the compiled model from file root_name.o. If an analysis file has been included (in position 3), analysis follows the link.
-es	Generates an input file for ESA using plot data from *root_name*.ezrpd

Python Multiprocessing

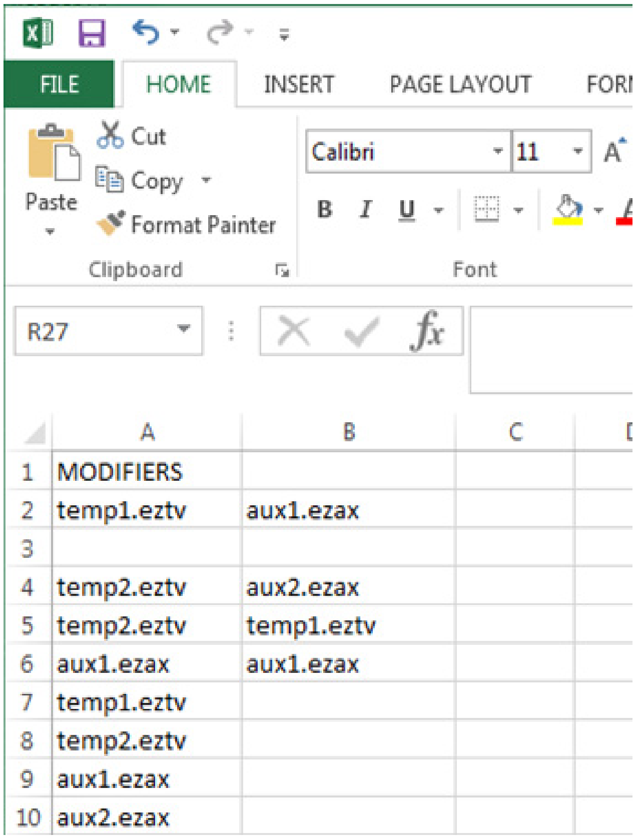
Easy5 batch processing can take advantage of the Python Multiprocessing and CSV modules. Two typical use cases for this capability:

1. Suppose one wishes to run an overnight batch of analyses on a machine with several processors with varying parameter values. The user would create a CSV file with parameter names on the first row and parameter values on the subsequent rows:



	A	B
1	C3_AF	TC_LA
2	0.1	0.1
3	0.2	0.1
4	0.3	0.1
5	0.4	0.1
6	0.5	0.1
7	0.6	0.1
8	0.7	0.1
9	0.8	0.1
10	0.9	0.1
11	1	0.1
12	1.1	0.1
13	1.2	0.1
14	1.3	0.1
15	1.4	0.1
16	1.5	0.1
17	1.6	0.1
18	1.7	0.1
19	1.8	0.1
20	1.9	0.1
21	2	0.1
22	0.1	1
23	0.2	1
24	0.3	1
25	0.4	1
26	0.5	1
27	0.6	1
28	0.7	1
29	0.8	1
30	0.9	1
31	1	1
32	1.1	1
33	1.2	1
34	1.3	1
35	1.4	1

2. One can also define the batch in terms of modifier files (temporary settings files and auxiliary input files). To do this, create a CSV file with the keyword "MODIFIERS" on the first row, and the names of the modifier files on the subsequent rows, one name per cell. For example:



	A	B	C
1	MODIFIERS		
2	temp1.eztv	aux1.ezax	
3			
4	temp2.eztv	aux2.ezax	
5	temp2.eztv	temp1.eztv	
6	aux1.ezax	aux1.ezax	
7	temp1.eztv		
8	temp2.eztv		
9	aux1.ezax		
10	aux2.ezax		

In this example, all cells of Row 3 are clear, so that row will correspond to baseline model conditions.

Each row of the batch starts with baseline conditions, the effects of the modifier files are not cumulative from row to row.

Windows

Perhaps the most convenient command to invoke Python Multiprocessing, in an Easy5 Command Shell, is simply

```
easy5mp  <ModelName> <#OfProcessors> <CSVFileName> <AnalysisType> <AnalysisID>
```

which invokes %EZHOME%\easy5mp.bat.

Linux (or Windows)

In an Easy5 Command Shell or a Linux shell that properly defines the Easy5 environment, the user would type python $WSHOME/engine/easy5mp.py <ModelName> <#OfProcessors> <CSVFileName> <AnalysisType> <AnalysisID>

The first argument <ModelName> is required. The remaining arguments are optional- default values are assumed, if the user does not supply values:

<#OfProcessors>	1
<CSVFileName>	<ModelName>.csv
<AnalysisType>	simulation
<AnalysisID>	simulation

The plot output files are named

<ModelName>.<AnalysisID>_Row1.ezrpd

ModelName>.<AnalysisID>_Row2.ezrpd

etc.

and the analysis output files are named

<ModelName>.<AnalysisID>_Row1.ezapl

<ModelName>.<AnalysisID>_Row2.ezapl

etc.

Note: The Python version must be 3.5 or higher.

Black Box (BBX) Export and Analysis

Note: An "Easy5 MATLAB Interface Toolkit" feature is required to use this feature.

Easy5 models can be packaged for batch simulation analysis by your customers or vendors as “black-box" executables by selecting **Build > Export Model As > Self contained (black box) executable.** The executable that is created is not licensed-managed, that is, it will run without a license on any computer with a compatible operating system. Note, however, that only a single simulation analysis may be incorporated into a BBX. Other analyses (linear analysis, steady-state and so on) as well as Multiple Analysis are not supported.

You are then prompted to enter an identifier for the analysis simulation input file that will form the baseline analysis for the model, and an encryption key that will protect the data in the baseline analysis input file. A new directory (ModelName_BBX) will be created and all of the necessary files placed there.

To run a BBX model after it has been exported (simply double-clicking on the *.exe* file is insufficient):

1. Open a shell window. Most any type of shell will do, such as a Windows Command shell, or Linux C-shell. However, **Python must be in the path**.
2. cd to the BBX directory containing the exported model.

3. Type: `python ez5_bbx_run.py <ModelName> <AnalysisInputFileName> <EncryptionKey> <AuxFileName>`

where

- `<AnalysisInputFileName>` is the encrypted analysis input file (default: ModelName.bbx.ezanl).
- `<EncryptionKey>` is the same EncryptionKey that was entered when the BBX was exported.
- `<AuxFileName>` is an optional unencrypted auxiliary input file. This auxiliary input file is where the recipient of the model would perturb the baseline analysis conditions, according to the instructions given by the provider of the model. See the Easy5 Reference Manual Topic "Auxiliary Input File". The auxiliary input file should begin with the label "$BBX".

Consider for example the AirCycle/CabinAirControl demo model from the GD library. This model contains a parameter named PressureInletVI. If this model were exported as a BBX, with encryption key "EASY5", a new directory named CabinAirControl_BBX would be created containing the necessary files. In addition to these files, one could create, with any text editor, a simple auxiliary input file named "bbx_aux" to perturb the baseline value of PressureInletVI:

```
$BBX
PARAMETER VALUE
PressureInletVI = 410
```

Then to execute an analysis one would type in a shell in the CabinAirControl_BBX directory:

```
    python ez5_bbx_run.py CabinAirControl CabinAirControl.bbx.ezanl
    EASY5 bbx_aux
```

E Program Limits

The Easy5 program can be used to model and analyze large dynamic systems. However, there are certain limitations caused by dimension statements within the model generation and analysis programs. These limits are listed below.

Modeling Limitations

FEATURE	MAX. ALLOWED
model_name length	32 characters
"user-defined name" length	60 characters
Fortran and C component input/output name length	60 characters
Library component input/output name (incl. port name, if used)	19 characters
Library component port name length	12 characters
number of components in a library	200
components per model	5000
components per hierarchical level (per schematic page)	400
components + connections per model	10000
connection lines into + out-of a component	200
number of submodels	500
tables per model	1000
number of states per model	9999
number of sort blocks per model	9999
component dimension (per dimension parameter)	99999
dimension size (given by a vector or array)	9999999
opened libraries (gp not included)	50
user-defined names	10,000
dimension parameters (I, J, K, L, M, N)	5
feature parameters (C, D, E, F, G, H)	6
revision parameter (R)	1

Analysis Limitations

FEATURE	MAX. ALLOWED
analysis settings file name length*	41-<model name>*
plotted quantities per simulation or steady-state	8000
stability margins parameters	10
print variables	40
maximum data plotted for Root Locus analysis	8000
frequency points plotted for Transfer Function analysis	2130
number of different sampling rates in discrete-time model	10
plot points for Plot Tables and Function Scan Analysis	999,999

Note: The length of the analysis settings file name depends on how large the model name is as follows: # model name characters + # analysis settings characters < 41.

For example, if the model name uses the maximum allowed 32 characters, then the analysis settings file name < 19 characters. However, if the model name is only 11 characters in length, then the analysis name can be as long as 30 characters.

Index

Reference Manual

Numerics
2nd Time Incr, 265

A
Accelerator keys, 5
Adams integration method, 158
Adding a Component, 9
Adding Components, 6
Analyses
    linear, 12
    nonlinear, 11
Analysis
    Eigenvalue Sensitivity Analysis, 116
    Interactive Simulation, 163
    Plot Tables Analysis, 232
    Power Spectral Density Analysis, 233
    Stability Margins Analysis, 278
    Steady-State Analysis, 287
    Transfer Function Analysis, 322
Analysis Data Form, 12
    file menu, 14
Analysis menu, 198
Analysis Settings File, 14
auxfile, external variable, 128
Auxiliary Input File
    auxfile, 19
    creating, 18
    data format, 21
    labeled data, 19

B
BBX, 388
BCS Gear integration method, 157
BEGIN INITIALIZATION code statement, 153
BEGIN TERMINATION CODE command, 318
bindopt, external variable, 128
Black Box (BBX) Export and Analysis, 388
Block diagram modeling method, 214
Branch connection, 74
Build menu, 197

C
C code, 29
    compiling external code, 43
    discrete system modeling, 105
Change Names, menu, 344
Close... submodel menu, 302
CNT_LABEL_BELOW_LEFT, external variable, 128
CNT_LABEL_HEAD, external variable, 127
CNT_LABEL_OFF, external variable, 127
CNT_LABEL_VERTICAL, external variable, 127
Code
    C code, 29
    component, 55
    mixed code, 46
Compiling External Code, 43
Component
    code component, 55
Component Data Table, 60
Component modeling method, 215
Component shortcut menu, 204

Components
- Add Component menu, 6
- adding components, 6
- adding components by name, 10
- adding FORTRAN components, 11
- arrays, defining input, 231
- data table, 60
  - defining outputs, 246
- default names, 52
- deleting, 100
- dimension, 58
- Extension component, 57
- inputs, 52
- locating, 253
- moving groups of components, 252
- moving individual components, 252
- naming convention, 52
- on-line information, 8
- output, 52

Connecting components
- connection line attributes, 80
- custom connections, 72
- incompatibly vectorized components, 75
- port connections, 70

Connection components
- branch connection, 74

Connection line
- label options, 84

Connection line shortcut menu, 206

Continuous states, 283

Controllability matrix, 165

CPU Time, 272

CPUSEC, reserved word, 243

Create Executable menu, 119

Create Executable with Debug, menu, 120

Current Schematic menu, 140

D

Data
- array parameters, 23
- Excel, 93
- integers, 130
- logical variables, 130
- parameters, 93
- scalar parameters, 22
- states, 93
- tables, 93
- tabular data, 26
- types, 93
- variables, 93

Debugger
- creating executable with debug, 120
- example of using symbolic debugger, 94, 97
- executing an analysis with the debugger, 266

Declaration statement
- delay state, 105
- sample-and-hold state, 105

DECLARATIONS, statement, 131

Default names, 52

Delay state, 105

Delay states, 285

Deleting components, 100

Digital filters, 103

Digital modelling, 103

Dimensioned components, 58

Discrete systems
- Transfer Function, 334
- transition matrix, 371

Documenting the model, 109

DPR, reserved word, 246

E

EASY5
- command options, 228

EASY5 Window
- message line, 114
- schematic window, 114

EASY5_BROWSER, external variable, 129

EASY5_FIRST_LIB, external variable, 126

EASY5_GROUP_LIB, external variable, 126

EASY5_IGNORE_LIB, external variable, 126

EASY5_OBJECT, external variable, 126, 177

EASY5_PDF_READER, external variable, 129
EASY5_SELECT_LIB, external variable, 126
EASY5_SITE_LIB, external variable, 126
EASY5_USER_LIB, external variable, 126
EASY5_XCOMP_DIR, external variable, 129
Edit menu, 192
Eigenvalue Sensitivity Analysis, 116
    parameter, defining the, 117
Eigenvalues of linear model, 171
END INITIALIZATION code statement, 153
END TERMINATION CODE command, 318
Entire Hiearachy menu, 140
Environment Variables, 124
EQMO, subroutine, 118
Error controls, 223
Euler integration method, 157
Example of Calling Fortran from C, 48
Excel data, 93
Executable
    output files, 121
    source file, 122
Exit routines, 295
Expand... menu, 304
Extension components, 57
External code, 175
External Variable
    setting and deleting external variables, 124
External Variables, 124
EZ_NO_CONFIRM, external variable, 129
ezdebug, external variable, 128
EZXLG, reserved word, 246

F
File menu, 190
FORTRAN
    compiling external code, 43
    DECLARATIONS, statement, 131
    discrete system modeling, 105
    integers, 130
    logical variables, 130
    matrix operations, 134, 183
    non-executable statements, 131
    reserved words, 132
    see also *EASY5 User's Guide, Chapter 10*
    sort blocks, 278
    sorting code, 135
    START SORT command, 135
    STOP SORT command, 135
Fortran
    calling C code, 50
FPZ, reserved word, 246
Frequency response method, 326
ftnopt, external variable, 128
Function Scan Data Form
    independent/dependent variables, defining, 137
    plots, suppressing, 139
    time, defining, 137
    two independent variables, defining, 138

G
Graphic formats, 139
GRE, reserved word, 246
GRM, reserved word, 246
Grouop shortcut menu, 208

H
Help menu, 200
HIDE_CNT_OUTPUT_LABELS, external variable, 128
HIDE_CNT_SUBMODEL_LABELS, external variable, 128
HTML documentation of a model, 109
Huen integration method, 157
Hybrid models, 104
Hybrid systems, 104

I
ICCALC, reserved word, 243
IDELAY, reserved word, 107, 243
IERR, reserved word, 244, 246

Implicit model, 145
INCALL, reserved word, 243
Info page, 8
Initial Conditions, 286
 ICCALC flag, 243
Initialization of code, 152
Input
 component, 52
INST, reserved word, 244
INT MODE, analysis command, 154, 358
Integration
 accuracy, 361
 Adams method, 158
 BCS Gear method, 157
 error controls, 162
 Euler method, 157
 fixed-step method, 155
 Huen method, 157
 methods, 153
 numerical stability, 358
 order, 155
 problems during simulation, 272
 Runge-Kutta method, 157
 selection guidelines, 160
 specifying integration mode, 258
 step-size control, 161
 user-defined method, 158
 variable-step method, 155
Interactive Simulation, 163
INTG - termination mode flag, 319
INTG, reserved word, 244
ISTOP flag, 295
ISTOP, reserved word, 244
ITINC, reserved word, 244
IWRITE, reserved word, 245

L
Labeled Data Blocks, 19
Labels
 external variables, 127
 options, 84
 submodel, 84, 302
Library
 see *EASY5 User's Guide*, Chapter 11
Linear analyses, 12
Linear model generation
 continuous systems, 168
 highly nonlinear systems, 172
 linearity, measurement of, 174
 linearization method, 170
 matrices, 168
 method, 168
 sampled-data (discrete) systems, 175
 sampled-data systems, 368
 setting up, see Linear Model Generation Data Form
 types of analyses, 163, 171
Linear Model Generation Data Form
 inputs, defining linear model, 166
 outputs, defining linear model, 167
 saving linear model system matrices, 168
LINEAR, reserved word, 245
Link
 EASY5_OBJECT method, 177
 external code, 120, 175
 link list method, 176
 macro routines, 179
Link External Object menu, 176
Link list, 176

M
M, 118
Macros
 integers, using, 181
 linking macro routines, 179
 logical variables, using, 181
 modeling discrete systems, 105
 RESUME SORT command, 277
 see *EASY5 User's Guide*, Chapter 12
 sort blocks, 278
 STOP SORT command, 277
Matrix
 controllability, 165
 discrete system transition matrix, 371
 FORTRAN operations, 134, 183
 modal, 165
 observability, 165
Menu bar, 113

Menu(s)
- Analysis menu, 198
- Build menu, 197
- closing menus, 203
- Edit menu, 192
- file, 113
- File menu, 190
- Help menu, 200
- Menu Bar, 113
- opening a menu, 113
- opening menus via
  - accelerator key input, 5
  - arrow key input, 202
- Options menu, 195
- shortcut menu, 203
- Submodel menu, 200
- View Menu, 115
- View menu, 194

Message line, 114
Metafile error, 226
Mixed code, 46
Modal matrix, 165
Model
- damaged Model, 226
- documenting and printing, 109
- executable source file, 122
- generation listing file, 122
- hybrid, 104
- implicit model, 145
- information button, 113
- sorting, 276

Model Info, 113
Modeling
- block diagram method, 214
- component method, 215
- digital, 103
- discrete (digital) systems, 103
- fundamental, 213
- hybrid systems, 104
- state variable method, 218
- user-defined component method, 217

Moving components, 252
Moving the window, 253
MULT ERROR BY, command, 162
Multiple Analysis Data Form
- analysis settings file, 224
- setting up the analysis, 220
- temporary settings file, 224

MUX function, 68

N

Naming convention, components, 52
NEXT VALUE OF, statement, 105
Nonlinear Analyses, 11

O

Observability matrix, 165
ONE, reserved word, 246
Operating Point
- creating an operating point, 228
- defining from a simulation, 229
- defining initial operating point, 16
- file, 229
- restoring an operating point, 223
- saving a simulation operating point, 266
- simulation initial operating point, 254
- steady-state (equilibrium) point, 229

Options, 228
Options menu, 195
Output
- component, 52
- transfer function, 324

P

Parameter data, 93
PFLAG, reserved word, 245
PI, reserved word, 246
Plot Rate Multiplier (Print Rate), 263
Plot Tables Analysis, 232
Plotter
- plot tables analysis, 232
- see also *EASY5 User's Guide*, Chapter 8

Plotting
- plot all output data method, 262

Plotting Data
- see *EASY5 User's Guide*, Chapter 8

Plotting secondary rates, 265
PostScript files, 139

Power Spectral Density Analysis, 233
 outputs, 238
 plots, 238
 setting up, see Power Spectral Densitiy Data Form
Power Spectral Density Data Form
 input, 234
 input noise distribution, 235, 236
 output, 234
 plot data options, 236
 time, defining, 234
PowerPoint graphics file, 139
Printing the model, 109

R

Reserved word
 CPUSEC, 243
 DPR, 246
 EZXLG, 246
 FPZ, 246
 GRE, 246
 GRM, 246
 ICCALC, 243
 IDELAY, 243
 IERR, 244, 246
 INCALL, 243
 INST, 244
 INTG, 244
 ISTOP, 244
 ITINC, 244
 IWRITE, 245
 LINEAR, 245
 ONE, 246
 PFLAG, 245
 PI, 246
 RPD, 246
 TAU0, 245
 TAUMAX, 245
 TIME, 245, 313
 TINC, 245
 TMAX, 245
 TSTEP, 245
 ZERO, 246
RESUME SORT, command, 278
Root Locus Analysis, 246
Root locus analysis
 method, 250
 plots, 250
 root locus zeros, 251
 setting up, see Root Locus Data Form
 zeros, calculating, 249
Root Locus Data Form
 parameter, specifying the root locus, 248
 plots, requesting, 250
 zeros, calculating, 250
RPD, reserved word, 246
Runge-Kutta integration method, 157

S

Sample period, 105
SAMPLE PERIOD OF statement, 105
Sample states, 284
Sample-and-hold state, 105
Sates
 continuous states, 283
Schematic shortcut menu, 203
Scroll Bar, 113
Shortcut menu, 203
 component, 204
 connection line, 206
 group, 208
 schematic, 203
 submodel, 206
 submodel stub, 207
Show Names, menu, 343
SHOW_CNT_PORT_LABELS, external variable, 128
Simulation
 failure, 269
 integration problems, 272
 killing a run, 272
 outputs, 267
 plotting results, 260
 printing results, 263
 saving final operating point, 266
 troubleshooting, 269

Simulation Data Form
  integration mode, 258
  plot specification form, 261
  plotting, 260
  print specification form, 264
  print/plot increment, specifying, 260
  printing, 263
  secondary plotting and printing rates, 265
  setting up a simulation, 254
  simulation time, 257
Sorting code
  FORTRAN sorting, 135
Stability Margins analysis, 278
  method, 280
  setting up, see Stability Margins Data Form
Stability Margins Data Form
  parameters, defining, 279
Standard Components, 54
START SORT command, 135
State Variable Modeling Method, 218
States
  defining states, 286
  definition, 93
  delay state declaration statement, 105
  delay states, 285
  error controls, 162, 223
  initial condition, 286
  outputs, 246
  sample states, 284
  sample-and-hold declaration statement, 105
  switch states, 286
Steady-State
  finding a single point, 290
  finding multiple points, 290
  guiding the analysis, 288
  method, 292
  operating point, 291
  printing results, 291
  troubleshooting, 293
  types of analyses, 288
  valid bounds, 288
Steady-State Data Form
  number of iterations, defining, 289
  printing results, 291
  scan option, 290
  setting up an analysis, 288
Stop Current Analysis menu, 272
Stop flags, 295
STOP SORT command, 135
STOP SORT, command, 278
Submodel menu, 200, 300
Submodel shortcut menu, 206
Submodel stub shortcut menu, 207
Submodels
  attributes, 303
  creating and using submodels, 292–305
  defining/creating a submodel, 300
  editing attributes, 303
  expanding a submodel, 304
  hierarchical layers,, 300
  labels, 302
  managing submodels, 304
  opening/closing a submodel, 301
Switch states, 286, 305

T
Table data, 93
Table data format, 26
TAU0, reserved word, 245
TAUMAX, reserved word, 245
Temporary Settings file
  applying to an analysis, 317
  creating a new file, 313
  entering and editing data, 314
  undo temporary settings, 224
Termination commands, 318
Termination mode flag, 319
Time
  defining analysis Time, 16
  print/plot time increment, 260
  testing on the value of TIME, 321
TIME, reserved word, 245, 313
TINC, reserved word, 245
TMAX, reserved word, 245

Transfer Function analysis
  method, 326
  output listing, 325
  outputs, 325
  plots, 325
  sampled data (discrete) systems, 328, 334
  troubleshooting, 329
Transfer Function Analysis Data Form, 324
  input, defining the transfer function, 323
  plot, requesting a transfer function, 324
TSTEP, reserved word, 245

U
User-Defined component modeling method, 217
User-defined integration method, 158
User-defined names, 322
  changing user-defined names, 344
  defining, 342
  menu options, 343

V
VALUE OF, statement, 105
Variable data, 93
Variables, Environment, 124
View Entire Schematic menu, 253
View menu, 115, 194

W
Window
  basics, 114
  locating components, 253
  moving the window, 253
Word graphics file, 139
WSANLSPACE, external variable, 126
WSCONCOLOR, external variable, 127
WSDEBUG, external variable, 127
WSDEBUGGERNAME, external variable, 128
WSLIBDIR, external variable, 126
WSMFSPACE, external variable, 126
WSMODAUX1, external variable, 128
WSMODAUX2, external variable, 128
WSNOAUTO_RESULTS, external variable, 127
WSPLOT_AUTO_SWL_LO, external variable, 129

X
xtralib, external variable, 128

Z
ZERO, reserved word, 246
Zoom In menu, 253
Zoom Out menu, 253