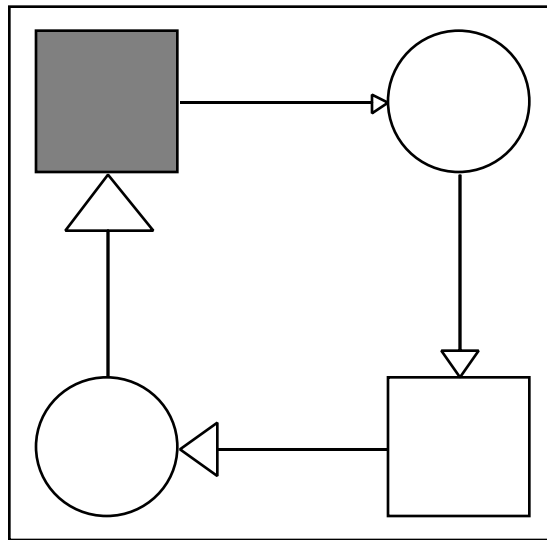


Design/CPN Reference Manual for X-Windows

Version 2.0



Meta Software Corporation



125 CambridgePark Drive
Cambridge, MA 02140 U.S.A.
Tel: (617) 576-6920
Fax: (617) 661-2008

© 1993 Meta Software

© 1993 Meta Software Corporation

125 CambridgePark Drive

Cambridge, MA 02140

(617) 576-6920

FAX: (617) 661-2008

email: cpn-tech-support@metasoft.com

Design/CPN is a trademark of Meta Software Corporation.

X-Windows is a trademark of the Massachusetts Institute of Technology.

Design/CPN Reference Manual for X-Windows

Version 2.0

Table of Contents

Part 1: Design/CPN User's Guide

Chapter 1

Getting Started With Design/CPN

Prerequisites.....	1-1
Terminology.....	1-1
What Is Design/CPN.....	1-2
Design/CPN and X-Windows.....	1-2
Design/CPN Multiprocessing.....	1-2
Design/CPN and the File System.....	1-3
Design/CPN Use of the Mouse.....	1-3
Design/CPN Use of the Keyboard.....	1-4
Figures in This Manual.....	1-4
Starting Design/CPN.....	1-4
Creating or Opening a Diagram.....	1-4
The Design/CPN User Interface.....	1-5
The Menu Bar.....	1-5
The Status Bar.....	1-5
The Page.....	1-5
Navigating a Diagram.....	1-6
Printing a Diagram.....	1-6
Closing a Diagram.....	1-7
Quitting Design/CPN.....	1-7

Chapter 2

CP Net Components

The CPN ML Language.....	2-1
A CP Net Example.....	2-2
CPN Data.....	2-3
Colorsets.....	2-3
Enumerated Colorsets.....	2-4
String and Integer Colorsets.....	2-4
Duplicate Colorsets.....	2-5
Tuple Colorsets.....	2-5
Tuple Constructors.....	2-6
Tuple Patterns.....	2-6
Other Colorsets.....	2-7
Tokens.....	2-7
Multisets of Tokens.....	2-7
Specifying Multisets.....	2-8
Multiset Addition.....	2-8
Multiset Subtraction.....	2-8
Multiset Subsets.....	2-9
CPN Variables.....	2-9
Places.....	2-10
Place Markings.....	2-11
States and Markings.....	2-11
Initial Marking Regions.....	2-12
Appearance of Markings.....	2-12
Transitions.....	2-12
Arcs.....	2-13
Bidirectional Arcs.....	2-13
Arc Inscriptions.....	2-14
Guards.....	2-14
CP Net Execution.....	2-15

Chapter 3

The Design/CPN Editor: Introduction

Design/CPN Graphical Objects.....	3-2
Graphics Editor Modes.....	3-2
Graphics Mode.....	3-3
Text Mode.....	3-3
Creating Graphical Objects.....	3-3
Exiting Creation Mode.....	3-4
Drawing Tools.....	3-4
Autoscrolling.....	3-4
Resetting the Drawing Environment.....	3-5
Caps Lock Under X-Windows.....	3-5

Chapter 3**The Design/CPN Editor: Introduction (cont'd)**

Working With Rectangles.....	3-6
Creating Rectangles.....	3-6
Enter Rectangle Creation Mode.....	3-6
Specify the First Corner.....	3-6
Specify the Diagonal Corner.....	3-7
Finish the Rectangle.....	3-7
Leave Rectangle Creation Mode.....	3-7
Reshaping Rectangles.....	3-7
Moving Rectangles.....	3-8
Deleting Rectangles.....	3-8
Moving Rectangles During Creation.....	3-8
Adding Text to Rectangles.....	3-9
Creating a Series of Rectangles.....	3-9
Adding Text to Rectangles During Creation.....	3-10
Preserving a Rectangle's Aspect Ratio.....	3-10
Working With Ellipses.....	3-10
Creating an Ellipse.....	3-10
Enter Ellipse Creation Mode.....	3-10
Specify the First Corner.....	3-11
Specify the Diagonal Corner.....	3-11
Finish the Ellipse.....	3-11
Leave Ellipse Creation Mode.....	3-11
Other Operations With Ellipses.....	3-12
Working With Labels.....	3-12
Creating Labels.....	3-12
Enter Label Creation Mode.....	3-12
Create the Label.....	3-12
Other Operations With Labels.....	3-13
Working With Connectors.....	3-13
Creating Connectors.....	3-14
Routing Connectors.....	3-14
Editing Connectors.....	3-15
Automatic Rerouting of Connectors.....	3-15
Deletion of Dangling Connectors.....	3-15
Working with Other Types of Objects.....	3-15
Creating Regions.....	3-16
Designating Regions.....	3-16
Converting Regions into Nodes.....	3-17
Editing Parents and Regions.....	3-17
Moving Parents.....	3-17
Deleting Parents.....	3-17
Creating Objects From Text Mode.....	3-18
Selecting Graphical Objects.....	3-18
Current Objects.....	3-18
Selecting Objects.....	3-18
Working With More Than One Object Type.....	3-19

Chapter 3

The Design/CPN Editor: Introduction (cont'd)

Groups of Objects.....	3-19
Mixed Groups.....	3-20
Selecting Groups.....	3-20
Deselecting Groups.....	3-21
Reconstructing Groups.....	3-21
Operating on Groups.....	3-22
Aligning Graphical Objects.....	3-22
Horizontal Spread Command.....	3-22
Vertical Spread Command.....	3-23
Horizontal Command.....	3-23
Vertical Command.....	3-23
Other Alignment Commands.....	3-23
Undoing Alignment Commands.....	3-23
Matching Object Sizes.....	3-24

Chapter 4

The Design/CPN Editor: Creating a CP Net

Auxiliary Graphics and CPN Graphics.....	4-1
Setting the Graphical Environment.....	4-2
Levels of Attributes.....	4-2
Object Attributes.....	4-2
Diagram Default Attributes.....	4-2
System Default Attributes.....	4-2
Changing Display Attributes.....	4-2
Establishing an Environment.....	4-3
Creating CP Nets.....	4-4
General Technique for Creating CPN Regions.....	4-4
Creating a Transition.....	4-4
Naming a Transition.....	4-6
Creating a Guard.....	4-7
Creating a Place.....	4-7
Naming a Place.....	4-8
Specifying a Place's Colorset.....	4-9
Specifying a Place's Initial Marking.....	4-9
Input and Output Places.....	4-10
Creating an Arc.....	4-10
Creating an Arc Inscription.....	4-11
Cloning CPN Regions.....	4-11
Creating a Global Declaration Node.....	4-12
Creating a Page for Global Declarations.....	4-14
Creating a Page.....	4-14
Naming a Page.....	4-14
Renaming a Page.....	4-15
Moving a Global Declaration Node.....	4-15

Chapter 5

CPN Dynamics: Introduction

Executing CP Nets.....	5-1
The Design/CPN Simulator.....	5-1
Understanding CP Net Execution.....	5-2
When Can a Transition Occur?.....	5-2
Factors Determining Enablement.....	5-2
Input Place Multiset.....	5-3
Input Arc Inscriptions.....	5-3
Guards.....	5-3
Criteria for Enablement.....	5-3
Specifying Exact Token Values.....	5-4
Specifying a Single Token.....	5-4
The Simulator's Algorithm.....	5-5
Omitting a Count of One.....	5-5
Specifying More Than One Token Instance.....	5-5
Specifying More Than One Token Value.....	5-6
The General Rule.....	5-6
Non-enabled transitions.....	5-6
Specifying Variable Token Values.....	5-7
Binding an Arc Inscription Variable.....	5-7
Constraining Token Values With Guards.....	5-8
Guard Syntax.....	5-8
Use of Parentheses.....	5-9
Shortcut for andalso.....	5-9
Constraining a Single Token.....	5-9
More Complex Constraints.....	5-10
Constraining More Than One Token.....	5-11
Using a Guard to Create a Partial Constraint.....	5-12
What Happens When a Transition Occurs.....	5-12
CP Net Execution Example.....	5-13
Rebind Any CPN Variables per the Enabling Binding.....	5-13
Evaluate Each Input Arc Inscription.....	5-13
Remove the Enabling Multiset from Each Input Place.....	5-13
Evaluate Each Output Arc Inscription.....	5-14
Put the Output Multiset into the Output Place.....	5-14
Saving and Loading Execution States.....	5-15
Saving a State.....	5-15
Loading a Saved State.....	5-16
Starting With a Saved State.....	5-16

Chapter 6

CPN Dynamics: Executing A CP Net

Performing a Syntax Check.....	6-1
Performing the Check.....	6-2
Designating a Prime Page.....	6-3
Setting Up a Net for Execution.....	6-5
Generating Code for Different Types of Simulation.....	6-5
Generating Code for the Method of Simulation.....	6-7
Time Section.....	6-7
Code Segments Section.....	6-8
Selecting Different Types of Simulation.....	6-8
Specifying the Type of Execution.....	6-8
Specifying Termination Conditions.....	6-9
No Limit.....	6-9
Additional Steps.....	6-9
Until Step Number Is.....	6-9
Additional Time.....	6-9
Until Time is.....	6-9
Until Time Advances.....	6-9
Recording the Results of Execution.....	6-9
None.....	6-9
Step Information.....	6-10
Bindings.....	6-10
Simulation Report Example.....	6-10
Adding Information to the Report.....	6-11
Entering the Simulator.....	6-11
Entering the Simulator from the Editor After a Syntax Check.....	6-11
Entering the Simulator With a Saved State.....	6-12
The Sim Menu.....	6-12
Simulation Regions.....	6-13
Simulation Region Types.....	6-14
Simulation Regions Indicating Place Markings.....	6-14
Simulation Region Indicating Enablement and Firing.....	6-14
Key and Popup Regions.....	6-14
Repositioning Key and Popup Regions.....	6-15
Setting Interactive Options.....	6-15
Breakpoints Section.....	6-16
Update Graphics Section.....	6-16
Running an Interactive Simulation.....	6-16
Continuing from a Substep Breakpoint.....	6-16
Re-Executing a Net.....	6-18
Leaving the Simulator.....	6-18
Removing Simulation Regions.....	6-18
Removing Simulation Regions from Multiple Pages.....	6-19

Chapter 7

CPN Dynamics: Handling CP Net Syntax Errors

Missing Colorset Specification.....	7-2
Locating a Syntax Error.....	7-2
Text Pointers.....	7-3
Undeclared Variables.....	7-4
Illegal CPN ML Constructs.....	7-6

Chapter 8

CPN Dynamics: Concurrency and Choice

Concurrency Problems.....	8-1
Representing Concurrency.....	8-2
Multiple Enabling Bindings.....	8-2
Concurrent Transition Firing.....	8-3
Identical Enabling Bindings.....	8-3
Concurrent CP Net Execution.....	8-4
Initial State of the Net.....	8-4
Breakpoint 1: Beginning of Substep.....	8-4
Breakpoint 2: End of Substep.....	8-4
Execution Is Complete.....	8-4
Analysis of the Execution.....	8-5
Representing Conflict.....	8-5
Conflicts and Bindings.....	8-6
Concurrent Execution of SalesNet.....	8-7
Initial State of the Net.....	8-7
Breakpoint 1: Beginning of Substep.....	8-8
Breakpoint 2: End of Substep.....	8-8
Execution Is Complete.....	8-9
Changing a Net in the Simulator.....	8-9
The Simulator's Execution Algorithm.....	8-11
Executing SalesNet With Conflict.....	8-12
1: Establish Initial Markings.....	8-12
2: Put All Enabled Transitions on the Enabled List.....	8-12
3A: Scan the Enabled List and Construct an Occurrence Set.....	8-12
3B: Execute the Elements in the Occurrence Set.....	8-13
Executing an Occurrence Set.....	8-14
SalesNet's Appearance at Breakpoint 1.....	8-14
SalesNet's Appearance at Breakpoint 2.....	8-15
3C: Update the Enabled List.....	8-16
4: Continue Execution.....	8-17
5: Complete Execution.....	8-17
Controlling the Appearance of Concurrency.....	8-18
Review of Occurrence Sets.....	8-18
Constructing an Occurrence Set.....	8-18
What Is Concurrency?.....	8-19

Chapter 8

CPN Dynamics: Concurrency and Choice (cont'd)

Occurrence Set Parameters.....	8-20
Transitions.....	8-20
Different Bindings.....	8-21
Identical Bindings.....	8-21
Scope of Occurrence Set Parameters.....	8-21
Setting Occurrence Set Parameters.....	8-22

Chapter 9

CPN Hierarchy: Introduction

Definition of Hierarchical Decomposition.....	9-1
CPN Hierarchy.....	9-2
Fusion Places.....	9-2
Substitution Transitions.....	9-3
Top-Down and Bottom-Up Development.....	9-3

Chapter 10

CPN Hierarchy: Fusion Places

The Resource Use Model.....	10-1
Description of the Model.....	10-2
Results of Executing the Model.....	10-3
Fusion on a Single Page.....	10-3
Creating a Fusion Set.....	10-4
Physical Appearance of a Global Fusion Place.....	10-5
Adding Places to a Fusion Set.....	10-6
Initial Markings and Fusion Sets.....	10-7
Removing Places from a Fusion Set.....	10-7
Deleting a Fusion Set.....	10-8
Fusion Across More Than One Page.....	10-8
Working With More Than One Fusion Set.....	10-9
Page Fusion Sets.....	10-9
Relationship Among Page Fusion Constituents.....	10-11
Instance Fusion Sets.....	10-11
Creating Multiple Page Instances.....	10-11
Multiplicity and Fusion.....	10-12
Comparison With Page Fusion Sets.....	10-13
Working With Instance Fusion Sets.....	10-13
Observing Fusion Across Multiple Instances.....	10-14

Chapter 11

CPN Hierarchy: Substitution Transitions

Structure of a Model With Substitution.....	11-2
ResmodSubtrans Component Pages.....	11-2
The Hierarchy Page.....	11-2
The Superpage Resmod#1.....	11-3
The Subpage New#2.....	11-3
Ports and Sockets.....	11-4
Jumping Directly to a Superpage.....	11-5
Moving Existing Net Structure to a Subpage.....	11-5
ResourceModel Component Pages.....	11-5
Hierarchy Page.....	11-5
Resmod#1 Page.....	11-6
Designate the Net Components to Move to the Subpage.....	11-6
Initiate Subpage Creation.....	11-6
Specifying the Substitution Transition's Location.....	11-6
Name the Substitution Transition (If Desired).....	11-8
Improving a Superpage's Appearance.....	11-8
Rerouting an Arc.....	11-8
Moving Regions.....	11-9
Improving a Subpage's Appearance.....	11-11
Improving a Hierarchy Page's Appearance.....	11-12
Developing on a Subpage.....	11-14
Development Procedure.....	11-14
Creating the Substitution Transition.....	11-15
The Modified Hierarchy Page.....	11-16
The New Subpage.....	11-16
Using a Subpage More Than Once.....	11-17
Relationship of Pages in a Hierarchy.....	11-20
Subpages, Subroutines, and Macros.....	11-21
Removing Hierarchical Constructs.....	11-21
Reversing Substitution Transition Creation.....	11-21
Status of the Model.....	11-23
Deleting a Subpage.....	11-23
Status of the Model.....	11-24
Deleting a Reference to a Subpage.....	11-24
Manually Assigning Ports to Sockets.....	11-25

Chapter 12

Simulated Time

The Nature of Simulated Time.....	12-2
Representing Time in a CP Net.....	12-2
How Simulated Time Works.....	12-3
Simulated Time and Transition Enablement.....	12-3
The Simulated Clock.....	12-3
Other Uses for Simulated Time.....	12-4

Chapter 12

Simulated Time (cont'd)

Specifying Timed Simulation.....	12-4
Additional Time.....	12-6
Until Time is.....	12-6
Until Time Advances.....	12-6
Declaring a Timed Colorset.....	12-6
Giving a Token a Time Stamp.....	12-7
Delay Expressions in Time Regions.....	12-7
Delay Expressions on Output Arc Inscriptions.....	12-8
Omitting a Time Stamp.....	12-8
Time Stamps and Initial Markings.....	12-9
Time Stamps and Multisets.....	12-10
Example of Time Region Use.....	12-10
Executing a Timed CP Net.....	12-11
Simulation With and Without Time.....	12-14
More Realistic Timed Behavior.....	12-15

Chapter 13

Code Segments

Simulation with Code.....	13-1
Characteristics and Syntax.....	13-1
Input pattern.....	13-2
Output pattern.....	13-2
Code actions.....	13-2
Log regions.....	13-3

Chapter 14

Statistical Variables

Using Statistical Variables.....	14-1
Structure of Statistical Variables.....	14-2
Creating Statistical Variables.....	14-2
Initializing Statistical Variables.....	14-2
Updating Statistical Variables.....	14-3
Accessing Statistical Variables.....	14-3
Average.....	14-3
Count.....	14-3
Current Value.....	14-4
First.....	14-4
Maximum.....	14-4
Minimum.....	14-4
Standard Deviation.....	14-4
Sum.....	14-5
Sum of the Squares.....	14-5

Chapter 14 Statistical Variables (cont'd)

Accessing Statistical Variables (cont'd)	
Sum of the Squares of Deviation.....	14-5
Variance.....	14-5
Clearing Statistical Variables.....	14-5

Chapter 15 Chart Facilities

Overview of Charts.....	15-1
Creating Charts.....	15-1
Updating Charts.....	15-1
Bar Charts.....	15-2
Bar Chart Nomenclature.....	15-2
Chart Node.....	15-2
Title.....	15-3
Bars.....	15-3
Bar Names.....	15-3
Positive and Negative Values.....	15-3
Upper and Lower Values.....	15-3
Legend.....	15-3
Patterns.....	15-3
Legend Labels.....	15-4
Creating a Bar Chart.....	15-4
Bar Chart Dialog.....	15-5
Editing a Bar Chart.....	15-6
Supplying Bar Chart Labels.....	15-6
Redefining a Bar Chart.....	15-7
Updating a Bar Chart.....	15-8
Specifying Bar Chart Values.....	15-9
Requirement to Specify Values.....	15-10
Other Ways to Update a Bar Chart.....	15-10
Additional Chart Code Segment Capabilities.....	15-11
Initialization Code.....	15-11
Conditional Chart Update.....	15-12
Conditional Bar Update.....	15-12
Arbitrary Code in the Action Section.....	15-13
Copying, Cutting, and Pasting a Bar Chart.....	15-13
Deleting a Bar Chart.....	15-14
History Charts.....	15-14
Creating a History Chart.....	15-14
Retain.....	15-15
Other History Chart Options.....	15-16
Editing and Redefining a History Chart.....	15-16
Updating a History Chart.....	15-16
Specifying History Chart Values.....	15-16

Chapter 15 Chart Facilities (cont'd)

History Charts (cont'd)	
Copying, Cutting, Pasting, and Deleting a History Chart.....	15-17
Line Charts.....	15-17
Line Chart Nomenclature.....	15-17
Chart Node.....	15-18
Title.....	15-18
X and Y Axes.....	15-18
Axis labels.....	15-18
Box.....	15-18
Line.....	15-19
Legend.....	15-19
Patterns.....	15-19
Legend labels.....	15-19
Creating a Line Chart.....	15-19
Line Chart Dialog.....	15-20
Editing a Line Chart.....	15-21
Supplying Line Chart Labels.....	15-21
Redefining a Line Chart.....	15-23
Updating a Line Chart.....	15-24
Specifying Line Chart Values.....	15-25
Requirement to Specify Values.....	15-27
Other Ways to Update a Line Chart.....	15-27
Additional Chart Code Segment Capabilities.....	15-28
Initialization Code.....	15-28
Conditional Chart Update.....	15-29
Conditional Line Update.....	15-29
Conditional Drawing of Line Segments.....	15-30
Arbitrary Code in the Action Section.....	15-31
Copying, Cutting, and Pasting a Line Chart.....	15-31
Deleting a Line Chart.....	15-31

Chapter 16 Using Charts and Statistical Variables

Introduction to the Resource Use Model.....	16-1
Overview of the Model.....	16-1
Resource Use Model Data.....	16-2
Processes.....	16-2
Resources.....	16-2
Allocation Patterns.....	16-2
Resource Use Model Graphics and Global Declarations.....	16-3
Running the Model.....	16-4
Statistics in the Resource Use Model.....	16-5
Initializing the Statistical Variables.....	16-5
Updating the Statistical Variables.....	16-6

Chapter 16

Using Charts and Statistical Variables (cont'd)

Statistics in the Resource Use Model (cont'd)	
Using the Accumulated Data.....	16-7
Resource Use Model: Bar Charts.....	16-8
Definition of the Bar Chart.....	16-9
Analysis of the Bar Chart Data.....	16-10
Resource Use Model: History Charts.....	16-10
Definition of the History Chart.....	16-11
Analysis of the History Chart Data.....	16-11
Resource Use Model: Line Charts.....	16-12
Definition of the Line Chart.....	16-13
Analysis of the Line Chart Data.....	16-13
Improving the Resource Use Model.....	16-15
The Essential Problem.....	16-15
Possible Solutions.....	16-15
Seeking a Solution.....	16-15

Chapter 17

Reference Summaries

Navigating a Diagram.....	17-1
Navigating Objects.....	17-1
Occlusion Order on a Page.....	17-1
Layering Order for Connectors.....	17-1
Layering Order for Regions.....	17-1
Means of Navigation.....	17-2
Arrow Keys.....	17-2
Navigating in a Non-Text Mode.....	17-2
Navigating in Text Mode (Not Group Mode).....	17-3
Double-Clicking.....	17-3
Double-Clicking a Substitution Transition.....	17-3
Double-Clicking a Port Place.....	17-3
Double-Clicking a Page Node on the Hierarchy Page.....	17-3
Double-Clicking for Editor Regions.....	17-3
Double-Clicking for Simulator Regions.....	17-3
Commands.....	17-4
Parent Object and Child Object Commands.....	17-4
Next Object and Previous Object Commands.....	17-4
Select Command.....	17-4
Graphical Objects.....	17-5
Essential Graphical Objects.....	17-5
Classes of Objects.....	17-6
CPN Objects and Object Types.....	17-6
Auxiliary Objects and Object Types.....	17-7
System Objects and Object Types.....	17-7

Chapter 17 Reference Summaries (cont'd)

Graphical Objects (cont'd)	
Individual Classes and Object Types.....	17-7
Place.....	17-8
Transition.....	17-8
Arc.....	17-9
Declaration Nodes.....	17-9
Chart Nodes.....	17-9
Regions.....	17-10
Auxiliary Object Types.....	17-10
System Object Types.....	17-10
System Nodes.....	17-10
System Connectors.....	17-10
System Regions.....	17-10
Special Regions.....	17-11
Key and Popup Regions.....	17-11
Editor (Key/Popup) Regions.....	17-12
Simulator (Key/Popup) Regions.....	17-12
Creating And Deleting Key/Popup Regions.....	17-13
Editing Key/Popup Regions.....	17-13
Endpoint Regions.....	17-13
Creating Endpoint Regions.....	17-14
Preferences.....	17-15
Attributes and Options.....	17-15
Object Attributes.....	17-15
Page Attributes.....	17-15
Mode Attributes.....	17-16
Special Attributes.....	17-16
Graphical Attributes: Visibility and Selectability.....	17-16
Visibility.....	17-16
Changing Visibility.....	17-17
Selectability.....	17-17
Changing Selectability.....	17-17
Options.....	17-17
Diagram and System Defaults.....	17-18
System Defaults.....	17-18
CPN Settings.....	17-18
Diagram Defaults.....	17-18
Changing System And Diagram Defaults.....	17-19
Using System And Diagram Defaults.....	17-19
Copying Defaults.....	17-19
Names and Numbers.....	17-19
Using Names and Numbers.....	17-20
Different Kinds of Names.....	17-20
Syntax for Names.....	17-20
Syntax for Numbers.....	17-20
Generating Names.....	17-20

Chapter 17

Reference Summaries (cont'd)

Names and Numbers (cont'd)	
Entering and Changing Numbers and Names.....	17-21
Entering and Changing Place and Transition Names.....	17-21
Fusion Names.....	17-21
List of Compulsory Syntax Restrictions.....	17-22
Format for Syntax Violations.....	17-22
C.1 Missing global declaration node.....	17-22
C.2 Too many global/temporary declaration nodes.....	17-22
C.3 Too many local declaration nodes.....	17-22
C.4 Error in global/temporary declaration node.....	17-22
C.5 Error in local declaration node.....	17-22
C.6 Place must have a color set.....	17-23
C.7 Color set region must be legal.....	17-23
C.8 Initial marking must be legal.....	17-23
C.9 Guard region must be legal.....	17-23
C.10 Code segment must be legal.....	17-23
C.11 Arc expression must be legal.....	17-23
C.12 CPN variables in code segment.....	17-24
C.13 Code guard must be legal.....	17-24
C.14 Code action must be legal.....	17-24
C.15 Color set in port assignment.....	17-24
C.16 Color set in fusion set.....	17-24
C.17 Initial marking in fusion set.....	17-24
C.18 Socket must be assigned.....	17-25
C.19 Port type must match socket.....	17-25
C.20 Time region must be legal.....	17-25

Part 2: Design/CPN Menu Reference

Chapter 18

Overview of Design/CPN Menus

File Menu.....	18-1
Edit Menu.....	18-1
CPN Menu.....	18-1
Sim Menu.....	18-2
Aux Menu.....	18-2
Set Menu.....	18-2
Makeup Menu.....	18-2
Page Menu.....	18-2
Group Menu.....	18-2
Text Menu.....	18-3
Align Menu.....	18-3

Chapter 19

File Menu Commands

New.....	19-2
Open.....	19-2
Saved States.....	19-2
Close.....	19-3
Save.....	19-3
Save As.....	19-5
Revert.....	19-5
Page Preview.....	19-6
Page Setup.....	19-6
Output Form.....	19-6
Omit Page Borders.....	19-7
Print Hierarchy Page.....	19-7
Print.....	19-7
Save Subdiagram.....	19-7
Load Subdiagram.....	19-8
Save Text.....	19-9
Restrictions.....	19-9
Load Text.....	19-9
Blocking Options.....	19-9
Load IDEF.....	19-10
Save State.....	19-11
Load State.....	19-11
Loading Saved States With Open (File Menu).....	19-12
Restrictions.....	19-12
Enter Editor.....	19-12
Enter Simulator.....	19-13
Quit.....	19-14

Chapter 20 Edit Menu Commands

Undo/Redo.....	20-2
Cut.....	20-2
Effect on Object Types.....	20-2
Nodes.....	20-2
Connectors.....	20-2
Regions.....	20-2
Groups of Nodes.....	20-2
Text.....	20-3
Effect on Hierarchies.....	20-3
Restrictions.....	20-3
Copy.....	20-3
Effect on Object Types.....	20-3
Nodes.....	20-3
Connectors.....	20-4
Regions.....	20-4
Groups of Nodes.....	20-4
Text.....	20-4
Effect on Hierarchies.....	20-4
Restrictions.....	20-4
Paste.....	20-5
Effect on Object Types.....	20-5
Nodes.....	20-5
Connectors.....	20-5
Auxiliary Regions.....	20-5
CPN Regions.....	20-5
Groups of Nodes.....	20-5
Text.....	20-6
Clear.....	20-6
Special Cases.....	20-6
Page Nodes.....	20-6
Page Connectors or Page Tags.....	20-6
Hierarchy Regions.....	20-6
Socket or Port Places.....	20-6
Hierarchy, Fusion, Border, Code, or Mode Region.....	20-6
Get Info.....	20-7
Single Object Display.....	20-7
Group Display.....	20-7
Examples of Get Info Dialogs.....	20-8

Chapter 21 CPN Menu Commands

Place.....	21-2
Text Use.....	21-2
Changing Size and Position.....	21-2

Chapter 21 CPN Menu Commands (cont'd)

Place (cont'd)	
Creating Multiple Places.....	21-2
Terminating Place Creation Mode.....	21-2
Transition.....	21-3
Text Use.....	21-3
Changing Size and Position.....	21-3
Creating Multiple Transitions.....	21-3
Terminating Transition Creation Mode.....	21-3
Arc.....	21-4
Restrictions.....	21-4
Drawing Single-Segment Connectors.....	21-4
Drawing Multi-Segment Connectors.....	21-4
Effect of Moving Source/Destination Nodes.....	21-4
Endpoint Handle Use.....	21-5
Terminating Arc Creation Mode.....	21-5
CPN Region.....	21-5
Restrictions.....	21-5
Creating CPN Regions.....	21-5
Places.....	21-5
Transitions.....	21-6
Arcs.....	21-6
Multiple CPN Regions.....	21-6
Object Descendants.....	21-7
Text.....	21-7
Terminating CPN Region Creation Mode.....	21-7
Declaration Node.....	21-7
Changing Size and Position.....	21-8
Creating Multiple Declaration Nodes.....	21-8
Restrictions.....	21-8
Terminating Declaration Node Mode.....	21-8
Chart.....	21-8
Bar Chart Dialog.....	21-9
Name.....	21-10
Bars.....	21-10
No of Bars.....	21-10
No of Parts.....	21-10
Bar Height.....	21-10
History Chart.....	21-10
Retain.....	21-10
Grid Lines.....	21-11
Number.....	21-11
Distance.....	21-11
Tics Only.....	21-11
Regions.....	21-11
Title.....	21-11
Legend.....	21-11

Chapter 21
CPN Menu Commands (cont'd)

Chart (cont'd)

Regions (cont'd)

Bar Names.....	21-11
Pos. Values.....	21-11
Neg. Values.....	21-11
Upper Values.....	21-12
Lower Values.....	21-12
Initialization.....	21-12
Save Copy.....	21-12
Keep Contents.....	21-12
Update Period.....	21-12
End of Run.....	21-12
Time.....	21-12
Step.....	21-12
Value Type.....	21-13
Value Range.....	21-13
Line Chart Dialog.....	21-13
Name.....	21-14
Lines.....	21-14
No of lines.....	21-14
Box Size.....	21-14
Start at Origin.....	21-14
Horiz/Vert.....	21-14
Grid Lines.....	21-15
Number.....	21-15
Distance.....	21-15
Tics Only.....	21-15
Regions.....	21-15
Title.....	21-15
Legend.....	21-15
Axis Names.....	21-15
Initialization.....	21-15
Save Copy.....	21-15
Keep Contents.....	21-16
Update Period.....	21-16
End of Run.....	21-16
Time.....	21-16
Step.....	21-16
Value Type.....	21-16
Value Range.....	21-16
Origin.....	21-16
Overflow.....	21-17
Rescale Axis.....	21-17
Move Axis.....	21-17

Chapter 21

CPN Menu Commands (cont'd)

Move to Subpage.....	21-17
Terms.....	21-17
External Arcs.....	21-17
Perimeter Transitions.....	21-17
Socket Places.....	21-18
Port Places.....	21-18
Substitution Transition.....	21-18
Creating the Subpage.....	21-18
Subpage Contents.....	21-18
Hierarchy Regions.....	21-18
Replace by Subpage.....	21-19
Substitution Transition.....	21-19
Restrictions.....	21-20
Fusion Place.....	21-20
Create and Rename/Retype.....	21-21
Add to.....	21-22
Subtract from and Delete.....	21-22
Naming Conventions.....	21-23
Moving Places Between Fusion Sets.....	21-23
Turning Fusion Places Into Non-Fusion Places.....	21-23
Retyping Global Fusion Sets.....	21-23
Port Place.....	21-23
Port Types.....	21-24
General Port.....	21-24
Input Port.....	21-24
Output Port.....	21-24
Input & Output Port.....	21-24
Using Port Place.....	21-24
Port Assignment.....	21-24
Selecting Ports.....	21-25
Restrictions.....	21-25
Input Place.....	21-25
Output Place.....	21-25
Both an Input Place and an Output Place.....	21-25
Place of More Than One Substitution Transition.....	21-25
Port Assignment Exists.....	21-25
Deleting Port Assignments.....	21-25
Syntax Check.....	21-26
Error Reporting.....	21-26
Following the Text Pointers.....	21-26
Error Nodes.....	21-26
Status Bar Information.....	21-27
Specifying the ML File Name.....	21-27
Remove Sim Regions.....	21-28

Chapter 22

Sim Menu Commands

Bind.....	22-2
Bind Dialog Format.....	22-2
Editing Area.....	22-3
Selecting a Variable.....	22-3
Editing the Contents.....	22-3
Values for the Variables.....	22-4
Included and Potential Bindings.....	22-4
Syntax.....	22-4
Included Bindings List Box:.....	22-4
Potential Bindings List Box:.....	22-5
Included Bindings List Box.....	22-5
Potential Bindings List Box.....	22-5
Control Buttons.....	22-5
Upper Control Button Bar.....	22-6
Clear Button.....	22-6
One Button.....	22-6
All Button.....	22-6
Down and Up Buttons.....	22-6
Lower Control Button Bar.....	22-8
Reset Button.....	22-8
Cancel Button.....	22-8
OK Button.....	22-8
Occur Button.....	22-8
Invoking Bind Without the Bind Dialog.....	22-8
Occurrence Set.....	22-9
New Occurrence Set.....	22-9
Add To Occurrence Set.....	22-9
Delete Occurrence Set.....	22-9
Start Step.....	22-9
Interactive Run.....	22-10
Automatic Run.....	22-10
Continue.....	22-10
Stop.....	22-10
Change Marking.....	22-11
Initial Marking.....	22-11
Specified Marking.....	22-11
Empty Marking.....	22-11
Select Instance.....	22-12
Page Instances.....	22-12
Full Name.....	22-12
Select.....	22-13
Save Report.....	22-13
Clear Report.....	22-13
Update Chart.....	22-13
Reswitch.....	22-13

Chapter 23 Aux Menu Commands

Connector.....	23-2
Drawing Single-Segment Connectors.....	23-2
Drawing Multi-Segment Connectors.....	23-2
Effect of Moving Source/Destination Nodes.....	23-2
Endpoint Handle Use.....	23-2
Terminating Connector Creation Mode.....	23-3
Box.....	23-3
Text Use.....	23-3
Changing Size and Position.....	23-3
Creating Multiple Boxes.....	23-3
Terminating Box Creation Mode.....	23-3
Rounded Box.....	23-4
Text Use.....	23-4
Changing Size and Position.....	23-4
Creating Multiple Rounded Boxes.....	23-4
Terminating Rounded Box Creation Mode.....	23-4
Ellipse.....	23-4
Text Use.....	23-5
Changing Size and Position.....	23-5
Creating Multiple Ellipses.....	23-5
Terminating Ellipse Creation Mode.....	23-5
Polygon.....	23-5
Text Use.....	23-6
Changing Size and Position.....	23-6
Creating Multiple Polygons.....	23-6
Drawing Polygons.....	23-6
Terminating Polygon Creation Mode.....	23-6
Regular Polygon.....	23-7
Text Use.....	23-7
Changing Size and Position.....	23-7
Creating Multiple Rounded Boxes.....	23-7
Terminating Regular Polygon Creation Mode.....	23-7
Wedge.....	23-7
Text Use.....	23-8
Changing Size and Position.....	23-8
Creating Multiple Wedges.....	23-8
Terminating Wedge Creation Mode.....	23-8
Line.....	23-8
Terminating Line Creation Mode.....	23-8
Label.....	23-9
Dimensions.....	23-9
Restrictions.....	23-9
Moving Labels.....	23-9
Terminating Label Creation Mode.....	23-9

Chapter 23 Aux Menu Commands (cont'd)

Make Region.....	23-10
Creating a Region.....	23-10
Effect on Parents.....	23-10
Restrictions.....	23-10
Make Node.....	23-11
Convert to CPN.....	23-11
Nodes.....	23-11
Regions.....	23-11
Connectors.....	23-11
Dialogs.....	23-11
Nodes Selected:.....	23-12
Place Regions Selected:.....	23-12
Transition Regions Selected:.....	23-12
Convert to Aux.....	23-13
Nodes.....	23-13
Regions.....	23-13
Arcs.....	23-13
Restrictions.....	23-13
Start ML.....	23-13
Stop ML.....	23-13
ML Evaluate.....	23-14

Chapter 24 Set Menu Commands

Text Attributes.....	24-2
Saving Settings.....	24-2
Graphic Attributes.....	24-3
Saving Settings.....	24-3
Layering Logic.....	24-4
Copy.....	24-4
Selectable.....	24-4
Line Thickness.....	24-4
Line and Fill Pattern.....	24-5
Shape Attributes.....	24-5
Saving Settings.....	24-5
Dialog Determination.....	24-6
Initial State of the Dialog.....	24-6
Dialogs.....	24-6
Box Shape, Ellipse Shape, Picture Shape.....	24-6
Rounded Box Shape.....	24-7
Regular Polygon Shape.....	24-7
Wedge Shape.....	24-7
Connector Shape.....	24-8

Chapter 24

Set Menu Commands (cont'd)

Region Attributes.....	24-8
Initial State of the Dialog.....	24-8
Position.....	24-9
Nodes.....	24-9
Arcs.....	24-9
Size.....	24-9
Color.....	24-9
Popup.....	24-9
Page Attributes.....	24-10
Name & No.....	24-10
Master Page.....	24-10
Page Kind.....	24-10
Palette Page.....	24-10
Change On-Screen Page Border Size.....	24-11
Make Page Borders Visible or Invisible.....	24-11
Invisible.....	24-11
Saving Settings.....	24-11
Restrictions.....	24-12
Mode Attributes.....	24-12
Substitution Transitions Selected.....	24-12
Page Nodes Selected.....	24-13
Change Mode Attributes of Supernodes.....	24-13
Prime.....	24-13
Multiplicity.....	24-13
Chart Attributes.....	24-13
Bar Charts.....	24-13
Line Charts.....	24-14
Dialogs.....	24-14
Hierarchy Page Options.....	24-14
Interaction Options.....	24-15
Pages Overlay Palettes.....	24-15
Show Window Scroll Bars.....	24-15
Auto Scroll to Selected Object.....	24-15
Nodes Overlay Connector.....	24-15
Scroll Rate.....	24-15
Flash Rate.....	24-16
Merge Options.....	24-16
Merge Arcs.....	24-16
Merge Aux Connectors.....	24-16
Merge Text in Nodes.....	24-16
Merge Text in CPN Regions.....	24-17
Text Options.....	24-17
Load Brackets and Load Counts.....	24-17
Select Brackets.....	24-17

Chapter 24**Set Menu Commands (cont'd)**

Syntax Options.....	24-18
Missing Place Names.....	24-18
Duplicate Place Names.....	24-18
ML-illegal Place Names.....	24-19
Missing Transition Names.....	24-19
Duplicate Transition Names.....	24-19
ML-Illegal Transition Names.....	24-19
Duplicate Page Names.....	24-19
ML-Illegal Page Names.....	24-19
Missing Arc Expressions (Ordinary).....	24-19
Missing Arc Expressions (Substitutions).....	24-20
Names Illegal for OG Analyzer.....	24-20
Simulation Code Options.....	24-20
Mode.....	24-21
Fair Simulation (Interactive and Automatic).....	24-21
Fast Simulation (Automatic).....	24-21
Both.....	24-21
Time.....	24-21
With.....	24-21
Without.....	24-21
Both.....	24-21
Integer.....	24-22
Real.....	24-22
Code Segments.....	24-22
With.....	24-22
Without.....	24-22
Both.....	24-22
General Simulation Options.....	24-23
Simulate With.....	24-23
Fair Interactive.....	24-23
Fair Automatic.....	24-23
Fast Automatic.....	24-23
Time.....	24-24
Code Segments.....	24-24
Stop Criteria.....	24-24
No Limit.....	24-24
Additional Steps.....	24-24
Until Step Number Is.....	24-24
Additional Time.....	24-25
Until Time is.....	24-25
Until Time Advances.....	24-25
Record.....	24-25
None.....	24-25
Step Information.....	24-25
Bindings.....	24-26
Adding Information to the Report.....	24-26

Chapter 24

Set Menu Commands (cont'd)

Interactive Simulation Options.....	24-27
Breakpoints.....	24-27
Update Graphics.....	24-27
During Substeps.....	24-27
Between Substeps.....	24-28
Between Steps.....	24-28
End of Run.....	24-28
Occurrence Set Options.....	24-28
Settings.....	24-29
Pages.....	24-29
Page Instances.....	24-30
Transitions.....	24-30
Different Bindings.....	24-30
Identical Bindings.....	24-30
Occurrence Rule.....	24-30
Mutual Influence.....	24-30
Random Generator.....	24-31
Transition Feedback Options.....	24-31
ML Configuration Options.....	24-32
Setting the System Defaults.....	24-33
Setting the Diagram Defaults.....	24-33
Copying the System Defaults to the Diagram.....	24-33
Copy Defaults.....	24-33

Chapter 25

Makeup Menu Commands

Select.....	25-2
Selecting Visible Objects.....	25-2
Selecting Hidden Objects.....	25-2
Drag.....	25-2
Dragging Text.....	25-2
Dragging the Endpoints of a Connector.....	25-3
Endpoint Source.....	25-3
Node Source.....	25-3
Target.....	25-3
Endpoint Target.....	25-3
Node Target.....	25-3
Terminating Drag Mode.....	25-3
Displace.....	25-4
Adjust.....	25-4
Resizing in Two Dimensions.....	25-4
Resizing in One Dimension.....	25-4
Resizing Regions and Labels.....	25-4

Chapter 25
Makeup Menu Commands (cont'd)

Fit to Text.....	25-5
Width.....	25-5
Height.....	25-5
Word Wrap.....	25-5
Labels and Key Regions.....	25-5
Change Shape.....	25-5
Shape Changing.....	25-5
Region Changing.....	25-6
Duplicate Node.....	25-6
Duplicating Regions and Connectors.....	25-6
Text Mode Impact on Duplication.....	25-6
Effect on Hierarchy Objects.....	25-6
Move Node.....	25-7
Moving Nodes.....	25-7
Effect on Regions and Connectors.....	25-7
Effect on Hierarchy Objects.....	25-7
Substitution Transitions.....	25-7
Fusion Places.....	25-7
Port Places.....	25-8
Restrictions.....	25-8
Merge Node.....	25-8
Restrictions.....	25-8
Regions.....	25-8
Connectors.....	25-8
Targets.....	25-9
Hide Regions.....	25-9
Nodes and Connectors.....	25-9
Regions.....	25-9
Show Regions.....	25-9
Bring Forward.....	25-9
Reordering Object Layering.....	25-9
Restrictions.....	25-10
Parent Object.....	25-10
Text Pointers.....	25-10
Child Object.....	25-10
Substitution Transitions.....	25-10
Regions.....	25-11
Text Pointers.....	25-11
Next Object.....	25-11
Previous Object.....	25-11

Chapter 26

Page Menu Commands

New Page.....	26-2
Open Page.....	26-2
Close Page.....	26-2
Scroll.....	26-2
Blowup.....	26-2
Reduce.....	26-3
Cleanup.....	26-3
Redraw Hierarchy.....	26-3

Chapter 27

Group Menu Commands

Enter/Leave Group Mode.....	27-2
Enter Group Mode.....	27-2
Leave Group Mode.....	27-2
Select All Nodes.....	27-2
Select All Regions.....	27-2
Select All Connectors.....	27-2
Select Fusion Set.....	27-3

Chapter 28

Text Menu Commands

Enter/Leave Text Mode.....	28-2
Enter Text Mode.....	28-2
Leave Text Mode.....	28-2
Group Text Mode.....	28-2
Find.....	28-3
Search Method.....	28-3
Search.....	28-3
Replace.....	28-3
Replace All.....	28-3
Match Case.....	28-3
Search Domain.....	28-4
Hypertext.....	28-4
Current Object.....	28-4
Pages.....	28-4
Document.....	28-4
Searching and Replacing in a Group.....	28-4
Find Next.....	28-4
Find Beginning.....	28-4
Select All Text.....	28-5
Select to.....	28-5
Select to Bracket.....	28-5

Chapter 29 Align Menu Commands

Horizontal.....	29-2
Group Alignment.....	29-2
Shift.....	29-2
Option.....	29-2
Vertical.....	29-2
Group Alignment.....	29-2
Center.....	29-2
Group Alignment.....	29-3
Position.....	29-3
Group.....	29-3
Group Alignment.....	29-3
Dialogs.....	29-3
Cartesian Coordinates.....	29-3
Polar Coordinates.....	29-4
Horizontal Spread.....	29-4
Targets.....	29-4
To Perform the Horizontal Spread.....	29-4
Effect of the Horizontal Spread.....	29-4
Reference Objects Far Apart.....	29-5
Reference Objects Close Together.....	29-5
Vertical Spread.....	29-5
Circular Spread.....	29-5
Between.....	29-5
Projection.....	29-5
Left to Left.....	29-6
Left to Right.....	29-6
Right to Left.....	29-6
Right to Right.....	29-6
Top to Top.....	29-7
Top to Bottom.....	29-7
Bottom to Top.....	29-7
Bottom to Bottom.....	29-7

Part 3: CPN ML Reference

Chapter 30

Introduction to CPN ML

Standard ML Features.....	30-1
CPN ML Extensions to Standard ML.....	30-2
Colorsets.....	30-2
CPN Variables.....	30-2
Reference Variables.....	30-3
Declaration Nodes.....	30-3
CPN ML Restrictions and Modifications.....	30-3
Abstract Datatypes.....	30-4
Wildcards.....	30-4
Modules.....	30-4
Side Effects.....	30-4
@ Operator (List Concatenation Operator).....	30-4
Conventions.....	30-4
Syntax descriptions.....	30-4
Format.....	30-4
Examples.....	30-5
Testing Examples.....	30-5
To test non-CPN inscription examples:.....	30-5
To test CPN inscription examples:.....	30-5

Chapter 31

Identifiers

Examples.....	31-1
Reserved Words.....	31-2
Identifier Duplication.....	31-2
Predeclared Identifiers.....	31-2
Comments.....	31-3

Chapter 32

Colorsets

Classifying Colorsets.....	32-1
Size.....	32-1
Complexity.....	32-2
Simple Colorsets.....	32-2
Unit.....	32-2
Declaration Syntax.....	32-2
Optional With Clause.....	32-2
Declaration Example.....	32-2
Value Representation Example.....	32-2

Chapter 32

Colorsets (cont'd)

Simple Colorsets (cont'd)

Boolean.....	32-2
Declaration Syntax.....	32-2
Optional With Clause.....	32-3
Declaration Example.....	32-3
Value Representation Example.....	32-3
Boolean Operations.....	32-3
Boolean Selectors.....	32-4
Integers.....	32-4
Declaration Syntax.....	32-4
Optional With Clause.....	32-4
Declaration and Use Example.....	32-4
Integer Operations.....	32-5
Real Numbers.....	32-5
Declaration Syntax.....	32-5
Optional With Clause.....	32-5
Declaration and Use Example.....	32-6
Real Operations.....	32-6
Strings.....	32-6
Declaration Syntax.....	32-7
Optional With Clause.....	32-7
Optional And Clause.....	32-7
Declaration and Use Example.....	32-7
String Operations.....	32-8
String Compare.....	32-8
Escape Sequences.....	32-8
Enumerated Values.....	32-8
Declaration Syntax.....	32-9
Declaration Example.....	32-9
Value Representation Example.....	32-9
Suggested Uses.....	32-9
Indexed Values.....	32-9
Declaration Syntax.....	32-10
Declaration Example.....	32-10
Value Representation Example.....	32-10
Suggested Uses.....	32-10
Compound CPN Colorsets.....	32-10
Tuples.....	32-10
Declaration Syntax.....	32-10
Declaration Example.....	32-11
Representation Examples.....	32-11
Suggested Uses.....	32-11
Records.....	32-11
Declaration Syntax.....	32-11
Selector Functions.....	32-12
Declaration Example.....	32-12

Chapter 32 Colorsets (cont'd)

Compound CPN Colorsets (cont'd)

Records (cont'd)	
Value Representation Example.....	32-12
Selector Example.....	32-13
Suggested Uses.....	32-13
Lists.....	32-13
Declaration Syntax.....	32-13
Optional With Clause.....	32-13
Declaration Example.....	32-13
Value Representation Example.....	32-13
List Operators and Functions.....	32-14
Suggested Uses.....	32-14
Subsets.....	32-15
Declaration Syntax.....	32-15
Function Form.....	32-15
By Clause.....	32-15
List Form.....	32-15
With Clause.....	32-15
Declaration and Use Example - List Form.....	32-15
Suggested Uses.....	32-16
Union.....	32-16
Declaration Syntax.....	32-16
Declaration Example.....	32-17
Suggested Uses.....	32-17
Alias CPN Colorset.....	32-17
Declaration Syntax.....	32-17
Declaration Example.....	32-17
Suggested Uses.....	32-18
Function.....	32-18
Declare Clause.....	32-18
Random Value.....	32-18
Syntax.....	32-18
Example.....	32-18
Less Than.....	32-19
Syntax.....	32-19
Example.....	32-19
String Representation of a Value.....	32-19
Syntax.....	32-20
Example.....	32-20
String Representation of a Multiset.....	32-20
Syntax.....	32-20
Example.....	32-20
Multiset, Size, First, and Last Constants.....	32-21
ms.....	32-21
size.....	32-21
first.....	32-21

Chapter 32 Colorsets (cont'd)

Declare Clause (cont'd)	
Multiset, Size, First, and Last Constants (cont'd)	
last.....	32-22
Examples.....	32-22
Ordinal Number and Value.....	32-22
Syntax.....	32-23
Examples.....	32-23
Index Number and Value.....	32-23
Syntax.....	32-23
Examples.....	32-24
Distance and Rotation for Values.....	32-24
Syntax.....	32-24
Examples.....	32-25
dist example.....	32-25
rot example.....	32-25
Multiplication of Multisets.....	32-25
Syntax.....	32-25
Example.....	32-26
Membership of a Subset CPN Colorset.....	32-26
Syntax.....	32-26
Example.....	32-26
Membership of Union Base CPN Colorset.....	32-27
Syntax.....	32-27
Example.....	32-27
Ordering of CPN Colorsets.....	32-28

Chapter 33 Values

Overview.....	33-1
Declaration.....	33-1
Syntax.....	33-1
Value Representation.....	33-1
Examples.....	33-2
Colorsets.....	33-3
Other types of specifiers.....	33-4
Multiset Expression.....	33-4
Function expression.....	33-4
Value Uses in CPN Inscriptions.....	33-4

Chapter 34 CPN Variables

Term Definitions.....	34-1
Variables.....	34-1
Binding.....	34-1
Scope.....	34-1
Extent.....	34-1
CPN Variables.....	34-1
Declaration syntax.....	34-2
Example of Declaration and Use.....	34-2

Chapter 35 Reference Variables

Reference Variables.....	35-1
Val-defined Reference Variables.....	35-1
Caution.....	35-1
Declaration Syntax.....	35-2
Reference Syntax.....	35-2
Value Syntax.....	35-2
CPN Reference Variables.....	35-3
Global Reference Variables.....	35-3
Declaration syntax.....	35-3
Page Reference Variables.....	35-3
Declaration syntax.....	35-3
Instance Reference Variables.....	35-3
Declaration syntax.....	35-3

Chapter 36 Expressions

Term Definitions.....	36-1
Expression Syntax.....	36-2
Syntactic Specifiers and Reserved Words.....	36-2
Variables.....	36-2
Values.....	36-2
Constructors.....	36-3
Operators.....	36-3
Expression Evaluation.....	36-4
Evaluation Order.....	36-4
Operator Precedence.....	36-4
Association.....	36-4
Binary operators.....	36-5
Function Type Operator.....	36-5
Operator Precedence and Association Table.....	36-5

Chapter 36

Expressions (cont'd)

Expression Uses.....	36-6
Patterns.....	36-6
Tuple Pattern Matching.....	36-7
List Pattern Matching.....	36-7
Constructors.....	36-7
Tuple Constructors.....	36-8
List Constructors.....	36-8
Expression Evaluation in CP Nets.....	36-8
Arcs.....	36-8
Guards.....	36-9
Code Segments.....	36-9
Further Discussion.....	36-9

Chapter 37

Multisets

Term Definitions.....	37-1
Multiset Variables.....	37-1
Syntax.....	37-1
Use.....	37-1
Multiset Creation.....	37-2
Syntax.....	37-2
Use.....	37-2
Multiset expressions.....	37-2
Syntax.....	37-2
Examples.....	37-2
Multiset Constants, Operations, and Functions.....	37-3
Constant Definitions.....	37-3
Empty Multiset.....	37-3
Full Multiset.....	37-3
Addition, Subtraction, and Scalar Multiplication, of Multisets.....	37-3
Addition (+ operator).....	37-3
Subtraction (- operator).....	37-4
Scalar Multiplication (* operator).....	37-4
Multiplication of Multisets.....	37-5
Comparing Multisets.....	37-5
Equality (== operator).....	37-5
Not Equal (<><> operator).....	37-5
Less Than or Equal (<= operator).....	37-6
Less Than (<< operator).....	37-6
Greater Than or Equal (>= operator).....	37-6
Greater Than (>> operator).....	37-6
Coefficient.....	37-7
Size.....	37-7
Random Value From a Multiset.....	37-7

Chapter 37

Multisets (cont'd)

Multiset Constants, Operations, and Functions (cont'd)	
Conversion Between Lists and Multisets.....	37-8
Filter Function.....	37-8
Linear Extension of Functions.....	37-8
Multiset Input and Output.....	37-9
Internal Representation of Multisets.....	37-9
Construct, Compress, and Sort.....	37-10
Multiset constructor (!! operator).....	37-10
Timed Multisets.....	37-10

Chapter 38

Functions

Declarations.....	38-1
Syntax.....	38-1
Datatype.....	38-1
Example.....	38-1
Local Declarations - Let.....	38-1
Syntax.....	38-2
Control Structures.....	38-2
if-then-else.....	38-2
Syntax.....	38-2
Boolean Selectors.....	38-2
Case.....	38-3
Syntax.....	38-3
Function Invocation.....	38-3

Chapter 39

Timing

Time Stamps.....	39-1
Declaration.....	39-1
Defaults for Time Stamps.....	39-1
Timed Arc Inscriptions.....	39-2
Input Arc Inscriptions.....	39-2
Output Arc Inscriptions.....	39-2
Combined Use.....	39-2
Time region.....	39-3
Time-Related Functions.....	39-4
Append time list to one-element multiset (@ operator)...39-4	
Time and step.....	39-4
Simulation options.....	39-4

Chapter 40 Inscriptions

Example Conventions.....	40-1
Example Diagram.....	40-2
Places.....	40-2
Place Name Region.....	40-2
Colorset Region.....	40-3
Initial Marking Region.....	40-3
Optional Time Delays.....	40-3
Syntax.....	40-3
Examples.....	40-4
Arcs.....	40-4
Arc Inscription Region.....	40-5
Patterns and Constructors.....	40-5
Side Effects.....	40-5
Time Stamps.....	40-5
Conditional arc inscriptions.....	40-6
Function application.....	40-6
Input Arc Inscription Examples - Declarations.....	40-7
Colorset Declarations.....	40-7
Variable declarations.....	40-8
Input Arc Inscription Examples.....	40-8
Constant.....	40-8
Multiset expression with enumerated values:.....	40-9
Variable.....	40-9
Multiset expression with variables:.....	40-9
Multiset expression with constrained variables:.....	40-9
Indexed values:.....	40-10
Tuple colorset, CPN variable arc inscription:.....	40-10
Tuple colorset, pattern arc inscription:.....	40-10
Record colorset, pattern arc inscription with variables:.....	40-11
List colorset, list constructor arc inscription:.....	40-11
List colorset, list pattern arc inscription:.....	40-11
Union colorset:.....	40-12
Output Arc Inscription Examples.....	40-13
Function application.....	40-13
Conditional output arc inscriptions.....	40-13
Example 1.....	40-13
Example 2.....	40-13
Example 3.....	40-14
Unbound variables on an output arc.....	40-14
Transition Regions.....	40-14
Transition Name Region.....	40-14
Time Region.....	40-15
Guard Region.....	40-15
Syntax.....	40-15
Use.....	40-15
Side Effects.....	40-16

Chapter 40 Inscriptions (cont'd)

Transition Regions (cont'd)	
Conditional Expressions.....	40-16
Example.....	40-16
Code Segment.....	40-16
CPN Variable Restrictions.....	40-17
Syntax.....	40-17
Input Clause.....	40-17
Output Clause.....	40-18
Code Action Clause.....	40-18
Examples.....	40-19
Example 1:.....	40-19
Example 2:.....	40-19

Chapter 41 The CPN ML Runtime Environment

Predeclared Environment.....	41-1
Predefined Constants, Functions, and Operators.....	41-1
Empty Multiset.....	41-1
Addition, Subtraction, and Scalar Multiplication, of Multisets.....	41-1
Addition (+ operator).....	41-1
Subtraction (- operator).....	41-2
Scalar Multiplication (* operator).....	41-2
Multiplication of Multisets.....	41-2
Comparing Multisets.....	41-2
Equality (== operator).....	41-2
Not Equal (<> operator).....	41-2
Less Than or Equal (<= operator).....	41-3
Less Than (< operator).....	41-3
Greater Than or Equal (>= operator).....	41-3
Greater Than (> operator).....	41-3
Multiset Creation.....	41-3
Coefficient.....	41-3
Size.....	41-3
Random Value From a Multiset.....	41-4
Conversion Between Lists and Multisets.....	41-4
Filter Function.....	41-4
Linear Extension of Functions.....	41-4
Multiset Input and Output.....	41-5
Append time list to one-element multiset (@ operator).....	41-5
Time and step.....	41-5
Simulation options.....	41-6
Simulation management.....	41-6
Boolean Selector.....	41-7

Chapter 41**The CPN ML Runtime Environment (cont'd)**

Predefined Constants, Functions, and Operators (cont'd)	
Boolean Selector.....	41-6
Addition, Subtraction, and Scalar Multiplication of Functions.....	41-7
Addition (+ operator).....	41-7
Subtraction (- operator).....	41-7
Scalar Multiplication (* operator).....	41-7
Identity, Zero, and Ignore.....	41-7
Identity.....	41-7
Zero.....	41-8
Ignore.....	41-8
Graphical Functions.....	41-8
ColorIO Functions.....	41-8
ML Library.....	41-9
Standard Constants, Operations, and Functions.....	41-9
General Functions.....	41-10
Boolean Functions.....	41-10
Integer Functions.....	41-10
Real Functions.....	41-10
String Functions.....	41-11
List Functions.....	41-11
Reference Functions.....	41-12
I/O Functions.....	41-12
Real Time Functions.....	41-12

Appendixes

Appendix A Keys and Shortcuts

Design/CPN Use of the Alt Key.....	A-1
Keystroke Shortcuts.....	A-1
Modifier keys.....	A-3
Option key.....	A-3
Shift key.....	A-3
Option + Shift keys.....	A-3
Caps Lock key.....	A-4
Space bar.....	A-4

Appendix B Troubleshooting

CPN Settings File Missing or Obsolete.....	B-1
Problem Description.....	B-2
Problem Solution.....	B-2
ML Configuration Unspecified or Incorrect.....	B-2
Identifying the Problem.....	B-3
Copying Diagram Default ML Configuration Options.....	B-4
Setting ML Configuration Options.....	B-4
ML Interpreter Cannot Be Started.....	B-6

INDEX

Special Characters

- !! (double exclamation point),**
internal representation of multisets with; 37-9
operator,
constructing multisets with; 37-10
- ^^ (concatenate list operator),**
CPN ML use as list concatenation operator; 30-4
reference description; 32-14
- ^ (concatenate string operator),**
reference description; 32-8
- ~ (negation operator),**
reference description; 32-6
- ” (double quote),**
type variable use; 30-4
- % (if-then selector),**
output arc inscription example; 40-13
reference description; 41-6
- ‘ (single quote),**
type variable use; 30-4
- () (parentheses),**
expression specifier use; 36-2
guards use of; 5-9
- * (scalar multiplication operator),**
function scalar multiplication; 41-7
multiplying multisets; 37-4
reference description; 32-6, 41-2
- + (addition operator),**
adding multisets; 37-3
function addition; 41-7
reference description; 32-6, 41-1
- , (comma),**
expression specifier use; 36-2
guards use as shorthand for boolean andalso operator;
5-9
- (subtraction operator),**
function subtraction; 41-7
reference description; 32-6, 41-2
subtracting multisets; 37-4
- / (division operator),**
reference description; 32-6
- / (if-then-else selector),**
reference description; 41-6
- < (less than numeric operator),**
in guards; 5-8
- < (less than string operator),**
reference description; 32-8
- << (multiset less than operator),**
reference description; 41-3
syntax and use; 37-6
- <= (multiset less than or equal operator),**
reference description; 41-3
syntax and use; 37-6
- <= (less than or equal numeric operator),**
in guards; 5-8
- <= (less than or equal string operator),**
reference description; 32-8
- <> (not equal numeric operator),**
in guards; 5-8
reference description; 32-8
- <> (not equal string operator),**
reference description; 32-8
- <><> (multiset not equal operator),**
reference description; 41-2
syntax and use; 37-5
- = (equal numeric operator),**
in guards; 5-8
- = (equal string operator),**
reference description; 32-8
- = equals,**
expression specifier use; 36-2
- == (multiset equality operator),**
reference description; 41-2
syntax and use; 37-5
- > (greater than string operator),**
reference description; 32-8
- > (greater than numeric operator),**
boolean operator used in guards; 5-8
- >= (greater than or equal string operator),**
reference description; 32-8
- >= (greater than or equal numeric operator),**
boolean operator used in guards; 5-8
- >> (multiset greater than operator),**
reference description; 41-3
syntax and use; 37-6
- >>= (multiset greater than or equal operator),**
reference description; 41-3
syntax and use; 37-6
- @ (time append operator),**
appending time list to a one-value multiset with; 39-4
redefinition in CPN ML; 30-4
reference description; 41-5
- @ + (delay expression),**
characteristics and use with time stamps; 12-7
- @ + keyword,**
time delay expression use of,
output arc inscriptions; 39-2
time regions; 39-3
- @ignore keyword,**
ignoring timing on arc inscriptions with; 39-2
- [] (square brackets),**
expression specifier use; 36-2

[] (**square brackets**) (**cont'd**),
guards use as distinguishing characters; 5-8
\
(**backslash operator**),
inserting control characters in strings with; 32-8
` (**backquote multiset creation operator**),
reference description; 2-8, 41-3
syntax and use; 37-2
{ } (**curly braces**),
expression specifier use; 36-2

A

abs (absolute value) operation,
reference description; 32-6
abstract datatypes,
not permitted in CPN ML; 30-4
accessing,
record components,
with selector functions; 32-12
statistical variables; 14-3
action,
keyword,
code segment code action clause identified by;
40-18
term definition; 13-2
section,
bar chart code segments; 15-13
line chart code segments; 15-31
activities,
See transitions;
adding,
See Also creating;
functions,
with addition operator (+); 41-7
multisets; 2-8, 37-3
addition operator reference description; 41-1
places to fusion sets,
with Fusion Place command (CPN menu); 21-22
to occurrence sets; 22-9
addition operator (+),
adding multisets; 37-3
reference description; 41-1
function addition; 41-7
reference description; 32-6
Adjust (Makeup menu),
reference description; 25-4
adjusting,
See Also aligning;
adjustment tool,
characteristics and illustration; 3-6

adjusting (cont'd),
graphic object to fit text,
with Fit to Text command (Makeup menu); 25-5
algorithms,
See Also bindings;
occurrence set execution; 8-14
simulator; 8-11
illustrating with SalesNet model execution with
conflict; 8-12
alias colorset,
See Also colorsets;
syntax and characteristics; 32-17
Align menu,
See Also menus;
Between command; 29-5
Bottom to Bottom command; 29-7
Bottom to Top command; 29-7
Center command; 29-2
Circular Spread command; 29-5
Horizontal command; 29-2
Horizontal Spread command; 29-4
Left to Left command; 29-6
Left to Right command; 29-6
Position command; 29-3
Projection command; 29-5
reference description; 29-1
Right to Left command; 29-6
Right to Right command; 29-6
Top to Bottom command; 29-7
Top to Top command; 29-7
Vertical command; 29-2
Vertical Spread command; 29-5
aligning,
groups,
with Align menu commands; 29-2
nodes,
along a diagonal, *See* Vertical Spread (Align menu)
and Horizontal Spread (Align menu);
between reference points, with Between (Align
menu); 29-5
centered, with Center (Align menu); 29-2
equidistantly around a circle, with Circular Spread
(Align menu); 29-5
horizontally, with Horizontal (Align menu); 29-2
relatively, with Position (Align menu); 29-3
vertically, with Vertical (Align menu); 29-2
vertically, with Vertical Spread (Align menu);
29-5
with Align menu commands; 29-1
Alt key,
Design/CPN use of; A-1

-
- Alt-DownArrow keys,**
 - navigating to an error with; 7-4
 - andalso (boolean AND),**
 - boolean operator used in guards; 5-8
 - andalso (boolean conjunction),**
 - expression specifier use; 36-2
 - animation,**
 - code segments used for; 40-17
 - appearance,**
 - concurrency,
 - controlling; 8-18
 - global fusion place; 10-5
 - hierarchical CP nets,
 - improving; 11-8
 - hierarchy page,
 - improving; 11-12
 - markings; 2-12
 - subpage,
 - improving; 11-11
 - superpage,
 - improving; 11-8
 - appending,**
 - time lists to multisets; 39-4
 - time to a one-element multiset,
 - with time append operator (@); 41-5
 - Arc (CPN menu),**
 - creating arcs with; 4-10
 - reference description; 21-4
 - arcs,**
 - See Also* connectors; CPN objects, classes of;
 - arc creation mode,
 - term definition; 4-10
 - arc creation tool,
 - term definition and illustration; 4-10
 - arc inscription creation mode,
 - term definition; 4-11
 - bidirectional,
 - term definition; 2-13
 - characteristics; 2-13
 - CP net component; 2-1
 - creating,
 - connectors from; 23-13
 - from connectors; 23-11
 - with Arc (CPN menu); 4-10
 - drawing,
 - single and multi-segment; 21-4
 - endpoint handle use; 21-5
 - expression evaluation in; 36-8
 - external,
 - term definition; 21-17
 - input,
 - inscriptions, term definition; 2-14
 - arcs (cont'd),**
 - input (cont'd),
 - inscriptions, characteristics as CP net component; 2-1
 - term definition; 2-13
 - input arc inscriptions,
 - conditional; 40-6
 - examples; 40-7
 - function application use and restrictions; 40-6
 - if/then/else restrictions; 38-2
 - overriding time stamps with; 40-5
 - timing considerations; 39-2
 - inscription region,
 - syntax and characteristics; 40-4
 - term definition; 2-14
 - inscriptions,
 - creating; 4-11
 - missing, reporting with the Syntax Options
 - command (Set menu); 24-19
 - term definition and characteristics; 2-14
 - merging,
 - specifying options for; 24-16
 - output,
 - binding CPN variables on; 13-1
 - delay expressions on; 12-8
 - evaluating inscriptions during transition firing; 5-14
 - inscriptions, characteristics as CP net component; 2-1
 - inscriptions, term definition; 2-14
 - term definition; 2-13
 - output arc inscriptions,
 - appending time delays to; 40-6
 - conditional; 40-6
 - delay expressions on; 12-8
 - examples; 40-13
 - function application use and restrictions; 40-6
 - if/then/else use in; 38-2
 - timing considerations; 39-2
 - rerouting; 11-8
 - restrictions; 21-4
 - shape attributes,
 - changing; 24-8
 - shapes possible to; 24-5
 - source/destination nodes,
 - effect of moving on; 21-4
 - term definition; 2-13
 - terminating creation mode; 21-5
 - arctan operation (arctangent),**
 - reference description; 32-6
 - arctangent operation (arctan),**
 - reference description; 32-6

arithmetic operators,

expressions; 36-3

arrow keys,

navigating by means of; 17-2

selecting hidden regions with; 25-9

arrowhead,

changing the fill pattern; 24-5

ASCII characters,

syntax and characteristics; 32-6

aspect ratio,

rectangle,

preserving; 3-10

assigning,

port places,

with Port Assignment command (CPN menu);
21-24

time stamps,

example; 12-10

association,

expression evaluation; 36-4

attributes,

bar charts,

redefining; 15-7

chart,

changing; 24-13

diagram default,

term definition; 4-2

display,

changing; 4-2

term definition; 4-2

graphic,

changing for selected objects; 24-3

changing system and/or diagram defaults; 24-3

graphical,

components; 17-15

selectability, changing; 17-17

selectability, characteristics; 17-17

visibility, changing; 17-17

visibility, characteristics; 17-16

mode,

changing; 24-12

changing, for supernodes; 24-13

characteristics and components; 17-16

code segments, characteristics; 17-16

current, generating an initial system state with,
using Initial State command (Sim menu); 22-11

included, characteristics; 17-16

interactive, characteristics; 17-16

observable, characteristics; 17-16

prime page, characteristics; 17-16

proposed occurrence sets, characteristics; 17-16

attributes (cont'd),

object,

term definition; 4-2, 17-15

page,

changing; 24-10

changing system and/or diagram defaults; 24-11

term definition and components; 17-15

regions,

changing; 24-8

components; 17-15

shape,

changing; 24-5

components; 17-15

system default,

term definition; 4-2

term definition and characteristics; 17-15

text,

changing for selected objects; 24-2

changing system and/or diagram defaults; 24-2

components; 17-15

automatic mode,

cautions,

don't use free variables with; 40-5

Automatic Run (Sim menu),

reference description; 22-10

automatic simulation,

code generation,

with the Simulation Code Options command (Set
menu); 24-21

autoscrolling,

characteristics; 3-4

Aux menu,

See Also menus;

Box command; 23-3

creating rectangles with; 3-6

commands reference (chapter); 23-1

Connector command; 23-2

creating connectors with; 3-14

Convert to Aux command; 23-13

Convert to CPN command; 23-11

creating auxiliary objects with; 3-3

Ellipse command; 23-4

creating ellipses with; 3-10

Label command; 23-9

creating rectangles with; 3-12

Line command; 23-8

Make Node command; 23-11

Make Region command; 23-10

creating regions with; 3-16

ML Evaluate command; 23-14

Polygon command; 23-5

Regular Polygon command; 23-7

Aux menu (cont'd),

- Rounded Box command; 23-4
- Start ML command; 23-13
- Stop ML command; 23-13
- Wedge command; 23-7

auxiliary,

- connectors,
 - term definition and characteristics; 17-7
- nodes,
 - boxes, creation and manipulation; 23-3
 - CPN node creation from; 23-11
 - creating from auxiliary regions; 23-11
 - creating from CPN nodes; 23-13
 - ellipses, creation and manipulation; 23-4
 - labels, creation and manipulation; 23-9
 - lines, creation and manipulation; 23-8
 - polygons, creation and manipulation; 23-5
 - region creation from; 23-10
 - rounded boxes, creation and manipulation; 23-4
 - rounded polygons, creation and manipulation; 23-7
 - term definition and characteristics; 17-7
 - wedges, creation and manipulation; 23-7
- objects,
 - See Also* graphical objects;
 - characteristics; 17-6, 17-10
 - CPN objects compared with; 4-1
 - term definition; 3-1, 17-6
- regions,
 - code segment creation from; 23-11
 - creating; 23-10
 - creating auxiliary nodes from; 23-11
 - creating CPN regions from; 23-11
 - creating from CPN regions; 23-13
 - effect of Paste command on; 20-5
 - guards creation from; 23-11
 - initial markings creation from; 23-11
 - log regions creation from; 23-11
 - place names creation from; 23-11
 - term definition and characteristics; 17-7
 - transition names creation from; 23-11

average,

- statistical variable values,
 - obtaining, with SV'avrg; 14-3

axis,

- rescaling and moving,
 - as overflow handling methods for line charts; 21-17

B**backquote (`) multiset creation operator,**

- creating multisets with,
 - reference description; 41-3
- syntax and use; 37-2

backslash operator (\),

- inserting control characters in strings with; 32-8

bar charts,

- bar names,
 - characteristics; 15-3
 - specifying; 21-11
- characteristics and use; 15-2
- chart node,
 - characteristics; 15-2
- code segment,
 - capabilities and uses; 15-11
 - characteristics and use; 15-8
 - conditional section use; 15-12
 - control expression use; 15-12
 - initialization section use; 15-11
 - using action section; 15-13
- copying; 15-13
- creating; 15-4
 - dialog description; 15-5
 - with Chart command (CPN menu); 21-10
- cutting; 15-13
- data analysis,
 - in Resource Use model; 16-10
- defining,
 - in Resource Use model; 16-9
- deleting; 15-14
- editing; 15-6
- grid lines,
 - specifying; 21-11
- in Resource Use model; 16-8
- initializing; 21-12
- labels,
 - supplying; 15-6
- legend,
 - characteristics; 15-3
 - labels, characteristics; 15-4
- name,
 - specifying; 21-10
- pasting; 15-13
- patterns,
 - characteristics; 15-3
- positive and negative values,
 - characteristics; 15-3
- redefining the attributes of; 15-7

bar charts (cont'd),

- regions,
 - reference description; 21-11
- saving a copy; 21-12
- title,
 - characteristics; 15-3
 - specifying; 21-11
- updating,
 - from transition code segments; 15-10
 - specifying values for; 15-9
 - update period specification; 21-12
 - using SC_upd_chart function in a transition code segment; 15-10
 - with chart code segment; 15-8
- upper and lower values,
 - characteristics; 15-3
- values,
 - specifying the type and range; 21-13

bars,

- bar chart,
 - specifying; 21-10
- history charts,
 - specifying; 21-10

behavior,

- See Also* modeling;
- timed,
 - increasing the realism of; 12-15

Between (Align menu),

- reference description; 29-5

bidirectional arcs,

- term definition; 2-13

Bind (Sim menu),

- reference description; 22-2
- requirements; 22-2

bindings,

- See Also* algorithms; occurrence sets; tuples,
 - constructors; tuples, patterns;
- adding to occurrence sets,
 - with and without being executed; 22-8
- bind dialog,
 - clearing the editing area; 22-6
 - control button meanings; 22-5
 - editing area; 22-3
 - format description; 22-2
 - included bindings list box; 22-4
 - potential bindings list box; 22-4
- calculating,
 - all; 22-6
 - single; 22-6
- conflict and; 8-6
- CPN variables,
 - output arcs; 13-1

bindings (cont'd),

- creating and editing for CPN variables,
 - with Bind command (Sim menu); 22-3
- different,
 - explanation of the setting in the Occurrence Set Options dialog; 8-21
- elements,
 - executing; 8-13
 - term definition; 8-11, 8-18
- enabling,
 - identical; 8-3
 - multiple; 8-2
- identical,
 - explanation of the setting in the Occurrence Set Options dialog; 8-21
- included,
 - list box format; 22-4
 - term definition and characteristics of included bindings list box; 22-5
- input arc inscription variables; 5-7
- occurrence set contribution,
 - specifying, with Occurrence Set Options command (Set menu); 24-30
- partial,
 - term definition and handling in Bind dialog editing area; 22-4
- potential,
 - list box format; 22-4
 - term definition and characteristics of included bindings list box; 22-5
- recording information about,
 - specification with General Simulation Options (Set menu); 6-10
- regions,
 - characteristics; 17-11
- reporting information about,
 - with the Simulation Code Options command (Set menu); 24-26
- resetting; 22-8
- term definition; 34-1
- transferring from one area of Bind dialog to another; 22-6

Block by Character Count component (Load Text command),

- reference description; 19-10

Block by Delimiters component (Load Text command),

- reference description; 19-10

blocking,

- blocking options (Save Subdiagram command),
 - reference description; 19-9

blocking (cont'd),

- character count,
 - specifying brackets and character counts used by,
 - with Text Options command (Set menu); 24-17
- delimiter,
 - specifying brackets and character counts used by,
 - with Text Options command (Set menu); 24-17

Blowup (Page menu),

- See Also* Cleanup (Page menu); Reduce (Page menu);
- reference description; 26-2

boolean,

- See Also* colorsets; expressions; guards;
- AND (andalso),
 - boolean operator used in guards; 5-8
- colorsets,
 - ordering of; 32-28
 - syntax and characteristics; 32-2
- conditional (if/then/else),
 - expression specifier use; 36-2
- conjunction (andalso),
 - expression specifier use; 36-2
- disjunction (orelse),
 - expression specifier use; 36-2
- expressions,
 - bar chart code segment use of; 15-12
 - guard use; 40-15
 - line chart code segment use of; 15-29
 - operators; 36-3
- functions,
 - standard ML, list of supported; 41-10
- NOT (not),
 - boolean operator used in guards; 5-8
- operators,
 - syntax and characteristics; 32-3
 - used in guards; 5-8
- OR (orelse),
 - boolean operator used in guards; 5-8
- selectors; 41-6
 - syntax and characteristics; 38-2
- tests,
 - constraining token values with; 5-8
- value identifiers,
 - declaring; 33-2

borders,

- node,
 - changing line thickness; 24-4
 - changing line type; 24-5
- page,
 - changing the size; 24-11
 - rendering visible or invisible; 24-11
 - visibility, characteristics; 17-15

borders (cont'd),

- region,
 - deletion requirements; 20-6

Bottom to Bottom (Align menu),

- reference description; 29-7

Bottom to Top (Align menu),

- reference description; 29-7

bound,

- term definition; 5-7

Box (Aux menu),

- creating rectangles with; 3-6
- reference description; 23-3

boxes,

- See Also* rectangles;
- creating, modifying, and moving; 23-3
- shape attributes,
 - changing; 24-6
- term definition and characteristics; 17-5

brackets,

- selecting text to a specified,
 - with Select to Bracket (Text menu); 28-5
- square,
 - guards use as distinguishing characters; 5-8
 - used by Load Text and Select to Brackets commands,
 - specifying, with Text Options command (Set menu); 24-17

breakpoints,

- See Also* simulation;
- beginning of substep,
 - characteristics; 6-16
 - concurrent execution; 8-4
 - concurrent execution of SalesNet model; 8-8
 - SalesNet's appearance; 8-14
- continuing execution after,
 - with Continue command (Sim menu); 6-16
- end of substep,
 - characteristics; 6-16
 - concurrent execution; 8-4
 - concurrent execution of SalesNet model; 8-8
 - SalesNet's appearance; 8-15
- restarting simulation after,
 - with Continue command (Sim menu); 22-10
- setting,
 - all, with Option key plus Stop command (Sim menu); 22-11
 - with Interactive Simulation Options command (Set menu); 6-15, 24-27

Bring Forward (Makeup menu),

- reference description; 25-9

C

- C.1 syntax error,**
 - meaning; 17-22
- C.2 syntax error,**
 - meaning; 17-22
- C.3 syntax error,**
 - meaning; 17-22
- C.4 syntax error,**
 - meaning; 17-22
- C.5 syntax error,**
 - meaning; 17-22
- C.6 syntax error,**
 - meaning; 17-23
- C.7 syntax error,**
 - meaning; 17-23
- C.8 syntax error,**
 - meaning; 17-23
- C.9 syntax error,**
 - meaning; 17-23
- C.10 syntax error,**
 - meaning; 17-23
- C.11 syntax error,**
 - meaning; 17-23
- C.12 syntax error,**
 - meaning; 17-24
- C.13 syntax error,**
 - meaning; 17-24
- C.14 syntax error,**
 - meaning; 17-24
- C.15 syntax error,**
 - meaning; 17-24
- C.16 syntax error,**
 - meaning; 17-24
- C.17 syntax error,**
 - meaning; 17-24
- C.18 syntax error,**
 - meaning; 17-25
- C.19 syntax error,**
 - meaning; 17-25
- C.20 syntax error,**
 - meaning; 17-25
- canonical expressions,**
 - term definition; 36-1
- Caps Lock key behavior,**
 - preserving rectangle aspect ratio during size change;
3-10
 - X-Windows; 3-5
- cartesian,**
 - coordinates,
 - aligning nodes relative to; 29-3
 - product,
 - See tuples;
- case,**
 - case/of (case),
 - expression specifier use; 36-2
 - control structure,
 - syntax and characteristics; 38-3
 - expression,
 - output arc inscription example; 40-14
- case sensitivity,**
 - required,
 - with Select to Bracket (Text menu); 28-5
 - specifying,
 - with Find (Text menu); 28-3
- causality,**
 - See modeling; representation;
- Center (Align menu),**
 - reference description; 29-2
- cf (coefficient) function,**
 - syntax and characteristics; 37-7
- cf function,**
 - reference description; 41-3
- Change Marking (Sim menu),**
 - reference description; 22-11
- Change Shape (Makeup menu),**
 - reference description; 25-5
- changing,**
 - chart attributes; 24-13
 - connectors,
 - overlay options; 24-15
 - diagram and/or defaults for shape attributes; 24-5
 - display attributes; 4-2
 - fill pattern; 24-5
 - fusion place type,
 - with Fusion Place command (CPN menu); 21-21
 - fusion set type; 21-23
 - graphic attributes,
 - for selected objects; 24-3
 - hierarchy page,
 - layout and redrawing options; 24-14
 - instance fusion place type,
 - with Fusion Place command (CPN menu); 21-21
 - interaction system options; 24-15
 - mode attributes; 24-12
 - for supernodes; 24-13
 - node border,
 - line thickness; 24-4
 - line type; 24-5

changing (cont'd),

- nodes,
 - overlay options; 24-15
 - size and position; 21-8
- pages,
 - attributes; 24-10
 - borders, size; 24-11
 - overlay options; 24-15
 - page fusion place type; 21-21
 - palette, overlay options; 24-15
- places,
 - markings; 22-11
 - size and position; 21-2
- regions,
 - attributes; 24-8
 - color; 24-9
 - position; 24-9
 - size; 24-9
 - with Change Shape command (Makeup menu); 25-6
- scrolling display options; 24-15
- shape attributes; 24-5
 - arcs; 24-8
 - boxes; 24-6
 - connectors; 24-8
 - ellipses; 24-6
 - lines; 24-8
 - pictures; 24-6
 - polygon; 24-7
 - rounded boxes; 24-7
 - wedges; 24-7
 - with Change Shape command (Makeup menu); 25-5
- substitution transition modes; 24-12
- system and/or diagram defaults,
 - for graphic attributes; 24-3
 - for page attributes; 24-11
 - for text attributes; 24-2
- text attributes,
 - for selected objects; 24-2
- transitions,
 - size and position; 21-3

characters,

- character count blocking,
 - defining brackets and character counts used by; 24-17
 - used by Load Text and Select to Brackets commands, specifying; 24-17
- corresponding to an octal code,
 - string operation; 32-8
- syntax and characteristics; 32-6

Chart (CPN menu),

- reference description; 21-8

Chart Attributes (Set menu),

- reference description; 24-13

charts,

- attributes,
 - changing; 24-13
- bar,
 - bar name characteristics; 15-3
 - bar characteristics; 15-3
 - characteristics and use; 15-2
 - chart node characteristics; 15-2
 - code segment, capabilities and uses; 15-8, 15-11
 - code segment, conditional section use; 15-12
 - code segment, control expression use; 15-12
 - code segment, initialization section use; 15-11
 - code segment, using action section; 15-13
 - copying; 15-13
 - creating; 15-4
 - creating, dialog description; 15-5
 - creating, with Chart command (CPN menu); 21-10
 - cutting; 15-13
 - data analysis, in Resource Use model; 16-10
 - defining, in Resource Use model; 16-9
 - deleting; 15-14
 - editing; 15-6
 - in Resource Use model; 16-8
 - initializing; 21-12
 - legend characteristics; 15-3
 - legend label characteristics; 15-4
 - name, specifying; 21-10
 - nomenclature and illustration; 15-2
 - pasting; 15-13
 - pattern characteristics; 15-3
 - positive and negative values characteristics; 15-3
 - regions, reference description; 21-11
 - saving a copy; 21-12
 - specifying bars; 21-10
 - specifying grid lines; 21-11
 - specifying the type and range of values; 21-13
 - specifying title; 21-11
 - update period specification; 21-12
 - updating, from transition code segments; 15-10
 - updating, specifying values for; 15-9
 - updating, using SC_upd_chart function in a transition code segment; 15-10
 - updating, with chart code segment; 15-8
 - upper and lower values characteristics; 15-3
- creating; 15-1
 - with Chart command (CPN menu); 21-8
- displaying statistical data with,
 - in Resource Use model; 16-7

charts (cont'd),

- facilities,
 - (chapter); 15-1
- history,
 - characteristics and use; 15-14
 - copying; 15-17
 - creating; 15-14
 - creating with Chart command (CPN menu); 21-10
 - cutting; 15-17
 - data analysis, in Resource Use model; 16-11
 - defining, in Resource Use model; 16-11
 - deleting; 15-17
 - editing; 15-16
 - in Resource Use model; 16-10
 - initializing; 21-12
 - pasting; 15-17
 - redefining; 15-16
 - regions, reference description; 21-11
 - saving a copy; 21-12
 - specifying bars; 21-10
 - specifying grid lines; 21-11
 - specifying the type and range of values; 21-13
 - specifying title; 21-11
 - specifying values; 15-16
 - update period specification; 21-12
 - updating; 15-16
- line,
 - characteristics and use; 15-17
 - chart node characteristics; 15-18
 - code segments, characteristics and use; 15-24, 15-28
 - code segments, conditional section use; 15-29
 - code segments, control expression use; 15-29
 - code segments, initialization section use; 15-28
 - code segments, using action section; 15-31
 - conditional drawing of line segments; 15-30
 - copying; 15-31
 - creating; 15-19
 - creating with Chart command (CPN menu); 21-13
 - creating, dialog description; 15-20
 - cutting; 15-31
 - data analysis, in Resource Use model; 16-13
 - defining, in Resource Use model; 16-13
 - deleting; 15-31
 - dialog box; 15-20
 - grid lines, specifying the characteristics of; 21-15
 - in Resource Use model; 16-12
 - initializing; 21-15
 - legend characteristics; 15-19
 - line characteristics; 15-19
 - line characteristics, specifying; 21-14
 - nomenclature; 15-17

charts (cont'd),

- line (cont'd),
 - pasting; 15-31
 - pattern characteristics; 15-19
 - redefining; 15-23
 - saving a copy; 21-15
 - specifying methods for handling overflow; 21-17
 - specifying name; 21-14
 - specifying origin; 21-16
 - specifying title; 21-15
 - specifying values for; 15-25
 - step updating period; 21-16
 - time updating period; 21-16
 - title characteristics; 15-18
 - update period specification; 21-16
 - updating; 15-24
 - updating, from transition code segments; 15-27
 - updating, using LC_upd_chart function in a transition code segment; 15-27
 - updating, with chart code segment; 15-24
 - values, specifying the type and range; 21-16
- nodes,
 - See Also* CPN objects, classes of;
 - characteristics and region; 17-9
- overview of facilities; 15-1
- parts,
 - term definition; 15-2
- statistical variables and,
 - using (chapter); 16-1
- updating; 15-1
 - with Update Chart command (Sim menu); 22-13

Child Object (Makeup menu),

- navigating with; 17-4
- reference description; 25-10
- selecting a region with; 11-9

child objects,

- selecting,
 - with Child Object (Makeup menu); 25-10

children,

- term definition and characteristics; 3-16, 17-5

choice,

- See Also* modeling;
- concurrency and,
 - (chapter); 8-1
- term definition; 8-1

chr (character-octal code) string operation,

- reference description; 32-8

circle,

- spacing nodes equidistantly around,
 - with Circular Spread (Align menu); 29-5

Circular Spread (Align menu),

- reference description; 29-5

- classifying,**
 - colorsets,
 - complexity; 32-2
 - size; 32-1
- cleaning up,**
 - pages,
 - with Cleanup (Page menu); 26-3
- Cleanup (Page menu),**
 - See Also* Blowup (Page menu);
 - reference description; 26-3
- Clear (Edit menu),**
 - reference description; 20-6
- Clear Report (Sim menu),**
 - reference description; 22-13
- clearing,**
 - occurrence set; 22-9
 - simulation reports,
 - with Save Report command (Sim menu); 22-13
 - statistical variables; 14-5
- clock,**
 - simulated,
 - mechanism characteristics; 12-3
 - term definition; 12-2
- cloning,**
 - regions; 4-11
- Close (File menu),**
 - reference description; 19-3
- Close Page (Page menu),**
 - reference description; 26-2
- closing,**
 - diagram; 1-7
 - pages,
 - with Close Page (Page menu); 26-2
- clr' (value) function,**
 - syntax and characteristics; 32-23
- code,**
 - generating,
 - for different types of execution, Simulation Code Options (Set menu); 6-5
- code region,**
 - See* code segments;
- code segments,**
 - bar charts,
 - capabilities and uses; 15-11, 15-8
 - conditional section use; 15-12
 - control expression use; 15-12
 - initialization section use; 15-11
 - characteristics and syntax,
 - (chapter); 13-1
 - code action clause,
 - term definition and characteristics; 13-2
- code segments (cont'd),**
 - controlling the execution of,
 - for pages; 24-13
 - for substitution transitions; 24-12
 - creating; 21-6
 - from auxiliary regions; 23-11
 - deletion requirements; 20-6
 - expression evaluation in; 36-9
 - functions permitted with,
 - with_code; 41-6
 - with_time; 41-6
 - I/O,
 - multisets; 37-9
 - input pattern,
 - term definition and characteristics; 13-2
 - line charts,
 - capabilities and uses; 15-24, 15-28
 - conditional section use; 15-29
 - control expression use; 15-29
 - initialization section use; 15-28
 - using action section; 15-31
 - mode attribute,
 - characteristics; 17-16
 - output pattern,
 - term definition and characteristics; 13-2
 - reading,
 - multisets; 37-9
 - specifying,
 - code generation for; 24-20
 - compilation of; 24-22
 - simulation; 24-24, 41-6
 - term definition; 13-1
 - transitions,
 - code action clause syntax and use; 40-18
 - examples; 40-19
 - input clause syntax and use; 40-17
 - output clause syntax and use; 40-18
 - region reference description; 40-16
 - updating bar charts from; 15-10
 - updating line charts from; 15-27
 - writing,
 - multisets; 37-9
- coefficient,**
 - coefficient (cf) function,
 - syntax and characteristics; 37-7
 - multiset value,
 - obtaining, with cf function; 41-3
- col' (value) function,**
 - syntax and characteristics; 32-22
- color,**
 - regions,
 - changing; 24-9

colored Petri nets,

See Also CP nets; Design/CPN;

term definition; 1-1

colorsets,

See Also places;

(chapter); 32-1

alias,

 syntax and characteristics; 32-17

boolean,

 ordering of; 32-28

 syntax and characteristics; 32-2

characteristics; 2-3

classifying,

 complexity; 32-2

 size; 32-1

composite,

 term definition; 2-5

compound,

See colorsets - alias, lists, records, subsets,

 tuples, union;

CPN ML extension; 30-2

creating from auxiliary regions; 23-11

declaring value identifiers with; 33-3

duplicate,

 term definition and characteristics; 2-5

enumerated values,

 cyclic manipulation of values; 32-24

 obtaining position of a particular value; 32-22

 obtaining value at particular position; 32-22

 ordering of; 32-28

 syntax and characteristics; 32-8

 term definition; 2-4

first value,

 obtaining for small colorsets; 32-21

indexed values,

 cyclic manipulation of values; 32-24

 obtaining identifier-value for index number; 32-23

 obtaining index number for identifier-value; 32-23

 obtaining position of a particular value; 32-22

 obtaining value at particular position; 32-22

 ordering of; 32-28

 syntax and characteristics; 32-9

integers,

 ordering of; 32-28

 syntax and characteristics; 32-4

 term definition; 2-4

 testing for membership in; 32-26

large,

 characteristics; 32-1

last value,

 obtaining for small colorsets; 32-22

colorsets (cont'd),

lists,

 ordering of; 32-28

 syntax and characteristics; 32-13

missing,

 detecting and handling; 7-2

ordering of; 32-28

reading,

 with input_col function; 41-8

real numbers,

 ordering of; 32-28

 syntax and characteristics; 32-5

 testing for membership in; 32-26

records,

 constructing multiset for; 32-25

 ordering of; 32-28

 syntax and characteristics; 32-11

region,

 characteristics; 40-3

 creating with CPN Region command; 21-5

 creation mode, term definition; 4-9

simple,

See colorsets - boolean, enumerated values,
 indexed values, integers, real numbers, strings,
 unit;

size,

 classification; 32-1

 obtaining for small colorsets; 32-21

small,

 characteristics; 32-1

 constants that can be declared for; 32-21

 creating multisets for a; 32-21

 obtaining first value; 32-21

 obtaining last value; 32-22

 obtaining size; 32-21

specifying,

 with CPN Region (CPN menu); 4-9

strings,

 ordering of; 32-28

 representation operation; 32-8, 32-19

 syntax and characteristics; 32-6

 term definition; 2-4

 testing for membership in; 32-26

subsets,

 ordering of; 32-28

 syntax and characteristics; 32-15

 testing for membership in; 32-26

term definition; 2-3

timed,

 declaring; 12-6, 39-1

 defaults; 39-1

 removing timing from; 39-1

- colorsets (cont'd),**
 - timed (cont'd),
 - term definition; 12-2
 - tuples,
 - constructing multiset for; 32-25
 - ordering of; 32-28
 - syntax and characteristics; 32-10
 - term definition; 2-5
 - union,
 - syntax and characteristics; 32-16
 - testing for membership in; 32-27
 - unit,
 - ordering of; 32-28
 - syntax and characteristics; 32-2
 - writing,
 - with output_col function; 41-8
- comma (,),**
 - guards use as shorthand for boolean andalso operator; 5-9
- commands reference,**
 - Align menu (chapter); 29-1
 - Aux menu (chapter); 23-1
 - CPN menu (chapter); 21-1
 - Edit menu (chapter); 20-1
 - File menu (chapter); 19-1
 - Group menu (chapter); 27-1
 - Makeup menu (chapter); 25-1
 - Page menu (chapter); 26-1
 - Set menu (chapter); 24-1
 - Sim menu (chapter); 22-1
 - Text menu (chapter); 28-1
- comments,**
 - CPN ML; 31-3
 - in models,
 - auxiliary objects use for; 3-1
- communication,**
 - code segments used for; 40-17
- comparing,**
 - multisets; 37-5
 - equality operator reference description; 41-2
 - greater than operator reference description; 41-3
 - greater than or equal operator reference description; 41-3
 - less than operator reference description; 41-3
 - less than or equal operator reference description; 41-3
 - non equal operator reference description; 41-2
- comparison,**
 - operators,
 - strings; 32-8
 - used in guards; 5-8
- compiling,**
 - See* switching;
- complexity,**
 - colorsets,
 - as classification dimension; 32-2
- composite colorset,**
 - See Also* colorsets;
 - term definition; 2-5
- compound,**
 - See Also* colorsets - alias, lists, records, subsets, tuples, union;
 - colorsets,
 - characteristics; 32-2
 - expressions,
 - term definition; 36-1
- compress function,**
 - syntax and characteristics; 37-10
- concatenate (^) list operator,**
 - reference description; 32-14
- concatenation string operation (^),**
 - reference description; 32-8
- concurrency,**
 - See Also* time;
 - characteristics and implications for CP nets; 8-19
 - choice and,
 - (chapter); 8-1
 - concurrent activities,
 - term definition; 8-1
 - concurrent system,
 - term definition; 8-1
 - conflict as limiting factor in; 8-6
 - controlling the appearance of; 8-18
 - CP net execution; 8-4
 - firing multiple concurrent transitions; 8-3
 - problems with; 8-1
 - representing; 8-2
 - term definition; 8-1
- concurrent,**
 - term definition; 8-19
- cond keyword,**
 - bar chart code segment use of; 15-12
 - line chart code segment use of; 15-29
- conditional,**
 - arc inscriptions,
 - syntax and characteristics; 40-6
 - control structures,
 - if/then/else syntax and characteristics; 38-2
 - expressions,
 - guard use; 40-16
 - output arc inscription example; 40-13
 - section,
 - term definition; 15-12, 15-29

conflict,

See Also concurrency;
bindings and; 8-6
executing SalesNet model with; 8-12
representing; 8-5
term definition; 8-1

Connector (Aux menu),

creating connectors with; 3-14
reference description; 23-2

connectors,

See Also arcs;
arcs (formal CPN connector),
 See arcs;
auxiliary,
 term definition and characteristics; 17-7
creating; 3-14
 a group of; 27-2
 arcs from; 23-11
 from arcs; 23-13
deleting,
 dangling; 3-15
dragging endpoints of,
 with Drag command (Makeup menu); 25-3
drawing,
 single-segment and multi-segment; 23-2
duplicating,
 with Duplicate Node command (Makeup menu);
 25-6
editing; 3-15
effect of,
 Copy command on; 20-4
 Cut command on; 20-2
 hiding regions on; 25-9
 merging nodes on; 25-8
 moving nodes between pages on; 25-7
 Paste command on; 20-5
endpoint,
 handle, using to reattach a connector; 23-2
 regions, creating; 17-14
 regions, term definition and characteristics; 17-13
layering order for; 17-1
merging,
 specifying options for; 24-16
moving to a different attach point; 23-2
overlay options,
 changing; 24-15
page,
 deletion requirements; 20-6
routing; 3-14
 automatic; 3-15
shape attributes; 24-5
 changing; 24-8

connectors (cont'd),

source and destination nodes,
 effect of moving on; 23-2
substitution,
 characteristics; 17-10
system,
 characteristics; 17-10
term definition; 3-2, 3-13
 and characteristics; 17-5
terminating creation mode; 23-3

constants,

See Also CPN variables;
first value in colorset,
 syntax and characteristics; 32-21
input arc inscription example; 40-8
last value in colorset,
 syntax and characteristics; 32-22
multisets,
 syntax and characteristics; 32-21, 37-3
number of values in colorset,
 syntax and characteristics; 32-21
predefined,
 reference descriptions; 41-1
size of colorset,
 syntax and characteristics; 32-21
small colorsets; 32-21
specifying exact token values with; 5-4
term definition; 41-1

constraining,

See Also guards;
token values; 5-8
 multiple; 5-11
 with more complex guards; 5-10
 with simple guards; 5-9

constraints,

partial,
 creating with guards; 5-12
term definition; 5-8

constructing,

See Also creating;
lists,
 operators; 32-14
multisets; 37-10
occurrence sets; 8-18

constructors,

characteristics and use; 36-7
lists,
 input arc inscription example; 40-11
tuple,
 term definition and characteristics; 2-6

- Continue (Sim menu),**
 - continuing execution after a breakpoint with; 6-16, 8-15
 - reference description; 22-10
- continuing,**
 - CP net execution,
 - with Continue command (Sim menu); 6-16
- control characters,**
 - inserting in strings; 32-8
- control expression,**
 - term definition; 15-12, 15-29
- control structures,**
 - boolean selectors,
 - syntax and characteristics; 38-2
 - case,
 - syntax and characteristics; 38-3
 - if/then/else,
 - syntax and characteristics; 38-2
- controlling,**
 - code segment execution,
 - for pages; 24-13
 - for substitution transitions; 24-12
 - page presence,
 - during interactive runs; 24-13
 - during simulation; 24-13
 - in occurrence sets; 24-13
 - substitution transition presence,
 - during interactive runs; 24-12
 - during simulation; 24-12
 - in occurrence sets; 24-12
- conventions,**
 - CPN ML notation; 30-4
 - inscription examples; 40-1
- Convert to Aux (Aux menu),**
 - reference description; 23-13
- Convert to CPN (Aux menu),**
 - reference description; 23-11
- converting,**
 - lists,
 - to multisets, with list_to_ms function; 41-4
 - multisets,
 - to lists, with ms_to_list function; 41-4
- Copy (Edit menu),**
 - reference description; 20-3
 - restrictions; 20-4
- Copy Defaults (Set menu),**
 - copying diagram defaults with; 4-3
 - reference description; 24-33
- copying,**
 - bar charts; 15-13
 - diagram defaults,
 - with Copy Defaults (Set menu); 4-3
- copying (cont'd),**
 - history charts; 15-17
 - line charts; 15-31
 - regions; 4-11
- cos operation (cosine),**
 - reference description; 32-6
- cosine operation (cos),**
 - reference description; 32-6
- count,**
 - statistical variable values,
 - obtaining, with SV'count; 14-3
- CP nets,**
 - changing,
 - in the simulator; 8-9
 - components (chapter); 2-1
 - creating,
 - requirements for; 4-4
 - with Design/CPN editor (chapter); 4-1
 - example description; 2-2
 - executing,
 - (chapter); 6-1
 - example; 5-13
 - observation of; 6-15
 - overview; 2-15, 5-1
 - hierarchical,
 - developing on a subpage; 11-14
 - incremental development,
 - prime page role; 6-3
 - modularity,
 - prime page role; 6-3
 - setting up for execution; 6-5
 - structure,
 - term definition; 1-1
 - syntax errors,
 - handling, (chapter); 7-1
 - term definition; 1-1
 - and characteristics; 17-5
- CPN,**
 - term definition; 1-1
- CPN data objects,**
 - See* tokens;
- CPN menu,**
 - See Also* menus;
 - Arc command; 21-4
 - creating arcs with; 4-10
 - Chart command; 21-8
 - commands reference (chapter); 21-1
 - CPN Region command; 21-5
 - assigning time stamps with; 12-10
 - fixing a syntax error with; 7-4
 - naming transitions with; 4-6
 - Declaration Node command; 21-7

CPN menu (cont'd),

- Declaration Node command (cont'd),
 - creating a global declaration node with; 4-12
- Fusion Place command; 21-20
 - adding places to a fusion set with; 10-6
 - creating a fusion set with; 10-4
 - deleting fusion sets with; 10-8
 - removing places from a fusion set with; 10-7
- Move to Subpage command; 21-17
 - creating a subpage with; 11-5
 - top-down hierarchical CP net development with; 11-14
- Place command; 21-2
- Port Assignment command; 21-24
 - manually assigning ports to sockets with; 11-27
- Port Place command; 21-23
- Remove Sim Regions command; 21-28
 - removing simulation regions with; 6-19
- Replace by Subpage command; 21-19
- Substitution Transition command; 21-19
- Syntax Check command; 21-26
 - performing a syntax check with; 6-2
- Syntax Options command,
 - selecting optional syntax restrictions with; 6-1
- Transition command; 21-3

CPN ML language,

- comments; 31-3
- configuration options,
 - preserving; 4-3
 - specifying, with ML Configuration Options command (Set menu); 24-32
- errors,
 - detecting; 7-6
- evaluating current text object as; 23-14
- extensions to Standard ML; 30-2
- features; 30-1
- file,
 - characteristics and components; 1-3
- file name,
 - specifying; 21-27
- identifiers (chapter); 31-1
- interpreter,
 - starting; 23-13
 - stopping; 23-13
- introduction (chapter); 30-1
- library,
 - characteristics and location; 41-9
- ML Configuration Options (Set menu),
 - reference description; 24-32
 - saving ML configuration options with; 4-3
- ML Evaluate (Aux menu),
 - reference description; 23-14

CPN ML language (cont'd),

- restrictions and modifications of Standard ML; 30-3
- role in CP nets; 2-1
- runtime environment (chapter); 41-1
- specifying image location,
 - with ML Configuration Options command (Set menu); 24-32
- standard function,
 - list of supported; 41-9

CPN model,

- term definition; 1-1
- and characteristics; 17-5

CPN nodes,

- creating,
 - auxiliary nodes from; 23-13
 - from auxiliary nodes; 23-11
- term definition; 4-1

CPN objects,

- auxiliary objects compared with; 4-1
- classes of,
 - See* arcs; chart nodes; declaration nodes; places; regions; transitions;
- graphical,
 - term definition; 4-1
- term definition; 3-1, 4-1
- and characteristics; 17-6

CPN reference variables,

- global,
 - syntax and characteristics; 35-3
- instance,
 - syntax and characteristics; 35-3
- page,
 - syntax and characteristics; 35-3

CPN Region (CPN menu),

- assigning time stamps with; 12-10
- creating guards with; 4-7
- fixing a syntax error with; 7-4
- naming transitions with; 4-6
- reference description; 21-5

CPN regions,

- creating; 21-5
 - auxiliary regions from; 23-13
 - from auxiliary regions; 23-11
 - general technique; 4-4
 - multiple; 21-6
 - place regions; 21-5
- effect of Paste command on; 20-5
- restriction on use of CPN Region command; 21-5
- term definition; 4-1
- terminating creation mode; 21-7
- text,
 - specifying options for merging; 24-17

CPN Settings,

term definition and characteristics; 17-18

CPN variables,

(chapter); 34-1

as CPN ML extension; 30-2

binding values to,

with Bind command (Sim menu); 22-3

calculated,

identification of in included bindings list box;
22-4

identification of in potential bindings list box;
22-5

code segment,

restrictions; 40-17

use of; 13-1

declaration syntax; 34-2

detecting and handling undeclared; 7-4

expression use; 36-2

global reference,

syntax and characteristics; 35-3

input arc inscription example; 40-9

output arcs,

binding of; 13-1

rebinding during transition firing; 5-13

records with,

input arc inscription example; 40-11

specifying token values with; 5-7

term definition and characteristics; 2-9, 34-1

tuples with,

input arc inscription example; 40-10

unbound ,

output arc inscription example; 40-14

creating,

See Also constructing; drawing; editing;

arc inscriptions,

with CPN Region (CPN menu); 4-11

arcs,

from connectors; 23-11

with Arc (CPN menu); 4-10

auxiliary nodes,

from auxiliary regions; 23-11

from CPN nodes; 23-13

auxiliary regions; 23-10

from CPN regions; 23-13

bar charts; 15-4

dialog description; 15-5

bindings for CPN variables,

with Bind command (Sim menu); 22-3

charts; 15-1

with Chart command (CPN menu); 21-8

code segments,

for transitions; 21-6

creating (cont'd),

code segments (cont'd),

from auxiliary regions; 23-11

colorset region (place node),

with CPN Region command; 21-5

colorsets,

from auxiliary regions; 23-11

connectors; 3-14

from arcs; 23-13

CP nets,

requirements for; 4-4

with Design/CPN editor (chapter); 4-1

CPN nodes,

from auxiliary nodes; 23-11

CPN regions,

from auxiliary regions; 23-11

general technique; 4-4

custom palette page; 24-10

diagram; 1-4

ellipses; 3-10

fusion places,

with Fusion Place command (CPN menu); 21-20

fusion sets,

global; 10-4

instance; 10-11

page; 10-10

with Fusion Place command (CPN menu); 21-20

global declaration node,

with Declaration Node (CPN menu); 4-12

graphical objects; 3-3

from text mode; 3-18

group,

connectors, with Select All Connectors (Group menu); 27-2

nodes, with Select All Nodes (Group menu); 27-2

regions, with Select All Regions (Group menu);
27-2

guard region (transition node); 21-6

guards,

from auxiliary regions; 23-11

with CPN Region (CPN menu); 4-7

hierarchical CP nets,

by developing on a subpage; 11-14

hierarchy regions,

when moving detail to a subpage with Move to
Subpage command (CPN menu); 21-18

history charts; 15-14

with Chart command (CPN menu); 21-10

initial markings,

from auxiliary regions; 23-11

with CPN Region command; 21-5

labels; 3-12

creating (cont'd),

- labels (cont'd),
 - bar charts; 15-6
- line charts; 15-19
 - dialog description; 15-20
 - with Chart command (CPN menu); 21-13
- lists,
 - operators; 32-14
- log regions; 21-6
 - from auxiliary regions; 23-11
- master page; 24-10
- multisets; 2-8, 37-2, 37-10
 - with multiset creation operator, reference description; 41-3
- name regions; 21-6
- page instances; 10-11
- pages,
 - with New Page (Page menu); 4-14, 26-2
- partial constraints,
 - with guards; 5-12
- perimeter transitions,
 - when moving detail to a subpage with Move to Subpage command (CPN menu); 21-17
- place names,
 - from auxiliary regions; 23-11
- places,
 - multiple; 21-2
 - with Place (CPN menu); 4-7
- port places,
 - when moving detail to a subpage with Move to Subpage command (CPN menu); 21-17
 - with Port Place command (CPN menu); 21-23
- rectangles; 3-6
 - a series of; 3-9
 - adding text while; 3-10
- regions; 3-16
- socket places,
 - when moving detail to a subpage with Move to Subpage command (CPN menu); 21-17
- statistical variables; 14-2
- style sheet; 24-10
- subpages; 11-5
- substitution transitions; 11-5, 11-15
 - when moving detail to a subpage with Move to Subpage command (CPN menu); 21-18
 - with Substitution Transition command (CPN menu); 21-19
- templates; 24-10
- time regions,
 - for transitions; 21-6
 - from auxiliary regions; 23-11

creating (cont'd),

- transitions,
 - multiple; 21-3
 - names from auxiliary regions; 23-11
 - regions from auxiliary regions; 23-11
 - with Transition (CPN menu); 4-4

current,

- marking key region,
 - term definition; 6-14
- marking regions,
 - characteristics; 17-10
 - removing with Remove Sim Regions command (CPN menu); 21-28
 - term definition; 6-14
- markings,
 - term definition; 2-11
- object,
 - term definition; 3-18
- state,
 - term definition; 2-11
- value statistical variable,
 - obtaining, with SV'value; 14-4

cursor keys,

- navigating to an error with; 7-4

custom palette page,

- creating; 24-10
- using; 24-11

Cut (Edit menu),

- reference description; 20-2
- restrictions; 20-3

cutting,

- bar charts; 15-13
- history charts; 15-17
- line charts; 15-31

cyclic manipulation,

- of enumerated and indexed value colorsets; 32-24

D

data,

- analysis,
 - bar charts in Resource Use model; 16-10
 - history charts in Resource Use model; 16-11
 - line charts in Resource Use model; 16-13
- characteristics as CP net component; 2-1
- objects,
 - See* tokens;

datatypes,

- See Also* colorsets;
- abstract,
 - not permitted in CPN ML; 30-4

datatypes (cont'd),

- CPN,
 - See* colorsets;
- functions; 38-1
- user-defined,
 - See* colorsets;

DB file,

- characteristics and components; 1-3

debugging,

- ML Evaluate command (Aux menu),
 - use for testing new and changed code; 23-14
- optional syntax errors,
 - specifying which to be reported, with the Syntax Options command (Set menu); 24-18

Declaration Node (CPN menu),

- creating a global declaration node with; 4-12
- reference description; 21-7

declaration node tool,

- term definition and illustration; 4-13

declaration nodes,

- See Also* CPN objects, classes of;
- as CPN ML extension; 30-3
- creating,
 - multiple; 21-8
 - with Declaration Node command (CPN menu); 21-7
- global,
 - creating, with Declaration Node (CPN menu); 4-12
 - term definition; 2-3
- restrictions; 21-8
- term definition and types; 17-9

declarations,

- See* colorsets; global, declaration node;

declare clause,

- syntax and characteristics; 32-18

declaring,

- functions,
 - syntax and characteristics; 38-1
- local variables,
 - with let construct; 38-1

decomposition,

- See* substitution transitions;

defaults,

- diagram,
 - changing; 17-19
 - copying; 17-19
 - page attributes, changing; 24-11
 - shape attributes, changing; 24-5
 - term definition and characteristics; 17-18
 - text attributes, changing; 24-2
 - using; 17-19
- standard,
 - term definition; 17-18

defaults (cont'd),

- system,
 - changing; 17-19
 - copying; 17-19
 - page attributes, changing; 24-11
 - shape attributes, changing; 24-5
 - term definition and characteristics; 17-18
 - text attributes, changing; 24-2
 - using; 17-19

defining,

- bar charts,
 - in Resource Use model; 16-9
- history charts,
 - in Resource Use model; 16-11
- line charts,
 - in Resource Use model; 16-13
- occurrence set,
 - with a single binding; 22-8

delay expressions,

- See Also* time;
- term definition and syntax; 12-7

deleting,

- See Also* creating;
- bar charts; 15-14
- connectors,
 - dangling; 3-15
- fusion sets; 10-8
 - with Fusion Place command (CPN menu); 21-22
- graphical objects; 3-19
- history charts; 15-17
- line charts; 15-31
- occurrence set; 22-9
- parents; 3-17
- rectangles; 3-8
- subpages; 11-23
 - references to; 11-24

delimiter blocking,

- specifying brackets and character counts used by,
 - with Text Options command (Set menu); 24-17

descendants,

- objects,
 - impact of CPN Region command on; 21-7
- term definition and characteristics; 17-5

deselecting,

- See Also* selecting;
- groups; 3-21

Design/CPN,

- See Also* CP nets;
- characteristics and components; 1-2
- data,
 - characteristics; 2-3

Design/CPN (cont'd),

- dynamics,
 - executing a CP net, (chapter); 6-1
 - introduction, (chapter); 5-1
- editor,
 - creating a CP net (chapter); 4-1
 - introduction (chapter); 3-1
- getting started with,
 - (chapter); 1-1
- hierarchy,
 - characteristics; 9-2
- quitting; 1-7
- settings file missing or obsolete,
 - problem symptoms and solutions; B-1
- simulator,
 - See* simulator;
- term definition; 1-1
- terminology; 1-1
- user interface components; 1-5
- X-Windows, characteristics relative to; 1-2

designating,

- See Also* specifying;
- prime page; 6-3

detecting errors,

- See* troubleshooting;

diagonal,

- spacing nodes along,
 - See* Vertical Spread (Align menu) and Horizontal Spread (Align menu);

diagrams,

- characteristics and components; 1-3
- closing; 1-7
- creating; 1-4
- defaults,
 - attributes, term definition; 4-2
 - changing; 17-19
 - changing graphic attributes; 24-3
 - changing shape attributes; 24-5
 - copying; 4-3, 17-19
 - page attributes, changing; 24-11
 - setting for ML configuration options; 24-33
 - setting system defaults with; 24-33
 - term definition and characteristics; 4-2, 17-18
 - text attributes, changing; 24-2
 - using; 17-19
- file,
 - characteristics and components; 1-3
- navigating; 1-6, 17-1
- opening; 1-4
- printing; 1-6
- reswitching,
 - with Reswitch command (Sim menu); 22-13

diagrams (cont'd),

- term definition; 1-2

dialog box,

- appearance; 1-4

different bindings,

- explanation of the setting in the Occurrence Set Options dialog; 8-21

Displace (Makeup menu),

- reference description; 25-4

displacing,

- objects,
 - with Displace command (Makeup menu); 25-4

display,

- attributes,
 - changing; 4-2
 - graphical objects, term definition; 3-2
 - term definition; 4-2
- during simulation,
 - specifying, with Interactive Simulation Options command (Set menu); 24-27
- overlapping objects,
 - specifying the; 24-4

display attributes,

- See* attributes;

displaying,

- page borders,
 - rendering visible or invisible; 24-11
- page instance name,
 - with Select Instance command (Sim menu); 22-12
- regions,
 - with Show Regions (Makeup menu); 25-9

dist' (distance) function,

- syntax and characteristics; 32-24

division operation (/),

- reference description; 32-6

documenting,

- models,
 - with auxiliary objects; 3-1

double quote (") ,

- type variable use; 30-4

double-clicking,

- editor regions; 17-3
- hierarchy page node; 17-3
- port places; 17-3
- simulator regions; 17-3
- substitution transitions; 17-3

Drag (Makeup menu),

- moving a region with; 11-9
- reference description; 25-2

drag tool,

- illustration; 25-2

dragging,
 endpoints of connectors,
 with Drag command (Makeup menu); 25-3
 objects,
 with Drag command (Makeup menu); 25-2

drawing,
See Also creating;
 arcs,
 single and multi-segment; 21-4
 connectors,
 single-segment and multi-segment; 23-2
 environment,
 resetting; 3-5
 expression,
 term definition; 15-30
 graphics tool used for; 3-3
 rectangles; 3-6
 tool,
 term definition; 3-2

duplicate colorsets,
See Also colorsets;
 term definition and characteristics; 2-5

Duplicate Node (Makeup menu),
 reference description; 25-6

duplicating,
 objects,
 with Duplicate Node command (Makeup menu);
 25-6

dynamics,
 Design/CPN,
 executing a CP net, (chapter); 6-1
 introduction, (chapter); 5-1

E

Edit menu,
See Also menus;
 Clear command; 20-6
 commands reference (chapter); 20-1
 Copy command; 20-3
 Cut command; 20-2
 Get Info command; 20-7
 Paste command; 20-5
 Redo command; 20-2
 Undo command; 20-2

editing,
See Also creating;
 area,
 bind dialog; 22-3
 bar charts; 15-6

editing (cont'd),
 bindings for CPN variables,
 with Bind command (Sim menu); 22-3
 connectors; 3-15
 history charts; 15-16
 line charts; 15-21
 regions; 3-17
 text; 3-13

editor,
 Design/CPN,
 creating a CP net (chapter); 4-1
 introduction (chapter); 3-1
 key/popup regions,
 creating and deleting; 17-13
 list; 17-12
 options,
 command that access; 17-17
 regions,
 changing the visibility of; 17-3

Ellipse (Aux menu),
 creating ellipses with; 3-10
 reference description; 23-4

ellipses,
See Also graphical objects; places;
 creating; 3-10
 and manipulating; 23-4
 creation mode,
 characteristics; 3-11
 shape attributes,
 changing; 24-6
 tool,
 characteristics and illustration; 3-11

empty multiset,
See Also multisets;
 constant,
 reference description; 41-1
 syntax and characteristics; 37-3
 term definition; 2-7

enabled,
 list,
 putting all enabled transitions on, for SalesNet
 model execution with conflict; 8-12
 scanning, for SalesNet model execution with
 conflict; 8-12
 term definition; 8-11
 updating, for SalesNet model execution with
 conflict; 8-16

enablement,
See Also algorithms; bindings; occurrence sets;
 criteria for; 5-3
 enabling bindings,
 identical; 8-3

enablement (cont'd),

- enabling bindings (cont'd),
 - multiple; 8-2
- factors determining; 5-2
- simulated time impact on; 12-3
- simulation region identifying; 6-14
- term definition; 5-2

enabling,

- bindings,
 - conflict issues; 8-6
 - term definition; 5-8
- multiset,
 - term definition; 5-3

endpoint,

- arcs,
 - handle use; 21-5
- handle,
 - connectors, using to reattach a connector; 23-2
- regions,
 - creating; 17-14
 - term definition and characteristics; 17-11, 17-13

enlarging,

- pages,
 - with Blowup (Page menu); 26-2

Enter Editor (File menu),

- leaving the simulator with; 6-18
- reference description; 19-12

Enter Group Mode (Group menu),

- reference description; 27-2

Enter Simulator (File menu),

- entering the simulator with; 6-11
- reference description; 19-13
- specifying which errors reported,
 - with the Syntax Options command (Set menu); 24-18

Enter Text Mode (Text menu),

- reference description; 28-2

entering,

- See Also* leaving;
- group mode,
 - with Enter Group Mode (Group menu); 27-2
- simulator; 6-11
 - with a saved state; 6-12
- text; 3-13
 - mode; 28-2

entities,

- See* modeling; representation;

enumerated values,

- See Also* colorsets;
- colorsets,
 - cyclic manipulation of values; 32-24
 - obtaining position of a particular value; 32-22

enumerated values (cont'd),

- colorsets (cont'd),
 - obtaining value at particular position; 32-22
 - ordering of; 32-28
 - syntax and characteristics; 32-8
 - term definition; 2-4

environment,

- drawing,
 - resetting; 3-5
- establishing; 4-3
- graphical,
 - setting; 4-2
- runtime,
 - ML (chapter); 41-1

equality multiset (==) operator,

- syntax and use; 37-5

equal numeric operator (=),

- boolean operator used in guards; 5-8

equal string operator (=),

- reference description; 32-8

equality operator (==),

- comparing multisets with,
 - reference description; 41-2

errors,

- error box,
 - interpreting; 7-3
 - term definition; 7-2
- locating,
 - with text pointers; 7-3
- missing colorset,
 - fixing; 7-4
- ML,
 - detecting; 7-6
- reporting,
 - Syntax Check command (CPN menu) use for; 21-26
- syntax,
 - detecting, with the Syntax Check command (CPN menu); 6-1
 - handling, (chapter); 7-1
 - list of syntax check error messages; 17-22
 - missing colorset; 7-2
 - optional, specifying which to be reported, with the Syntax Options command (Set menu); 24-18
 - undeclared CPN variables; 7-4
- undeclared CPN variables,
 - fixing; 7-4

essential graphical objects,

- term definition and characteristics; 17-5

establishing,

- See Also* creating; specifying;

establishing (cont'd),
 environment; 4-3

evaluating,
 ML code; 23-14
 output arc inscription,
 during transition firing; 5-14

examples,
 procedures for testing; 30-5

exception mechanism,
 CPN ML feature; 30-2

exclamation points,
 double (!),
 internal representation of multisets with; 37-9
 operator for constructing multisets; 37-10

executing,
 CP nets,
 (chapter); 6-1
 concurrent execution; 8-4
 overview; 5-1
 occurrence set elements; 8-13
 SalesNet model with conflict; 8-12

execution,
 concurrent,
 SalesNet model; 8-7
 CP nets,
 overview; 2-15

exp operation (exponential),
 reference description; 32-6

experimenting,
See modeling;

exponential operation (exp),
 reference description; 32-6

expressions,
 (chapter); 36-1
 arcs,
 missing, reporting with the Syntax Options
 command (Set menu); 24-19
 boolean,
 bar chart code segment use of; 15-12
 line chart code segment use of; 15-29
 used in guards; 5-8
 canonical,
 term definition; 36-1
 compound,
 term definition; 36-1
 constructors; 36-3, 36-7
 lists; 36-8
 tuples; 36-8
 control,
 term definition; 15-29
 delay,
 syntax and characteristics; 12-7

expressions (cont'd),
 drawing,
 term definition; 15-30
 evaluation; 36-4
 association; 36-4
 in CP nets; 36-8
 operator precedence; 36-4
 order; 36-4
 inscription,
 term definition; 36-1
 multisets,
 syntax; 37-2
 patterns; 36-6
 list pattern matching; 36-7
 tuple pattern matching; 36-7
 simple,
 term definition; 36-1
 syntax specifiers and reserved words; 36-2
 term definition; 36-1

extent,
 term definition; 34-1

external arcs,
 term definition; 21-17

ext_col function,
 reference description; 41-4
 syntax and characteristics; 37-8

ext_ms function,
 reference description; 41-4
 syntax and characteristics; 37-8

F

fair automatic simulation,
 characteristics; 6-7
 specifying, with General Simulation Options
 command (Set menu); 24-23

fair interactive simulation,
 characteristics; 6-7
 specifying, with General Simulation Options
 command (Set menu); 24-23

fair simulation,
 term definition and code generation; 24-21

fast automatic simulation,
 characteristics; 6-7
 specifying,
 with General Simulation Options command (Set
 menu); 24-23
 term definition and code generation,
 with the Simulation Code Options command (Set
 menu); 24-21

fast automatic simulation (cont'd),

- removing,
 - with Remove Sim Regions command (CPN menu); 21-28
- term definition; 6-14

File menu,

- See Also* menus;
- Close command; 19-3
- commands reference (chapter); 19-1
- Enter Editor command; 19-12
 - leaving the simulator with; 6-18
- Enter Simulator command; 19-13
 - entering the simulator with; 6-11
- Load IDEF command; 19-10
- Load State command; 19-11
 - loading execution states with; 5-16
- Load Subdiagram command; 19-8
- Load Text command; 19-9
- New command; 19-2
- Open command; 19-2
- Page Preview command; 19-6
- Page Setup command; 19-6
- Print command; 19-7
- Quit command; 19-14
- Revert command; 19-5
- Save As command; 19-5
- Save command; 19-3
- Save State command; 19-11
 - saving execution states with; 5-15
- Save Subdiagram command; 19-7
- Save Text command; 19-9

file system,

- characteristics and Design/CPN components; 1-3

fill pattern,

- changing; 24-5

filter function,

- reference description; 41-4
- syntax and characteristics; 37-8

filtering,

- multisets; 37-8
 - with filter function; 41-4

Find (Text menu),

- reference description; 28-3

Find Beginning (Text menu),

- reference description; 28-4

Find Next (Text menu),

- reference description; 28-4

finding,

- text,
 - in a group, with Find (Text menu); 28-4
 - with Find (Text menu); 28-3

fire,

- term definition; 5-12

firing,

- concurrent transitions; 8-3
- simulation region identifying; 6-14

first constant,

- syntax and characteristics; 32-21

first value,

- statistical variable,
 - obtaining, with SV'first; 14-4

Fit to Text (Makeup menu),

- reference description; 25-5

fn (function),

- expression specifier use; 36-2

fn reserved word,

- function datatype indicated by expression using; 38-1

free variables,

- term definition; 40-5

full multiset constant,

- syntax and characteristics; 37-3

fun reserved word,

- declaring functions with; 38-1

functional language,

- CPN ML feature; 30-1

functions,

- (chapter); 38-1
- addition of,
 - with addition operator (+); 41-7
- application,
 - output arc inscription example; 40-13
- arc inscription use and restrictions; 40-6
- boolean,
 - standard ML, list of supported; 41-10
- cf (coefficient) function; 37-7, 41-3
- clr' function; 32-23
- col' function; 32-22
- compress function; 37-10
- datatype; 38-1
- declaring,
 - syntax and characteristics; 38-1
- Design/OA,
 - use and restrictions; 41-8
- dist' (distance) function; 32-24
- expressions; 33-4
 - declaring value identifiers with; 33-4
- ext_col function; 37-8, 41-4
- ext_ms function; 37-8, 41-4
- filter function; 37-8, 41-4
- fn (function); 36-2
 - expression specifier use; 36-2
- function; 41-1, 41-5

functions (cont'd),

- function type operator,
 - association rule; 36-5
- I/O,
 - standard ML, list of supported; 41-12
- identity function; 41-7
- ign function; 41-7
- in' function; 32-26
- index' (index number) function; 32-23
- input_col function; 41-8
- input_ms function; 37-9, 41-5, 41-8
- input_tms function; 41-8
- integer,
 - standard ML, list of supported; 41-10
- invoking; 38-3
- less than (lt') function; 32-16, 32-19
- linear extension; 37-8
 - creating, with ext_col and ext_ms functions; 41-4
- list,
 - standard ML, list of supported; 41-11
- list_to_ms function; 37-8, 41-4
- lt' (less than) function; 32-16, 32-19
- membership (in') function; 32-26
- membership (of_id') function; 32-27
- ms_to_list function; 37-8, 41-4
- mult' function; 32-25, 37-5, 41-2
- multiply multiset (mult') function; 32-25
- of_id' function; 32-27
- ord' (ordinal number) function; 32-22
- output_col function; 41-8
- output_ms function; 37-9, 41-5, 41-8
- output_tms function; 41-8
- predefined,
 - reference descriptions; 41-1
- ran' (random value) function; 32-16, 32-18
- random function; 37-7, 40-5, 40-15, 41-4
- random value (ran') function; 32-16, 32-18
- real,
 - standard ML, list of supported; 41-10
- realtime,
 - standard ML, list of supported; 41-12
- reference,
 - standard ML, list of supported; 41-12
- rot' (rotation) function; 32-24
- scalar multiplication of,
 - with scalar multiplication operator (*); 41-7
- size function; 37-7, 41-3
- step function; 41-5
- stop_simulation function; 41-6
- string,
 - standard ML, list of supported; 41-11

functions (cont'd),

- subtraction of,
 - with subtraction operator (-); 41-7
- SV'avrg function; 14-3
- SV'count function; 14-3
- SV'first function; 14-4
- SV'init function; 14-5
- SV'maximum function; 14-4
- SV'minimum function; 14-4
- SV'ss function; 14-5
- SV'ssd function; 14-5
- SV'std function; 14-4
- SV'sum function; 14-5
- SV'value function; 14-4
- SV'vari function; 14-5
- term definition; 41-1
- time function; 39-2, 40-15, 41-5
- time-related; 39-4
- trigonometric; 32-6
- value (clr') function; 32-23
- value (col') function; 32-22
- with_code function; 41-6
- with_time function; 39-4, 41-6
- write_report function; 41-6
- zero function; 41-7

fusion,

- key region,
 - term definition and illustration; 10-6
- names,
 - characteristics and use; 17-21
 - term definition; 17-20
- places,
 - (chapter); 10-1
 - changing the type of; 21-21
 - characteristics; 9-2
 - creating; 21-20
 - effect of duplication on; 25-6
 - effect of moving nodes between pages on; 25-7
 - instance, changing the type of; 21-21
 - instance, renaming; 21-21
 - instance, term definition; 10-13
 - multiple pages; 10-8
 - page, changing the type of; 21-21
 - page, renaming; 21-21
 - page, term definition; 10-10
 - renaming; 21-21
 - single page; 10-3
 - term definition; 9-2, 10-1
 - turning non-fusion places into; 21-23
- regions,
 - CPN Region command restricted from creating; 21-5

fusion (cont'd),

regions (cont'd),

deletion requirements; 20-6

term definition; 10-6

sets,

adding places to; 10-6, 21-22

changing the type of; 21-23

creating; 21-20

deleting; 21-22

global, creating; 10-4

global, term definition; 10-5

impact on the results of the Change Marking
command (Sim menu); 22-11

initial markings and; 10-7

instance, creating; 10-11

instance, simulation with; 10-14

moving places between; 21-23

multiple, working with; 10-9

multiplicity and; 10-12

naming conventions; 21-23

page, creating; 10-9

page, term definition; 10-9

page-spanning, working with; 10-8

removing places from; 10-7

selecting; 27-3

subtracting places from; 21-22

term definition; 9-2, 10-1

turning fusion places into non-fusion places;
21-23

subsets,

instance, term definition; 10-13

page, term definition; 10-10

Fusion Place (CPN menu),

adding places to a fusion set with; 10-6

creating a fusion set with; 10-4

deleting fusion sets with; 10-8

reference description; 21-20

removing places from a fusion set with; 10-7

G

general port place,

characteristics; 21-24

General Simulation Options (Set menu),

reference description; 24-23

selecting the termination conditions with; 6-9

selecting the type of simulation with; 6-8

generating,

See Also creating;

generating (cont'd),

code,

for different types of execution, Simulation Code

Options (Set menu); 6-5

initial system state,

with Initial State command (Sim menu); 22-11

Get Info (Edit menu),

reference description; 20-7

global,

declaration nodes,

creating; 4-12, 21-7

creating a page for; 4-14

creation mode, term definition; 4-13

declaring a timed colorset in; 12-6

term definition; 2-3

fusion places,

changing the type of; 21-21

physical appearance; 10-5

renaming; 21-21

term definition; 10-5

fusion set,

term definition; 10-5

reference variables,

syntax and characteristics; 35-3

term definition and characteristics; 17-6

Graphic Attributes (Set menu),

reference description; 24-3

graphical,

animation,

code segments used for; 40-17

attributes,

changing for selected objects; 24-3

changing system and/or diagram defaults; 24-3

components; 17-15

environment,

setting; 4-2

term definition; 4-2

functions,

Design/OA, use and restrictions; 41-8

objects,

adjusting to fit text; 25-5

creating; 3-3

creating, from text mode; 3-18

deleting; 3-19

editing; 3-2

multiple, working with; 3-18

multiple, working with different types; 3-15

selecting; 3-18

term definition and characteristics; 3-1, 17-5

graphics,

Design/CPN editor,

(chapter); 3-1

graphics (cont'd),

- enabled transitions,
 - specifying with Transition Feedback Options
 - command (Set menu); 24-31
- graphics tool,
 - term definition and illustration; 3-3
- mode,
 - characteristics; 3-2
 - term definition; 3-2
- updating during simulation,
 - specifying, with Interactive Simulation Options
 - command (Set menu); 24-27

greater than multiset operator (>>),

- comparing multisets with,
 - reference description; 41-3
- syntax and use; 37-6

greater than numeric operator (>),

- in guards; 5-8

greater than string operator (>),

- reference description; 32-8

greater than or equal multiset operator (>>=),

- comparing multisets with,
 - reference description; 41-3
- syntax and use; 37-6

greater than or equal numeric operator (>=),

- in guards; 5-8

greater than or equal string operator (>=),

- reference description; 32-8

grid lines,

- bar chart,
 - specifying; 21-11
- history chart,
 - specifying; 21-11
- line chart,
 - specifying the characteristics of; 21-15

Group menu,

- See Also* menus;
- commands reference (chapter); 27-1
- Enter Group Mode command; 27-2
- Leave Group Mode command; 27-2
- Regroup command,
 - reconstructing groups with; 3-21
- Select All Connectors command; 27-2
- Select All Nodes command; 27-2
- Select All Regions command; 27-2
- Select Fusion Set command; 27-3
- selecting groups with; 3-20
- Ungroup command,
 - deselecting groups with; 3-21

groups,

- aligning,
 - with Align menu commands; 29-2

groups (cont'd),

- creating,
 - with Select All Connectors (Group menu); 27-2
 - with Select All Nodes (Group menu); 27-2
 - with Select All Regions (Group menu); 27-2
- deselecting; 3-21
- effect of,
 - Copy command on; 20-4
 - Cut command on; 20-2
 - Paste command on; 20-5
- group tool,
 - characteristics and illustration; 3-20
- mixed,
 - restrictions; 3-20
- mode,
 - detecting when system is in both text mode and;
 - 28-2
 - entering; 27-2
 - leaving; 27-2
 - Select command (Makeup menu) not available
 - during; 25-2
 - term definition; 3-20
- moving,
 - to another page, with Move Node command
 - (Makeup menu); 25-7
 - to beginning of; 28-4
- operations on; 3-22
- reconstructing; 3-21
- searching and replace in,
 - with Find (Text menu); 28-4
- selecting; 3-20
- term definition; 3-19

guards,

- See Also* input arc inscriptions;
- characteristics,
 - and term definition; 2-14
 - as CP net component; 2-1
- constraining,
 - input arc inscriptions with; 40-9
 - the values of tokens with; 5-8
- creating,
 - from auxiliary regions; 23-11
 - partial constraints with; 5-12
 - with CPN Region (CPN menu); 4-7
- expression evaluation in; 36-9
- functions not permitted with,
 - with_code; 41-6
 - with_time; 41-6
- guard region creation mode,
 - term definition; 4-7
- if/then/else restrictions; 38-2

guards (cont'd),

- regions,
 - creating; 21-6
- role in determining enablement; 5-3
- syntax; 5-8
- term definition; 5-8
- transition,
 - region syntax, use, and restrictions; 40-15

H

handles,

- See Also* arcs; connectors;
- connector endpoint,
 - using; 23-2
- term definition; 3-7

handling,

- syntax errors,
 - (chapter); 7-1

hidden,

- regions,
 - selecting with arrow keys; 25-9

Hide Regions (Makeup menu),

- reference description; 25-9

hiding,

- regions,
 - with Hide Regions (Makeup menu); 25-9

hierarchy,

- See Also* fusion; substitution transitions;
- effect of,
 - Copy command on; 20-4
 - Cut command on; 20-3
- hierarchical CP nets,
 - improving appearance; 11-8
- hierarchical decomposition,
 - term definition and characteristics; 9-1
- hierarchy key region,
 - term definition; 11-7
- introduction,
 - (chapter); 9-1
- objects,
 - effect of duplication on; 25-6
 - effect of moving nodes between pages on; 25-7
- page,
 - appearance after top-down development; 11-16
 - changing the layout and redrawing options; 24-14
 - characteristics and components; 1-6
 - error box location on; 7-2
 - for a hierarchical CP net diagram; 11-2
 - improving appearance; 11-12

hierarchy (cont'd),

- page (cont'd),
 - moving detail to a subpage of with Move to Subpage command (CPN menu); 21-17
 - navigating to pages from; 17-3
 - redrawing; 26-3
 - renaming pages from; 4-15
- regions,
 - CPN Region command restricted from creating; 21-5
 - creating when moving detail to a subpage; 21-18
 - deletion requirements; 20-6
 - term definition; 11-8
- term definition; 9-2

Hierarchy Page Options (Set menu),

- reference description; 24-14

history charts,

- bar names,
 - specifying; 21-11
- bars,
 - specifying; 21-10
- characteristics and use; 15-14
- copying; 15-17
- creating; 15-14
 - with Chart command (CPN menu); 21-10
- cutting; 15-17
- data analysis,
 - in Resource Use model; 16-11
- defining,
 - in Resource Use model; 16-11
- deleting; 15-17
- editing; 15-16
- grid lines,
 - specifying; 21-11
- in Resource Use model; 16-10
- initializing; 21-12
- pasting; 15-17
- redefining; 15-16
- regions,
 - reference description; 21-11
- saving a copy; 21-12
- specifying values; 15-16
- title,
 - specifying; 21-11
- updating; 15-16
 - update period specification; 21-12
- values,
 - specifying the type and range; 21-13

Horizontal (Align menu),

- reference description; 29-2

Horizontal Spread (Align menu),

- reference description; 29-4

hostname,
 specifying,
 with ML Configuration Options command (Set menu); 24-32

hypertext,
 using text pointers as; 28-4

I, J

I/O,
 arc inscriptions not permitted to use; 40-5
 code segments used for; 40-17
 functions,
 standard ML, list of supported; 41-12
 input/output port,
 characteristics; 21-24
 multiset; 37-9
 with input_ms and output_ms functions; 41-5

identical bindings,
 explanation of the setting in the Occurrence Set Options dialog; 8-21

identifiers,
 CPN ML (chapter); 31-1
 duplication of,
 when permitted; 31-2
 predeclared; 31-2

identity function,
 reference description; 41-7

if-then selector (%),
 reference description; 41-6

if-then-else expression,
 output arc inscription example; 40-13

if-then-else selector (/),
 reference description; 41-6

if/then/else (boolean conditional),
 expression specifier use; 36-2

ign function,
 reference description; 41-7

ignoring,
 with zero function; 41-7

in' (membership) function,
 syntax and characteristics; 32-26

included (mode attribute),
 characteristics; 17-16

incremental net development,
 prime page role; 6-3

indeterminacy,
 term definition; 8-1

index' (index number) function,
 syntax and characteristics; 32-23

indexed values,
 See Also colorsets;
 colorsets,
 cyclic manipulation of values; 32-24
 obtaining identifier-value for index number; 32-23
 obtaining index number for identifier-value; 32-23
 obtaining position of a particular value; 32-22
 obtaining value at particular position; 32-22
 ordering of; 32-28
 syntax and characteristics; 32-9
 input arc inscription example; 40-10

init keyword,
 bar chart code segment use of; 15-11
 line chart code segment use of; 15-28

initial markings,
 See Also places;
 changing; 8-9
 creating from auxiliary regions; 23-11
 creation mode,
 term definition; 4-9
 establishing,
 for SalesNet model execution with conflict; 8-12
 functions permitted with,
 with_code; 41-6
 with_time; 41-6
 fusion sets and; 10-7
 region (place),
 syntax and characteristics; 40-3
 regions,
 creating with CPN Region command; 21-5
 initializing with Initial State command (Sim menu); 8-10
 term definition and characteristics; 2-12
 specifying,
 with CPN Region (CPN menu); 4-9
 term definition; 40-3
 time stamps and; 12-9

initial state,
 term definition; 2-11

Initial State (Sim menu),
 initializing SalesNet after changing in the simulator;
 8-10
 initializing the CP net state with; 6-18
 reference description; 22-11

initialization section,
 term definition; 15-11, 15-28

initializing,
 bar charts; 21-12
 history charts; 21-12
 line charts; 21-15
 statistical variables; 14-2

initiating,

- automatic simulation,
 - with Automatic Run command (Sim menu); 22-10
- interactive simulation,
 - with Interactive Run command (Sim menu); 22-10

input,

- keyword,
 - code segment input clause identified by; 40-17
 - term definition; 13-2
- multiset; 37-9
- pattern,
 - code segment, term definition and characteristics; 13-2
- place,
 - multiset, role in determining enablement; 5-3
 - term definition; 2-13
- port,
 - characteristics; 21-24
- token regions,
 - characteristics; 17-11
 - removing with Remove Sim Regions command (CPN menu); 21-28

input arcs,

- See Also* arcs; guards; inscriptions;
- inscriptions,
 - characteristics as CP net component; 2-1
 - conditional; 40-6
 - evaluating; 5-13
 - examples; 40-7
 - function application use and restrictions; 40-6
 - functions not permitted with, `with_code`; 41-6
 - functions not permitted with, `with_time`; 41-6
 - if/then/else restrictions; 38-2
 - overriding time stamps with; 40-5
 - role in determining enablement; 5-3
 - specifying, with constants; 5-4
 - specifying, with CPN variables; 5-7
 - term definition; 2-14
 - timing considerations; 39-2
- term definition; 2-13

input_col function,

- reference description; 41-8

input_ms function,

- reference description; 41-5, 41-8
- syntax and characteristics; 37-9

input_tms function,

- reference description; 41-8

inscriptions,

- (chapter); 40-1
- arc inscription region,
 - term definition; 2-14

inscriptions (cont'd),

- arcs,
 - detecting errors in; 7-6
 - missing, reporting with the Syntax Options command (Set menu); 24-19
 - term definition and characteristics; 2-14
- example diagram; 40-2
- expressions,
 - term definition; 36-1
- if/then/else restrictions and uses; 38-2
- input arc,
 - characteristics as CP net component; 2-1
 - conditional; 40-6
 - examples; 40-7
 - function application use and restrictions; 40-6
 - overriding time stamps with; 40-5
 - term definition; 2-14
 - timing considerations; 39-2
- output arc,
 - appending time delays to; 40-6
 - characteristics as CP net component; 2-1
 - conditional; 40-6
 - delay expressions on; 12-8
 - evaluating during transition firing; 5-14
 - examples; 40-13
 - function application use and restrictions; 40-6
 - term definition; 2-14
 - timing considerations; 39-2
- term definition; 40-1
- types of; 40-1

instances,

- fusion places,
 - changing the type of; 21-21
 - renaming; 21-21
 - term definition; 10-13
- fusion sets,
 - creating; 10-11
 - simulation with; 10-14
- fusion subsets,
 - term definition; 10-13
- pages,
 - creating; 10-11
 - displaying name of; 22-12
 - specifying the number of; 24-13
- reference variables,
 - syntax and characteristics; 35-3

instrumentation,

- code segments used for; 40-17

integers,

- See Also* colorsets;
- colorsets,
 - ordering of; 32-28

integers (cont'd),
 colorsets (cont'd),
 syntax and characteristics; 32-4
 term definition; 2-4
 testing for membership in; 32-26
 functions,
 standard ML, list of supported; 41-10
 value identifiers,
 declaring; 33-2

interaction,
 system options,
 changing; 24-15

Interaction Options (Set menu),
 reference description; 24-15

interactive,
 language,
 CPN ML feature; 30-1
 mode attribute,
 characteristics; 17-16
 runs,
 controlling the presence of pages during; 24-13
 controlling the presence of substitution
 transitions during; 24-12
 simulation,
 code generation; 24-21
 specifying observation methods for; 24-27

Interactive Run (Sim menu),
 constructing an occurrence set with; 8-12
 executing CP nets with; 6-16
 reference description; 22-10

Interactive Simulation Options (Set menu),
 reference description; 24-27
 setting breakpoints with; 6-15

invisible,
 objects,
 rendering; 24-5
 page borders,
 rendering for display or printing; 24-11

invoking,
 functions; 38-3

K

key regions,
 characteristics; 17-11
 editor,
 creating and deleting; 17-13
 list; 17-12
 simulator,
 creating and deleting; 17-13
 list; 17-12

key regions (cont'd),
 term definition; 6-14
 and characteristics; 17-7, 17-11

keyboard,
 Design/CPN use of; 1-4

keys and shortcuts,
 (chapter); A-1

keywords,
 action,
 term definition; 13-2
 cond,
 bar chart code segment use of; 15-12
 line chart code segment use of; 15-29
 init,
 bar chart code segment use of; 15-11
 line chart code segment use of; 15-28
 input,
 term definition; 13-2
 output,
 term definition; 13-2

L

Label (Aux menu),
 creating rectangles with; 3-12
 reference description; 23-9

labels,
 bar charts,
 creating; 15-6
 creating, modifying, and moving; 3-12, 23-9
 differences between graphical objects and; 3-13
 label creation mode,
 characteristics; 3-12
 label tool,
 characteristics and illustration; 3-12
 resizing,
 without moving the center, with Adjust command
 (Makeup menu); 25-4
 term definition; 3-2
 and characteristics; 17-5

language characteristics,
 CPN ML; 30-1

large,
See Also colorsets;
 colorsets,
 characteristics; 32-1

last constant,
 syntax and characteristics; 32-22

layered,
 term definition; 17-1

layering,

- objects,
 - changing, with Bring Forward (Makeup menu); 25-9
- overlapping objects,
 - specifying the; 24-4

layering order,

- regions; 17-1

Leave Group Mode (Group menu),

- reference description; 27-2

Leave Text Mode (Text menu),

- reference description; 28-2

leaving,

- See Also* entering;
- group mode,
 - with Leave Group Mode (Group menu); 27-2
- text mode; 28-2

Left to Left (Align menu),

- reference description; 29-6

Left to Right (Align menu),

- reference description; 29-6

length,

- lists,
 - specifying; 32-13
- strings; 32-8

less than (lt') function,

- syntax and characteristics; 32-19
- union colorset selector use; 32-16

less than multiset operator (<=>),

- comparing multisets with,
 - reference description; 41-3
- syntax and use; 37-6

less than numeric operator (<),

- boolean operator used in guards; 5-8

less than string operator (<),

- reference description; 32-8

less than or equal multiset operator (<=>),

- comparing multisets with,
 - reference description; 41-3
- syntax and use; 37-6

less than or equal numeric operator(<=),

- boolean operator used in guards; 5-8

less than or equal string operator (<=),

- reference description; 32-8

let construct,

- declaring local variables within a function; 38-1
- expression specifier use; 36-2

Line (Aux menu),

- reference description; 23-8

line charts,

- axis labels,
 - characteristics; 15-18

line charts (cont'd),

- box,
 - characteristics; 15-18
- characteristics and use; 15-17
- chart node,
 - characteristics; 15-18
- code segment,
 - capabilities and uses; 15-24, 15-28
 - conditional section use; 15-29
 - control expression use; 15-29
 - initialization section use; 15-28
 - using action section; 15-31
- conditional drawing of line segments; 15-30
- copying; 15-31
- creating; 15-19
 - dialog description; 15-20
 - with Chart command (CPN menu); 21-13
- cutting; 15-31
- data analysis,
 - in Resource Use model; 16-13
- defining,
 - in Resource Use model; 16-13
- deleting; 15-31
- dialog box; 15-20
- editing; 15-21
- grid lines,
 - specifying the characteristics of; 21-15
- in Resource Use model; 16-12
- initializing; 21-15
- labels,
 - supplying; 15-21
- legend,
 - characteristics; 15-19
 - labels, characteristics; 15-19
- line,
 - characteristics; 15-19
- lines,
 - specifying the characteristics of; 21-14
- name,
 - specifying; 21-14
- nomenclature; 15-17
- origin,
 - specifying; 21-16
- overflow,
 - specifying methods for handling; 21-17
- pasting; 15-31
- patterns,
 - characteristics; 15-19
- redefining; 15-23
- regions,
 - specifying; 21-15
- saving a copy; 21-15

- line charts (cont'd),**
 - specifying; 15-25
 - lines characteristics; 21-14
 - step updating period; 21-16
 - time,
 - updating period; 21-16
 - title,
 - characteristics; 15-18
 - specifying; 21-15
 - updating; 15-24
 - from transition code segments; 15-27
 - specifying values for; 15-25
 - update period specification; 21-16
 - using LC_upd_chart function in a transition code segment; 15-27
 - with chart code segment; 15-24
 - values,
 - specifying the type and range; 21-16
 - X and Y axes,
 - characteristics; 15-18
- line feed,**
 - inserting in strings; 32-8
- linear extension,**
 - functions; 37-8
 - creating, with ext_col and ext_ms functions; 41-4
- lines,**
 - changing,
 - the fill pattern; 24-5
 - thickness; 24-4
 - creating, modifying, and moving; 23-8
 - line chart,
 - specifying the characteristics of; 21-14
 - shape attributes; 24-5
 - changing; 24-8
- list concatenation operator,**
 - redefinition in CPN ML; 30-4
- lists,**
 - See Also* colorsets;
 - colorsets,
 - ordering of; 32-28
 - syntax and characteristics; 32-13
 - constructing,
 - operators; 32-14
 - constructors,
 - characteristics and use; 36-8
 - input arc inscription example; 40-11
 - converting,
 - multisets to/from; 37-8
 - to multisets; 41-4
 - expression,
 - constructors; 36-3
 - operators; 36-3
- lists (cont'd),**
 - functions,
 - standard ML, list of supported; 41-11
 - length,
 - specifying; 32-13
 - operators and functions; 32-14
 - patterns,
 - characteristics and use; 36-7
 - input arc inscription example; 40-11
 - specifying,
 - in expressions; 36-2
 - in subset declarations; 32-15
 - value identifiers,
 - declaring; 33-3
- list_to_ms function,**
 - syntax and characteristics; 37-8
- ln operation (natural logarithm),**
 - reference description; 32-6
- Load IDEF (File menu),**
 - reference description; 19-10
- Load State (File menu),**
 - loading execution states with; 5-16
 - reference description; 19-11
 - restrictions; 19-12
- Load Subdiagram (File menu),**
 - reference description; 19-8
- Load Text (File menu),**
 - reference description; 19-9
- Load Text command,**
 - specifying brackets and character counts used by,
 - with Text Options command (Set menu); 24-17
- loading,**
 - See Also* saving;
 - execution states,
 - with Load State (File menu); 5-16
- local,**
 - (let/in/end),
 - expression specifier use; 36-2
 - nodes,
 - creating; 21-7
 - term definition and characteristics; 17-6
 - variables,
 - declaring with let construct; 38-1
- locality,**
 - See* concurrency; modeling;
- locking,**
 - objects; 24-4
 - regions relative to the parent,
 - with Size component (Region Attributes command-Set menu); 24-9
- log regions,**
 - characteristics and use; 13-3

log regions (cont'd),

- creating; 21-6
- from auxiliary regions; 23-11

logarithm,

- natural,
- operation (ln); 32-6

lt' (less than) function,

- union colorset selector use; 32-16, 32-19

M

macro,

- subpages compared to; 11-21

Make Node (Aux menu),

- reference description; 23-11

Make Region (Aux menu),

- creating regions with; 3-16
- reference description; 23-10

Makeup menu,

- See Also* menus;
- Adjust command; 25-4
- Bring Forward; 25-9
- Change Shape command; 25-5
- Child Object command; 25-10
 - navigating with; 17-4
 - selecting a region with; 11-9
- commands reference (chapter); 25-1
- Displace command; 25-4
- Drag command; 25-2
 - moving a region with; 11-9
- Duplicate Node command; 25-6
- Fit to Text command; 25-5
- Hide Regions command; 25-9
- Merge Node command; 25-8
- Move Node command; 25-7
- Next Object command; 25-11
 - navigating with; 17-4
- Parent Object command; 25-10
 - navigating with; 17-4
- Previous Object command; 25-11
 - navigating with; 17-4
- Select command; 25-2
- Show Regions command; 25-9

mapping,

- inputs to output,
- role of the simulator in; 8-3
- multisets; 37-8
- with filter function; 41-4

markings,

- appearance of; 2-12

markings (cont'd),

- current,
- simulation regions describing; 6-14
- term definition; 2-11
- initial,
- characteristics; 2-12
- places,
- changing, with Change Marking command (Sim menu); 22-11
- term definition and characteristics; 2-11

master page,

- creating; 24-10

maximal occurrence rule,

- term definition; 24-30

maximum,

- statistical variable,
- obtaining, with SV'max; 14-4

membership,

- testing for,
- in a subset of a colorset; 32-26
- in a union colorset component; 32-27

membership (in') function,

- syntax and characteristics; 32-26

membership (of_id') function,

- syntax and characteristics; 32-27

menus,

- See Also* Align menu; Aux menu; CPN menu; Edit menu; File menu; Group menu; Makeup menu; Page menu; Set menu; Sim menu; Text menu;
- Design/CPN,
- overview (chapter); 18-1
- menu bar,
- characteristics; 1-5

Merge Node (Makeup menu),

- See Also* Merge Options (Set menu);
- reference description; 25-8

Merge Options (Set menu),

- reference description; 24-16

merging,

- arcs,
- specifying options for; 24-16
- connectors,
- specifying options for; 24-16
- nodes,
- specifying options for; 24-16
- with Merge Node command (Makeup menu); 25-8
- text in CPN regions,
- specifying options for merging; 24-17
- text in nodes,
- specifying options for merging; 24-16

- minimum,**
 - statistical variable,
 - obtaining, with SV'min; 14-4
- mkst_col' operation,**
 - string representation for colorset operation; 32-8
 - syntax and characteristics; 32-19
- mkst_ms' operation,**
 - string representation for multiset operation; 32-8
 - syntax and characteristics; 32-20
- ML,**
 - See* CPN ML;
- ML Configuration Options (Set menu),**
 - See Also* CPN ML language;
 - reference description; 24-32
 - saving ML configuration options with; 4-3
- ML Evaluate (Aux menu),**
 - See Also* CPN ML language;
 - reference description; 23-14
- mode,**
 - attributes,
 - changing; 24-12
 - changing for supernodes; 24-13
 - characteristics and components; 17-16
 - current, generating an initial system state with; 22-11
 - graphics,
 - term definition; 3-2
 - graphics editor,
 - term definition; 3-2
 - pages,
 - changing; 24-13
 - non-text,
 - navigating in; 17-2
 - regions,
 - characteristics; 17-10
 - deletion requirements; 20-6
 - substitution transitions,
 - changing; 24-12
 - text,
 - navigating in; 17-3
 - term definition; 3-2
- Mode Attributes (Set menu),**
 - creating multiple page instances with; 10-11
 - designating prime pages with; 6-4
 - reference description; 24-12
- model,**
 - development,
 - specifying which optional syntax errors to be reported; 24-18
 - testing new and changed code; 23-14
 - options,
 - characteristics and commands that access; 17-17
- model (cont'd),**
 - ResmodSubtrans,
 - hierarchy page; 11-2
 - subpage; 11-3
 - superpage; 11-3
 - term definition; 1-1
 - time,
 - term definition; 12-3
- modeling,**
 - See Also* behavior; choice; representation;
 - Resource Use model; 16-1
 - improvement suggestions; 16-15
- models,**
 - See* FirstModel model; FirstNet model; FirstNetDemo model; Resource Use model; Sales Order model; SalesNet model;
- modifier keys,**
 - characteristics and use; A-3
- modularity,**
 - See Also* hierarchy;
 - CP nets,
 - prime page role; 6-3
- modules,**
 - not permitted in CPN ML; 30-4
- mouse,**
 - Design/CPN use of; 1-3
- Move Node (Makeup menu),**
 - reference description; 25-7
- Move to Subpage (CPN menu),**
 - creating a subpage with; 11-5
 - reference description; 21-17
 - top-down hierarchical CP net development with; 11-14
- moving,**
 - See Also* navigating;
 - connectors to a different attach point; 23-2
 - groups,
 - to another page, with Move Node command (Makeup menu); 25-7
 - with Align menu commands; 29-2
 - nodes,
 - to a different page, with Open Page (Page menu); 4-15
 - to another page, with Move Node command (Makeup menu); 25-7
 - with Align menu commands; 29-1
 - objects,
 - with Displace command (Makeup menu); 25-4
 - with Drag command (Makeup menu); 25-2
 - parents; 3-17
 - places between fusion sets; 21-23
 - rectangles; 3-8

moving (cont'd),

- rectangles (cont'd),
 - during creation; 3-8
- regions; 11-9
- to beginning of current text object,
 - with Find Beginning (Text menu); 28-4

ms reserved word,

- declaring multiset variables with; 37-1
- syntax and characteristics; 32-21

ms_to_list function,

- syntax and characteristics; 37-8

multiplication,

- (*) operator,
 - reference description; 32-6
 - scalar, multiplying multisets; 37-4
- mult' function,
 - multiplying multisets with; 37-5
 - reference description; 41-2
 - syntax and characteristics; 32-25

multiplicity,

- fusion and; 10-12
- page instances,
 - specifying; 24-13
- term definition; 10-11, 17-16

multiplying,

- functions,
 - scalar multiplication, with scalar multiplication operator (*); 41-7
- multisets; 32-25
 - mult' function reference description; 41-2
 - scalar multiplication operator reference description; 41-2
 - with mult' function; 37-5
 - with scalar multiplication (*) operator; 37-4

multiprocessing,

- Design/CPN use; 1-2

multiset creation operator (^),

- creating multisets with,
 - reference description; 41-3
- syntax and use; 37-2

multisets,

- See Also* colorsets;
- (chapter); 37-1
- adding; 2-8, 37-3
 - addition operator reference description; 41-1
- appending time list to; 39-4
- coefficient of values in,
 - obtaining, with cf function; 41-3
- comparing; 37-5
 - equality operator reference description; 41-2
 - greater than operator reference description; 41-3

multisets (cont'd),

- comparing (cont'd),
 - greater than or equal operator reference description; 41-3
 - less than operator reference description; 41-3
 - less than or equal operator reference description; 41-3
 - non equal operator reference description; 41-2
- constants,
 - syntax and characteristics; 37-3
- constructing; 37-10
 - for tuple or record colorsets; 32-25
- converting,
 - lists to/from; 37-8
 - to lists; 41-4
- creating; 37-2
 - for a small colorset; 32-21
 - with multiset creation operator, reference description; 41-3
- declaring value identifiers with; 33-4
- designator,
 - term definition and characteristics; 2-8
- empty,
 - constant syntax and characteristics; 37-3
 - predefined constant representing; 41-1
 - term definition; 2-7
- enabling,
 - term definition; 5-3
- expressions,
 - syntax; 37-2
- filtering,
 - with filter function; 41-4
- full,
 - constant syntax and characteristics; 37-3
- I/O of; 37-9
- input place,
 - role in determining enablement; 5-3
- internal representation of; 37-9
- linear extension of functions,
 - creating, with ext_col and ext_ms functions; 41-4
- mapping; 37-8
- multiplying; 32-25
 - mult' function reference description; 41-2
 - scalar multiplication operator reference description; 41-2
 - with mult' function; 37-5
 - with scalar multiplication (*) operator; 37-4
- number of elements in,
 - obtaining, with size function; 41-3
- obtaining,
 - random values from; 37-7
 - the number of elements in; 37-7

multisets (cont'd),

- output,
 - term definition; 5-14
- random value,
 - obtaining, with random function; 41-4
- reading,
 - with input_ms function; 41-5, 41-8
- removing enabling from each input place,
 - during transition firing; 5-13
- specifying; 2-8
- string representation operation; 32-8, 32-20
- subsetting; 2-9
- subtracting; 2-8, 37-4
 - subtraction operator reference description; 41-2
- term definition and characteristics; 2-7, 37-1
- time stamps and; 12-10
- timed; 37-10
- variables,
 - term definition, syntax, and use; 37-1
- with CPN variables,
 - input arc inscription example; 40-9
- with enumerated values,
 - input arc inscription example; 40-9
- writing,
 - with output_ms function; 41-5, 41-8

N

names,

- bar chart,
 - specifying; 21-10
- bars,
 - specifying for bar and history charts; 21-11
- changing; 17-21
- creating; 21-6
 - with CPN Region command; 21-5
- entering; 17-21
- fusion,
 - characteristics and use; 17-21
 - term definition; 17-20
- generating; 17-20
- line charts,
 - specifying; 21-14
- name region creation mode,
 - term definition; 4-6
- page,
 - duplicate, reporting with the Syntax Options command (Set menu); 24-19
 - instance, displaying with Select Instance command (Sim menu); 22-12

names (cont'd),

- page (cont'd),
 - ML-illegal, reporting with the Syntax Options command (Set menu); 24-19
 - term definition; 17-20
- places,
 - changing; 17-21
 - creating from auxiliary regions; 23-11
 - duplicate, reporting with the Syntax Options command (Set menu); 24-18
 - entering; 17-21
 - missing, reporting with the Syntax Options command (Set menu); 24-18
 - ML-illegal, reporting with the Syntax Options command (Set menu); 24-19
 - syntax and characteristics; 40-2
 - term definition; 17-20
- syntax; 17-20
- term definition; 17-19
- transition,
 - changing; 17-21
 - creating from auxiliary regions; 23-11
 - duplicate, reporting with the Syntax Options command (Set menu); 24-19
 - entering; 17-21
 - missing, reporting with the Syntax Options command (Set menu); 24-19
 - ML-illegal, reporting with the Syntax Options command (Set menu); 24-19
 - region reference description; 40-14
 - term definition; 17-20
- types of; 17-20
- unacceptable to the Occurrence Graph Analyzer,
 - reporting with the Syntax Options command (Set menu); 24-20
- using; 17-20

naming,

- conventions,
 - fusion sets; 21-23
- pages,
 - with Page Attributes (Set menu); 4-14
- places,
 - with CPN Region (CPN menu); 4-8
- simulation reports,
 - with Save Report command (Sim menu); 22-13
- substitution transitions; 11-8
- transitions,
 - with CPN Region (CPN menu); 4-6

natural logarithm operation (ln),

- reference description; 32-6

navigating,

- See Also* moving;

navigating (cont'd),

- by arrow keys; 17-2
- diagrams; 17-1
 - hierarchy page use for; 1-6
- from subpage to superpage,
 - by double-clicking on a port; 11-5
- in non-text mode; 17-2
- in text mode; 17-3
- means of; 17-2
- objects; 17-1
- with Design/CPN commands; 17-4

negation operation (~),

- reference description; 32-6

New (File menu),

- reference description; 19-2

New Page (Page menu),

- creating a page for global declarations with; 4-14
- reference description; 26-2

Next Object (Makeup menu),

- navigating with; 17-4
- reference description; 25-11

nodes,

- See Also* regions;
- aligning,
 - between reference points, with Between (Align menu); 29-5
 - centered, with Center (Align menu); 29-2
 - horizontally, with Horizontal (Align menu); 29-2
 - relatively, with Position (Align menu); 29-3
 - vertically, with Vertical (Align menu); 29-2
 - with Align menu commands; 29-1
- auxiliary,
 - boxes, creating, modifying, and moving; 23-3
 - creating CPN nodes from; 23-11
 - creating from auxiliary regions; 23-11
 - creating from CPN nodes; 23-13
 - creating regions from; 23-10
 - ellipses, creating, modifying, and moving; 23-4
 - labels, creating, modifying, and moving; 23-9
 - lines, creating, modifying, and moving; 23-8
 - polygons, creating, modifying, and moving; 23-5
 - rounded boxes, creating, modifying, and moving; 23-4
 - rounded polygons, creating, modifying, and moving; 23-7
 - term definition and characteristics; 17-7
 - wedges, creating, modifying, and moving; 23-7
- border,
 - changing line thickness; 24-4
 - changing line type; 24-5
- changing,
 - shape; 25-5

nodes (cont'd),

- changing (cont'd),
 - size and position; 21-8
- chart,
 - characteristics and region; 17-9
 - creating with Chart command (CPN menu); 21-8
- connector source and destination,
 - effect of moving on; 23-2
- converting regions into; 3-17
- CPN,
 - creating auxiliary nodes from; 23-13
 - creating from auxiliary nodes; 23-11
- creating a group of,
 - with Select All Nodes (Group menu); 27-2
- declaration,
 - as CPN ML extension; 30-3
 - creating multiple; 21-8
 - creating with Declaration Node command (CPN menu); 21-7
 - restrictions; 21-8
 - term definition and types; 17-9
- duplicating,
 - with Duplicate Node command (Makeup menu); 25-6
- effect of,
 - Copy command on; 20-3
 - Cut command on; 20-2
 - hiding regions on; 25-9
 - Paste command on; 20-5
- error,
 - characteristics; 21-26
- global,
 - creating with Declaration Node command (CPN menu); 21-7
- global declaration,
 - term definition; 2-3
- local,
 - creating with Declaration Node command (CPN menu); 21-7
- merging,
 - into a single node; 25-8
 - specifying options for; 24-16
- moving,
 - to a different page, with Open Page (Page menu); 4-15
 - to another page, with Move Node command (Makeup menu); 25-7
 - with Align menu commands; 29-1
- overlay options,
 - changing; 24-15
- page,
 - characteristics; 17-10

nodes (cont'd),

- page (cont'd),
 - deletion requirements; 20-6
 - navigating to a page from; 17-3
- projecting a line of,
 - with Projection (Align menu); 29-5
- regions and; 3-16
- repositioning,
 - with Align menu commands; 29-1
- resizing,
 - with Change Shape command (Makeup menu); 25-5
 - without moving the center, with Adjust command (Makeup menu); 25-4
- selecting all,
 - with Select All Nodes (Group menu); 27-2
- shapes possible to; 24-5
- source/destination,
 - effect on arcs of moving; 21-4
- spreading,
 - along a diagonal, *See* Vertical Spread (Align menu) and Horizontal Spread (Align menu);
 - equidistantly around a circle, with Circular Spread (Align menu); 29-5
 - horizontally, with Horizontal Spread (Align menu); 29-4
 - vertically, with Vertical Spread (Align menu); 29-5
- system,
 - characteristics; 17-10
- temporary,
 - creating with Declaration Node command (CPN menu); 21-7
- term definition; 3-2, 3-13
 - and characteristics; 17-5
- terminating declaration mode; 21-8
- text,
 - specifying options for merging; 24-16

non-text mode,

- navigating in; 17-2

not (boolean NOT),

- boolean operator used in guards; 5-8

not equal multiset (<><>) operator,

- comparing multisets with,
 - reference description; 41-2
- syntax and use; 37-5

not equal numeric operator (<>),

- used in guards; 5-8

not equal string operator (<>),

- reference description; 32-8

notation,

- CPN ML; 30-4

numbers,

- See Also* integers; real numbers;
- changing; 17-21
- entering; 17-21
- syntax; 17-20
- term definition; 17-19
- using; 17-20

O**objects,**

- See Also* arcs; auxiliary; connectors; labels; nodes; regions; text;
- attributes,
 - term definition and components; 17-15
- auxiliary,
 - characteristics; 17-7, 17-10
 - term definition; 17-6
- changing,
 - preventing the; 24-4
 - the order on a page for; 25-9
 - the shape of; 25-5
- CPN,
 - classes of, *See* arcs; chart nodes; declaration nodes; places; regions; transitions;
 - compared with auxiliary; 3-1
 - term definition and characteristics; 17-6
- custom palette page,
 - creating; 24-10
 - using; 24-11
- data,
 - See* tokens;
- descendants,
 - impact of CPN Region command on; 21-7
- difficult to select,
 - accessing with Select (Makeup menu); 17-4
- duplicating,
 - with Duplicate Node command (Makeup menu); 25-6
- essential graphical,
 - term definition and characteristics; 17-5
- getting information about; 20-7
- graphical,
 - term definition and characteristics; 3-1, 17-5
- hidden,
 - selecting, with Select command (Makeup); 25-2
- hierarchy,
 - term definition and characteristics; 17-5
- invisible,
 - rendering; 24-5
- locking; 24-4

objects (cont'd),

- navigating; 17-1
- non-selectable,
 - locking and unlocking; 24-4
- original parent,
 - term definition and characteristics; 17-5
- overlapping,
 - specifying display and printing of; 24-4
- repositioning,
 - with Displace command (Makeup menu); 25-4
 - with Drag command (Makeup menu); 25-2
- resizing,
 - with Change Shape command (Makeup menu); 25-5
 - without moving the center, with Adjust command (Makeup menu); 25-4
- selecting,
 - the next object in the layering order, with Next Object (Makeup menu); 25-11
 - the previous object in the layering order, with Previous Object (Makeup menu); 25-11
 - with Select command (Makeup menu); 25-2
- selecting the parent of,
 - with Parent Object (Makeup menu); 25-10
- system,
 - term definition and characteristics; 17-6
- types,
 - auxiliary, characteristics; 17-10
 - effect of Copy command on; 20-3
 - effect of Cut command on; 20-2
 - effect of Paste command on; 20-5
 - system, characteristics; 17-10
 - term definition and characteristics; 17-6
- unlocking; 24-4
- visible,
 - selecting, with Select command (Makeup); 25-2

observability,

- controlling for,
 - pages; 24-13
 - substitution transitions; 24-12
- mode attribute characteristics; 17-16

obtaining,

- multisets,
 - random values from; 37-7
 - the number of elements in; 37-7

occlusion order,

- term definition; 3-19, 17-1

occur,

- term definition; 5-2

Occur button (Bind dialog, Sim menu),

- invoking without displaying the dialog; 22-8

occurrence,

- rule,
 - setting, with Occurrence Set Options command (Set menu); 24-30
- what happens when a transition occurs; 5-12

Occurrence Graph Analyzer,

- reporting names not acceptable to,
 - with the Syntax Options command (Set menu); 24-20

Occurrence Set (Sim menu),

- reference description; 22-9

Occurrence Set Options (Set menu),

- examining and changing occurrence set parameters with; 8-20
- reference description; 24-28

occurrence sets,

- See Also* algorithms; bindings;
- adding to; 22-9
- calculation of,
 - specifying, with Occurrence Set Options command (Set menu); 24-28
- clearing; 22-9
- constructing; 8-18
 - for SalesNet model execution with conflict; 8-12
- controlling the presence of,
 - pages in; 24-13
 - substitution transitions in; 24-12
- defining,
 - with a single binding; 22-8
- deleting; 22-9
- editing area consistency; 22-4
- executing,
 - algorithm for; 8-14
 - the elements in; 8-13
- listing of bindings included in; 22-5
- parameters,
 - examining and changing; 8-20
- retaining; 22-8
- review of; 8-18
- term definition; 8-11, 8-18

octal code string operation (ord),

- reference description; 32-8

of_id' (membership) function,

- syntax and characteristics; 32-27

Omit Page Borders component (Page Setup command),

- reference description; 19-7

omitting,

- See Also* deleting;
- time stamp; 12-8

Open (File menu),

- reference description; 19-2

Open Page (Page menu),

moving nodes between pages with; 4-15
 reference description; 26-2
 selecting a page with; 6-4

opening,

diagram; 1-4
 pages,
 with Open Page (Page menu); 26-2

operators and operations,

See Also expressions;

^^ (concatenate list); 30-4, 32-14
 ^ (concatenation) string; 32-8
 ~ (negation); 32-6
 * (scalar multiplication); 32-6, 37-4, 41-2, 41-7
 + (addition); 32-6, 37-3, 41-1, 41-7
 - (subtraction); 32-6, 37-4, 41-2, 41-7
 / (division); 32-6
 :: (prepend) list; 32-14
 < (less than) string; 32-8
 << (multiset less than); 37-6, 41-3
 <<= (multiset less than or equal); 37-6, 41-3
 <= (less than or equal) string; 32-8
 <> (not equal) string; 32-8
 <><> (multiset not equal); 37-5, 41-2
 = (equal) string; 32-8
 == (multiset equal); 37-5, 41-2
 > (greater than) string; 32-8
 >= (greater than or equal) string; 32-8
 >> (multiset greater than); 37-6, 41-3
 >>= (multiset greater than or equal); 37-6, 41-3
 @ (time append); 41-5
 @ (time stamp); 30-4, 39-4
 \ (backslash); 32-8
 ` (backquote) multiset creation operator; 37-2, 41-3

abs; 32-6

absolute value (abs); 32-6

addition (+); 32-6, 37-3, 41-7
 reference description; 41-1

arctan (arctangent); 32-6

arctangent (arctan); 32-6

backquote (`) multiset creation; 2-8, 37-2

backslash (\); 32-8

binary,

 association rule; 36-5

boolean,

 used in guards; 5-8

chr (character-octal code) string; 32-8

comparison,

 used in guards; 5-8

concatenate string (^); 32-8

concatenate list (^^); 32-14

operators and operations (cont'd),

cos (cosine); 32-6

cosine (cos); 32-6

division (/); 32-6

equal string (=); 32-8

equal multiset (==); 37-5
 reference description; 41-2

exp (exponential); 32-6

exponential (exp); 32-6

expressions; 36-3

 precedence; 36-4

function type,

 association rule; 36-5

greater than multiset (>>); 37-6

greater than string (>); 32-8

greater than or equal multiset (>>=); 37-6
 reference description; 41-3

greater than or equal string (>=); 32-8

less than multiset (<<); 37-6
 reference description; 41-3

less than string (<); 32-8

less than or equal multiset (<<=); 37-6
 reference description; 41-3

less than or equal string (<=); 32-8

list concatenation; 30-4

ln (natural logarithm); 32-6

mkst_col'; 32-8, 32-19

mkst_ms'; 32-8, 32-20

multiplication (*); 32-6

 multiplying multisets; 37-4

multiset,

 addition (+); 41-1

 creation (^); 37-2, 41-3

 creation (^), reference description; 41-3

 creation (^), syntax and use; 37-2

 equality (==); 41-2

 greater than (>>); 41-3

 greater than or equal (>>=); 41-3

 less than (<<); 41-3

 less than or equal (<<=); 41-3

 not equal (<><>); 37-5, 41-2

 scalar multiplication (*); 41-2

 subtraction (-); 41-2

 time append (@); 41-5

natural logarithm (ln); 32-6

negation (~); 32-6

not equal string (<>); 32-8

octal code string (ord); 32-8

operation; 41-1

ord (octal code) string; 32-8

precedence and association (table); 36-5

operators and operations (cont'd),

- predefined,
 - reference descriptions; 41-1
- prepend (::) list; 32-14
- scalar multiplication (*); 37-4, 41-2, 41-7
- sin (sine); 32-6
- sine (sin); 32-6
- size string; 32-8
- sqrt (square root); 32-6
- square root (sqrt); 32-6
- subtraction (-); 32-6, 37-4, 41-7
 - reference description; 41-2
- tan (tangent); 32-6
- tangent (tan); 32-6
- term definition; 41-1
- time append (@); 41-5

option key,

- navigating in text mode with; 17-3

options,

- model,
 - characteristics and commands that access; 17-17
- term definition; 17-15, 17-17

ord (octal code) string operation,

- reference description; 32-8

ord' (ordinal number) function,

- syntax and characteristics; 32-22

order occlusion,

- for connectors; 17-1
- for regions; 17-1
- on a page; 17-1
- term definition; 17-1

ordering,

- colorsets; 32-28

ordinal number (ord') function,

- syntax and characteristics; 32-22

or else (boolean disjunction),

- expression specifier use; 36-2

or else (boolean OR),

- boolean operator used in guards; 5-8

origin,

- line charts,
 - specifying; 21-16

original parent object,

- term definition and characteristics; 17-5

output,

- keyword,
 - code segment output clause identified by; 40-18
 - term definition; 13-2
- multiset,
 - term definition; 5-14

output (cont'd),

- pattern,
 - code segment, term definition and characteristics; 13-2
- place,
 - term definition; 2-13
- port,
 - characteristics; 21-24
- token regions,
 - characteristics; 17-11
 - removing; 21-28

output arcs,

- binding CPN variables on; 13-1
- inscriptions,
 - appending time delays to; 40-6
 - characteristics as CP net component; 2-1
 - conditional; 40-6
 - delay expressions on; 12-8
 - evaluating; 5-14
 - examples; 40-13
 - free variables on, uses and cautions; 40-5
 - function application use and restrictions; 40-6
 - functions permitted with, with_code; 41-6
 - functions permitted with, with_time; 41-6
 - if/then/else use in; 38-2
 - term definition; 2-14
 - timing considerations; 39-2
- term definition; 2-13

Output Form component (Page Setup command),

- reference description; 19-6

output_col function,

- reference description; 41-8

output_ms function,

- reference description; 41-8
- syntax and characteristics; 37-9

output_tms function,

- reference description; 41-8

overflow,

- line charts,
 - specifying methods for handling; 21-17

overloading,

- CPN ML feature; 30-1

P

Page Attributes (Set menu),

- naming pages with; 4-14
- reference description; 24-10

page kind (page attribute),

- characteristics; 17-15

Page menu,

- See Also* menus;
- Blowup command; 26-2
- Cleanup command; 26-3
- Close Page command; 26-2
- commands reference (chapter); 26-1
- New Page command; 26-2
 - creating a page for global declarations with; 4-14
- Open Page command; 26-2
 - moving nodes between pages with; 4-15
 - selecting a page with; 6-4
- Redraw Hierarchy; 26-3
- Redraw Hierarchy command,
 - redrawing the hierarchy page with; 11-13
- Reduce command; 26-3
- Scroll command; 26-2

Page Preview (File menu),

- reference description; 19-6

Page Setup (File menu),

- reference description; 19-6

pages,

- attributes,
 - changing; 24-10
 - changing system and/or diagram defaults; 24-11
 - term definition and components; 17-15
- borders,
 - changing the size; 24-11
 - rendering visible or invisible; 24-11
 - visibility, characteristics; 17-15
- characteristics and components; 1-5, 17-5
- cleaning up,
 - with Cleanup (Page menu); 26-3
- closing,
 - with Close Page (Page menu); 26-2
- connectors,
 - deletion requirements; 20-6
- creating,
 - with New Page (Page menu); 4-14, 26-2
- enlarging,
 - with Blowup (Page menu); 26-2
- fusion place,
 - term definition; 10-10

pages (cont'd),

- fusion sets,
 - creating; 10-9
 - term definition; 10-9
- fusion subsets,
 - term definition; 10-10
- hierarchy,
 - changing the layout and redrawing options; 24-14
 - for a hierarchical CP net diagram; 11-2
 - moving detail to a subpage of with Move to
 - Subpage command (CPN menu); 21-17
 - renaming pages from; 4-15
- instances,
 - creating; 10-11
 - displaying name of; 22-12
 - specifying the number of; 24-13
 - switching among; 22-12
 - term definition; 10-11
- master,
 - creating; 24-10
- mode,
 - changing; 24-13
- moving,
 - nodes and groups to; 25-7
 - nodes to a different; 4-15
- names,
 - duplicate, reporting with the Syntax Options
 - command (Set menu); 24-19
 - ML-illegal, reporting with the Syntax Options
 - command (Set menu); 24-19
 - term definition; 17-20
- naming,
 - with Page Attributes (Set menu); 4-14
- navigating,
 - among; 1-6
 - to, from a hierarchy page node; 17-3
- nodes,
 - characteristics; 1-6, 17-10
 - deletion requirements; 20-6
 - navigating to a page from; 17-3
- numbers,
 - specifying pages to print with; 1-6
 - term definition; 17-20
- occlusion order on; 17-1
- occurrence set contribution,
 - specifying, with Occurrence Set Options command
 - (Set menu); 24-29
- opening,
 - with Open Page (Page menu); 26-2
- overlay options,
 - changing; 24-15

pages (cont'd),

- Page Menu commands reference,
(chapter); 26-1
- page mode key region,
 - term definition; 6-5
- page mode region,
 - term definition; 6-5
- palette,
 - changing overlay options; 24-15
 - creating; 24-10
 - using; 24-11
- prime,
 - designating; 6-3
 - specifying; 24-13
- redrawing diagrams,
 - with Cleanup (Page menu); 26-3
- reference variables,
 - syntax and characteristics; 35-3
- relationship in a hierarchy; 11-20
- renaming,
 - from the hierarchy page; 4-15
- scrolling,
 - with Scroll (Page menu); 26-2
- specifying occurrence set contribution; 24-30
- size,
 - characteristics; 17-15
 - enlarging, with Blowup (Page menu); 26-2
 - reducing, with Reduce (Page menu); 26-3
- tags,
 - deletion requirements; 20-6
- term definition; 17-5

palette pages,

- creating; 24-10
- overlay options,
 - changing; 24-15
- using; 24-11

Parent Object (Makeup menu),

- navigating with; 17-4
- reference description; 25-10

parentheses (()),

- guards use of; 5-9

parents,

- See Also* regions;
- deleting; 3-17
- locking regions relative to,
 - with Size component (Region Attributes command-Set menu); 24-9
- moving; 3-17
- operations performed on auxiliary regions effect on;
23-10

parents (cont'd),

- selecting,
 - with Parent Object (Makeup menu); 25-10
- term definition; 3-16
- and characteristics; 17-5
- unlocking regions relative to,
 - with Size component (Region Attributes command-Set menu); 24-9

parts (charts),

- term definition; 15-2

Paste (Edit menu),

- reference description; 20-5

pasting,

- bar charts; 15-13
- history charts; 15-17
- line charts; 15-31

patterns,

- characteristics and use; 36-6
- lists; 40-11
- records,
 - input arc inscription example; 40-11
- tuples,
 - input arc inscription example; 40-10
 - term definition and characteristics; 2-6

performing,

- See Also* executing;
- syntax check; 6-1

perimeter transitions,

- creating when moving detail to a subpage,
 - with Move to Subpage command (CPN menu);
21-17
- term definition; 21-17

Petri nets,

- See* CP nets; Design/CPN;

Place (CPN menu),

- reference description; 21-2

places,

- See Also* colorsets; CPN objects, classes of; ellipses;
- initial markings;
- adding to fusion sets,
 - with Fusion Place command (CPN menu); 21-22
- changing size and position; 21-2
- characteristics,
 - and regions; 17-8
 - and term definition; 2-10
 - as CP net component; 2-1
- colorset region,
 - syntax and characteristics; 40-3
- creating,
 - multiple; 21-2
 - with Place (CPN menu); 4-7

places (cont'd),

- current marking simulation regions,
 - characteristics; 6-14
- fusion,
 - (chapter); 10-1
 - changing the type of, with Fusion Place command (CPN menu); 21-21
 - creating, with Fusion Place command (CPN menu); 21-20
 - global, physical appearance; 10-5
 - global, term definition; 10-5
 - instance, changing the type of, with Fusion Place command (CPN menu); 21-21
 - instance, renaming, with Fusion Place command (CPN menu); 21-21
 - instance, term definition; 10-13
 - multiple pages; 10-8
 - page, changing the type of, with Fusion Place command (CPN menu); 21-21
 - page, renaming, with Fusion Place command (CPN menu); 21-21
 - page, term definition; 10-10
 - renaming, with Fusion Place command (CPN menu); 21-21
 - single page; 10-3
 - term definition; 10-1
 - turning non-fusion places; 21-23
- initial marking region,
 - syntax and characteristics; 40-3
- input,
 - term definition; 2-13
- markings,
 - changing, with Change Marking command (Sim menu); 22-11
 - term definition and characteristics; 2-11
- moving between fusion sets; 21-23
- name region,
 - syntax and characteristics; 40-2
- names,
 - changing; 17-21
 - creating from auxiliary regions; 23-11
 - duplicate, reporting with the Syntax Options command (Set menu); 24-18
 - entering; 17-21
 - missing, reporting with the Syntax Options command (Set menu); 24-18
 - ML-illegal, reporting with the Syntax Options command (Set menu); 24-19
 - term definition; 17-20
- naming,
 - with CPN Region (CPN menu); 4-8

places (cont'd),

- output,
 - term definition; 2-13
- place creation mode,
 - term definition; 4-7
- place tool,
 - term definition and illustration; 4-7
- port,
 - assigning with Port Assignment command (CPN menu); 21-24
 - creating when moving detail to a subpage with Move to Subpage command (CPN menu); 21-17
 - creating with Port Place command (CPN menu); 21-23
 - deletion requirements; 20-6
 - effect of Replace by Subpage command (CPN menu) on; 21-19
 - general, characteristics; 21-24
 - term definition; 21-18
- regions,
 - creating; 21-5
 - creating from auxiliary regions; 23-11
- removing from fusion sets; 10-7
- socket,
 - effect of Replace by Subpage command (CPN menu) on; 21-19
 - term definition; 21-18
 - with Move to Subpage command (CPN menu) creating when moving detail to a subpage; 21-17
- subtracting from fusion sets,
 - with Fusion Place command (CPN menu); 21-22
- terminating creation mode; 21-2
- text use in; 21-2

pointer tool,

- characteristics and illustration; 3-16

pointer variables,

- See* reference variables;

polar coordinates,

- aligning nodes relative to,
 - with Position (Align menu); 29-3

Polygon (Aux menu),

- reference description; 23-5

polygons,

- creating, modifying, and moving; 23-5
- shape attributes,
 - changing; 24-7

popup regions,

- term definition; 6-14
- and characteristics; 17-11

Port Assignment (CPN menu),

- manually assigning ports to sockets with; 11-27

Port Assignment (CPN menu) (cont'd),

reference description; 21-24

Port Place (CPN menu),

reference description; 21-23

ports,

assignments,

impact on the results of the Change Marking
command (Sim menu); 22-11

effect of duplication on; 25-6

input,

characteristics; 21-24

input/output,

characteristics; 21-24

manually assigning to sockets; 11-25

navigating from a subpage to a superpage by double-
clicking on; 11-5

number,

specifying; 24-32

output,

characteristics; 21-24

places,

assigning; 21-24

creating; 21-23

creating when moving detail to a subpage; 21-17

deleting assignments; 21-25

deletion requirements; 20-6

effect of moving nodes between pages on; 25-8

effect of Replace by Subpage command (CPN
menu) on; 21-19

general characteristics; 21-24

navigating to a superpage from; 17-3

term definition; 21-18

types of; 21-24

port key region,

term definition; 11-4

regions,

CPN Region command restricted from creating;
21-5

term definition; 11-4

socket relationship to; 11-4

term definition; 11-1

position,

nodes,

changing; 21-8

places,

changing; 21-2

regions,

changing; 24-9

transitions,

changing; 21-3

Position (Align menu),

reference description; 29-3

power commands,

keys and shortcuts,

(chapter); A-1

precedence,

operators,

expression evaluation; 36-4

preferences,

characteristics and components; 17-15

prepend (::) list operator,

reference description; 32-14

preserving,

See aligning;

Previous Object (Makeup menu),

navigating with; 17-4

reference description; 25-11

prime,

term definition; 17-16

prime pages,

See Also occurrence sets; simulation;

characteristics; 17-16

designating; 6-3

specifying; 24-13

term definition; 6-3

Print (File menu),

reference description; 19-7

**Print Hierarchy Page component (Page Setup
command),**

reference description; 19-7

printing,

diagram; 1-6

overlapping objects,

specifying the; 24-4

page borders,

rendering visible or invisible; 24-11

products,

See tuples;

projecting,

a line of nodes,

with Projection (Align menu); 29-5

Projection (Align menu),

reference description; 29-5

proposed occurrence sets (mode attribute),

characteristics; 17-16

Q

Quit (File menu),
 reference description; 19-14

quitting,
See Also entering;
 Design/CPN; 1-7

R

random values,
 obtaining from a multiset,
 with random function; 41-4

ran' (random value) function,
 arc inscriptions not permitted to use; 40-5
 guards not permitted to use; 40-16
 reference description; 41-4
 syntax and characteristics; 32-18, 37-7
 time region use; 40-15
 union colorset selector use; 32-16

random number generation,
 setting seed value for, with Occurrence Set Options
 command (Set menu); 24-31

reading,
See Also colorsets;
 colorsets,
 with input_col function; 41-8

multisets; 37-9
 timed, with input_tms function; 41-8
 with input_ms function; 41-5, 41-8

real numbers,
See Also colorsets;
 colorsets,
 ordering of; 32-28
 syntax and characteristics; 32-5
 testing for membership in; 32-26

functions,
 standard ML, list of supported; 41-10

operations predefined for; 32-6

value identifiers,
 declaring; 33-2

realtime,
 functions,
 standard ML, list of supported; 41-12

reattaching,
 connectors; 23-2

rebinding,
See bindings; CPN variables;

reconstructing,
 groups; 3-21

recording,
 results of simulation,
 adding information, with write_report function;
 6-11
 specification with General Simulation Options
 (Set menu); 6-9

records,
See Also colorsets;
 colorsets,
 constructing multiset for; 32-25
 ordering of; 32-28
 syntax and characteristics; 32-11

expression constructors; 36-3

multiplying multisets of; 37-5, 41-2

patterns,
 input arc inscription example; 40-11

selector functions; 32-12

specifying in expressions; 36-2

rectangles,
See Also transitions;
 adding text to; 3-9
 creating; 3-6
 a series of; 3-9
 adding text while; 3-10
 deleting; 3-8
 moving; 3-8
 during creation; 3-8
 preserving the aspect ratio during size change; 3-10

rectangle creation mode,
 characteristics; 3-6

rectangle tool,
 characteristics and illustration; 3-6

reshaping; 3-7

redefining,
 bar chart attributes; 15-7
 history charts; 15-16
 line charts; 15-23

Redo (Edit menu),
 reference description; 20-2

Redraw Hierarchy (Page menu),
 redrawing the hierarchy page with; 11-13
 reference description; 26-3

redrawing,
 hierarchy page,
 with Redraw Hierarchy (Page menu); 26-3

pages,
 with Cleanup (Page menu); 26-3

Reduce (Page menu),
See Also Blowup (Page menu);
 reference description; 26-3

reducing,

- pages,
 - with Reduce (Page menu); 26-3

reference,

- functions,
 - standard ML, list of supported; 41-12
- summaries,
 - (chapter); 17-1

reference variables,

- (chapter); 35-1
- arc inscriptions not permitted to use; 40-5
- CPN,
 - global, syntax and characteristics; 35-3
 - instance, syntax and characteristics; 35-3
 - page, syntax and characteristics; 35-3
- CPN ML extension; 30-3
- expression use and restrictions; 36-2
- restrictions on use; 35-1
- val-defined; 35-1
 - syntax and characteristics; 35-2

Region Attributes (Set menu),

- reference description; 24-8

regions,

- See Also* CPN objects, classes of; nodes;
- arc inscription,
 - term definition; 2-14
- arc inscription region,
 - syntax and characteristics; 40-4
- arcs; 17-9
- attributes,
 - changing; 24-8
 - components; 17-15
- auxiliary,
 - creating; 23-10
 - creating auxiliary nodes from; 23-11
 - creating code segments from; 23-11
 - creating CPN regions from; 23-11
 - creating from auxiliary regions; 23-11
 - creating from CPN regions; 23-13
 - creating guards from; 23-11
 - creating initial markings from; 23-11
 - creating log regions from; 23-11
 - creating place names from; 23-11
 - creating transition names from; 23-11
 - effect of Paste command on; 20-5
 - term definition and characteristics; 17-7
- binding,
 - characteristics; 17-11
- border,
 - deletion requirements; 20-6

regions (cont'd),

- changing,
 - with Change Shape command (Makeup menu); 25-6
- charts; 17-9
 - bar, specifying; 21-11
 - history, specifying; 21-11
- code,
 - creating; 21-6
 - deletion requirements; 20-6
- color,
 - changing; 24-9
- colorsets,
 - characteristics; 40-3
 - creating with CPN Region command; 21-5
 - creation mode, term definition; 4-9
- converting into nodes; 3-17
- copying; 4-11
- CPN,
 - characteristics and relationships with parent objects; 17-10
 - creating auxiliary regions from; 23-13
 - creating from auxiliary regions; 23-11
 - creating multiple; 21-6
 - effect of Paste command on; 20-5
 - place regions, creating; 21-5
 - terminating creation mode; 21-7
- creating; 3-16
 - a group of; 27-2
- current marking,
 - characteristics; 17-10
 - removing with Remove Sim Regions command (CPN menu); 21-28
- displaying,
 - with Show Regions (Makeup menu); 25-9
- duplicating,
 - with Duplicate Node command (Makeup menu); 25-6
- editing; 3-17
- editor,
 - changing the visibility of; 17-3
- editor key/popup,
 - list; 17-12
- effect of,
 - Child Object (Makeup menu) on; 25-11
 - Copy command on; 20-4
 - Cut command on; 20-2
 - merging nodes on; 25-8
 - moving nodes between pages on; 25-7
- endpoint,
 - creating; 17-14
 - term definition and characteristics; 17-11, 17-13

regions (cont'd),

- feedback,
 - removing with Remove Sim Regions command (CPN menu); 21-28
- fusion,
 - CPN Region command restricted from creating; 21-5
 - deletion requirements; 20-6
 - key region, term definition and illustration; 10-6
 - term definition; 10-6
- guard,
 - creating; 21-6
- hidden,
 - selecting with arrow keys; 25-9
- hiding,
 - with Hide Regions (Makeup menu); 25-9
- hierarchy,
 - CPN Region command restricted from creating; 21-5
 - creating when moving detail to a subpage with Move to Subpage command (CPN menu); 21-18
 - deletion requirements; 20-6
 - key region, term definition; 11-7
 - term definition; 11-8
- initial marking,
 - creating with CPN Region command; 21-5
 - syntax and characteristics; 40-3
 - term definition; 2-12
- input token,
 - characteristics; 17-11
 - removing with Remove Sim Regions command (CPN menu); 21-28
- inscription,
 - See* inscriptions; specific names;
- key,
 - editor, creating and deleting; 17-13
 - simulator, creating and deleting; 17-13
 - term definition and characteristics; 17-7, 17-11
- layering order for; 17-1
- line charts; 21-15
- locking relative to the parent,
 - with Size component (Region Attributes command-Set menu); 24-9
- log,
 - characteristics and use; 13-3
 - creating; 21-6
- mode,
 - characteristics; 17-10
 - deletion requirements; 20-6
- moving; 11-9
 - the parent of; 3-17

regions (cont'd),

- name,
 - creating; 21-6
- output token,
 - characteristics; 17-11
 - removing with Remove Sim Regions command (CPN menu); 21-28
- place,
 - creating; 21-5
 - name, syntax and characteristics; 40-2
- places and; 17-8
- popup,
 - changing; 24-9
 - term definition and characteristics; 17-11
- port,
 - CPN Region command restricted from creating; 21-5
 - key, term definition; 11-4
 - term definition; 11-4
- position,
 - changing; 24-9
- region tool,
 - term definition and illustration; 4-6
- resizing,
 - without moving the center, with Adjust command (Makeup menu); 25-4
- selecting; 11-9
- shapes possible to; 24-5
- simulation,
 - changing the visibility of; 17-3
 - characteristics and purpose; 6-13
 - key/popup, list; 17-12
 - removing with Remove Sim Regions command (CPN menu); 6-18, 21-28
- size,
 - changing; 24-9
- substitution tag,
 - term definition; 11-2
- system,
 - characteristics; 17-10
- term definition and characteristics; 3-16, 17-5
- time,
 - assigning time stamps with CPN Region (CPN menu); 12-10
 - creating; 21-6
 - creating from auxiliary regions; 23-11
 - term definition; 12-7
- transitions; 17-8
 - creating; 21-6
 - feedback, characteristics; 17-11
 - reference description; 40-14
 - time region; 39-3

regions (cont'd),

- unlocking relative to the parent,
 - with Size component (Region Attributes command-Set menu); 24-9

Regroup (Group menu),

- reconstructing groups with; 3-21

Regular Polygon (Aux menu),

- reference description; 23-7

relational,

- expression operators; 36-3

Remove Sim Regions (CPN menu),

- reference description; 21-28
- removing simulation regions with; 6-19

removing,

- See Also* creating;
- simulation regions,
 - with Remove Sim Regions command (CPN menu); 6-18

renaming,

- fusion places,
 - with Fusion Place command (CPN menu); 21-21
- instance fusion places,
 - with Fusion Place command (CPN menu); 21-21
- page fusion places,
 - with Fusion Place command (CPN menu); 21-21
- pages,
 - from the hierarchy page; 4-15

rendering,

- objects invisible; 24-5

Replace by Subpage (CPN menu),

- reference description; 21-19

replacing,

- substitution transition with subpage,
 - using the Replace by Subpage command (CPN menu); 21-19

text,

- in a group, with Find (Text menu); 28-4
- with Find (Text menu); 28-3

reports,

- simulation,
 - clearing, with Save Report command (Sim menu); 22-13
 - customizing, with the Simulation Code Options command (Set menu); 24-26
 - naming, with Save Report command (Sim menu); 22-13
 - saving, with Save Report command (Sim menu); 22-13
 - specifying, with the Simulation Code Options command (Set menu); 24-25
- writing,
 - with write_report function; 41-6

repositioning,

- groups,
 - with Align menu commands; 29-2
- nodes,
 - with Align menu commands; 29-1
- objects,
 - with Displace command (Makeup menu); 25-4
 - with Drag command (Makeup menu); 25-2

representation,

- concurrency; 8-2
- conflict; 8-5
- multisets,
 - internal; 37-9
- state,
 - transition use as; 8-3
- time,
 - simulated (chapter); 12-1

rerouting,

- See Also* aligning;
- arcs; 11-8

reserved words,

- CPN ML; 31-2

resetting,

- drawing environment; 3-5

reshaping,

- See Also* aligning; creating;
- rectangles; 3-7

resizing,

- labels,
 - without moving the center, with Adjust command (Makeup menu); 25-4
- nodes,
 - with Change Shape command (Makeup menu); 25-5
- objects,
 - with Change Shape command (Makeup menu); 25-5
 - without moving the center, with Adjust command (Makeup menu); 25-4
- regions,
 - without moving the center, with Adjust command (Makeup menu); 25-4

ResmodSubtrans model,

- hierarchy page; 11-2
- subpage; 11-3
- superpage; 11-3

Resource Use Model,

- data; 16-2
- description; 10-1
- executing; 10-3
- graphics and global declarations; 16-3
- improvement suggestions; 16-15

Resource Use Model (cont'd),

introduction and overview; 16-1
running the model; 16-4

resources,

allocation of; 8-1
competition for,
as concurrency problem; 8-1

restarting,

simulation after breakpoint or interruption by Stop
command; 22-10

restoring,

drawing environment; 3-5

Reswitch (Sim menu),

reference description; 22-13
reswitching SalesNet after changing in the simulator;
8-10

reswitching,

after changing a CP net in the simulator; 8-10
diagrams,
with Reswitch command (Sim menu); 22-13

retaining,

occurrence set; 22-8

Revert (File menu),

reference description; 19-5

Right to Left (Align menu),

reference description; 29-6

Right to Right (Align menu),

reference description; 29-6

rot' (rotation) function,

syntax and characteristics; 32-24

rotation (rot') function,

syntax and characteristics; 32-24

rounded,

boxes,
changing shape attributes; 24-7
creation and manipulation; 23-4
polygons,
creation and manipulation; 23-7

Rounded Box (Aux menu),

reference description; 23-4

routing,

See Also aligning;
connectors; 3-14
automatic; 3-15

rows,

aligning nodes in,
See Horizontal (Align menu); Vertical (Align
menu);

running,

See executing;

runtime environment,

ML (chapter); 41-1

S

SalesNet model,

changing in the simulator; 8-9
concurrent execution of; 8-7
executing with conflict; 8-12

satisfy,

term definition; 5-10

Save (File menu),

reference description; 19-3

Save As (File menu),

reference description; 19-5

Save Report (Sim menu),

reference description; 22-13

Save State (File menu),

reference description; 19-11
saving execution states with; 5-15

Save Subdiagram (File menu),

reference description; 19-7

Save Text (File menu),

reference description; 19-9

saved states,

entering the simulator with; 6-12
loading with Open command (File menu); 19-12
opening; 19-2
term definition; 5-15

saving,

See Also loading;
bar chart copy; 21-12
execution states,
with Save State (File menu); 5-15
history chart copy; 21-12
line charts; 21-15
simulation reports,
with Save Report command (Sim menu); 22-13

scalar multiplication operator (*),

function scalar multiplication; 41-7
scalar multiplication of multisets with; 37-4, 41-2

scope,

local,
declaring local variables within a function; 38-1
term definition; 34-1

scoping,

static,
CPN ML feature; 30-2

Scroll (Page menu),

reference description; 26-2

scrolling,

autoscrolling characteristics; 3-4

scrolling (cont'd),

- display options,
 - changing; 24-15
- pages,
 - with Scroll (Page menu); 26-2

searching for,

- text,
 - in a group, with Find (Text menu); 28-4
 - with Find (Text menu); 28-3

segments,

- term definition; 3-14

Select (Makeup menu),

- accessing difficult to select objects with; 17-4
- reference description; 25-2

Select All Connectors (Group menu),

- reference description; 27-2

Select All Nodes (Group menu),

- reference description; 27-2

Select All Regions (Group menu),

- reference description; 27-2

Select All Text (Text menu),

- reference description; 28-5

Select Fusion Set (Group menu),

- reference description; 27-3

Select Instance (Sim menu),

- reference description; 22-12

Select to (Text menu),

- reference description; 28-5

Select to Bracket (Text menu),

- reference description; 28-5

Select to Bracket command,

- specifying brackets and character counts used by,
 - with Text Options command (Set menu); 24-17

selectability (graphical attribute),

- changing; 17-17
- characteristics; 17-17

selectable,

- term definition; 17-17

selecting,

- all nodes,
 - with Select All Nodes (Group menu); 27-2
- child objects,
 - with Child Object (Makeup menu); 25-10
- fusion sets,
 - with Select Fusion Set (Group menu); 27-3
- graphical objects; 3-18
- hidden regions,
 - with arrow keys; 25-9
- objects,
 - the next object in the layering order, with Next Object (Makeup menu); 25-11

selecting (cont'd),

- objects (cont'd),
 - the previous object in the layering order, with Previous Object (Makeup menu); 25-11
 - with Select command (Makeup menu); 25-2
- parents,
 - with Parent Object (Makeup menu); 25-10
- regions; 11-9
- simulation type of,
 - with General Simulation Options (Set menu); 6-8
- text,
 - with Select All Text (Text menu); 28-5
 - with Select to (Text menu); 28-5
 - with Select to Bracket (Text menu); 28-5

selectors,

- boolean; 41-6
- expression operators; 36-3
- selector functions,
 - records; 32-12
- union colorsets,
 - specifying; 32-16

Set menu,

- See Also* menus;
- Chart Attributes command; 24-13
- commands reference (chapter); 24-1
- Copy Defaults command; 24-33
 - copying diagram defaults with; 4-3
- General Simulation Options,
 - selecting the type of simulation with; 6-8
- General Simulation Options command; 24-23
 - selecting the termination conditions with; 6-9
- Graphic Attributes command; 24-3
- Hierarchy Page Options command; 24-14
- Interaction Options command; 24-15
- Interactive Simulation Options command; 24-27
 - setting breakpoints with; 6-15
- Merge Options command; 24-16
- ML Configuration Options,
 - saving ML configuration options with; 4-3
- ML Configuration Options command; 24-32
- Mode Attributes command; 24-12
 - creating multiple page instances with; 10-11
 - designating prime pages with; 6-4
- Occurrence Set Options command; 24-28
 - examining and changing occurrence set parameters with; 8-20
- Page Attributes command; 24-10
 - naming pages with; 4-14
- Region Attributes command; 24-8
- Shape Attributes command; 24-5
- Simulation Code Options command; 24-20

Set menu (cont'd),

- Simulation Code Options command (cont'd),
generating code for different types of execution
with; 6-5
- Syntax Options command; 24-18
- Text Attributes command; 24-2
- Text Options command; 24-17
- Transition Feedback Options command; 24-31

sets,

- See* multisets;

setting,

- breakpoints,
all, with Option key plus Stop command (Sim
menu); 22-11
with Interactive Simulation Options command (Set
menu); 6-15, 24-27
- graphical environment; 4-2

shape,

- attributes,
changing; 24-5
components; 17-15
- changing,
with Change Shape command (Makeup menu);
25-5

Shape Attributes (Set menu),

- reference description; 24-5

shift key,

- navigating in text mode with; 17-3

Show Regions (Makeup menu),

- reference description; 25-9

showing,

- regions,
with Show Regions (Makeup menu); 25-9

side effects,

- arc inscriptions not permitted to have; 40-5
- CPN ML restrictions on the use of; 30-4

Sim menu,

- See Also* menus;
- Automatic Run command; 22-10
- Bind command; 22-2
- Change Marking command; 22-11
- characteristics and purpose; 6-12
- Clear Report command; 22-13
- commands reference (chapter); 22-1
- Continue command; 22-10
continuing execution after a breakpoint with;
6-16, 8-15
- Initial State command; 22-11
initializing SalesNet after changing in the
simulator; 8-10
initializing the CP net state with; 6-18
- Interactive Run command; 22-10

Sim menu (cont'd),

- Interactive Run command (cont'd),
constructing an occurrence set with; 8-12
executing CP nets with; 6-16
- Occurrence Set command; 22-9
- Reswitch command; 22-13
reswitching SalesNet after changing in the
simulator; 8-10
- Save Report command; 22-13
- Select Instance command; 22-12
- Start Step command; 22-9
- Stop command; 22-10
- Update Chart command; 22-13
updating bar charts with; 15-10
updating line charts with; 15-27

simple,

- See Also* colorsets - boolean, enumerated values,
indexed values, integers, real numbers, strings,
unit;
- colorsets,
characteristics; 32-2
- expressions,
term definition; 36-1

simplifying,

- See* aligning;

simulation,

- See Also* breakpoints;
- automatic,
initiating with Automatic Run command (Sim
menu); 22-10
- controlling the presence of,
pages during; 24-13
substitution transitions during; 24-12
- generating code for different types of,
with Simulation Code Options (Set menu); 6-5
- interactive,
initiating with Interactive Run command (Sim
menu); 22-10
- managing,
with write_report and stop_simulation functions;
41-6
- options,
specifying with with_time and with_code
functions; 41-6
testing for the with time option; 39-4
- recording results of,
adding information, with write_report function;
6-11
specification with General Simulation Options
(Set menu); 6-9
- regions,
characteristics and purpose; 6-13

simulation (cont'd),

- regions (cont'd),
 - current marking; 6-14
 - feedback; 6-14
 - removing, with Remove Sim Regions command (CPN menu); 6-18
 - term definition; 6-13
 - transition feedback; 6-14
- reports,
 - clearing, with Save Report command (Sim menu); 22-13
 - naming, with Save Report command (Sim menu); 22-13
 - saving, with Save Report command (Sim menu); 22-13
 - specifying, with the Simulation Code Options command (Set menu); 24-25
- restarting after breakpoint or interruption by Stop command; 22-10
- selecting the type of,
 - with General Simulation Options (Set menu); 6-8
- specifying,
 - parameters for; 24-23
 - prime pages; 24-13
 - time for; 12-4
- step,
 - starting; 22-9
- stopping,
 - specifying criteria for, with the Simulation Code Options command (Set menu); 24-24
 - with Stop command (Sim menu); 22-10
 - with stop_simulation function; 41-6
- term definition; 1-1
- termination,
 - specifying conditions for, with General Simulation Options (Set menu); 6-9
- type of,
 - specifying, with General Simulation Options command (Set menu); 24-23
- with and without code,
 - code segment characteristics, syntax, and use; 13-1
 - specifying, with General Simulation Options command (Set menu); 24-24
- with and without time; 12-14
 - mechanism; 12-3
 - specifying, with General Simulation Options command (Set menu); 24-24
- with instance fusion sets; 10-14
- writing a report on status of,
 - with write_report function; 41-6

Simulation Code Options (Set menu),

- generating code for different types of execution with; 6-5
- reference description; 24-20

simulator,

- changing a net in; 8-9
- code options,
 - specifying which to use, with the Simulation Code Options command (Set menu); 24-20
- entering; 6-11
 - with a saved state; 6-12
- executing CP nets with,
 - (chapter); 6-1
- execution algorithm; 8-11
 - illustrating with SalesNet model execution with conflict; 8-12
- key/popup regions,
 - creating and deleting; 17-13
 - list; 17-12
- leaving,
 - with Enter Editor command (File menu); 6-18
- options,
 - command that access; 17-17
- overview; 5-1
- regions,
 - changing the visibility of; 17-3
 - removing with Remove Sim Regions command (CPN menu); 21-28
- role in mapping inputs to outputs; 8-3

sin operation (sine),

- reference description; 32-6

sine operation (sin),

- reference description; 32-6

single quote ('),

- type variable use; 30-4

size,

- colorsets,
 - as classification dimension; 32-1
- multiset,
 - obtaining, with size function; 41-3
- nodes,
 - changing; 21-8
- page,
 - borders, changing; 24-11
 - characteristics of page attribute; 17-15
 - enlarging, with Blowup (Page menu); 26-2
 - reducing, with Reduce (Page menu); 26-3
- places,
 - changing; 21-2
- rectangle,
 - preserving the aspect ratio while changing; 3-10

-
- size (cont'd),**
 - regions,
 - changing; 24-9
 - size constant,
 - syntax and characteristics; 32-21
 - size function,
 - reference description; 41-3
 - syntax and characteristics; 37-7
 - size string operation,
 - reference description; 32-8
 - transitions,
 - changing; 21-3
 - small colorsets,**
 - See* colorsets, small;
 - socket places,**
 - creating when moving detail to a subpage,
 - with Move to Subpage command (CPN menu); 21-17
 - effect of Replace by Subpage command (CPN menu) on; 21-19
 - term definition; 21-18
 - sockets,**
 - deletion requirements; 20-6
 - manually assigning ports to; 11-25
 - port relationship to; 11-4
 - term definition; 11-1
 - spacing,**
 - nodes,
 - along a diagonal, *See* Vertical Spread (Align menu) and Horizontal Spread (Align menu);
 - equidistantly around a circle, with Circular Spread (Align menu); 29-5
 - vertically, with Vertical Spread (Align menu); 29-5
 - specifying,**
 - bar charts,
 - bar names; 21-11
 - grid lines; 21-11
 - title; 21-11
 - character count blocking used by Load Text and Select to Brackets commands,
 - with Text Options command (Set menu); 24-17
 - code generation for,
 - code segments; 24-20
 - simulation type; 24-21
 - timed simulation; 24-21
 - colorsets,
 - with CPN Region (CPN menu); 4-9
 - graphics updating during simulation,
 - with Interactive Simulation Options command (Set menu); 24-27
 - specifying (cont'd),**
 - history charts,
 - bar names; 21-11
 - grid lines; 21-11
 - title; 21-11
 - values; 15-16
 - initial marking,
 - with CPN Region (CPN menu); 4-9
 - input arc inscriptions,
 - with CPN variables; 5-7
 - line charts,
 - grid lines characteristics; 21-15
 - lines characteristics; 21-14
 - methods for handling overflow; 21-17
 - name; 21-14
 - origin; 21-16
 - regions; 21-15
 - values, the type and range; 21-16
 - ML configuration options,
 - with ML Configuration Options command (Set menu); 24-32
 - multisets; 2-8
 - number of pages instances; 24-13
 - occurrence set parameters,
 - with Occurrence Set Options command (Set menu); 24-28
 - options,
 - merging arcs; 24-16
 - merging connectors; 24-16
 - merging nodes; 24-16
 - merging text in CPN regions; 24-17
 - merging text in nodes; 24-16
 - overlapping object display and printing; 24-4
 - prime page; 6-3, 24-13
 - simulation parameters,
 - with the General Simulation Options command (Set menu); 24-23
 - text brackets used by Load Text and Select to Brackets commands,
 - with Text Options command (Set menu); 24-17
 - tokens,
 - exact values with constants; 5-4
 - multiple tokens with the different values; 5-6
 - multiple tokens with the same value; 5-5
 - with CPN variables; 5-7
 - spreading,**
 - See Also* aligning;
 - nodes,
 - along a diagonal, *See* Vertical Spread (Align menu) and Horizontal Spread (Align menu);
 - equidistantly around a circle, with Circular Spread (Align menu); 29-5

spreading (cont'd),

- nodes (cont'd),
 - horizontally, with Horizontal Spread (Align menu); 29-4
 - vertically, with Vertical Spread (Align menu); 29-5

sqrt operation (square root),

- reference description; 32-6

square root operation (sqrt),

- reference description; 32-6

standard defaults,

- term definition; 17-18

standard deviation,

- statistical variable,
 - obtaining, with SV'std; 14-4

Standard ML,

- See Also* CPN ML language;
- CPN ML,
 - extensions to; 30-2
 - restrictions and modifications; 30-3

Start ML (Aux menu),

- See Also* CPN ML language;
- reference description; 23-13

Start Step (Sim menu),

- reference description; 22-9

starting,

- See Also* entering; quitting; stopping;
- Design/CPN; 1-4
- simulation step; 22-9

states,

- CP net,
 - initializing with Initial State command (Sim menu); 6-18
- current,
 - term definition; 2-11
- execution,
 - loading, with Load State (File menu); 5-16
 - saving, with Save State (File menu); 5-15
- initial,
 - concurrent execution; 8-4
 - term definition; 2-11
- saved,
 - entering the simulator with; 6-12
 - starting execution with; 5-16
 - term definition; 5-15
- system,
 - generating an initial, with Initial State command (Sim menu); 22-11
- term definition and characteristics; 2-11
- time in relation to; 12-2
- transitions as representation of; 8-3

static scoping,

- CPN ML feature; 30-2

statistical variables,

- (chapter); 14-1
- accessing; 14-3
- charts and,
 - using (chapter); 16-1
- clearing; 14-5
- creating; 14-2
- initializing; 14-2
 - in Resource Use model; 16-5
- methodology for use; 14-1
- structure; 14-2
- updating; 14-3
 - in Resource Use model; 16-6

statistics,

- Resource Use model; 16-5

status bar,

- characteristics; 1-5

status bar information,

- error reporting from Syntax Check command (CPN menu); 21-27

steps,

- line chart updating period; 21-16
- number,
 - accessing with step function; 39-4
- recording information about,
 - specification with General Simulation Options (Set menu); 6-10
- reporting information about,
 - with the Simulation Code Options command (Set menu); 24-26
- simulation,
 - starting; 22-9
- step function,
 - reference description; 41-5
 - syntax and characteristics; 39-4
- term definition; 8-12

Stop (Sim menu),

- reference description; 22-10

stop criteria,

- simulation,
 - specifying conditions for, with General Simulation Options (Set menu); 6-9

Stop ML (Aux menu),

- See Also* CPN ML language;
- reference description; 23-13

stopping,

- simulation,
 - specifying criteria for, with the Simulation Code Options command (Set menu); 24-24
 - with Stop command (Sim menu); 22-10

-
- stopping (cont'd),**
 - simulation (cont'd),
 - with stop_simulation function; 41-6
 - stop_simulation function,**
 - reference description; 41-6
 - strings,**
 - See Also* colorsets;
 - colorsets,
 - ordering of; 32-28
 - syntax and characteristics; 32-6
 - term definition and characteristics; 2-4
 - testing for membership in; 32-26
 - comparison operators; 32-8
 - control character insertion; 32-8
 - escape sequences; 32-8
 - expression operators; 36-3
 - functions,
 - standard ML, list of supported; 41-11
 - length; 32-8
 - operations; 32-8
 - representing colorset value as; 32-19
 - with mkst_col' operation; 32-8
 - representing multiset value as; 32-8
 - selecting text to a specified,
 - with Select to (Text menu); 28-5
 - value identifiers,
 - declaring; 33-2
 - structure,**
 - statistical variables; 14-2
 - style sheet,**
 - creating; 24-10
 - submodel,**
 - term definition; 9-3, 11-1
 - subnet,**
 - term definition; 9-3, 11-2
 - subpages,**
 - creating; 11-5
 - with Move to Subpage command (CPN menu); 21-18
 - deleting; 11-23
 - references to; 11-24
 - designating net components to move to; 11-6
 - improving the appearance; 11-11
 - moving detail to,
 - with Move to Subpage command (CPN menu); 21-17
 - navigating,
 - from to the superpage; 11-5
 - to from substitution transition; 17-3
 - replacing substitution transition with,
 - using the Replace by Subpage command (CPN menu); 21-19
 - subpages (cont'd),**
 - ResmodSubtrans model; 11-3
 - term definition; 11-1
 - using more than one; 11-17
 - subroutine,**
 - subpages compared to; 11-21
 - subsets,**
 - See Also* colorsets;
 - colorsets,
 - ordering of; 32-28
 - syntax and characteristics; 32-15
 - testing for membership in; 32-26
 - subsetting,**
 - multiset(s); 2-9
 - substep,**
 - term definition; 8-12
 - substitution,**
 - connectors,
 - characteristics; 17-10
 - tag region,
 - term definition; 11-2
 - tags,
 - characteristics; 17-10
 - transitions,
 - (chapter); 11-1
 - changing the mode of; 24-12
 - characteristics; 9-3
 - creating; 11-5, 11-15
 - creating, when moving detail to a subpage; 21-18
 - creating, with Substitution Transition command (CPN menu); 21-19
 - effect of Child Object (Makeup menu) on; 25-10
 - effect of duplication on; 25-6
 - effect of moving nodes between pages on; 25-7
 - naming; 11-8
 - navigating to a subpage from; 17-3
 - replacing with subpage; 21-19
 - reversing the creation of; 11-21
 - specifying the location of; 11-6
 - substitution transition creation mode, term
 - definition; 11-6
 - term definition; 9-3, 11-1, 21-18
 - Substitution Transition (CPN menu),**
 - reference description; 21-19
 - subtracting,**
 - functions,
 - with subtraction operator (-); 41-7
 - multisets; 2-8, 37-4
 - subtraction operator reference description; 41-2
 - places from fusion sets,
 - with Fusion Place command (CPN menu); 21-22

subtraction operator (-),

- reference description; 32-6
- function subtraction; 41-7
- subtracting multisets; 37-4
 - reference description; 41-2

sum,

- statistical variable,
 - obtaining, with SV'sum; 14-5

sum of the squares,

- statistical variable,
 - obtaining, with SV'ss; 14-5

sum of the squares of deviation,

- statistical variable,
 - obtaining, with SV'ssd; 14-5

summaries,

- reference,
 - (chapter); 17-1

supernodes,

- changing mode attributes for; 24-13

superpage,

- improving the appearance; 11-8
- navigating,
 - to from a port place; 17-3
 - to from the subpage; 11-5
- ResmodSubtrans model; 11-3
- term definition; 11-1

SV'avrg function,

- reference and use; 14-3

SV'count function,

- reference and use; 14-3

SV'first function,

- reference and use; 14-4

SV'init function,

- reference and use; 14-5

SV'maximum function,

- reference and use; 14-4

SV'minimum function,

- reference and use; 14-4

SV'ss function,

- reference and use; 14-5

SV'ssd function,

- reference and use; 14-5

SV'std function,

- reference and use; 14-4

SV'sum function,

- reference and use; 14-5

SV'value function,

- reference and use; 14-4

SV'vari function,

- reference and use; 14-5

switching,

- among page instances,
 - with Select Instance command (Sim menu); 22-12
- simulation code options that affect the speed of;
 - 24-21

syntax,

- check,
 - compulsory restrictions, list and characteristics;
 - 17-22
 - performing; 6-1
- compulsory restrictions,
 - list; 17-22
- descriptions,
 - format; 30-4
- errors,
 - deciphering ambiguous messages; 7-5
 - handling (chapter); 7-1
 - locating; 7-2
 - missing colorset; 7-2
 - optional, specifying which to be reported; 24-18
 - undeclared CPN variables; 7-4
- guards; 5-8

Syntax Check (CPN menu),

- performing a syntax check with; 6-2
- reference description; 21-26
- specifying which errors reported,
 - with the Syntax Options command (Set menu);
 - 24-18

Syntax Options (CPN menu),

- selecting optional syntax restrictions with; 6-1

Syntax Options (Set menu),

- place name validation criteria; 40-2
- reference description; 24-18

system,

- connectors,
 - characteristics; 17-10
- defaults,
 - attributes, term definition; 4-2
 - changing; 17-19
 - changing graphic attributes; 24-3
 - changing shape attributes; 24-5
 - changing text attributes; 24-2
 - copying; 17-19
 - page attributes, changing; 24-11
 - setting diagram defaults with; 24-33
 - setting for ML configuration options; 24-33
 - term definition and characteristics; 4-2, 17-18
 - using; 17-19
- nodes,
 - characteristics; 17-10
- objects,
 - characteristics; 17-7

system (cont'd),
 objects (cont'd),
 term definition; 17-6
 types; 17-10
 regions,
 characteristics; 17-10
 state,
 generating an initial; 22-11

T

tab,
 inserting in strings; 32-8
tags,
 page,
 deletion requirements; 20-6
 substitution,
 characteristics; 17-10
tan operation (tangent),
 reference description; 32-6
tangent operation (tan),
 reference description; 32-6
templates,
 creating; 24-10
temporary,
 declaration nodes,
 creating with Declaration Node command (CPN
 menu); 21-7
 term definition and characteristics; 17-6
terminating,
 arc creation mode; 21-5
 CPN region creation mode; 21-7
 nodes declaration mode; 21-8
 place creation mode; 21-2
termination,
 simulation,
 specifying conditions for, with General
 Simulation Options (Set menu); 6-9
terminology,
 reference description; 1-1
testing,
 ML Evaluate command (Aux menu),
 use for testing new and changed code; 23-14
 optional syntax errors,
 specifying which to be reported, with the Syntax
 Options command (Set menu); 24-18
text,
 adding to,
 a rectangle; 3-9
 a rectangle, while creating it; 3-10

text (cont'd),
 adjusting a graphic object to fit,
 with Fit to Text command (Makeup menu); 25-5
 attributes,
 changing for selected objects; 24-2
 changing system and/or diagram defaults; 24-2
 brackets used by Load Text and Select to Brackets
 commands,
 specifying, with Text Options command (Set
 menu); 24-17
 character counts used by Load Text and Select to
 Brackets commands,
 specifying, with Text Options command (Set
 menu); 24-17
 CPN regions,
 specifying options for merging; 24-17
 editing; 3-13
 effect of,
 Copy command on; 20-4
 Cut command on; 20-3
 Paste command on; 20-6
 entering; 3-13
 into a global declaration node; 4-13
 finding,
 with Find (Text menu); 28-3
 impact of CPN Region command on; 21-7
 labels,
 creating, modifying, and moving; 23-9
 mode,
 characteristics; 3-3
 creating objects from; 3-18
 detecting when system is in both group mode and;
 28-2
 entering; 28-2
 impact on duplication; 25-6
 leaving; 28-2
 navigating in; 17-3
 term definition; 3-2
 moving,
 to beginning of current text object; 28-4
 with Drag command (Makeup menu); 25-2
 nodes,
 specifying options for merging; 24-16
 place use of; 21-2
 pointers,
 following to find the error location; 21-26
 locating an error with; 7-3
 moving to the node pointed to, with Child Object
 (Makeup menu); 25-11
 moving to the node pointed to, with Parent Object
 (Makeup menu); 25-10
 term definition; 7-3

text (cont'd),

- pointers (cont'd),
 - using as hypertext; 28-4
- replacing,
 - with Find (Text menu); 28-3
- searching for,
 - with Find (Text menu); 28-3
- selecting,
 - with Select All Text (Text menu); 28-5
 - with Select to (Text menu); 28-5
 - with Select to Bracket (Text menu); 28-5
- tool,
 - term definition and illustration; 3-3
- transition use of; 21-3

text attributes,

- components; 17-15

Text Attributes (Set menu),

- reference description; 24-2

Text menu,

- See Also* menus;
- commands reference (chapter); 28-1
- Enter Text Mode command; 28-2
- Find Beginning command; 28-4
- Find command; 28-3
- Find Next command; 28-4
- Leave Text Mode command; 28-2
- Select All Text command; 28-5
- Select to Bracket command; 28-5
- Select to command; 28-5
- Turn On Text command,
 - adding text to rectangles with; 3-9

Text Options (Set menu),

- reference description; 24-17

time,

- See Also* concurrency;
- (chapter); 39-1
- appending,
 - time lists to multisets; 39-4
 - time lists to multisets, with time append operator (@); 41-5
- delay,
 - transition time region, reference description; 40-15
- function; 41-5
 - output arc inscription use of; 39-2
 - syntax and characteristics; 39-4
 - time region use; 39-3, 40-15
- increasing the realism of simulation with; 12-15
- initial marking region (place),
 - time delay syntax and characteristics; 40-3
- input arc inscriptions,
 - overriding time stamps with @ignore; 40-5

time (cont'd),

- line chart,
 - updating period; 21-16
- multisets; 37-10
 - reading, with input_tms function; 41-8
 - writing, with output_tms function; 41-8
- output arc inscriptions; 12-8
 - appending time delays to; 40-6
- real,
 - term definition; 12-2
- regions,
 - assigning time stamps with CPN Region (CPN menu); 12-10
 - creating; 21-6
 - creating from auxiliary regions; 23-11
 - term definition; 12-7
- removing from a colorset; 39-1
- representation methods in CP nets; 12-2
- simulated,
 - (chapter); 12-1
 - characteristics; 12-2
 - impact on enablement; 12-3
 - term definition; 12-2
 - uses for; 12-3
- simulation with,
 - and without; 12-14
 - disabling, with the Simulation Code Options command (Set menu); 24-21
 - generating code for, with the Simulation Code Options command (Set menu); 24-21
 - specifying; 12-4
 - specifying, with General Simulation Options command (Set menu); 24-24
 - specifying, with the with_time function; 41-6
- stamps,
 - @ operator use for appending; 30-4
 - assigning, example; 12-10
 - colorset defaults; 39-1
 - giving to a token; 12-7
 - initial markings and; 12-9
 - multisets and; 12-10
 - omitting; 12-8
 - term definition and characteristics; 12-2, 39-1
- state in relation to; 12-2
- testing for the with time simulation option; 39-4
- time append operator (@); 41-5
- time region,
 - characteristics and use; 39-3
- timed colorsets,
 - See Also* colorsets;
 - declaration; 39-1
 - declaring; 12-6

- time (cont'd),**
 - timed colorsets (cont'd),
 - defaults; 39-1
 - term definition; 12-2
 - timed token,
 - term definition; 12-2
 - transition,
 - region reference description; 40-15
- title,**
 - bar charts,
 - specifying; 21-11
 - history charts,
 - specifying; 21-11
 - line charts,
 - specifying; 21-15
- tokens,**
 - See Also* multisets;
 - constraining,
 - multiple; 5-11
 - with more complex guards; 5-10
 - with simple guards; 5-9
 - giving a time stamp to; 12-7
 - specifying,
 - exact values with constants; 5-4
 - multiple tokens with the different values; 5-6
 - multiple tokens with the same value; 5-5
 - with CPN variables; 5-7
 - term definition; 2-3, 2-7, 37-1
 - timed,
 - delay expression syntax; 12-7
 - values,
 - constraining; 5-8
 - constraining with the use of guards; 5-8
- Top to Bottom (Align menu),**
 - reference description; 29-7
- Top to Top (Align menu),**
 - reference description; 29-7
- top-down development,**
 - hierarchical CP nets; 11-14
- Transition (CPN menu),**
 - reference description; 21-3
- transition creation mode,**
 - term definition; 4-5
- Transition Feedback Options (Set menu),**
 - reference description; 24-31
- transition names,**
 - changing; 17-21
 - entering; 17-21
 - term definition; 17-20
- transition tools,**
 - term definition and illustration; 4-5
- transitions,**
 - See Also* CPN objects, classes of; rectangles;
 - as state representation; 8-3
 - characteristics,
 - and regions; 17-8
 - and term definition; 2-12
 - as CP net component; 2-1
 - code segments,
 - examples; 40-19
 - reference description; 40-16
 - updating bar charts from; 15-10
 - updating line charts from; 15-27
 - concurrent,
 - firing; 8-3
 - creating,
 - multiple; 21-3
 - regions of; 21-6
 - with Transition (CPN menu); 4-4
 - enablement,
 - reference variables restricted from effects on; 35-1
 - explanation of the setting in the Occurrence Set
 - Options dialog; 8-20
 - factors controlling,
 - enablement; 5-2
 - occurrence; 5-2
 - graphical appearance when enabled,
 - specifying with Transition Feedback Options
 - command (Set menu); 24-31
 - guards,
 - syntax, use, and restrictions; 40-15
 - names,
 - duplicate, reporting with the Syntax Options
 - command (Set menu); 24-19
 - missing, reporting with the Syntax Options
 - command (Set menu); 24-19
 - ML-illegal, reporting with the Syntax Options
 - command (Set menu); 24-19
 - reference description; 40-14
 - naming,
 - with CPN Region (CPN menu); 4-6
 - occurrence of; 5-12
 - occurrence set contribution,
 - specifying, with Occurrence Set Options command
 - (Set menu); 24-30
 - perimeter,
 - creating when moving detail to a subpage with
 - Move to Subpage command (CPN menu); 21-17
 - term definition; 21-17
 - regions,
 - creating from auxiliary regions; 23-11
 - reference description and illustration; 40-14

transitions (cont'd),

- substitution,
 - (chapter); 11-1
 - changing the mode of; 24-12
 - creating, when moving detail to a subpage with
 - Move to Subpage command (CPN menu); 21-18
 - creating, with Substitution Transition command (CPN menu); 21-19
 - replacing, with subpage using the Replace by Subpage command (CPN menu); 21-19
 - term definition; 11-1, 21-18
- term definition; 17-5
- terminating creation mode; 21-3
- text use in; 21-3
- time regions,
 - characteristics, syntax, and use; 12-7, 39-3
 - reference description; 40-15
- transition feedback region,
 - term definition; 6-14
 - characteristics; 17-11

trigonometric functions,

- reference description; 32-6

troubleshooting,

- common problem symptoms and solutions,
 - (chapter); B-1
- memory problems,
 - problem symptoms and solutions; B-6
- ML,
 - configuration not specified, problem symptoms and solutions; B-2
 - interpreter cannot be started, problem symptoms and solutions; B-6
- settings file missing or obsolete,
 - problem symptoms and solutions; B-1

tuples,

- See Also* colorsets;
- colorsets,
 - constructing multiset for; 32-25
 - ordering of; 32-28
 - syntax and characteristics; 32-10
 - term definition; 2-5
- constructors,
 - characteristics and use; 36-8
 - term definition; 2-6
- expression constructors; 36-3
- formats in syntax description; 30-5
- multiplying multisets of; 37-5, 41-2
- patterns; 36-7
 - input arc inscription example; 40-10
 - term definition and characteristics; 2-6
- specifying in expressions; 36-2
- term definition; 2-5

tuples (cont'd),

- value,
 - declaring; 33-2
 - term definition; 2-5
 - with CPN variables,
 - input arc inscription example; 40-10
- Turn On Text (Text menu),**
 - adding text to rectangles with; 3-9
- type variables,**
 - term definition; 30-4
- typing,**
 - polymorphic,
 - CPN ML feature; 30-1
 - strong,
 - CPN ML feature; 30-1

U

unbound,

- term definition; 2-9, 5-7

unbound variables,

- expression,
 - output arc inscription example; 40-14

Undo (Edit menu),

- reference description; 20-2

Ungroup (Group menu),

- deselecting groups with; 3-21

union colorsets,

- See Also* colorsets;
- input arc inscription example; 40-12
- selectors; 32-19
 - specifying; 32-16
- syntax and characteristics; 32-16
- testing for membership in; 32-27

unit colorsets,

- See Also* colorsets;
- ordering of; 32-28
- syntax and characteristics; 32-2

unlocking,

- objects; 24-4
- regions relative to the parent,
 - with Size component (Region Attributes command-Set menu); 24-9

untimed reserved word,

- making a colorset with; 39-1

Update Chart (Sim menu),

- reference description; 22-13
- updating,
 - bar charts with; 15-10
 - line charts with; 15-27

updating,
 bar charts,
 from transition code segments; 15-10
 specifying values for; 15-9
 update period specification; 21-12
 using SC_upd_chart function in a transition code
 segment; 15-10
 with chart code segment; 15-8
 charts; 15-1
 with Update Chart command (Sim menu); 22-13
 history charts; 15-16
 update period specification; 21-12
 line charts; 15-24
 from transition code segments; 15-27
 specifying values for; 15-25
 update period specification; 21-16
 using LC_upd_chart function in a transition code
 segment; 15-27
 with chart code segment; 15-24
 statistical variables; 14-3
use command,
 arc inscriptions not permitted to use; 40-5
user interface,
 code segments used for; 40-17
 design,
 See appearance;
 Design/CPN components; 1-5

V

val reserved word,
 reference variables defined with; 35-1
 syntax and characteristics; 33-1
val-defined identifiers,
 expression use; 36-2
validating,
 place names; 40-2
value (clr') function,
 syntax and characteristics; 32-23
value (col') function,
 syntax and characteristics; 32-22
values,
 (chapter); 33-1
 bar charts,
 specifying the type and range; 21-13
 binding to variables,
 with Bind command (Sim menu); 22-3
 declaring; 33-1
 expression use; 36-2
 history charts,
 specifying the type and range; 21-13

values (cont'd),
 line charts,
 specifying the type and range; 21-16
 representation; 33-1
 statistical variables,
 average; 14-3
 count; 14-3
 current value; 14-4
 first value; 14-4
 maximum; 14-4
 minimum; 14-4
 standard deviation; 14-4
 sum; 14-5
 sum of the squares; 14-5
 sum of the squares of deviation; 14-5
 variance; 14-5
 tokens,
 constraining; 5-8
 constraining with the use of guards; 5-8
 specifying multiple tokens with the different
 values; 5-6
 specifying multiple tokens with the same value;
 5-5
 specifying with constants; 5-4
 specifying with CPN variables; 5-7
 tuple,
 term definition; 2-5
variables,
 CPN,
 See Also CPN variables;
 (chapter); 34-1
 as CPN ML extension; 30-2
 binding values to with Bind command (Sim menu);
 22-3
 calculated, identification of in included bindings
 list box; 22-4
 calculated, identification of in potential bindings
 list box; 22-5
 code segment restrictions; 40-17
 code segment use of; 13-1
 declaration syntax; 34-2
 expression use; 36-2
 global reference, syntax and characteristics; 35-3
 input arc inscription example; 40-9
 output arcs, binding of; 13-1
 records with, input arc inscription example; 40-11
 specifying token values with; 5-7
 term definition and characteristics; 2-9, 34-1
 tuples with, input arc inscription example; 40-10
 free,
 term definition; 40-5

variables (cont'd),

- instance reference,
 - syntax and characteristics; 35-3
- local,
 - declaring, with let construct; 38-1
- multisets,
 - term definition, syntax, and use; 37-1
- page reference,
 - syntax and characteristics; 35-3
- reference,
 - See Also* reference variables;
(chapter); 35-1
 - arc inscriptions not permitted to use; 40-5
 - as CPN ML extension; 30-3
 - expression use and restrictions; 36-2
 - restrictions on use; 35-1
 - val-defined; 35-1
 - val-defined, syntax and characteristics; 35-2
- statistical,
 - (chapter); 14-1
 - accessing; 14-3
 - charts and, using (chapter); 16-1
 - clearing; 14-5
 - creating; 14-2
 - initializing; 14-2
 - initializing, in Resource Use model; 16-5
 - methodology for use; 14-1
 - structure; 14-2
 - updating; 14-3
 - updating, in Resource Use model; 16-6
- term definition; 34-1
- type,
 - term definition; 30-4
- unbound ,
 - output arc inscription example; 40-14

variance,

- statistical variable,
 - obtaining, with SV'vari; 14-5

variant record,

- See* union colorset;

vertexes,

- term definition; 3-14

Vertical (Align menu),

- reference description; 29-2

Vertical Spread (Align menu),

- reference description; 29-5

visibility (graphical attribute),

- changing; 17-16
- characteristics; 17-16
- of editor regions,
 - changing; 17-3

visibility (graphical attribute) (cont'd),

- of simulator regions,
 - changing; 17-3

visibility of page border (page attribute),

- characteristics; 17-15

visible,

- page borders,
 - rendering for display or printing; 24-11

W

Wedge (Aux menu),

- reference description; 23-7

wedges,

- creating, modifying, and moving; 23-7
- shape attributes,
 - changing; 24-7

wildcards,

- not permitted in CPN ML; 30-4

with_code function,

- reference description; 41-6

with_time function,

- reference description; 41-6
- syntax and characteristics; 39-4

write_report function,

- reference description; 41-6

writing,

- colorsets,
 - with output_col function; 41-8
- multisets,
 - timed, with output_tms function; 41-8
 - with output_ms function; 41-5, 41-8
- reports,
 - with write_report function; 41-6

X, Y

X-Windows,

- Caps Lock key behavior; 3-5

Z

zero function,

- reference description; 41-7

zeroing,

- with zero function; 41-7

PART 1

Design/CPN User's Guide

Chapter 1

Getting Started With Design/CPN

Prerequisites

This manual assumes that you have taken The Design/CPN Tutorial, which is available from Meta Software, or have become thoroughly familiar with colored Petri nets in some other way.

Terminology

The following terms are used in this manual:

- **Model:** A symbolic representation of some aspects of a system.
- **Simulation:** The execution of a model to derive information about a system.
- **Colored Petri Net:** A specialized graphical network with associated computer language statements, typically used for system modeling and simulation.
- **CP Net:** Short for Colored Petri Net.
- **Net:** Unless otherwise stated, a CP net.
- **Net Structure:** Graphics and text that comprise a CP net. Net structure is to a net as code is to a program.
- **CPN:** A programming language that provides the syntactic, graphical, and textual capabilities needed for building CP nets.
- **CPN Model:** A CP net that is specifically intended to model a system.
- **Design/CPN:** An interactive utility for creating, modifying, and executing CP nets.

- **Diagram:** A CP net built with Design/CPN and stored on a computer. A diagram is to a net as a text file is to text.

More detailed definitions of these term will be given as needed throughout this manual.

What Is Design/CPN

Design/CPN is an interactive computer tool for performing modeling and simulation with CP nets. Design/CPN provides:

- An editor for creating and manipulating CP nets.
- A syntax checker for validating CP nets.
- A simulator for executing CP nets.
- Interactive monitoring and debugging capabilities.
- Facilities for organizing a CP net into a hierarchy of modules.
- Charting facilities for displaying simulation results.

These capabilities allow CP nets to be conveniently created, modified, organized, executed, debugged, examined, and validated.

Design/CPN and X-Windows

Design/CPN is a standard X-Windows program. All standard X-Windows user interface techniques are available, and work as they do in X-Windows programs generally.

Design/CPN Multiprocessing

When Design/CPN is active, it sometimes accesses a second process, called the *ML process*, that interprets and compiles computer code. The two processes communicate and cooperate with each other to execute CP nets.

In order to be executed, a net must contain some information that Design/CPN uses to access the ML process. This information is automatically included in a new net when it is created. When a pre-existing net is imported from another system, the necessary information must be explicitly loaded into the net before it can be executed.

Design/CPN and the File System

A *diagram* is a CP net and optional additional graphics created using Design/CPN. Design/CPN stores a diagram as a group of three files. Multiple files are used rather than one because a single file would often be inconveniently large. The files are:

1. The *diagram file*. This contains data that represents the diagram in a form suitable for displaying and editing.
2. The *DB file*. This contains a database describing the diagram. It has the same name as the diagram file, with the addition of the suffix “.DB”.
3. The *ML file*. This contains code that represents the diagram in executable form. It has the same name as the diagram file, with the addition of the suffix “.ML”.

Thus the diagram AirlineModel would be stored in the files:

AirlineModel
AirlineModel.DB
AirlineModel.ML

The diagram and DB files contain a complete description of the model. The ML file is not generated until it is needed, and can be regenerated if necessary using the information in the diagram and DB files. Therefore a diagram may not have an ML file, either because the file was never created by Design/CPN, or because it was subsequently deleted. ML files are quite large, and are often deleted to save disk space.

DB and ML files are for internal use only by Design/CPN. When you want to open a particular diagram, open the diagram file; DB and ML files cannot be opened directly.

The only time you need to think about DB and ML files is when you copy, move, or rename a diagram via file system commands. Be sure to treat all of the files that constitute the diagram in the same way.

Design/CPN Use of the Mouse

When you manipulate windows and scroll bars while using Design/CPN, the three mouse buttons have their standard functions in the X-Windows user interface. When you perform a mouse operation that is specific to Design/CPN rather than to X-Windows in general, use the left button. Design/CPN uses only one mouse button (to allow for porting to one-button mouse systems such as the Macintosh), so it ignores the middle and right buttons.

Design/CPN Use of the Keyboard

Design/CPN documentation specifies the use of the ALT key in various keystroke shortcuts. Some keyboards use a DIAMOND () key instead; occasionally other keys are used. If ALT does not produce the described results, check your terminal configuration.

Figures in This Manual

This version of the Design/CPN Reference Manual shows Macintosh-style dialog boxes rather than X-Windows-style boxes. In almost every case, the Macintosh and X-Windows boxes are functionally identical, differing only in the details of their appearance. Where there is a functional difference that bears on a topic under discussion in the manual, the difference is explicitly described.

Other than dialog box appearance, there should be no differences between what this manual describes and what happens under X-Windows.

Starting Design/CPN

Start Design/CPN as you would any X-Windows program.

If you see a dialog that mentions a problem of any kind, see Appendix B, "Troubleshooting."

Creating or Opening a Diagram

To create a new diagram:

Choose **N****e****w** from the **F****i****l****e** menu.

To open an existing diagram:

Choose **O****p****e****n** from the **F****i****l****e** menu.

The Design/CPN User Interface

The Design/CPN user interface has three components:

1. A *menu bar* at the top of a small window.
2. A *status bar*, immediately below the menu bar in the same window.
3. A *page* in a second window.

The Menu Bar

This is an ordinary pulldown menu bar.

The Status Bar

The status bar is used by Design/CPN to post brief messages that describe CP net components, editor and simulator states and actions, and various other things that you may want to know about as you work with Design/CPN.

If you are ever in doubt about where you are in Design/CPN, or want to see if anything is going on that you should know about but do not, check the status bar. It can be extremely informative.

The Page

A page is a “blank slate” on which you can create CP net structure. A diagram may contain any number of pages.

Every page is displayed in a separate window. A newly created page is centered under its window, rather than extending down and to the right as a file in a text editor would. It is centered because CP nets, being graphical rather than textual, tend to grow radially rather than linearly, so that empty drawing space is as likely to be needed in one direction as in another.

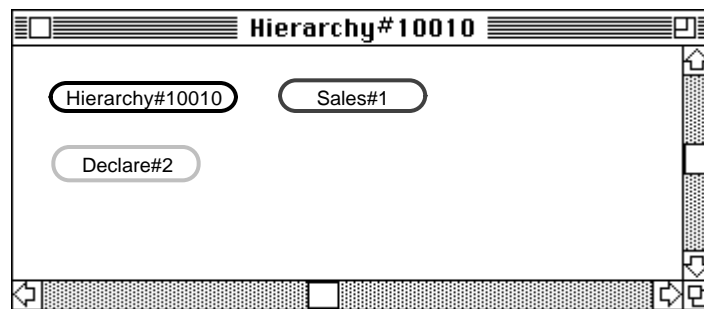
The center of a page is marked by a rectangle, called the *page border*. The dimensions of this rectangle are the **Page Width** and **Page Height** specified in the **Page Attributes** dialog (**Set** menu). If you print a page and do not specify any other behavior (via **Page Setup** from the **File** menu), the area inside the page border will be printed. The page border can be made invisible via the **Page Attributes** dialog.

Navigating a Diagram

A Design/CPN diagram typically consists of several pages. A given page may be *open*, in which case it is visible in a window, or *closed*, in which case it is not visible.

When a diagram contains more than one page, a method is necessary for keeping track of the pages and their relationship to each other. To provide this capability, every diagram contains a special page called a *hierarchy page*.

A diagram's hierarchy page contains a small oval, called a *page node*, for every page in the diagram, including itself. Each page node contains the name of the corresponding page. For example:



When a diagram contains many pages linked into a hierarchical CP net, the hierarchy page becomes a rich source of information about the composition and structure of the net.

You can use the hierarchy page to navigate to any other page in a diagram by selecting the page node and either double-clicking or choosing **Open Page** from the **Page** menu. See the section "Navigating a Diagram" in Chapter 17, "Reference Summaries," for further techniques.

Printing a Diagram

Printing a diagram is similar to printing any other file. The details of the printer dialog depend on the particular printer you are using, so they cannot be covered here.

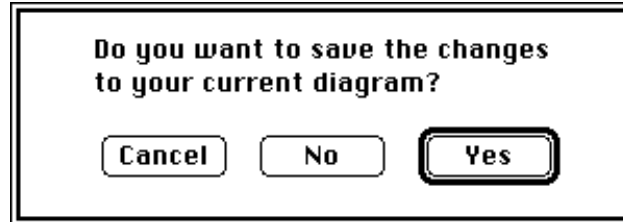
Every Design/CPN page name includes a page number, e.g. Sales#1 and Declare#2. Page numbers can be used to tell the printer to print a particular page or range of pages. Diagram pages have no intrinsic order, so Design/CPN numbers them by assigning each new page the lowest number not currently in use in the diagram. This number can be changed via the **Page Attributes** command (**Set** menu).

Closing a Diagram

To close a diagram:

Choose **Close** from the **File** menu.

If you have made any changes to the diagram, the **Save Changes** dialog appears:



Quitting Design/CPN

To quit Design/CPN:

Choose **Quit** from the **File** menu.

If you have made any changes to the diagram, the **Save Changes** dialog appears.

Chapter 2

CP Net Components

A CP net is a graphical structure with associated computer language statements. The principal components of a CP net are:

- **Data:** CP nets make use of datatypes, data objects, and variables that hold data values. CPN data is defined using a computer language called *CPN ML*.
- **Places:** Locations for holding data.
- **Transitions:** Activities that transform data.
- **Arcs:** Connect places with transitions, to specify data flow paths.
- **Input Arc Inscriptions:** Specify data that must exist for an activity to occur.
- **Guards:** Define conditions that must be true for an activity to occur.
- **Output Arc Inscriptions:** Specify data that will be produced if an activity occurs.

This chapter defines all of these components, and shows how they interrelate syntactically to form a CP net. This chapter does not discuss how they work together dynamically to define a CP net's behavior when it is executed. That is covered in Chapters 5 through 8, "CPN Dynamics."

The CPN ML Language

The language ML (for Meta Language, no relation to Meta Software) is a strongly typed functional language that provides great economy of expression, is easily extended by the user, and can execute interpretively or be compiled.

To facilitate the use of ML with CP nets, Meta Software has added various extensions, resulting in the language CPN ML (for Colored

Design/CPN User's Guide

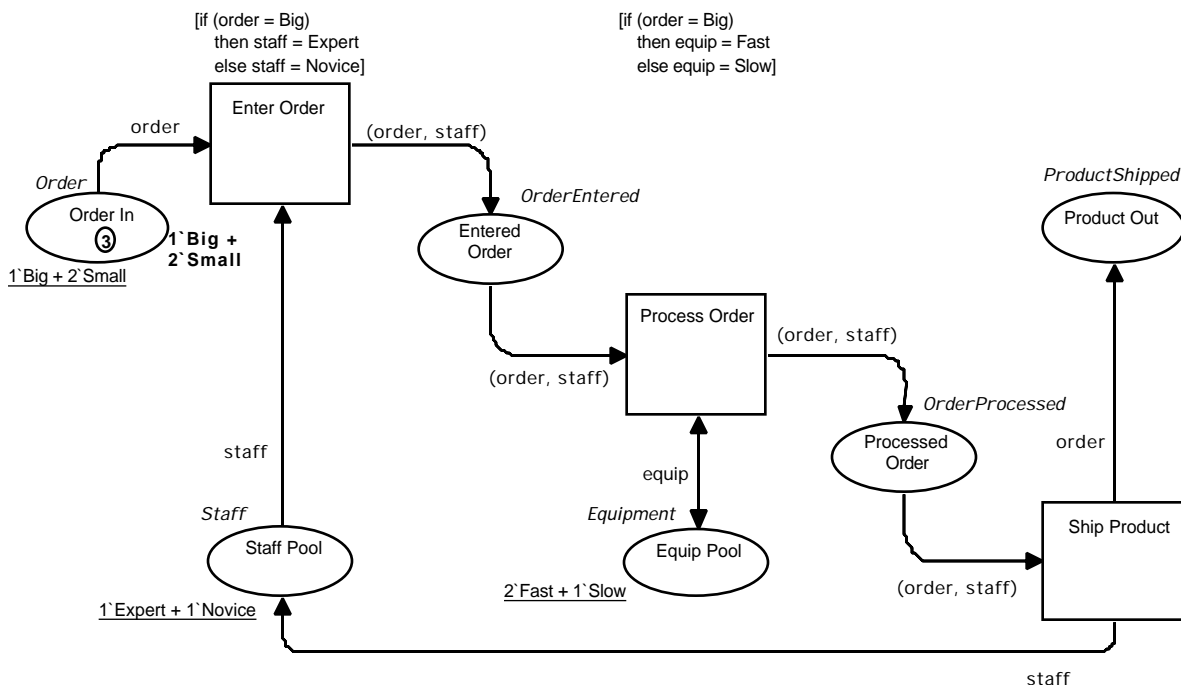
Petri Net ML). Very little of ML is ever needed for working with CP nets, so in practice CPN ML is a very small language.

CP nets use CPN ML statements to declare data, define arc inscriptions and guards, and provide communication between a CP net and its environment.

You do not need to become a CPN ML programmer in order to use CP nets. The CPN equivalent of programming is done by creating graphical CP net structure, not by writing language statements. You need only learn the particular syntactic conventions through which CPN ML does various low-level things that any computer language does: declaring datatypes and variables, comparing one data value with another, and so on.

A CP Net Example

The various CP net components will be discussed in the context of a net called OrderProcessing:



CPN Data

CP nets use datatypes, data objects, and variables. CPN datatypes are called *colorsets*. CPN data objects are called *tokens*. Tokens are somewhat like objects in an object-oriented programming system.

All CPN datatypes and variables must be declared. They are declared in a declaration box called a *global declaration node*.

OrderProcessing's global declaration node contains the following declarations:

```
color Order = with Big | Small;           (* Orders for products *)
color ProductShipped = Order;             (* Orders shipped *)
color Staff = with Expert | Novice;       (* Staff members *)
color Equipment = with Fast | Slow;       (* Pieces of equipment *)

color OrderEntered = product Order * Staff; (* Orders entered but unprocessed *)
color OrderProcessed = OrderEntered;      (* Orders processed but unshipped *)

var order: Order;                         (* An order *)
var staff: Staff;                         (* A staff member *)
var equip: Equipment;                     (* A piece of equipment *)
```

Colorsets

CP net tokens can be of all the datatypes generally available in a computer language: integers, reals, strings, booleans, lists, tuples, records, and so on. In the context of CP nets, these types are called colorsets.

“Colorset” is really just a synonym for “token datatype.” The term “colorset” is used, rather than a standard term such as “datatype,” to avoid confusion between token datatypes, which are extensions to standard ML, and ordinary ML datatypes.

Just as every piece of data in an ordinary computer program is of some datatype, so every token in a CP net is of some colorset. All colorsets used in a net must be explicitly declared. The declaration syntax is:

```
color name = definition;
```

where name is the name of the colorset, and definition specifies what it consists of. The syntax for definition varies, depending on what sort of colorset is being declared.

Enumerated Colorsets

One common colorset defines tokens that may have any of a set of predefined literal values. The possible values are all enumerated in the colorset declaration, so this colorset is called an *enumerated colorset*. The syntax of an enumerated colorset declaration is:

```
color name = with value { | value } ...;
```

The construction “{ } . . .” indicates that the material in braces may be repeated zero or more times. Underlining indicates a term that is to be replaced with an actual value.

Examples:

```
color oneval = with EXISTS;
color booltok = with yes | no;
color chord = with Major | Minor |
Augmented | Diminished;
```

The first declaration defines an enumerated colorset named `oneval`. All tokens of this colorset have the same value: `EXISTS`. The second defines an enumerated colorset `booltok`. Every token of type `booltok` has either the value `yes` or the value `no`. The third declaration defines `chord`, whose tokens may have any of the values `Major`, `Minor`, `Augmented`, or `Diminished`. All of these names and values, and all other CPN ML keywords and identifiers, are case-sensitive.

OrderProcessing uses three enumerated colorsets:

```
color Order = with Big | Small;
color Staff = with Expert | Novice;
color Equipment = with Fast | Slow;
```

The first declaration indicates that `Order` tokens may have either of the values `Big` or `Small`. The others should be obvious.

String and Integer Colorsets

Strings and integers are commonly used in CP nets. The syntax for a string colorset declaration is:

```
color name = string;
```

The syntax for an integer colorset declaration is:

```
color name = int;
```

It is possible to restrict the length and character set of a string colorset, and the range of an integer colorset, by adding clauses to the

declaration. These are discussed in Part 3, “The CPN ML Reference Guide.”

Duplicate Colorsets

It is often useful, both in programming and in modeling (which is really just a kind of programming), to have more than one colorset that consists of the same values. In CPN ML, such a colorset is called a *duplicate colorset*. The declaration syntax is:

```
color DuplicateName = ExistingName;
```

where DuplicateName is the name of the duplicate type being defined, and ExistingName is the name of some colorset that has already been declared. OrderProcessing declares a duplicate colorset called `ProductShipped`:

```
color ProductShipped = Order;
```

`ProductShipped` tokens may have the same values that `Order` tokens can: `Big` or `Small`.

Tuple Colorsets

A *composite colorset* is a colorset that is constructed of other colorsets. CPN ML provides many composite colorsets: records, lists, tuples, and various others.

A *tuple* is a composite colorset that is a cartesian product of two or more other colorsets. Every member of a tuple colorset is an ordered sequence of values, each of which is drawn from the subsidiary colorset defined for its position in the tuple. The syntax of a tuple colorset declaration is:

```
color name = product colorset1 * colorset2;
```

For example, OrderProcessing includes the colorset `OrderEntered`:

```
color OrderEntered = product Order * Staff;
```

`OrderEntered` is a tuple. Thus every member of `OrderEntered` is a sequence of two values, the first of which is of type `Order`, and the second of which is of type `Staff`.

A *tuple value* is denoted by listing the constituent values in parentheses separated by commas. For example, `OrderEntered` consists of the four values:

```
(Big, Expert)
(Big, Novice)
```

```
(Small, Expert)
(Small, Novice)
```

These values are simply the cartesian product of the colorsets `Order` and `Staff`.

Tuple Constructors

A *tuple constructor* is a method of specifying a tuple value by using CPN variables rather than constants. CPN variables are discussed in the section “CPN Variables” later in this chapter. The syntax is:

```
(variable1, variable2)
```

For example, the colorset of `Entered Order` is `OrderEntered`, which was declared as:

```
color OrderEntered = product Order * Staff;
```

That is, every member of `OrderEntered` is a tuple the first element of which has the value `Big` or `Small` (the elements of `Order`) and the second element of which has the value `Expert` or `Novice` (the elements of `Staff`).

The output arc inscription to `Entered Order` has the form of an `OrderEntered` tuple, but instead of using wired-in values of appropriate type, as in:

```
(Big, Expert)
```

it uses CPN variables of appropriate type:

```
(order, staff)
```

If `order` is bound to `Big` and `staff` is bound to `Expert`, then `(order, staff)` evaluates to `(Big, Expert)`.

Tuple Patterns

The specification on the arc between `Entered Order` and `Process Order` is called a *tuple pattern*. It is the inverse of a tuple constructor. Where a constructor takes bound variables and produces a composite value, a pattern takes a composite value and binds its constituents to variables. The binding of variables is discussed in the section “Binding an Arc Inscription Variable” in Chapter 5, “CPN Dynamics: Introduction.”

When the simulator checks the enablement of a transition with that has a pattern on an input arc, it does not bind whole token values to variables. Instead it matches the value of each token that it looks at against the pattern, and binds the variables in the pattern to the com-

ponents in the token value. The bound values are then available for use just as if the variables had been bound directly, as `order` and `staff` are in the context of the `Enter Order` transition. Enablement is discussed in the section “Criteria for Enablement” in Chapter 5, “CPN Dynamics: Introduction.”

Other Colorsets

Part 3, “The CPN ML Reference Guide,” describes in detail all the colorsets supported by CPN ML.

Tokens

CPN data objects are called *tokens*. They are called tokens rather than objects to avoid confusion with graphical objects. Every token used in a CP net must be of one of the colorsets declared for the net.

A CPN token is represented by giving its value. For example:

```
Xenia  
"Clifton"  
45387
```

The first example specifies a token of an enumerated colorset. The token's value is `Xenia`. The second specifies token of a string colorset (a string token) with the value `"Clifton"`. The third specifies an integer token; its value is `45387`.

Note that when there is more than one colorset that includes some possible value, there is no way to tell by looking at a token which colorset the token belongs to. This doesn't create ambiguity, because the way tokens are used in CP nets always indicates the correct colorset contextually.

Multisets of Tokens

In dealing with CP nets, it is often necessary to manipulate and refer to collections of tokens. A collection of zero or more tokens is called a *multiset*. A multiset is similar to an ordinary mathematical set, except that it may contain more than one instance of the same element. That is, a multiset may contain multiple tokens that have the same value. All tokens in a multiset must be of the same colorset. Any subset or union of multisets is again a multiset.

A multiset is not a data structure that exists separately from the tokens it contains. It is just a convenient way to refer to a collection of tokens. Thus a single token and a multiset of one token are the same thing. A multiset of no tokens has no properties of its own, so there is only one such multiset, called the *empty multiset*. When the empty multiset must be designated explicitly, it is written as `empty`.

Specifying Multisets

A multiset is specified by giving an expression that describes the multiset. The simplest such expression specifies a multiset of identical tokens. Such an expression is called a *multiset designator*.

A multiset designator consists of an integer denoting the number of tokens in the multiset, followed by a *backquote* (```), the *multiset creation operator*, followed by the value of the tokens:

```
count`value
```

For example:

```
1`Xenia
3`"Clifton"
2`45387
```

The first example specifies a (multiset containing a) single token of an enumerated colorset. The token's value is `Xenia`. The second specifies a multiset of three string tokens, each with the value `"Clifton"`. The third specifies a multiset of two integer tokens, each with the value 45387.

Multiset Addition

Multisets of the same colorset may be added:

```
1`"Springfield" + 3`"Springfield" =>
4`"Springfield"
```

A multiset containing tokens with different values is specified by giving an expression that is a sum of multiset designators. Thus:

```
7`67 + 7`3 + 3`1
```

is a multiset expression. It specifies a multiset of 17 integer tokens: seven with the value 67, seven with the value 3, and three with the value 1. (The same multiset would result if the terms were added in any other order.)

Multiset Subtraction

Multisets of the same colorset may be subtracted:

```
5`"Clifton" - 3`"Clifton" => 2`"Clifton"
2`Gegner - 2`Gegner => empty
```

The empty multiset may be subtracted from any multiset:

```
1`513 - empty => 1`513
```


Multiset subtraction may not attempt to take away tokens that do not exist. Therefore:

```
2`10 - 3`5
1`"Helen" - 1`"Glen"
empty - 2`Birch
```

are not valid subtractions, and would result in an error.

Multiset Subsets

When all the elements of one multiset also exist in a second multiset, so that the first could be subtracted from the second, the second is a *subset* of the first. (There is no need to call it a “submultiset”, because the context always prevents confusion with ordinary sets and subsets.) By definition, the empty multiset is a subset of every multiset.

Thus the subsets of the multiset:

```
1`Land + 2`Sea
```

are:

```
1`Land + 2`Sea
1`Land + 1`Sea
1`Land
2`Sea
1`Sea
empty
```

CPN Variables

CP net execution often requires token values to be accessed and used in various ways. Such usage is no different than the manipulation of data values that occurs in ordinary computer programs.

It would not work to restrict a CP net to handling only token values that are known in advance. This would be equivalent to requiring a computer program to use only constants. Therefore CP nets contain variables that can take on token values as needed. These variables are called *CPN variables*.

Like colorsets, CPN variables must be explicitly declared. Every CPN variable is of a particular colorset, and can take on only values of that type. When a CPN variable has taken on the value of some token, it is said to be bound to that value. When a CPN variable is not bound to any value, and is said to be *unbound*.

The OrderProcessing example uses three variables: `order`, `staff`, and `equipment`:

Design/CPN User's Guide

```
color Order = with Big | Small;          (* Orders for products *)
color ProductShipped = Order;            (* Orders shipped *)
color Staff = with Expert | Novice;      (* Staff members *)
color Equipment = with Fast | Slow;      (* Pieces of equipment *)

color OrderEntered = product Order * Staff; (* Orders entered but unprocessed *)
color OrderProcessed = OrderEntered;      (* Orders processed but unshipped *)

var order: Order;                        (* An order *)
var staff: Staff;                        (* A staff member *)
var equip: Equipment;                    (* A piece of equipment *)
```

The variable `Order` can be bound to the values `Big` or `Small`.

The variable `Staff` can be bound to the values `Expert` or `Novice`.

The variable `Equipment` can be bound to the values `Fast` or `Slow`.

See the section “Binding an Arc Inscription Variable” in Chapter 5, “CPN Dynamics: Introduction,” for further details about the use of CPN Variables.

Places

Tokens would be of little use if they could not be stored and accessed as needed. CP nets keep tokens in locations called places.

A *place* is a location that can contain zero or more tokens (a multiset) of some particular colorset. A place's colorset is one of the colorsets defined for the net. All tokens in a place must be of that colorset.

A place may optionally have a name. Such a name is useful for indicating what the place means as a part of a model, for identifying the place when humans confer about the net, and for labeling computer-generated information about activity in the place during simulation runs.

A place is graphically represented as an ellipse. Its name and/or colorset may optionally be displayed next to or inside the place. They are graphically represented as labels that are regions of the place. The regions are named for their contents: a place's name is kept in a *name region*, its colorset in a *colorset region*, etc. The section “Creating Regions” in Chapter 3, “The Design/CPN Editor: Introduction,” discusses regions and their use.

The example net, OrderProcessing, contains six places:

- Order In
- Staff Pool
- Entered Order
- Equip Pool
- Processed Order
- Product Out

Place Markings

The purpose of a place is to hold tokens. The tokens in a place, like any group of tokens, constitute a multiset. The multiset in a place is called the *marking* of the place. A place always has a marking: if it contains no tokens, its marking is the multiset `empty`.

When a place is empty, no marking is shown for it. (An explicit `empty` could be written, but that would serve no purpose.) The marking of a nonempty place is depicted as a circle with a number inside indicating the number of tokens in the multiset, followed by an expression that describes its contents:

① 1`"Xenia"

④ 4`Big

⑪ 7`67 + 7`3 + 3`1

States and Markings

Taken together, the markings in all the places in a CP net constitute the *state* of the net. The first state of a net, before execution begins, is called the *initial state* of the net, and the place markings that constitute it are called the *initial markings* of the places.

As a CP net executes, tokens are put into and removed from places, so that the state of the net changes. Its state at a given time is called its *current state*, and the place markings that constitute that state are the *current markings* of the places. Thus a CP net's initial state is the first of a succession of current states, and a place's initial marking is the first of a succession of current markings.

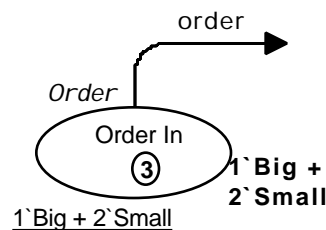
Initial Marking Regions

In the example net OrderProcessing, Order In has an associated multiset expression: $1\text{'Big} + 2\text{'Small}$. This is another region of the place, called the *initial marking region*. When OrderProcessing is executed, the tokens specified in this region will be put into the place Order In before execution starts, providing Order In's initial marking.

Product Out has no initial marking region, so no tokens will be put in it before execution starts. Put another way, its initial marking is empty. The section "Creating Regions" in Chapter 3, "The Design/CPN Editor: Introduction," discusses regions and their use.

Appearance of Markings

The initial marking and current marking of Order In place in the Order Processing example looks like this:



In this figure, the place Order In contains three tokens, one of value Big and two of value Small, as specified by the initial marking region. These tokens constitute the initial marking of the place, and are its current marking as well.

The existence of both an initial marking region and a current marking for a place is not a redundancy. The initial marking region is just a label with some text in it that specifies some tokens that will be put in the place before execution starts. The current marking represents the actual tokens.

A place's current marking may be depicted anywhere that is convenient. Placing it as shown above is just a convention, used because it gives a good appearance in this case.

Transitions

A *transition* is an activity whose occurrence can change the number and/or value of tokens in one or more places. A transition may optionally have a name, kept in a name region, which serves the same purposes as a place name. A transition is graphically represented as a rectangle. If it has an associated name, it may optionally be dis-

played next to or inside the transition. The section “Creating Regions” in Chapter 3, “The Design/CPN Editor: Introduction,” discusses regions and their use.

OrderProcessing has three transitions:

- Enter Order
- Process Order
- Ship Product

Arcs

An *arc* is a connection between a place and a transition. Every arc connects one place with one transition. Every arc has a direction, either from a place to a transition, or from a transition to a place.

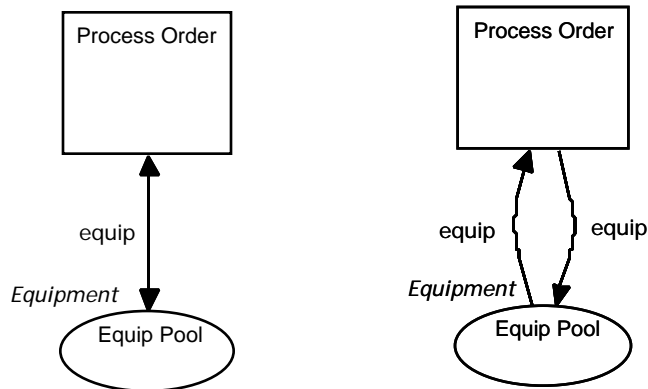
An arc is represented graphically as an arrow with a head that indicates its direction.

An arc that runs from a place to a transition is called an *input arc*, and the place it connects to is called an *input place*. An arc that runs from a transition to a place is called an *output arc*, and the place to which it connects is called an *output place*. It is possible for a place to be both an input place and an output place.

When a transition occurs, it may remove tokens from any or all of its input places, and put tokens into any or all of its output places. The number and values of tokens removed and added are determined by arc inscriptions and guards.

Bidirectional Arcs

The arc between Equip Pool and Process Order is bidirectional. Such an arc can be used (but does not have to be) whenever both an input and an output arc connect a given place and transition, and the inscriptions on the two arcs are identical. Thus these two structures are exactly equivalent:



Bidirectional arcs are commonly used when transition enablement requires the existence of a resource that firing does not consume, or of a condition that firing does not change.

Arc Inscriptions

An *arc inscription* is a multiset expression associated with an arc. It is kept in a region of the arc called an *arc inscription region*. The section “Creating Regions” in Chapter 3, “The Design/CPN Editor: Introduction” discusses regions and their use.

An arc inscription on an input arc is called an *input arc inscription*; an arc inscription on an output arc is called an *output arc inscription*.

The tokens in the multiset specified by an input arc inscription must be present in the input place in order for the transition to occur (there may be other tokens there also). These tokens will be removed from the input place if the transition does occur. The multiset specified by an output arc inscription will be put into the output place if the transition occurs.

An arc inscription need not be given explicitly. The default arc inscription is `empty`, the multiset of no tokens. When an input arc inscription is given as or defaults to `empty`, no tokens need to be in the input place in order for the transition to occur. When an output arc inscription is given as or defaults to `empty`, no tokens will be put into the output place if the transition does occur.

Guards

A *guard* is a boolean expression associated with a transition. Guards are customarily written inside square brackets, to help distinguish them from other regions. The section “Creating Regions”

in Chapter 3, “The Design/CPN Editor: Introduction” discusses regions and their use.

A transition's guard must evaluate to boolean **true** in order for the transition to occur.

A guard need not be given explicitly. The default guard is boolean **true**. When a guard is given as or defaults to **true**, the guard never restricts the transition from occurring.

The section “Constraining Token Values with Guards” in Chapter 5, “CPN Dynamics: Introduction,” describes the use of guards during CP net execution.

CP Net Execution

We have now looked at the principal components of a CP net, and have seen how they relate to each other syntactically. But that is only part of the story. We wouldn't need all this machinery just to describe the structure of a system: a static modeling paradigm such as IDEF0 can do that with much less effort.

A CP net is more than a description of system structure: CP nets are intended to be executed. Such execution can provide information about system dynamics that could never be derived by looking at a static model and considering its implications. There is just too much information involved in the functioning of a complex system for the unaided human mind to cope with.

This chapter has not addressed the question of how CPN components work together to define how a CP net will behave during execution. This question is answered in Chapters 5 through 8, “CPN Dynamics.” Chapter 4, “The Design/CPN Editor: Creating a CP Net,” describes how the CPN components combine into CP nets.

Chapter 3

The Design/CPN Editor: Introduction

The Design/CPN editor provides facilities for dealing with both geometric forms and textual information. It represents such information as *graphical objects*. Each graphical object represents one graphical entity: a rectangle, ellipse, text string, etc. A graphical object is stored as a set of instructions for drawing the particular object on the screen.

Design/CPN graphical objects are of two kinds: auxiliary objects and CPN objects. An *auxiliary object* is just an ordinary graphical object, such as any graphics editor can produce. A *CPN object* is a graphical object that is part of a CP net.

The difference is not one of appearance, but one of function. An auxiliary object is just itself; there is nothing to it but its physical form. A CPN object is more than its physical form: it also has a meaning as a CP net component.

Auxiliary objects are used primarily to add commentary to CP nets. Objects can be converted between auxiliary and CPN status as needed. This permits objects to be deactivated as CP net components without having to be deleted, just as otherwise executable lines are often “commented out” for various reasons during development of an ordinary program.

This chapter demonstrates all the essential Design/CPN editor techniques. It illustrates these techniques using auxiliary objects rather than CPN components. This approach makes it possible to ignore the details of CPN syntax and concentrate on describing graphical operations. Chapter 4 covers using the Design/CPN Editor to create CP nets. The section “Graphical Objects” in Chapter 17, “Reference Summaries,” provides detailed descriptions of all the Design/CPN graphical objects and their characteristics.

Design/CPN Graphical Objects

The Design/CPN editor provides many types of graphical objects, including rectangles, ellipses, connectors, labels, and many more. All of these are created and edited in essentially the same way; the details differ slightly from type to type, to suit the individual characteristics of each.

Every Design/CPN graphical object is either a node or a connector. A *node* is any object that can exist in its own right, such as a rectangle or an ellipse. A *connector* is an arrow that runs between one node and another. A connector cannot exist independently: it can exist only when there are two nodes for it to connect.

Every Design/CPN graphical object consists of one or both of:

- A geometric form.
- A text field.

One object type, the *label*, has no geometric form and a required text field. Every other object type has a geometric form and an optional text field.

A graphical object's geometric form and/or text field define its fundamental properties, but do not specify the details of its physical appearance: its line thickness, fill pattern, color, font, etc. All aspects of an object's appearance can be controlled by setting various graphic and textual attributes of the object. Such attributes are called *display attributes*.

Graphics Editor Modes

It is often useful to perform the same operation repeatedly while using the editor: to create a series of rectangles, for example, or to edit text in a series of locations. To facilitate such repeated operations, the editor offers a variety of *editor modes*. When the editor is in a particular mode, you can perform the mode's characteristic action repeatedly using only the mouse and keyboard, without the need to respecify each time what you want to do.

In each editor mode, Design/CPN displays a distinctive mouse pointer that indicates the mode. Such a pointer is known as a *drawing tool*. Specific drawing tools are described later in this chapter.

The capabilities needed for editing geometric forms differ from those needed for editing text. The editor therefore has two *editing modes*: graphics mode and text mode. In *graphics mode*, you can move and reshape an object's geometric form using the mouse. In *text mode*,

you can use both the mouse and the keyboard to edit an object's text field.

Graphics Mode

When the editor is in graphics mode, the status bar displays Text: Off. If you are not performing any particular activity, the mouse pointer is the *graphics tool*:



(The tool is shown larger than actual size.)

The graphics tool simply indicates a location: the location it indicates is the pixel under the dot at the center of the tool. All drawing tools that indicate a specific pixel location include such a dot.

As you perform particular activities in graphics mode, the mouse pointer changes to various other tools to indicate the activity currently underway, then reverts to the graphics tool when the activity is complete.

Text Mode

When the editor is in text mode, the status bar displays Text: On. If you are not performing any particular activity, the mouse pointer is the *text tool*:



This tool indicates a location, as the graphics tool does, and contains a “T” to indicate text mode. As you perform particular activities in text mode, the mouse pointer changes to various other tools to indicate the activity currently underway, then reverts to the text tool when the activity is complete.

Creating Graphical Objects

When you are in the editor, you can create new graphical objects at any time. The general sequence for creating a new auxiliary object is:

1. Use the **Aux** menu to choose the type of object you want to draw.
2. Use the mouse to specify the location and shape of the object.
3. Create more objects of the specified type if desired.

When you choose an object type from the **Aux** menu, the editor switches from whichever editing mode it is in (graphics or text) to a *creation mode*. Mouse movement then creates objects of the chosen type.

Exiting Creation Mode

When you have finished creating objects of a particular type, you exit the creation mode. There are two ways to exit a creation mode:

1. Press ESC.
2. Click the mouse anywhere in the menu bar (with or without executing a command).

When you do one of these, the editor leaves the creation mode, and returns to whichever editing mode it was in before you began creating. For brevity, the rest of this user guide specifies using ESC to exit a creation mode. If you prefer, you can click in the menu bar instead.

Drawing Tools

When the editor is in a creation mode, and you are not performing any particular activity, the mouse pointer is a tool that indicates the mode. As you go through the various steps that create an object, the mouse pointer changes to various other tools to indicate the current step, then reverts to the relevant creation tool when the step is complete. Specific drawing tools used during object creation are described in the following sections.

Autoscrolling

Usually you can use the mouse and the scroll bars to move to any part of a page. During some operations for creating and editing graphical objects, the mouse becomes dedicated to a particular purpose, and could not be used for scrolling without disrupting the operation's progress.

If you begin creating or editing a graphical object, then discover that you want to extend the operation to a location not currently visible in

the window, move the mouse pointer off the edge of the window in the direction of the location, and the window will scroll automatically. When the desired location is in view, move the mouse pointer back onto the window, and scrolling will stop.

If you move the mouse pointer off the window without intending to autoscroll, but the window begins autoscrolling, the reason is that you have inadvertently begun some editor operation but have not completed it. Complete or cancel the operation, and the autoscrolling will cease to occur.

Resetting the Drawing Environment

If at any time you find yourself in a state that you don't know how to get out of, reset the drawing environment by doing the following:

Press ESC. This cancels any object creation sequence that is in progress.

Pull down the **Text** menu. If its first item is **Turn Off Text**, choose that item. Repeat until the first item in the menu is **Turn On Text**, then leave the menu without choosing anything. This cancels any text-editing operation that may be in progress.

Press the DELETE key until a warning appears that “This command cannot be applied when the active page is empty.” This deletes any leftover objects.

These steps leave the editor in graphics mode with no operations in progress and an empty page on display.

CAPS LOCK Under X-Windows

Under X-Windows, setting CAPS LOCK changes the behavior of the editing environment in various ways, as described later in this chapter. Except where noted, the instructions in this chapter assume that CAPS LOCK is off. If you find that the results of following the instructions differ from those that are described, check that CAPS LOCK is still off: a finger slip may have set it.

Working With Rectangles

This section shows you how to create, reshape, move, add text to, and delete rectangles.

Creating Rectangles

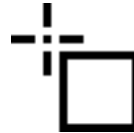
To create a rectangle, execute the following steps:

Enter Rectangle Creation Mode

Choose **Box** from the **Aux** menu

Move the mouse pointer over the page on which you will draw the rectangle.

The editor enters *rectangle creation mode*; the mouse pointer changes to the *rectangle tool*:



The dot in the tool indicates the specific pixel that the tool points to.

Specify the First Corner

Position the rectangle tool anywhere over the page, then depress and hold the mouse button.

(Generic references to “the mouse button” in this user guide refer to the left button.)

When you depress the mouse button, the editor does three things:

1. Draws a small square whose upper right corner is at the location of the rectangle tool.
2. Changes the rectangle tool to the *adjustment tool*:



3. Positions the adjustment tool to point at the lower left corner of the square.

The purpose of the adjustment tool is to set or change the shape of a graphical object.

Specify the Diagonal Corner

Holding the mouse button depressed, move the adjustment tool around on the page.

As the tool moves, the corner at which it points moves with it. This allows you to give the rectangle whatever dimensions you like. You can position the corner in a location that is not currently visible in the window by moving the tool off the window in the direction of the location. The window will autoscroll until you move the tool back onto the window.

Finish the Rectangle

To finish the rectangle:

Release the mouse button.

A small editing box appears that just covers the rectangle. The box has scroll bars and a text insertion cursor. You can use this box to write text inside the rectangle.

Leave Rectangle Creation Mode

To leave rectangle creation mode:

Press ESC.

The rectangle tool is replaced by the graphics tool, and eight small black squares appear on the rectangle, one at each corner and one at the center of each side.

Look at the left side of the status bar. It now displays Auxiliary Node, indicating that the rectangle you just created is not a CPN object, and is a node.

Reshaping Rectangles

The black squares on a rectangle are called *handles*. They can be used to modify a rectangle's shape.

Position the graphics tool over one of the rectangle's handles.

Depress and hold the mouse button.

The adjustment tool appears next to the handle.

Move the adjustment tool around on the page.

The handle tracks the tool, and the rectangle changes shape appropriately.

Release the mouse button.

The graphics tool reappears.

Moving Rectangles

A rectangle once created may be moved anywhere on the page:

Move the graphics tool to the interior of the rectangle.

Depress and hold the mouse button.

Move the mouse pointer around on the page.

The rectangle tracks the mouse pointer, keeping the same position relative to it that it had when you depressed the mouse button.

Release the mouse button.

The rectangle remains in the position to which you have moved it.

Deleting Rectangles

To delete a rectangle:

Press the DELETE key.

The rectangle disappears.

Moving Rectangles During Creation

It might be that the point at which you place the initial corner of a rectangle turns out not to be the correct location. You could finish drawing the rectangle, and then move it; or you can move it while you are still drawing it, by using the SHIFT key.

If you press and hold SHIFT while you are using the adjustment tool to define the shape of a rectangle, the rectangle will stop changing shape as you move the tool. Instead it will track the tool by moving as a whole, as if it were tracking the drag tool. When you release

SHIFT, the rectangle will go back to changing shape as the tool moves.

Adding Text to Rectangles

You can add textual information to rectangles by switching from graphics mode to text mode. To do this:

Choose **Turn On Text** from the **Text** menu.

Four things happen:

1. The mouse pointer becomes the text tool.
2. The handles disappear from the edges of the rectangle.
3. A small editing box appears that just covers the rectangle. The box has scroll bars and a text insertion cursor.
4. The status bar displays Text: On.

You can now perform simple text editing operations. You can enter characters via the keyboard, and delete them by pressing DELETE. If you move the text tool onto the rectangle, it will change to an I-beam. The I-beam can be used to position the insertion cursor, and to select sections of text; these sections can be copied, cut, and pasted.

The editing box is part of the X-Windows environment; it is not specific to Design/CPN. Therefore you can use all three mouse buttons to manipulate its scroll bars and edit its text, using the techniques characteristic of the X-Windows user interface.

Creating a Series of Rectangles

You don't have to enter and leave rectangle creation mode once for each rectangle you create. Once you are in it, you can create as many rectangles as you like, by repeating the sequence:

Depress and hold the mouse button,

Use the adjustment tool to position the diagonal corner,

Release the mouse button,

once for each rectangle.

Adding Text to Rectangles During Creation

When you create a rectangle, an editing box with scroll bars and a cursor appears over the rectangle after you specify the second corner. This facility exists as a convenience to allow immediate text entry.

Immediately after you create a rectangle, the status bar displays Text: On even though you did not explicitly enter text mode. The editor has automatically entered a special form of text mode, called *creation text mode*, to give you the opportunity to enter text during the rectangle creation process. It will leave creation text mode as soon as you do anything other than enter or edit text, so you don't have to leave it explicitly.

Preserving a Rectangle's Aspect Ratio

A rectangle's aspect ratio is the ratio of its height to its width. It is sometimes useful to preserve the aspect ratio while adjusting a rectangle. For example, suppose the rectangle is a square, and you want to change its size while maintaining its squareness: it would then be inconvenient to have the height and width vary independently during the adjustment process.

To preserve a rectangle's aspect ratio during either the adjustment phase of rectangle creation, or later reshaping of the rectangle, set CAPS LOCK before you begin adjusting its shape, or at any time during the adjustment process. While CAPS LOCK is set, the rectangle's aspect ratio is fixed, and motion of the adjustment tool changes only the rectangle's size. When you release CAPS LOCK, the aspect ratio ceases to be fixed, and again varies with motion of the adjustment tool. You can set and release CAPS LOCK as desired during the shape-adjustment process.

Working With Ellipses

Working with ellipses is very similar to working with rectangles. This section shows you how to create, reshape, move, and delete ellipses.

Creating an Ellipse

To create a single ellipse, execute the following steps:

Enter Ellipse Creation Mode

Choose **E**llipse from the **Aux** menu.

Move the mouse pointer over the page on which you will draw the ellipse.

The editor enters *ellipse creation mode*; the mouse pointer changes to the *ellipse tool*:



Specify the First Corner

Position the ellipse tool anywhere over the page, and depress and hold the mouse button.

The editor draws a small circle that fills an invisible square whose upper right corner is at the location of the ellipse tool, switches to the adjustment tool, and positions it to point at the lower left corner of the invisible square.

Specify the Diagonal Corner

Holding the mouse button depressed, move the adjustment tool around on the page.

As the tool moves, the invisible corner at which it points moves with it, and the ellipse changes shape so that it always fills the invisible rectangle. Autoscrolling works as with rectangles.

Finish the Ellipse

To finish the ellipse:

Release the mouse button.

The ellipse tool reappears, and an editing box appears over the ellipse. It can be used to write text inside the ellipse.

Leave Ellipse Creation Mode

Press ESC.

The ellipse tool is replaced by the graphics tool, and eight handles appear on the invisible rectangle that the ellipse fills. These can be used to reshape the ellipse.

Other Operations With Ellipses

All the operations that work with rectangles work with ellipses in exactly the same way. The reason ellipses and rectangles are so similar is that an ellipse is completely defined by the rectangle that contains it. See the previous section, “Working with Rectangles,” for details.

CAPS LOCK preserves an ellipse's aspect ratio just as it does that of a rectangle.

Working With Labels

A *label* is a node that consists entirely of text; it has no associated geometric form. For editing purposes, a label is no different from the text field that is associated with every rectangle or ellipse. Its lack of an accompanying geometric form is its only unusual property.

Creating Labels

To create a label, execute the following steps:

Enter Label Creation Mode

Choose **Label** from the **Aux** menu.

Move the mouse pointer over the page on which you will draw the label.

The editor enters *label creation mode*; the mouse pointer changes to the *label tool*:



Create the Label

Position the label tool anywhere over the page, then click the mouse.

A text insertion cursor appears at the location where you clicked the mouse.

Enter and Edit Text

You can now enter and edit text just as if the editor were in text mode, and you were editing the text field of a rectangle or ellipse.

Type and edit several short lines of text.

Note that there are no scroll bars. Since a label consists only of its text, there is no frame in which the text must fit, and hence no need for scrolling.

Other Operations With Labels

To edit a label's text after the label has been created, select the label in text mode. You can either select it first or enter text mode first.

For most purposes, labels are the same as rectangles and ellipses, and can be treated in the same way. The only differences are:

- You can't move a label while you are creating it, because its creation is a one-step process.
- You can't reshape a label in graphics mode, since it has no geometric form: its shape is just the shape of the text it consists of, and changes only when you edit that text.
- An empty label would serve no purpose, so the editor deletes any label that contains no text.

Working With Connectors

Objects of the types we have worked with so far can exist independently of other objects. In Design/CPN parlance, such objects are called *nodes*.

A *connector* is an arrow that runs between one node and another, establishing a directed link between them. A connector cannot exist independently: it can exist only when there are two nodes for it to connect. Every Design/CPN auxiliary object is either a node or a connector that links two nodes.

This section shows you how to create, reroute, edit, and delete connectors. In general, the methods are the same as for other objects.

Creating Connectors

To create a connector:

Choose **Connector** from the **Aux** menu.

The editor is now in *connector creation mode*, and the mouse pointer is the *connector tool*:



Position the connector tool in the interior of some node.

Depress and hold the mouse button.

Move the connector tool to the interior of some other node.

Release the mouse button.

A connector now points from the first node to the second. The editor is still in connector creation mode, so you can create additional connectors.

Look at the left side of the status bar: it displays Auxiliary Connector, because that is the type of the object you have just created.

Routing Connectors

When there are many objects on a page, the best route for a connector may not be a straight line. You can make a connector follow any path you want.

Position the connector tool inside a node, and depress and hold the mouse button.

Move the tool to a location not inside any node, and release the mouse button.

Move the tool around without depressing the mouse button.

The location at which you released the mouse button becomes a fixed point. As you move the tool, the connector tracks it, extending between the fixed point and the current connector tool location.

Click the mouse somewhere outside any node.

Another fixed point results. You can repeat this sequence as often as you like, creating an arbitrarily complex connector. The fixed

points are called *vertexes*. The lines between them are called *segments*.

Editing Connectors

To reroute a connector, drag its handles. If you drag the point at the center of a segment, resulting in a new vertex and two new segments, new handles appear on the two new segments.

You can use the handles at either end of a connector to detach it from one node and attach it to another: just drag the handle to the edge or interior of the node you want to connect to.

You can edit connectors in most of the ways that you can edit objects generally. You can make a connector current in text mode and add text to it, delete it with DELETE, etc.

Automatic Rerouting of Connectors

If you move a node that has an attached connector, the connector automatically reroutes itself so that it remains connected to the node.

No algorithm for automatic rerouting can give ideal results in all cases, so you will sometimes need to manually adjust a connector's route after automatic rerouting has occurred.

Deletion of Dangling Connectors

If you delete a node that has any attached connector, the connector no longer has a node at each end. Such a connector is illegal; the editor deletes it automatically.

Working with Other Types of Objects

Rectangles, ellipses, labels, and connectors are sufficient for representing all CP net components, as described in the next chapter. To aid in documenting CP nets, Design/CPN also provides many other graphical objects. These are described in Chapter 23, "Aux Menu Commands."

Creating Regions

It is frequently useful to define a node as being logically subordinate to some other object. Such a subordinate node is called a *region*. A node that is a region does not cease to be a node. It just takes on some additional properties that make it a region as well.

An object may have any number of regions. These may in turn have regions, sometimes called subregions, and so on indefinitely. An object and any associated regions always form a tree; non-tree-structured relationships may not be defined. Objects and their regions are referred to as *parents* and *children*, as is customary with tree structures.

When a node is a region, it is affected not only by operations performed on itself, but also by operations performed on its parent. For example, when a region's parent is deleted, the region is automatically deleted as well. Details appear later in this section.

Both nodes and connectors can have regions, but only a node can be a region: a connector cannot be. The reason is that allowing a connector to be a region would lead to unresolvable contradictions between its role as a link and its role as a subordinate object.

Designating Regions

You can designate a node to be a region of an object whenever the editor is in graphics mode or text mode. The operation of designating a region does not affect the editing mode.

To designate a node as a region:

Select the node that will be the region.

Choose **Make Region** from the **Aux** menu.

The status bar displays Select Parent, and the mouse pointer becomes the *pointer tool*:



Move the tool so that it is on or inside the object that is to be the parent of the region.

A dark border flashes on and off around the prospective parent. An object that could not legally become the parent, because the result would not be a tree, will not show such a border.

Click on the object that is to be the parent.

The selected node is now a region of the object you designated as its parent.

You can execute **Make Region** on a node that is already a region. It will cease to be a region of its current parent, and become instead a region of the newly designated parent.

Converting Regions into Nodes

You can disconnect a region from its parent, restoring its independent node status, whenever the editor is in graphics or text mode. If the region has subregions, their status relative to the region is not affected by its disconnection.

Select a region that is to be returned to independent status.

Choose **Make Node** from the **Aux** menu.

The region is returned to independent status. It has the same properties that it would have if it had never been a region.

Editing Parents and Regions

Editing a region does not affect its parent, but editing the parent can affect the region in a number of ways:

Moving Parents

When a region's parent is a node, and the parent is moved, the region moves with it so that they retain the same relative positions.

When a region's parent is a connector, the situation is complicated by the fact that a connector (unless it is a straight line) has no center point that can be taken as its position. Consequently there is no way to define how rerouting a connector would move a region. Rerouting a connector therefore does not affect region position.

The position of a connector is defined as the midpoint of a straight line connecting its endpoints. When that point moves, because one of the connected nodes moves, any regions will move also.

Deleting Parents

When a region's parent is deleted, the region is automatically deleted along with it.

The deletion of the node deletes the connector, and the deletion of the connector deletes the region.

Creating Objects From Text Mode

You can enter a creation mode directly from text mode. While the editor is in the creation mode, it makes no difference that it got there from text mode. The only difference is that when you leave the creation mode, the editor will return to text mode, since that is where it started. The editor's automatic activation and deactivation of creation text mode during object creation is orthogonal to the underlying text mode.

Selecting Graphical Objects

When there is more than one graphical object on a page, and you want to edit one of them, you must tell the editor which one, or it will not know where to apply your instructions. The way to give the editor this information is to *select* the object.

Current Objects

The object that is currently selected is called the *current object*. A description of the current object (e.g. Auxiliary Node) is displayed on the left side of the status bar. If the editor is in graphics mode, handles appear around the current object, so you can change its shape. In text mode, an editing box appears on top of it, so you can edit its text field.

When you create a new object, it automatically becomes the current object.

Selecting Objects

When a page contains only one object, that object is always selected. When there is more than one object, and the object you wish to edit is not selected, click the mouse anywhere on or inside the object of interest. That object is thereby selected, and the object that was selected previously ceases to be selected.

You can select an object whenever the editor is in graphics or text mode. Its appearance (with handles or covered by an editing box) will become the one appropriate to the mode. When you change modes, the current object will change its appearance accordingly.

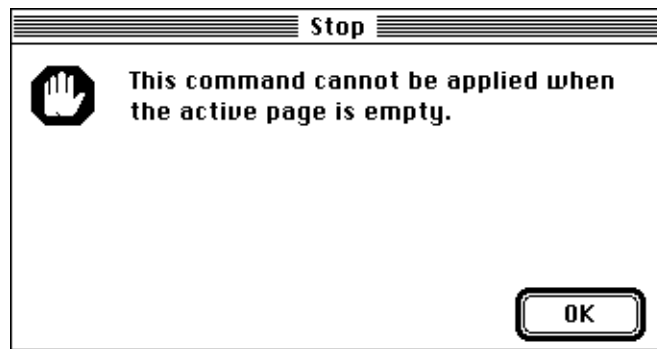
Selection After Deletion

When there is more than one object on a page, the editor keeps them in an order, called the *occlusion order*. When an object is created, it is put at the front of the occlusion order.

The occlusion order is used primarily to control what happens when objects with opaque interiors are drawn on top of each other. The objects you create in this chapter are all transparent, so the occlusion order does not affect their display appearance.

Another use of the occlusion order is to determine what object will become selected when the current object is deleted. The rule is: when an object is deleted, the object at the front of the occlusion order becomes current. This behavior can be useful when you have created several objects in sequence, but then decide to delete them all and try a different approach.

If you press DELETE when there are no objects on the page, a dialog appears:



Working With More Than One Object Type

When a page contains objects of more than one type, they are all kept in a single occlusion order.

You don't have to leave one object creation mode before you enter another. You can go directly from one to another by choosing the new mode while you are still in the old one.

Groups of Objects

It is frequently necessary to perform the same operation on more than one object, and it would be inefficient to have to select and then operate on them one at a time. Therefore the editor allows you to designate a *group* of objects and operate on them simultaneously.

There is then no one selected object: all the objects in the group are selected.

When a group has been selected, the editor indicates the group by drawing a gray border around each of its members. Handles do not appear around group members, since they intrinsically refer only to individual objects.

Whenever you select more than one object simultaneously, the editor enters *group mode*. This mode is not an alternative to graphics or text mode; it is orthogonal to them.

When the editor is in group mode, the mouse pointer becomes the *group tool*:



This tool is displayed whether the editor is also in graphics or text mode. Examine the status bar to tell which mode is in effect.

Mixed Groups

The properties of nodes, connectors, and regions are so different that it is not possible to create a useful group that combines objects from more than one of these classes. The reasons are:

- Whenever a node is changed, any connectors and regions associated with it automatically adjust themselves. Therefore there is never any reason to include connectors and regions in a group that contains nodes.
- Allowing connectors and regions to be part of the same group could lead to unresolvable conflicts among the various operations that are performed on them automatically when their associated nodes are edited.

Therefore when more than one object is selected at a time, they must all be nodes, or all connectors, or all regions.

Selecting Groups

There are many ways to select a group. None of them can be used to create a mixed group.

- The **Group** menu contains the commands **Select All Nodes**, **Select All Regions**, and **Select All Connectors**. These have the obvious effects. They may be

combined with mouse and keyboard techniques to further specify the group.

- Pressing and holding SHIFT and then clicking on an object selects it (if it was not selected and is of appropriate type) or deselects it (if it was selected) without affecting the selection status of other objects.
- Depressing the mouse button when it is not inside an object, then moving it, creates a rectangular selection area. When the mouse button is released, all objects in the area (that are not selected and are of appropriate type) will become selected, and any other selected objects will cease to be selected.
- Pressing and holding SHIFT, then using the mouse to create a rectangular selection area, toggles the selection status of all objects in the area (except that none becomes selected that is not of appropriate type) without affecting that of other objects.

Whenever a group rather than a single object is current, the status bar displays a description of the content of the group.

Deselecting Groups

When the editor is in group mode, the command **Ungroup** is available in the **Group** menu. If you choose this command, the group will cease to be selected, and the object that was current before you entered group mode will become current again.

A group is also deselected if you select some other group in its place, or use the mouse to select a single object that is not a member of the group.

Reconstructing Groups

Sometimes a particular group will be useful more than once during an editing session, but is not useful continuously. To save you the effort of reconstructing such a group each time you need it, the editor always records the composition of the group that is current when you leave group mode. It can use this information later to reconstruct the group.

When the editor is not in group mode, the command **Regroup** is available in the **Group** menu. If you choose this command, and you had previously defined a group at some time, the editor will re-enter group mode and reselect the group.

Operating on Groups

You can do anything to a group of objects that:

- Could be done to each of its members individually, and:
- Is meaningful when applied to more than one object simultaneously.

For example, you can move a group, delete a group, make all the members of a group regions of the same parent, or turn a group of regions back into nodes even if they had different parents. But you cannot edit the text in a group of nodes, or reroute a group of connectors, since there is no way to deal with text fields or connector routes generically.

There are also operations that are meaningful for groups but not for the individuals within them. Most of them concern the physical layout of the group's members on the page, and are used to improve the physical appearance of a net.

Aligning Graphical Objects

The **Align** menu provides commands that position graphical objects in relation to each other. Different types of nodes can be mixed freely in any alignment operation.

Horizontal Spread Command

Horizontal Spread evenly spaces graphical objects horizontally between two designated objects. To use it:

Select the nodes to be horizontally spread. Be sure nothing else is selected.

Choose **Horizontal Spread** from the **Align** menu.

You can now designate two nodes, called *reference nodes*, between which the selected objects will be evenly spread. The status bar prompts for the first reference node by displaying: Please select node as reference for alignment. The reference node does not have to be one of the nodes in the selected group.

Move the mouse pointer to the interior of node that will be the first reference node.

A dark border flashes on and off around the node.

Click the mouse on the node.

The status bar prompts for the second reference node by displaying:
Please select second node.

Click the mouse on the node that will be the second reference node.

The nodes are now evenly spaced horizontally.

Vertical Spread Command

Vertical Spread is exactly like **Horizontal Spread**, except that it spaces graphical objects vertically rather than horizontally.

Horizontal Command

Horizontal aligns nodes horizontally, so that the centers of the nodes lie exactly in a row. The row passes through the center of a particular node called a *reference node*. To use the command:

Select the node, or create a group that contains all the nodes, that are to be aligned horizontally. This group may contain the reference node, but does not have to.

Choose **Horizontal** from the **Align** menu.

Click the mouse on the desired reference node.

The selected node(s) are now horizontally aligned with the reference node.

Vertical Command

Vertical is exactly like **Horizontal**, except that it aligns nodes into a column rather than a row.

Other Alignment Commands

The **Align** menu also contains many other commands for aligning graphical objects. For details, see Chapter 29, “Align Menu Commands.”

Undoing Alignment Commands

If you don't like the results of any command in the **Align** menu, you can undo the operation by choosing **Undo** from the **Edit** menu.

Matching Object Sizes

It is difficult to get different graphical objects to have exactly the same size and shape using the mouse. Design/CPN therefore provides an easy way to make one graphical object look exactly like another.

If necessary, reshape one of the objects so that it has the size and shape you want them all to have.

Select the object, or make a group that contains all the objects, that are to be reshaped.

Choose **Change Shape** from the **Makeup** menu.

The status bar displays: Select Node or Region as a model for shape change.

Move the mouse over an object that has the desired shape.

The object's border flashes on and off.

Click the mouse.

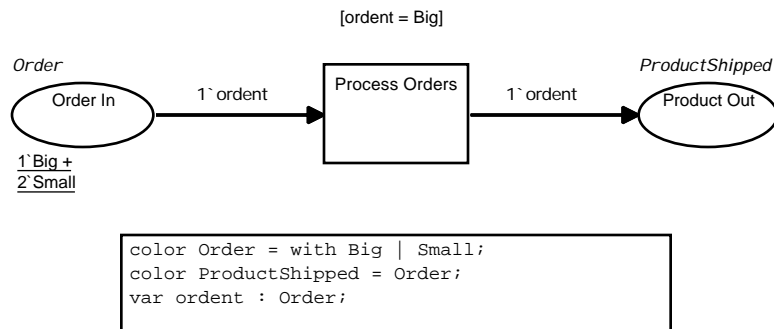
The selected object(s) are reshaped to be identical to the one that you clicked on.

This technique may be applied to any graphical object except a connector.

Chapter 4

The Design/CPN Editor: Creating a CP Net

This chapter shows you how to use the Design/CPN editor to create a CP net. The various techniques are demonstrated by tracing the creation of a net called SmallNet:



Auxiliary Graphics and CPN Graphics

You could draw graphics that look exactly like SmallNet net using the techniques covered in the previous chapter. But the result would not be a CP net: it would just be a collection of auxiliary graphical objects.

The Design/CPN editor provides commands that let you draw a graphical object, make it a region if appropriate, specify its CPN meaning, and give it display attributes appropriate to that meaning, using only one command per object. These commands are available from the **CPN** menu. This chapter shows you how to use them.

A graphical object that is also a CP net component is called a *CPN graphical object*, which is usually shortened to *CPN object*. A node that is also a CP net component is called a *CPN Node*. A region that is also a CP net component is called a *CPN Region*. As these names suggest, the realms of auxiliary objects and of CPN objects are exactly parallel. The only difference between an auxiliary object and the corresponding CPN object is the presence or absence of a CPN meaning: graphically the objects are identical.

This chapter does not reiterate the information about creating and manipulating graphical objects that was covered in the previous chapter. Refer to that chapter as needed to obtain or review this information.

Setting the Graphical Environment

A CP net contains many kinds of information. All of them are represented in some way in the physical appearance of the net. To help make nets and their behavior during execution visually self-explanatory, different types of information are customarily given different physical appearances, using a variety of graphical and textual conventions. These conventions are called *display attributes*. Their combined effect is to establish a particular *graphical environment*.

Display attributes exist at three levels: object attributes, diagram default attributes, and system default attributes.

Levels of Attributes

Object Attributes

Each graphical object has a set of display attributes that governs its individual appearance. These are the *object attributes*.

Diagram Default Attributes

Each diagram has a set of display attributes that it applies by default to every newly created graphical object. These are the *diagram default attributes*. For brevity they are usually called the *diagram defaults*.

System Default Attributes

Design/CPN keeps a master list of display attributes that it copies by default into every newly created diagram. These are the *system default attributes*. For brevity they are usually called the *system defaults*. (This “system” has nothing to do with the computer's operating system; it refers only to Design/CPN's internal record-keeping system.)

Changing Display Attributes

Display attributes can be changed whenever a net is not being executed. The change can be applied to a selected object or group of

objects, to the diagram default attributes, to the system default attributes, or to any combination of these.

It is also possible to replace all of a diagram's default attributes with the current system default attributes, or vice versa.

Establishing an Environment

To establish a new system environment, copy diagram defaults from an existing diagram into Design/CPN's system defaults, as follows:

Open the diagram containing the diagram defaults.

Preserving the ML Configuration Options

When Design/CPN is installed, one of the steps is to record in the system defaults some information that Design/CPN needs in order to communicate with the ML process that was described at the beginning of this chapter. This information must be preserved when a new system environment is established. To preserve it:

- Choose **ML Configuration Options** from the **Set** menu.

A dialog appears. One of its options is **Load**.

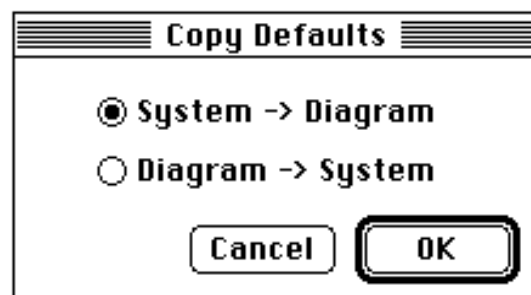
- Click **Load**.
- Click **OK**.

The necessary information about the ML process is copied from the system defaults to the existing diagram's diagram defaults. You can now make those diagram defaults the system defaults without erasing the information about the ML process.

Copying the Diagram Defaults

Choose **Copy Defaults** from the **Set** menu.

The **Copy Defaults** dialog appears:



Click **Diagram -> System**.

Click **OK**.

The dialog disappears. The system defaults are now the same as the diagram defaults.

Creating CP Nets

The components of a CP net can be created in almost any order. The only requirements are:

1. A region's parent must be drawn before the region is drawn.
2. The nodes that an arc connects must be drawn before the arc is drawn.

Within these requirements, the choice of how to create a net is a matter of individual style.

General Technique for Creating CPN Regions

Each of the CPN objects described in this chapter has associated regions. The technique for creating all of these regions is the same:

Select the object that is to have the region.

Choose **CPN Region** from the **CPN** menu.

Indicate the desired type of region.

Type in the region.

Adjust appearance as needed.

The section “Creating Regions” in Chapter 3, “The Design/CPN Editor: Introduction.” discusses regions and their use.

Creating a Transition

From the graphical standpoint, creating a transition is the same as creating a rectangle. The only difference is in the command you use to specify its creation.

Choose **Transition** from the **CPN** menu.

The editor enters *transition creation mode*. The *transition tool* appears:



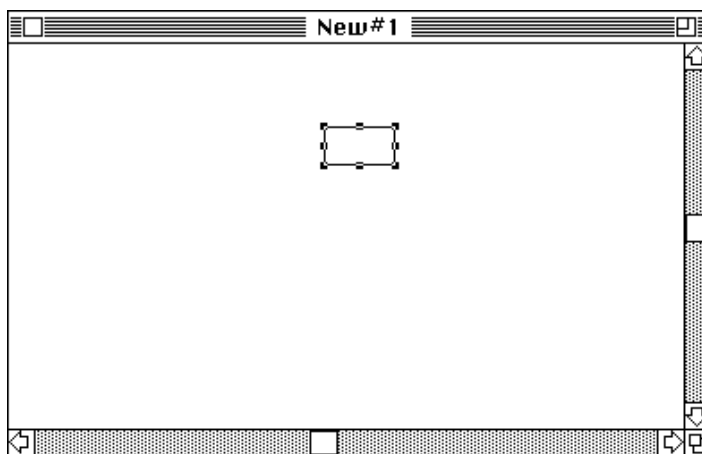
In transition creation mode, you can draw rectangles that are predefined as representing transitions.

Draw a rectangle (transition).

A small editing box appears over the rectangle, just as with auxiliary rectangle creation (described in Chapter 3, “The Design/CPN Editor: Introduction”). If you entered text, and did not specify a separate name region later, the text would by default become the name of the transition. If you did create a name region later, the text would thereafter have no functional significance. Such text is often used as a comments field.

Leave transition creation mode (Press ESC or click the mouse in the menu bar).

The rectangle you have just drawn looks like any other, and has all the usual graphical properties of a rectangle:



The transition acts exactly like an ordinary rectangle. Graphically it is a rectangle. Its CPN status as a transition has nothing to do with its graphical nature. However, the status bar shows that you have drawn a transition, not just a rectangle. The editor has automatically performed the additional operations needed to designate the rectangle as a transition.

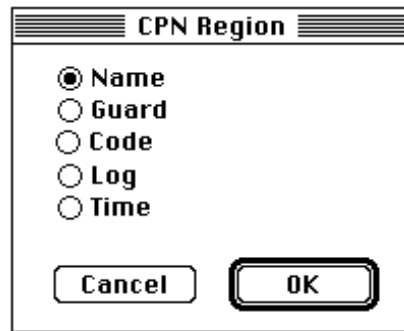
Naming a Transition

To name a transition, attach a label region to it and type the name into the label. This is done with a single command. When there is more than one node on the page you must first select which one you want to name.

Select the transition to be named.

Choose **CPN Region** from the **CPN** menu.

The **CPN Region** dialog for transitions appears:



The default is **Name**, which is what we want, so:

Click **OK**.

The dialog disappears. The editor enters *name region creation mode*. The mouse pointer becomes the *region tool*:



This tool is exactly like the label tool, except that the result of using it is a CPN region of whatever kind you have indicated, rather than a generic label.

Move the tool to the inside top of the transition and click the mouse.

Type the name. Example: `Process Orders`.

Leave name region creation mode.

The result looks like a label, and has all the graphical properties of a label, but the status bar shows that it is also a name region. The editor has automatically performed the additional operations needed to designate the label as a region that specifies a name.

Creating a Guard

Creating a guard is just like creating a name, except that you specify a guard region rather than a name region in the **CPN Region** menu.

Select the transition.

Choose **CPN Region** from the **CPN** menu.

The **CPN Region** dialog for transitions appears.

Click **Guard**.

Click **OK**.

The dialog disappears. The editor enters *guard region creation mode*. The mouse pointer is again the region tool.

Move the tool to the desired guard location and click the mouse.

Type the expression enclosed in square brackets. Example:

```
[ordent = Big].
```

Leave guard region creation mode.

Creating a Place

Creating a place is graphically the same as creating an ellipse. The only difference is in the command you use:

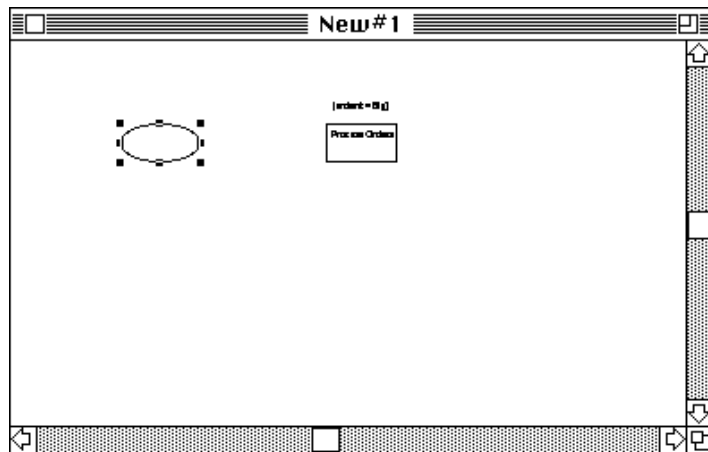
Choose **Place** from the **CPN** menu.

The editor enters *place creation mode*. The *place tool* appears.



Draw an ellipse (place).

The status bar shows that you have drawn a place, not just an ellipse.



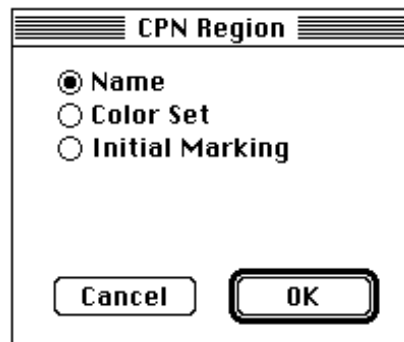
Naming a Place

Naming a place is essentially the same as naming a transition.

Select the place.

Choose **CPN Region** from the **CPN** menu.

The **CPN Region** dialog for places appears:



The default is **Name**, which is what we want, so:

Click **OK**.

The dialog disappears. The editor enters name region creation mode, and the mouse pointer becomes the region tool.

Move the tool to the inside top of the place and click the mouse.

Type the name. For example: `Order In.`

Leave name region creation mode.

The status bar shows that the region is a name region.

Specifying a Place's Colorset

Specifying a place's colorset is just like specifying its name, except that you specify a colorset region rather than a name region in the **CPN Region** menu.

Select the place.

Choose **CPN Region** from the **CPN** menu.

Click **Color Set**.

Click **OK**.

The editor enters *colorset region creation mode*; the mouse pointer becomes the region tool.

Move the tool to the desired location for the colorset region and click the mouse.

Type the colorset name. For example: `Order`.

Leave colorset region creation mode.

Specifying a Place's Initial Marking

A place's initial marking is just another CPN region.

Select the place.

Choose **CPN Region** from the **CPN** menu.

Click **Initial Marking**.

Click **OK**.

The editor enters *initial marking region creation mode*; the mouse pointer becomes the region tool.

Move the tool to the desired location for the initial marking region and click the mouse.

Type the initial marking. For example: `1`Big + 2`Small`.

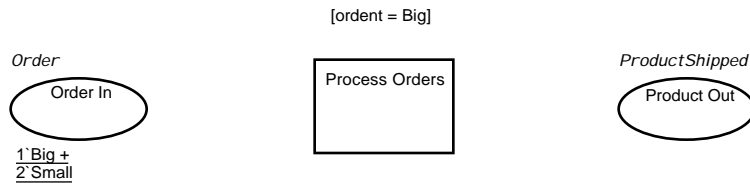
Be careful to use backquote (the multiset creation operator) not single quote.

Leave initial marking region creation mode.

Input and Output Places

Input and output places are graphically identical, and are created in the same way. Whether a place is an input place, an output place, or both, is determined by the arcs connected to the place, not by the place itself.

SmallNet has one output place. Its name is `Product Out` and its colorset is `ProductShipped`:



Creating an Arc

Creating an arc is graphically the same as creating a connector. Input and output arcs are graphically identical. The only difference is in the direction of the arc. An input arc runs from a place to a transition; an output arc runs from a transition to a place. To create an arc:

Choose **Arc** from the **CPN** menu.

The editor enters *arc creation mode*. The *arc creation tool* appears:

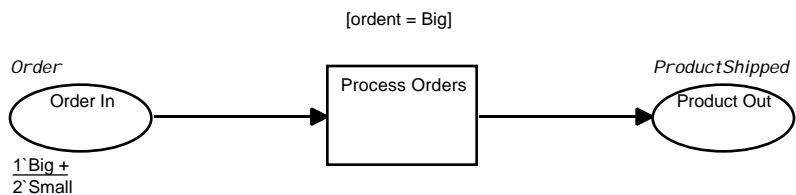


Draw a connector (arc) from a place to a transition, or a transition to a place.

Leave arc creation mode.

The status bar shows that the connector is an arc.

SmallNet contains two arcs:



Creating an Arc Inscription

Creating an arc inscription is graphically the same as adding a text region to a connector. To add an inscription to an arc:

Select the arc.

Choose **CPN Region** from the **CPN** menu.

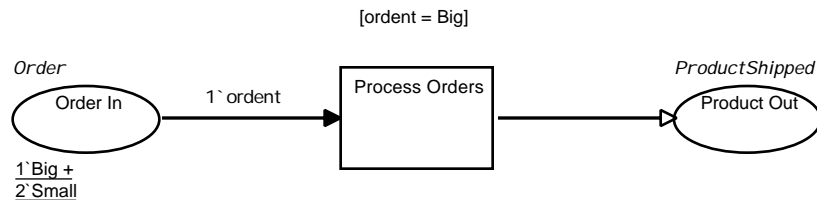
No **CPN Region** dialog appears. None is needed, because an arc can have only one type of region. The editor enters *arc inscription creation mode*, and the mouse pointer becomes the label tool.

Position the label tool in the desired location for the arc inscription.

Type the inscription. Example: 1`ordent. (Backquote, not single quote.)

Leave arc inscription creation mode.

With the addition of 1`ordent as the inscription on the arc from Order In to Process Orders, SmallNet looks like this:



Cloning CPN Regions

When a net needs more than one copy of the same region, you don't have to type them in individually. You can copy a region that already exists, and paste it into as many locations (of appropriate type) as you like. This technique works with all types of CPN text regions. For example, to copy an arc inscription region:

Select the region to be copied.

Execute **Copy**.

Execute **Paste**.

The editor does not paste the region anywhere yet. The mouse pointer becomes the pointer tool, and the status bar displays: Select Arc for next arc inscription region.

Design/CPN User's Guide

Move the mouse over the arc that is to receive a copy of the inscription.

The arc flashes.

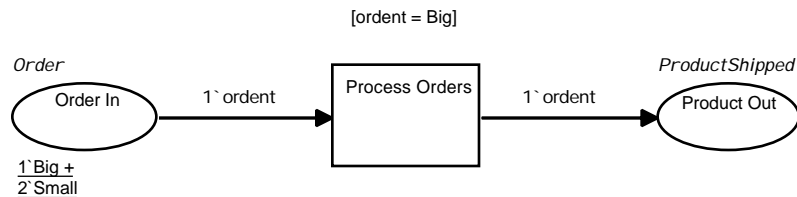
Click the mouse on the target arc.

The copied inscription is pasted.

The problem of where to paste a cloned region has no reliable algorithmic solution. Therefore manual adjustment is sometimes necessary:

If necessary, move the pasted arc inscription region to the correct position for it.

If the techniques described were used to clone the input arc inscription 1`ordent in SmallNet to the output arc between Process Orders and Product Out, SmallNet would look something like this:

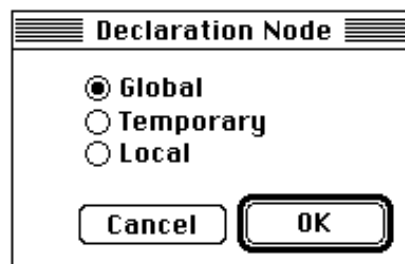


Creating a Global Declaration Node

Graphically, creating a global declaration node is just a matter of drawing a rectangle and typing some text into it. To create a global declaration node:

Choose **Declaration Node** from the **CPN** menu.

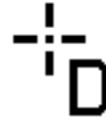
The **Declaration Node** dialog appears:



Global is the default, indicating a global declaration node.

Click **OK**.

The editor enters *global declaration node creation mode*, and the mouse pointer becomes the *declaration node tool*:



This tool draws a rectangle that is also a declaration node.

Draw a rectangle (global declaration node) in the desired location.

Recall that depressing SHIFT while creating a rectangle causes the rectangle to move with the mouse rather than reshaping.

A small editing box appears over the rectangle. Text entered now or later into the declaration node's text field will be treated by Design/CPN as CPN ML code that specifies global declarations. You can stay in the creation mode and enter text immediately, but it is generally more convenient to enter it from text mode.

Leave the creation mode.

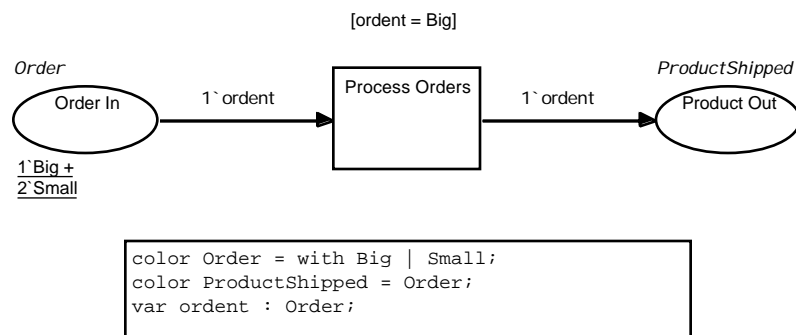
Enter text mode.

Type the desired declarations. For example:

```
color Order = with Big | Small;  
color ProductShipped = Order;  
var ordent : Order;
```

Leave text mode.

You have now created a global declaration node. SmallNet with a global declaration node added might look like this:



Creating a Page for Global Declarations

SmallNet is contained on a single page. This page contains both the data declarations for the net (in the global declaration node) and all graphical structure that makes use of those declarations.

Putting the global declaration node on the same page as a net's graphics can be helpful when you are first learning about CP nets, but otherwise it is not the best practice. As a net grows, the global declaration node ends up competing with it for page space, and has to be moved aside where it can be seen only by scrolling over to it. Furthermore, it is of little interest once you know what is in it. Therefore it is customarily kept on a page of its own, where it can be quickly accessed when needed and ignored otherwise.

Creating a Page

Putting a global declaration node on a page of its own is just a matter of creating the page, and then creating or moving the declaration node there. New pages can be created only in the editor.

Enter the editor if necessary.

Choose **New Page** from the **Page** menu.

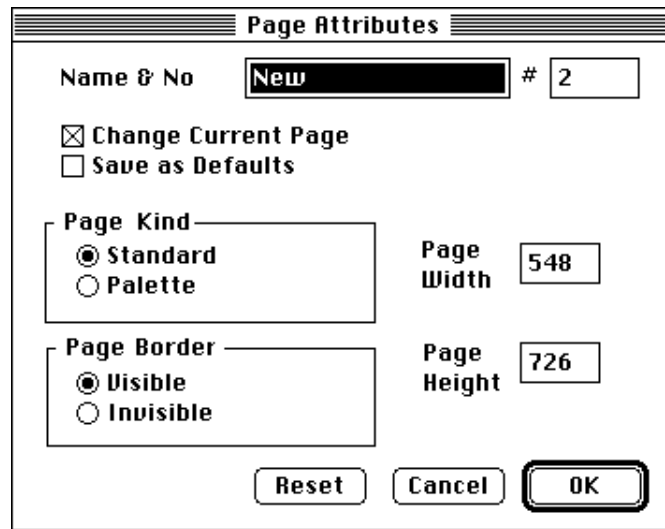
A new, blank page appears.

Naming a Page

The default page name, New, does not tell much about a page's intended purpose. To rename the page:

Choose **Page Attributes** from the **Set** menu.

The **Page Attributes** dialog appears:



The page name field contains the current page name.

Edit the page name field to specify the desired name.

Click **OK**.

Renaming a Page

You don't have to make a page the current page in order to rename it. You can rename it, and make many other changes to its status, directly from the hierarchy page.

Make the hierarchy page the current page.

Select the page node.

Choose **Page Attributes** from the **Set** menu.

Rename the page.

Moving a Global Declaration Node

To move a global declaration node another page:

Make the page where the node is located the current page.

Select the node.

Execute **Cut**.

Choose **Open Page** from the **Page** menu.

The hierarchy page appears.

Double-click on the page node of the page that will receive the declaration node.

Execute **Paste**.

Note that neither putting the global declaration node on its own page nor giving that page any particular name has any functional significance. The global declaration node may appear on any page, and that page may have any name.

Chapter 5

CPN Dynamics: Introduction

CP net components are covered in Chapter 2. This chapter shows how those components interrelate to specify the behavior of an executing CP net.

This chapter does not give complete details on any topic that it covers. Chapter 6, “CPN Dynamics: Executing a CP Net,” provides details about how to set up and execute a CP net. Chapter 8, “CPN Dynamics: Concurrency and Choice,” provides details about the simulator algorithm and its handling of complex execution decisions.

Executing CP Nets

CP nets are actually programs, expressed in a hybrid language of graphics and text. CPN syntax is designed to insure that any legal CP net is completely and unambiguously defined.

CP nets do not execute themselves. Like any program, a CP net is executed. A CP net could be compiled into a stand-alone executable file, and executed directly by the computer. The problem with this method is that the file would have to contain a lot of code to manage the details of net execution, and to provide an interactive interface to it. This code would be the same in each file, which would be very wasteful. Therefore CP nets are executed by a run-time package called the *simulator*.

The Design/CPN Simulator

Design/CPN contains a CP net simulator that is closely integrated with the editor. The simulator both manages and displays CP net execution, and allows its course to be controlled in various ways to facilitate study and debugging.

The simulator is not just a passive intermediary between the computer and an executing net. It is an active agency that drives net exe-

cution forward by investigating the state of the net, determining how to change that state, and making the requisite changes.

When a net is given to the simulator for execution, the simulator first creates tokens as specified by any initial marking regions, and puts the tokens in the places. This establishes the initial state of the net. The simulator then executes the net by identifying transitions that can *occur* and effecting their occurrence.

Understanding CP Net Execution

In order to build an understanding of CP net execution, and of what the simulator does to accomplish it, we must first look at two topics:

- When can a transition occur?
- What happens when a transition occurs?

When Can a Transition Occur?

A transition can *occur* whenever certain conditions are met. When the conditions are met, the transition is said to be *enabled*. The fact that a transition is enabled does not mean that it will actually occur: some other enabled transition might occur first, and change the state of the net so that the first transition is no longer enabled.

Factors Determining Enablement

Three factors work together to determine whether a transition is enabled:

1. The multiset of tokens in each input place of the transition.
2. The input arc inscription on each input arc connected to the transition.
3. The transition's guard.

These three factors work together to determine whether a transition can occur. They work together so closely that none of them can be fully understood without understanding the other two. Chapter 2, “CP Net Components,” provides additional detail on the input place multiset, input arc inscription, and guard.

Input Place Multiset

The multiset of tokens present in an input place determines what tokens are available when the simulator is determining if a transition is enabled.

Input Arc Inscriptions

An input arc inscription is an expression on an arc that connects a place to a transition. The inscription (possibly in conjunction with a guard) specifies a multiset, which may be empty. The default input arc inscription is `empty`.

Guards

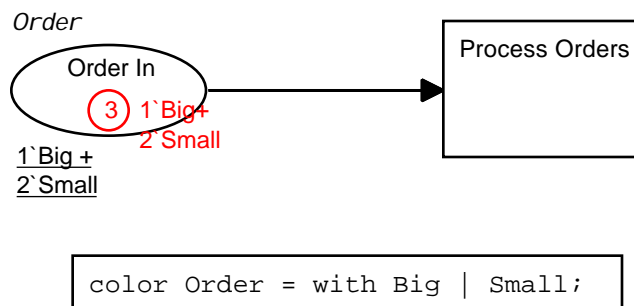
A *guard* is a boolean expression that is associated with a transition. This expression must evaluate to **true** in order for the transition to be enabled. The default guard is **true**.

Criteria for Enablement

If each of a transition's input places contains the multiset specified by the place's input arc inscription (possibly in conjunction with the guard), and the guard evaluates to true, the transition is enabled, and can occur. Otherwise it is not enabled, and cannot occur. A transition's output places have no effect on its enablement.

A multiset whose existence allows a transition to be enabled is called an *enabling multiset*. When an enabling multiset exists, so that a transition is enabled, the transition is said to be *enabled with that multiset*.

The SmallNet example used in Chapter 4, “The Design/CPN Editor: Creating a CP Net,” has more in it than we need for examining transition enablement. Therefore we will start with an even smaller net:



The input place has an initial marking and current marking of 1`Big and 2`Small. There is no guard and the input arc inscription has not yet been specified.

Order In's current marking is shown, since it is a factor in whether `Process Orders` is enabled. From this point on we will show place markings whenever they are relevant to some point being made. But keep in mind that markings are not part of a CP net, as places and transitions are, but rather indicators of its state.

`Process Orders` has no (explicit) guard. Such a guard defaults to **true**, and so can be ignored: its requirement is guaranteed to be satisfied irrespective of other factors.

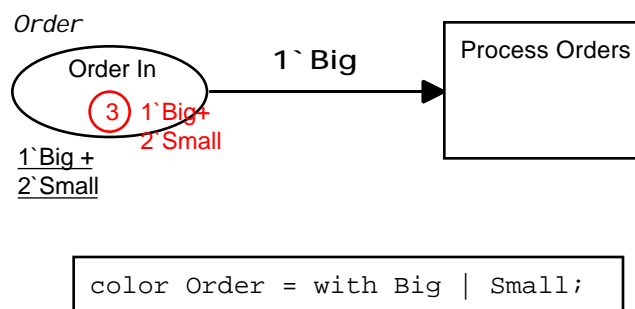
Output places don't affect enablement, so there is no output place in this net. Enablement in this simple case is determined only by the tokens in the input place, and the input arc inscription.

Specifying Exact Token Values

Sometimes the exact token values that are to enable a transition are known in advance. In such a case, input arc inscriptions can specify the needed values literally, and no guard is needed.

Specifying a Single Token

The simplest form of input arc inscription specifies a multiset of one token, and gives the tokens value as a constant:



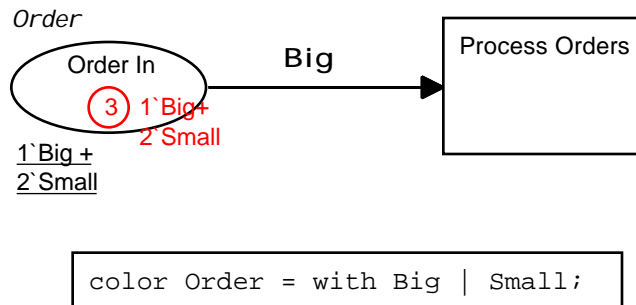
Here the input arc inscription specifies the multiset 1`Big: that is, a multiset of one token whose value is `Big`. Does such a token exist in the input place `Order In`? Yes. Therefore `Process Orders` is enabled (can occur). The fact that there are also two `Small` tokens in `Order In` makes no difference one way or the other.

The Simulator's Algorithm

When the simulator examines the above net to see whether `Process Orders` is enabled, it scans the tokens in `Order In` and tests each one to see whether its value is `Big`. If such a token is found (as it will be in this case), the simulator ends its search at that point, and puts `Process Orders` on a list called the *enabled list*. See “The Simulator’s Algorithm” in Chapter 8, “CPN Dynamics: Concurrency and Choice,” for complete details on the simulator’s algorithm.

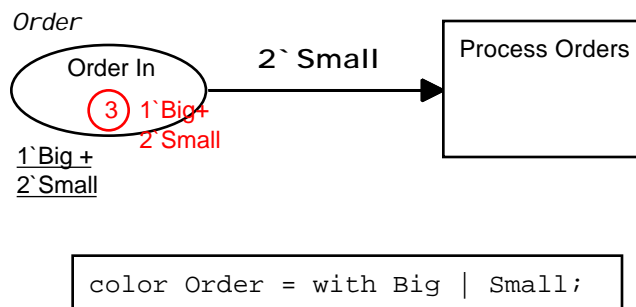
Omitting a Count of One

As a convenience, the count in an arc inscription that specifies just one token can be omitted. Thus the following net is exactly equivalent to the one above:



Specifying More Than One Token Instance

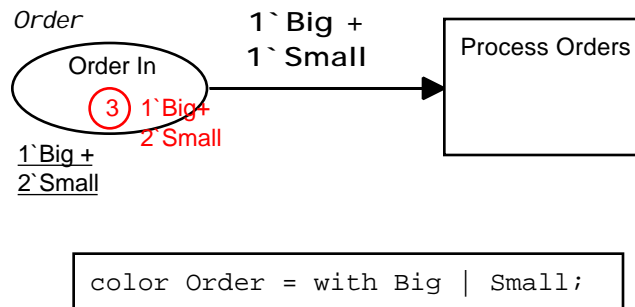
An input arc inscription can specify more than one token of a given value:



Here the input arc inscription requires two tokens of value `Small`. The tokens exist, so the transition is enabled. The simulator will end its search as soon as it encounters the second value `Small` token, and put `Process Orders` on the enabled list.

Specifying More Than One Token Value

An arc inscription can specify tokens of more than one value. For example:



Process Orders is again enabled.

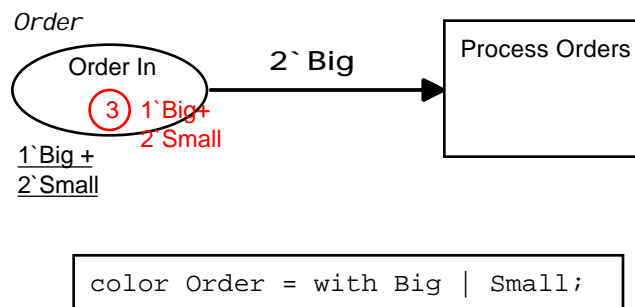
The General Rule

In general, Process Orders will be enabled whenever the multiset specified by the input arc inscription is a subset of the multiset in Order In. Thus any of the following arc inscriptions (some of which we have seen already) will enable Process Orders:

1`Big + 2`Small
1`Big + 1`Small
1`Big
2`Small
1`Small
empty

Non-enabled transitions

The following example illustrates a situation when the transition is not enabled:

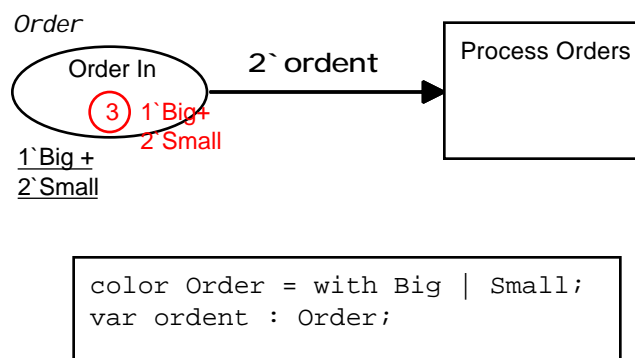


There is only one Big token in Order In, but the arc inscription requires two, so Process Orders is not enabled. There is an in-

finite number of possible non-enabling arc inscriptions. All share the same property: they define a multiset that is not a subset of the multiset in `Order In`.

Specifying Variable Token Values

Sometimes it does not matter exactly what values enabling tokens have: all that matters is that they are present in appropriate numbers. To allow for such a case, input arc inscriptions can use CPN variables rather than constants. For example:



Here `ordent` (which stands for “order entered”) is a CPN variable that can take on any value from the colorset `Order`. CPN variables are discussed in the section “CPN Variables” in Chapter 2, “CP Net Components.”

Binding an Arc Inscription Variable

Initially the CPN variable `ordent` in the example above has no particular value: it is said to be *unbound*. Since `ordent` is unbound, the arc inscription that uses it does not evaluate to a multiset: trying to evaluate it would just produce an error.

The simulator's task at this point is to determine whether `Process Orders` is enabled. To do that it needs to evaluate the arc inscription `2` ordent`, and to perform the evaluation it must have a value for `ordent`. Lacking any other source for a value, it chooses one from among the possible legal values and makes it the value of `ordent`. When this has occurred, `ordent` is said to be *bound* to that value.

Suppose the simulator binds `ordent` to `Big`. The inscription then evaluates to `2` Big`. But there is only one `Big` token in `Order In`, so `Process Orders` is not enabled when `ordent` is bound to `Big`. The simulator does not give up, however: it next tries binding `ordent` to `Small`. The arc inscription then evaluates to

2 'Small. There are two Small tokens in Order In, so Process Orders is enabled when ordent is bound to Small.

A binding that causes a transition to be enabled is called an *enabling binding*, and the transition is said to be *enabled with that binding*. In this case, there is one enabling binding, `ordent = Small`. The simulator will put Process Orders on the enabled list along with a record of the enabling binding.

Constraining Token Values With Guards

We have now looked at a way to specify the values of enabling tokens exactly (with constants), and a way to leave their values unspecified (with variables). Obviously these methods would not be enough in a realistic model. In order to model real systems, we need a way to require token values to meet any criterion we can define. Such a requirement is called a *constraint*.

In order to constrain token values, CP nets use arc inscriptions in conjunction with guards. The method is to bind CPN variables in arc inscriptions, and perform boolean tests on those values in a guard.

Guard Syntax

A *guard* is a boolean expression that operates on token values (or parts of composite values). Guards have the syntax typical of boolean expressions with infix notation. The comparison operators used are:

=	equal
>=	greater than or equal
>	greater than
<=	less than or equal
<	less than
<>	not equal

These may be linked together with (in order of precedence):

not	boolean NOT
andalso	boolean AND
orelse	boolean inclusive OR

Guards are customarily written enclosed in brackets, to help distinguish them visually from other CP net components.

Use of Parentheses

When **not**, **andalso**, or **orelse** are used, parentheses are required around the comparisons that they link. Thus:

```
[A < B andalso C > D]
```

is illegal. Instead use:

```
[(A < B) andalso (C > D)]
```

Parentheses may also be used as needed for clarity or to override the precedence order.

Shortcut for andalso

A comma may be used in place of **andalso**. Thus:

```
[(A < B) andalso (C > D)]
```

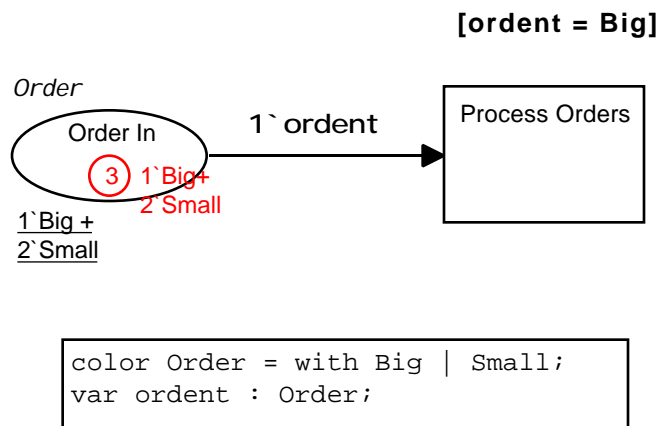
and:

```
[(A < B), (C > D)]
```

mean the same thing.

Constraining a Single Token

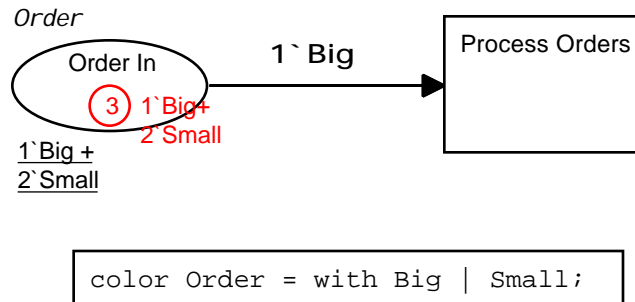
The simplest constraint restricts the value of a single token:



In order for `Process Orders` to be enabled, there must be a binding for `ordent` such that `1`ordent` evaluates to a multiset that exists in `Order In`, and `[ordent = Big]` is true.

Design/CPN User's Guide

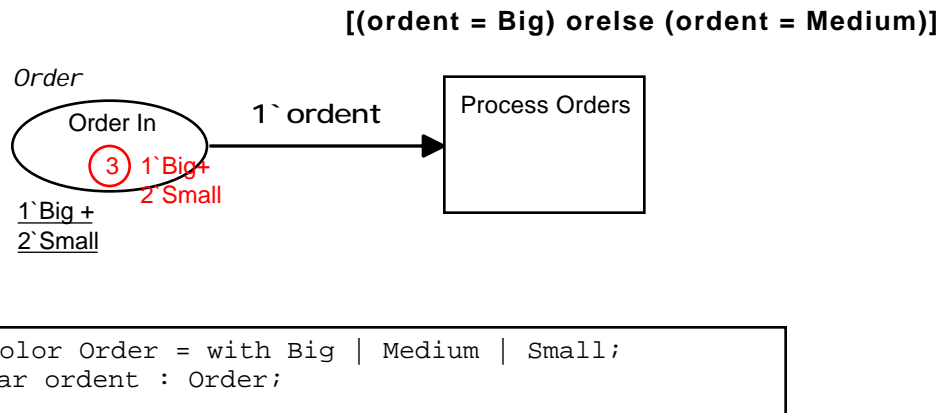
Obviously this is not much of a constraint: the transition is enabled when `ordent = Big`. We might as well have stuck with our very first example:



The equivalence between these two nets is a simple example of a very general truth: the realms of arc inscriptions and guards are not disjoint. A given constraint can often be implemented in several ways, each of which uses the capabilities of arc inscriptions and guards in a different manner to produce the same effect.

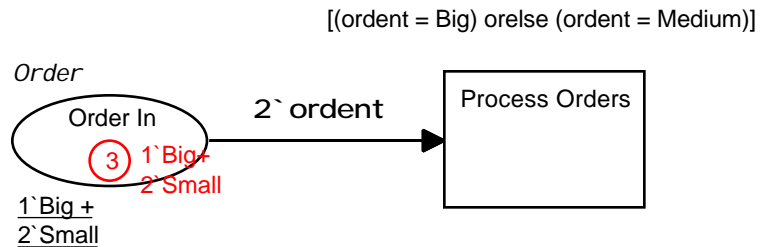
More Complex Constraints

A slightly more interesting example includes more possibilities:



There are now three types of `Order`: `Big`, `Medium`, and `Small`; and hence three possible bindings for `ordent`. Binding `ordent` to `Small` doesn't work, because then the guard is not true. Binding it to `Medium` doesn't work, because there are no `Medium` tokens in `Order In`. Binding it to `Big` works. There is a `Big` token in `Order In`, which satisfies the arc inscription, and the binding causes the guard to evaluate to true. Such a binding is sometimes said to *satisfy* the guard.

On the other hand, consider:



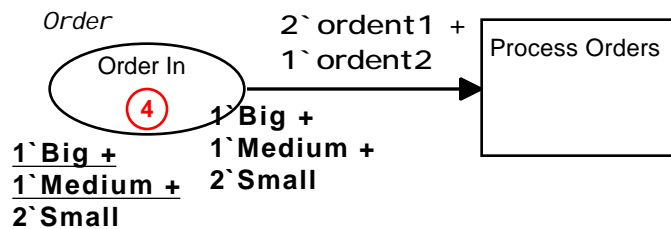
```
color Order = with Big | Medium | Small;
var ordent : Order;
```

Here the transition is not enabled. If `ordent = Big` or `ordent = Medium`, there are not enough tokens to satisfy the requirements of the arc inscription, but if `ordent = Small`, the guard is false.

Constraining More Than One Token

The techniques we have looked at are completely general: you can do anything with them that makes syntactic sense. For example:

**[[(ordent1 = Big) orelse (ordent1 = Medium)]
andalso (ordent1 <> ordent2)]**



```
color Order = with Big | Medium | Small;
var ordent1, ordent2 : Order;
```

In this case there are two CPN variables to bind. The simulator will try various bindings for each of them, looking for a binding that satisfies the arc inscription and the guard. A successful binding will bind `ordent1` to `Big` or `Medium`, and `ordent2` to either of the values that `ordent1` is not bound to; and will cause the arc inscription to evaluate to a multiset such that `Order In` contains two of whatever `ordent1` is bound to, and one of whatever `ordent2` is bound to.

This transition is not enabled.

Using a Guard to Create a Partial Constraint

The guards we have described so far have been very simple. Their only effect has been to predicate enablement on the existence of tokens with fixed values or combinations of values. This is an extremely common use of guards, but it barely scratches the surface of their capabilities.

The definition of a guard is very general. It is a boolean expression that must evaluate to **true** in order for a transition to be enabled. Nothing requires a guard to constrain tokens to fixed values. Nothing requires a guard to constrain tokens at all! A guard can be anything we want it to be. If it evaluates to **true**, the transition can be enabled; if it evaluates to **false**, the transition cannot be enabled. The rest is up to us.

For example, suppose we want a guard that enforces a constraint in some cases, and no constraint in others. The requisite guard has a **then** clause that requires a particular binding, and an **else** clause that is always **true** irrespective of any binding:

```
[if (order = Big)
  then staff = Expert
  else true]
```

What Happens When a Transition Occurs

The previous sections describes what is necessary in order for a transition to be enabled: that is, in order for it to be able to occur. When the simulator effects the occurrence of a transition, it is said to *fire* the transition. For brevity, transitions themselves are often said to fire, but it is important to remember that the simulator actually does everything: the transition just defines what is to be done.

Now we will examine what happens if the transition actually occurs. The if is significant. Not every enabled transition actually occurs: some other enabled transition might occur first, and change the net so that the first transition is no longer enabled.

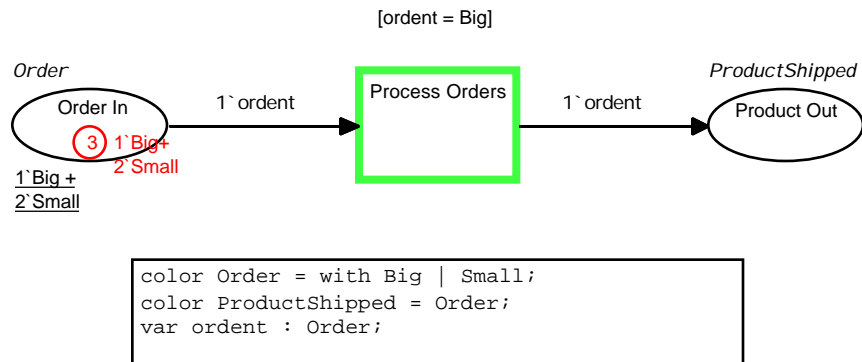
When the simulator fires a transition, it does the following:

1. Rebind any CPN variables as indicated by the enabling binding.
2. Evaluate each input arc inscription.
3. Remove the resulting multiset from the input place.

4. Evaluate each output arc inscription.
5. Put the resulting multiset into the output place.

CP Net Execution Example

Let's use SmallNet to illustrate CP net execution:



Note the highlighting around *Process Orders*. This is a convention that indicates that the transition is enabled. The section “Simulation Regions” in Chapter 6, “CPN Dynamics: Executing a Net,” provides details about such conventions.

Rebind Any CPN Variables per the Enabling Binding

A transition doesn't necessarily fire as soon as it is found to be enabled. The enabling binding has to be restored before firing can proceed.

The enabling binding was *ordent = Big*, so *ordent* becomes bound to the value *Big*.

Evaluate Each Input Arc Inscription

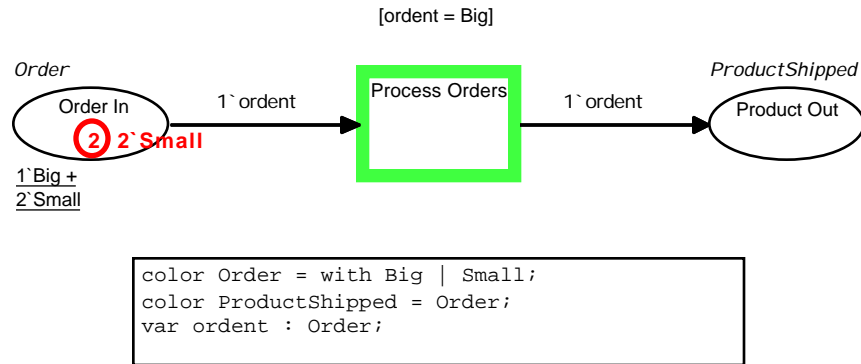
There is only one input place, so there is only one input arc inscription to evaluate. When *1`ordent* is evaluated with the binding *ordent = Big*, the result is the multiset *1`Big*. This is the enabling multiset: its existence is what caused *Process Orders* to be listed as enabled.

Remove the Enabling Multiset from Each Input Place

This is just a matter of multiset subtraction. There is a multiset in the input place, and there is an enabling multiset. Subtracting the

Design/CPN User's Guide

latter from the former removes the enabling multiset from the input place. When the subtraction has been accomplished, SmallNet looks like this:



`Process Orders` is still highlighted, but the highlighting is now twice as thick. Double-thickness highlighting indicates that a transition is in the process of firing. The marking of `Order In` is now `2`Small`, reflecting the subtraction of the enabling multiset `1`Big` from the place's previous marking, `1`Big + 2`Small`.

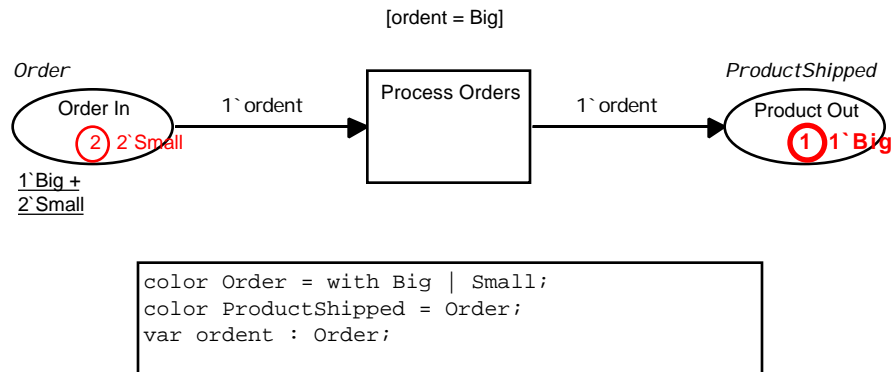
Subtracted tokens don't go anywhere: they just disappear. This is the principal difference between a token and an object in an object-oriented programming system: tokens are really indicators of the state of a CP net, not objects that are conserved.

Evaluate Each Output Arc Inscription

There is only one output place, so there is only one output arc inscription to evaluate. The inscription uses the CPN variable `ordent`, which is currently bound to `Big`. When `1`ordent` is evaluated with the binding `ordent = Big`, the result is the multiset `1`Big`. This is the *output multiset*.

Put the Output Multiset into the Output Place

This is just a matter of multiset addition. There is a multiset in the output place (in the current case it is the multiset `empty`), and there is a newly created multiset. Adding the latter to the former puts the output multiset into the output place, where it joins any tokens that were already there. When the addition has been accomplished, SmallNet looks like this:



There is no highlighting now, because firing is complete and the transition is no longer enabled. The marking of `Product Out` is now `1`Big`, reflecting the addition of the output multiset `1`Big` to the output place's previous marking, `empty`.

The `1`Big` that has been added to `Product Out` has no connection whatever with `1`Big` that was subtracted from `Order In`. It is a new token: the fact that its value is the same as that of the subtracted token results from the fact that both got their values from the binding of `ordent`. If the output arc inscription had been `10`ordent`, there would now be ten `Big` tokens in `Product Out`.

Saving and Loading Execution States

After a complex net has been executed for a long time, its state represents a considerable investment of time: if that state were lost, the information contained in it could be recovered only by repeating the whole execution process.

When a diagram is saved with an ordinary **Save** command, all state information is lost: if it is reopened, the net will revert to its initial state on entry to the simulator.

You can also save a net along with its current state. A diagram that contains a net saved in a particular state is called a *saved state*.

When a saved state is reopened, the net is still in the state it was in when it was saved, and execution can continue from that point as if there had been no interruption.

Saving a State

You can create a saved state whenever you are in the simulator and execution is stopped between steps.

Choose **Save State** from the **File** menu.

Design/CPN User's Guide

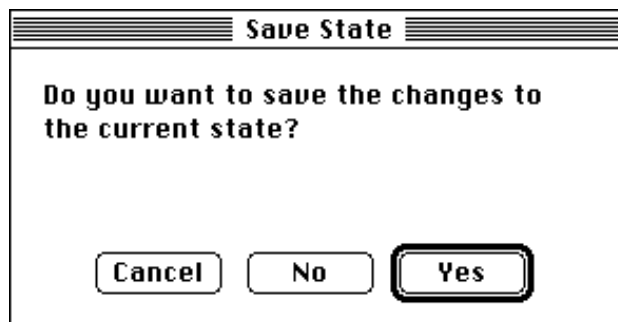
The **Save As** dialog appears. If there is already a saved copy of the net, you can overwrite it, or you can create a new copy by giving a different name. In either case the net is saved along with its current state.

Loading a Saved State

Loading a saved state is no different from opening a diagram generally, except that it is done from within the simulator.

Choose **Load State** from the **File** menu.

The **Save State** dialog appears:

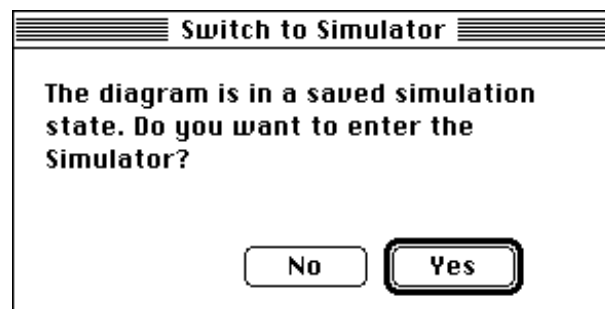


If you clicked **Yes**, the **Save As** dialog would appear; you could then save the net's current state, just as if you had explicitly chosen **Save State** from the **File** menu.

Starting With a Saved State

You don't have to already be in the simulator and use **Load State** in order to make use of a saved state. You can open a saved state from the editor, or you can start Design/CPN by opening the saved state, as you could with any ordinary diagram.

If you took a saved state into the editor, and then into the simulator, the state would be destroyed and the net's initial state established. To protect against this, Design/CPN displays a dialog when you open a saved state rather than loading it:



No: The saved state does not open. Design/CPN remains the active application.

Yes: The saved state opens in the simulator. Everything is just as it would have been if you had already been in the simulator and had loaded the saved state with **Load State**.

Chapter 6

CPN Dynamics: Executing a CP Net

This chapter shows you how to use the Design/CPN simulator to execute a CP net.

Performing a Syntax Check

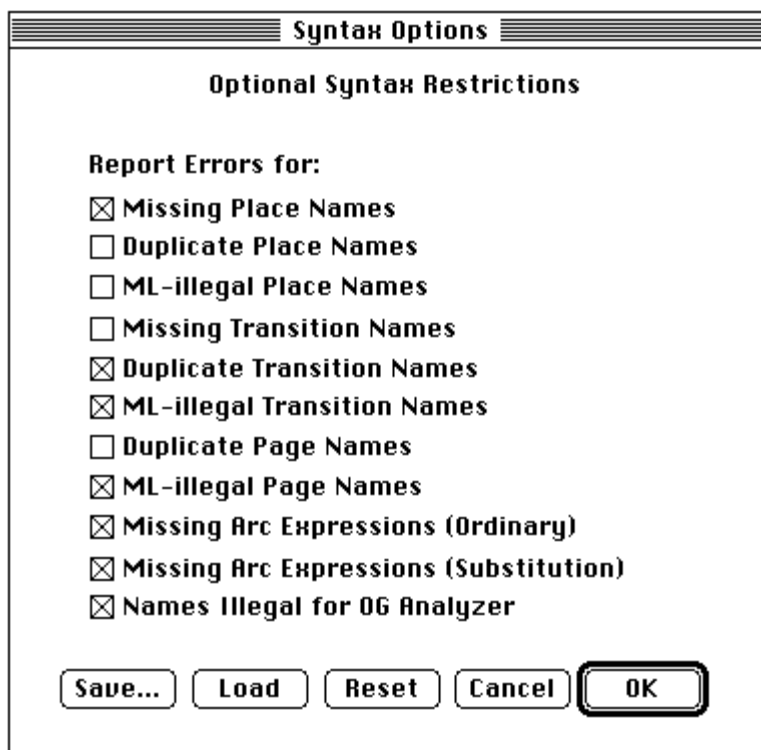
There would be no use attempting to execute a net that had syntax errors. Therefore Design/CPN requires that a syntax check be performed before you take a net into the simulator.

The **Syntax Options** (**Set** menu) command permits you to specify which optional syntax errors are to be reported during syntax check. The settings chosen in this dialog affect what happens when either the **Syntax Check** (**CPN** menu) command or **Enter Simulator** (**File** menu) command is invoked.

These optional restrictions can be very helpful when creating a large net spread over many pages. They increase the length of time to syntax check a net, however, so when a small change is to be syntax checked it is probably a good idea to turn them off.

Choose **Syntax Options** from the **Set** menu.

The **Syntax Options** dialog appears:



Each of these options is described in the command description in Part 2, "The Design/CPN Menu Reference."

Choose the options desired.

Click **OK**.

Performing the Check

Choose **Syntax Check** from the **CPN** menu.

If you see a dialog that mentions a problem of any kind, see Appendix B, "Troubleshooting."

If the diagram on which you are performing the check once had an ML file, but no longer does, a dialog appears that states:

Cannot find ML file.

The dialog offers you two options:

Build from scratch

Modify ML file name

The first choice, **Build from scratch**, is the default: choosing it will cause the simulator to build a new ML file based on information in the diagram and DB files.

Click **OK**.

If the syntax check is successful, the status bar displays Finished Syntax Check, and the **Syntax Check Successful** dialog appears:



Click **OK**.

If there are errors, an error message is displayed. Chapter 7, "CPN Dynamics: Handling CPN Syntax Errors," describes how to handle syntax errors uncovered during the syntax check.

If you direct Design/CPN to enter the simulator, and have not first done a syntax check, Design/CPN does one automatically. If the check is unsuccessful, Design/CPN remains in the editor and displays information about the error(s), as described in Chapter 7.

Designating a Prime Page

The process of creating and debugging a large net is a substantial undertaking. It would not be feasible to require that a net be completely executable before any of it could be executed. The same consideration arises in ordinary programming: a program must be usable in pieces, with the pieces in various stages of completion, in order for efficient development work to be done.

To facilitate incremental net development, Design/CPN does not automatically execute an entire net if it executes any of it. It executes only those parts of the net that either appear on a specially designated type of page, called a *prime page*, or are in some way referenced (directly or indirectly) by a prime page. A net may have any

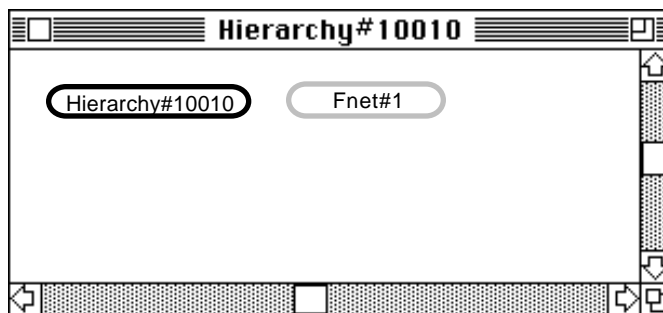
Design/CPN User's Guide

number of prime pages. In order to execute, a net must have at least one prime page, or the simulator will not find anything to execute.

A page must be designated as a prime page even if it is the only page in the net. To so designate it:

Choose **Open Page** from the **Page** menu.

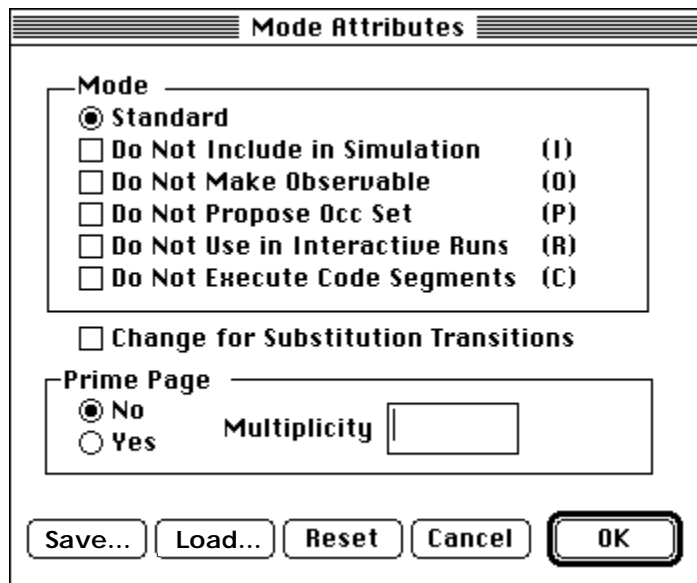
The hierarchy page appears:



Select the page node. In this example, Fnet#1.

Choose **Mode Attributes** from the **Set** menu.

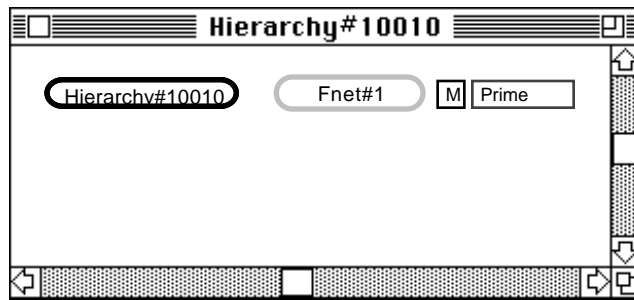
The **Mode Attributes** dialog appears:



Under **Prime Page**, click **Yes**.

Click **OK**.

The designated page is now a prime page. Some new information appears on the hierarchy page to indicate this. For example:



The new information consists of two regions. The **M** is called a *page mode key region*; the **Prime** is called a *page mode region*. These regions together indicate a prime page. The section “Creating Regions” in Chapter 3, “The Design CPN Editor: Introduction,” describes regions and their relationship to their parent nodes.

Setting Up a Net for Execution

The simulator has three different parameters for execution that are controlled through generating different types of code:

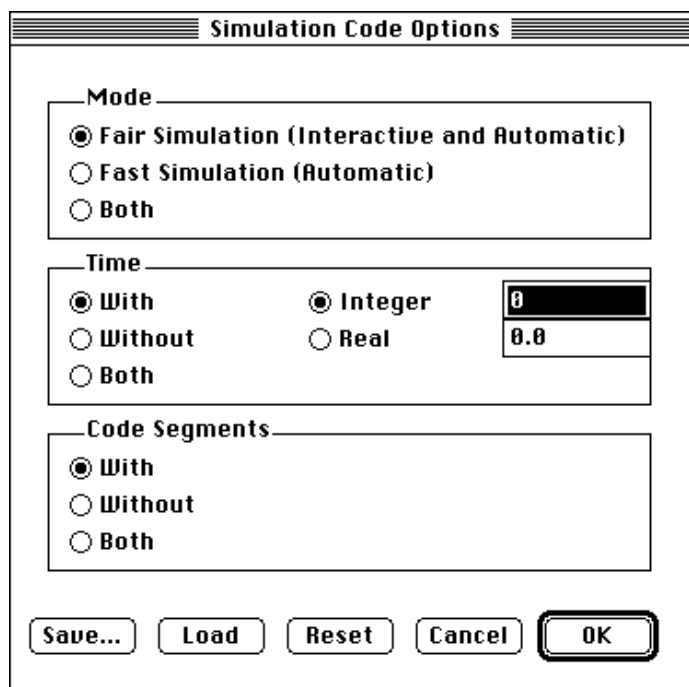
1. Occurrence set selection algorithm
2. Timed simulation
3. Simulation with code segments

Each affects the type of simulation, its speed, and outcome. Two commands in the **Set** menu control the generation of code for these options when the net is compiled (**Simulation Code Options** command) and the determination of what code to use when the net is executed (**General Simulation Options** command).

Generating Code for Different Types of Simulation

To generate code for a particular type of simulation:

Choose the **Simulation Code Options** command (**Set** menu). The following dialog appears:



The **Simulation Code Options** dialog determines what code will be generated for the simulation and what options are available for simulation in the **General Simulation Options** command (**Set** menu):

- **Mode: Fair Simulation** - turns off **Fast Automatic** in the **General Simulation Options** dialog.
- **Mode: Fast Simulation** - locks on **Fast Automatic** and turns off **Fair Automatic** and **Fair Interactive** in the **General Simulation Options** dialog.
- **Time: With** - locks on **Time** in the **General Simulation Options** dialog.
- **Time: Without** - turns off **Time** in the **General Simulation Options** dialog.
- **Code Segments: With** - locks on **Code Segments** in the **General Simulation Options** dialog.
- **Code Segments: Without** - turns off **Code Segments** in the **General Simulation Options** dialog.

In all three sections - **Mode**, **Time**, and **Code Segments** - selection of **Both** causes the generation of code for all possibilities listed in the section, and enables the selection of any of these possibilities in the **General Simulation Options** dialog.

Generating Code for the Method of Simulation

The simulator provides three modes of execution:

- Fair Interactive
- Fair Automatic
- Fast Automatic

They differ in speed of code generation, size of code generated, and in speed of execution.

Each method affects which options can be set in the **General Simulation Options** dialog and which commands are available in the **Sim** menu.

Selecting **Fair Simulation (Interactive and Automatic)** in the **Mode** section causes Design/CPN to generate code for both the Fair interactive simulation and Fair automatic simulation.

- **Fair** means that the simulator makes a random selection among potential enabling bindings. Each such binding is equally likely to be selected. This method is slower than **Fast** but helps ensure that each enabled token has an equal opportunity to be chosen.
- **Fast** means that the simulator chooses as fast as possible among the enabled tokens. This is faster than the **Fair** method, but can result in skewed choices among the tokens.

Fair interactive is used for debugging and is the slowest in execution. It provides the ability to stop net execution at specific points, called *breakpoints*. While execution is stopped various experiments can be performed, such as changing the bindings, modifying markings, etc. Some of these options are discussed in this chapter and some are discussed in Chapter 8, “CPN Dynamics: Concurrency and Choice.”

Fair Automatic is used for execution when execution time is less important than modeling accuracy.

Fast Automatic is used for execution when execution time is more important than modeling accuracy.

Time Section

With causes Design/CPN to generate code for timed simulation.

Without causes Design/CPN to generate code for non-timed simulation.

Both causes Design/CPN to generate code for both timed and non-timed simulation.

Timed simulation is discussed in Chapter 12.

Code Segments Section

With causes Design/CPN to generate code for executing code segments during simulation code.

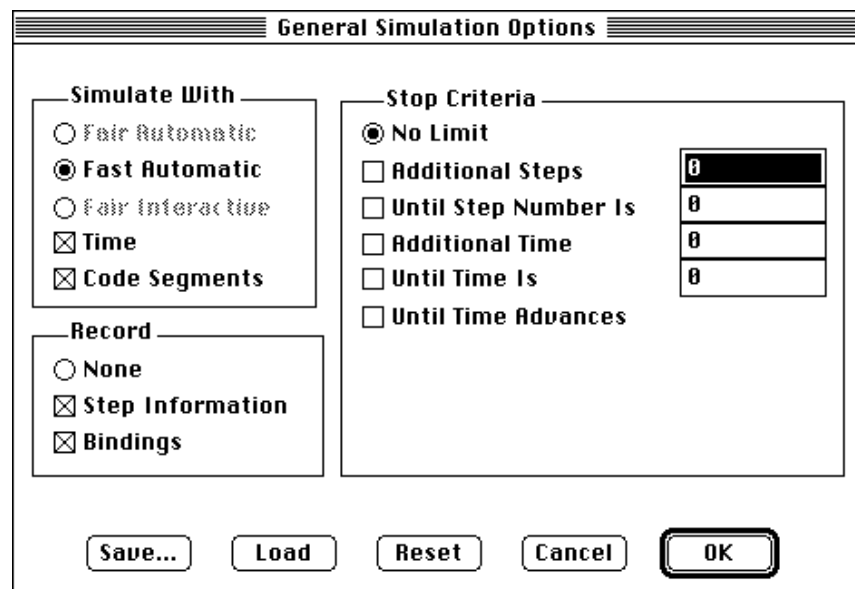
Without causes Design/CPN to generate code ignores code segments during simulation.

Both causes Design/CPN to generate code for simulation with and without code segment execution.

Selecting Different Types of Simulation

To select the type of simulation:

Choose the **General Simulation Options** command (Set menu). The following dialog appears:



The dialog box titled "General Simulation Options" contains several sections for configuring simulation parameters. The "Simulate With" section has radio buttons for "Fair Automatic", "Fast Automatic" (selected), and "Fair Interactive", and checkboxes for "Time" and "Code Segments". The "Record" section has radio buttons for "None" and checkboxes for "Step Information" and "Bindings". The "Stop Criteria" section has a radio button for "No Limit" (selected) and checkboxes for "Additional Steps", "Until Step Number Is", "Additional Time", "Until Time Is", and "Until Time Advances". To the right of the last four checkboxes are input fields with the value "0". At the bottom are buttons for "Save...", "Load", "Reset", "Cancel", and "OK".

Section	Option	Value
Simulate With	Fair Automatic	<input type="radio"/>
	Fast Automatic	<input checked="" type="radio"/>
Simulate With	Fair Interactive	<input type="radio"/>
	Time	<input checked="" type="checkbox"/>
Simulate With	Code Segments	<input checked="" type="checkbox"/>
	Record	None
Record	Step Information	<input checked="" type="checkbox"/>
	Bindings	<input checked="" type="checkbox"/>
Stop Criteria	No Limit	<input checked="" type="radio"/>
	Additional Steps	<input type="checkbox"/>
	Until Step Number Is	0
	Additional Time	<input type="checkbox"/>
	Until Time Is	0
	Until Time Advances	<input type="checkbox"/>

Specifying the Type of Execution

The **Simulate With** section specifies the type of execution. Which options are available depends on the options that were selected in the **Simulation Code Options** dialog. The section "Generating Code for Different Types of Simulation" in this chapter describes the various code generation options.

Specifying Termination Conditions

The **Stop Criteria** section specifies the conditions under which the simulation will terminate:

No Limit

Simulation continues until there are no enabled transitions. Provides the possibility for non-terminating simulations.

Additional Steps

Simulation continues for the number of steps specified in the value box.

Until Step Number Is

Simulation continues until the step number specified in the value box is reached.

Additional Time

Simulation continues for the number of time units specified in the value box.

Until Time is

Simulation continues until the time value is that specified in the value box.

Until Time Advances

Simulation continues until time changes.

Recording the Results of Execution

The **Record** section specifies the recording of simulation data in a report. Recorded data can be written to a file using the **Save Report** dialog (**Sim** menu).

None

No data is recorded. This results in slightly faster simulation.

Step Information

Step information is recorded. For each simulation step, a line is stored whose syntax is:

Step M Transition@(Instance:Page)

For example:

1 I Process@(1:Fnet#1)

This line specifies that in Step 1, while executing in an Interactive simulation mode, a transition named `Process` occurred on Instance 1 of page `Fnet#1`. If the transition had occurred during Automatic simulation, the line would have read:

1 A Process@(1:Fnet#1)

Bindings

Bindings are recorded. For each simulation step, a line is stored whose syntax is:

{variable = value [,variable = value]...}

For example:

{ordent = Big, staff = Expert}

This line specifies that a step occurred in which a CPN variable named `ordent` was bound to the value `Big`, and a CPN variable named `Staff` was bound to the value `Expert`.

Simulation Report Example

The following report shows both step information and bindings for a net in which a transition named `T1` fired alone in three consecutive steps:

```
Simulation Report
1 I T1@(1:New#1)
  {x = 1, y = 3}
2 I T1@(1:New#1)
  {x = 2, y = 1}
3 I T1@(1:New#1)
  {x = 1, y = 3}
```

The first line is a standard header; it is included in every simulation report.

Adding Information to the Report

It is also possible to include information in a report by calling an ML function from the code segment of a transition. The function is:

```
write_report: string -> unit
```

The information contained in `string` will appear immediately before the automatically generated information that describes the firing of the transition. For example, if the code segment for transition T1 contained the statement:

```
write_report  
  ("The time is: " ^^  
   (makestring (time())));
```

the example report above might look like this:

```
Simulation Report  
The time is: 0  
1 I T1@(1:New#1)  
  {x = 1, y = 3}  
The time is: 5  
2 I T1@(1:New#1)  
  {x = 2, y = 1}  
The time is: 7  
3 I T1@(1:New#1)  
  {x = 1, y = 3}
```

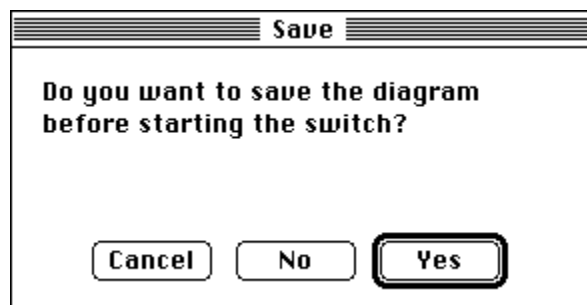
Entering the Simulator

Entering the Simulator from the Editor After a Syntax Check

Choose **Enter Simulator** from the **File** menu.

If you see a dialog that mentions a problem of any kind, see Appendix B, “Troubleshooting.”

Design/CPN displays the following dialog:



Cancel: Design/CPN remains in the editor.

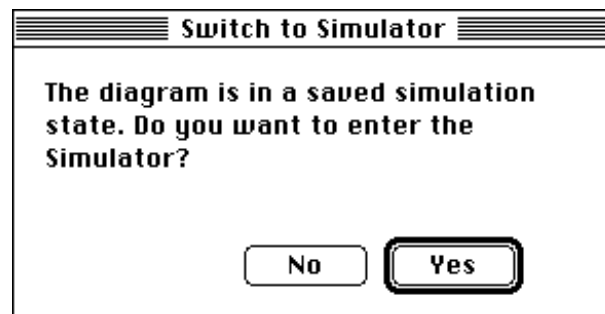
Design/CPN User's Guide

Yes: Design/CPN saves the diagram. The diagram now includes the ML code that was generated during the syntax check.

No: Design/CPN enters the simulator without saving the diagram.

Entering the Simulator With a Saved State

If you double click on a diagram icon that has been saved with the **Save State** dialog described later in this chapter in the section “Leaving the Simulator,” the simulator offers the opportunity to immediately enter the simulator:



No: The saved state does not open. Design/CPN remains the active application.

Yes: The saved state opens in the simulator. Everything is just as it would have been if you had already been in the simulator and had loaded the saved state with **Load State**.

The Sim Menu

On entry to the simulator the **CPN** menu item is replaced by **Sim**. The **CPN** menu is no longer needed since in the simulator you cannot create new net structure (though you can modify existing structure to some extent). The **Sim** menu commands prepare the net for simulation and allow you to regulate the simulation cycle, switch between instances, and return the net to its original state. (The section “Instance Fusion Sets” in Chapter 10, “CPN Hierarchy: Fusion Places,” describes what instances are and one particular use of them.)

The appearance of the menu is controlled by the simulation options selected in the **General Simulation Options** command (**Set** menu). The following commands are grayed out unless the requirement listed is met:

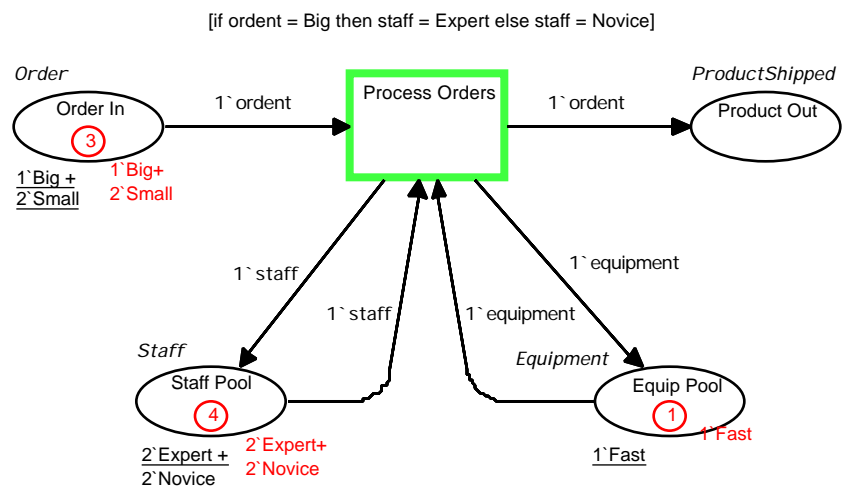
- **Bind** requires Fair Interactive Simulation.

- **Occurrence Set** requires Fair Interactive Simulation.
- **Start Step** requires Fair Interactive Simulation.
- **Interactive Run** requires Fair Interactive Simulation.
- **Automatic Run** requires either Fair Automatic simulation or Fast Automatic simulation.
- **Continue** requires Fair Interactive Simulation.
- **Stop** requires Fair Interactive Simulation.

Simulation Regions

The menu bar is not the only thing changed on entry to the simulator; the net itself has a different appearance in the simulator than it did in the editor. New regions, called simulation regions, have appeared.

For example:



This model, SalesNet, illustrates changes that occur during CP net execution. In this example, a multiset representation has appeared inside Order In, indicating its current marking, and Process Orders has been highlighted, indicating that it is enabled. All of these indications are provided automatically by the simulator to make the state of the net more obvious.

The simulator changes a net's appearance to indicate its state by adding *simulation regions* to the net. Simulation regions are similar to the CPN regions in that they are subsidiary graphical objects.

The difference is that they are created automatically by the simulator to indicate net state, rather than by a human to indicate net structure.

Simulation Region Types

Simulation Regions Indicating Place Markings

The multiset representation in `Order In` consists of two regions: the number in a circle is called the *current marking key region*, and the `count`value` pairs next to it constitute the *current marking region*. Together these regions indicate that there are currently three tokens in the place, one of value `Big` and two of value `Small`, as specified by the initial marking region `1`Big + 2`Small`.

The existence of both the initial marking region and the current marking regions is not a redundancy. The initial marking region is just a label region with some text in it that tells the simulator what tokens to create as the place's initial marking. The current marking regions represent actual tokens, created on entry to the simulator.

As the net executes, the current marking regions change to indicate the changing marking of `Order In`, but the initial marking region remains the same. Current marking regions are used during simulation to indicate all non-empty place markings.

Simulation Region Indicating Enablement and Firing

The highlighting around `Order In` indicates that it is enabled, as you can verify by looking at its marking and the arc inscription. This highlighting is actually a region called a *transition feedback region*, or for brevity a *feedback region*.

Whenever a transition is enabled or is in the process of firing, it is highlighted with a feedback region. When the transition is enabled, this region looks as it does in the case of `Order In`. When a transition is firing, the thickness of the region doubles.

Key and Popup Regions

There is a regular pattern in the names given to the current marking, input token, and output token regions. There is always a *key region*, a number in a circle that tells the number of tokens in the multiset, and an associated region that shows the exact composition of the multiset. These associated regions are called *popup regions*.

Popup regions are so called because they can be made to appear (“pop up”) and disappear at any time by double-clicking on the as-

sociated key region. This trick works in both the editor and the simulator.

Popup regions are often a great convenience: the detailed information in the popup region can be displayed when it is useful, and hidden when it is not. When it is hidden, the key region remains to provide a summary and to permit quick access to the complete information in the popup region.

Popup regions are used for more than just representing multisets. The Page Mode Region associated with a prime page on the hierarchy page is actually a popup region, and would disappear and reappear if you double-clicked the associated Page Mode Key Region. Whenever a region is described as a key region, there is an associated popup region, and vice versa.

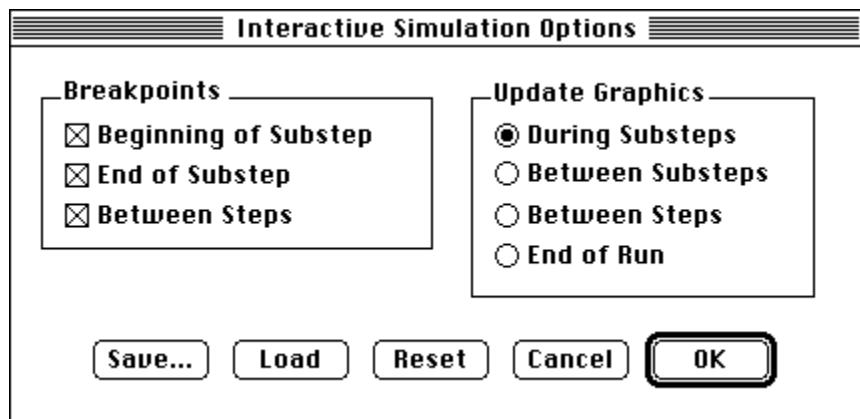
Repositioning Key and Popup Regions

Key and popup regions are just ordinary graphical objects. You can move them anywhere you like by dragging them with the mouse. A popup region is a region of its key region: you can drag the key region and the popup will follow, tracking the parent.

Setting Interactive Options

To better display the details of CP net execution:

Choose **Interactive Simulation Options** from the **Set** menu.



Click options as needed to set breakpoints (**Breakpoints** section), and to show tokens (**Update Graphics** section):

Breakpoints Section

- **Beginning of Substep.** Selecting this breakpoint causes the simulator to pause when it has determined what input tokens to subtract, and what output tokens to add, but before the transition fires. For brevity, the status bar describes this breakpoint as Breakpoint 1.
- **End of Substep.** Selecting this breakpoint causes the simulator to pause immediately after the firing transition has subtracted the input tokens and added the output tokens. For brevity, the status bar describes this breakpoint as Breakpoint 2.
- **Between Steps.** Selecting this breakpoint causes the simulator to at the end of each step.

Update Graphics Section

- **During Substeps.** Selecting this option causes the simulator to update graphics as they are changed.
- **Between Substeps.** Selecting this option causes the simulator to update graphics at the end of each substep.
- **Between Steps.** Selecting this breakpoint causes the simulator to update graphics at the end of each step.
- **End of Run.** Selecting this breakpoint causes the simulator to update graphics at the end of the run.

Running an Interactive Simulation

Interactive simulation provides the ability to stop the simulation at breakpoints set in the **Interactive Simulation Options** dialog (**Set** menu). To start the simulation:

Choose **Interactive Run** from the **Sim** menu.

If breakpoints have been set, the simulator stops when the breakpoint is reached. Substep breakpoints do not present a dialog but are indicated by status bar messages.

Continuing from a Substep Breakpoint

To continue from a substep breakpoint:

Choose **Continue** from the **Sim** menu.

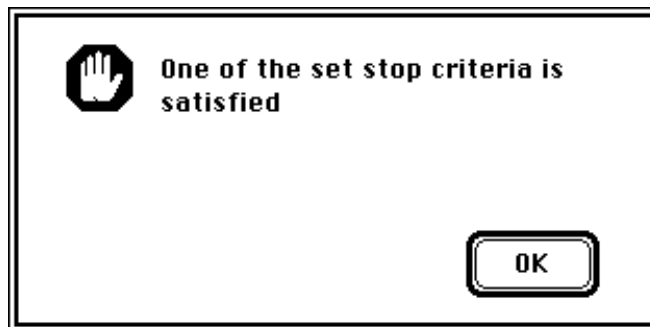
When a step is finished the simulator stops and displays a **Step Finished** dialog that displays the step number just finished and provides options for further action. For example:



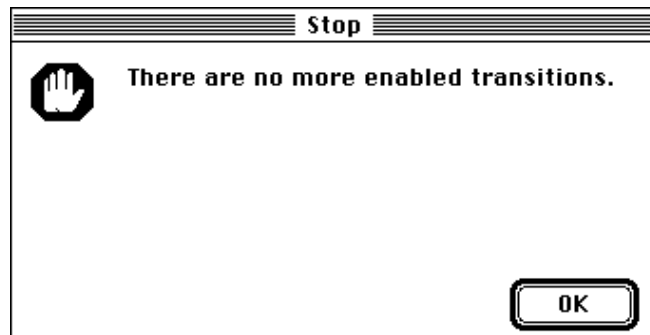
Stop: Stops the simulation and provides the opportunity to modify breakpoints, graphic update parameters, and certain components of the net such as guards, initial markings, and arc inscriptions. Net structure itself may only be modified by leaving the simulator and returning to the editor.

Cont: Continues the simulation.

When any stop criterion is satisfied (See the section “Setting Substep Options” earlier in this chapter) the simulator displays:



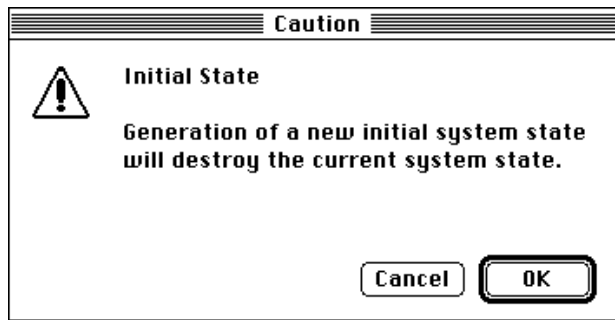
When net execution ends because there is nothing left to do, the simulator displays:



Re-Executing a Net

When a CP net cannot be re-executed in its current state because there are no enabled transitions, you must restore its original state before you can execute it again.

Choose **Initial State** from the **Sim** menu.



When a complex net has been executing for a long time, its current state represents a considerable investment in time, since it could be recreated only by redoing the entire execution process. This dialog helps to protect against accidental erasure of such a state.

Cancel: Cancels the initial state request, thereby providing the opportunity to issue the **Save State** command (**File** menu).

OK: Continue with state initialization.

When initialization is complete, the status bar displays Finished Initializing State. The net is now in exactly the same state that it was in before it executed.

Leaving the Simulator

You can leave the simulator only when a simulation is stopped, either through simulator termination or through cancellation at the end of a step.

Choose **Enter Editor** from the **File** menu.

Removing Simulation Regions

Design/CPN doesn't automatically remove simulation regions when you transfer back to the editor. You might want to keep the information in them around for some reason, perhaps to help with decisions about net modification.

If you don't want to keep leftover simulation regions after you are back in the editor, you can remove them by selecting them and deleting them with DELETE. However this method would be inconvenient if there were a large number of unwanted simulation regions, so Design/CPN allows you to eliminate them all in one operation.

Choose **Remove Sim Regions** from the **CPN** menu.

The **Remove Sim Regions** dialog appears:



The four options describe various types of simulation region. To remove all simulation regions on the current page:

Click **OK**.

Any simulation regions left over from net execution are gone. The results of any manual adjustments of simulation regions are gone also, whether or not the regions were visible in the editor when you removed all simulation regions. If you now re-entered the simulator and re-executed the net, the various simulation regions would appear in their default positions: with respect to simulation regions, it is as if you had never entered the simulator at all.

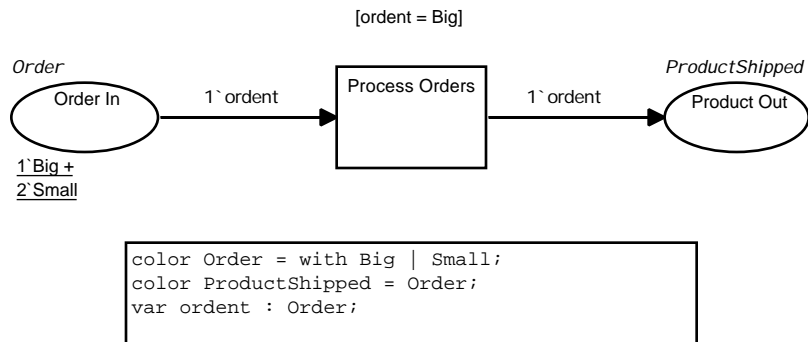
Removing Simulation Regions from Multiple Pages

You can remove simulation regions from pages other than the current page. To do this, open the hierarchy page, select a group containing the page nodes for all the pages whose simulation regions you want removed, and execute **Remove Sim Regions** as described above.

Chapter 7

CPN Dynamics: Handling CPN Syntax Errors

This chapter shows you the general method for handling CPN syntax errors, and illustrates appropriate responses to some specific errors. Examples in this chapter use SmallNet, whose construction and execution were covered in the preceding chapters.



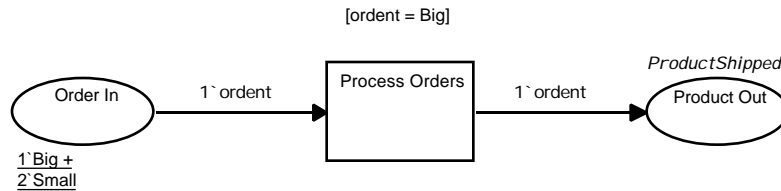
CPN syntax errors never involve illegal net structure, because the Design/CPN editor does not allow you to create anything structurally illegal. Most CPN syntax errors involve one of the following:

- Missing net components.
- Typographical errors.
- CPN ML syntax errors.

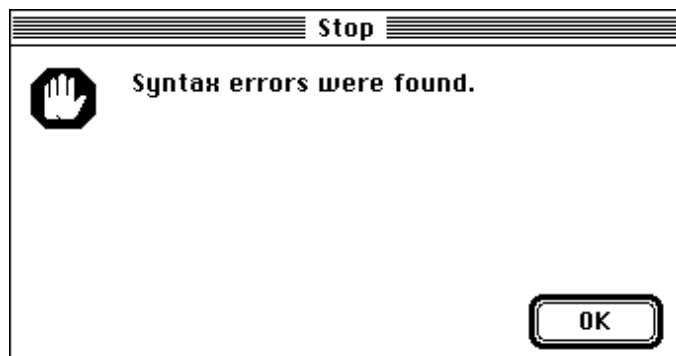
The errors illustrated in this chapter do not cover the spectrum of possible errors, nor do they need to. The goal here is to show you how Design/CPN responds when an error occurs. Once you have the general picture, you can deal with any particular error by reading the resulting error message(s) and responding as needed.

Missing Colorset Specification

Failure to specify a place's colorset is one of the most common syntax errors. In this example, the `Order` colorset is not specified for the input place `Order In`.



When this net is syntax checked a dialog appears:



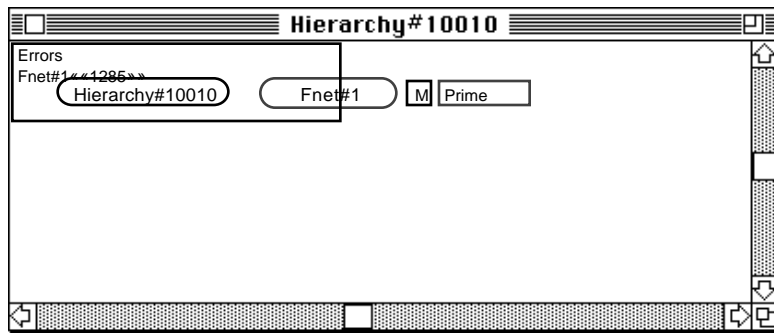
Click **OK**.

This dialog appears whenever you request a syntax check on a net that is correct enough to be checkable, but that contains one or more syntax errors.

In this case, you know where the error is and what it consists of, but this would not ordinarily be the case. Unintentional errors must first be located in the net, then analyzed, then repaired.

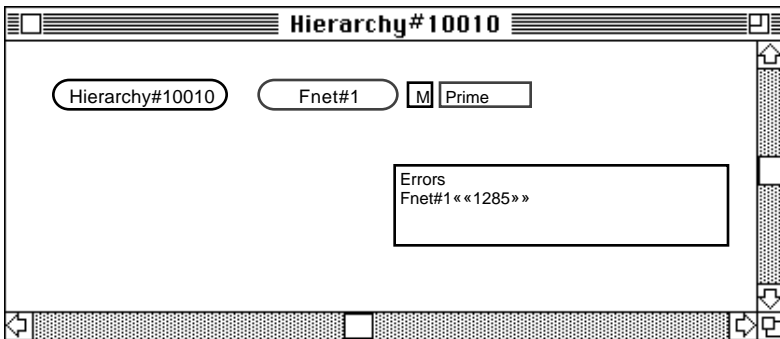
Locating a Syntax Error

When syntax errors are found, Design/CPN brings the hierarchy page to the front, and adds to that page an auxiliary rectangle called an *error box*. This box is initially positioned in the upper left corner of the page border. Its text field identifies each page that has one or more errors.



The error box is really just an auxiliary rectangle: it can be moved and reshaped just as any rectangle can be.

Move the hierarchy page error box away from the page nodes, and enlarge it if necessary until you can read its contents:



In this example, the text in the box indicates that there is an error on the page Fnet#1. If there were other pages with one or more errors, the name of each page would be listed below the Fnet#1 line.

Text Pointers

Every Design/CPN error message that relates to a particular component of a net contains a pointer to that component. Such a pointer is called a *text pointer*. A text pointer consists of a number surrounded by angle braces. In the example error message depicted above, the text pointer is ««1285»». The number in a text pointer is just an arbitrary number that Design/CPN generates to distinguish one pointer from another.

You can use a text pointer to jump directly to whatever it points to. The text pointer in each error notification in a hierarchy page error box can take you to the page on which the error exists. To follow a text pointer:

Enter text mode.

Design/CPN User's Guide

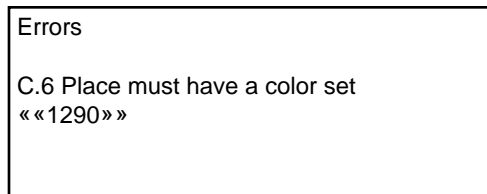
Place the text insertion cursor anywhere inside the text pointer.

Press ALT-DOWN-ARROW.

In the SmallNet example, Design/CPN opens the page Fnet#1.

When an unsuccessful syntax check has been performed, Design/CPN creates an error box on every page that has any errors. This box is positioned in the upper left corner of the page border. The text pointer in a hierarchy page error notification actually points to the error box on the page that contains the error. Following the pointer opens the page and selects the error box. The box may have to be moved and expanded to make its contents completely visible.

The error box on page Fnet#1 would look something like this:



Now all you need to do is find a place on Fnet#1 that lacks a colorset region, and you have located the error. The text pointer in the error message can take you directly to the place in question.

Follow the text pointer in the error message.

Design/CPN selects the place `Order In`. Since you are still in text mode, you could now edit the place's text field if that were appropriate. In this case, the appropriate action would be:

Leave text mode.

Supply the missing colorset region by using **CPN Region** (CPN menu).

Undeclared Variables

Another common error is failure to declare a variable. For example, suppose that SmallNet's global declaration node was not:

```
color Order = with Big | Small;  
color ProductShipped = Order;  
var ordent : Order;
```

but rather:

```
color Order = with Big | Small;  
color ProductShipped = Order;  
var XXX : Order;
```

Running a syntax check will now produce an error box on Fnet#1 that looks about like this:

```
Errors  
  
C.9 Guard region must be legal  
  
Type checking error in:ordent  
Unbound value identifier: ordent  
[Closing <string>  
« «1303» »
```

Navigating the text pointer in this error message would lead to the transition `Process Orders`, even though the actual error, a failure to provide a variable declaration, occurred in the global declaration node.

Design/CPN does not attempt to analyze the ultimate origin of the errors it encounters: doing so would be unacceptably time-consuming, and could not produce reliable results in any case. Instead it indicates the location at which it encountered a problem, leaving it to you to determine whether the problem originates there or is a consequence of a problem somewhere else.

In this case, Design/CPN happened to check the transition before checking either arc inscription, and found an unbound variable in the guard. The error message therefore points to the transition. It is up to you to determine whether the wrong variable is used in the guard, or the right variable is used but has not been declared: this is a semantic rather than a syntactic question, so Design/CPN cannot determine the answer. In this case, the appropriate action would be:

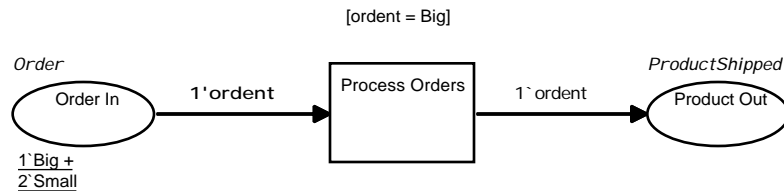
Remain in text mode.

Select the global declaration node.

Edit it to correctly define the variable **ordent**.

Illegal CPN ML Constructs

The CPN ML parts of a CP net can experience the same kinds of errors that can occur with programming languages generally. Sometimes an almost invisible error can occur: Design/CPN's way of handling errors can lead you directly to it. In the example below, the backquote (') in the input arc inscription between Order In and Process Orders is a single quote (').



Following a syntax check, the Fnet#1 error box would look about like this:

```
Errors

C.11 Arc expression must be legal

Parse error:
  Was expecting ",'"

In: ... fun CPN'AF6 ( CPN'bb : CPN'BT4 ) : Order
ms = 1 <?> 'ordent

[Closing <string>]
[Closing <string>]
```

The description in the error box is not that informative. It describes the error as it appeared to the CPN ML syntax checker, not as it appears to a human observer. The answer is to follow the text pointer in the error box message. This will select the input arc that has the erroneous inscription. Direct inspection can then reveal what the problem is.

Chapter 8

CPN Dynamics: Concurrency and Choice

All of the nets we have looked at so far executed sequentially: only one thing happened at a time. Such execution does not require any particular sophistication or complexity on the part of the CPN simulator. It does one thing, and then the next, and then the next, as long as anything remains to be done.

Sequential operation is not typical of real systems. Systems that perform many operations and/or deal with many entities usually do more than one thing at a time. Activities that happen at the same time are called *concurrent activities*. A system that contains such activities is called a *concurrent system*. The phenomenon of concurrent activities is called *concurrency*.

Concurrency Problems

Concurrency can create problems that do not arise in sequential systems. These problems typically involve competition for resources. In a system in which there is only one active agency, there can be no such competition, because there is nothing for the agency to compete with. But where there are concurrent activities that need the same resources, and there are not enough resources to insure that every activity will always have all that it needs, the activities are forced to compete for what resources there are.

There are two possible ways to deal with competition for resources. One is to implement a determinate algorithm for resource allocation. This solution is simple, but it is not always possible or desirable. Where it is not, the only way to resolve the competition is to randomly allocate resources to some contenders, and require others to wait. The subsequent course of events in the net often depends on which activities prevailed and which had to wait.

Activities that vie for resources in the absence of a determinate allocation mechanism are said to be *in conflict*. The act of adjudicating such a conflict is called *choice*. Since the choice is made at random,

and may affect the subsequent course of execution in unpredictable ways, the result of choice is *indeterminacy*.

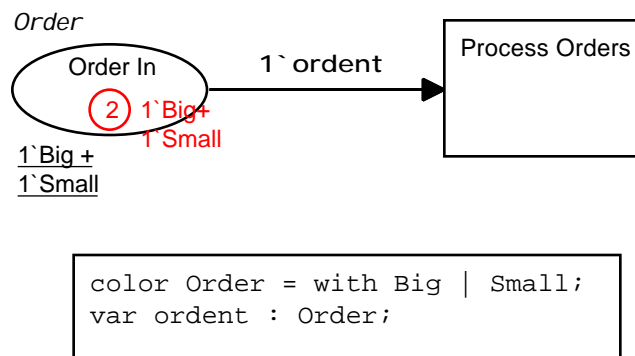
Due to the possibility of conflict and the resulting need for choice, it is impossible for the simulator to get by using only the simple methods we have ascribed to it so far. It must be able to handle conflicts and make choices that can be quite complex, and it must be able to do so efficiently, or the execution of a net that requires many choices will be too slow. This chapter describes the capabilities that the simulator uses to adjudicate conflicts by making choices.

Representing Concurrency

Before we see how the simulator handles conflict and choice, we need to see how concurrency and conflict are actually represented in a CP net, and how they affect its execution.

Multiple Enabling Bindings

Example:



Here there are two orders waiting to be processed, a Big order and a Small order. Consequently there are two bindings of `ordent` for which `Process Orders` is enabled: `ordent = Big` and `ordent = Small`.

How should the simulator handle this situation? The two possibilities are:

1. Process one of the orders, and then process the other.
2. Process both orders at the same time.

The first is straightforward enough. It would entail `Process Orders` firing with one binding, and later firing with the other.

CPN Dynamics: Concurrency and Choice

The second option may seem a bit strange: how can the same activity occur in two different ways at the same time? How can one thing do several things simultaneously?

Concurrent Transition Firing

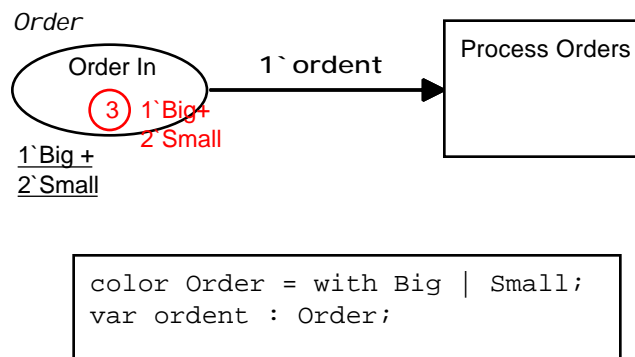
Nothing in the net specifies that `Process Orders` can do more than one thing at a time. But neither does anything specify that it cannot! CP nets make no assumption of sequential behavior: they allow as many things as possible to happen at the same time. Since concurrency abounds in real systems, this property is extremely convenient in a modeling paradigm.

In this case there are two jobs ready to process, represented by the two tokens that produce the two enabling bindings, so `Process Orders` will process them concurrently. If we don't want them processed concurrently, we must explicitly specify something that prevents it, as described later in this chapter.

It is tempting to see a transition like `Process Orders` as an active principle that maps inputs to outputs in a sequential way, but this image is wrong. A transition is just a representation of a way in which the state of a CP net can change: it has no life of its own. The mapping of inputs to outputs is done by the simulator, not by the transition. Nothing prevents the simulator from carrying out more than one such mapping at a time, and thereby implementing concurrency.

Identical Enabling Bindings

Nothing requires that enabling bindings be unique. For example:

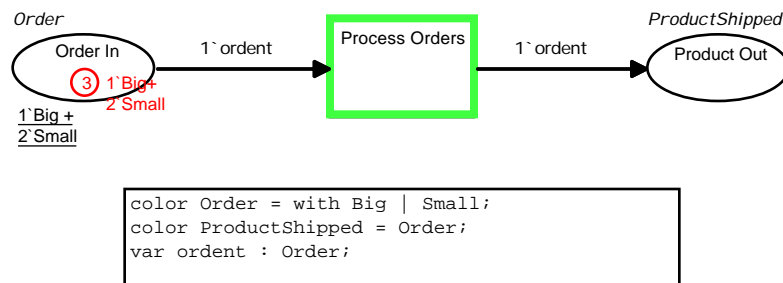


Here there are three enabling bindings, not two. The fact that two of them are identical does not mean they are not distinct entities. If we allowed only one binding at a time with a given value, this would be equivalent to ruling out the possibility of concurrently processing identical entities in the same way. This would obviously be unacceptable as a general restriction in a modeling paradigm.

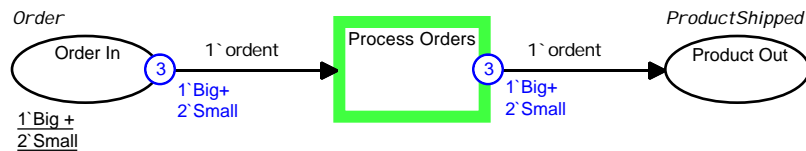
Concurrent CP Net Execution

Let's take a look at a concurrent execution of SmallNet. One possible sequence is:

Initial State of the Net



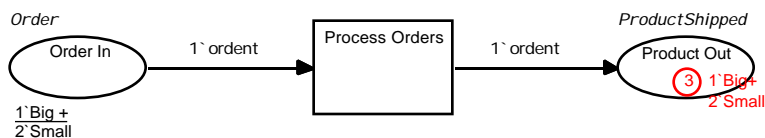
Breakpoint 1: Beginning of Substep



Breakpoint 2: End of Substep



Execution Is Complete



Analysis of the Execution

The only difference between this execution of SmallNet and those described in Chapter 6 is that three `Order` tokens were processed in the same step, while previously only one `Order` token was processed, after which there was nothing left to do. This merely reflects the fact that this time there are three enabling bindings, one for each of three `Order` tokens, while previously there was only one enabling binding.

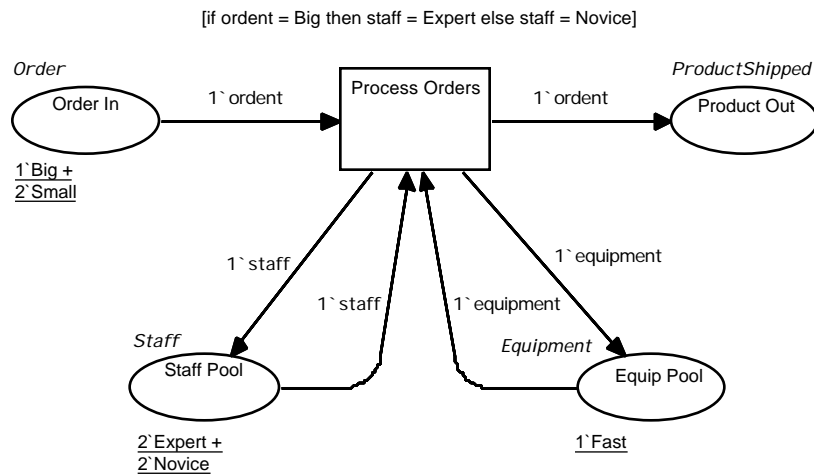
Note that nothing special was done to `Process Orders` to make it process one order or three: it just processes the orders that give rise to enabling bindings. If the result is concurrent processing, then it is. In a CP net, no special effort is required to specify either concurrent or sequential behavior: one just creates a net that has the desired structure, and its behavior, whether concurrent or sequential, follows as a matter of course.

Representing Conflict

SmallNet places no limitation on the number of orders that can be processed concurrently. There could be a thousand or a million orders, and they would all be processed at once. Obviously this is unrealistic: in a real system, there is always some limit to how much can be done at one time.

Nothing requires that a CP net be realistic, or protects automatically against the consequences of unrealistic modeling. The problem here is that SmallNet is too simple. It contains no representation of anything that would limit concurrent processing of orders, so there is no limit. If we want to limit concurrency in a net, we must explicitly specify some limiting factor. A CP net is exactly what we make it be, and nothing more.

SalesNet shows how concurrency can be limited:



SalesNet specifies that a `Big` order can be processed only when there is an `Expert` staff member and one piece of equipment available, while a `Small` order can be processed only when there is a `Novice` staff member and one piece of equipment available.

There is no problem with staff availability: there are enough staff members to process all three orders concurrently. But there is only one piece of equipment. Since each order must have a piece of equipment in order to be processed, and there is only one piece, the three orders cannot be processed concurrently. One must use it and then replace it, after which another can use it, and so on.

The fact that the three orders cannot use the piece of equipment simultaneously means that they must compete to obtain it. That is, they are in conflict for the piece of equipment, and this conflict prevents them from being processed concurrently. *It is conflict that limits concurrency.*

If there were two pieces of equipment, concurrency would be partially limited. Two orders could be processed at a time, but there are three; the third would have to wait until one of the two has returned its equipment to `Equip Pool`. If there were three or more pieces of equipment, there would be no conflict in the net as shown, but increasing the number of orders to be processed would cause renewed conflict for equipment and possibly for staff as well.

Conflicts and Bindings

It may seem strange to describe “orders” as being in conflict. This is really just a shorthand for saying that the activities of processing the orders are in conflict.

In SalesNet the three activities of processing the three orders are represented by the transition `Process Orders` in conjunction with

CPN Dynamics: Concurrency and Choice

three enabling bindings. The activities are in conflict because the transition can fire with only one of these bindings at a time.

The reason it can fire with only one binding at a time is that any one firing will remove the equipment token from `Equip Pool`, so that it is not there to be removed by some other firing. Consequently there can be no other firing until the equipment has been returned.

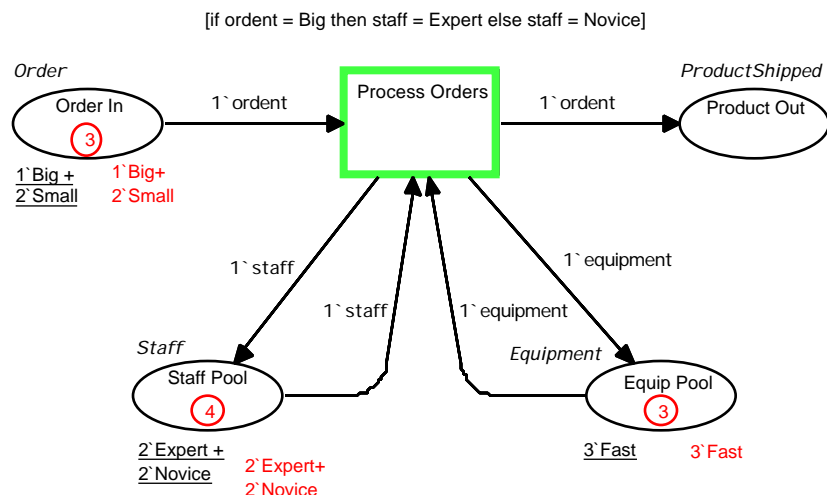
In other words, though there are three enabling bindings, only one of them at a time can actually be used. As soon as the transition fires with any one of them, there is no equipment token for use by any other. The other enabling bindings have in fact disappeared, since any enabling binding for `Process Orders` must include a piece of equipment and there is no longer any such piece.

When the transition has finished firing, restoring the equipment, two enabling bindings will remain, only one of which can actually fire; and so on, until no enabling bindings remain.

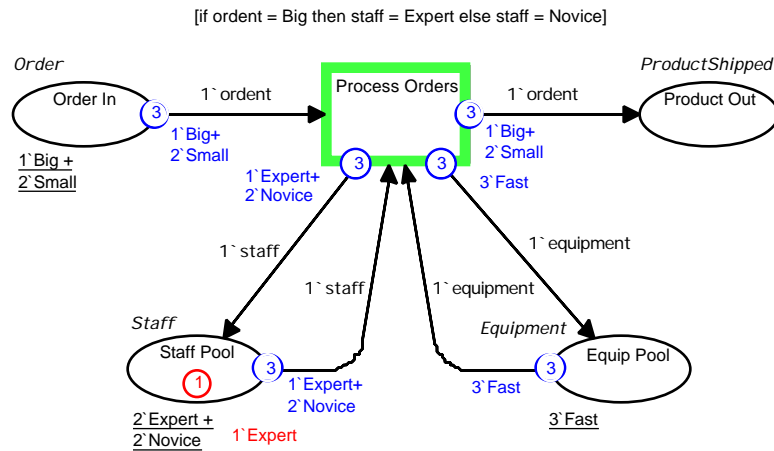
Concurrent Execution of SalesNet

Let's take a look at a concurrent execution of SalesNet. We'll look first at the case where there are enough resources to prevent conflict, then cut the number down so that conflict arises.

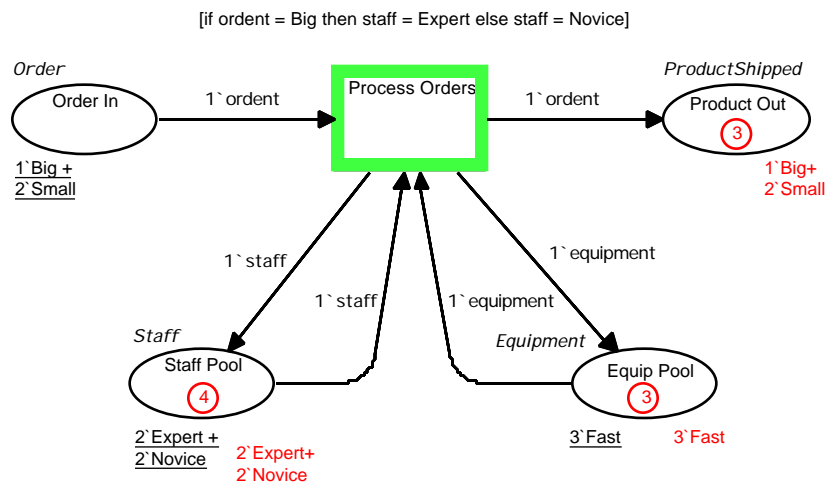
Initial State of the Net



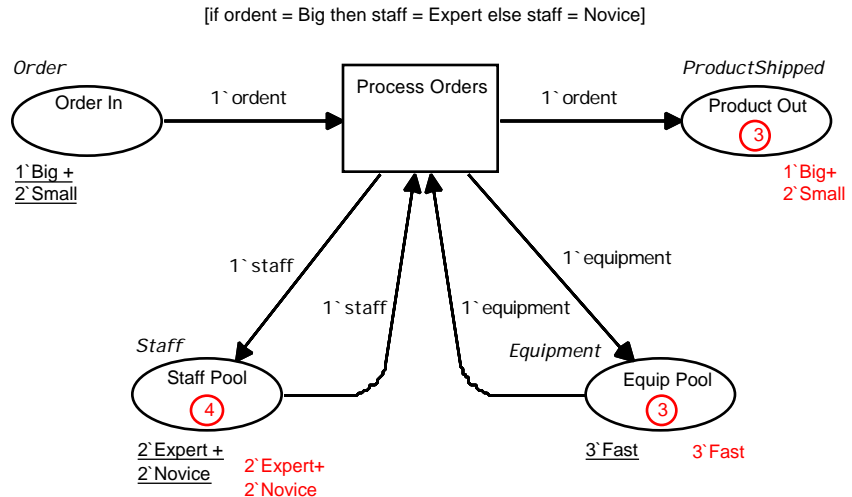
Breakpoint 1: Beginning of Substep



Breakpoint 2: End of Substep



Execution Is Complete



Changing a Net in the Simulator

The execution of SalesNet would be more informative if we cut back to two pieces of equipment. Then we could watch the orders compete for them. To accomplish this, we need to change the initial marking of Equip Pool. There is no need to return to the editor to make minor changes such as this. They can be made directly in the simulator, as follows:

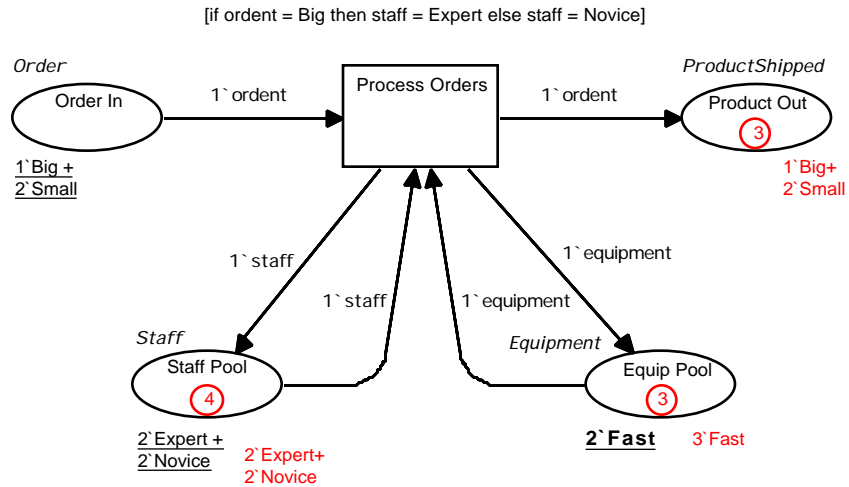
Select the initial marking region.

Enter text mode.

Edit the region.

Leave text mode.

After the initial marking region of Equip Pool has been edited to reduce the equipment number from three to two, SalesNet looks like this:



After you make a change in the simulator, the parts of the net that you have changed must be syntax checked again, and new executable code must be generated for them. This process is called *reswitching*.

Choose **Reswitch** from the **Sim** menu.

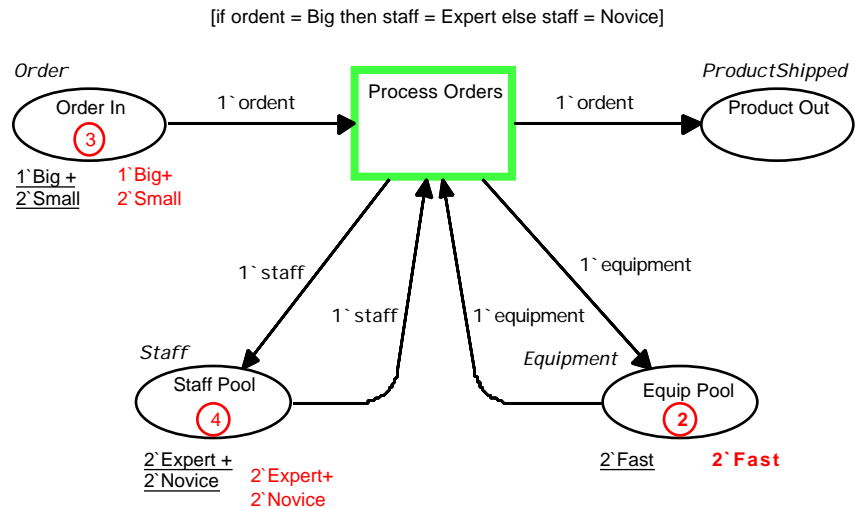
Executing the reswitch does not change the marking of `Equip Pool` has not changed because you changed its initial marking region. The reason is that the creation of initial tokens as specified by an initial marking region happens only when a net's initial state is established on entry to the simulator, or via the **Initial State** command.

Choose **Initial State** from the **Sim** menu.

When the new initial state has been established, the status bar displays Finished Initializing State.

CPN Dynamics: Concurrency and Choice

There are now two `Fast` tokens in `Equip Pool`, as specified by its initial marking region:



You can now run the net just as if `Equip Pool`'s initial marking had always been 2^2Fast . All effects of previous executions were erased when you executed **Initial State**.

The Simulator's Execution Algorithm

When a net is given to the simulator for execution, the simulator does the following:

1. Evaluate any initial marking regions and put tokens into places as the regions specify.
2. Scan the net and make a list of all transitions for which there exists at least one enabling binding. This is the *enabled list*.
3. Do the following:
 - 3A. Scan the enabled list and construct a list of transitions and enabling bindings such that there is no conflict among the bindings. Such a list is called an *occurrence set*. Its elements are called *binding elements*.
 - 3B. For every binding element in the occurrence set, fire the transition it indicates with the binding it indicates. Since the bindings are nonconflicting, all these firings can happen concurrently.
 - 3C. Update the enabled list by rechecking the enablement of all transitions whose input places were

changed by the firings just completed. (Transitions whose input places have not changed do not need to be rechecked.)

4. If the enabled list is not empty after the update in 3C, repeat from 3A.
5. If the enabled list is empty, terminate execution. (You can also terminate execution at any time by pressing ESC.)

Each iteration of 3A, 3B, and 3C is called a *Step*. The firing of an individual binding element is called a *Substep*.

Executing SalesNet With Conflict

Let's trace the execution of SalesNet with insufficient equipment, and look at everything the simulator does as it follows its algorithm.

1: Establish Initial Markings

This has already been done by the **Initial State** command. The resulting markings are shown in the above figure.

2: Put All Enabled Transitions on the Enabled List

This has already been done by the **Initial State** command. There was at least one enabling binding for `Process Orders`, so it is enabled and has been put on the enabled list. The simulator indicates the transition's enabled status by highlighting it, as shown in the above figure.

3A: Scan the Enabled List and Construct an Occurrence Set

Choose **Interactive Run** from the **Sim** menu

The status bar displays Constructing Occurrence Set. The simulator now examines all transitions on the enabled list, and constructs a list of transitions with enabling bindings such that there is no conflict among the bindings. This list is the occurrence set for the step that is currently underway.

When several transitions exist and have mutual conflicts, adjudicating the conflicts can become quite complex, but the situation is fundamentally no different when only one transition is involved: the same sorts of decisions have to be made, and the same rules are followed in making them. In the current case there is only one

CPN Dynamics: Concurrency and Choice

transition, `Process Orders`, and there are three enabling bindings for it. There is one binding:

```
ordent: Big
staff: Expert
equip: Fast
```

And there are two identical bindings:

```
ordent: Small
staff: Novice
equip: Fast
```

Any two of these bindings can fire concurrently, because there are two (and only two) `Fast` tokens available. There is no way to prefer one binding over another, so the simulator chooses any two at random. This is the choice that must be made whenever conflict exists. Since it is made at random, the result is indeterminacy.

In this case the indeterminacy is of little import, due to the simplicity of the net, but a more complex net could behave very differently in future steps depending on what orders are processed now and what orders are forced to wait.

3B: Execute the Elements in the Occurrence Set

Chapter 6, “CPN Dynamics: Executing a CP Net,” described the algorithm for firing a transition as follows:

1. Rebind any CPN variables as indicated by the enabling binding.
2. Evaluate each input arc inscription. The result is a multiset of tokens called input tokens.
3. Evaluate each output arc inscription. The result is a multiset of tokens called output tokens.
4. Subtract each multiset of input tokens from each input place.
5. Add each multiset of output tokens to each output place.

It was convenient but imprecise to describe this as the algorithm for firing a transition. It is more correct to call it the algorithm for executing a binding element. When an occurrence set has only one element (as they all did in previous chapters) the distinction is unimportant, but now we can be more precise.

Executing an Occurrence Set

Since the elements in an occurrence set are not in conflict, they can be executed in any order. On a multiprocessing computer they could all be executed simultaneously by different processors. This would be faster than executing them in some order, but neither the presence of some particular order nor the absence of any order makes any difference, because the net ends up in the same state in any case.

This lack of dependence on, or even need for, an ordering for binding element execution allows the simulator to intersperse such executions. This makes possible the breakpoints **Beginning of Substep** and **End of Substep**.

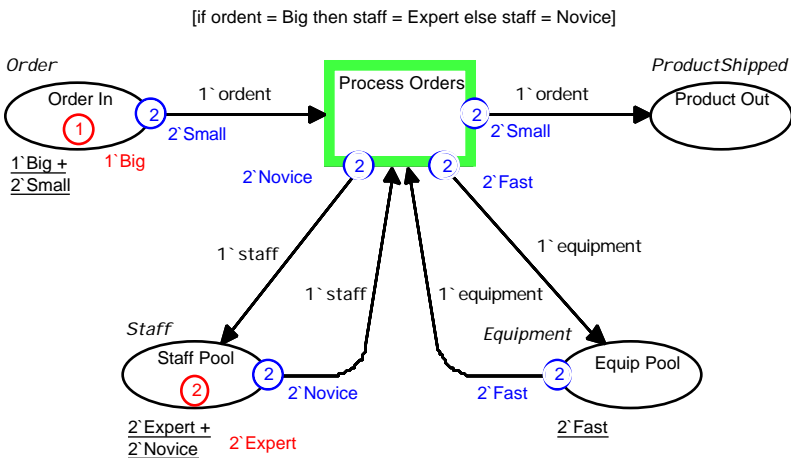
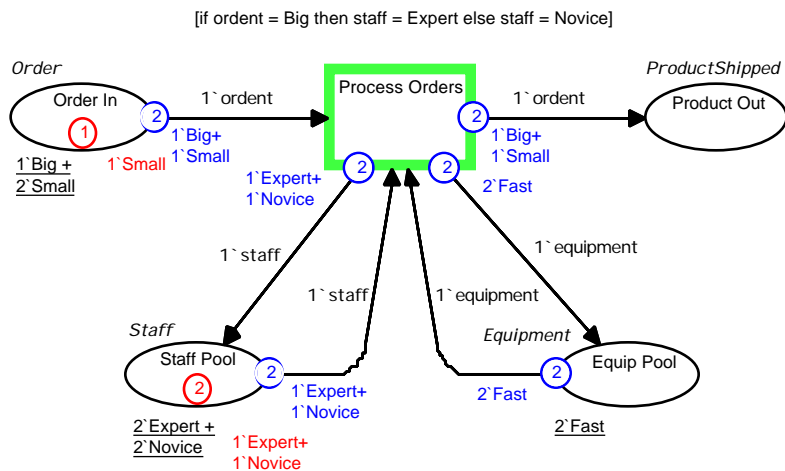
The algorithm for executing an occurrence set is:

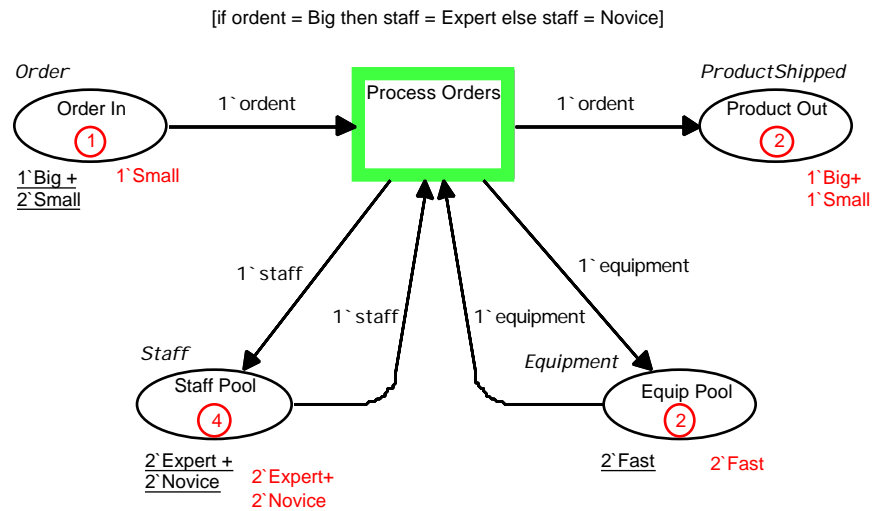
1. For each element in the set:
 - 1A. Rebind any CPN variables as indicated by the enabling binding.
 - 1B. Evaluate each input arc inscription.
 - 1C. Evaluate each output arc inscription.
2. If **Show Input Tokens** is set, display the input tokens.
3. If **Show Output Tokens** is set, display the output tokens.
4. If the breakpoint **Beginning of Substep** is set, pause execution.
5. When execution continues:
 - 5A. Subtract each multiset of input tokens from each input place.
 - 5B. Add each multiset of output tokens to each output place.
6. If the breakpoint **End of Substep** is set, pause execution.

Execution of the occurrence set is now complete. When net execution continues, the simulator will recheck enablement, as described below.

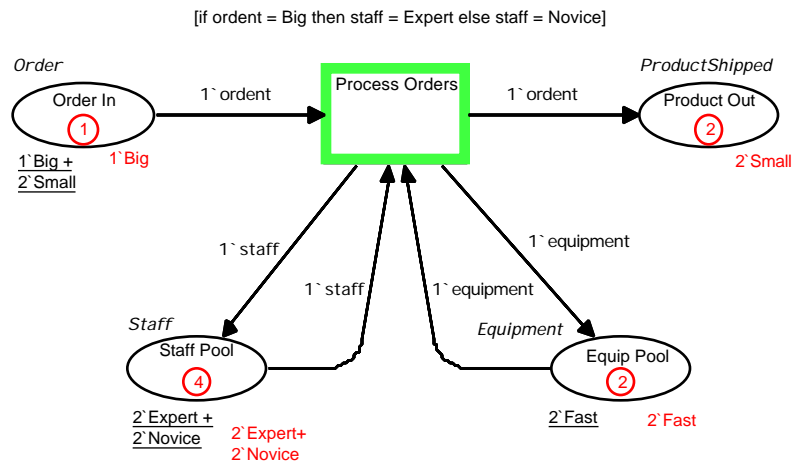
SalesNet's Appearance at Breakpoint 1

When you chose **Interactive Run** above, the simulator constructed an occurrence set and began to execute it. Breakpoint 1 is set, so execution has paused after creating and displaying the input and output tokens.





If your simulator constructed an occurrence set in which both elements specified `ordent = Small`, in which case the net you see will look like this:



Execution of the occurrence set is now complete.

3C: Update the Enabled List

Choose **Continue** from the **Sim** menu.

Transitions whose input places have not changed need not be rechecked for enablement, because their status cannot have changed. `Process Orders` has a changed input place, so it will be rechecked and found to be enabled. It is therefore again highlighted.

With the enabled list has been updated, the step is over. The break-point **Between Steps** is set, so the **Step Finished** dialog appears



4: Continue Execution

Click **Cont**.

Since `Process Orders` is enabled, the simulator executes another step (3A-3C). This step involves no choices, since there is only one order to be processed, but the stages are the same.

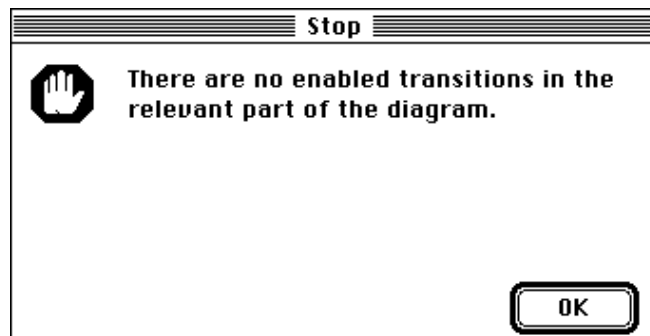
Continue execution through the two breakpoints

The **Step Finished** dialog reappears.

5: Complete Execution

Click **Cont**.

Now the situation is different. When the simulator rechecked enablement, it found that there is no enabling binding for any transition. It therefore displays a dialog that states this fact:



Click **OK**.

Controlling the Appearance of Concurrency

While concurrency is a distinguishing feature of Petri nets, it can get in the way when we need a very detailed and localized view of exactly what a net is doing. This is particularly true when a net is being debugged. When dozens of transitions are concurrently enabled, each with dozens of enabling bindings, so much can happen in one execution step that debugging is almost impossible.

Therefore Design/CPN provides a way to control how concurrency is presented to the observer. By setting some parameters, you can control how much, or how little, is done in each step of the simulator's execution algorithm. Such control has no effect on the meaning of the net: it has the same structural and behavioral properties no matter how its execution is made to appear.

By controlling the appearance of concurrency you can in effect look at net execution through a microscope. You can see each individual microevent of net execution, without interference from other things that are happening concurrently. This is exactly what is needed when it is unclear exactly what a net does, and/or why it is not doing what it is supposed to do.

The appearance of concurrency is controlled by tuning the algorithm that Design/CPN uses to construct occurrence sets.

Review of Occurrence Sets

In Chapter 6, "CPN Dynamics: Executing a CP Net," we looked in detail at the simulator's execution algorithm. Part of that algorithm is the construction of an occurrence set. An *occurrence set* is a list of elements called *binding elements*, each of which specifies a binding for a particular transition.

The simulator changes the state of a net by executing the elements in an occurrence set. Executing a binding element consists of rebinding the arc inscription variables of the transition that the element indicates to the values in the binding that the element indicates, and then firing the transition.

By definition, the elements in an occurrence set are not in conflict: the transition firings that the set indicates can all occur in the same step without attempting to subtract more tokens from any input place than currently exist in the place. That is, the firings can occur concurrently.

Constructing an Occurrence Set

When it constructs an occurrence set, the simulator has many decisions to make. Even SalesNet, which has only one transition, pro-

vided the possibility of several different occurrence sets. When there are many transitions with many bindings that conflict in many ways, the number of possible occurrence sets may be very large.

There is no one best algorithm for constructing an occurrence set. One algorithm might result in very fast execution, but obscure the details of net execution by doing too many things at once. Another might execute slowly but illuminate every detail. A third might accomplish both these ends very well, and a fourth very poorly. An algorithm that gives the desired results with one net might be ineffective with another.

Therefore CPN does not require that occurrence sets be constructed in any particular way, or have any particular property other than the nonexistence of conflict among the constituent binding elements. In particular, nothing requires the simulator to construct the largest possible occurrence set, or one of the largest possible. If we want to look at very small increments in the behavior of a net, we might want very small occurrence sets.

The obvious question is: how can we be sure that changing the occurrence set algorithm will not change the meaning of the net itself? How can we be sure that the behavior of a net is invariant of the way in which occurrence sets are constructed? To answer this question, we need a precise definition of concurrency.

What Is Concurrency?

Informally, concurrent activities are those that happen “at the same time”. But what exactly does that mean? If we are to have a useful modeling paradigm, it cannot mean “in the same instant,” because no real activity is instantaneous; there is always some duration. Nor can it mean “occurring during exactly the same interval,” because rarely if ever could this property be guaranteed for different activities at different physical locations.

CPN defines concurrency as follows: A collection of activities is *concurrent* if the result of their occurrence is unaffected by the presence or even the existence of any particular occurrence order. That is, the defining property of concurrency is not that activities *do* overlap in time, but only that they *can*. It makes no difference whether they actually do or not, and if they do, it makes no difference just how they overlap. When all are complete, the result will be the same regardless of all such details.

As a result of this definition, we have complete freedom to construct occurrence sets in any way we like. We know that all the events we might put into an occurrence set can happen concurrently, because they are guaranteed not to be in conflict. Therefore, by the definition of concurrency, no ordering of events that results from this or

that choice of occurrence sets makes any difference, because the ordering does not matter at all.

Similarly it makes no difference whether we execute a set of concurrent events by creating a single occurrence set or several of them. At the extreme we could require that every occurrence set consist of just one binding element, and it would make no difference: the effect would be the same as if we concatenated binding elements in all the sets into a single set, and then executed that set; or into several sets; or in different orders into one or several sets. It makes no difference at all to the outcome.

Occurrence Set Parameters

Most features of the simulator's execution algorithm are determined by the rules of CPN dynamics (Chapter 6, "CPN Dynamics: Executing a CP Net"), and so cannot be changed. But the algorithm by which the simulator constructs occurrence sets can be anything that is useful. Since different algorithms are better with different nets and for different purposes, Design/CPN allows you to specify various features of the algorithm that the simulator uses. This is done by setting parameters called *occurrence set parameters*.

To examine and change these parameters, first:

Choose **Occurrence Set Options** from the **Set** menu.

The **Occurrence Set Options** dialog appears:

Under X-Windows, the slider bars shown in the above dialog do not appear, but the box is otherwise similar.

Transitions

The setting for **Transitions** determines how the simulator will construct occurrence sets when more than one transition is simultaneously enabled on a page.

When the setting is 100%, all enabled transitions will be represented in the set, subject to the restriction that no occurrence set will be constructed that contains conflicting binding elements.

Settings between 1% and 99% define the probability that a particular enabled transition will be represented in the occurrence set. For example, at a setting of 50%, each candidate transition has a 50% chance of being represented. Note that setting a value of 50% does *not* mean that 50% of the candidates will be represented: more or less might be, depending on chance and the requirement to avoid conflicting binding elements in a set.

CPN Dynamics: Concurrency and Choice

There would be no purpose in constructing an empty occurrence set. Therefore at least one transition will be represented in the set no matter what the outcome of any random choices. When the setting is 0%, exactly one enabled transition will be represented in the set.

Different Bindings

The setting for **Different Bindings** determines how the simulator will construct occurrence sets when a given transition is enabled with two or more nonidentical bindings.

When the setting is 100%, all enabling bindings will be represented in the set, subject to the restriction that no occurrence set will be constructed that contains conflicting binding elements.

Settings between 1% and 99% define the probability that a particular binding will be represented in the occurrence set.

Subject to the restriction on conflicting binding elements, at least one binding will be represented in the set no matter what the outcome of any random choices. When the setting is 0%, exactly one of the bindings will be represented.

Identical Bindings

The setting for **Identical Bindings** determines how the simulator will construct occurrence sets when a transition is enabled with two or more bindings that are identical, i.e. consist of the exactly the same values.

Subject to the restriction on conflicting binding elements: when the setting is 100%, all the bindings will be represented in the occurrence set; when it is 0%, exactly one will be; and intermediate settings give proportionate intermediate results.

Scope of Occurrence Set Parameters

Each of the five parameter settings works within the bounds established by those higher in the list. Thus **Transitions** determines how many enabled transitions (if there is more than one) will be represented in an occurrence set; **Different Bindings** determines how many different bindings (if there is more than one) will be included for each represented transition; and **Identical Bindings** determines how many copies (if there is more than one) will be included for each represented binding.

Setting Occurrence Set Parameters

To set any of the occurrence set parameters:

 Edit the number in the box to the right of the name of the particular parameter.

 The value must be between 0 and 100.

Chapter 9

CPN Hierarchy: Introduction

Effective CPN modeling requires the ability to distribute a CP net across multiple pages, so as to divide it into modules small enough to keep track of. Such a module is called a *submodel*.

Distributing a net across multiple pages requires some mechanism for interconnecting the submodels on the various pages, so that the state of one can influence the state of another. Otherwise we would have several disconnected nets rather than one distributed net.

Design/CPN offers two mechanisms for interconnecting CP net structure on different pages: substitution transitions and fusion places. A *substitution transition* is a transition that stands for a whole page of net structure. A *fusion place* is a place that has been equated with one or more other places, so that the fused places act as a single place with a single marking.

Substitution transitions and fusion places together provide a very general capability for organizing a CP net into submodels. This capability is called *CPN Hierarchy*.

Definition of Hierarchical Decomposition

The SalesNet model represents a very high-level view of the system it models. Such a view can be useful, of course; but it would also be useful to have more detailed information about how orders are processed, staff and equipment used, and products shipped. Ideally we would like to represent this additional information without having to lose the simplicity of the high-level overview that SalesNet currently provides.

In order to add detail to a model without losing overview, a transition may have associated with it a separate page of CP net structure called a *subpage*. This page contains a more detailed view of the activity that the transition represents. Such a transition is called a *substitution transition*. The details on the subpage are called the decom-

position of the transition. The page that holds the transition is called the *superpage*.

Transitions on a subpage may in turn have associated subpages, and so on. Since this method of representing details results in a hierarchy of subpages that contain decompositions, it is called *hierarchical decomposition*.

CPN Hierarchy

Models whose primary purpose is educational, or that represent very small systems, can often be drawn on a single page. However this practice would not suffice for making a realistic model of a large and/or complex system. Trying to model such a system on a single page would be like trying to write a complex program without using subroutines. Such a model, or such a program, would be far too complex, poorly structured, and redundant to be useful, or in many cases even constructible.

The answer is to allow a CP net to be kept on multiple pages that can be organized into a functioning whole, much as an ordinary program can be written as multiple modules that can be linked into an executable file. The system CP nets use to provide such modularization is known as *hierarchy*.

CPN hierarchy consists of two capabilities: fusion places and substitution transitions. These capabilities allow a net to be divided into modules, and provide facilities for linking the modules in various ways.

Fusion Places

The fusion place capability allows CP net places that exist in different locations in a net to act functionally as if they were the same place. Such places are called *fusion places*, and a group of such places is called a *fusion set*.

Fusion places are similar to global variables in a conventional program. Just as references to a global variable by different parts of a program refer to the same variable and yield the same value, so uses of “different” places in a fusion set by different parts of a net are actually uses of the same place, and will find that place to have the same marking.

Fusion places can be used in many ways to simplify and generalize a net. For example, a net might model many different activities that all make use of the same pool of resources. Representing the pool as a

fusion place would allow the various activities to be drawn on different pages and yet all have access to a single shared resource pool.

Fusion places are also useful in the context of a single page. When many arcs connect to a place, and these arcs come from physically distant locations on the page, it is often clearer to represent the place more than once on the page, and equate the various representations by including them all in a fusion set.

Substitution Transitions

The substitution transition capability allows a CP net transition to represent an entire page of net structure. The effect is the same as if the page that the transition represents appeared physically at the site of the transition. Such a transition is called a *substitution transition*, and the page of net structure that it represents is called a *subnet* or a *submodel*.

Substitution transitions are similar to subroutines in a conventional program. The effect is not identical, because the substitution occurs physically, as with a macro, rather than by invocation, but the result is essentially the same:

1. A net can be implemented as multiple modules that can be modified independently of each other.
2. The various modules can be linked together as needed to create a net.
3. The same module can be used repeatedly at different places in a net, so that redundant logic need not be created to handle the same situation in different contexts.

For example, a net might model a computer installation with many identical workstations working in parallel on different tasks. Creating a submodel that represents the details of a workstation, and using that submodel as the value of many different substitution transitions, would allow just one submodel to represent all of the workstations.

Top-Down and Bottom-Up Development

There are two different ways to specify a decomposition for a transition:

1. Create a page containing a submodel, then link the submodel to the transition. The page then becomes a subpage, and the

transition becomes a substitution transition on a superpage. This is *bottom-up development*.

2. Start with a transition, have Design/CPN create a subpage for it, then edit the subpage to create the submodel. This is *top-down development*.

Neither of these methods is intrinsically preferable. The choice of which to use is largely a matter of how one prefers to work.

Chapter 10

CPN Hierarchy: Fusion Places

In the CP nets we have worked with so far, each place has been an independent entity: there was no relationship between places except that provided by arcs and transitions.

Another form of relationship is possible in a CP net. We can establish a method for defining sets of places so that anything that happens to each place in a set also happens to all the other places in the set. The places are then functionally identical. Such places are called *fusion places*, and a set of fusion places is a *fusion set*.

Fusion adds nothing fundamentally new. If all the members of a fusion set are on the same page, we could replace the set with a single place and connect to it all the arcs that connected to any member of the set. If the members are on different pages, we could copy everything on the several pages to a single page, and again collapse the set.

Conversely, if a net contains a place that has many arcs connecting to it, or requires very long arcs to reach it, we could unfold it into several places, on the same or different pages, and so simplify the net's graphical structure without changing its meaning. Such unfolding is a common event during the process of CP net development. Frequently the need for it can be anticipated, and fusion places used from the beginning.

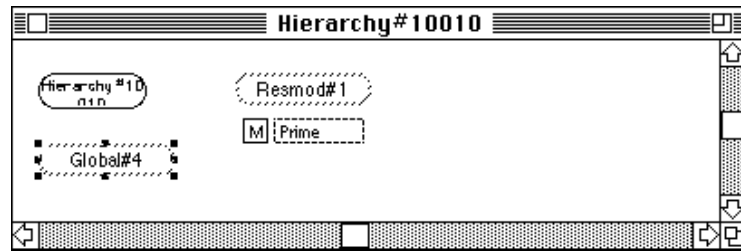
We really need only one type of fusion place to derive all the benefits fusion can provide. But as we shall see, it is useful to have different types that have different scopes. The rest of this chapter shows how to use fusion places.

The Resource Use Model

This chapter demonstrates fusion place techniques using a model called the Resource Use Model. Let's take a quick look at this model before we begin working with it.

Design/CPN User's Guide

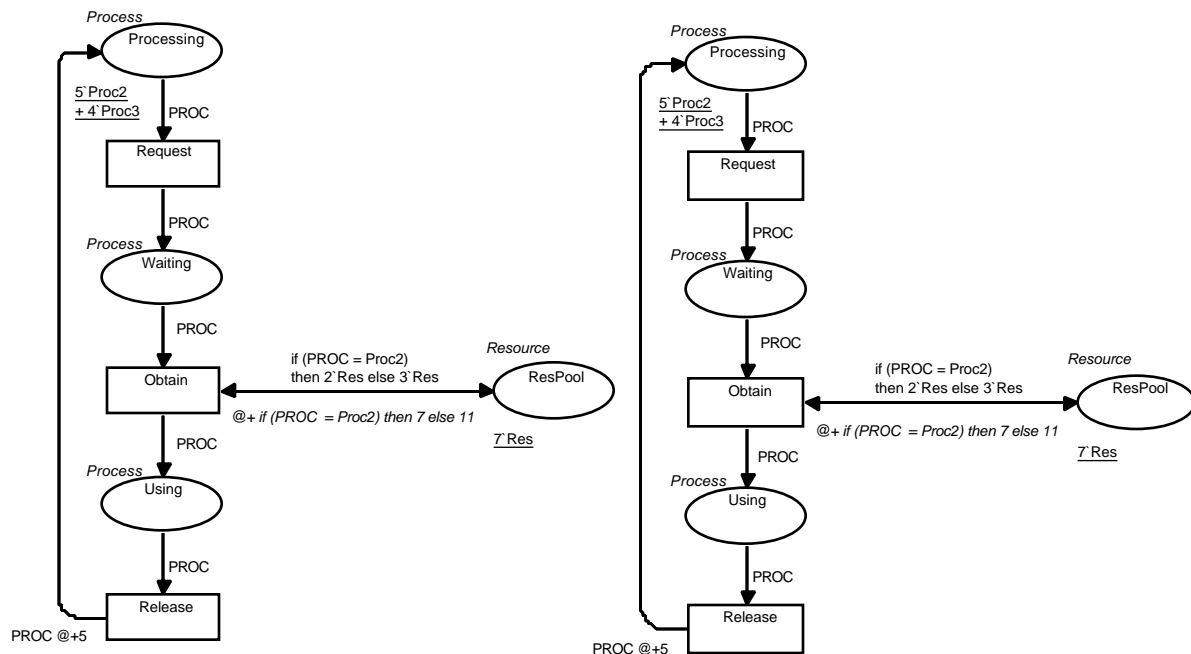
The hierarchy page:



The page Global#4 contains the global declaration node:

```
color Process = with Proc2 | Proc3 timed;  
var PROC:Process;  
color Resource = with Res timed;
```

The page Resmod#1 contains the executable part of the model:



Description of the Model

Resmod#1 contains two identical copies of the Resource Use Model. This model represents a simple computer system in which processes vie for resources. Each process goes through a cycle in which it develops a need for resources, requests them, possibly waits for them, then obtains, uses, and releases them.

There are two kinds of processes, `Proc2` and `Proc3`, as defined by the colorset `Process`. There is only one type of resource, `Res`, defined by the colorset `Resource`. A `Proc2` process uses two `Res` resources at a time, and a `Proc3` process uses three `Res` resources at a time, as defined by the inscription on the input/output arc between `Obtain` and `ResPool`. A `Proc2` process uses resources for 7 time units, and a `Proc3` process uses them for 11 time units, as defined by the time region on the transition `Obtain`.

The rest of this chapter refers to the Resource Use Model as Resnet, and the two copies as Resnet1 (on the left) and Resnet2 (on the right).

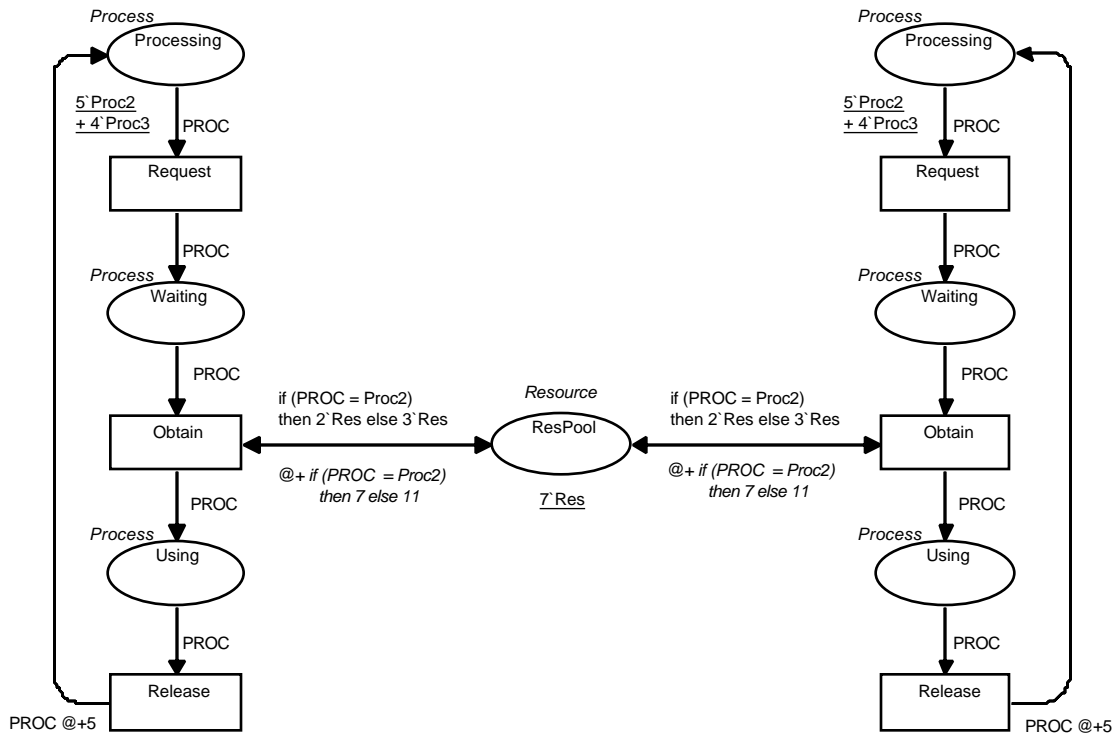
Results of Executing the Model

If the model were executed as it is now, what would happen? Since there is no functional connection between the two copies, they would execute side by side just as if each existed alone. They would have no effect on each other at all. We could consider the copies to be two independent nets or a single net with two disjoint components. Nothing requires a CP net to be connected, so the two interpretations are equally valid.

However, it would obviously be useless to have multiple identical independent nets executing simultaneously. None of them would do anything that the others would not do; the only effect would be to slow execution down.

Fusion on a Single Page

To study fusion places we will first look at an example in which Resnet1 and Resnet2 share a common pool of resources. We could of course do this by deleting one of the resource pools and connecting the other to both copies. The result could look something like:



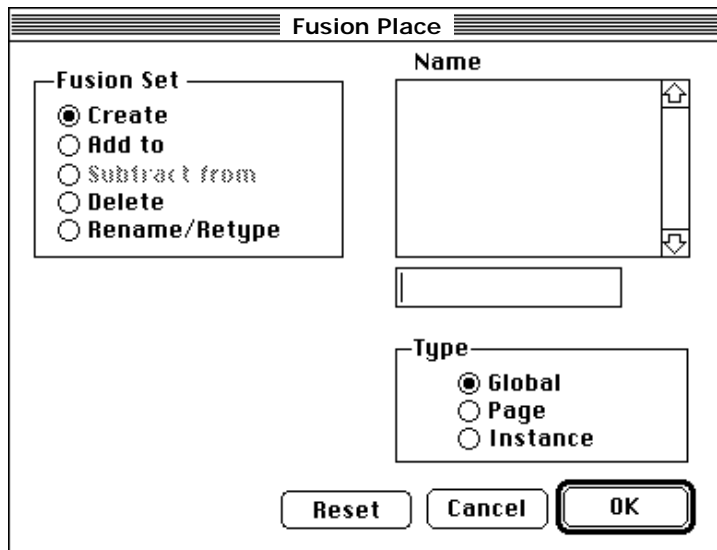
This technique would be enough for the simple case we are working with, but if the resource pool were needed in many more locations that were much farther apart on the page, a lot of very long arcs would be needed. These would clutter up the page. It would be better to leave the resource pools where they are, and combine them functionally into a single pool by putting them in a fusion set.

Creating a Fusion Set

Select the first place to include in the fusion set. For example, ResPool in Resnet1.

Choose **Fusion Place** from the **CPN** menu.

The **Fusion Place** dialog appears:

The image shows a dialog box titled "Fusion Place". It has two main sections. The "Fusion Set" section on the left contains five radio buttons: "Create" (selected), "Add to", "Subtract from", "Delete", and "Rename/Retype". The "Name" section on the right has a large text area with a vertical scrollbar and a smaller text input box below it. Below the text input box is a "Type" section with three radio buttons: "Global" (selected), "Page", and "Instance". At the bottom of the dialog are three buttons: "Reset", "Cancel", and "OK".

This dialog allows you to create and edit fusion sets. **Create**, the operation we want, is already selected as the default.

When you create a new fusion set, you must give it a name and indicate its type. For example, the name of the new resource model fusion set will be "RPOOL".

Type the name of the new fusion set into the edit box (immediately below the **Name** section). For example, "RPOOL".

A fusion set that simply equates all constituent places wherever they occur is called a *global fusion set*. Since **Global** is the default type in the dialog, it isn't necessary to do anything explicit to indicate the new set's type. (**Page** and **Instance** fusion sets will be explained later in this chapter.)

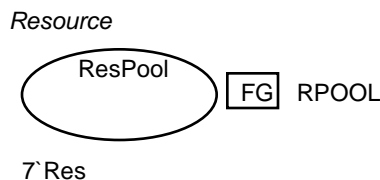
Click **OK**.

The dialog disappears.

The example global fusion set named **RPOOL**, contains one place, **ResPool**. A place that is a member of a global fusion set is called a *global fusion place*.

Physical Appearance of a Global Fusion Place

In the example, the status bar lists the type of **ResPool** as Place, Global-Fusion. **ResPool**'s appearance changes to include a fusion place indicator:



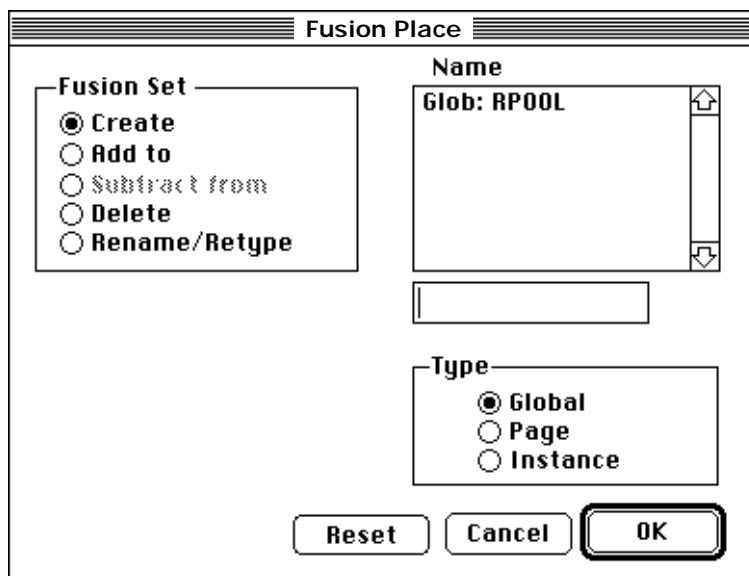
The **FG** marker is a *Fusion Key Region*. It indicates that the place is a global fusion place. The name of the fusion set to which the place belongs, **RPOOL**, is indicated next to the Fusion Key Region, in another region called the *Fusion Region*. A Fusion Region is a popup, so you can hide and redisplay it by double-clicking on the associated key region.

Adding Places to a Fusion Set

Select the other place(s) to be added to the fusion set. For example, **ResPool** in Resnet2.

Choose **Fusion Place** from the **CPN** menu.

The **Fusion Place** dialog reappears:



In this example, note the listing of **Glob: RPOOL** in the box under **Name**. This box lists all existing fusion sets, both to help you keep track of what they are, and to let you select an existing set without retyping its name.

Click **Add To** under **Fusion Set**.

The edit box and the **Type** section disappear, since you can add only to a fusion set that already exists.

CPN Hierarchy: Fusion Places

Click on the desired fusion set name in the box under **Name**. In this example, **glob: RPOOL**.

Click **OK**.

The dialog disappears. The selected place is now part of the fusion set. The places in set are functionally not two places, but one place that is physically represented several times in several different locations.

Multiple places may be selected when a fusion set is initially created, thereby eliminating the need to first create and then add to a fusion set.

Initial Markings and Fusion Sets

Both places in **RPOOL** have an initial marking region, and each region specifies the same marking. In the editor, they could just as well have had different markings: they are just text regions, so it does not matter if they agree or not.

The simulator is another matter. When you enter the simulator, any initial marking regions are evaluated, and the tokens they specify are put into the corresponding places: marking regions are converted into actual markings. But the places in a fusion place, being functionally one place, intrinsically can have only one marking. Therefore all places in a fusion set must have the same initial marking region (if any) before entry to the simulator. If they do not, a syntax error will occur, and you will remain in the editor.

Removing Places from a Fusion Set

Removing places from a fusion set is just the inverse of adding them.

Select the place(s) to be removed from the fusion set. Be sure no other places are selected.

Choose **Fusion Place** from the **CPN** menu.

Subtract From is already selected under **Fusion Set**. That is the operation we want, so:

Click **OK**.

The place removed from the fusion set is now an ordinary place.

If the place removed from the fusion set is the last place in the fusion set, the fusion set is deleted.

Shortcut: you can remove a place from a fusion set by deleting its Fusion Key Region.

Deleting a Fusion Set

It is sometimes useful to delete a fusion set. All of its constituent places then revert to independent status.

Choose **Fusion Place** from the **CPN** menu.

Click **Delete** under **Fusion Set**.

Select the fusion set to be deleted from the **Name** box. For example, **Glob: RPOOL**.

Click **OK**.

In this example, **RPOOL** is gone; all **ResPool** places are now free-standing, as if **RPOOL** had never existed.

Removing the last place from a fusion set also deletes the fusion set.

Fusion Across More Than One Page

In the context of a single page, fusion does not add any fundamental power, since anything fusion can do on a single page could be done by drawing arcs. Fusion on a page can be very convenient, but the real purpose of fusion is to make it possible to establish connections between net structures that exist on different pages. Without fusion there would be no way to do this, because there is no way to draw an arc that runs between one page and another. With fusion we can equate places on one page with places on another, and so connect the pages.

Working with a fusion set that equates places on more than one page is essentially the same as working with a single-page set. The only differences result from the fact that it is impossible to form a group of places (or anything else) that extends across pages.

To create a fusion set that spans multiple pages:

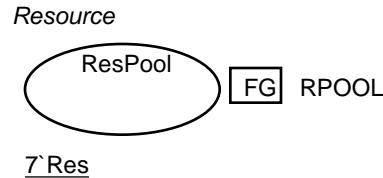
Create a fusion set from the places on one page.

Make the next page with places to be added to the fusion set the current page.

Add those places to the newly-created fusion set.

Repeat the second and third steps as many times as there are pages with places to be added to the fusion set.

For example, if there were three Resnets in the example described above, and one was on a different page, we could not create a fusion set containing all three ResPools in one operation. We would need to follow the procedure above to create the RPOOL fusion set. All three ResPools would have the same appearance, however:



This is just how the two ResPools on Resmod#1 looked when RPOOL equated places on only one page. Its extension to equate places on more than one page adds nothing new.

Working With More Than One Fusion Set

The existence of more than one fusion set adds nothing fundamentally new. The only requirement is that each fusion set must have a unique name.

Page Fusion Sets

A shared resource pool is often a good idea. For example, we would not want every printer user to have its own dedicated printer if one shared printer would be enough to serve them all. On the other hand, complete resource sharing is not always desirable. A computer network might be distributed through several buildings, but its users would probably want print jobs to be done only on printers in their own building. How can Resmod#1 be modified to model this kind of situation.

The obvious answer is to use multiple fusion sets. But if a large model required many similar but separate fusion sets, this technique would result an annoying proliferation of different names for different instances of essentially the same thing.

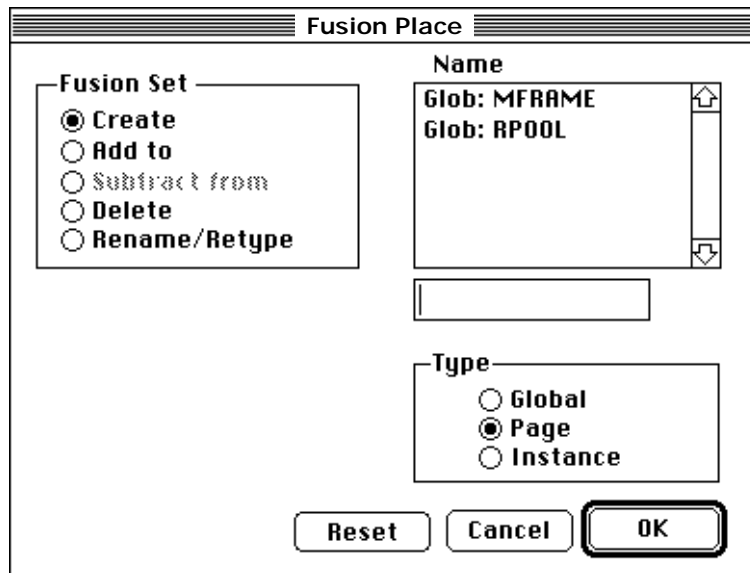
There is a better way: *page fusion sets*. Page fusion sets are identical with global fusion sets (the kind we have been working with so far) in every way but one: while a global fusion set equates every constituent place irrespective of page, a page fusion set is divided

Design/CPN User's Guide

into subsets, each of which equates only constituent places that are on the same page.

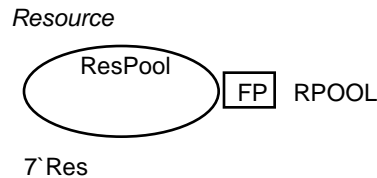
Put another way, a page fusion set is a collection of fusion sets that have the same name but exist on different pages. The members of such a collection are called *page fusion subsets*, and each constituent place is called a *page fusion place*.

Page fusion sets are created and modified much as global sets are. The only difference occurs at the beginning: when a page fusion set is first created, select **Page** rather than **Global** in the **Type** section of the **Fusion Place** dialog:



Once a page fusion set exists, all the techniques you have just used for global fusion sets will work on it exactly as they would if the set were a global set.

All of the places in a page fusion set look the same. For example:



Note the change from **FG** to **FP**, indicating the change from a global fusion set to a page fusion set. There is nothing in the place's appearance to indicate which page fusion subset the place belongs to: its presence on the particular page is indication enough.

Relationship Among Page Fusion Constituents

- All members of a page fusion set must have the same colorset
- The members of a page fusion set on any one page must agree as to initial marking
- The members of a page fusion set on different pages can have different initial markings.

Instance Fusion Sets

Page fusion sets require the various fusion subsets to be physically represented on different pages. A few copies of a page is not that bad, but suppose we want to model a situation in which there are many hundreds or thousands of instances of the same entity, and that this entity is too complex to represent with a token but must be modeled with a piece of net structure? We would not want to put hundreds or thousands of copies of the same structure on a page, or worse yet, have hundreds or thousands of copies of the same page. Either method would render the net completely unmanageable.

To deal with such situations, Design/CPN allows you to create a single page in the editor, and then use that page as many times as needed in the simulator. This allows us to have it both ways: physically there is only one page, but functionally there can be as many copies of the page as we need. This capability is called *multiplicity*, and the different functional copies of the same page are called *page instances*.

As we noted above, completely disjoint copies of the same net structure would serve no purpose, since none would do anything the others do not do. The same argument applies when the copies are multiple instances of a page: the instances must be interconnected in some way to form a larger whole, or there is no reason to have them. As with physically separate pages, the interconnection is accomplished by using fusion places.

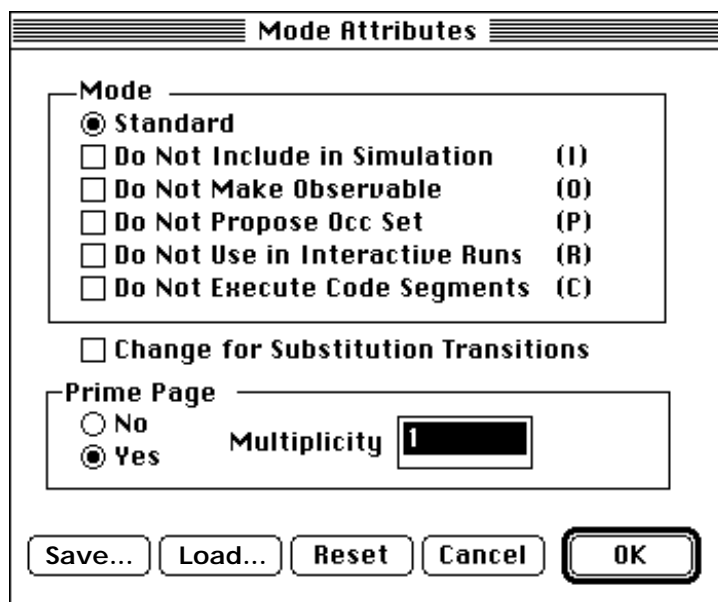
Creating Multiple Page Instances

Open the hierarchy page.

Select the page node for which multiple instances are needed. For example, Resmod#1.

Choose **Mode Attributes** from the **Set** menu.

The **Mode Attributes** dialog appears:

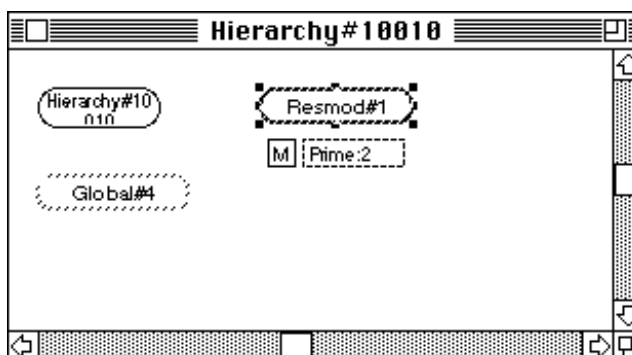


In the example above the figure for **Multiplicity** is 1, indicating that there is only one instance of Resmod#1.

Type the number of page instances desired. For example, "2".

Click **OK**.

In our example there are now two instances of Resmod#1. The hierarchy page now looks as follows:



Note the new designation Prime:2 for Resmod#1. The "2" indicates that there are now two instances of the page.

Multiplicity and Fusion

Let's take a moment to look at the overall structure of the model, with particular attention to relationships between fusion sets and multiple page instances.

The model `ResourceModel` has one executable page: `Resmod#1`. There are furthermore two instances of `Resmod#1`. These are of course identical with each other. There are physically two copies of `Resnet`; but functionally there are four copies, because each copy on `Resmod#1` exists twice, once on each instance of the page.

There is one fusion set: **RPOOL**. **RPOOL** is a page fusion set. What should it do with the `ResPools` in the two instances of `Resmod#1`?

There are two possibilities, equally reasonable:

1. Form a single fusion subset that includes all instances of the page, on the grounds that there is really only one page and a page fusion set, by definition, equates all constituent places on each page.
2. Form a separate fusion subset for each instance of the page, on the grounds that the instances are so much like separate pages that they deserve separate fusion subsets.

In practice, there are situations where the first method would more useful, and situations when the second would be. Therefore Design/CPN allows both possibilities. Page fusion sets conform to the first possibility: there is one page fusion subset for all instances of a page. To create a separate fusion subset for each instance of a page, we use the third type of fusion set: the *instance fusion set*.

Comparison With Page Fusion Sets

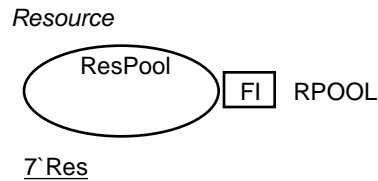
Instance fusion sets are identical with page fusion sets in every way but one: while a page fusion set equates every constituent place irrespective of page instance, an instance fusion set is divided into subsets, each of which equates only constituent places that are on the same page instance.

In other words, an instance fusion set is a collection of fusion sets that have the same name but exist on different page instances. The members of such a collection are called *instance fusion subsets*, and each constituent place is called an *instance fusion place*.

Working With Instance Fusion Sets

Instance fusion sets are created and modified much as global and page sets are. The only difference occurs at the beginning: when you first create an instance fusion set, select **Instance** in the **Type** section of the **Fusion Place** dialog.

In the example below, note the change in the appearance of the `ResPool` place:



The fusion key region contains FI, indicating instance fusion.

Instance sets are created on a per-page basis. For example, if **RPOOL** is a page instance set spread over two pages, each page is treated separately. Thus **RPOOL** could be a page fusion set on one page and an instance fusion set on another.

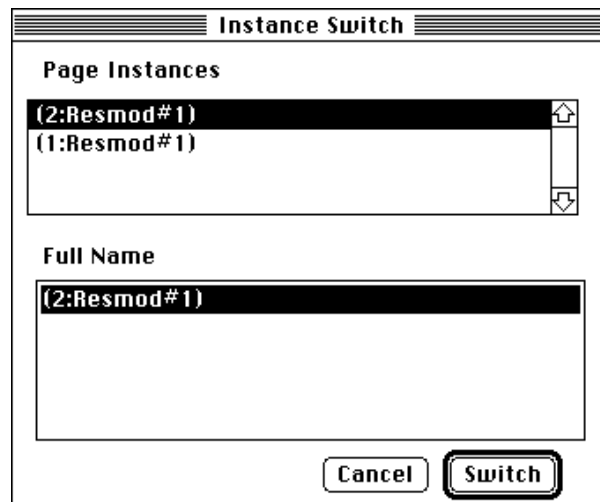
Observing Fusion Across Multiple Instances

To switch between instances of a page, use the **Instance Switch** dialog. To activate it:

Depress the SHIFT key.

Click the mouse on the page title bar.

You can get the same effect by choosing **Select Instance** from the **Sim** menu. In either case, the **Instance Switch** dialog appears:



In the example above, all the instances of Resmod#1 are listed under **Page Instances**. The instance currently on display in the Resmod#1 window is highlighted. Note that the highlighted name matches the name in the Resmod#1 window's title bar. The title bar of a window that displays a page with multiple instances always tells which instance is currently visible.

CPN Hierarchy: Fusion Places

Use the mouse to select another instance.

Click **Switch**.

The dialog disappears. The title bar now indicates that the other instance is on display. At the beginning of a simulation there won't be any other change, because both instances of the page are in the same initial state. Once simulation begins they will diverge.

Chapter 11

CPN Hierarchy: Substitution Transitions

In CP nets that do not use substitution transitions, each transition is a fundamental unit: there is no functional significance to a transition other than that defined by any associated places, arcs, arc inscriptions, guard, code segment, and/or time region.

Another form of transition is possible in a CP net. We can establish a method by which a transition can stand for an entire piece of net structure, so that the net containing the transition executes as if the logic that the transition represents were physically present at the location of the transition. Such a transition is called a *substitution transition*.

Substitution transitions add nothing fundamentally new. Everything that can be done with them can also be done by using fusion places. But like fusion places, substitution transitions, add so much convenience that they can make the difference between feasibility and total impossibility.

When a CP net uses a substitution transition, the logic that the transition represents must be kept somewhere. It is kept on a page called a *subpage*, and the logic on the subpage is called a *subnet*, or sometimes a *submodel*. The page that contains the substitution transition is called a *superpage*. Superpages and subpages are connected by equating places on the two pages using special-purpose fusion sets. A place that belongs to such a fusion set is called a *port* if it is on a subpage, and a *socket* if it is on a superpage.

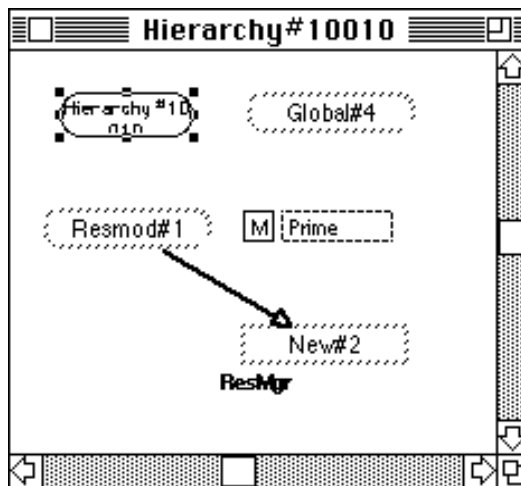
It would be inconvenient to have to create substitution transitions by manually creating the requisite fusion sets, though it could be done. Design/CPN provides extensive capabilities for facilitating the creation and use of substitution transitions. This chapter tells you what those capabilities are, and shows you how to use substitution transitions to create hierarchical CP nets.

Structure of a Model With Substitution

This section illustrates substitution transitions in the context of a model called ResmodSubtrans.

ResmodSubtrans Component Pages

The Hierarchy Page



ResmodSubtrans contains a global declaration page, Global#1, and two executable pages, Resmod#1 and New#2. The arrow linking the two page nodes indicates that Resmod#1 contains a substitution transition representing net structure that is kept on New#2. When two pages are related in this way, the page that contains the substitution transition is called a *superpage*, and the page that contains the net structure that the transition represents is called a *subpage*. The net structure on a subpage is sometimes referred to as a *subnet* or a *submodel*.

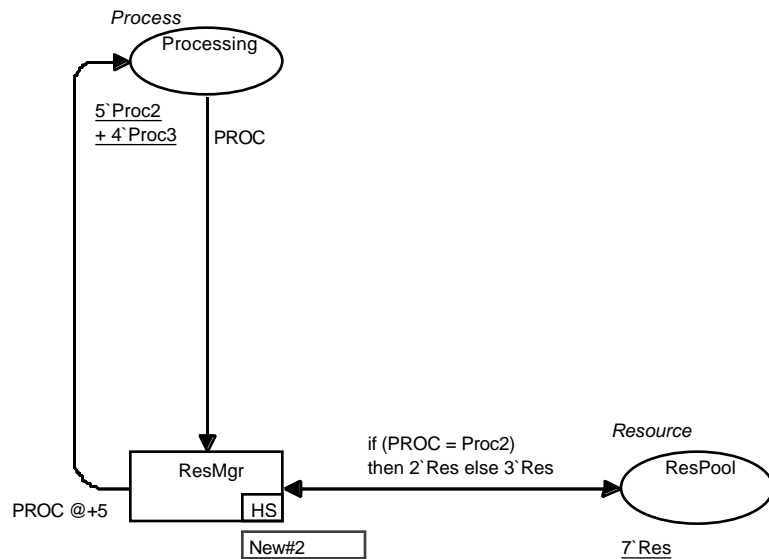
Resmod#1 is a prime page, so the simulator will execute it. New#2 is not a prime page, but it does not have to be: when a superpage is prime, all subpages that it uses are automatically included in simulation.

Below the node for New#2 the string "ResMgr" appears, in a region called a *substitution tag region*. This indicates that the substitution transition in Resmod#1 has a name: ResMgr.

CPN Hierarchy: Substitution Transitions

The Superpage Resmod#1

The page looks like this:



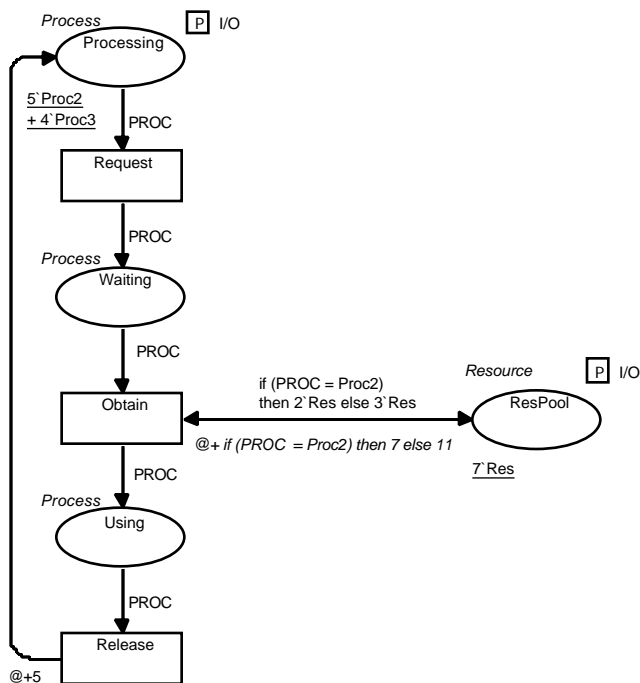
Note that this page is, in and of itself, a functional CP net. If ResMgr were an ordinary transition, and everything else were as shown, we could take this net into the simulator and execute it.

When the place ResPool is selected, the Status Bar describes it as a Place, I/O Socket. This indicates that ResPool is a socket: somewhere, on some other page (by definition a subpage), there is another place, a port, associated with it.

The Subpage New#2

New#2, as we saw on the hierarchy page, is the subpage for the substitution transition ResMgr on Resmod#1, so the ports must be on New#2.

To get to New#2, you could reopen the hierarchy page, then open New#2 like any other page. But there is a more convenient method. By double-clicking on a substitution transition, you can jump directly to the subpage that it represents.



The P is called a *port key region*. It indicates that the associated place is a port. The “I/O” is contained in a region called a *port region*. The “I/O” means that the ports have both input arcs and output arcs connecting to them on the subpage.

In the example above, when the place `ResPool` is selected, the Status Bar describes it as a Place, I/O-Port. This indicates that `ResPool` is a port: somewhere, on some other page (by definition a superpage), there is another place, a socket, associated with it.

Ports and Sockets

How can you tell what port is equated with what socket? In this case it is easy: they have the same name. The `Processing` places on the two pages are one port-socket pair, and the `ResPool` places are another.

If the names did not match, Design/CPN would show additional information to tell you what is equated with what.

A port-socket pair is nothing more than a two-member fusion set. Port-socket pairs are what link superpages and subpages so as to implement substitution transitions. Therefore substitution transitions are in practice just a specialized application of fusion sets.

CPN Hierarchy: Substitution Transitions

Jumping Directly to a Superpage

You can jump directly from a subpage to its superpage, bypassing the hierarchy page, by double-clicking on any port on the subpage.

Moving Existing Net Structure to a Subpage

The simplest way to create a substitution transition is to move part of an existing net to a subpage, leaving a superpage behind. The steps in this process are:

Select the net component(s) that are to move to the subpage.

Choose **Move to Subpage** from the **CPN** menu.

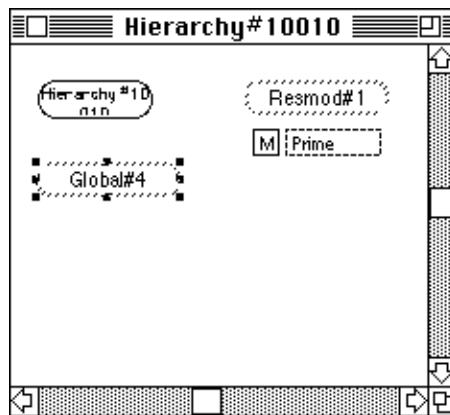
Specify where on the superpage you want the substitution transition to appear.

Name the substitution transition (if desired).

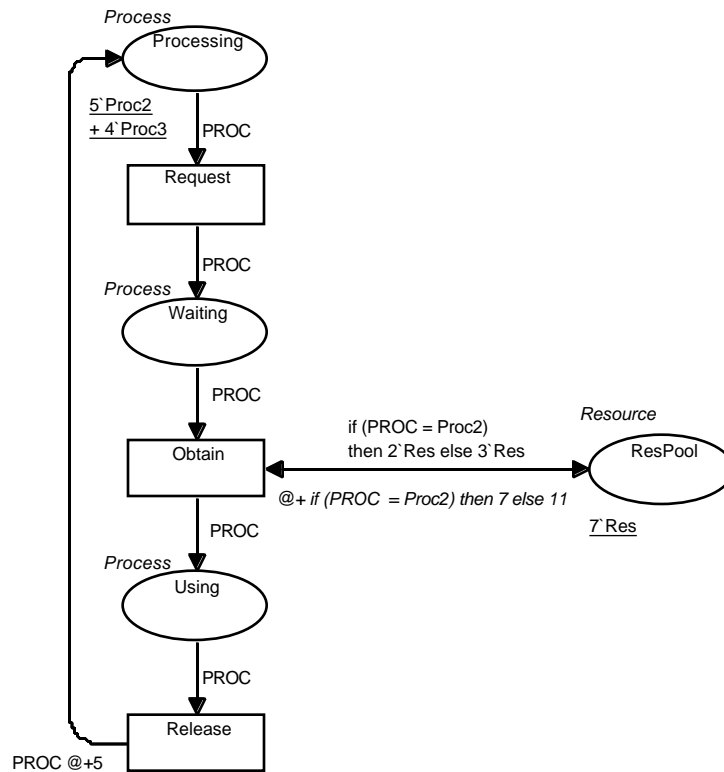
The following example illustrates this procedure using the model ResourceModel.

ResourceModel Component Pages

Hierarchy Page



Resmod#1 Page



Designate the Net Components to Move to the Subpage

To move a single transition, click the mouse on it so that it becomes the selected object.

To move a larger piece of net structure, form a group that contains all of the constituent nodes.

In the example we are using, Resnet, we select the following five nodes that comprise a subnet: Request, Waiting, Obtain, Using, and Release.

Initiate Subpage Creation

Choose **Move to Subpage** from the **CPN** menu.

Specifying the Substitution Transition's Location

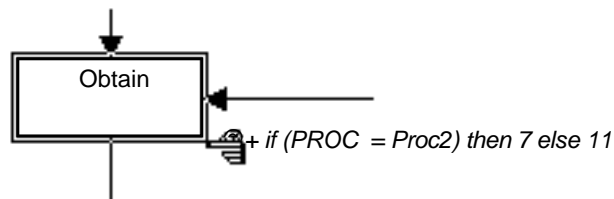
Design/CPN has no way to know where you want the substitution transition to appear. It therefore enters a specialized editor mode: *substitution transition creation mode*. This mode is similar to ordinary transition creation mode, except that the result is a substitution

CPN Hierarchy: Substitution Transitions

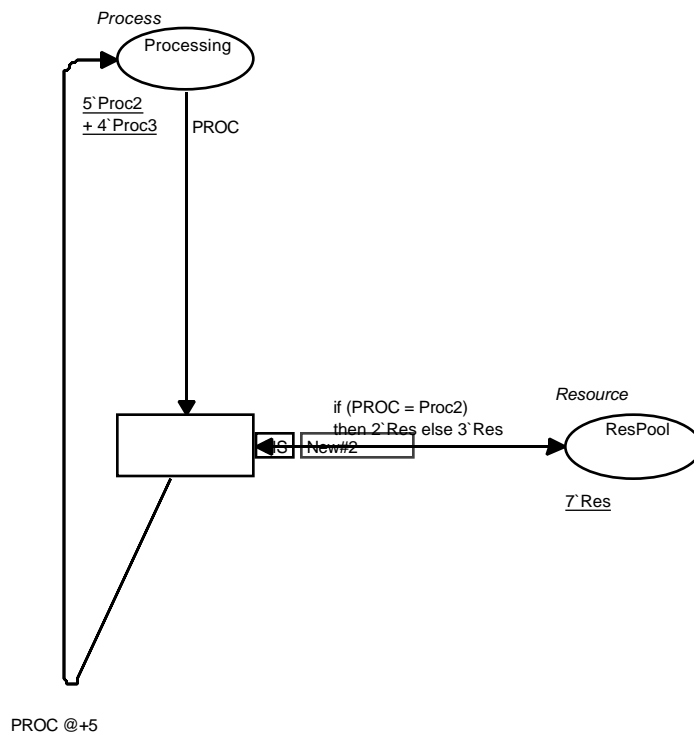
transition. As with ordinary transition creation, moving the mouse during the creation process reshapes the transition at its current location, while depressing SHIFT and moving the mouse moves the transition while preserving its shape.

Use the mouse with the SHIFT key to create a transition that just surrounds the existing transition Obtain.

Just before you release the mouse button, Obtain should look like this:



When you have finished creating the transition, Design/CPN moves all the nodes you selected, and any associated regions, to a new page that it creates for this purpose. That page is a subpage, and the page you have been working on is a superpage. The superpage now looks like this:



The transition is now a substitution transition. To indicate this, a region called the *hierarchy key region* has been created. This consists of a box, **HS**. The box is currently partly obscured by the arc

to `ResPool`. The next section shows you how to move this region to a better location.

Next to the hierarchy key region is another region, called the *hierarchy region*, containing the text `New#2`. This region indicates that the net structure that the substitution transition stands for is on the page `New#2`. It is a popup region, so you can make it appear and disappear by double-clicking on the key region.

Name the Substitution Transition (If Desired)

A substitution transition does not have to have a name. If it does not, three dots will appear on the hierarchy page by the node for the transition's subpage, where the transition's name would otherwise be displayed. But it is generally a good idea to provide a name, particularly if a model is complex.

Naming a substitution transition is exactly like naming any other transition: select the transition, then use the **CPN Region** command from the **CPN** menu.

Improving a Superpage's Appearance

It is rare for a superpage to have an ideal appearance from the first. Usually some tuning is appropriate, as in the current case. In the example, two changes would obviously be useful:

1. Reroute the arc that runs from the substitution transition to `Processing`.
2. Move the hierarchy key region and hierarchy region so that the arc does not obscure them.

Rerouting an Arc

Position the mouse pointer over the adjustment point just above the arc inscription.

Depress the mouse button.

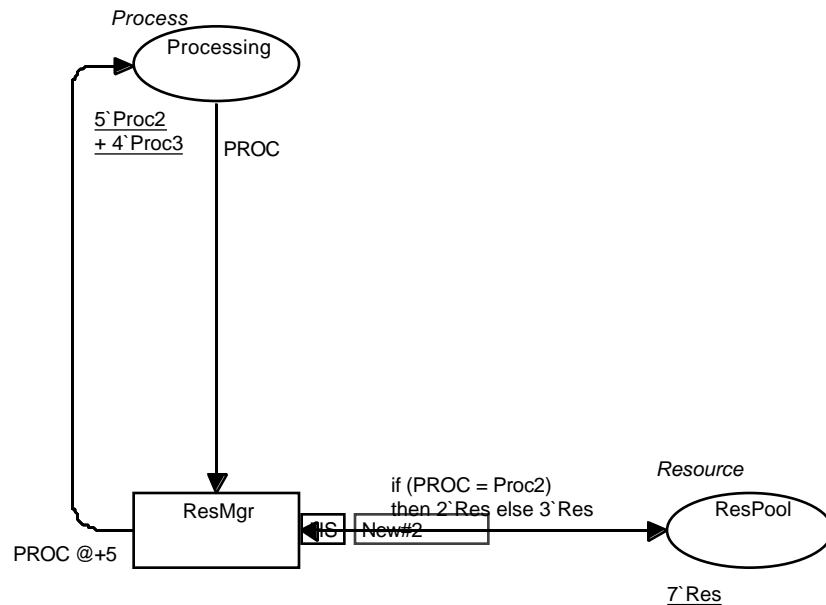
Move the adjustment point so that the arc follows a right angle (no "jaggies" in the adjacent segments).

Release the mouse button.

Use the mouse to move the arc inscription until it is just below the bottom segment of the arc.

The superpage now looks like this:

CPN Hierarchy: Substitution Transitions



Moving Regions

Moving regions can present a problem: they may be so small that it is difficult to grab onto them with the mouse without getting the arc instead. In the example, the hierarchy key region is particularly hard to get hold of.

To deal with situations like this, Design/CPN provides commands in the **Makeup** menu that allow you to select and move objects without using the mouse.

Select the substitution transition.

Choose **Child Object** from the **Makeup** menu.

The transition has only one child object, the hierarchy key region, so the region becomes selected. (If there were more than one child object, and **Child Object** did not select the needed object, you could use **Next Object** and **Previous Object** to select among the child objects.)

To move the region:

Choose **Drag** from the **Makeup** menu.

The editor is now in drag mode. The mouse pointer becomes the drag tool:



In drag mode, you can move objects without having to position the mouse pointer directly over them. If you depress the mouse button anywhere over the current window, then move the mouse, all selected objects will move along with the mouse.

Move the drag tool to the general vicinity of the obscured hierarchy key region.

Depress the mouse button.

Move the mouse around.

Note how the key region tracks the movement of the mouse. Since the hierarchy region is a region of the key region, it follows the movement also.

Position the hierarchy key region in the desired location.
Example: the lower right corner of the substitution transition.

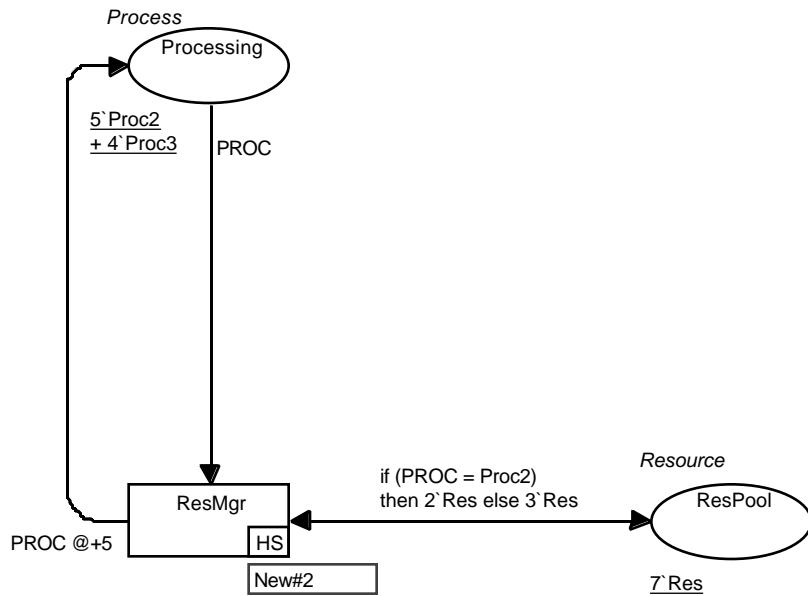
To exit drag mode:

Press ESC.

The editor returns to graphics mode. Additional adjustments can now be done with the mouse. Example: positioning the hierarchy region so that it is directly below the key region.

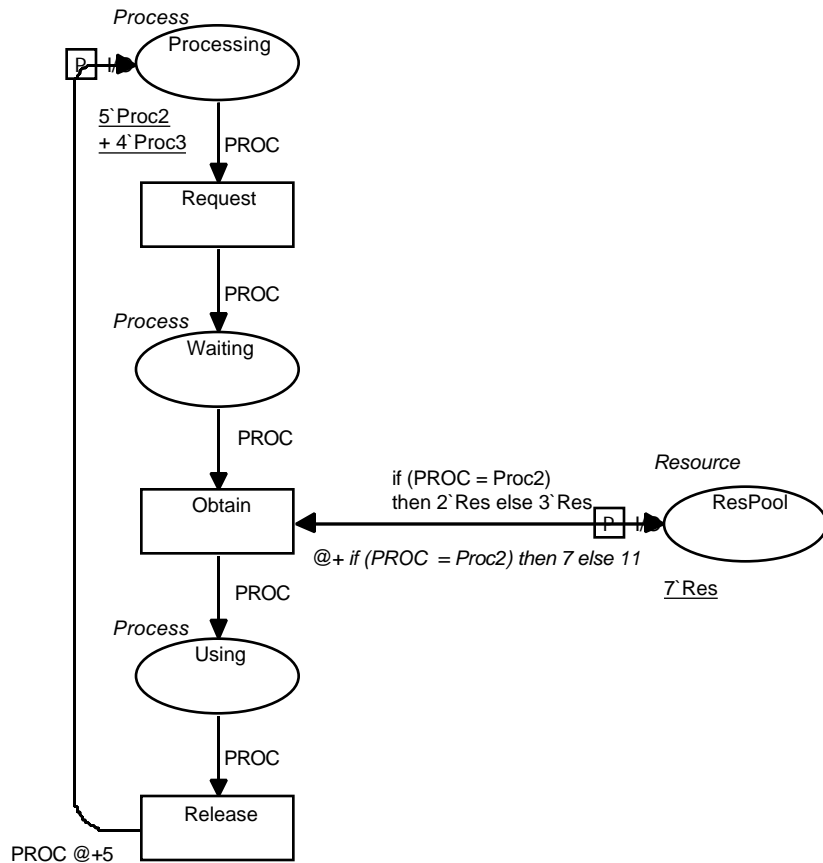
If the described adjustments were carried out, the superpage would look about like this:

CPN Hierarchy: Substitution Transitions



Improving a Subpage's Appearance

A subpage created by moving part of an existing page usually needs little appearance tuning, but sometimes improvements can be made.

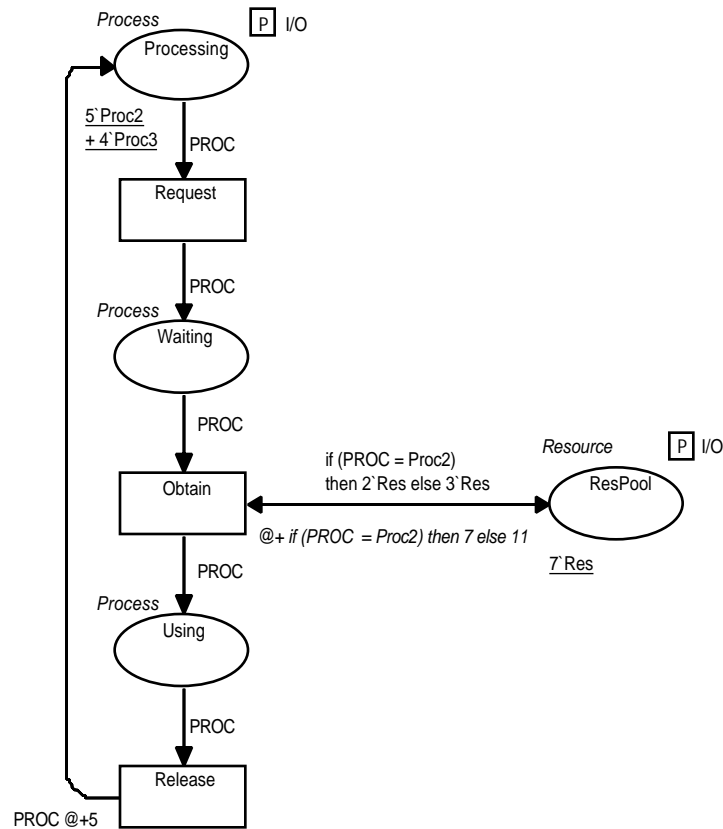


Design/CPN User's Guide

This page looks pretty good except for the placement of the port regions. To move them to better locations:

Use **Child Object**, **Next Object**, **Previous Object**, and **Drag**, from the **CPN** menu, to position the port regions at the upper right edge of their places.

The subpage would then look like this:



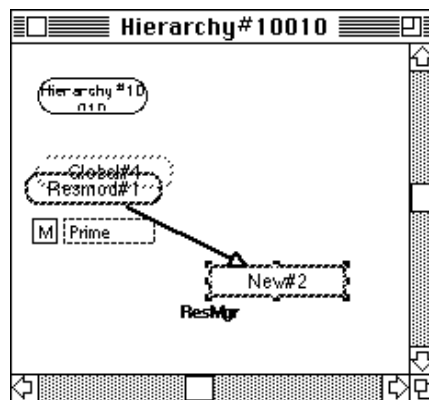
Improving a Hierarchy Page's Appearance

After you create a new page, it is a good idea to check the hierarchy page to be sure that it still has a good appearance.

Choose **Open Page** from the **Page** menu.

The hierarchy page appears. For example:

CPN Hierarchy: Substitution Transitions

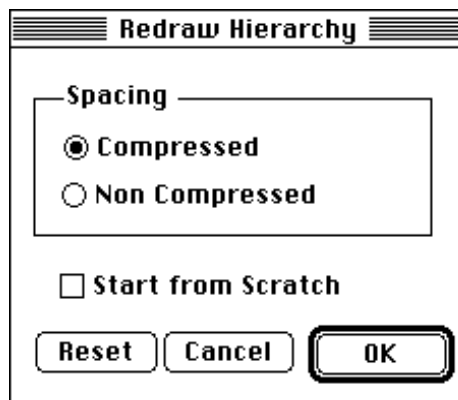


When you create a substitution transition, Design/CPN automatically updates the hierarchy page. It does not attempt to do this intelligently, because such an attempt could destroy a hand-crafted hierarchy page organization that could not have been produced algorithmically.

Obviously this is not an acceptable layout. But we don't have to improve it by hand. Design/CPN can do the job automatically.

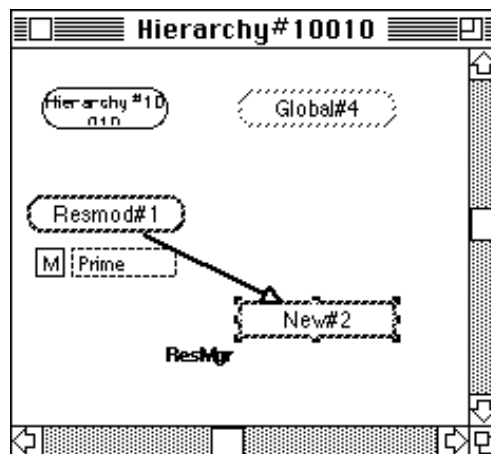
Choose **Redraw Hierarchy** from the **Page** menu.

The **Redraw Hierarchy** dialog appears:



Click **OK**.

The dialog disappears. Design/CPN redraws the hierarchy page:



Developing on a Subpage

In the preceding example, we created a net first, then moved part of it to a subpage, leaving behind a substitution transition to represent it.

It is also possible to create a substitution transition first, open the resulting subpage, and do development work on the subpage. This method is generally preferable to creating a net and then extracting from it, because it never requires everything to be on the same page at the same time. After all, the whole purpose of hierarchy is to avoid having to put everything on the same page.

Development Procedure

Select the transition that is to become the substitution transition.

Choose **Move to Subpage** from the **CPN** menu.

Draw the a substitution transition so that it just surrounds the selected transition.

Make the new subpage current.

Develop the net on the new subpage.

The following example illustrates this procedure using the model ResmodSubtrans described at the beginning of this chapter.

CPN Hierarchy: Substitution Transitions

Creating the Substitution Transition

To create the substitution transition:

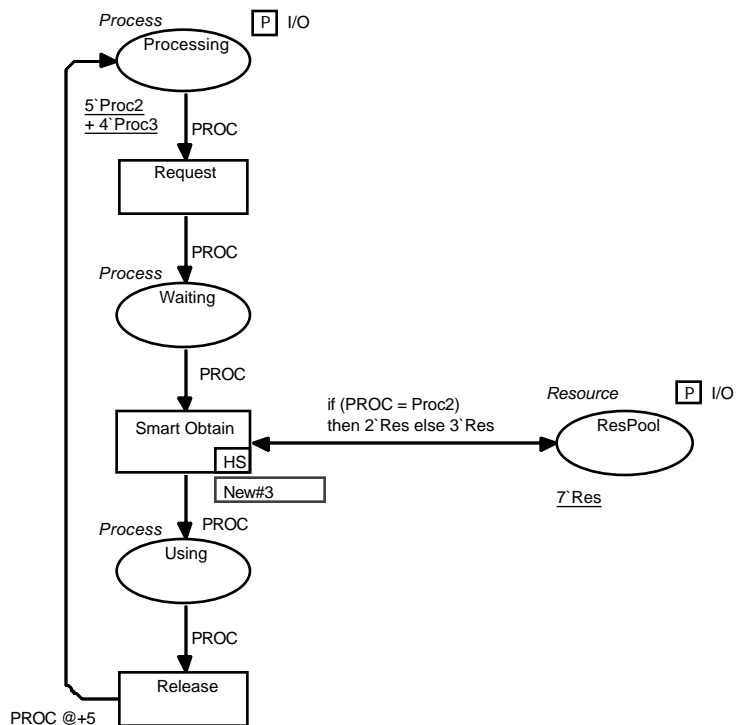
Select the transition *Obtain*. Be sure nothing else is selected.

Choose **Move to Subpage** from the **CPN** menu.

Draw the substitution transition so that it just surrounds *Obtain*, just as you did before.

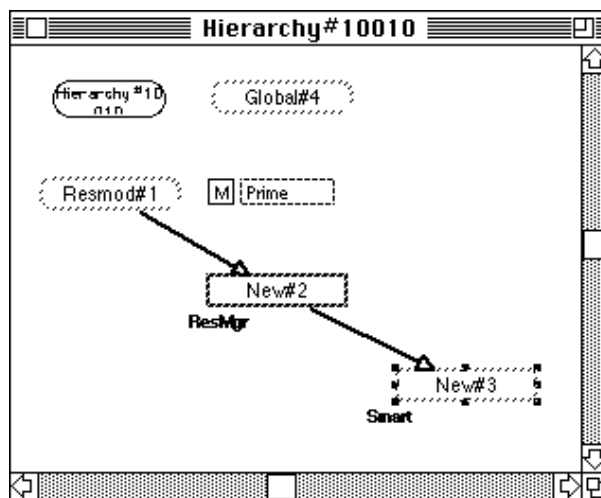
Name the substitution transition “Smart Obtain”.

New#2 should now look about like this:



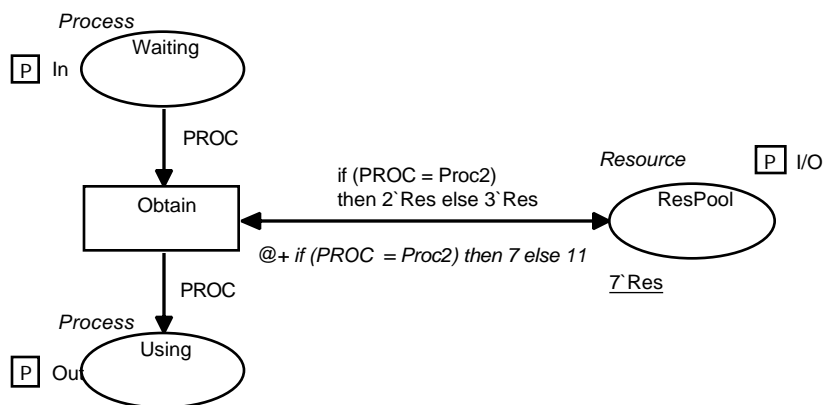
The Modified Hierarchy Page

After the operations described, the hierarchy page looks like this:



The page shows that New#2 is a subpage of Resmod#1 and contains the details of the substitution transition ResMgr, and that New#3 is a subpage of New#2 and contains the details of the substitution transition Smart Obtain, which has been truncated to Smart. Design/CPN does this to avoid cluttering up the hierarchy page with long names.

The New Subpage



There is only one general difference between this subpage and New#2, the subpage (which is now also a superpage) that you created previously. Both ports on New#2 were listed as “I/O”. Here Waiting is listed as “In”, and Using is listed as “Out”. These designations refer to the status of the ports in the context of the subpage: Waiting is an “In” port because it is an input place of Obtain, and Using is an “Out” port because it is an output place of Obtain.

CPN Hierarchy: Substitution Transitions

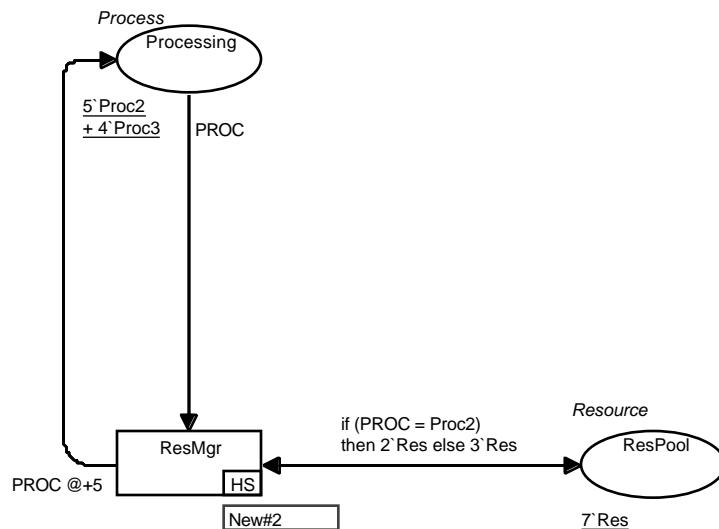
ResPool now exists on three different pages. It is a socket on Resmod#1, both a port and a socket on New#2, and a port here on New#3. All this really means is that ResPool is now part of a three-member fusion group that spans three pages. It is still the same place on all three pages: nothing new has been added.

Using a Subpage More Than Once

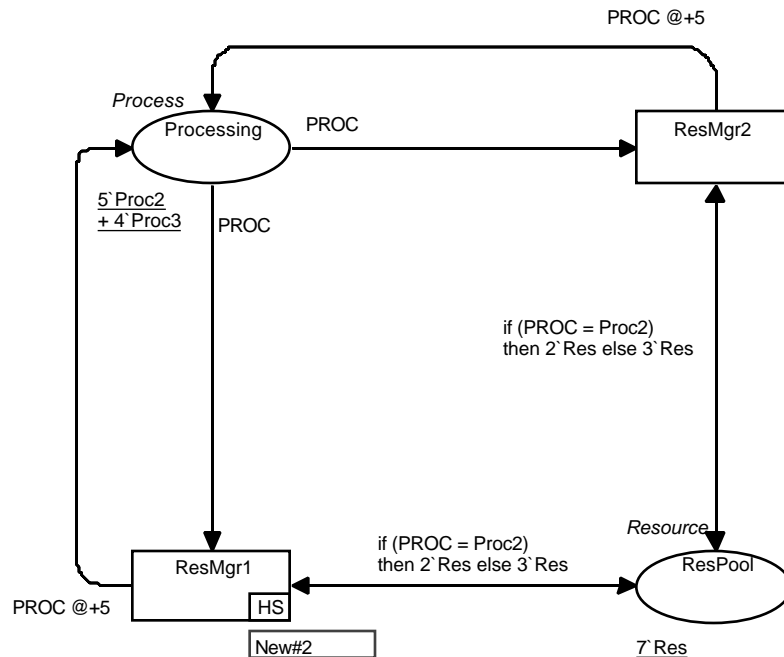
The capabilities described so far in this chapter would be more than enough to make substitution transitions a useful technique. But these capabilities are only the beginning. The real power of substitution transitions lies in the fact that a subpage need not be the value of only one substitution transition: it can be used repeatedly, as the value of any number of substitution transitions on any number of superpages.

Example

In our example, the page Resmod#1 currently looks like:



Suppose the page were modified as follows:



There is now a second `ResMgr` transition, `ResMgr2`. It is an ordinary transition, while `ResMgr1` (previously `ResMgr`) is a substitution transition. The next step is to convert `ResMgr2` to a substitution transition, then associate it with the existing subpage New#2.

Select `ResMgr2`.

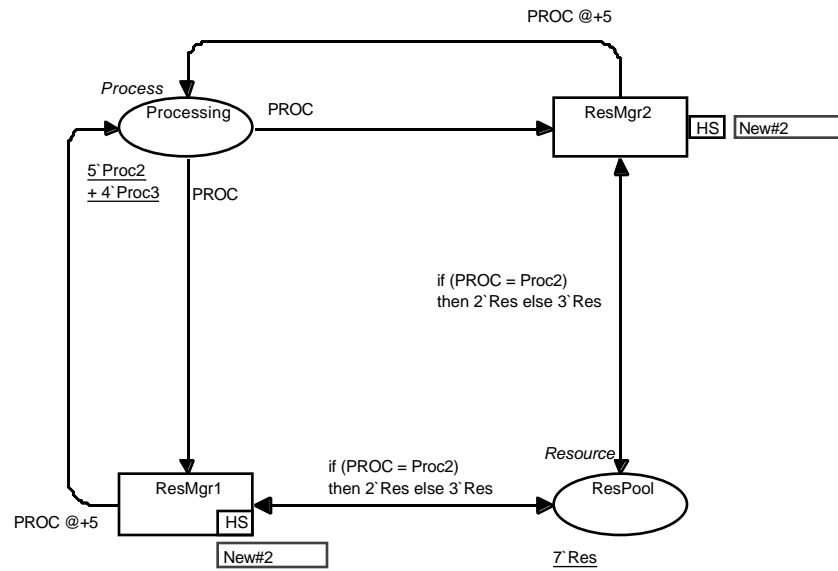
Choose **Substitution Transition** from the **CPN Menu**.

In order to convert `ResMgr2` into a substitution transition, Design/CPN needs to know what subpage to associate with the transition. It therefore displays the hierarchy page, allowing you to use the mouse to indicate the subpage.

Click the mouse on the page node for New#2.

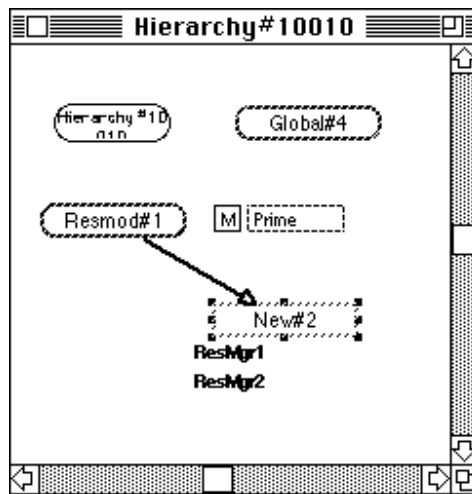
Resmod#1 reappears. `ResMgr2` is now a substitution transition whose value is the subpage New#2:

CPN Hierarchy: Substitution Transitions

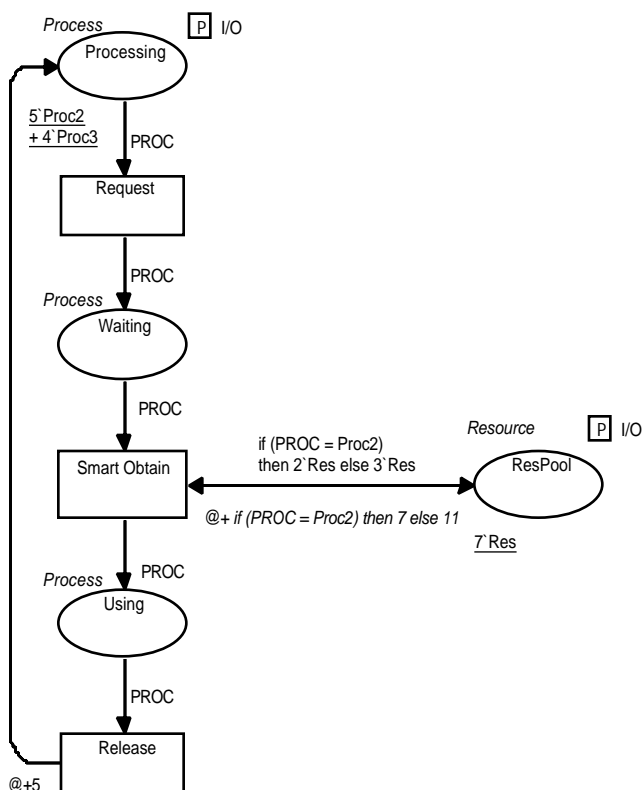


The two port places `Processing` and `ResPool` on the subpage have the same names as two socket places connected to `ResMgr2` on the superpage. Design/CPN has used this information to assign the ports to the sockets automatically. If any port had a name that did not match that of a socket, you would need to assign that port to a socket manually, as described later in this chapter.

The hierarchy page changes to reflect the second use of page New#2:



New#2 is now the value of two substitution transitions, so both names appear below its page node in substitution tag regions:



New#2's appearance hasn't changed at all. It still looks the same as it did when it was the value of only one substitution transition. It would remain the same no matter how many substitution transitions it was the value of.

Relationship of Pages in a Hierarchy

The various pages in a hierarchy created with substitution transitions are connected only by their ports and sockets. They have no other functional relationship to each other. Therefore they can be modified completely independently of each other: changes made on a subpage have no effect whatever on the net structure on a superpage, and changes made on a superpage have no effect on a subpage.

Subpages and superpages interact directly only during simulation. Their interaction consists only of the consequences of the fact that a port and a socket are functionally the same place, so that any change that a subpage makes to a port's marking intrinsically changes the superpage's socket's marking, and vice versa.

Thus a subpage functions as a module in much the same way that a subroutine does. There is a clearly defined interface, implemented in this case by the ports and sockets, and the only connection be-

CPN Hierarchy: Substitution Transitions

tween a module and anything that uses it is the interface. However a subpage is not exactly like a subroutine: it is in some ways more like a macro.

Subpages, Subroutines, and Macros

The similarity between a subpage and a subroutine should now be obvious. Just as you can create a subroutine independently of other code, have only one copy of it, and call it from many points in a program, so you can create a submodel independently of other net structures, have only one copy of it, and use it in many locations in a model.

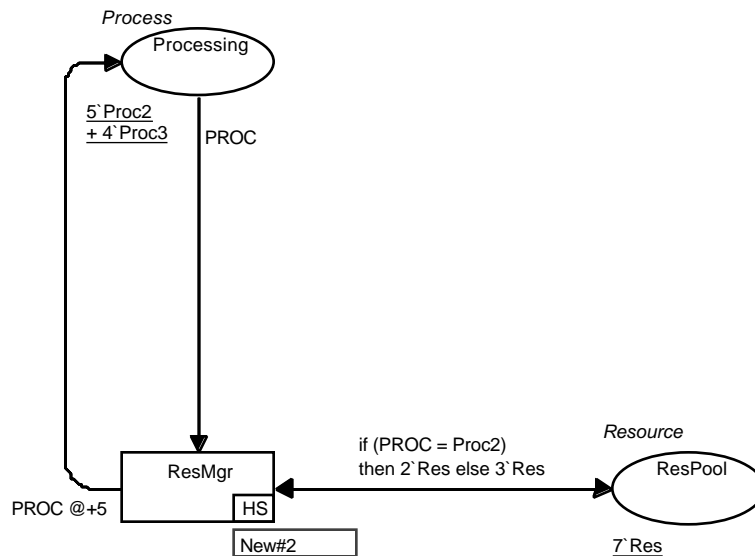
However, subpages differ from subroutines in one important way. Although they are not physically copied into superpages, the effect of such copying being provided by using instances, the functional effect is the same as if they actually had been copied. Consequently substitution transitions are really more like macros than like subroutines: they are not called during execution and instantiated only when and if they are called, as subroutines are, but are replaced by their values during compilation, as macros are.

In practice, the macro-like quality of substitution transitions has only one consequence: substitution transitions cannot be used recursively, either directly or indirectly, for such usage would lead to an infinite loop of substitution. Design/CPN does not permit you to specify recursive substitution.

Removing Hierarchical Constructs

Reversing Substitution Transition Creation

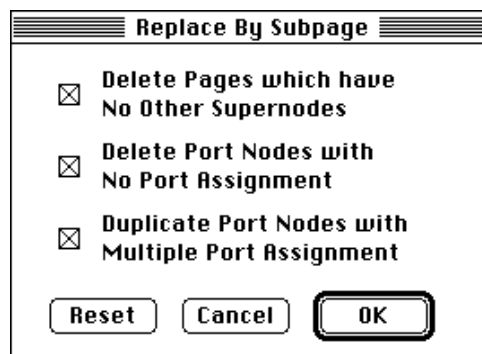
As a model evolves, it sometimes happens that a substitution transition that was once useful ceases to be so. When this occurs, you can reverse the creation process, copying the subpage into the superpage and eliminating the substitution transition.



Select the substitution transition to be replaced: in the example above, the transition `ResMgr`.

Choose **Replace by Subpage** from the **CPN** menu.

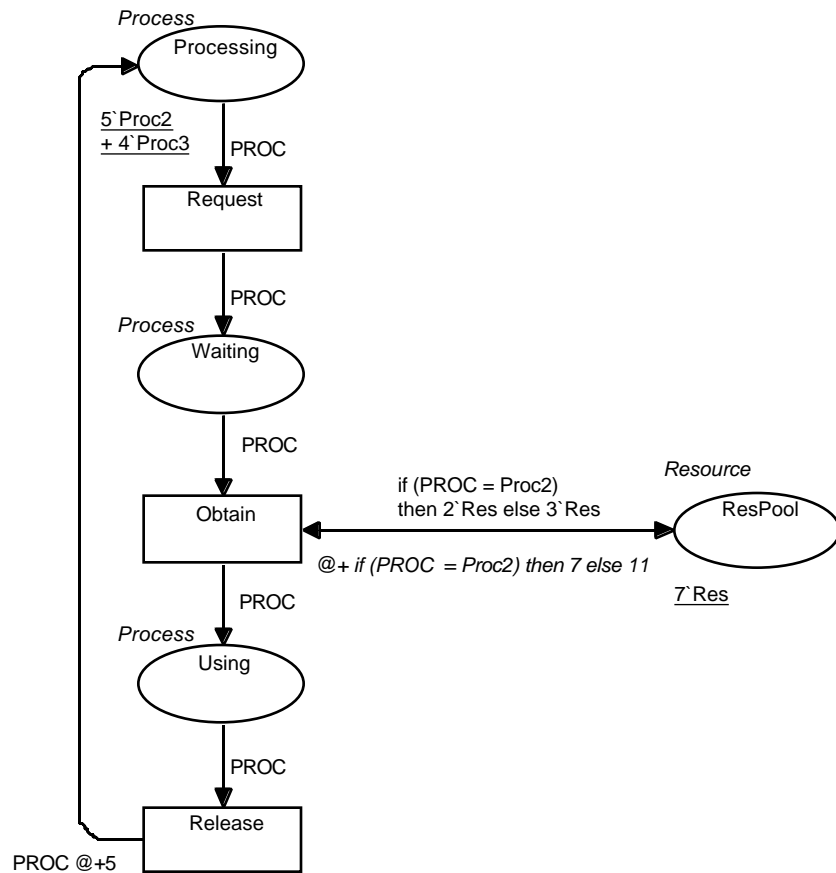
The **Replace By Subpage** dialog appears:



Click **OK**.

The dialog disappears. Design/CPN copies the subpage back into the superpage. The result is:

CPN Hierarchy: Substitution Transitions



Status of the Model

The model is now exactly as it was before the substitution transition was created. It is not structurally or functionally different in any way for having been split into two hierarchical pages and then re-assembled. The two operations are exactly inverse.

Deleting a Subpage

As a model evolves, it sometimes happens that a subpage is no longer needed for some reason. The need then is not to copy it to a superpage, but to get rid of it entirely.

Deleting such a subpage is no different from deleting any other page. Design/CPN automatically makes all adjustments necessary to reflect the fact that the subpage no longer exists.

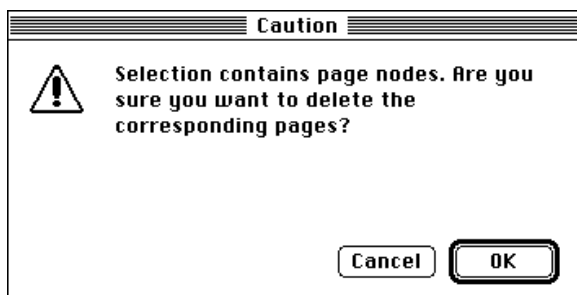
For example, suppose you decided that implementing a more intelligent algorithm for **Obtain** is not necessary after all, so that the subpage New#3 on which you might have implemented such an algorithm is no longer necessary. To delete the subpage:

Open the hierarchy page.

Select the page node for New#3.

Press DELETE. (You can't **Cut** a whole page.)

A confirmation dialog appears:



Click **OK**.

The page node for New#3, and the connector designating it as a subpage of New#2, both disappear. New#3 is gone.

Open New#2.

`Smart Obtain` is now just an ordinary transition, and the regions designating it as a substitution transition have disappeared.

Status of the Model

Is the net functionally the same as before `Smart Obtain` was created? It is not! `Smart Obtain` does not have a time region (See Chapter 12, "Simulated Time," for a discussion of time regions), but `Obtain` did: the region was moved to a subpage along with `Obtain`, and disappeared from the model when the subpage was deleted. Creating and then deleting a subpage, even if only one transition was moved to the subpage, can permanently change a superpage.

Deleting a Reference to a Subpage

Sometimes there is a reason to break the association between a substitution transition and its subpage. When such an association is broken, the substitution transition becomes an ordinary transition. The subpage remains associated with any other substitution transitions that reference it, and continues to be available for use by additional substitution transitions.

CPN Hierarchy: Substitution Transitions

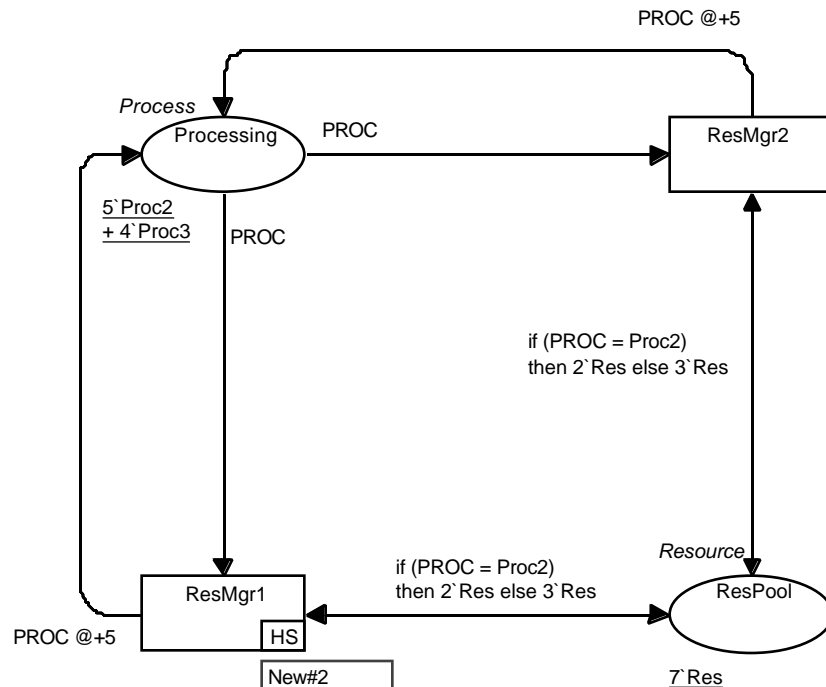
To delete a reference to a subpage, delete the hierarchy key region from the substitution transition that references it.

Open Resmod#1.

Select the hierarchy key region **HS** next to ResMgr2.

Press DELETE.

The hierarchy key region and associated hierarchy region disappear:



ResMgr2 is now an ordinary transition. Its status is the same as if it had never been a substitution transition.

Manually Assigning Ports to Sockets

It was easy to convert Resmgr2 into a substitution transition in the example above, because there was no need to indicate explicitly which port should be associated with which socket: Design/CPN made the assignment automatically by matching port names with socket names.

Equating ports and sockets by matching their names can be a great convenience, but it does not always provide what is needed. Requiring port and socket names to match would be like requiring a subroutine always to be called with the same variable names in the argument list. The disadvantages are obvious. To avoid such

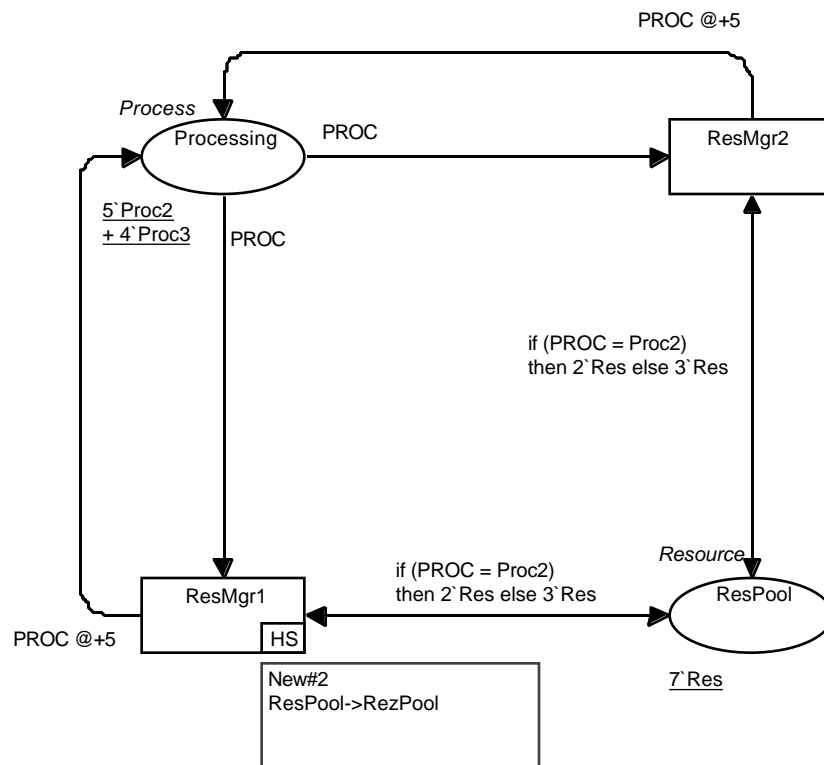
Design/CPN User's Guide

problems, Design/CPN allows you to manually specify what port will be matched with what socket when a substitution transition is created.

In the `Resmod` example, page `New#2` is currently the value of the substitution transition `ResMgr1`, and `ResMgr2` is an ordinary transition, having been restored to that status in the previous section.

Suppose that the port named `ResPool` on page `New#2` in the example were renamed `RezPool`. This makes no functional difference. Design/CPN matched the names when `ResMgr1` became a substitution transition, but the association of port with socket that was created then has no further connection with the name of either place. Port and socket names may therefore be changed without reference to one-another.

When names don't match, we need some other way to determine what port is associated with what socket. The necessary information is kept in the hierarchy region of the substitution transition. If you expanded the hierarchy region of `ResMgr1` to display all of its contents, the page would look like this:

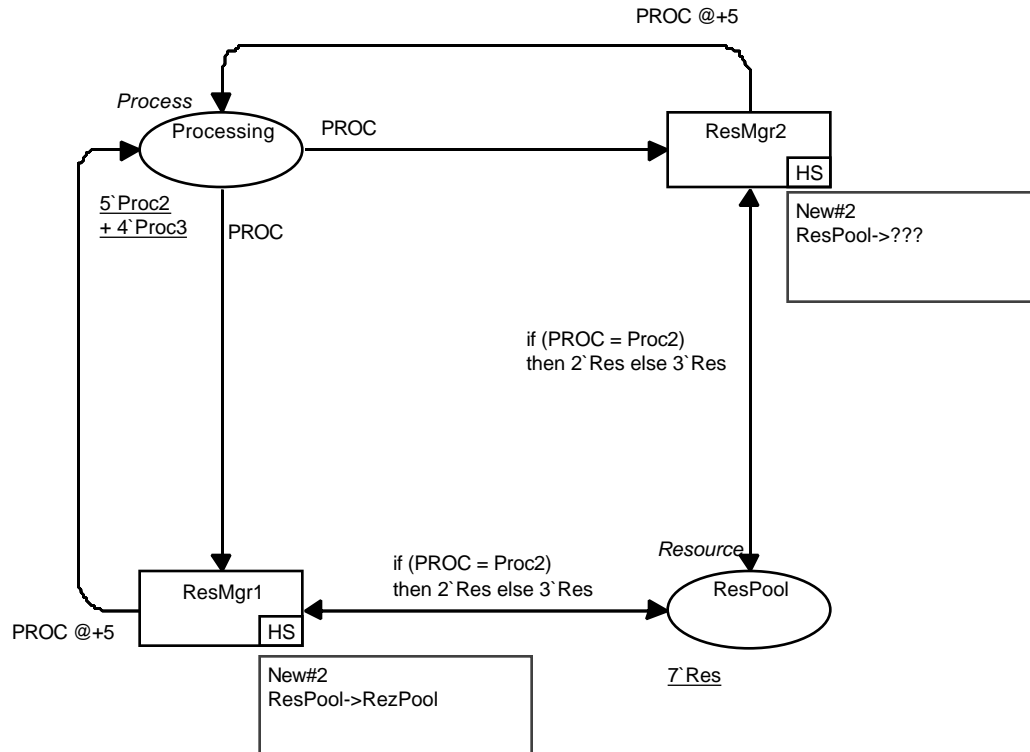


The second line in the region indicates that the socket `ResPool` in `Resmod#1` is associated with a port named `RezPool`. The first line indicates that `RezPool` is on `New#2`. There is no line for `Processing`, because it has the same name as its port (at least in the context of `ResMgr1`). The region could contain a line `"Processing->Processing"`, but for economy such lines are just

CPN Hierarchy: Substitution Transitions

omitted. If both `Processing` and `ResPool` matched their ports, the region would contain only the name of the relevant subpage.

If you made `ResMgr2` a substitution transition whose value is the page `New#2`, then expanded `ResMgr2`'s hierarchy region to display its full contents, the page would look like this:



(The region has been repositioned to make the figure fit on the page.)

The port and socket named `Processing` have been matched automatically, so no reference to them appears in the hierarchy region. The region indicates that `ResMgr2` has one unassigned socket: `ResPool`.

To assign the port `ResPool` to the socket `ResPool`:

Select `ResPool`.

Choose **Port Assignment** from the **CPN** menu.

The status bar displays Select one of the compound Nodes.

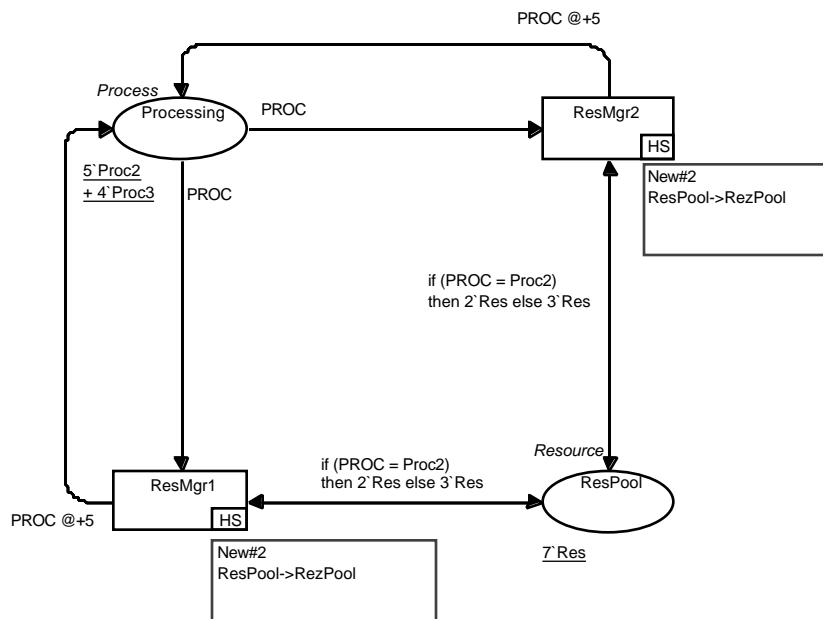
Click on `ResMgr2`.

Design/CPN displays the subpage for `ResMgr2`, so that you can indicate which port you want.

Design/CPN User's Guide

Click on `RezPool`.

Design/CPN returns you to `Resmod#1`. The hierarchy region for `ResMgr2` now shows that the socket `ResPool` is associated with the port `RezPool`:



You can also use this technique to reassign ports and sockets. Just make the assignment you want, and the existing assignment will be replaced by the new one.

Chapter 12

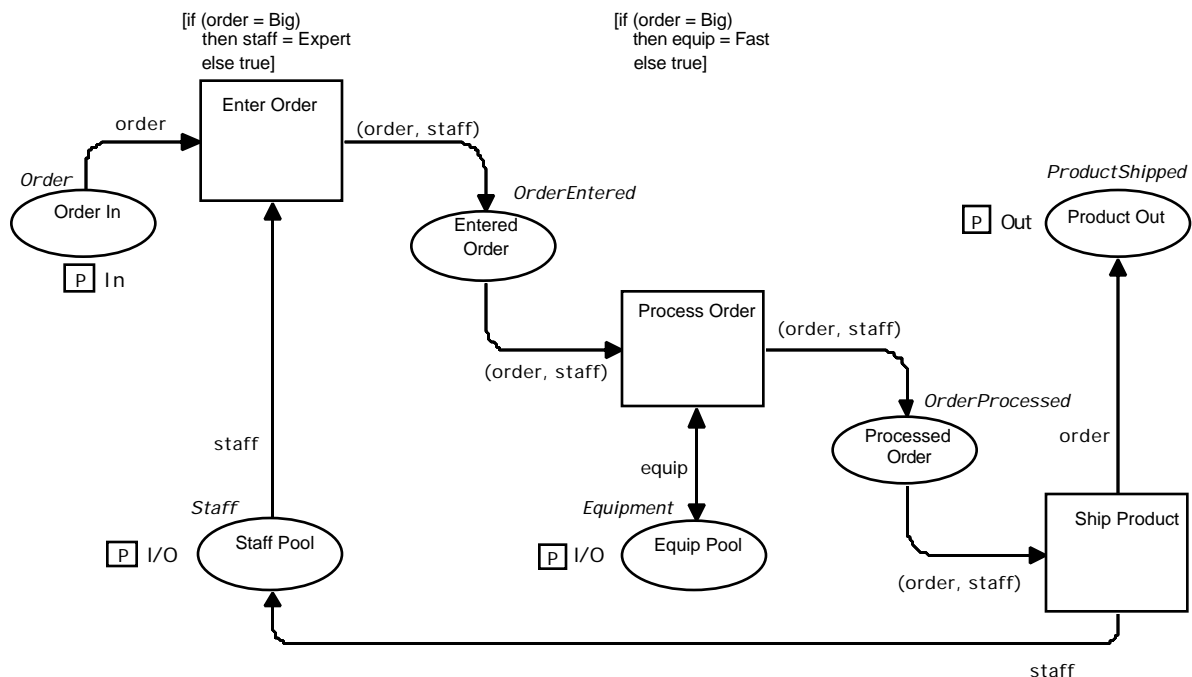
Simulated Time

The model we will use in this chapter, FmodTimed, consists of a superpage containing an overview of a sales order system, a subpage containing a decomposition of that overview, and (as always) a global declaration node.

FmodTimed, as shown below, gives a fairly realistic, though quite high-level, view of a simple Sales Order System, except for one thing: it contains no representation of time.

This chapter shows you how to represent and manage time in a CP net, and how to use this understanding to create a version of FmodTimed that takes account of the fact that no activity happens instantaneously.

We don't need to be concerned with the hierarchy page or the superpage in this chapter. The subpage and global declaration node look like this:



```
color Order = with Big | Small;           (* Orders for products *)
color ProductShipped = Order;             (* Orders shipped *)
color Staff = with Expert | Novice;       (* Staff members *)
color Equipment = with Fast | Slow;       (* Pieces of equipment *)

color OrderEntered = product Order * Staff; (* Orders entered but unprocessed *)
color OrderProcessed = OrderEntered;      (* Orders processed but unshipped *)

var order: Order;                         (* An order *)
var staff: Staff;                         (* A staff member *)
var equip: Equipment;                     (* A piece of equipment *)
```

The Nature of Simulated Time

In order to understand how time is represented in a CP net, we must first distinguish clearly between real time and simulated time. *Real time* is just that: time in the real physical world in which the simulator executes a model and we can watch what happens. *Simulated time* is just a symbolic representation of time that we may optionally build into a model. It has no reality outside the symbolic world of the model that contains it.

Real time and simulated time have no intrinsic relationship whatsoever. We may or may not build a symbolic representation of time into a model. If we do not, the sequence of states that an executing model passes through has no temporal interpretation: it is just a sequence of states.

Representing Time in a CP Net

In order to take time into account, we need a way to represent and manipulate time within a model. A surprisingly simple methodology provides everything we need in order to represent time in a CP net:

- A token may have an associated number, called a *time stamp*. Such a token is called a *timed token*, and its colorset is a *timed colorset*.
- The simulator contains a counter called the *clock*. The clock is just a number (integer or real) whose current value is the current time.
- A timed token is not available for any purpose whatever unless the clock time is greater than or equal to the token's time stamp.

- When there are no enabled transitions, but there would be if the clock had a greater value, the simulator increments the clock by the minimum amount necessary to enable at least one transition.

That's all we need to create a dimension of simulated time that has exactly the properties that we need. Let's look more exactly at how these rules work to provide simulated time.

How Simulated Time Works

Simulated time has nothing to do with the external time during which the simulator steps through net execution and observers possibly watch the execution process, or with the numbered sequence of steps by which the simulator executes a net. Simulated time is just an incrementable number that is globally available within an executing model. The value of this number can be thought of as the time indicated by a simulated clock. When the number is incremented, the clock moves forward to a later time.

The units of simulated time do not inherently represent any particular absolute time unit. We may interpret simulated time units as microseconds or millennia, depending on what we are modeling, but syntactically time is just a number. For brevity, simulated time is sometimes referred to as *model time*.

Simulated Time and Transition Enablement

The state of a net changes only when enabled transitions fire. In order for simulated time to affect on net execution, it must therefore affect transition enablement and firing.

This effect is produced by the rule that a timed token is unavailable for any purpose unless the clock is greater than or equal to the token's time stamp. Such a token is ignored when transitions are checked for enablement: it might as well not be there at all.

In effect, a timed token does not exist until the clock reaches a certain time, given by the token's time stamp. When the clock reaches that time, the token suddenly springs into existence, becomes a factor in determining enablement, and can be subtracted by transition firing.

The Simulated Clock

Simulated time could tick forward continually, as real time does, but that would be a very inefficient way to do things: the clock would frequently waste real time counting off intervals of simulated time

during which the model remains unchanged. Such counting would accomplish nothing. It is more efficient in such a case to jump the simulated clock immediately to the next time when some change is possible, and proceed with executing the net.

Therefore the simulated clock does not move at a steady rate. Instead, it remains at its current value as long as there are any enabled transitions. When there are no enabled transitions left the clock jumps forward by the smallest amount necessary for at least one transition to become enabled. This implies that time will never move forward in a net that always has enabled transitions independently of time.

Simulated time passes when and only when the clock is incremented. Everything that happens while the clock remains at a particular setting is both simultaneous and instantaneous in simulated time.

This is convenient when a system contains an event that happens at a particular moment, but that is sufficiently complex that modeling it by firing a sequence of simple transitions, rather than one complex transitions, is most convenient. Leaving the clock unincremented during such a sequence lets us model the event as a manageable series of small changes, while preserving the instantaneity of the event's effect on the state of the model.

Other Uses for Simulated Time

Simulated time is just a mechanism that follows certain rules. Nothing requires this mechanism to be used only for the purpose of simulating the passage of time. It can be used in any way that is useful.

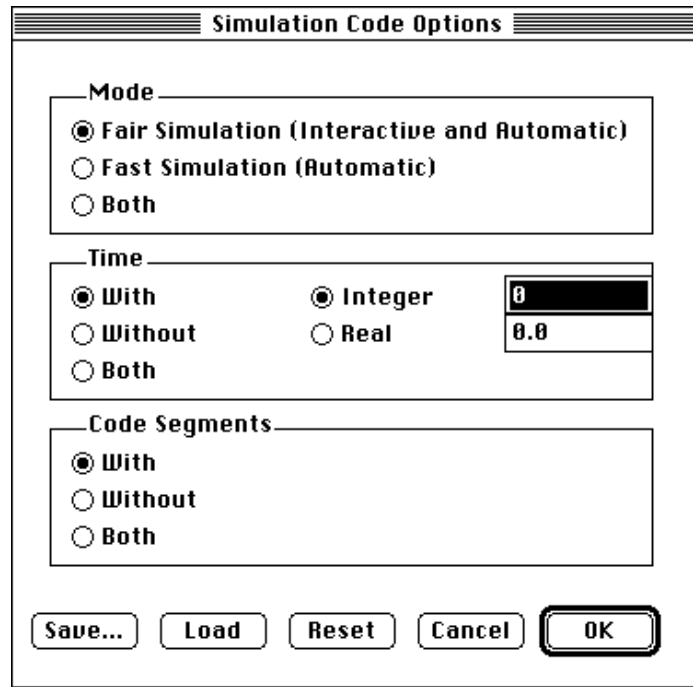
A net that will use the charting facility to display information about net execution will probably need to initialize one or more statistical variables at the start of execution. You can accomplish this by using timing, but it is much more convenient to use the `init` section of a chart's code segment, as described in Chapter 15.

Specifying Timed Simulation

Not all nets need to use timing. A net that does not has no need of the logic necessary to manage simulated time. Including such logic in the model would just add overhead.

A net that will use timing must specify the fact in the **Time** section of the **Simulation Code Options** dialog (**Set** menu). The specification may include an initial time value, which must be nonnega-

tive and defaults to 0. Before execution begins, the clock will be set to the time indicated in the dialog.



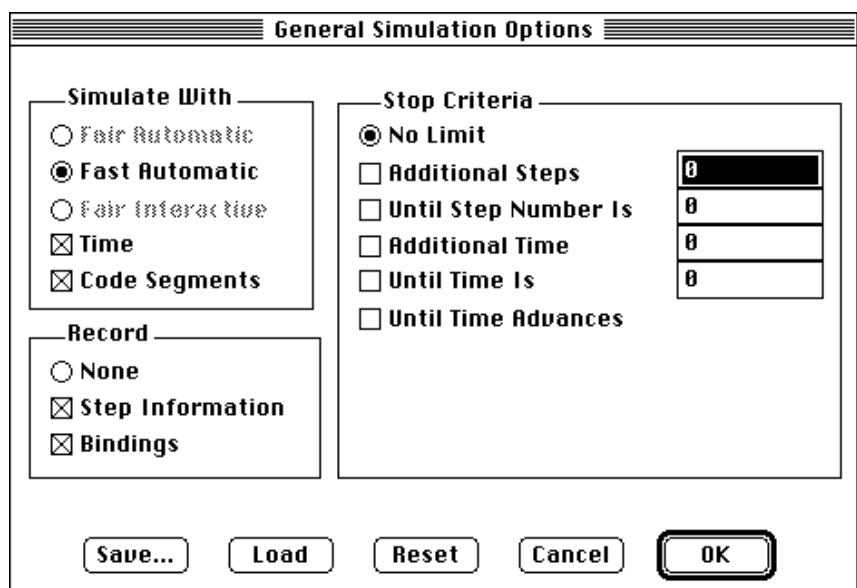
The **Simulation Code Options** dialog box contains three sections:

- Mode:**
 - ☒ Fair Simulation (Interactive and Automatic)
 - ☐ Fast Simulation (Automatic)
 - ☐ Both
- Time:**
 - ☒ With
 - ☒ Integer: 0
 - ☐ Real: 0.0
 - ☐ Without
 - ☐ Both
- Code Segments:**
 - ☒ With
 - ☐ Without
 - ☐ Both

Buttons at the bottom: Save..., Load, Reset, Cancel, OK.

The option **Both** causes code to be generated to support both timed and untimed simulation. This option is useful with a model that will sometimes be run with time, and sometimes without it.

The **Time** option in the **Simulate With** section of the **General Simulation Options** dialog (**Set** menu) specifies whether a particular simulation run is to be timed or not:



The **General Simulation Options** dialog box contains two main sections:

- Simulate With:**
 - ☐ Fair Automatic
 - ☒ Fast Automatic
 - ☐ Fair Interactive
 - ☒ Time
 - ☒ Code Segments
- Record:**
 - ☐ None
 - ☒ Step Information
 - ☒ Bindings
- Stop Criteria:**
 - ☒ No Limit
 - ☐ Additional Steps: 0
 - ☐ Until Step Number Is: 0
 - ☐ Additional Time: 0
 - ☐ Until Time Is: 0
 - ☐ Until Time Advances

Buttons at the bottom: Save..., Load, Reset, Cancel, OK.

Design/CPN User's Guide

The **Stop Criteria** section of the **General Simulation Options** dialog permits termination of the simulation to be controlled by time:

Additional Time

Simulation stops as soon as the number of time units specified in the value box have elapsed. This option is available only if **Time** is chosen in the **Simulate With** section.

Until Time is

Simulation stops as soon as the clock value equals or exceeds the time specified in the value box. This option is available only if **Time** is chosen in the **Simulate With** section.

Until Time Advances

Simulation stops as soon as the clock advances. This option is available only if **Time** is chosen in the **Simulate With** section.

Declaring a Timed Colorset

To declare a timed colorset, declare it as you ordinarily would, and append the keyword `timed` to the declaration.

When a colorset is timed, duplicate colorsets and composite colorsets that include it are timed also, and therefore do not need be explicitly declared `timed`.

For example, the colorsets `Order`, `Staff`, and `Equipment` in `FmodTimed` are specified as follows:

```
color Order = with Big | Small timed;      (* Orders for products *)
color ProductShipped = Order;              (* Orders shipped *)
color Staff = with Expert | Novice timed;   (* Staff members *)
color Equipment = with Fast | Slow timed;   (* Pieces of equipment *)

color OrderEntered = product Order * Staff; (* Orders entered but unprocessed *)
color OrderProcessed = OrderEntered;        (* Orders processed but unshipped *)

var order: Order;                          (* An order *)
var staff: Staff;                          (* A staff member *)
var equip: Equipment;                      (* A piece of equipment *)
```

`ProductShipped`, `OrderEntered`, and `OrderProcessed` inherit the `timed` attribute, so all colorsets are now timed.

Giving a Token a Time Stamp

Tokens get time stamps via expressions called *delay expressions*. A delay expression has the form:

@+ expression

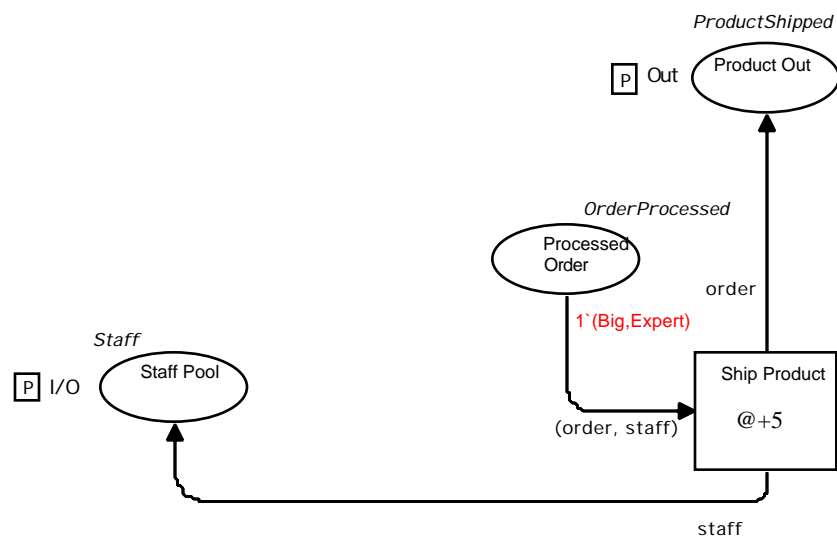
where “@+” appears literally, and *expression* is an arithmetic expression.

A delay expression defines a time equal to the current simulated time (symbolized by the @ sign) plus (+) the value of the *expression*. This value becomes the time stamp of any tokens created under the aegis of the delay expression.

There are two ways to use a delay expression to provide time stamps for tokens: by putting it in a time region, and by appending it to an output arc inscription.

Delay Expressions in Time Regions

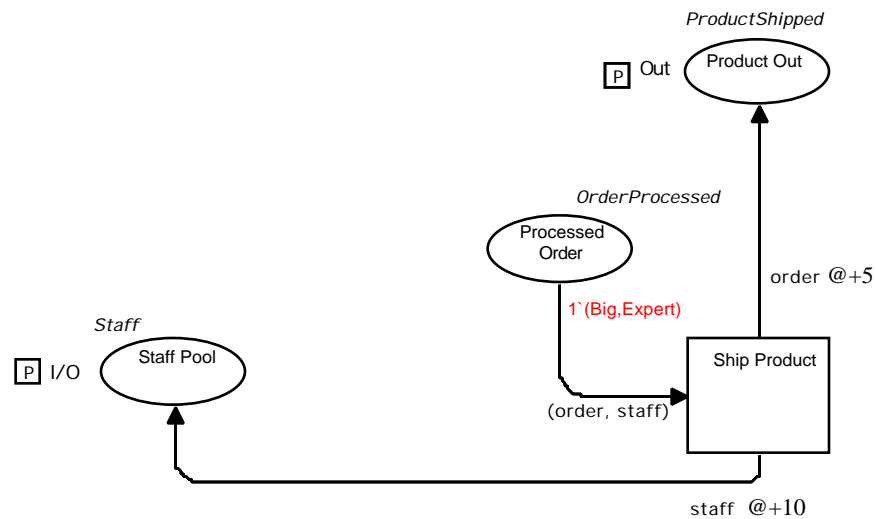
A *time region* is a region associated with a transition. The region contains a delay expression. Every output token of a timed colorset that is created by the transition will have a time stamp as designated by the delay expression. Output tokens of non-timed colorsets will be handled just as if there was no time region. For example:



All tokens that *Ship Product* adds to *Product Out* or *Staff Pool* will have a time stamp equal to the model time when the tokens were created plus 5. They will therefore be effectively nonexistent until the clock has been incremented by at least 5. This might represent a situation in which shipping a product takes 5 minutes, after which the staff member is again available to process orders.

Delay Expressions on Output Arc Inscriptions

When a time region is used, all output tokens of timed colorsets necessarily get the same time stamp. It is often convenient to give different time stamps to the tokens in different output places. This can be accomplished by appending delay expressions to individual output arc inscriptions. For example:



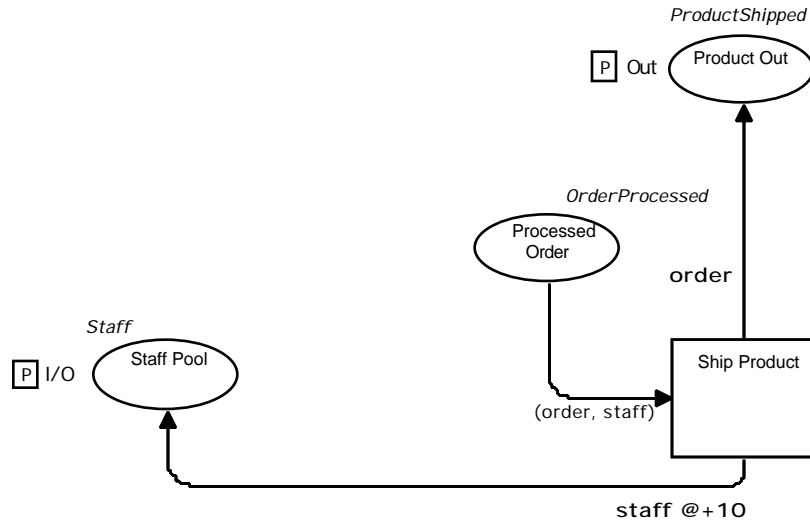
All tokens that `Ship Product` adds to `Product Out` will have a time stamp equal to the model time when the tokens were created plus 5, and all tokens that it adds to `Staff Pool` will have a time stamp equal to the model time plus 10.

This might represent a situation in which shipping a product takes 5 minutes, after which the staff member takes a 5 minute break. The product is thus available for further processing after 5 minutes, but the staff member is not available to process another order for 10 minutes. (Since no logic is shown that uses the tokens in `Product Out`, their time stamp would make no actual difference, but such logic could easily exist.)

Omitting a Time Stamp

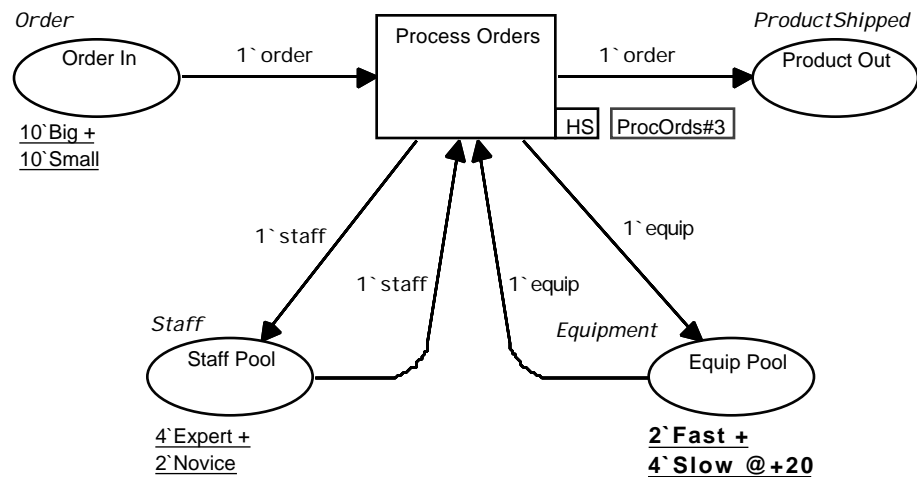
The fact that a colorset is timed does not mean that time is necessarily of interest in everything a token of that colorset does. Therefore a timed token does not have to have a time stamp; it only has the option to have one. If no time stamp is given to a timed token, the default time stamp is the current time. Since the clock never has a negative value, such a token is guaranteed to be immediately available, just as if it came from a non-timed colorset but was otherwise the same.

For example, since time delays on tokens in `Product Out` accomplish nothing because the tokens will never be used again, the preceding example would be more economically expressed as:



Time Stamps and Initial Markings

It is often useful to give a time stamp to the tokens in a place's initial marking. This is accomplished by appending a delay expression to the initial marking region. For example:



This could represent a situation in which equipment must warm up for 20 minutes at the beginning of a work day, and is not available until the warmup time has elapsed.

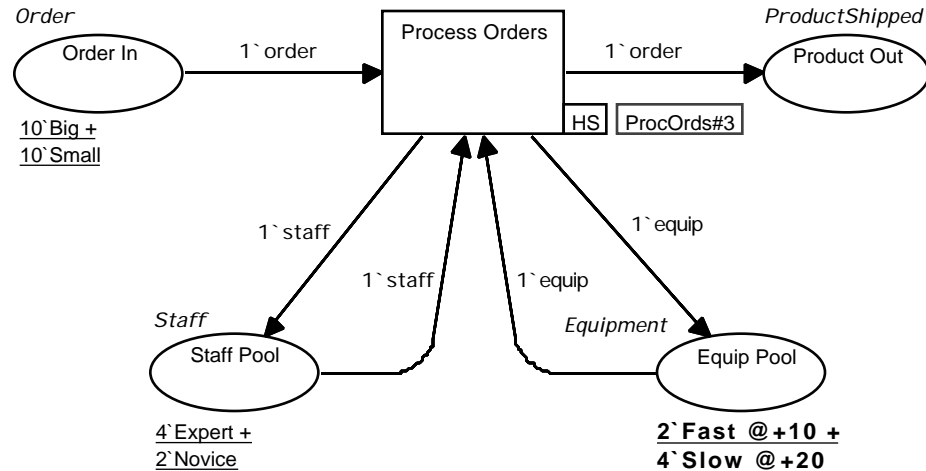
When no initial time stamp is specified for an initial token of a timed colorset, the default initial time stamp is the time specified in the

Design/CPN User's Guide

Time section of the **Simulation Code Options** dialog (**Set** menu).

Time Stamps and Multisets

When a multiset that defines more than one token is created, by an initial marking region or in any other way, all of the tokens in the multiset get the same time stamp. Thus:



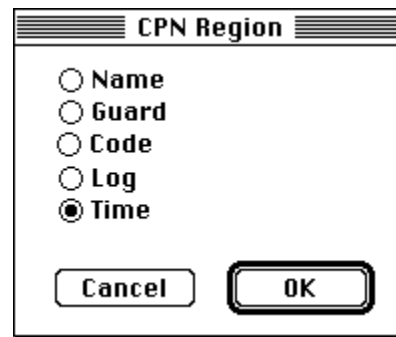
This is illegal, and would cause a syntax error.

Example of Time Region Use

In the following examples the FmodTimed model uses time regions instead of output arc delay expressions and time stamps in initial markings. Examples of these techniques would not demonstrate anything fundamental that examples of time regions do not.

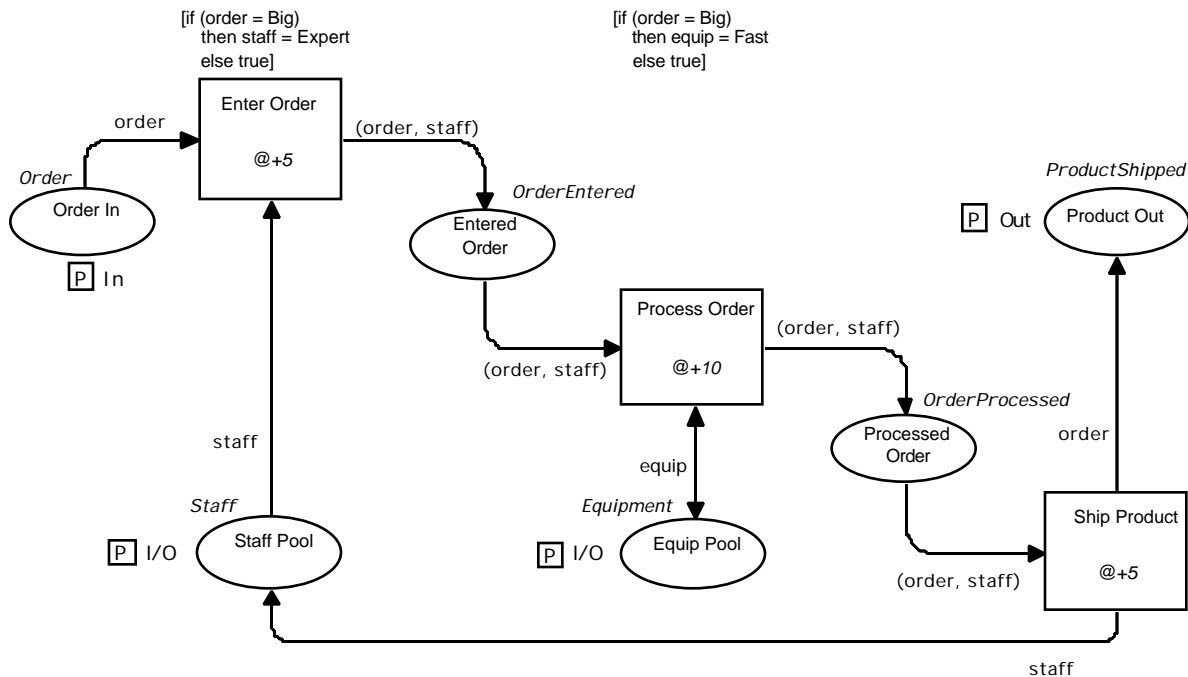
Suppose every order takes 5 minutes to enter, 10 minutes to process, and 5 minutes to ship. To represent this, **Process Order** needs the time region $@+10$, and the other two transitions need time regions of $@+5$.

CPN Region command (**CPN** menu) sets the time regions:



Creating a time region is no different graphically from creating a guard or any other text region.

When all the time regions have been added the net should look like this:



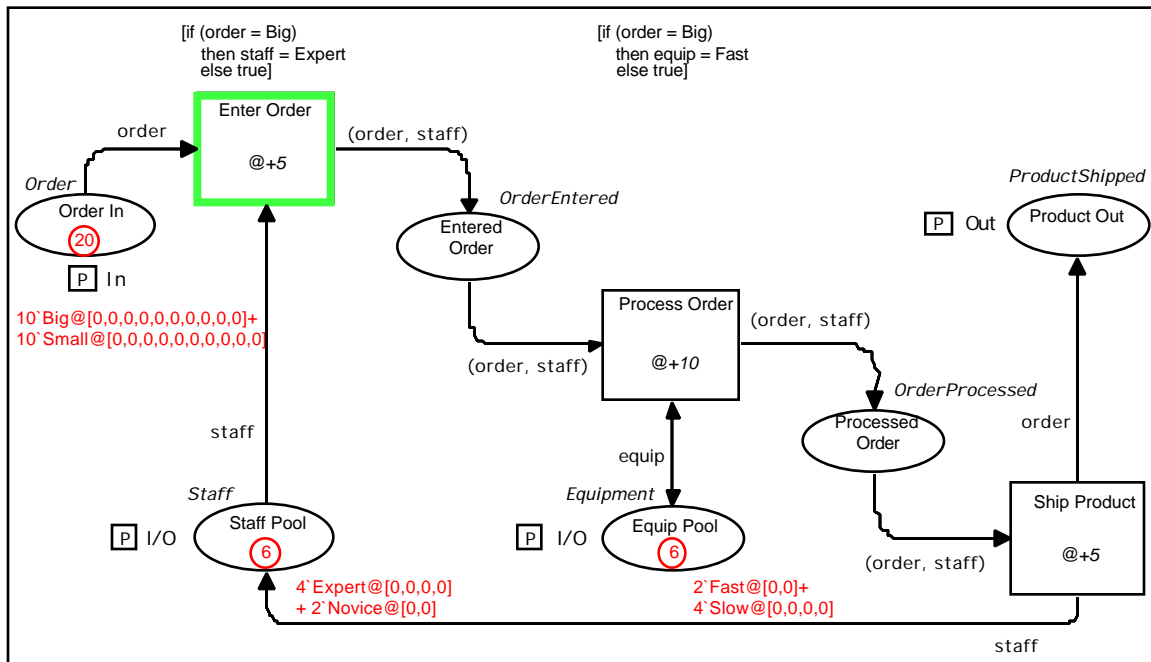
The rest of this chapter illustrates timed execution in the context of the above net.

Executing a Timed CP Net

A timed net is executed just as any net is. The only difference is in what the net does as it executes.

At the beginning of simulation the example net would look something like:

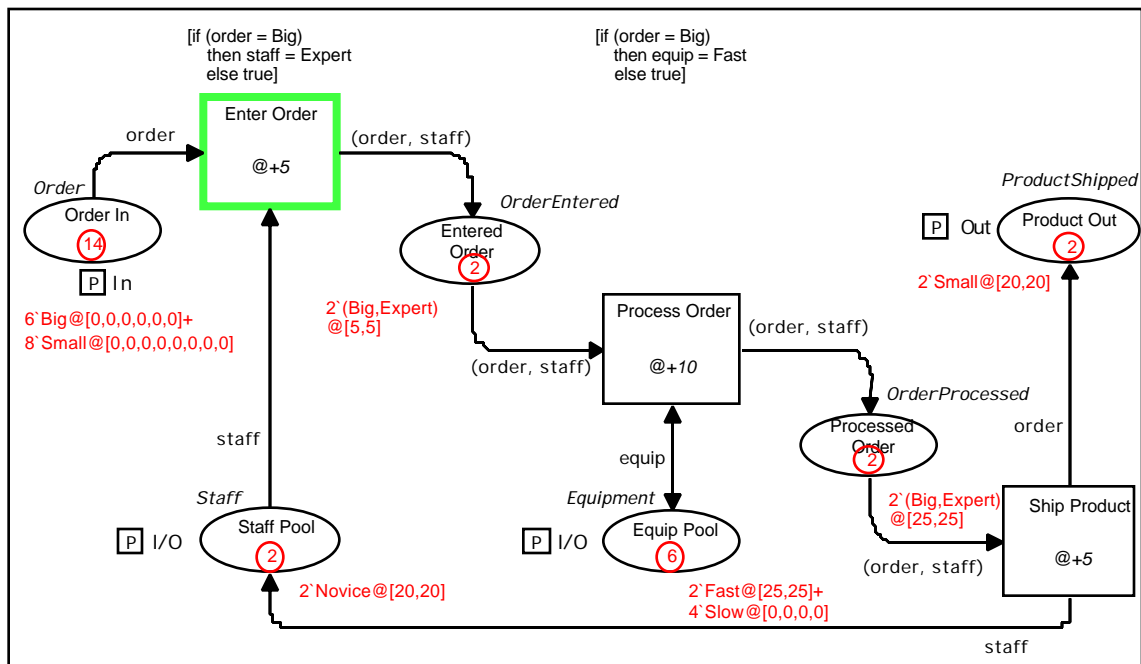
Design/CPN User's Guide



Note the way the time stamps of the individual tokens are indicated. The times indicated are not delays, but the actual times that the clock must reach before the various tokens become available. The time stamps all happen to be the same now, but this will not generally be the case as the net executes.

The status bar displays: Time: 0 Step: 1.

If we begin execution, then stop execution when the simulator has finished Step 3, timing doesn't make any great difference in the sorts of things you see as a net executes. All it has added is more realistic behavior with respect to time. At the end of Step 3, the net would look like this:



Carefully examine Process Order and its input places. There are two Big orders in Entered Order, each with an associated Expert, and there are two Fast equipment pieces in Equip Pool, but Process Order is not enabled. Why not?

The status bar shows that the time is 20. Now look at the time stamps in the two Fast equipment pieces. Both stamps are 25. Therefore the tokens are not available; that is why Process Order is not enabled.

The pieces of equipment that the two Fast tokens represent are actually still in use to process the two orders whose tokens are now in Processed Order, also with time stamps of 25. Note that Ship Order is also not enabled: the jobs will not be ready to be shipped until the model time reaches 25.

Tokens that represent unavailable entities are visible in their places, but functionally they might as well not be there at all. The simulator could have been designed to make such tokens actually invisible, which would better reflect their functional status, but doing so would make a net harder to understand by looking at it, since information would be missing.

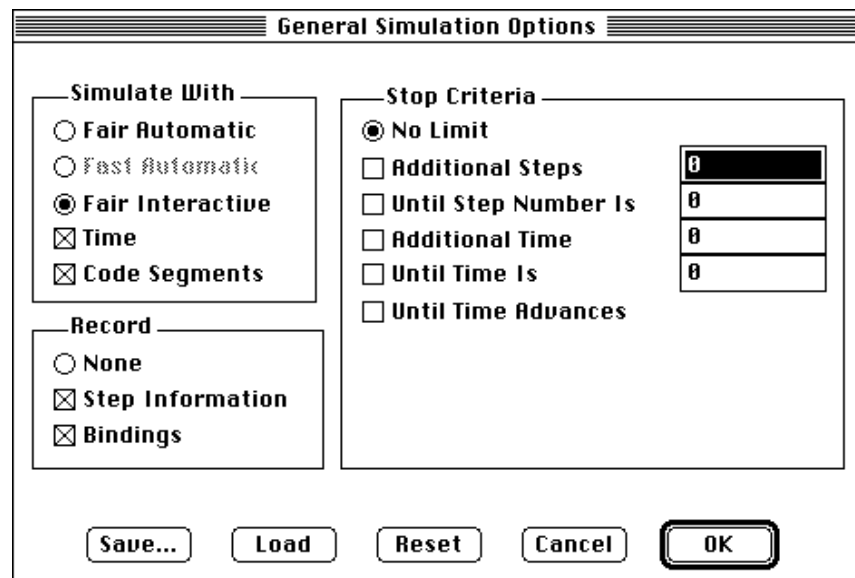
Simulation With and Without Time

It is sometimes useful to execute a timed net without taking account of time. Such execution can make it easier to examine the causal structure of a model, which sometimes becomes obscured when timing complicates the model's behavior.

Timed simulation can be switched on and off whenever execution is stopped between steps.

Choose **General Simulation Options** from the **Set** menu.

The **General Simulation Options** dialog appears:



The dialog box titled "General Simulation Options" contains the following sections and controls:

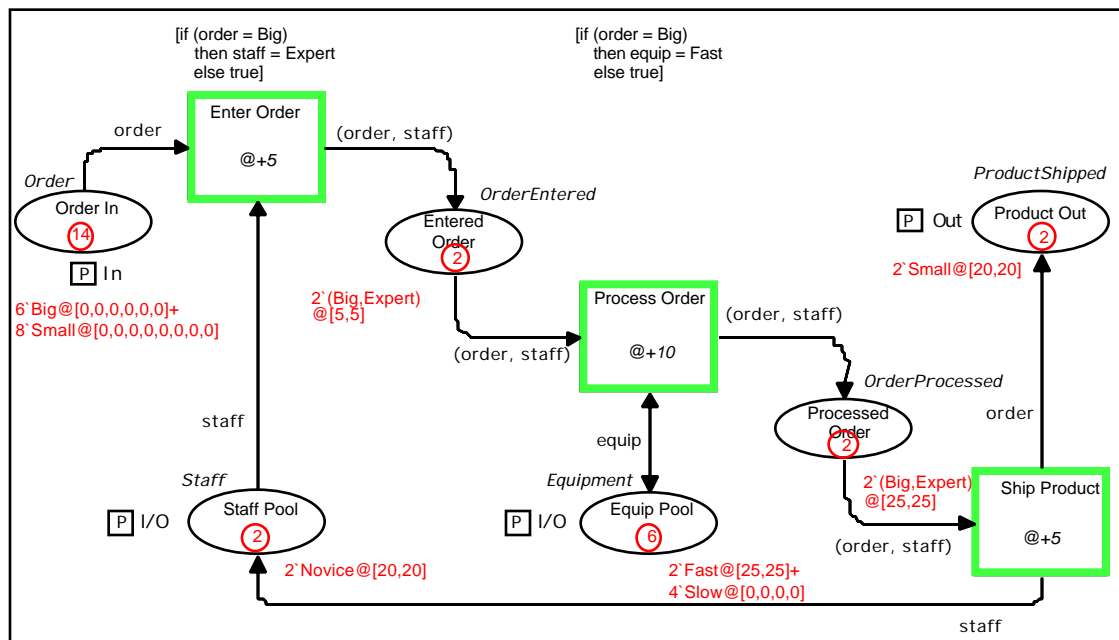
- Simulate With**:
 - ☐ Fair Automatic
 - ☐ Fast Automatic
 - ☒ Fair Interactive
 - ☒ Time
 - ☒ Code Segments
- Record**:
 - ☐ None
 - ☒ Step Information
 - ☒ Bindings
- Stop Criteria**:
 - ☒ No Limit
 - ☐ Additional Steps: 0
 - ☐ Until Step Number Is: 0
 - ☐ Additional Time: 0
 - ☐ Until Time Is: 0
 - ☐ Until Time Advances

Buttons at the bottom: Save..., Load, Reset, Cancel, and OK.

Under **Simulate With**, click **Time**, so that it becomes de-selected.

Click **OK**.

The net now ignores all time stamps: tokens are available if they exist, no matter how they are stamped. Therefore both **Process Order** and **Enter Order** are now enabled:



If we return to doing timed simulation, time stamps are again operative and Process Order and Enter Order are no longer enabled.

More Realistic Timed Behavior

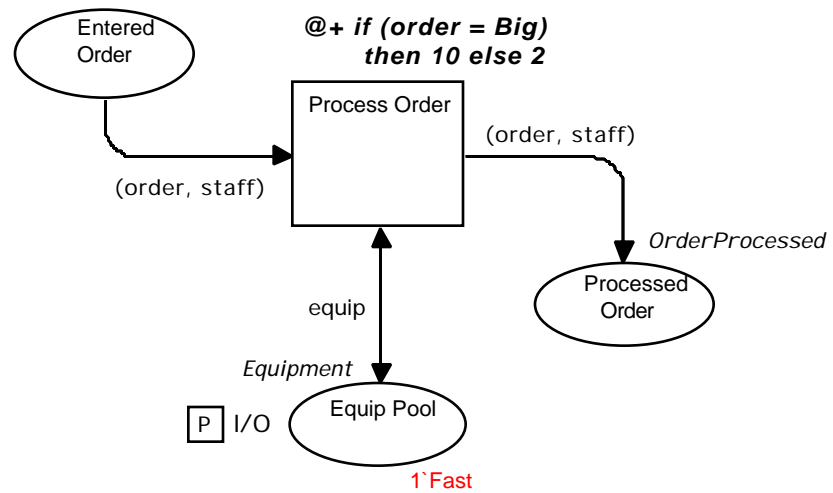
To increase the realism of the model we will change it so that orders of different types are handled differently. To accomplish the change, all we need to do is have one or more time regions assign a delay that is conditional on the type of the order.

All bound arc variables are available to a time region, so everything we need to make this change is already available.

Moving Process Order's time region to the clear area above the transition and editing Process Order's time region to be

```
@+ if (order = Big)
    then 10 else 2
```

The net now looks like this:



Executing the net now produces more complex and realistic behavior.

Chapter 13

Code Segments

Simulation with Code

Each transition may have an attached *code segment* which contains ML code, typically used for I/O such as chart displays and file communication. Code segments may be executed when their parent transition occurs.

You can globally and locally (for each page instance) define whether you want a simulation to generate code for execution with code and then whether or not to execute the code segments. Code must have been generated for execution with code in order to specify code segment execution.

Code generation is specified with the **Simulation Code Options** command (**Set** menu).

Global code segment execution is specified with the **General Simulation Options** command (**Set** menu).

Page level code segment execution is specified with the **Mode Attributes** command (**Set** menu). This command permits you to specify either individual pages or substitution transitions.

Characteristics and Syntax

Code segments are key/popup regions and the keys contain “C” (for Code). Code segments may use CPN variables and may bind CPN variables located on output arcs that are not bound elsewhere.

Each code segment may contain:

- Input pattern (optional)
- Output pattern (optional)
- Code action (mandatory)

Input pattern

An *input pattern* is a tuple of CPN variables, preceded by the keyword `input`. The input pattern lists the CPN variables that can be used in the code action. The code action can use the values of these CPN variables but it cannot change them.

The CPN variables listed in the input pattern can be used in the code action even if you have declared an ML identifier with the same name in the declaration node.

If the input clause is omitted, it implies that no CPN variables can be used in the code action.

Output pattern

An *output pattern* is a tuple of CPN variables, preceded by the keyword `output`. The output pattern lists the CPN variables to be changed as a result of the execution of the code action.

An output pattern must be a CPN variable or a tuple of CPN variables without repetitions.

If the output clause is omitted, it implies that no CPN variables are calculated.

Code actions

A *code action* is an ML expression, preceded by the keyword `action`.

The code action cannot contain any declaration of colorsets, CPN variables, or reference variables. It can, however, apply user-declared and predeclared constants, operations, and functions. In addition, new functions and constants can be defined for local use by means of `let/in/end`.

The code action is executed as a local declaration in an environment containing the CPN variables specified in the input pattern. This guarantees that the code action cannot directly change any CPN variables but only local copies of them.

When the code action has been executed, its result is applied to bind the CPN variables in the output pattern. The code action, when evaluated in an environment containing the input pattern variables must yield a result of the same type as the output pattern. If no output pattern is given, its type is assumed to be unit.

Log regions

When a code segment creates output for the predeclared output stream called “log”, the text argument is appended to the text in the log region for the corresponding transition.

The log region is a key/popup region and the key contains “L” (for log). If the transition does not already have a log region, a new log region is automatically created.

Chapter 14

Statistical Variables

Statistical variables provide the ability to accumulate and access statistics about data generated during the execution of a model. The available statistics include standard deviation, variance, average, and many others.

To use a statistical variable, you declare it, initialize it, update it with values, and access the statistics that the variable has accumulated about those values. Updating and accessing can be freely inter-mixed. A statistical variable may be cleared (reinitialized) at any time, allowing a new set of statistics to be accumulated.

This chapter shows how statistical variables are created, updated, and accessed. Chapter 16 contains an example of statistical variables, and shows how to use them with charts to display the results of model execution.

Using Statistical Variables

The general method for using a statistical variable is:

1. Create the variable.
2. Use the variable to accumulate data about some aspect of an executing model. This is accomplished by calling a statistical update function on the variable and supplying the new data.
3. Use the variable to derive the results of statistical calculations performed on the data it has accumulated. This is accomplished by calling statistical access functions on the variable.

Statistical variables are declared in the global declaration node, and initialized, updated, and accessed in code segments or user-supplied functions called by code segments.

Structure of Statistical Variables

A statistical variable could be implemented as a list of all the data that it has accumulated, and the task of calculating statistics about the data could be left until the statistics are needed. But accumulating a large quantity of data would use up too much memory, and the process of calculating statistics on it would cause significant delays.

Instead a statistical variable is a structure with a slot for the value of each supported statistical function. Each time the variable is updated, each of its slots is recalculated to contain the value of its particular statistical function computed over all the data that the variable has accumulated. A statistical variable retains the most recent value with which it was updated, but earlier values are not kept individually. This approach conserves memory and distributes the time needed to calculate statistical results into imperceptible increments.

Creating Statistical Variables

To declare an integer statistical variable, for example “StatVarName”, type:

```
val StatVarName = SV'creatint ();
```

The value returned is:

```
> val StatVarName = statvar {Avrg = ref 0.0, Count = ref 0, First = ref 0, Maxi = ref 0, Mini = ref 0, SSum = ref 0, Sx = ref 0, Ssd = ref 0.0, Std = ref 0.0, Value = ref 0, Vari = ref 0.0}
```

Note the structure of the variable: it is a record with a slot for each statistical function that the variable keeps track of.

The variable StatVarName now exists. It must be initialized before values are added and used.

Initializing Statistical Variables

To initialize the integer statistical variable StatVarName created above, type:

```
SV'init StatVarName;
```

The variable StatVarName is now ready to receive values via the function SV'upd.

Updating Statistical Variables

To update the integer statistical variable `StatVarName` with the value 4, type:

```
SV'upd (StatVarName, 4);
```

`StatVarName` now has one value, the integer 4. Subsequent uses of `SV'upd` updates `StatVarName` with additional values. For example, typing:

```
SV'upd (StatVarName, 2);  
SV'upd (StatVarName, 8);
```

updates `StatVarName` with the values 2 and 8. Its first value is 4, its current value is 8, and the sum of all accumulated values is 14.

Accessing Statistical Variables

The following functions can be used to access the values stored in a statistical variable. Except as noted, these values are not individual data values with which the variable has been updated, but calculations performed on all such values. The sample values shown are those that would be returned from a statistical variable updated with the values shown in the previous section.

Average

To obtain the average of the values accumulated in `StatVarName`, type:

```
SV'avrg StatVarName;
```

The value returned is 4.666666666666667.

Count

To obtain the count of the number of values accumulated in `StatVarName`, type:

```
SV'count StatVarName;
```

The value returned is 3.

Current Value

To obtain the most recent value with which `StatVarName` was updated, type:

```
SV'value StatVarName;
```

The value returned is 8.

First

To obtain the first value with which `StatVarName` was updated, type:

```
SV'first StatVarName;
```

The value returned is 4.

Maximum

To obtain the maximum of the values accumulated in `StatVarName`, type:

```
SV'max StatVarName;
```

The value returned is 8.

Minimum

To obtain the minimum of the values accumulated in `StatVarName`, type:

```
SV'min StatVarName;
```

The value returned is 2.

Standard Deviation

To obtain the standard deviation of the values accumulated in `StatVarName`, type:

```
SV'std StatVarName;
```

The value returned is 3.055050463303893

Sum

To obtain the sum of the values accumulated in `StatVarName`, type:

```
SV'sum StatVarName;
```

The value returned is 14.

Sum of the Squares

To obtain the sum of the squares of the values accumulated in `StatVarName`, type:

```
SV'ss StatVarName;
```

The value returned is 84.

Sum of the Squares of Deviation

To obtain the sum of the squares of deviation of the values accumulated in `StatVarName`, type:

```
SV'ssd StatVarName;
```

The value returned is 18.666666666666666

Variance

To obtain the variance of the values accumulated in `StatVarName`, type:

```
SV'vari StatVarName;
```

The value returned is 9.333333333333329

Clearing Statistical Variables

To clear `StatVarName`, type:

```
SV'init StatVarName;
```

`SV'init` is the same function with which the variable was originally initialized. Using it again clears all values in `StatVarName`. The variable is again ready to receive values via the function `SV'upd`.

Chapter 15

Chart Facilities

Overview of Charts

Charts are used to display numeric information in graphical form. The information is often drawn from statistical variables, but it can come from any source.

The CPN chart facility provides three types of charts:

1. **Bar Chart:** Displays values as the lengths of sections within one or more bars. Updating the chart modifies the lengths of any or all bar sections.
2. **History Chart:** Displays values as the lengths of sections within a single bar. Updating the chart creates a new bar; older bars are scrolled automatically to make room for the new bar. Each bar remains on display until it scrolls off the edge of the chart
3. **Line Chart:** Displays values as distances of lines from an axis. Updating the chart adds a new segment to any or all lines. All data written to the chart remains visible indefinitely. The chart's display scale is automatically adjusted as needed to provide room.

Creating Charts

A model can create and display a chart at any time during execution. To create a chart, use the **Chart** command (**CPN** menu). To change a chart definition, use **Chart Attributes** (**Set** menu).

Updating Charts

Every chart has an associated code segment. This code segment contains function calls that supply new values to be displayed in the chart. As model execution proceeds, every chart's code segment is executed as specified by the chart's attributes. This execution calls the functions listed in the code segment. The chart is thereby updated.

A chart can also be updated in the simulator by selecting the chart, then executing **Update Chart (Sim menu)**.

Bar Charts

Bar charts display the values of expressions as the lengths of one or more bars. The values of several expressions can be displayed as separate sections, called *parts*, within an individual bar. Typically a bar chart displays values of several expressions calculated at the same simulation step or model time.

This section covers the essentials of bar chart creation and use. A complete example of bar chart use appears in the next chapter.

Bar Chart Nomenclature

This figure shows a typical bar chart, and gives the names of its components:

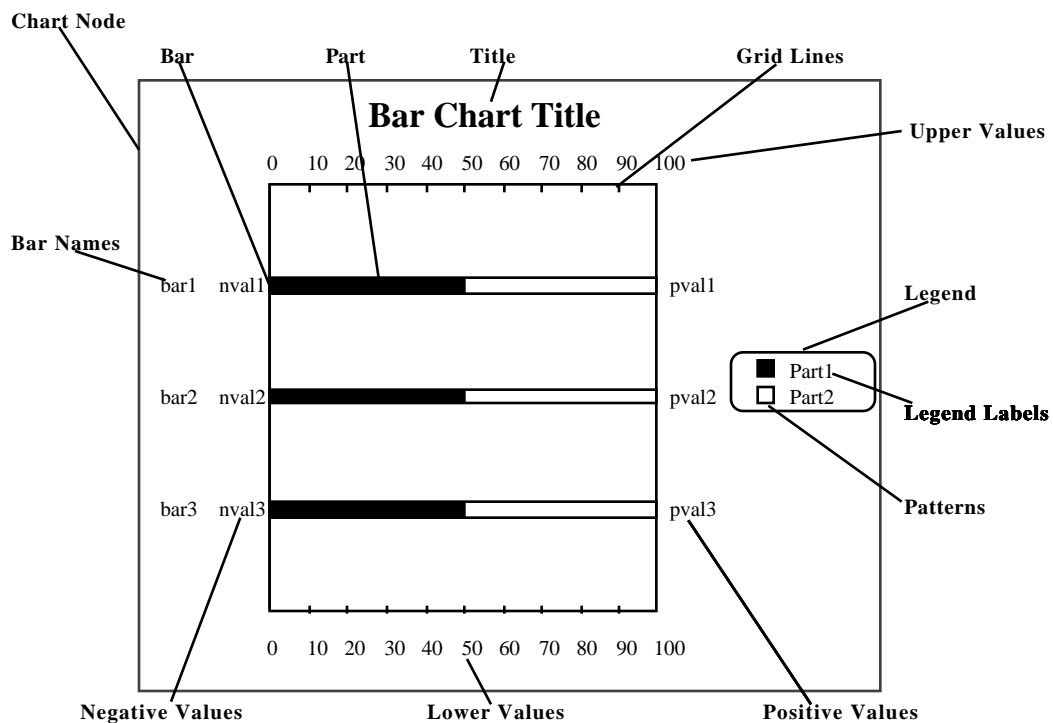


Chart Node

The chart node surrounds the other chart objects. It has a box shape, white fill and distinctive border.

Title

The title gives the chart a name. It exists to describe the chart to the user; it has no formal significance.

Bars

The bars form a two-dimensional display in which each row is a bar. Bars may optionally be subdivided into two or more sections called *parts*. A bar that has not been subdivided has a single part. Every part in a bar has a distinctive fill pattern.

Bar Names

The bar labels contain text strings that describe the contents of the corresponding bar.

Positive and Negative Values

The positive and negative values are labels that contain integer/real values that reflect the current length of the corresponding bar. A negative value label contains the sum of all parts that have negative values, and similarly for a positive value label. These values are automatically updated when the update functions are called.

Upper and Lower Values

The upper and lower values are labels that contain integer/real values that give the scale of the horizontal axis. When the chart is updated, they are automatically updated as needed to reflect the range of values currently displayed in the chart.

Legend

The legend describes the meaning of the patterns used in the bar parts.

Patterns

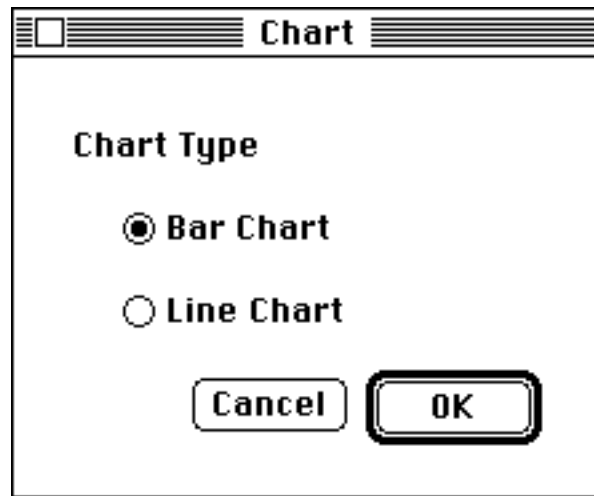
The patterns contained in the legend are those used in the bar parts. The systems assigns the specific patterns used.

Legend Labels

The legend labels are positioned next to the patterns in the legend and contain text strings that describe the contents of the corresponding bar column.

Creating a Bar Chart

To create a bar chart, select the **Chart** command (**CPN** menu). Design/CPN displays the following dialog:



If no charts have been created, the default is **Bar Chart**. Otherwise it is the type of the most recently created chart.

Select **Bar Chart** if necessary.

Click **OK**.

The chart tool appears:



Creating a bar chart is graphically the same as creating a rectangle. Depressing the mouse creates a default size chart node. The chart tool is positioned at the lower right corner. Dragging this corner adjust the shape of the chart. To move the chart as a whole, hold down **SHIFT** while dragging the corner.

Releasing the mouse terminates node creation and causes the system to display the **Bar Chart** dialog.

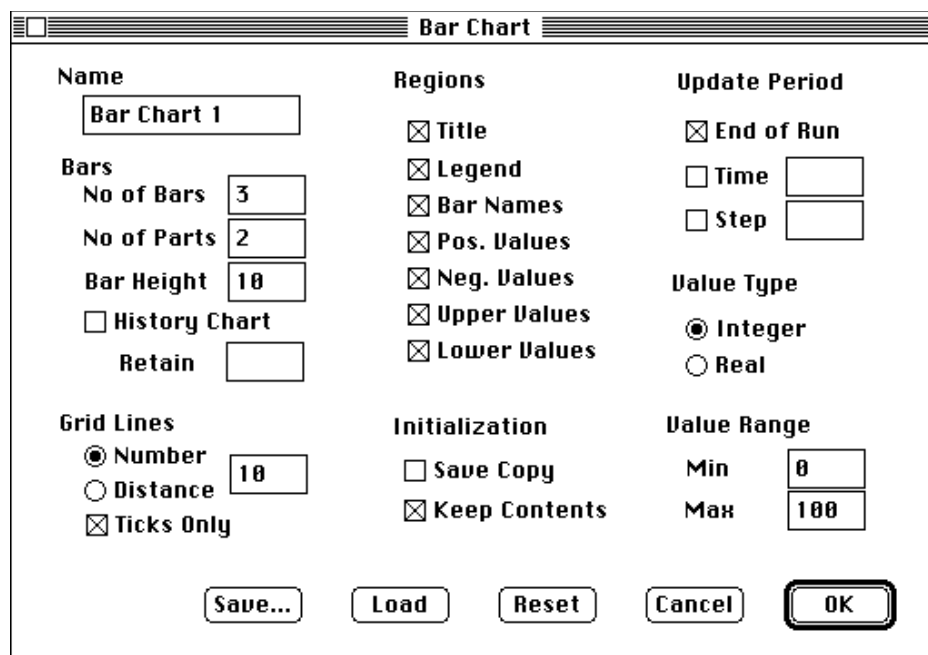
Bar Chart Dialog

Bar Chart		
Name <input type="text" value="Bar Chart 1"/>	Regions	Update Period
Bars	<input checked="" type="checkbox"/> Title	<input checked="" type="checkbox"/> End of Run
No of Bars <input type="text" value="1"/>	<input type="checkbox"/> Legend	<input type="checkbox"/> Time <input type="text"/>
No of Parts <input type="text" value="1"/>	<input checked="" type="checkbox"/> Bar Names	<input type="checkbox"/> Step <input type="text"/>
Bar Height <input type="text" value="10"/>	<input checked="" type="checkbox"/> Pos. Values	Value Type
<input type="checkbox"/> History Chart	<input checked="" type="checkbox"/> Neg. Values	<input checked="" type="radio"/> Integer
Retain <input type="text"/>	<input checked="" type="checkbox"/> Upper Values	<input type="radio"/> Real
Grids Lines	Initialization	Value Range
<input checked="" type="radio"/> Number <input type="text" value="10"/>	<input type="checkbox"/> Save Copy	Min <input type="text" value="0"/>
<input type="radio"/> Distance	<input checked="" type="checkbox"/> Keep Contents	Max <input type="text" value="100"/>
<input checked="" type="checkbox"/> Ticks Only		
<input type="button" value="Save..."/> <input type="button" value="Load"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/> <input type="button" value="OK"/>		

This dialog defines the appearance, initialization, method of updating, and value range for the chart. The defaults are as shown in the dialog above. The meanings of the options in this dialog are covered in Chapter 21, "CPN Menu Commands," section "Chart," subsection "Bar Chart Dialog." Please read the information there before you proceed.

Select options and enter values to describe the desired bar chart.

The values that produce the bar chart shown in the nomenclature bar chart above are:



The dialog box is titled "Bar Chart". It contains several sections of controls:

- Name:** A text field containing "Bar Chart 1".
- Bars:** Three numeric input fields: "No of Bars" (3), "No of Parts" (2), and "Bar Height" (10). Below them is a checkbox for "History Chart" and a "Retain" checkbox.
- Regions:** A list of checkboxes: "Title", "Legend", "Bar Names", "Pos. Values", "Neg. Values", "Upper Values", and "Lower Values". All are checked.
- Update Period:** A checkbox for "End of Run" (checked), and two input fields for "Time" and "Step".
- Value Type:** Two radio buttons: "Integer" (selected) and "Real".
- Grid Lines:** Three radio buttons: "Number" (selected), "Distance", and "Ticks Only". A numeric input field next to "Number" contains "10".
- Initialization:** Two checkboxes: "Save Copy" and "Keep Contents".
- Value Range:** Two input fields: "Min" (0) and "Max" (100).

At the bottom are five buttons: "Save...", "Load", "Reset", "Cancel", and "OK".

When you have specified the desired options and values:

Click **OK**.

The dialog closes, and the chart appears on the page.

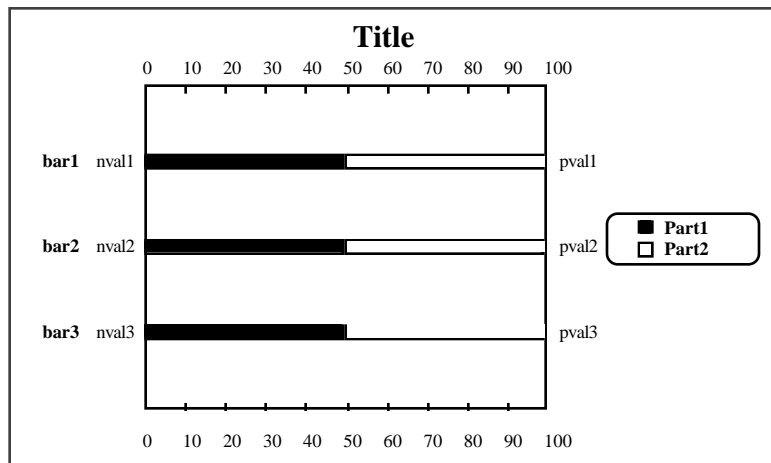
Editing a Bar Chart

Every component of a bar chart is a graphical object, and can be changed in the editor or the simulator. Such changes can be made as soon as the chart is created, or anytime thereafter.

All bar chart components are regions of the chart node. This allows a chart to be scaled by dragging any of its handles with the mouse. All components except fonts will scale themselves proportionately. To change font sizes, select the textual object(s) to be changed, then use **Text Attributes** (**Set** menu).

Supplying Bar Chart Labels

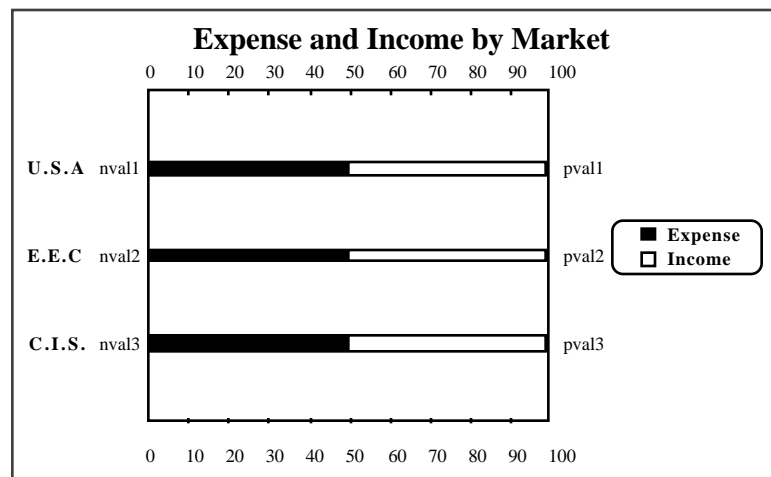
When a newly created bar chart has a title, a legend, and/or bar names, a label that contains dummy text appears for each of these that is present. These dummy labels, including the title, are shown in bold in the following figure:



None of these labels is functionally significant, so there is no requirement to change any of them. Values should be supplied in order to make the chart self-explanatory. To supply these values:

Edit the dummy labels in text mode to specify the desired values.

You can also change the positions, fonts, and styles of any or all dummy labels, and you can resize the chart or any component of it to make room for the edited labels. For example, the labels in the above chart could be edited to make the chart like this:



Redefining a Bar Chart

Any of the attributes specified for a bar chart at its creation may subsequently be changed, with the exception of the chart name, which remains the same throughout the life of the chart. Changes can be made in the editor or the simulator. To change an existing chart:

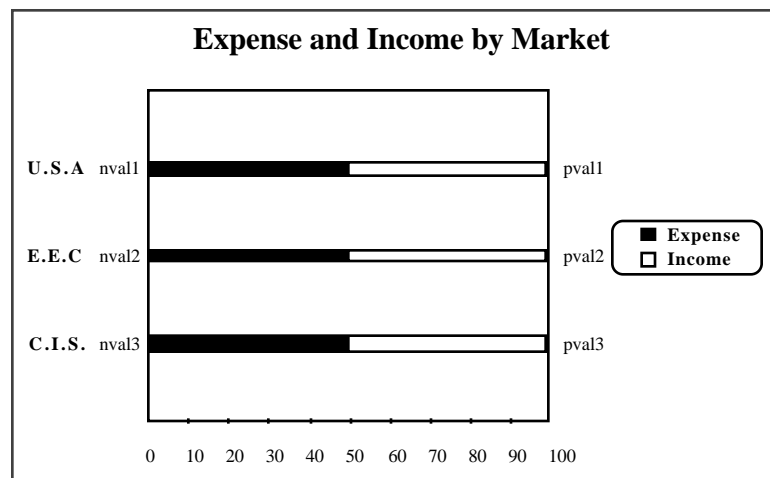
Select **Chart Attributes** from the **Set** menu.

The **Chart Attributes** dialog appears. The chart name does not appear, because it cannot be changed. All other attributes look as they did when you clicked **OK** to approve the chart definition currently in effect.

Make any desired changes to chart attributes.

Click **OK** to approve the changes.

For example, suppose you redefined the above chart by deselecting the **Upper Values** option. The chart would then look like this:



Some changes require reswitching, chart initialization, and/or replacement of the chart's code segment. Design/CPN will notify you if any of these is necessary.

Updating a Bar Chart

When a bar chart is created, it is automatically supplied with a code segment. This segment contains everything that is necessary to update the chart except the update values themselves. For example, the code segment for the bar chart in the above figure initially looks like this:

```
action
SC_upd_chart{
  sc = "Bar Chart 0",
  values = [
    (1, [v1, v2], true),
    (2, [v1, v2], true),
    (3, [v1, v2], true)]};
```

The bars in a bar chart are numbered from top to bottom. The chart in the example has three bars, so its code segment has a line for each. Each line begins with the number of the bar it controls: 1, 2, and 3.

The parts in a bar are numbered from left to right. The chart in the example has two parts, so the line for each bar has a field for each constituent part: `v1` and `v2`.

Thus the arrangement of the fields in a bar chart code segment is visually similar to the arrangement of the corresponding bars and parts in the chart itself. This makes it easy to keep track of which code segment field corresponds to which bar and part.

Specifying Bar Chart Values

To specify the values to be written to a bar chart each time it is updated:

Use text mode to replace each code segment value field with an expression that when evaluated will yield the desired information.

The only restriction on the expressions supplied is that each must evaluate to an integer if the chart is an integer chart, or a real if it is a real chart. The most commonly used source of bar chart data is a function that accesses a statistical variable, as described in Chapter 14, “Statistical Variables.”

Using constants for bar chart fields would rarely be useful in practice, but there is nothing wrong with doing so. Suppose the bar chart in the above figure were updated by executing this code segment:

```
action
SC_upd_chart{
sc = "Bar Chart 0",
values = [
(1, [50, 20], true),
(2, [~20, 40], true),
(3, [10, ~30], true)];
```

Note that unary minus is indicated by a tilde, rather than an ordinary minus sign, consistent with ML practice generally. Following the update, the chart would look like this:



The chart has automatically rescaled the horizontal axis to accommodate the new values, and has reversed the order of the parts in the third bar to put the part representing a negative value on the negative side of the vertical axis.

Requirement to Specify Values

The value fields `v1`, `v2`, etc. that are initially supplied in a bar chart code segment are not themselves legal values. They are only placeholders to indicate where values must be supplied. Failure to replace any of them with an actual value will result in a syntax error.

Code segment examples in this chapter that do not concern value fields sometimes show unsubstituted fields. In order for these segments to actually execute, values would have to be substituted for all such fields.

Other Ways to Update a Bar Chart

There are two other ways in which a bar chart may be updated:

1. A call to `SC_upd_chart`, the function that performs the actual update, may appear in the code segment of any transition. The chart will then be updated whenever the transition fires. If that is the only kind of updating that is desired, the chart's code segment can be deleted.
2. The chart can be selected, and the **Update Chart** command chosen from the **Sim** menu.

Additional Chart Code Segment Capabilities

A bar chart's code segment provides several capabilities additional to those mentioned so far:

- It can contain initialization code that executes only when the simulator is entered or **Initial State (Sim menu)** is chosen.
- It can specify that updating a chart is conditional on a boolean expression. Updating will occur only if this expression evaluates to **true** when the code segment is executed.
- It can specify that updating an individual bar is conditional on a boolean expression. Updating of the particular bar will occur only if this expression evaluates to **true** when the code segment is executed.
- It's action section is not restricted to updating a bar chart. It can contain any CPN ML code that could appear in a transition's code segment.

Initialization Code

A bar chart code segment may contain a section called an *initialization section*. This section begins with the keyword `init`. Code in an initialization section will be executed when the simulator is entered or **Initial State (Sim menu)** is chosen.

The presence of an initialization section has no effect on execution of a bar chart code segment's action section. Code in the action section will be executed only when the chart is to be updated, just as if there were no initialization section.

To add an initialization section to a bar chart code segment:

Use text mode to insert the keyword `init` followed by the code to be executed.

For example, suppose we wanted to initialize a statistical variable named `statvar` whenever a model's initial state was established or re-established. The following code segment would accomplish this:

```
init
SV'init statvar;

action
SC_upd_chart{
sc = "Bar Chart 0",
values = [
(1, [v1, v2], true),
(2, [v1, v2], true),
(3, [v1, v2], true)];
```

Conditional Chart Update

A bar chart code segment may contain a *conditional section*. This section begins with the keyword `cond`. A conditional section must contain a boolean expression. This expression will be evaluated whenever the chart is to be updated. If it evaluates to **true**, the update will take place. If it evaluates to **false**, the update will not take place, and the chart will be left unchanged.

To add a conditional section to a bar chart code segment:

Use text mode to insert the keyword `cond` followed by a boolean expression.

For example, suppose we wanted to update a chart only when the standard deviation stored in the statistical variable `statvar` is greater than 2. The following code segment would accomplish this:

```
cond
(SV'std statvar) > 2;

action
SC_upd_chart{
sc = "Bar Chart 0",
values = [
(1, [v1, v2], true),
(2, [v1, v2], true),
(3, [v1, v2], true)];
```

Conditional Bar Update

Each code segment line that specifies the updating of a particular bar consists of three components. The first gives the number of the bar, and the second gives the values with which the bar is to be updated, as described above. The third is a boolean expression that controls updating of the bar. This expression is called the *control expression*.

When a chart is being updated, the control expression in each update line is evaluated before that bar is changed. If the expression evaluates to true, the bar will be updated. If it evaluates to false, the bar will be left unchanged.

The default control expression in an update line is **true**. This value is specified in each line when the initial code segment is created. To make bar updating conditional:

Use text mode to replace the default control expression **true** in the update line for the bar with a boolean expression.

For example, suppose we have a bar chart with three bars and two parts, want to update the first and third bar with every update, and want to update the second only when the variance stored in the statistical variable `statvar` is less than 5. The following code segment would accomplish this:

```
action
SC_upd_chart{
sc = "Bar Chart 0",
values = [
(1, [v1, v2], true),
(2, [v1, v2], ((SV'vari statvar) < 5),
(3, [v1, v2], true)];
```

Arbitrary Code in the Action Section

The essential purpose of the action section of a bar chart's code segment is to contain code that updates the chart. However, it can also contain any CPN ML code that could appear in a transition's code segment. This code may come before and/or after the call to the update function `SC_upd_chart`, and will be executed whenever the chart is updated. If there is a conditional section that evaluates to **false**, so that updating does not take place, no other code in the action section will be executed either.

Copying, Cutting, and Pasting a Bar Chart

A bar chart can be copied, cut, and pasted as any graphical object can. However, a pasted chart is no longer functional: it just an auxiliary node with various regions. It retains its appearance, but will not be updated by further model execution, and cannot be changed by the **Chart Attributes** command (**Set** menu).

Deleting a Bar Chart

To delete a bar chart, select the chart, then press **DELETE** or execute **Cut**.

History Charts

History charts are essentially the same as bar charts. The only difference is in the method of updating. When a bar chart is updated, each bar changes to reflect the new values specified for it. When a history chart is updated, a new bar is created at the bottom of the chart, and the existing bars scroll up to make room for it. If the chart is full, updating causes old bars to be discarded at the top of the chart to make room for new bars added at the bottom.

This section focuses on areas in which history charts differ from ordinary bar charts. Be sure to read the preceding section, “Bar Charts,” before you proceed. A complete example of history chart use appears in the next chapter.

Creating a History Chart

To create a history chart:

 Select the **Chart** command (**CPN** menu).

The **Chart Type** dialog appears.

 Specify a bar chart.

 Click **OK**.

The Editor enters chart creation mode, and the chart tool appears.

 Use the mouse to indicate the shape and position of the chart.

The **Bar Chart** dialog appears:

To specify that the chart is to be a history chart:

Check the **History** option.

Retain

The **Retain** field determines how many bars will be retained when the history chart is scrolled up to make room for a new bar. The maximum permissible value is one less than **No of Bars**. This value will cause the chart to retain all but one bar when it scrolls, so that room for each new bar is made by discarding the oldest existing bar.

Lower **Retain** values cause more than one bar to be discarded each time the chart is scrolled. Subsequent updates progressively fill the resulting empty space, from the top down, until the chart is again full. The next update will then reduce the number of bars again to the **Retain** value.

If **Retain** is zero, the chart will be cleared completely by the first update after it becomes full, then filled again by that and subsequent updates.

Specify the desired figure for **Retain**.

Other History Chart Options

All the options and their effects are the same as for ordinary bar charts.

Editing and Redefining a History Chart

These are done exactly as they are for ordinary bar charts. You can redefine a snapshot chart to be a history chart, or vice versa, by toggling the **History** option. Doing this requires reswitching, and causes Design/CPN to replace the chart's code segment with one appropriate to its new definition.

Updating a History Chart

The code segment of a history chart is somewhat different than the segment for a snapshot chart. If the example chart used in the previous section is defined as a history chart, its code segment initially looks like this:

```
action
HC_upd_chart{
hc = "Bar Chart 0",
values = [
([v1, v2], true)],
tags = ["Tag"]};
```

History charts are updated one bar at a time, so there is only one line to specify bar values. Bars in the example chart have two parts, so the update line contains two value fields: `v1` and `v2`.

A history chart's code segment defines a parameter that is not applicable to bar charts: the `tags` field. This field can be used to supply a label for each bar when it is created. Such a label is called a *tag*.

Specifying History Chart Values

To specify the values to be written to each new bar that is created each time a history chart is updated:

Use text mode to replace each code segment value field with an expression that when evaluated will yield the desired information.

The expressions must evaluate to an integer if the chart is an integer chart, or a real if it is a real chart.

To specify the value to be written to the tag of each new bar:

Use text mode to replace the string “Tag” with an expression that when evaluated will yield a string that is the desired tag.

It is often convenient to give each bar a tag that indicates the simulated time at which the data in the chart was generated. This information can be obtained by calling the `time` function. This function returns an integer, but a string is needed. The integer can be converted to a string by using the `makestring` function.

Suppose we have a history chart that consists of bars with two parts, and two functions `getval1` and `getval2`, which return the values that are to be displayed in the two parts of each bar. The following code segment would accomplish the desired updating:

```
action
HC_upd_chart{
hc = "Bar Chart 0",
values = [
([getval1(), getval2()], true)],
tags = [makestring (time())]};
```

Copying, Cutting, Pasting, and Deleting a History Chart

These operations are the same as for ordinary bar charts.

Line Charts

Line charts display values as distances of lines from an X and a Y axis. Updating the chart adds a new segment to any or all lines. All data written to the chart remains visible indefinitely; the chart's display scale is automatically adjusted as needed to provide room.

This section covers the essentials of line chart creation and use. In order to be self-contained, this section repeats some of the information that appears in the section on bar charts earlier in this chapter. If you are reading this chapter sequentially, you can safely skip over the repeated information. A complete example of line chart use appears in the next chapter.

Line Chart Nomenclature

This figure shows a typical line chart, and gives the names of its components:

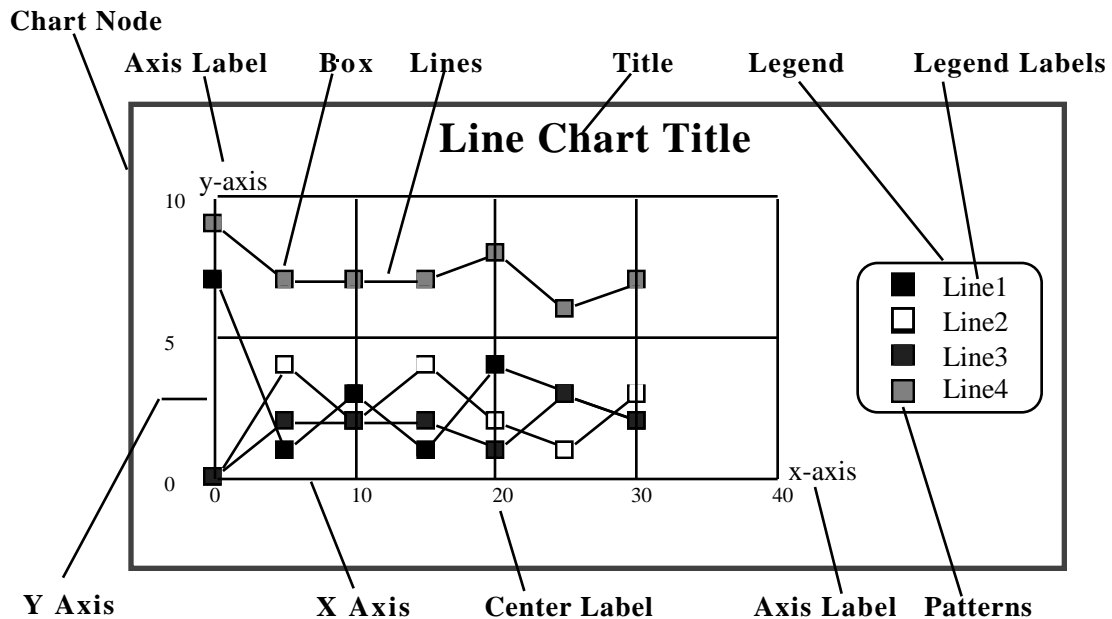


Chart Node

The chart node surrounds the other chart objects. It has a box shape, white fill and distinctive border.

Title

The title gives the chart a name. It exists to describe the chart to the user; it has no formal significance.

X and Y Axes

The X and Y axes define the coordinate space of the chart.

Axis labels

Axis labels name the entities which are represented along the two axes.

Box

A box terminates each line segment and displays the pattern associated with that line.

Line

Each chart line is shown as a series of line segments. The segments may be terminated by boxes. The system distinguishes each line by a unique pattern displayed in the box terminator.

Line charts are commonly incremented left-to-right, but this is just a convention: a new point on a chart line may be drawn anywhere. For example, object movement within a plane could be displayed using a line chart.

Legend

The legend describes the patterns associated with the lines.

Patterns

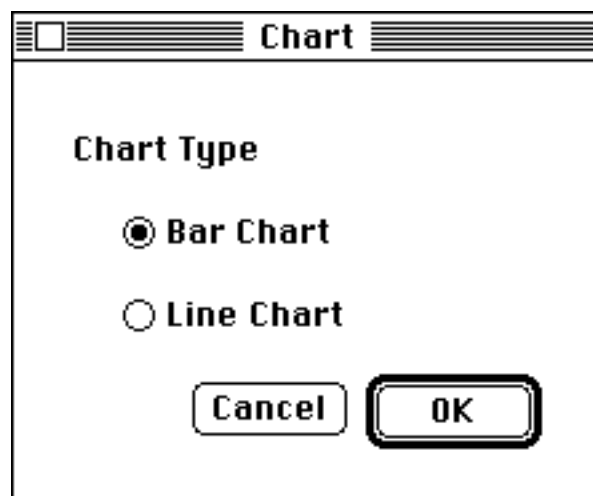
The patterns contained in the legend are those used in the chart line box terminators. The system assigns the specific patterns used.

Legend labels

The legend labels are positioned next to the patterns in the legend and contain text strings that describe the contents of the corresponding chart line.

Creating a Line Chart

To create a line chart, select the **Chart** command (CPN menu). Design/CPN displays the following dialog:



Design/CPN User's Guide

If no charts have been created, the default is **Bar Chart**. Otherwise it is the type of the most recently created chart.

Select **Line Chart** if necessary.

Click **OK**.

The Editor enters chart creation mode. The chart tool appears:



Creating a line chart is graphically the same as creating a rectangle. Depressing the mouse creates a default size chart node. The chart tool is positioned at the lower left corner. Dragging this corner adjusts the shape of the chart. To move the chart as a whole, hold down SHIFT while dragging the corner.

Releasing the mouse terminates node creation and causes the system to display the **Line Chart** dialog.

Line Chart Dialog

Line Chart			
Name Line Chart 1		Regions <input checked="" type="checkbox"/> Title <input type="checkbox"/> Legend <input checked="" type="checkbox"/> Axis Names	
Lines No of Lines: 1 Box Size: 10 <input type="checkbox"/> Start at Origin <input type="checkbox"/> Horiz/Vert		Update Period <input checked="" type="checkbox"/> End of Run <input type="checkbox"/> Time <input type="checkbox"/> Step	
Grid Lines x: <input checked="" type="radio"/> Number: 10 <input type="radio"/> Distance: 10 <input checked="" type="checkbox"/> Ticks Only		Value Type <input checked="" type="radio"/> Integer <input type="radio"/> Real	
y: <input checked="" type="radio"/> Number: 10 <input type="radio"/> Distance: 10 <input checked="" type="checkbox"/> Ticks Only		Value Range x: Min: 0, Max: 100 y: Min: 0, Max: 100	
Initialization <input type="checkbox"/> Save Copy <input checked="" type="checkbox"/> Keep Contents <input type="checkbox"/> Start at Origin		Origin x: 0 y: 0	
		Overflow <input checked="" type="radio"/> Rescale Axis <input type="radio"/> Move Axis	
Save... Load Reset Cancel OK			

This dialog defines the appearance, initialization, method of updating, and value range for the chart. The defaults are as shown in the dialog above. The meanings of the options in this dialog are covered in Chapter 21, "CPN Menu Commands," section "Chart," subsection "Line Chart Dialog." Please read the information there before you proceed.

Select options and enter values to describe the desired bar chart.

The values that produce the line chart shown in the nomenclature line chart above are:

The dialog box is titled "Line Chart" and contains the following sections:

- Name:** Line Chart 0
- Lines:**
 - No of Lines: 4
 - Box Size: 10
 - ☐ Start at Origin
 - ☐ Horiz/Vert
- Regions:**
 - ☒ Title
 - ☒ Legend
 - ☒ Axis Names
- Initialization:**
 - ☐ Save Copy
 - ☒ Keep Contents
 - ☐ Start at Origin
- Update Period:**
 - ☒ End of Run
 - ☐ Time: []
 - ☐ Step: []
- Origin:**
 - x: 0
 - y: 0
- Value Type:**
 - ☒ Integer
 - ☐ Real
- Overflow:**
 - ☒ Rescale Axis
 - ☐ Move Axis
- Grid Lines:**
 - x: ☐ Number, ☒ Distance: 10
 - y: ☐ Number, ☒ Distance: 5
 - ☐ Ticks Only
- Value Range:**
 - x: Min: 0, Max: 40
 - y: Min: 0, Max: 10

Buttons at the bottom: Save..., Load, Reset, Cancel, OK.

When you have specified the desired options and values:

Click **OK**.

The dialog closes, and the chart appears on the page.

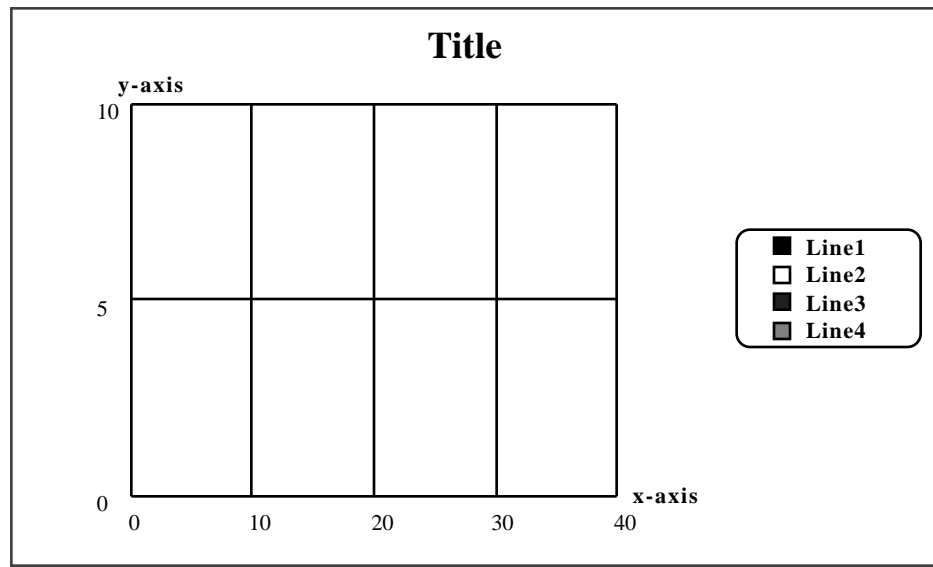
Editing a Line Chart

Every component of a line chart is a graphical object, and can be changed in the editor or the simulator. Such changes can be made as soon as the chart is created, or anytime thereafter.

All line chart components are regions of the chart node. This allows a chart to be scaled by dragging any of its handles with the mouse. All components except fonts will scale themselves proportionately. To change font sizes, select the textual object(s) to be changed, then use **Text Attributes (Set menu)**.

Supplying Line Chart Labels

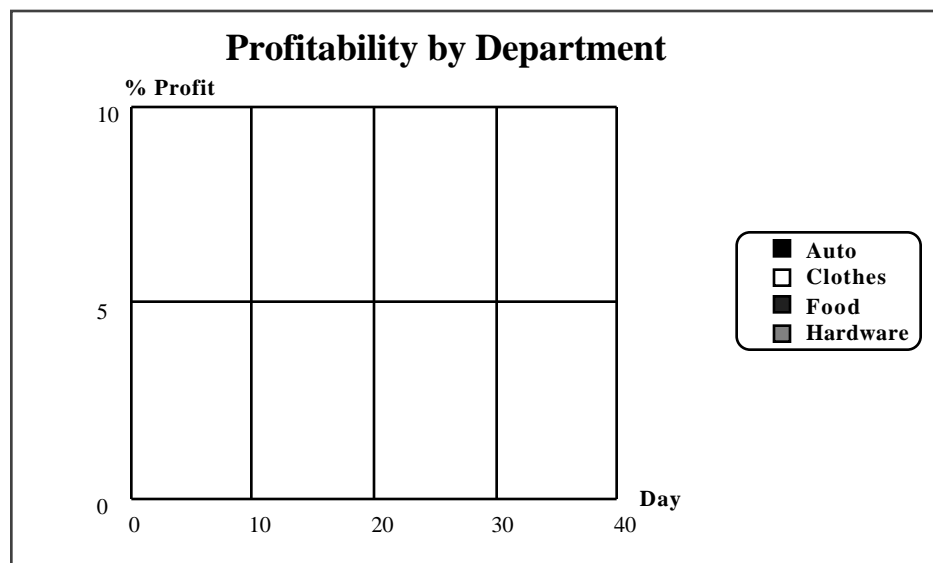
When a newly created line chart has a title, axis labels, and/or a legend, a label that contains dummy text appears for each of these that is present. These dummy labels, including the title, are shown in bold in the following figure:



None of these labels is functionally significant, so there is no requirement to change any of them. Values should be supplied in order to make the chart self-explanatory. To supply these values:

Edit the dummy labels in text mode to specify the desired values.

You can also change the positions, fonts, and styles of any or all dummy labels, and you can resize the chart or any component of it to make room for the edited labels. For example, the labels in the above chart could be edited to make the chart look like this:



Redefining a Line Chart

Any of the attributes specified for a line chart at its creation may subsequently be changed, with the exception of the chart name, which remains the same throughout the life of the chart. Changes can be made in the editor or the simulator. To change an existing chart:

Select **Chart Attributes** from the **Set** menu.

The **Chart Attributes** dialog appears. The chart name does not appear, because it cannot be changed. All other attributes look as they did when you clicked **OK** to approve the chart definition currently in effect.

Make any desired changes to chart attributes.

Click **OK** to approve the changes.

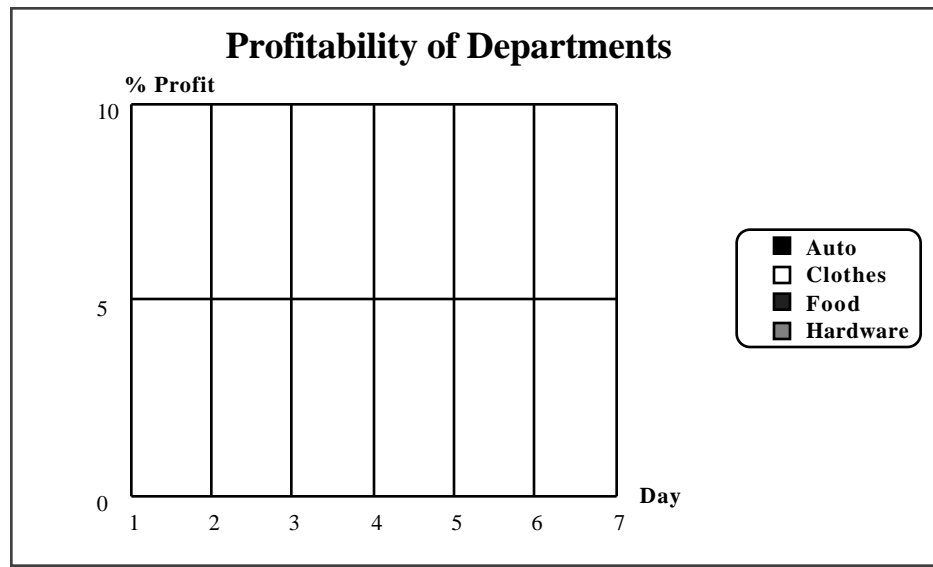
For example, suppose you needed to redefine the above chart by redefining the X axis to run from 1 to 7, so that the chart can be used to represent a one-week period. The way to do this is to redefine the **Value Range**, **Grid Lines** and **X Origin** as shown:

The dialog box is titled "Line Chart" and contains several sections for configuring chart attributes:

- Lines:**
 - No of Lines: 4
 - Box Size: 10
 - ☐ Start at Origin
 - ☐ Horiz/Vert
- Regions:**
 - ☒ Title
 - ☒ Legend
 - ☒ Axis Names
- Initialization:**
 - ☐ Save Copy
 - ☒ Keep Contents
 - ☐ Start at Origin
- Update Period:**
 - ☒ End of Run
 - ☐ Time
 - ☐ Step
- Origin:**
 - x: 1
 - y: 0
- Value Type:**
 - ☒ Integer
 - ☐ Real
- Overflow:**
 - ☒ Rescale Axis
 - ☐ Move Axis
- Grid Lines:**
 - x:** ☐ Number, ☒ Distance 1, ☐ Ticks Only
 - y:** ☐ Number 5, ☒ Distance, ☐ Ticks Only
- Value Range:**
 - x:** Min 1, Max 7
 - y:** Min 0, Max 10

Buttons at the bottom: Save..., Load, Reset, Cancel, OK.

The chart would then look like this:



Some changes require reswitching, chart initialization, and/or replacement of the chart's code segment. Design/CPN will notify you if any of these is necessary.

Updating a Line Chart

When a line chart is created, it is automatically supplied with a code segment. This segment contains everything that is necessary to update the chart except the update values themselves. For example, the code segment for the line chart in the above figure initially looks like this:

```
action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, vx, vy, true, true),
(2, vx, vy, true, true),
(3, vx, vy, true, true),
(4, vx, vy, true, true)]};
```

A line chart's code segment contains a line of code, called an update line, for each line in the chart. The lines in a line chart are numbered from top to bottom in the same order that they are listed in the legend box. The chart in the example has four lines, so its code segment has four update lines. Each update line begins with the number of the chart line it controls: 1, 2, and 3, and 4.

Updating a chart line consists of drawing a box at a particular X and Y position, and connecting that box to the box previously drawn, or to the origin if it is the first box drawn and **Start at Origin** was checked in the **Chart Attributes** dialog. Each update line there-

fore contains two value fields: v_x for the X coordinate, and v_y for the Y coordinate.

Specifying Line Chart Values

To specify the values to be written to a line chart each time it is updated:

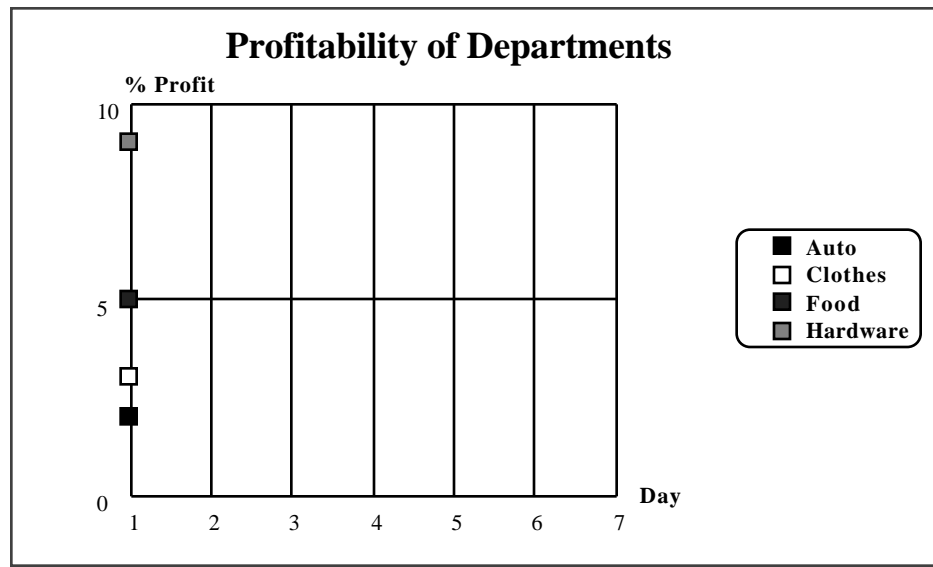
Use text mode to replace each update line value field with an expression that when evaluated will yield the desired information.

The only restriction on the expressions supplied is that each must evaluate to an integer if the chart is an integer chart, or a real if it is a real chart. The most commonly used source of line chart data is a function that accesses a statistical variable, as described in Chapter 14, "Statistical Variables."

Using constants for line chart update fields would rarely be useful in practice, but there is nothing wrong with doing so. Suppose the line chart in the above figure were updated by executing this code segment:

```
action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, 1, 2, true, true),
(2, 1, 3, true, true),
(3, 1, 5, true, true),
(4, 1, 9, true, true)]};
```

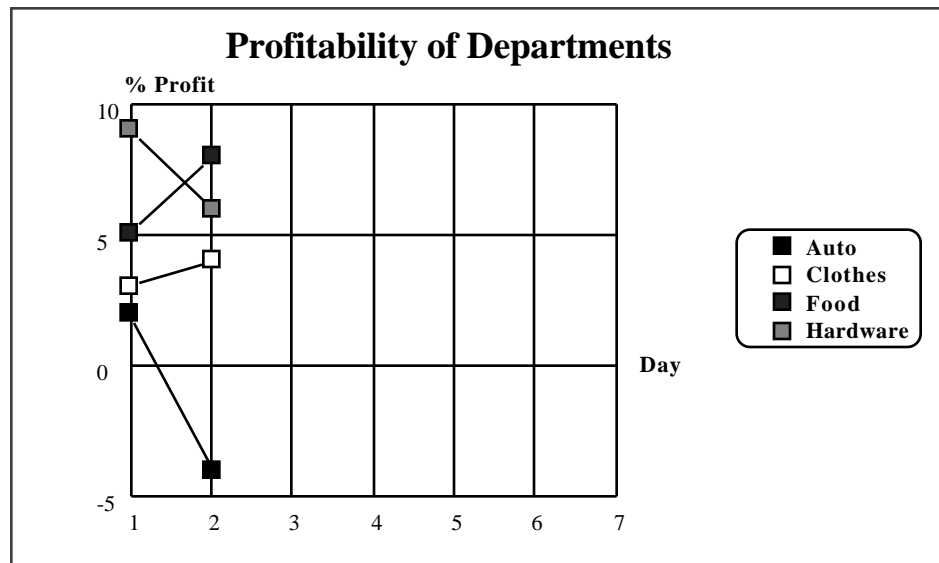
Following the update, the chart would look like this:



Figures for each day can be added by performing additional updates. If a department lost rather than made money on a given day, the loss could be represented as a negative profit. For example, suppose that on Day 2, the Auto department experienced a 4% loss. This can be indicated by using a negative number in an update, as shown in the following code segment:

```
action
LC_upd_chart{
  lc = "Line Chart 0",
  values = [
    (1, 2, ~2, true, true),
    (2, 2, 4, true, true),
    (3, 2, 8, true, true),
    (4, 2, 6, true, true)]};
```

Note that unary minus is indicated by a tilde, rather than an ordinary minus sign, consistent with ML practice generally. After updating with this code segment, the chart would look like this:



Design/CPN has automatically rescaled the Y axis to accommodate the negative value.

There is no requirement that successive updates to a line chart proceed from left to right, or show any other regular pattern. Updates may be made with any values in any order. Design/CPN will draw lines and rescale axes as needed to make the updates.

Requirement to Specify Values

The value fields `v1`, `v2`, etc. that are initially supplied in a line chart code segment are not themselves legal values. They are only placeholders to indicate where values must be supplied. Failure to replace any of them with an actual value will result in a syntax error.

Code segment examples in this chapter that do not concern value fields sometimes show unsubstituted fields. In order for these segments to actually execute, values would have to be substituted for all such fields.

Other Ways to Update a Line Chart

There are two other ways in which a line chart may be updated:

1. A call to `LC_upd_chart`, the function that performs the actual update, may appear in the code segment of any transition. The chart will then be updated whenever the transition fires. If that is the only kind of updating that is desired, the chart's code segment can be deleted.
2. The chart can be selected, and the **Update Chart** command selected from the **Sim** menu.

Additional Chart Code Segment Capabilities

A line chart's code segment provides several capabilities additional to those mentioned so far:

- It can contain initialization code that executes only when the simulator is entered or **Initial State (Sim menu)** is chosen.
- It can specify that updating a chart is conditional on a boolean expression. Updating will occur only if this expression evaluates to **true** when the code segment is executed.
- It can specify that updating an individual line is conditional on a boolean expression. Updating of the particular line will occur only if this expression evaluates to **true** when the code segment is executed.
- It can specify that drawing of the segment that connects a new update point and the previous update point for any line is conditional on a boolean expression.
- It's action section is not restricted to updating a line chart. It can contain any CPN ML code that could appear in a transition's code segment.

Initialization Code

A line chart code segment may contain a section called an *initialization section*. This section begins with the keyword `init`. Code in an initialization section will be executed when the simulator is entered or **Initial State (Sim menu)** is chosen.

The presence of an initialization section has no effect on execution of a line chart code segment's action section. Code in the action section will be executed only when the chart is to be updated, just as if there were no initialization section.

To add an initialization section to a line chart code segment:

Use text mode to insert the keyword `init` followed by the code to be executed.

For example, suppose we wanted to initialize a statistical variable named `statvar` whenever a model's initial state was established or re-established. The following code segment would accomplish this:


```
init
SV'init statvar;

action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, vx, vy, true, true),
(2, vx, vy, true, true),
(3, vx, vy, true, true),
(4, vx, vy, true, true)];
```

Conditional Chart Update

A line chart code segment may contain a *conditional section*. This section begins with the keyword `cond`. A conditional section must contain a boolean expression. This expression will be evaluated whenever the chart is to be updated. If it evaluates to **true**, the update will take place. If it evaluates to **false**, the update will not take place, and the chart will be left unchanged.

To add a conditional section to a line chart code segment:

Use text mode to insert the keyword `cond` followed by a boolean expression.

For example, suppose we wanted to update a chart only when the standard deviation stored in the statistical variable `statvar` is greater than 2. The following code segment would accomplish this:

```
cond
(SV'std statvar) > 2;

action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, vx, vy, true, true),
(2, vx, vy, true, true),
(3, vx, vy, true, true),
(4, vx, vy, true, true)];
```

Conditional Line Update

An update line in a line chart code segment consists of five components. The first gives the number of the line, and the second and third give the X and Y coordinates of the new update point, as described above. The fourth is covered in the next section. The fifth

is a boolean expression that controls updating of the line. This expression is called the *control expression*.

When a chart is being updated, the control expression in each update line is evaluated before that line is changed. If the expression evaluates to true, the line will be updated. If it evaluates to false, the line will be left unchanged.

The default control expression in an update line is **true**. This value is specified in each update line when the initial code segment is created. To make line updating conditional:

Use text mode to replace the default control expression **true** in the update line with a boolean expression.

For example, suppose we have a four-line chart, want to update the first, third line, and fourth lines with every update, and want to update the second only when the variance stored in the statistical variable `statvar` is less than 5. The following code segment would accomplish this:

```
action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, vx, vy, true, true),
(2, vx, vy, true, ((SV'vari statvar) < 5)),
(3, vx, vy, true, true),
(4, vx, vy, true, true)]};
```

Conditional Drawing of Line Segments

The fourth field of an update line is a boolean expression called the *drawing expression*. This expression is evaluated whenever the chart line is updated. If it evaluates to **true**, a segment is drawn between the new update point and the previous one. If it evaluates to **false**, no segment is drawn. Drawing of boxes is unaffected.

The default drawing expression in an update line is **true**. This value is specified in each update line when the initial code segment is created. To make the drawing of segments conditional:

Use text mode to replace the default drawing expression **true** in the update line with a boolean expression.

Sometimes no segments are desired. To specify this:

Use text mode to replace the default drawing expression **true** in the update line with **false**.

For example, suppose we have a four-line chart, and that we want to:

- Draw segments for the first and third line with every update.
- Never draw segments for the second line.
- Draw a segment for the fourth line only when sum of the squares stored in the statistical variable `statvar` is greater than 100.

The following code segment would accomplish this:

```
action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, vx, vy, true, true),
(2, vx, vy, false, true),
(3, vx, vy, true, true),
(4, vx, vy, ((SV'ss statvar)) > 100, true)]];
```

Arbitrary Code in the Action Section

The essential purpose of the action section of a line chart's code segment is to contain code that updates the chart. However, it can also contain any CPN ML code that could appear in a transition's code segment. This code may come before and/or after the call to the update function `LC_upd_chart`, and will be executed whenever the chart is updated. If there is a conditional section that evaluates to **false**, so that updating does not take place, no other code in the action section will be executed either.

Copying, Cutting, and Pasting a Line Chart

A line chart can be copied, cut, and pasted as any graphical object can. However, a pasted chart is no longer functional. It retains its appearance, but will not be updated by further model execution, and cannot be changed by the **Chart Attributes** command (**Set** menu).

Deleting a Line Chart

To delete a line chart, select the chart, then press **DELETE** or execute **Cut**.

Chapter 16

Using Charts and Statistical Variables

This chapter uses a simple model, the Resource Use Model, to demonstrate the capabilities of charts and statistical variables. Be sure you have read Chapters 14 and 15 before you proceed: this chapter does not repeat the material presented there.

Introduction to the Resource Use Model

This section introduces the Resource Use Model. We will use this model to illustrate the use of statistical variables and the creation and use of each type of chart. You should take some time to become familiar with this model in general before examining its use of charts and statistical variables.

Overview of the Model

The Resource Use Model represents a computer in which processes request, use, and release resources. Each process goes through a seven-step cycle:

1. Do processing without any resources (`Processing Locally`).
2. Ask the computer to allocate resources (`Request Resources`).
3. If necessary wait for resources to be allocated (`Awaiting Resources`).
4. Take control of allocated resources (`Obtain Resources`).
5. Do processing with the resources (`Using Resources`).

6. Return the resources to the computer (`Release Resources`).
7. Return to Step 1.

Resources that are not currently in use are kept in a pool (`Resource Pool`).

Resource Use Model Data

The data in the model, and the colorsets used to represent it, are:

Processes

The model includes seven processes; they are represented as an indexed colorset (`Proc`), and are named `P(1)`, `P(2)`, ..., `P(7)`. The processes are not used directly, but as a component of the tuple colorset `ProcRes` (below).

Resources

The model includes nine resources; they are represented as another indexed colorset (`Res`), and are named `R(1)`, `R(2)`, ..., `R(9)`. The resources are kept in a pool (`Resource Pool`) when they are not in use.

Allocation Patterns

Each process requests the same three resources every time through the loop. The association of a process and the resources it requests is established using a tuple colorset (`ProcRes`). The tuples have the form:

`(process, resource, resource, resource)`

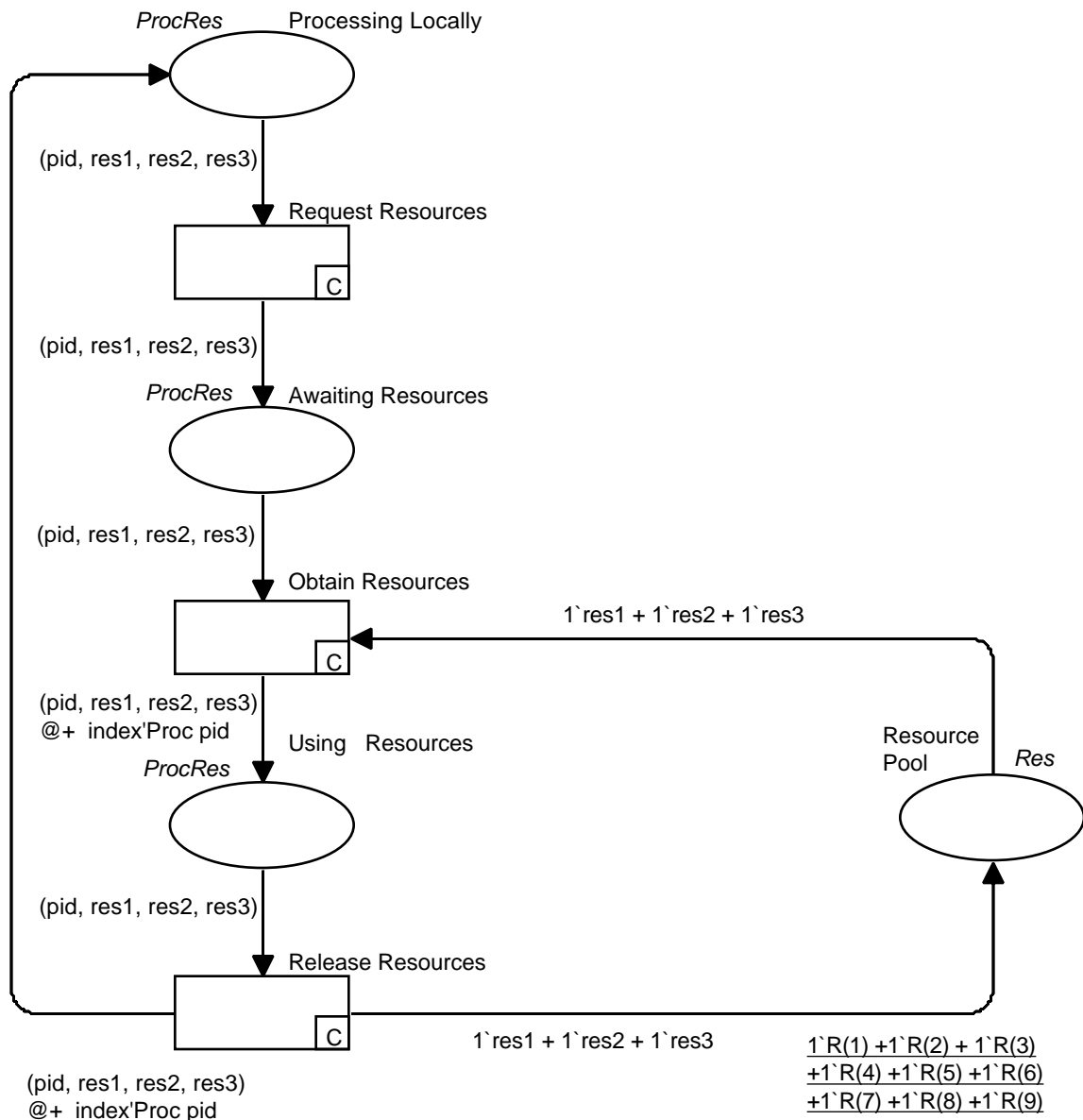
where **process** is one of the processes `P(1)` . . . `P(7)`, and each **resource** is one of the resources `R(1)` . . . `R(9)`. To provide a convenient way of keeping track of which processes use which resources, `ProcRes` tokens, rather than `Proc` tokens, represent processes during simulation.

The association of a process with its required resources is an arbitrary one; the associations defined in the model have been selected to generate considerable competition for resources.

Resource Use Model Graphics and Global Declarations

The following are the graphics and the Global Declaration Node of the Resource Use Model. Declarations relating to statistical variables and token counts, and the techniques by which the model uses them to accumulate statistics, will be explained in later sections.

1`P(1), R(1), R(2), R(3))
+ 1`P(2), R(2), R(3), R(4))
+ 1`P(3), R(3), R(4), R(5))
+ 1`P(4), R(4), R(5), R(6))
+ 1`P(5), R(5), R(6), R(7))
+ 1`P(6), R(6), R(7), R(8))
+ 1`P(7), R(7), R(8), R(9))



```
(* STATISTICAL VARIABLES AND CHART FACILITIES EXAMPLES *)
(* GLOBAL DECLARATIONS *)

(* DECLARATIONS FOR THE RESOURCE USE MODEL. *)

val ProcCount = 7;          (* Number of processes being simulated. *)
val ResCount = 9;           (* Number of resources being simulated. *)
val MaxCount = max (ProcCount, ResCount); (* Max num. of entities. *)

color Proc = index P with 1..ProcCount declare index; (* Processes. *)
color Res = index R with 1..ResCount;                  (* Resources. *)
color ProcRes = product Proc * Res * Res * Res timed; (* Proc & Res *)

var pid:Proc;          (* Holds a process identifier. *)
var res1, res2, res3:Res; (* Each holds a resource. *)

(* STATISTICAL VARIABLES FOR ACCUMULATING DATA. *)
val isv_ProcLoc = SV'createint (); (* Tokens in Processing Locally. *)
val isv_AwaitRes = SV'createint (); (* Tokens in Awaiting Resources. *)
val isv_UseRes = SV'createint (); (* Tokens in Using Resources. *)
val isv_ResPool = SV'createint (); (* Tokens in Resource Pool. *)

(* REFERENCE VARIABLES FOR HOLDING PREVIOUS TOKEN COUNTS. *)
val old_ProcLoc = ref 0; (* Old token count in Processing Locally. *)
val old_AwaitRes = ref 0; (* Old token count in Awaiting Resources. *)
val old_UseRes = ref 0; (* Old token count in Using Resources. *)
val old_ResPool = ref 0; (* Old token count in Resource Pool. *)
```

Running the Model

When execution of the Resource Use Model begins, nine tokens of colorset `ProcRes`, one for each process, are processing locally (without resources). As execution proceeds, each requests the three resources that are wired into its definition, waits (if necessary) until they are available, obtains them from the resource pool, then uses and releases them.

The progress of `ProcRes` tokens through the loop is controlled by the availability of resources, and by timing delays. Each `ProcRes` token first processes locally, then processes with resources, for a length of time given by index number of the process referenced in the token. That is, the token for `P (1)` processes for one time unit, the token for `P (2)` processes for two time units, etc. This delay pattern has been selected to give each process a different behavior over time.

The dependence of time delay on index number is accomplished via the CPN ML function `index`. This function inputs a member of an index colorset and returns an integer that is the member's index number. For example, if `pid` is bound to `P (3)`, `(index 'Proc`

`pid`) returns 3. The `index` function must be used because the numeric characters an index colorset member's name do not constitute an integer: they are just characters.

Statistics in the Resource Use Model

The statistics kept by the Resource Use Model when it executes all concern the number of tokens that are present in the model's places at various times. These statistics are displayed in charts that are updated at intervals during model execution.

For each of the four places in the model, there is a statistical variable. This variable accumulates statistics about the number of tokens in the place. The names of the variables are:

<u>Transition Name</u>	<u>Statistical Variable</u>
Processing Locally	<code>isv_ProcLoc</code>
Awaiting Resources	<code>isv_AwaitRes</code>
Using Resources	<code>isv_UseRes</code>
Resource Pool	<code>isv_ResPool</code>

For each of the four places in the model, there is also an ordinary reference variable. This variable is used to store the number of tokens that were in the place prior to the most recent change to that number. The names of these variables are:

<u>Transition</u>	<u>Reference Variable</u>
Processing Locally	<code>old_ProcLoc</code>
Awaiting Resources	<code>old_AwaitRes</code>
Using Resources	<code>old_UseRes</code>
Resource Pool	<code>old_ResPool</code>

The purpose of these variables will be explained in the section on bar charts.

Initializing the Statistical Variables

Before execution begins, the four statistical variables are initialized (via `SV'init`), and each is updated with the number of tokens in its place (via `SV'upd`). The initializing and updating happen in the `init` section of the code segment of one of the charts that the model displays. (Any chart's code segment can be used in this way.) The `init` section looks like this:

```
init
(* Initialize the statistical variable structures. *)
SV'init isv_ProcLoc;
SV'init isv_AwaitRes;
SV'init isv_UseRes;
SV'init isv_ResPool;

(* Initialize the statistical variable values. *)
SV'upd (isv_ProcLoc, ProcCount);
SV'upd (isv_AwaitRes, 0);
SV'upd (isv_UseRes, 0);
SV'upd (isv_ResPool, ResCount);
```

ProcCount and ResCount are global variables that give the initial numbers of processes and resources. They are used to avoid wiring these numbers directly into the model. (Strictly speaking, they should also be used to generate the initial markings of Processing Locally and Resource Pool, but doing this would complicate the model in ways that do not bear on the subject of this chapter.)

Updating the Statistical Variables

There are three transitions in the model: Request Resources, Obtain Resources, and Release Resources. When any of these transitions fires, it changes the token counts in the places connected to it. Each of the three transitions has a code segment. This code segment updates the statistical variable for each place whose token count the transition changes.

When a token is added to a place, the statistical variable for that place is updated with the value 1. When a token is subtracted from a place, the statistical variable for that place is accumulated with the value -1 (expressed as ~1 in ML). The result is that calling SV' sum on any of the statistical variables will return the number of tokens currently in its place.

All three code segments follow the same pattern:

1. Save the existing (now outdated) SV' sum of the statistical variable for each place that the transition affects.
2. Update the statistical variables for each of the places that the transition affects.

The code segments look like this:

Using Charts and Statistical Variables

```
(* Request Resources Code Segment *)
action

(* Save previous counts. *)
old_ProcLoc := SV'sum isv_ProcLoc;
old_AwaitRes := SV'sum isv_AwaitRes;

(* Accumulate data. *)
SV'upd (isv_ProcLoc, ~1);
SV'upd (isv_AwaitRes, 1);
```

```
(* Obtain Resources Code Segment *)
action

(* Save previous counts. *)
old_AwaitRes := SV'sum isv_AwaitRes;
old_UseRes := SV'sum isv_UseRes;
old_ResPool := SV'sum isv_ResPool;

(* Accumulate data. *)
SV'upd (isv_AwaitRes, ~1);
SV'upd (isv_UseRes, 1);
SV'upd (isv_ResPool, ~1);
```

```
(* Release Resources Code Segment *)
action

(* Save previous counts. *)
old_ProcLoc := SV'sum isv_ProcLoc;
old_UseRes := SV'sum isv_UseRes;
old_ResPool := SV'sum isv_ResPool;

(* Accumulate data. *)
SV'upd (isv_ProcLoc, 1);
SV'upd (isv_UseRes, ~1);
SV'upd (isv_ResPool, 1);
```

Using the Accumulated Data

Statistical data is of little use unless it can be conveniently accessed. Tables of figures are often used for such purposes, but graphical charts often provide a much more useful view. The rest of this chapter contains an example of each type of chart that Design/CPN provides. The charts will show the following information:

Bar Chart: The number of processes in Processing Locally, Awaiting Resources, and Using

Resources; the number of resources currently available in Resource Pool; and the previous values of each of these quantities. This information is updated every 5 time units.

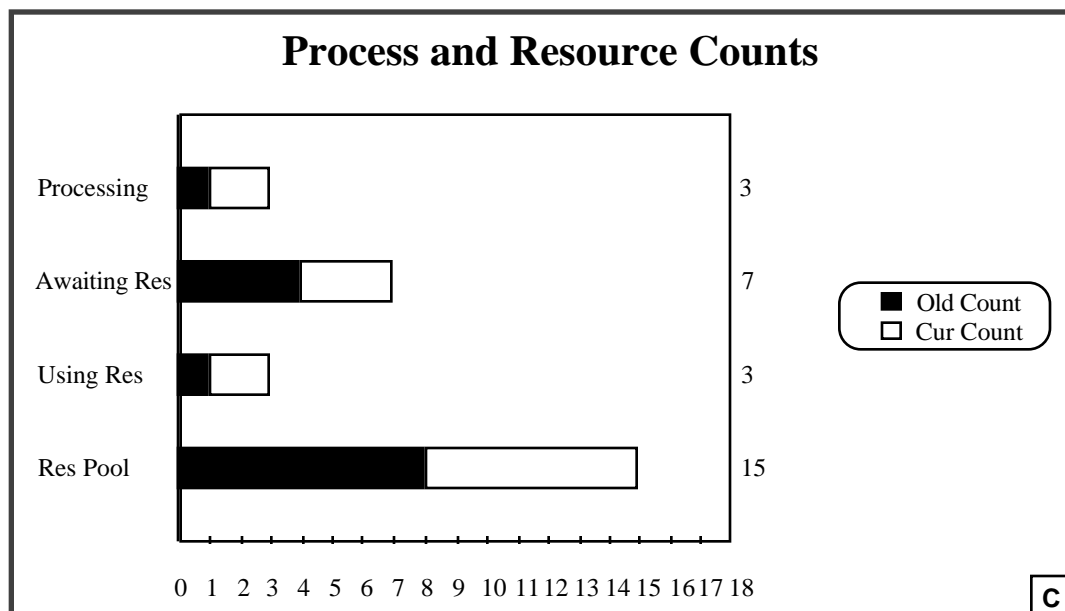
History Chart: The number of processes currently in Processing Locally, Awaiting Resources, and Using Resources; and the values that these quantities had 5, 10, 15, and 20 time units ago.

Line Chart: The number of processes currently in Processing Locally, Awaiting Resources, and Using Resources; the number of resources currently available in Resource Pool; and the values that these quantities had at the end of every period of 5 time units since the simulation began.

As we look at each type of chart in turn, we will use the information it displays to examine the efficiency of the modeled computer system, confirm a suspicion that its efficiency is low, analyze the cause of the inefficiency, and suggest possible improvements.

Resource Use Model: Bar Charts

The following bar chart was produced by the Resource Use Model during a typical execution. The chart shows the number of processes currently in Processing Locally, Awaiting Resources, and Using Resources; the number of resources currently available in Resource Pool; and the previous values of each of these quantities.



Definition of the Bar Chart

The **Chart Attributes** dialog for the above chart look like this:

Bar Chart

Bars

No of Bars: 4

No of Parts: 2

Bar Height: 20

☐ History Chart

Retain: ☐

Regions

☒ Title

☒ Legend

☒ Bar Names

☒ Pos. Values

☐ Neg. Values

☐ Upper Values

☒ Lower Values

Update Period

☒ End of Run

☒ Time: 5

☐ Step:

Value Type

☒ Integer

☐ Real

Grid Lines

☐ Number

☒ Distance: 1

☒ Ticks Only

Initialization

☐ Save Copy

☒ Keep Contents

Value Range

Min: 8

Max: 18

Save... Load Reset Cancel OK

As mentioned above, the bar chart's code segment contains an `init` section that initializes the statistical variables. The section of code segment that updates the chart looks like this:

```
action
SC_upd_chart{
sc = "Bar Chart 0",
values = [
(1, [!old_ProcLoc, SV'sum isv_ProcLoc], true),
(2, [!old_AwaitRes, SV'sum isv_AwaitRes], true),
(3, [!old_UseRes, SV'sum isv_UseRes], true),
(4, [!old_ResPool, SV'sum isv_ResPool], true)];
```

As this segment shows, the first part of each bar shows the previous number of tokens in the place that the bar describes. This information is read from the reference variable for the place. The second part is the number of tokens currently in the place, read from its statistical variable.

Various other statistical functions could be used instead of `SV'sum`, or in addition to it, to retrieve and display information about how the number of tokens in the place has varied over time.

Analysis of the Bar Chart Data

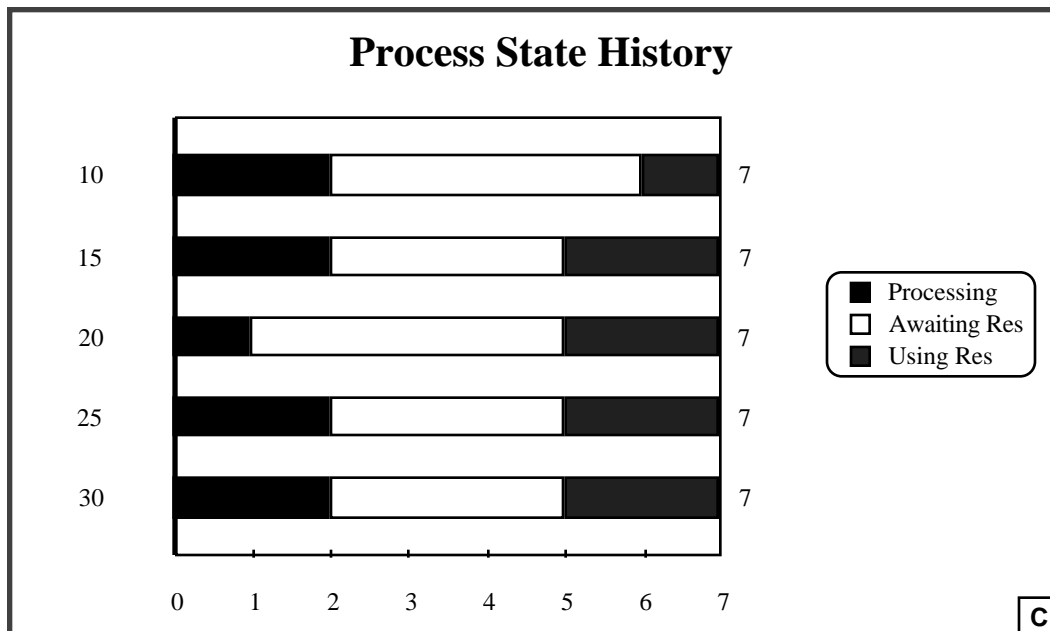
The bar chart indicates that the computer is not very efficient in its use of resources. There are many processes waiting for resources, but only a few are using them, even though many resources are available.

How reliable is this indication? On the basis of this chart alone, we cannot tell. The chart shows only the current and most recent previous value for the number of processes and resources in each state, so the information displayed may not be representative. It is just a snapshot.

To see whether this is a representative picture, we need to be able to track conditions farther into the past than this bar chart permits. We could do this by creating additional columns, to represent views increasingly far into the past, but historical information of this kind is more clearly depicted by using a history chart.

Resource Use Model: History Charts

The following history chart was produced by the Resource Use Model at Time=30 during a typical execution. The chart shows the number of processes currently in *Processing Locally*, *Awaiting Resources*, and *Using Resources*; and the values that these quantities had 5, 10, 15, and 20 time units ago.



Definition of the History Chart

The **Chart Attributes** dialog for the above chart look like this:

The screenshot shows a dialog box titled "Bar Chart". It contains several sections for configuring the chart's appearance and behavior:

- Bars**: Includes "No of Bars" (5), "No of Parts" (3), "Bar Height" (20), a checked "History Chart" checkbox, and "Retain" (4).
- Regions**: Includes checkboxes for "Title", "Legend", "Bar Names", "Pos. Values", "Neg. Values", "Upper Values", and "Lower Values".
- Update Period**: Includes a checked "End of Run" checkbox, "Time" (5), and an unchecked "Step" checkbox.
- Value Type**: Includes radio buttons for "Integer" (selected) and "Real".
- Grid Lines**: Includes radio buttons for "Number" (unselected), "Distance" (selected), and a checked "Ticks Only" checkbox.
- Initialization**: Includes checkboxes for "Save Copy" (unselected) and "Keep Contents" (checked).
- Value Range**: Includes "Min" (8) and "Max" (7) input fields.

At the bottom of the dialog are buttons for "Save...", "Load", "Reset", "Cancel", and "OK".

The history chart's code segment looks like this:

```
action
HC_upd_chart{
hc = "Bar Chart 1",
values = [
([SV'sum isv_ProcLoc, SV'sum isv_AwaitRes, SV'sum isv_UseRes], true)],
tags = [makestring (time())]};
```

As this segment shows, the first part of each bar shows the number of tokens in *Processing Locally*, the second the number of tokens in *Awaiting Resources*, and the third the number in *Using Resources*. Each of these figures is read from the appropriate statistical variable. Each bar is tagged with the time at which the bar was created, obtained by polling the simulator and converting the result to a string.

Analysis of the History Chart Data

The history chart supports the suspicion generated by the bar chart, but it also shows that the data depicted in the bar chart was not entirely representative. The bar chart gave the impression that processes spend most of their time waiting, while the history chart shows that processes divide their time about equally between waiting for resources and using them. There is still too much waiting

going on, but a fair amount of processing is nevertheless taking place.

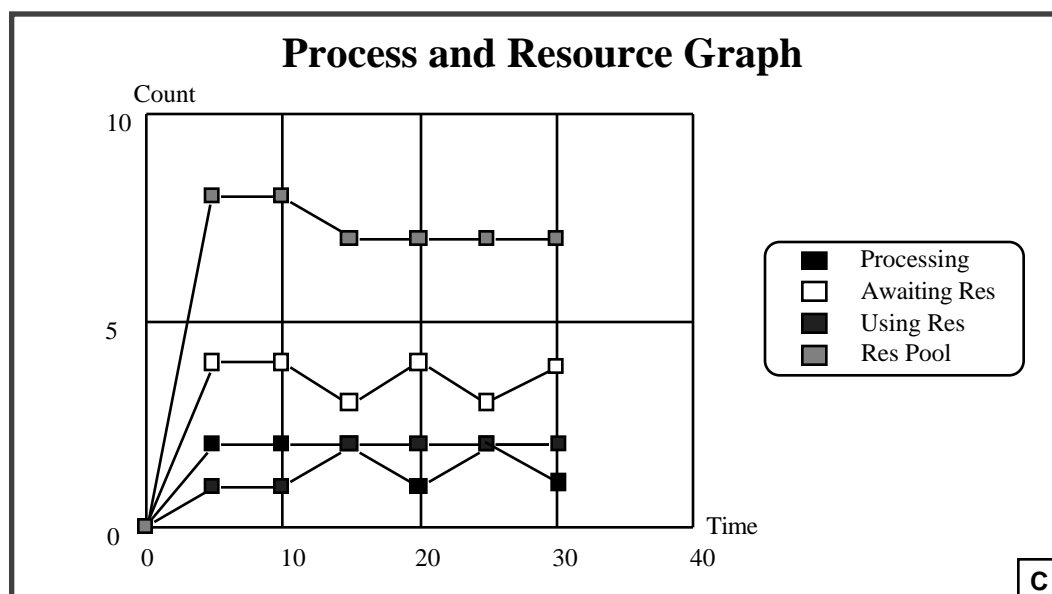
Note that this chart does not show resources: only processes are depicted. The reason is that a history chart is not well adapted to showing different types of data simultaneously: putting data of different kinds into the same bar can be confusing. We could have gotten away with it in this case, but it is not a good practice in general.

Information on the number of resources in use is implicit in the number of processes that are using them, but only because every process uses exactly three resources. A real computer would probably not show such a regular pattern of resource usage.

What we need is a way to simultaneously display historical information for dissimilar types of entity. One method would be to put up two (or more) history charts, but this could require a lot of paging back and forth among the charts to derive an overall picture. A more convenient solution is to use a line chart to display the information.

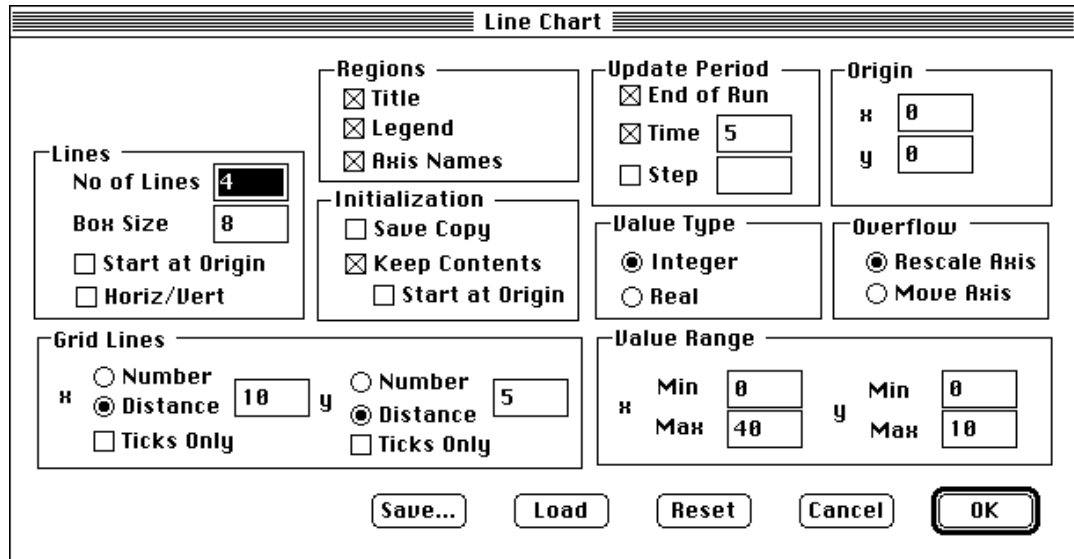
Resource Use Model: Line Charts

The following line chart was produced by the Resource Use Model at Time=30 during a typical execution. The chart shows the number of processes currently in `Processing Locally`, `Awaiting Resources`, and `Using Resources`; the number of resources currently available in `Resource Pool`; and the values that these quantities had at the end of every period of five time units since the simulation began.



Definition of the Line Chart

The **Chart Attributes** dialog for the above chart look like this:



The dialog box is titled "Line Chart" and contains several sections for configuring the chart's appearance and behavior. The "Lines" section has "No of Lines" set to 4 and "Box Size" set to 8, with checkboxes for "Start at Origin" and "Horiz/Uert". The "Regions" section has checkboxes for "Title", "Legend", and "Axis Names", all of which are checked. The "Initialization" section has checkboxes for "Save Copy", "Keep Contents", and "Start at Origin", with "Keep Contents" checked. The "Update Period" section has checkboxes for "End of Run", "Time", and "Step", with "Time" checked and set to 5. The "Origin" section has input fields for "x" and "y", both set to 0. The "Value Type" section has radio buttons for "Integer" (selected) and "Real". The "Overflow" section has radio buttons for "Rescale Axis" (selected) and "Move Axis". The "Grid Lines" section has radio buttons for "Number" and "Distance" for both "x" and "y" axes, with "Distance" selected for both, and checkboxes for "Ticks Only". The "Value Range" section has input fields for "Min" and "Max" for both "x" and "y" axes. At the bottom are buttons for "Save...", "Load", "Reset", "Cancel", and "OK".

The line chart's code segment looks like this:

```
action
LC_upd_chart{
lc = "Line Chart 0",
values = [
(1, time (), SV'sum isv_ProcLoc, true, true),
(2, time (), SV'sum isv_AwaitRes, true, true),
(3, time (), SV'sum isv_UseRes, true, true),
(4, time (), SV'sum isv_ResPool, true, true)];
```

As this segment shows, each update adds a new data point to each of the lines of the chart. The X value of each point is the time of the update, obtained by polling the simulator. The Y value of each point is the number of tokens in the place that the line describes. Each of these token counts is read from the appropriate statistical variable.

Analysis of the Line Chart Data

The line chart leaves little doubt: the computer is not efficiently using the resources that it does have, and processes spend a lot of time waiting for resources that do not exist. A single glance is enough to produce a verdict.

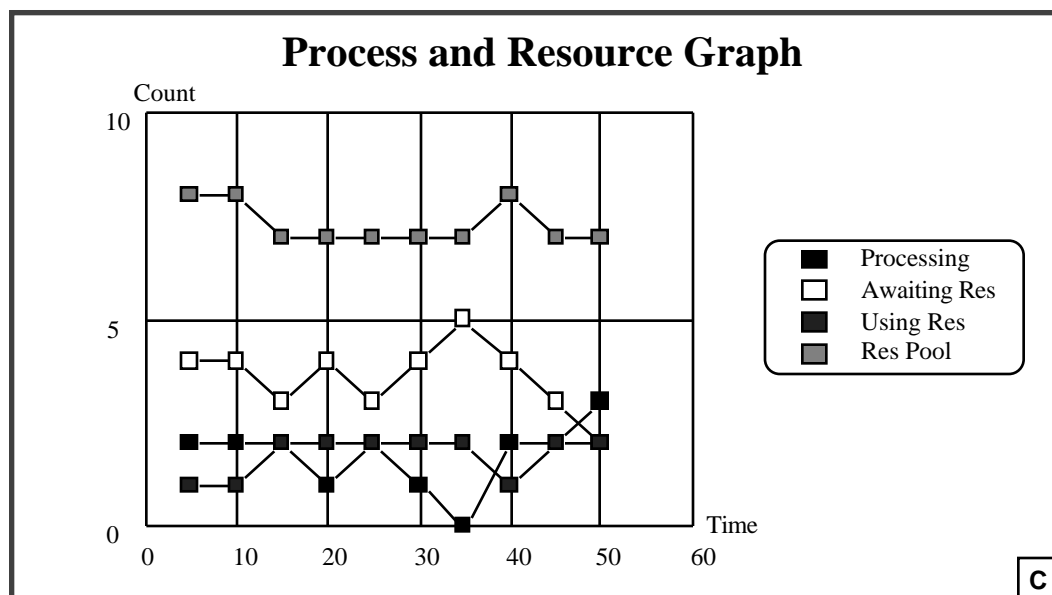
Is there any chance that the state this chart shows is an artifact? Perhaps the model's initial conditions were very far from its equilibrium state, so that a different picture would ultimately emerge if the simulation ran long enough. Perhaps an exceptionally unlucky

Design/CPN User's Guide

series of random choices during the simulation has produced a distorted picture.

To explore these possibilities, it would be convenient to be able to run the simulation for very long intervals, and display the results without having to lose previous information to make room for more recent data, as happened with the bar and history charts. Line charts allow such display by automatically readjusting their scale when they are given data that would otherwise fall outside the current display area.

For example, continuing the simulation shown in the above chart to Time=50 resulted in the following chart:



If the simulation had gone on longer, additional X-axis scale compressions would have occurred. Eventually the chart would become too compressed to read, but the increasing compression would not lead to an error. The problem of excessive compression could be handled, if necessary, by editing the chart to make it physically larger.

In any event, the additional simulation gives no sign of revealing anything new. Of course, no finite amount of simulation can absolutely guarantee that representative results have been obtained, but a picture like the above is certainly unambiguous enough to justify the conclusion that a problem exists and should be investigated.

Improving the Resource Use Model

There is no reasonable doubt that something is wrong with the computer that is represented in the Resource Use Model. But what is wrong, and what is the most efficient way to remedy it?

The Essential Problem

The first obvious conclusion is that there are simply not enough resources. But this does not explain why the resources that are present are used so little.

The reason can be seen by examining the way processes use resources. Due to the closely overlapping pattern of resource requirements, any process that is using resources will cause three others to wait until it is done, even though most of the resources needed by the waiting processes are idle.

Of course, this very inefficient pattern is artificial: it is unlikely that anything so regular would characterize the operations of a real computer. But the example is nevertheless realistic in that system performance sometimes varies dramatically with differing load patterns, giving rise to very nonlinear results as conditions change.

Possible Solutions

One obvious solution to improving the computer would be to provide two of each type of resource, or even three. Either of these solutions would reduce waiting, but neither is ideal because both provide more of some resources than is needed to meet even the heaviest demand. Only one process ever needs Resource 1 or 9, and only two need Resource 2 or 8. Providing resources in proportion to demand would be more economical than just increasing their number overall.

However, another factor is also operating: processes with higher index numbers go for longer times between requests for resources, and hold on to resources longer once they have them. Proportioning resources to the number of processes that use them may not be enough. It may also be necessary to take into account how long each resource is retained, on the average, once it has been allocated. Or perhaps the fact that some processes use resources longer is offset by the fact that they use them less frequently.

Seeking a Solution

The Resource Use Model is intentionally simple: it was chosen as a teaching tool rather than as a realistic example of the kind of system for which models are typically built. Yet even this simple model has

given rise to a question that cannot easily be answered just by thinking about it. What does the computer need?

It would be ideal if simulation could automatically generate guaranteed formulas for system improvement, but this is generally not possible. In order to discover what improvements an inadequate system needs, a cycle of hypothesis and experiment is typically necessary. Simulation cannot substitute for this cycle.

The purpose of simulation is to provide quantitative analyses of system performance, so that existing problems can be characterized, and to predict the effect on performance of proposed changes, so that experiments can be done inexpensively. Appropriate use of charts and statistical variables can do much to make the results of analyses and experiments accessible, and to present them in ways that are likely to lead to rapid convergence on a useful answer.

Chapter 17

Reference Summaries

Navigating a Diagram

You can move from object to object on a page, or from page to page in a diagram, using the mouse, and/or various Design/CPN commands.

Navigating Objects

Pages consist of objects. When you navigate a page, your movement is defined by the *occlusion order* on the page.

Occlusion Order on a Page

Objects on a page are *layered* or stacked in the order of creation; the most recently created objects are on the top of the stack. Design/CPN remembers the order in which objects are created. You can change the layering order with **Bring Forward (Makeup menu)**.

Layering Order for Connectors

Analogously, each page contains an ordered set of connectors. You can determine whether nodes are drawn on top of connectors, or vice versa, by means of a check box in **Interaction Options (Set menu)**.

Layering Order for Regions

Each object has an ordered set of regions which is often empty. Each set of regions is drawn immediately on top of the parent.

When a node becomes a region (by means of **Make Region (Aux menu)**), it is removed from the layering order for the nodes, and entered into the layering order for regions of the new parent object.

When a region becomes a node (by means of **Make Node (Aux menu)**), it is removed from the layering order for regions of its parent object, and is entered into the layering order for nodes.

Means of Navigation

You can navigate through a page or the hierarchy using one or more of the following Macintosh/X-Window System navigation techniques:

- arrow keys
- double-clicking
- commands
- mouse selection

Design/CPN makes extensive use of the arrow keys, double clicks, and commands to move on a page or in the hierarchy. Each of the features is listed and described in some detail.

Arrow Keys

The arrow keys can be used to select other objects or to move the text cursor.

Navigating in a Non-Text Mode

The arrow keys can be used to jump from one object to another on a page, according to their layering order:

- LEFT-ARROW moves you from the selected object to the previous object, in a node layering, a connector layering, or a region layering.
- RIGHT-ARROW moves you from the selected object to the next object in the same order, in a node layering, a connector layering, or a region layering.
- The DOWN-ARROW moves you to the first child, that is, the first object in the corresponding region layering.
- The UP-ARROW moves you to the parent.

The arrow keys cannot be used in certain conditions (for example, when you cannot move down because the selected object has no region).

Navigating in Text Mode (Not Group Mode)

No key: Move the text cursor.

OPTION: Layering hierarchy. The system stays in text mode and the entire text of the object becomes selected.

SHIFT: Text pointer hierarchy, used only in syntax error messages.

Double-Clicking

Double-clicking on certain objects with the mouse produces the following results:

Double-Clicking a Substitution Transition

Double-clicking a substitution transition on a related node opens the corresponding subpage and makes it active.

Double-Clicking a Port Place

Double-clicking a port place opens the superpage and makes it active. When there is more than one superpage, all of them are opened and one of them becomes active.

Double-Clicking a Page Node on the Hierarchy Page

Double-clicking a page node opens the corresponding page and makes it active.

Double-Clicking for Editor Regions

Double-clicking a key region changes the visibility and selectability of the corresponding popup region. When the popup region is visible, it becomes non-visible and non-selectable; if the popup region is non-visible, it becomes visible and selectable.

Double-clicking a selectable popup region makes the popup region non-visible and non-selectable.

Double-Clicking for Simulator Regions

The presence and display of a simulator popup region is governed by the region attribute of the (parent) key region. The popup can be shown (both key and popup will be updated and displayed during simulation), hidden (only the key region will be displayed), or missing (only the key region will be updated and displayed during

simulation). The default value of the popup attribute is specified in **Region Attributes (Set menu)** for the (parent) key region in the Editor and the Simulator.

In the Editor, double-clicking a simulator key region or a selectable popup region works exactly as described above for the editor regions — this operation does not influence the popup attribute of the actual key region.

In the Simulator, double-clicking a simulator key region changes the popup attribute as follows: hidden -> shown, missing -> shown, shown -> hidden. Double-clicking a popup region changes the popup attribute of the (parent) key region to hidden.

Commands

You can use Design/CPN commands to navigate a page or a hierarchy.

Parent Object, Child Object, Next Object and Previous Object Commands

These commands work in the same way as the corresponding arrow keys. They have the same modifier keys, which also work when the commands are called by means of user-defined keyboard equivalents.

Select Command

When objects are layered, closely spaced, or contained within other objects, they may be difficult to select with the graphic tool. The **Select** command (**Makeup** menu) allows you to access the object.

Graphical Objects

A *CPN model* is composed of *graphical objects*. The objects appear on the primary unit of a CPN model, called a *page*. This chapter presents the basic elements of a page in a CPN model, the classes into which the elements are grouped, and the types of objects within each class. Cross-references indicate where you can locate detailed information in the manual on specific topics.

Essential Graphical Objects

Nodes, regions, and connectors are *essential graphical objects* that is, these objects cannot be defined in terms of other graphical objects.

A *node* is the building block of a model. Nodes can be created in many shapes, such as boxes, rounded boxes, ellipses, polygons, wedges, and labels. In this manual, for the sake of brevity, a box-shaped node will be called a *box*. Note that the shape of a node is only one of its many attributes.

Other attributes of a node include its size, placement, pattern for interior filling, and border. Typically, a node appears with a border around it which can be assigned different graphical attributes.

A node may also contain text in a text field. A *label* is a node where the size is determined by its text contents. It usually has no visible borders.

Two nodes are linked by a *connector* (a line containing one or more segments and bending points). A connector may contain text and it may be assigned arrows and graphical attributes. Connectors always remain attached to their source and destination and are automatically redrawn when their connected nodes are moved.

Nodes linked by connectors form a graph called a *net*. Petri nets are a special type of net with two kinds of nodes called *places* and *transitions*, and a type of connector called an *arc*. Places can be connected to transitions by an arc. However, you cannot connect two nodes of the same type (for instance, a place with another place) in a Petri net.

An object (a node or a connector) may relate to a subordinate object (node, region, or connector) in an *object hierarchy*. A subordinate object, one level or more down in the object hierarchy, is called a *region*.

The first-level regions of an object are called *children* of the *parent* object while all regions (together with the children) are called *descendants*. This parent object is also called the *original parent object*. Changes to a parent can affect its descendants. For instance,

when you move a parent object, you also move its descendants; the latter preserve their positions with respect to the parent object.

An original parent object, such as a node or a connector, may sometimes have a long line of descendants. Notice that a region cannot be an original parent but all objects can have a chain of regions as their descendants.

Graphically, nodes and regions look similar, whereas connectors have a distinctly separate form. However, the system recognizes the distinction in the object hierarchy and indicates it as such in the status bar for the selected object.

Classes of Objects

CPN models consist of three major classes of objects: *CPN objects*, *auxiliary objects*, and *system objects*. CPN objects have formal meaning whereas auxiliary objects do not. System objects are system-defined and maintained and reflect important structures of the model.

Each of the three major classes of objects contain nodes, regions, and connectors. As a result, CPN objects, auxiliary objects, and system objects have corresponding nodes, regions, and connectors. For instance, CPN objects have CPN nodes; auxiliary objects have auxiliary connectors, and so on.

For the sake of precision, CPN objects and system objects are further subdivided into *object types* which are recognized by the system as such in the type field of the status bar.

The shape of an object does not determine its object type but only its graphical appearance.

CPN Objects and Object Types

The CPN objects consist of the following object types: places, transitions, arcs, CPN regions, and declaration nodes. Places and transitions are nodes. Technically, arcs are CPN connectors. (The terms “arc”, “place” and “transition” comply with the Petri net terminology and will be used as such throughout the manual.) All CPN objects have a formal meaning — that is, they describe as well as influence the behavior of the net.

Places, transitions, and arcs have CPN regions. (CPN regions are central to the tool and are designated as such to distinguish them from other regions.)

CPN objects also include three kinds of declaration nodes: *global*, *temporary*, and *local*.

Auxiliary Objects and Object Types

Nodes, connectors, and regions may be auxiliary objects, called *auxiliary nodes*, *auxiliary connectors*, and *auxiliary regions*, respectively. Auxiliary objects have no formal meaning — that is, they do not influence the behavior of the net. Consequently, they are not divided into object types but only by the essential graphical elements such as shapes etc. Auxiliary objects enhance the readability of a CPN model. They:

- create a better overview
- make the model neater
- show additional relationships between the objects
- add extra explanation
- provide user-coded animation

They may be compared to the comments added in a programming language.

CPN objects and auxiliary objects can have any number of auxiliary regions. Auxiliary objects *cannot* have CPN regions.

System Objects and Object Types

CPN models contain a third kind of object type called system objects. These objects are generated by the system and provide graphical feedback on system structures. For instance, an indicator generated by the system called a *key region* differentiates transitions that have hierarchical properties. Such key regions are used to mark special objects.

Other groups of objects generated by the system provide page information, simulation information, or a means for positioning the connectors' ends.

Individual Classes and Object Types

The object type determines the allowable types for regions of the object. Each of the following lists provides a short summary of the object type for each class of objects, a brief description of its corresponding regions, and a cross reference to sections or chapters with more details.

CPN objects can be classified into places, transitions, arcs, declaration nodes, chart nodes, and regions.

Place

A *place* is a CPN node, typically drawn as an ellipse. A place can have the following kinds of CPN regions:

- **Name:** An optional user-defined identification. Typically, a label. The text field of the place may be used instead.
- **Colorset:** A designated type of data residing in a place. Typically, a label. The syntax is described in CPN ML.
- **Initial Marking:** An optional specification of initial data for a place. Typically, a label. The syntax is described in CPN ML.
- **Hierarchy:** Typically, a box. If present, the node is called a substitution transition and the hierarchy region contains the details of the hierarchy relationship. The hierarchy region's text is maintained by the system and it has a system key region with text HS.
- **Fusion:** Typically, a box. If present, the place is a fusion set. The fusion region contains the name of the set. Its text is maintained by the system and it has a system key region with the text FP.
- **Port:** A box. If present, the place is a port type/exit node. Its text is maintained by the system. Contents of the port region indicate the type of port.

The term “typically” indicates the shape commonly used and provided in the standard defaults. You may use other shapes for these objects (see the commands **Change Shape** and **Convert to CPN** for more details).

Transition

A *transition* is a CPN node, typically drawn as an box. A transition can have the following kinds of CPN regions:

- **Name:** An optional user-defined identification. Typically, a label. The text field of the transition may be used instead.
- **Guard:** An optional region for restricting the transition. Typically, a label. It complies with the Standard CPN terminology. The syntax is described in CPN ML.
- **Code:** An object for specifying code for execution during simulation. Typically, a box. A code segment has a system key region. The syntax is described in CPN ML.

- **Time:** An object that allows you to specify delays for timed simulation. Typically, a label. The syntax is described in CPN ML.
- **Log:** An object that is written into by the code in the code segment during simulation. Typically, a box. A log has a system key region.
- **Hierarchy:** Typically, a box. If present, a substitution transition. The hierarchy region contains the details of the hierarchy relationship and its text is maintained by the system.

Arc

An *arc* is another name for the CPN connector between a place and a transition. The arc's direction is determined by the position of the arrowhead, and not by the order in which the two nodes were selected when the arc was constructed. Two nodes may be connected by more than one arc.

Arcs can also be undirected (with no arrowheads) or bidirected (with two arrowheads). Undirected and bidirected arcs are equivalent. They are both a shortcut for two arcs — having opposite direction and the same arc inscription.

An arc has the following CPN region:

- **Arc inscription:** An object providing details of the token flow. Typically, a label.

Declaration Nodes

CPN *declaration nodes* contain declarations such as colorsets and variable declarations. The three types of declaration nodes are global, local, and temporary nodes. Declaration nodes cannot have CPN regions.

Chart Nodes

A *chart* is a CPN node. It can have the following region:

- **Chart code segment:** An object for specifying chart management code for execution during simulation. Typically, a box. A chart region has a system key region. The syntax is described in CPN ML.

Regions

CPN regions relate to their parent objects. Note that CPN regions can have auxiliary regions, but they *cannot* have other CPN regions as their descendants. However, when an object has a key region with a CPN region as its descendant, the object is regarded as the true parent of the CPN region; the system key region only performs the function of a “stand-in” parent.

Auxiliary Object Types

Unlike CPN objects and system objects, auxiliary objects are not governed by formal rules. As a result, auxiliary nodes can be placed anywhere in a model. All classes of objects can be assigned auxiliary regions. Auxiliary connectors can link any two nodes or regions.

System Object Types

System objects types are system nodes, system connectors, and system regions. The system generates system regions both in the Editor and the Simulator.

System Nodes

The system node is as follows:

- **Page nodes:** Represent pages in a model.

System Connectors

The system connector is as follows:

- **Substitution connectors:** Represent substitution relationships.

System Regions

The system regions are as follows:

- **Substitution tags:** Supply the name of the substitution transition corresponding to the substitution connector.
- **Mode regions:** Record the mode attributes or setting for the Simulator.
- **Current marking regions:** Show the state of simulation data.

- **Transition feedback regions:** Highlight transitions during simulation.
- **Input token regions:** Show the effects of simulating a step for an arc.
- **Output token regions :** Show the effects of simulating a step for an arc.
- **Binding regions :** Indicate bindings included for the next step.

Two system regions enjoy special status in the tool. They are:

- **Key regions:** Indicate additional model information.
- **Endpoint regions:** Optional feature for positioning the connectors' endpoints.

Special Regions

Key, popup, and endpoint regions perform non-trivial functions in the model. They therefore require special mention.

Key and Popup Regions

A *key region* is a graphical object that indicates when a parent object has additional model information associated with it. A key region has a *popup region* as its region. Key regions provide access to information in the popup region. They function, in some respects, like a key in a lock as well as resembling the key on a keyboard — they “lock” and “unlock” the information in the popup region.

A key region is typically drawn as a small solid circle/square, containing a number, a single capital letter, or two capital letters. The key region is always a region of a place, a transition, or a page node. As in the case of other objects, defaults govern the attributes assigned to key regions at creation. Key regions and popup regions contain many types of information, such as hierarchy information, code segments, and markings.

Design/CPN makes a distinction between editor regions and simulator regions. Editor regions are created and changed during editing to reflect the net structure, but remain unaltered during simulation. Simulator regions are created when switching to the Simulator — if not already present — and their contents are updated during simulation to reflect the current state.

Simulation key regions have a special attribute for dictating the presence and absence of popup regions. The attribute's defaults determine whether a popup region is created for a key region.

In the simulator, double-clicking on a key region without a popup region creates a popup region and changes the attribute accordingly.

All key regions are system regions. However, popup regions are of two types: those that are CPN regions and those that are merely system regions. Popup regions for hierarchy information, fusion information, port information, code segments and log information are CPN regions. The status bar displays the name of the selected key or popup region.

When we talk about a hierarchy region, we always mean the popup region — and not the key region — which is called a hierarchy key region. Analogously, for the other kinds of key/popup regions.

The editor and simulator key/popup regions and their corresponding parent nodes are listed below.

Editor (Key/Popup) Regions

These are as follows:

- Hierarchy information (region of a transition).
- Fusion information (region of a place).
- Port information (region of a place).
- Code segment (region of a transition or a chart).
- Mode information (region of a page node).
- Log information (region of a transition).

Simulator (Key/Popup) Regions

These are as follows:

- Current marking (region of a place).
- Input token (region of an arc).
- Output token (region of an arc).

Creating And Deleting Key/Popup Regions

The system generates the key and popup regions, except for code and log regions.

Editor key/popup regions are always created as a pair — that is, a key region with a corresponding popup region (which may initially be invisible depending on the defaults). However, as an exception to this rule, you can create code segments and log regions, while the system adds the corresponding key regions. When you delete one of the pair, the system automatically deletes the corresponding region.

Simulator key regions are created when you enter the Simulator, if they do not already exist. Some of these regions are displayed during simulation if defined in **Substep Options**, other regions are always displayed. Simulator popup regions are created according to the defaults for the corresponding key region. When a simulator popup region is deleted, the popup attribute of the parent key region is changed to missing. Simulator key regions can only be deleted in the Editor.

Editing Key/Popup Regions

You can define the default attributes — that is, the border, fill, font, shape, etc. — for each kind of key/popup region using the attribute commands in the **Set** menu.

You cannot accidentally change the size of the existing key regions as preset in the standard defaults. This restriction is imposed in order to prevent the key regions from being automatically resized when their parent objects are resized. Setting the **Region Attributes** allows you to then change the size of the key region.

Endpoint Regions

Endpoint regions allow you to customize the placement of connectors between two nodes. Traditionally, the position of a connector is determined from the center of one node to the center of the other node. The points where the two ends of the connectors meet the borders of the nodes are system determined. You can change the position of the connector, using its endpoint regions.

Connectors can have endpoint regions at one or both ends. When the connector has an endpoint region, the position of the connector is determined from the center of one endpoint region to another, if any. Typically, since the endpoint regions are located on the border of a node, the position of the connector is determined from border to border.

Endpoint regions are regions of a node. Hence, they can be placed outside the nodes they are connecting, even if this is not common practice. By default, an endpoint region is “hidden”; it is not printed.

Creating Endpoint Regions

Endpoint regions are not created automatically by the system. They are created by invoking **Drag** in the **Makeup** menu, which then automatically creates the endpoint region.

Alternatively, you can create or move endpoint regions by selecting the corresponding connector handle with the SHIFT key depressed, which puts the system in a drag mode. The endpoint region is created before the drag, near the border. When the drag is completed, a new endpoint position is calculated (in the same way as for a new connector). The endpoint region will be positioned at the border of the node near the cursor.

Endpoints regions are system regions and when selected, are indicated as such in the status bar.

Preferences

Design/CPN's defaults consist of the preferred attributes and options. You can alter the appearance and behavior of a graphical object or a model by changing its attributes and options. Names and numbers are tags of identification for objects and are determined by the user.

Attributes and Options

Attributes are characteristics of an object. You can set or alter the attributes to change the appearance or the behavior of your model. Attributes may be assigned to individual objects or pages. Characteristics assigned to the model as a whole are called *options*.

Object Attributes

Object attributes are characteristics of nodes, regions, and connectors. Hence, CPN objects, auxiliary objects, and system objects have object attributes. Object attributes are divided into five disjoint sets; the first four sets appear as commands in the **Set** menu.

- **Graphical attributes:** Line (thickness, color, pattern), fill (color, pattern), text (color), visibility, selectability (see Visibility and Selectability below), and layering logic.
- **Text attributes:** Font, style, size, justification and text scroll bars.
- **Shape attributes:** Size, orientation, and shape (for connectors).
- **Region attributes:** Relative offset from parent center, with color and resizing dependent upon the parent object (only for regions). In addition, simulator key regions have popup attributes.

Page Attributes

Pages have *page attributes*:

- **Page kind:** Specifies whether a page is a standard or a palette page.
- **Visibility of page border:** Specifies whether the page border is displayed.
- **Page size:** Specifies the boundary rectangle of the page. When the drawing exceeds the boundary, the page should be

scaled or reduced to the standard Page Setup requirements for printing.

The page attributes are accessible under **Page Attributes** in the **Set** menu.

Mode Attributes

Pages and substitution transitions have mode attributes. Mode attributes regulate the way in which pages are treated for simulation. The mode attributes are:

- **Prime page:** Specifies whether a page is a starting page called a *prime* page; *multiplicity* of prime page implies the number of prime instances for a prime page.
- **Included:** Specifies whether page instances are included
- **Observable:** Specifies whether pages instances are observable
- **Proposed occurrence sets:** Specifies whether page instances are proposed to participate in occurrence sets
- **Interactive:** Specifies whether pages instances participate in interactive runs.
- **Code segments:** Specifies whether pages instances are permitted execution with code segments

The mode attributes are accessible in **Mode Attributes (Set menu)**.

Special Attributes

Visibility and selectability are graphical attributes. Unlike the other attributes in this group, they require a more detailed description of their functions.

Graphical Attributes: Visibility and Selectability

Objects may be visible or non-visible and they may be selectable or non-selectable.

Visibility

Each object has a graphic attribute, which determines whether the object is visible or not. Non-visible objects are not deleted.

Although invisible, they continue to influence the behavior of the CP-net, if they are CPN objects.

Changing Visibility

A region's substructure can be hidden by means of **Hide Regions** (**Makeup** menu) and **Show Regions** (**Makeup** menu)

The attributes of visibility for popup regions can be changed by double-clicking on the corresponding key region or the popup region itself, when invisible.

Selectability

Each object has a graphic attribute that determines whether the object is *selectable* or not.

When an object is non-selectable it cannot be selected in the usual ways, for example by clicking with the mouse. It can, however, be selected in the following ways:

- By means of **Select All Nodes**, **Select All Regions**, and **Select All Connectors** (**Makeup** menu) commands when **OPTION** is pressed.
- By means of the **Child**, **Parent**, **Next Object**, and **Previous Object** (**Makeup** menu) commands.

Changing Selectability

Selectability is changed by means of **Graphic Attributes** (**Set** menu).

Options

A large number of properties relate to the entire model. Such properties are called *options*. The model options in the Editor are accessed through the **Set** menu commands **Hierarchy Page Options**, **Interaction Options**, **Merge Options**, **Text Options**, **Syntax Options** and **ML Configuration Options**. The model options in the Simulator are accessed through the **Set** menu commands **Simulation Code Options**, **General Simulation Options**, **Interactive Simulation Options**, **Occurrence Set Options**, and **Transition Feedback Options**.

Diagram and System Defaults

The two kinds of defaults are

- *System defaults*, which determine the diagram defaults for new diagrams.
- *Diagram defaults*, which determine the attributes of new pages and objects within a diagram.

System Defaults

System defaults determine diagram defaults for new diagrams. Each kind of object (that is, places, arcs, code segments, auxiliary boxes, and auxiliary polygons) has its own set of system defaults for its attributes. Moreover, a system default exists for each option.

Your copy of the CPN Editor has a set of system defaults proposed by the designers of Design/CPN in the CPN Settings file that accompanies your application (see below). These defaults are called the *standard defaults*.

CPN Settings

System defaults are recorded in a file called *CPN Settings* in the system folder on the Macintosh. This file is used each time Design/CPN has to save or load system defaults; if the file cannot be found, a new one is generated with the crude, hardcoded defaults. This means that you can change the system defaults back to the standard defaults by replacing the current CPN Settings file with the original one you received with the application. If you delete, rename, or move the current CPN Settings file, the system will generate a new one, but it will not contain the standard defaults.

System defaults are recorded in a file called CPN Settings in the home directory.

Diagram Defaults

The *diagram defaults* determine the default attributes of new objects and new pages.

When a new diagram is created, the diagram defaults are copied from the system defaults. Later changes to the system defaults do *not* change the diagram defaults for existing diagrams.

Each kind of object (i.e., places, arcs, code segments, connectors, polygons) has its own set of diagram defaults for its attributes. There are, however, *no* diagram defaults for options. The reason is

that options relate to the entire diagram and thus there is no difference between an option default and a current option value.

The diagram defaults are saved together with the diagram and they will automatically be in effect the next time the diagram is opened.

Changing System And Diagram Defaults

You can modify defaults by means of the **Save** button or by checking **Save Default** in the appropriate dialogs for commands relating to attributes and options.

When the **Save** button is invoked, a small dialog appears, prompting you to choose whether you want to save the current settings of the dialog as system defaults and/or as diagram defaults (options can only be saved as system defaults).

Using System And Diagram Defaults

Defaults can be loaded into a dialog by means of the **Load** button, which appears in the dialogs of some commands dealing with attributes or options. When this button is clicked, a small dialog appears and you choose whether you want to load the system defaults or the diagram defaults. (For options, you can only load system defaults; hence, there is no dialog.)

Loading a set of defaults changes the setting of a dialog. It does not change the attributes of any objects or pages or change any option until you click the **OK** button.

Copying Defaults

You can copy the entire set of system defaults into the diagram defaults of the current diagram. You can also copy the diagram defaults of the current diagram into the system defaults. Both operations are performed by means of **Copy Defaults** in the **Set** menu.

To reproduce the original CPN Settings file, use **Copy Defaults** to copy the diagram defaults from the Defaults diagram, provided in the release, into the system defaults. Alternatively, reinstall the CPN Settings file, provided in the release.

Names and Numbers

A *number* is a positive integer, while a *name* is a sequence of characters.

Using Names and Numbers

When you refer to a page, place or transition, you use names or numbers (for example, in code segments). Moreover, the system uses names and numbers as part of the feedback (for example, in hierarchy regions).

Names should be unique when they refer to a page, place, or transition in order to avoid confusion; this is not necessary when names provide system feedback (although non-uniqueness may create some confusion).

Different Kinds of Names

Design/CPN communicates the identity of certain kinds of objects to you by means of names and numbers. The names and numbers are automatically generated from text strings you enter. The four different kinds of names are *page names*, *place names*, *transition names* and *fusion names*, while there is only one kind of number: *page number*.

Syntax for Names

Each name is a sequence of at most eight characters, and these may not be whitespace characters (for example, created by SPACEBAR, TAB, or RETURN) or the # character (which is used for signaling the beginning of a number).

The feedback from the system separates and surrounds names by certain characters. Do not include the following characters in names if you wish to receive readable feedback:

? * @ : -> () [] { }

Additionally, you may want to comply with the standard for ML identifiers which start with a letter.

Syntax for Numbers

Each number is a sequence of at most 5 digits.

Generating Names

Each name is generated from a text string that you enter. The system skips any leading format characters. It then takes the succeeding characters until one of four conditions is met: eight characters have been taken, the end of the text string, a format character, or a # character has been reached.

Page names, place names and transition names (but not fusion names) may be identical to the empty text string (which the system displays as three stars).

Entering and Changing Numbers and Names

Page names and numbers appear in page nodes. They can be changed by means of **Page Attributes** (**Set** menu).

Page names and page numbers are used in different kinds of system feedback, that is, in window title bars and hierarchy regions.

Entering and Changing Place Names and Transition Names

Place names and transition names are usually generated from the name region of the place or transition. When the name region is missing, the name is generated from the text of the place or transition. You cannot change a name using a command. A name can be changed by typing in the text field or the name region of the CPN node.

Place and transition names are used in different kinds of system feedback, such as hierarchy regions and instance names.

Fusion Names

Fusion names are entered by the user in the **Fusion Place** dialog. Fusion names are used in different kinds of system feedback, for example, in fusion regions.

Fusion names have a formal meaning when you define a fusion set. This is necessary — at least for global fusion sets — where it is not possible to form a group containing objects from several pages.

Fusion set names must be non-empty. A name can designate only one fusion set on a page. It is permissible to use the same name for two non-global fusion sets if they belong to two different pages.

List of Compulsory Syntax Restrictions

A number of compulsory syntax restrictions are checked by the system when **Syntax Check** is invoked or the CPN Simulator is entered.

Format for Syntax Violations

Each displayed syntax violation has three sections:

- The first section contains a number and a short description of the violated syntax restriction.
- The second section is a description of the restriction.
- The third section indicates where the restriction was violated.

C.1 Missing global declaration node

Each diagram must contain one global declaration node.

C.2 Too many global/temporary declaration nodes

Each diagram must contain one global declaration node and may contain one temporary declaration node.

<<Pointer to Declaration Node>>

C.3 Too many local declaration nodes

Each page is allowed to contain at most one local declaration node.

<<Pointer to Declaration Node>>

C.4 Error in global/temporary declaration node

An error was found in a global/temporary declaration node.

<<Pointer to Declaration Node>>

C.5 Error in local declaration node

An error was found in a local declaration node.

<<Pointer to Declaration Node>>

C.6 Place must have a color set

Each place must have a non-empty colorset region.

<<Pointer to Place>>

C.7 Color set region must be legal

All colorset regions must be legal, that is, they must contain the name of a colorset declared in a global declaration box.

<<Pointer to Place>>

C.8 Initial marking must be legal

All initial markings must be legal, that is, they must contain a multi-set or a single value (of the correct colorset).

<<Pointer to Place>>

C.9 Guard region must be legal

All guard regions must be legal, that is, contain a boolean expression or a list of boolean expressions.

<<Pointer to Transition>>

C.10 Code segment must be legal

All code segments must be legal, that is, they must contain a code guard, input pattern, output pattern and a code action (of which some may be missing).

<<Pointer to Transition>>

C.11 Arc expression must be legal

All arc inscriptions must be legal, that is, evaluate to a multi-set or a single value (of the correct colorset).

<<Pointer to Arc>>

C.12 CPN variables in code segment

All code segments must use CPN variables in a legal way. For example, they must all occur on the surrounding arcs and none of them are allowed to be both in the input pattern and the output pattern.

<<Pointer to Transition>>

C.13 Code guard must be legal

A code guard must evaluate to a boolean expression or a list of boolean expressions.

<<Pointer to Transition>>

C.14 Code action must be legal

All code segments must have a legal code action which evaluates to the type determined by the output pattern.

<<Pointer to Transition>>

C.15 Color set in port assignment

All socket places must have the same prime colorset as the corresponding port place.

<<Pointer to Socket/Param>>

C.16 Color set in fusion set

All places in a fusion set must have the same prime colorset.

<<Pointer to Place>>

C.17 Initial marking in fusion set

All places in a fusion set must have the same initial marking (textually).

<<Pointer to Place>>

C.18 Socket must be assigned

Each socket node must be assigned to a port place.

<<Pointer to Socket>>

C.19 Port type must match socket

Each socket node must be assigned to a port place with a matching port type.

<<Pointer to Socket/Param>>

C.20 Time region must be legal

Each time region must start with “@+”, followed by an expression of the actual time type.

<<Pointer to Transition>>

PART 2

Design/CPN Menu Reference

Chapter 18

Overview of Design/CPN Menus

Part 2 of the Design/CPN Reference Manual lists and describes each Design/CPN menu and its individual commands. This chapter contains an overview of the Design/CPN menus. The rest of Part 2 contains a chapter for each menu.

File Menu

File menu commands affect entire diagrams. They create new diagrams, open existing ones, as well as saving closing, previewing, and printing diagrams.

The current page can also be saved as a subdiagram which can then be loaded and opened from within a diagram.

Edit Menu

Edit menu commands cut, copy, and paste graphical objects and text, and provide undoing and redoing of some graphical operations.

CPN Menu

The **CPN** menu is available only in the editor. In the simulator, it is replaced by the **Sim** menu.

CPN menu commands create CPN objects — places, transitions, arcs, CPN regions, declaration nodes, and charts. Other commands in the menu form hierarchies by creating or relating pages.

The menu also contains commands for syntax checking the diagram and removing the simulation regions that participate in simulation.

Sim Menu

Sim menu is available only in the simulator. In the editor, it is replaced by the **CPN** menu.

Sim menu commands prepare the diagram for simulation. They also regulate the simulation cycle, switch between page instances, or return the diagram to its initial state.

Aux Menu

Aux menu commands draw graphic objects (nodes and connectors) that have no significance as CP net components. They also start and stop the ML interpreter, and allow evaluation of individual ML statements.

Set Menu

Set menu commands specify object attributes, diagram options, and simulation options.

Makeup Menu

Makeup menu commands manipulate and alter graphic objects.

Page Menu

Page menu commands affect the appearance, labeling, and page numbering of the diagram. Other commands move around the pages and hierarchies of the diagram via the page hierarchy window.

Group Menu

Group menu commands group objects according to their object types. The objects within the group can be changed or manipulated as a single unit. A group is indicated by a gray outline around all its member nodes that are not regions.

Text Menu

Text menu commands manipulate text objects, permitting the creation, editing, formatting, selecting, and searching of text.

Align Menu

Align menu commands reposition the currently selected node or group by aligning it to one or two reference objects.

Chapter 19

File Menu Commands

New Open... Close Save Save As... Revert	Alt+O
Page Preview... Page Setup... Print...	Alt+P
Save Subdiagram... Load Subdiagram... Save Text... Load Text...	
Load IDEF...	
Save State... Load State...	
Enter Editor/Simulator Quit	Alt+Q

The **File** menu commands affect entire diagrams. They create new diagrams, open existing ones, as well as saving closing, previewing, and printing diagrams.

The current page can also be saved as a sub-diagram which can then be loaded and opened from within a diagram.

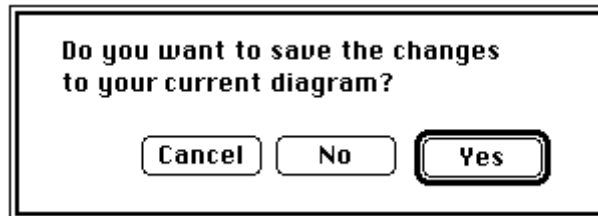
Only one diagram can be open at a time.

Design/CPN Menu Reference

New

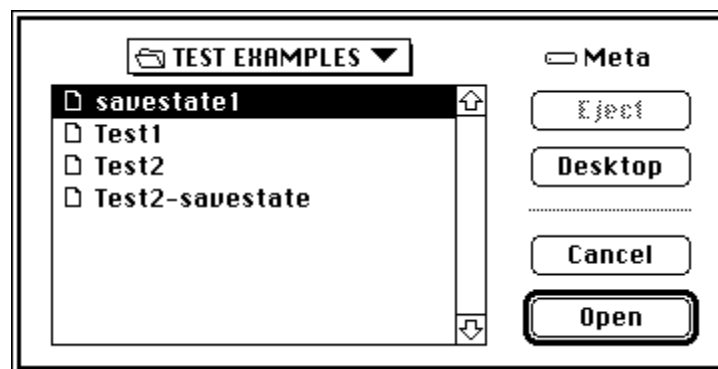
Creates a new, untitled CPN diagram with its hierarchy page. The diagram defaults are copied from the system defaults.

If a diagram is open, **New** displays the following dialog:



Open

Opens an existing diagram or saved state:

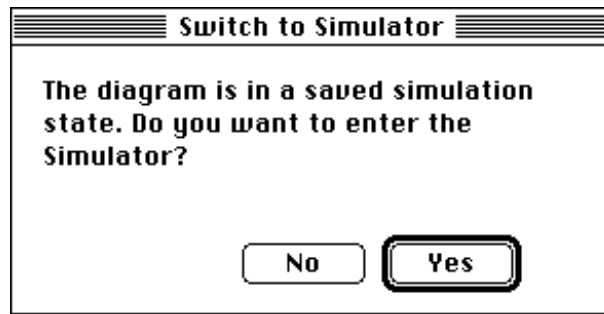


The pages are placed on the desktop in the form in which they were saved.

Diagrams created with earlier versions are converted to the current version when read.

Saved States

When opening a saved state, **Open** displays the following dialog:



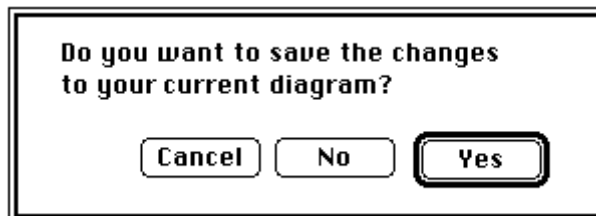
Clicking **Yes** loads the saved state and enters the Simulator.

Clicking **No** does not load the saved state. The system remains in the Editor.

Close

Closes a diagram. The name of the current document is listed in the menu next to the **Close** command.

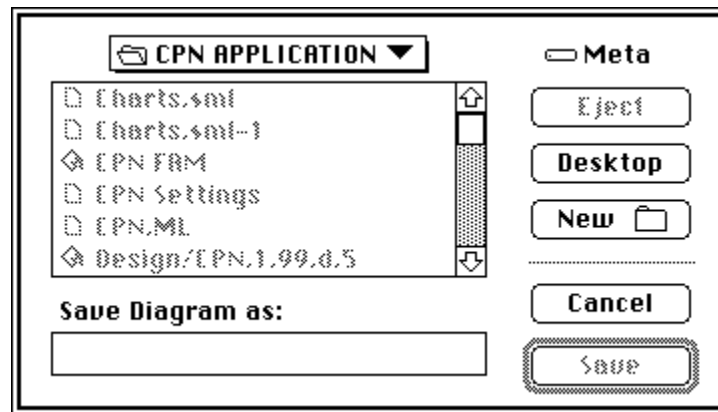
Close displays the same dialog as does **New** (**File** menu) if there have been any changes to the file:



Save

Saves a diagram on disk. If the current document is untitled, **Save** prompts for a name and location:

Design/CPN Menu Reference



The diagram is saved in two files if it has never been successfully syntax checked, or three files if it has. The two files are:

- A *diagram file*. This contains graphical information about the diagram. It has the name specified in the dialog.
- A *database file*. This contains the Petri Net representation of the diagram. It has the specified name with the extension “.DB”.

A saved diagram always has a diagram and a database file. If the diagram has been syntax checked successfully, it also has:

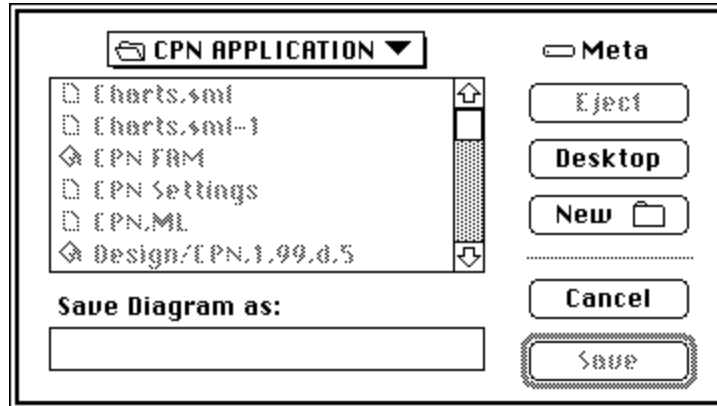
- An *ML file*. This contains the image produced during the syntax check, plus additional information if the diagram was switched to the simulator after a successful check. The ML file has the name specified in the dialog, with the extension “.ML”.

The three files must be kept in the same directory. If the saved diagram includes an ML file, you will be asked to specify (or verify) the file name. No path name can be specified.

If a diagram is new and untitled, the **Save** command is disabled. Use **Save As** to save the file.

Save As

Saves a diagram with a new name:

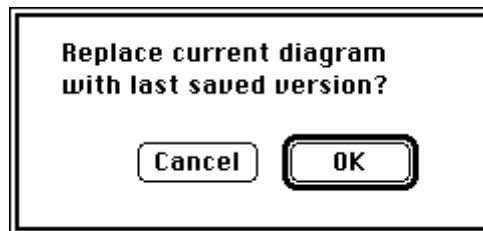


Saving under a new name is also a good way of making backup copies, since the original file is not removed. All files are saved in Design/CPN diagram format.

As with **Save (File menu)**, the .ML filename must be specified for a new diagram, or verified for an existing one.

Revert

Replaces the current diagram with the last saved version of that diagram:



Revert is available only if there have been changes since the last save.

Page Preview

Previews a diagram (or part of a diagram) before printing by displaying the current page as it appears when printed.

The **Page Preview** window does not display the effects of the master page or the **Page Setup** (**File** menu) options **Tiling**, **Scale to Fit**, **Omit Page Borders**, and **Print Hierarchy Page**.

Page Setup

Specifies how the pages of a diagram appear when printed. When a new diagram is created, the page setup options are copied from the previous diagram (or from the system defaults, if no previous diagram exists).

Most of the **Page Setup** dialog is specific to the printer that your system uses. For details see the documentation for your printer. The dialog also contains three components that are specific to Design/CPN:

- Output Form
- Omit Page Borders
- Print Hierarchy Page

Output Form

Specifies the number of pieces of paper each diagram page prints on:

Normal: The diagram page prints on one piece of the selected paper size. Any part of the diagram that doesn't fit is truncated.

Scale to Fit: Reduces the diagram if necessary so that every object will fit on the selected paper size.

Tile: Everything within the diagram page prints on the selected paper size, at the selected scale, on as many pages as necessary. The diagram prints according to the paper orientation setting, beginning with the top left corner.

Tile Overlap: Prints the diagram in the same way as **Tile**, but extends the printing on each piece of paper so that edges of the diagram sections overlap slightly.

Omit Page Borders

Page border lines, which are visible on the screen, do not print.

Master pages with visible borders cause the borders to print even if **Omit Page Borders** is checked.

Print Hierarchy Page

Prints the hierarchy page. The hierarchy page also prints if it is included in the pages selected in **Print** (**File** menu).

Print

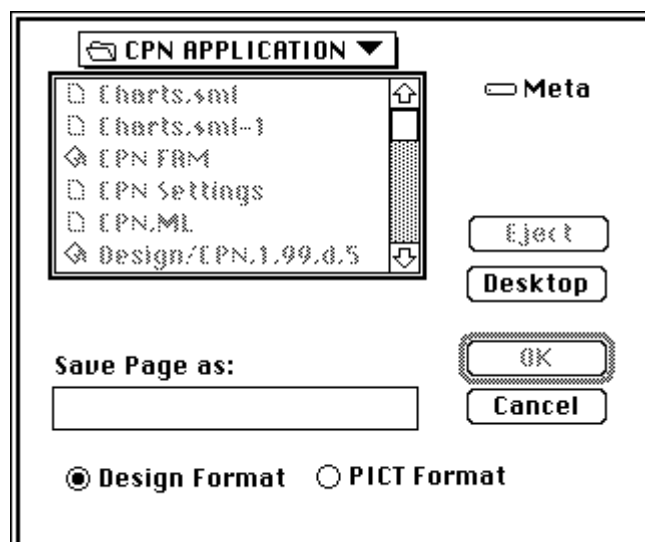
Prints all or part of a diagram according to **Page Setup** (**File** menu) instructions. **Print** uses printer-specific dialogs.

Select the number of copies to print and the page range in the dialog invoked by this command. Other selections vary according to the printer. Selecting **All** means all pages with a page number between 1 and 9999. The page scale influences the printing.

The master page in the document superimposes its imprint on every page. (See Master Page under **Page Setup** (**File** menu) for more information.)

Save Subdiagram

Saves the current page as a separate subdiagram:



Design/CPN Menu Reference

The subdiagram icon appears on the desktop, but can only be opened from within the Design/CPN application by **Load Subdiagram** (**File** menu). For example, to use a Design/CPN submodel in other diagrams, use this command to save it and the **Load Subdiagram** command to open it.

The **Save Subdiagram/Load Subdiagram** sequence preserves the exact appearance and formal meaning of the current page, with the following changes:

- All substitution transitions become ordinary transitions.
- All fusion places become ordinary places.
- All port places become general places.
- All declaration nodes are misinterpreted as transitions. These can be fixed by using **Convert to Aux** (**Aux** menu) and **Convert to CPN** (**Aux** menu).

Warning: CPN nodes that have the shape of a rectangle are recognized as transitions. CPN nodes that have the shape of an ellipse are recognized as places. If these defaults have been changed the subdiagram will not be loaded properly.

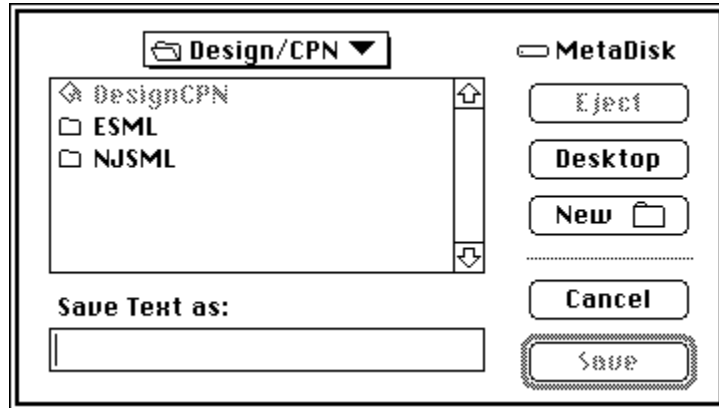
Load Subdiagram

Adds a previously saved subdiagram to any open Design/CPN diagram.

Once opened, this subdiagram becomes part of the current diagram. Use this command to make non-hierarchical submodels available across diagrams.

Save Text

Saves the text in the selected object:



The saved text can be loaded into a different Design/CPN diagram, using **Load Text** (**File** menu), as well as being edited with a standard text editor.

Restrictions

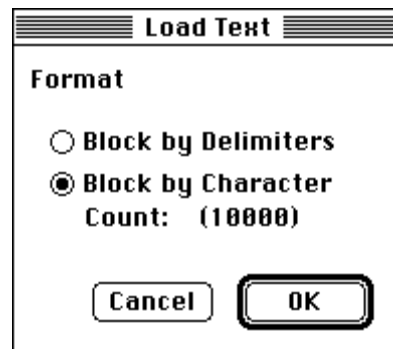
The command is not implemented for groups.

Load Text

Loads the contents of a text file into one or more box-shaped auxiliary nodes and placed on the current page. The status bar displays the number of characters in the file.

Blocking Options

Load prompts for a blocking option:



Design/CPN Menu Reference

Blocking by Delimiters

Each unit of text enclosed by matching delimiters is put in a node and placed on the page.

The minimum number of characters per node is 30, unless the default is changed by **Text Options** (**Set** menu).

Text within nested pairs of delimiters is placed in a node one level lower than the text enclosing it.

The top-level node always contains a text pointer to lower-level text boxes. For traveling along text pointers, see **Syntax Check** (**CPN** menu).

Blocking by Character Count

Text is divided into nodes according to the number of characters (including blanks) indicated in parentheses.

If the text length exceeds the count, more nodes are created and the top-level node contains a text pointer to its subordinate nodes.

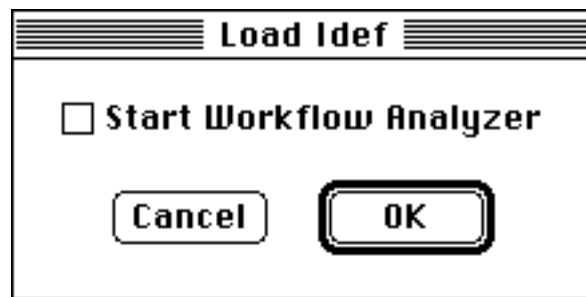
For traveling along text pointers, see **Syntax Check** (**CPN** menu).

The program default block size of 10000 characters may be changed in **Text Options** (**Set** menu).

Load IDEF

Loads a model exported by Design/IDEF (version 2.5 or later) for conversion to a CPN, either manually or by the Workflow Analyzer.

When **Load IDEF** is invoked, a dialog appears:



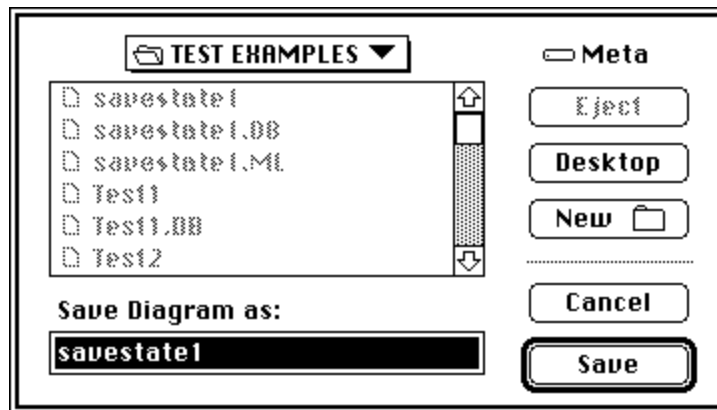
If the model is intended for conversion to CPN by the Workflow Analyzer, choose **Start Workflow Analyzer**, then click **OK**. If the model is intended for manual conversion to CPN, just click **OK**.

The **Open File** dialog appears, allowing you to indicate the file containing the model that is to be loaded. If there is already a diagram open in Design/CPN, it is augmented with the model in the file. Loading the file without an open diagram results in a new diagram.

For further details on generation of the file to load, consult *The Design/IDEF User's Manual*. For further information on the Workflow Analyzer, see *The Workflow Analyzer Tutorial*.

Save State

Saves a diagram and its current state from within the Simulator:



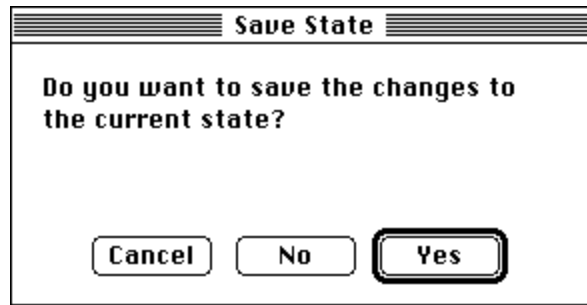
The saved state may be reopened from the Editor with **Open** (File menu) or from the Simulator with **Load State** (File menu).

Load State

Loads a diagram saved with **Save State** (File menu), permitting execution to start from the point that **Save State** was invoked.

If there is a diagram currently open whose state has either not been saved or has changed since the last **Save State**, the system displays the following dialog:

Design/CPN Menu Reference

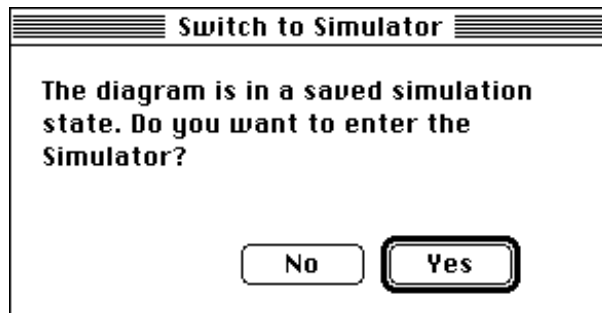


Clicking **Yes** causes the system to display the **Save As (File menu)** dialog before loading the saved state.

Clicking **No** causes the system to load the saved state, overwriting the current diagram.

Loading Saved States With Open (File Menu)

Open displays the following dialog:



Clicking **Yes** loads the saved state and enters the Simulator.

Clicking **No** does not load the saved state. The system remains in the Editor.

Restrictions

Load State is available only in the Simulator.

Enter Editor

Leaves the Simulator and enters the Editor. **Enter Editor** is available only in the Simulator. In the Editor, **Enter Simulator** appears in place in the **File** menu in place of **Enter Editor**.

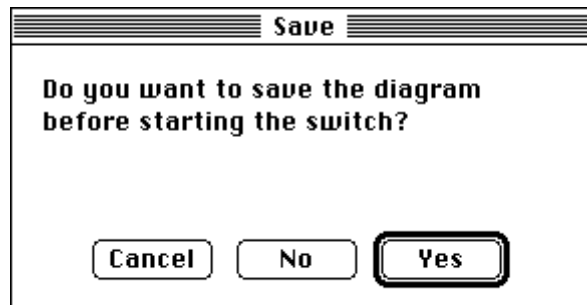
Enter Simulator

Leaves the Editor and enters the Simulator. **Enter Simulator** is available only in the Editor. In the Simulator, **Enter Editor** appears in the **File** menu in place of **Enter Simulator**.

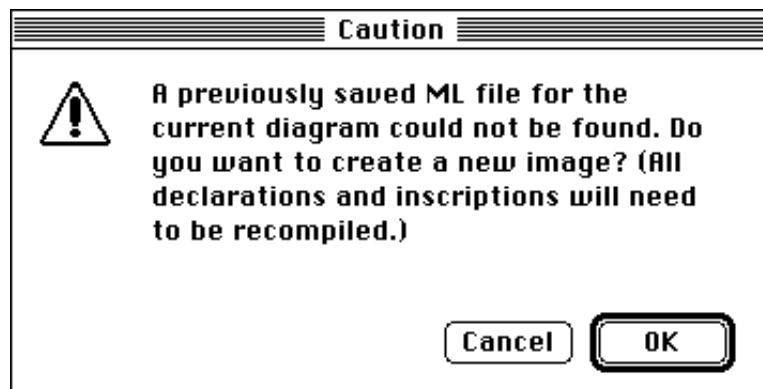
If you see a dialog that mentions a problem of any kind, see Appendix B, “Troubleshooting.”

Enter Simulator syntax checks the diagram and, if successful, generates the code necessary to run the simulation. This process is called a *switch*. Both the syntax check and the remaining part of the switch are incremental; the system only rechecks and reswitches those parts of the diagram that have been changed since the last syntax check/switch.

Before the switch begins, the system prompts:



After the syntax check is finished, the system tries to open the ML file for the diagram. If the diagram previously had an ML file, but the file no longer exists, the system prompts:



As the switch proceeds, the status bar displays the number of objects that have been checked or switched so far.

Quit

Leaves Design/CPN, closing the current document before quitting the system.

You can quit from either the editor or the simulator. If you quit from the editor, **Quit** displays the same dialog as does **New (File menu)** if there have been any changes to the file. If you quit from the simulator, **Quit** displays the same dialog as does **Save State (File menu)**.

Chapter 20

Edit Menu Commands

Undo Redo	Alt+Z
Cut Copy Paste Clear	Alt+X Alt+C Alt+V
Get Info	Alt+I

The **Edit** menu commands use the clipboard. They apply to the selected objects and text.

Undo/Redo

Undo negates the last **Align** menu command.

Redo performs the action that **Undo** negated.

Cut

Removes objects from the active page and puts them on the clipboard, replacing any previous clipboard contents. Objects placed on the clipboard retain all their attributes and the contents of their text fields.

Effect on Object Types

The effect of **Cut** varies with the type of object selected:

Nodes

Removes the node, its descendants and its connectors. Only the node and its descendants are placed on the clipboard. However, hierarchy, fusion, and port regions are lost because they are never put on the clipboard. See **Effect on hierarchies** below for more information.

Connectors

Removes the connector and its descendants. In a subsequent paste operation, the system prompts for the connector source and destination.

Regions

Removes the region and its descendants. In a subsequent paste operation, the system prompts for a region parent.

When the current selection is a code segment, **Cut** puts the corresponding key region on the clipboard along with its code segment.

Groups of Nodes

Removes everything contained within the group, including all internal connectors and descendants. Connectors with only their source or only their destination nodes in the group are removed without being saved to the clipboard.

Text

Selected text within an object can be cut as in any other X-Windows application.

Effect on Hierarchies

When the current selection contains socket or port places, **Cut** removes the port assignment (in the hierarchy region of the corresponding substitution transition).

To move objects without losing their hierarchical relationships (as happens when **Cut** is applied) use **Move Node** (**Makeup** menu) or palette pages (discussed in **Page Attributes** (**Set** menu)).

Restrictions

Cut cannot be applied (but **Clear** can be) to the following:

- Groups of regions or connectors.
- Page objects.
- Hierarchy, fusion, border, or mode regions, or one of their key regions.

Copy

Copies objects from the diagram to the clipboard by replacing any previous clipboard contents. Copied objects retain all their attributes and the contents of their text fields. **Copy** has the same effect as **Cut** except that there is no change on the active page.

Effect on Object Types

The effect of **Copy** varies with the type of object selected:

Nodes

Copies the node, its descendants and its connectors. Only the node and its descendants are placed on the clipboard. However, hierarchy, fusion, and port regions are not copied. See **Effect on hierarchies** below for more information.

Design/CPN Menu Reference

Connectors

Copies the connector and its descendants. In a subsequent paste operation, the system prompts for the connector source and destination.

Regions

Copies the region and its descendants. In a subsequent paste operation, the system prompts for a region parent.

When the current selection is a code segment, **Copy** puts the corresponding key region on the clipboard along with its code segment.

Groups of Nodes

Copies everything contained within the group, including all internal connectors and descendants. Connectors with only their source or only their destination nodes in the group are not copied.

Text

Selected text within an object can be copied to the clipboard as in any other X-Windows application.

Effect on Hierarchies

To duplicate objects without losing their hierarchical relationships (as happens when **Copy** is applied) use **Duplicate Node** (**Makeup** menu), or palette pages (discussed in **Page Attributes** (**Set** menu)).

Restrictions

Copy cannot be applied in the following circumstances:

- When the current selection is a group of regions or connectors.
- When the current selection contains page objects.
- When the current selection is a hierarchy, fusion, border, or mode region, or one of their key regions.

Paste

Pastes objects from the clipboard into the active window. Parent objects and their regions retain their relationships.

Effect on Object Types

The effect of **Paste** varies with the type of object to be pasted:

Nodes

The center of the node to be pasted is placed in the active window wherever the graphic tool is positioned. If the pointer is outside the active window, the copy will be placed in the center of the active window.

Connectors

When prompted, first place the graphic tool in the source node and click. Then place the graphic tool in the destination node and click again.

Auxiliary Regions

When prompted, place the graphic tool on the prospective parent node and click. The region retains the same relative position to the new parent as it had to its original parent object.

CPN Regions

When the clipboard contains a CPN region **Paste** works as described for auxiliary regions, except that only places, transitions, places/transitions or arcs (depending on the kind of CPN region on the clipboard) can be selected. When the prospective parent already has a CPN region of that kind, the text from the existing CPN region is replaced with the text from the region in the clipboard.

Groups of Nodes

Place the group pointer where the center of the upper left node of the group is to be drawn.

Design/CPN Menu Reference

Text

Text must be turned on. Place the insertion point where the copied text is to appear.

Clear

Deletes objects from the diagram without putting them on the clipboard.

The DELETE key has exactly the same effect as **Clear**.

Special Cases

Certain kinds of objects require special action when they are deleted:

Page Nodes

Removes the corresponding pages after a Caution Alert. When some of the deleted pages are subpages, the corresponding hierarchy regions are changed accordingly.

Page Connectors or Page Tags

Removes the corresponding part of the CPN hierarchy after a Caution Alert.

Hierarchy Regions:

Removes the corresponding part of the CPN hierarchy.

Socket or Port Places

Removes the port assignment in the hierarchy region of the corresponding substitution transitions.

Hierarchy, Fusion, Border, Code, or Mode Region

Removes the corresponding key regions.

Get Info

Displays detailed information about the current object or group. The information displayed depends on the object type and whether the selection is a single object or a group.

Single Object Display

- The object type, that is, place, transition, code segment, hierarchy key region, auxiliary region, or page node.
- If the object has been syntax checked and/or switched, the internal ML number (if any) is given after the object type.
- Whether the object is a substitution transition, or a fusion, socket, or port/exit place.
- Name, contents of CPN regions, etc. This kind of information is only provided for places, transitions, arcs, global declaration nodes and page nodes (see the dialogs below).
- How many CPN regions and auxiliary regions the object has (only children are counted). Three asterisks (***) indicate that the field is empty.
- The arc direction for an arc.

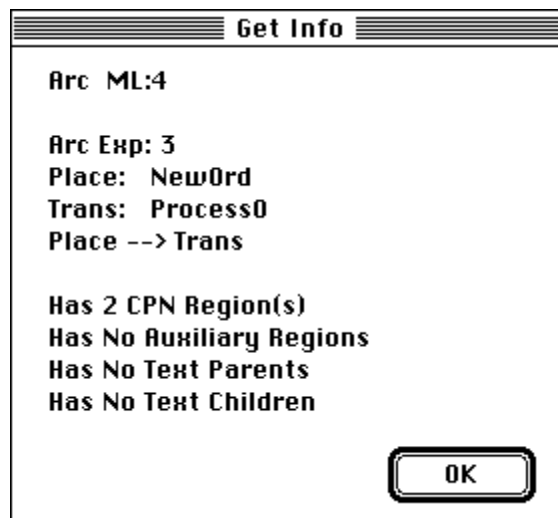
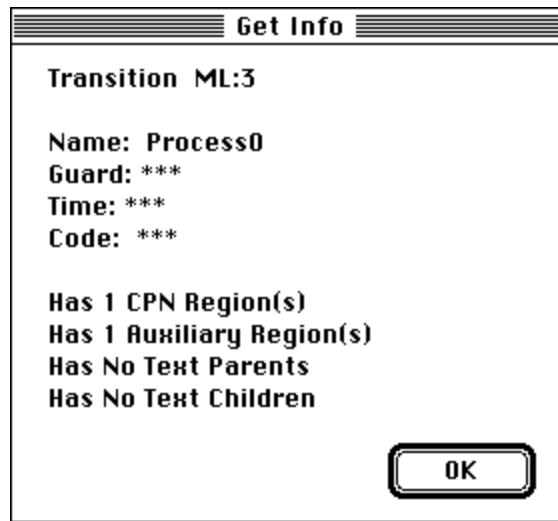
Group Display

- The object type, that is, the node, region or connector.
- The number of objects.

Colorsets, initial markings, guards, code segments and arc inscriptions are truncated after the first 15 characters.

Design/CPN Menu Reference

Examples of Get Info Dialogs



Chapter 21

CPN Menu Commands


Place	Alt+E
Transition	Alt+B
Arc	Alt+K
CPN Region	Alt+R
Declaration Node...	
Chart...	
Move to Subpage	
Replace by Subpage...	
Substitution Transition...	
Fusion Place...	
Port Place...	
Port Assignment...	
Syntax Check...	Alt+'
Remove Sim Regions...	

The **CPN** menu commands create CPN objects — places, transitions, arcs, CPN regions, declaration nodes, and charts — according to the defaults. Other commands in the menu form hierarchies by creating or relating pages.

The menu also contains commands for syntax checking the diagram and removing the simulation regions that participate in simulation.

Place

Creates places, with an ellipse as the default shape. The diagram defaults for places determine the attributes. For instance, to change the default size of the ellipse, choose **Shape Attributes (Set menu)** with a place selected.

The cursor, a bold ellipse , indicates that the system is in place creation mode.

A place is not created with CPN regions.

Text Use

Clicking the mouse creates a place and turns on text mode with the cursor inside the place. This provides the opportunity to immediately identify the place. Note, however, that this text is *not* the place name region. To create the place name region, use **CPN Region (CPN menu)**.

Changing Size and Position

Mouse clicks create default size places. Holding down the mouse and dragging the cursor also creates a node, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the node.

Creating Multiple Places

While in place creation mode each mouse click creates a new place.

Terminating Place Creation Mode


The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating place creation mode turns off text mode.

Transition

Creates transitions, with a rectangle as the default shape. The diagram defaults for transitions determine the attributes. For instance, to change the default size of the rectangle, choose **Shape Attributes** (**Set** menu) with a transition selected.

The cursor, a bold box , indicates that the system is in transition creation mode.

A transition is not created with CPN regions.

Text Use

Clicking the mouse creates a transition and turns on text mode with the cursor inside the transition. This provides the opportunity to immediately identify the transition. Note, however, that this text is *not* the transition name region. To create the transition name region see the **CPN Region** (CPN menu) command.

Changing Size and Position

Mouse clicks create default size transitions. Holding down the mouse and dragging the cursor also creates a transition, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the transition.

Creating Multiple Transitions

While in transition creation mode, each mouse click creates a new transition.

Terminating Transition Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating transition creation mode turns off text mode.

Design/CPN Menu Reference

Arc

Draws connectors between places and transitions. The diagram defaults for arcs determine the attributes.

The cursor, a bold arrow ➡, indicates that the system is in arc creation mode.

An arc is not created with an arc inscription region.

Restrictions

Each arc must start inside a place or a transition and end inside a CPN node of the opposite kind. This restriction is automatically enforced by the system: a first mouse-down outside a CPN node has no effect. Only a mouse-up (a click for segmented connectors) inside a CPN node of the opposite kind terminates the arc.

Drawing Single-Segment Connectors

Click inside one node (the source node) and drag the connector tool inside another node (the destination node). Design/CPN always draws arcs from border to border, along a line from the center of the source node to the center of the destination node.

Delete an arc by double-clicking outside a node during a drag. Arc length and placement are determined by the nodes they connect.

Drawing Multi-Segment Connectors

Follow the above process, with one difference: release the mouse at the location of the first bend point, outside the destination node. Subsequent clicks outside the destination node create other bend points. Click in the destination node to complete the arc.

Create a right angle by pressing the mouse on a bend point and pressing the COMMAND key.

Delete a bend point by placing the pointer tool on the bend point and by pressing the mouse button along with SPACEBAR.

Add a bend point by placing the pointer tool on a midpoint selection handle and dragging it to a location.

Effect of Moving Source/Destination Nodes

Design/CPN automatically redraws connectors when their source and destination nodes are moved or resized.

Endpoint Handle Use

Reattach an arc by selecting the endpoint handle and dragging it to another object (of allowable kind). After the arc has been created, use the endpoint regions to change the position of the source or destination endpoint.


Terminating Arc Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

CPN Region

Creates CPN regions. The diagram defaults for the specified kind of CPN region determine the attributes.

The cursor, , indicates that the system is in CPN region creation mode.

Restrictions

The command cannot be used to create hierarchy regions, fusion regions or port regions. These are created by means of the **CPN** menu commands **Move to Subpage**, **Substitution Transition**, **Fusion Place**, and **Port Place**.

Creating CPN Regions

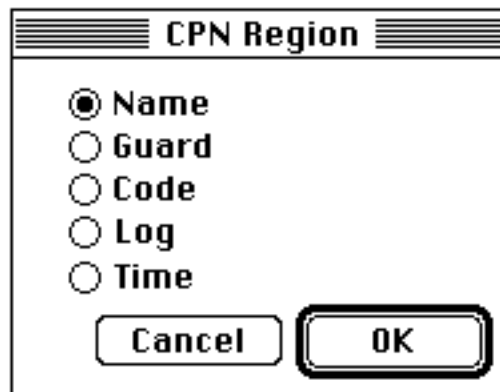
Places

When the current selection is a place, **CPN Region** displays the following dialog:



Transitions

When the current selection is a transition, **CPN Region** displays the following dialog:



Arcs

When the current selection is an arc, no dialog is displayed since the only possibility is an arc inscription region.

Multiple CPN Regions

Create a new CPN region for another object of the same object type (place, transition, or arc) by clicking on that object. Other kinds of clicks and pressing RETURN result in a prompt for a new parent object.

The created objects are CPN regions of the specified kind. Each of them is a region of the object that was selected immediately before the region was created. This means that the first CPN region becomes a region of the CPN object that was selected when the command was invoked. Similarly, the second CPN region becomes a region of the CPN object that was clicked between the creation of

the first and second CPN regions; the same process is applied to the other CPN regions, if any.

Object Descendants

The result is as if the corresponding ancestor node were selected.

Text

Text mode is automatically turned on.

Selecting a CPN object that already has a CPN region of the specified kind causes the entire text of the region to become selected for editing.


Terminating CPN Region Creation Mode

The mode is terminated by:

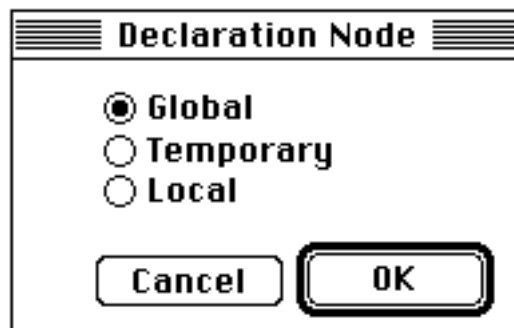
- Pressing ESC.
- Selecting another command.

Declaration Node

Creates declaration nodes. The diagram defaults for the specified declaration nodes determine the attributes.

The cursor, , indicates that the system is in declaration node mode.

Declaration Node displays the following dialog:



Design/CPN Menu Reference

When the **OK** button has been clicked, the command enters declaration creation mode. The created objects are declaration nodes of the specified kind.

Changing Size and Position

Mouse clicks create declaration nodes of the default size. Holding down the mouse and dragging the cursor also creates a node, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the node.

Creating Multiple Declaration Nodes

While in declaration creation mode, each mouse click creates a new declaration node

Restrictions

During a syntax check, a diagram is allowed to have only one global declaration node and at most one temporary declaration node. Moreover, each page can have at most one local declaration node.

Terminating Declaration Node Mode

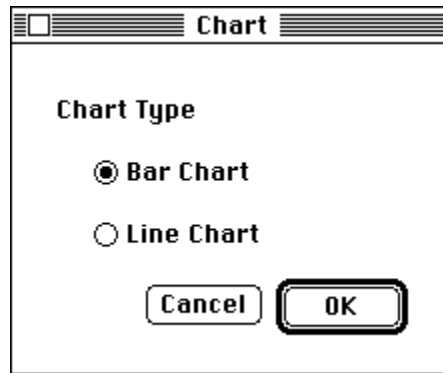
The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Chart

Creates a chart node. The diagram defaults for charts determine the default attributes of the new chart.

Design/CPN displays the following dialog:



After specifying the chart type and clicking **OK**, the Editor enters chart creation mode. The chart tool appears:



Clicking the mouse creates a default size chart node. The cursor location is the lower right corner.

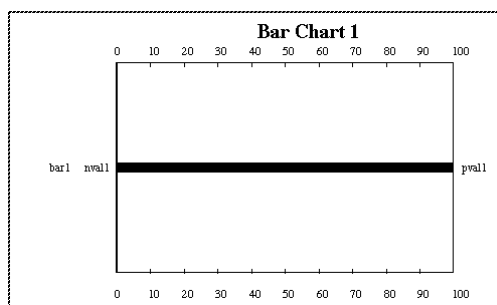
Holding down the mouse and dragging the cursor creates a node the size of which is determined by the extent of the drag. Holding down **Shift** while dragging the cursor moves the node. Releasing the mouse terminates node creation and causes the system to display the appropriate chart attributes dialog:

Bar Chart Dialog

Bar Chart		
Name Bar Chart 1	Regions	Update Period
Bars	<input checked="" type="checkbox"/> Title	<input checked="" type="checkbox"/> End of Run
No of Bars 1	<input type="checkbox"/> Legend	<input type="checkbox"/> Time
No of Parts 1	<input checked="" type="checkbox"/> Bar Names	<input type="checkbox"/> Step
Bar Height 10	<input checked="" type="checkbox"/> Pos. Values	
<input type="checkbox"/> History Chart	<input checked="" type="checkbox"/> Neg. Values	Value Type
Retain	<input checked="" type="checkbox"/> Upper Values	<input checked="" type="radio"/> Integer
	<input checked="" type="checkbox"/> Lower Values	<input type="radio"/> Real
Grids Lines	Initialization	Value Range
<input checked="" type="radio"/> Number 10	<input type="checkbox"/> Save Copy	Min 0
<input type="radio"/> Distance	<input checked="" type="checkbox"/> Keep Contents	Max 100
<input checked="" type="checkbox"/> Ticks Only		
Save...	Load	Reset
Cancel	OK	

Design/CPN Menu Reference

Defines the appearance, initialization, method of updating, and value range for the chart. The defaults are as shown in the dialog above. They produce a chart that looks like:



Name

Specifies the chart identifier, which is used by all the ML functions that reference this chart. The chart name must be unique within the diagram. Otherwise the system displays an error message when you try to click the **OK** button.

Bars

No of Bars

Specifies the number of bars in the chart.

No of Parts

Specifies the number of parts into which each bar is subdivided.

Bar Height

Specifies the height of the individual bars. The value is in pixels.

History Chart

Causes the system to generate a history chart.

Retain

Specifies the number of history chart bars to retain when the bars are moved up during updating. The number must be smaller than or equal to the number specified in the **No of Bars** option and greater than or equal to zero.

Grid Lines

Number

Causes the distance between two grid lines to be fixed while the numeric range between the grid lines varies. The number entered for Distance must be of the type specified in the **Value Type** option. If not, the system displays an error message.

Distance

Causes the numeric range between two grid lines to be fixed while the number of grid lines varies.

Tics Only

Specifies whether the grid lines are visible or not. If checked, only small tics can be seen where there are value regions.

Regions

Title

Causes the system to generate a title region for the chart. The initial title is the string "Title".

Legend

Causes the system to generate a legend describing the bar part patterns.

Bar Names

Causes the system to generate bar names.

Pos. Values

Causes the system to generate positive value regions containing default text. The font information may be changed in the Editor.

Neg. Values

Causes the system to generate negative value regions containing default text. The font information may be changed in the Editor.

Design/CPN Menu Reference

Upper Values

Causes the system to generate value range regions corresponding to the grid range above the chart. The text in these regions is generated from the min and max values (see the **Value Range** option) and can not be changed by the user. The font can be changed in the Editor.

Lower Values

Causes the system to generate value range regions corresponding to the grid range below the chart. The text in these regions is generated from the min and max values (see the **Value Range** option) and can not be changed by the user. The font information may be changed in the Editor.

Initialization

Save Copy

Causes the chart to be copied to a new page as an Aux node when the **Initial State** (**Sim** menu) command is invoked.

Keep Contents

Specifies whether or not the chart will be initialized when the **Initial State** (**Sim** menu) command is invoked. If checked, the chart will not be initialized when calling Initial State.

Update Period

End of Run

Specifies that the chart is to be updated at the end of the simulation run.

Time

Specifies the number of time units between chart updates for timed models being simulated with time. If both **Time** and **Step** are checked, the chart will be updated with both time and step intervals.

Step

Specifies the number of steps between chart updates. If both **Step** and **Time** are checked, the chart will be updated with both time and step intervals.

Value Type

Specifies whether values displayed in a chart are **integer** or **real**.

Value Range

Specifies the minimum to maximum range of values which are initially displayed in the plot area.

Real values must be entered for **min** and **max** if real is the **value type**.

The value of **min** must not be larger than 0; **max** must not be smaller than 0.

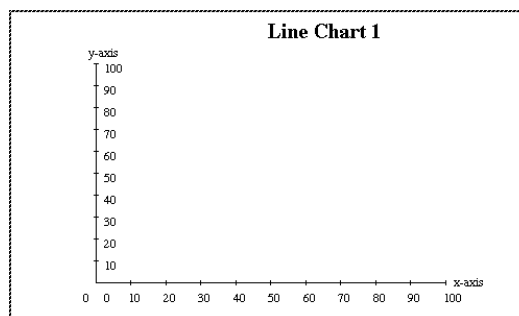
Line Chart Dialog

The dialog box is titled "Line Chart" and contains the following sections:

- Name:** Line Chart 1
- Lines:**
 - No of Lines: 3
 - Box Size: 10
 - ☐ Start at Origin
 - ☐ Horiz/Vert
- Grids Lines:**
 - ☒ Number (x: 10, y: 10)
 - ☐ Distance
 - ☒ Ticks Only
- Regions:**
 - ☒ Title
 - ☐ Legend
 - ☒ Axis Names
- Initialization:**
 - ☐ Save Copy
 - ☒ Keep Contents
 - ☐ Start at Origin
- Update Period:**
 - ☒ End of Run
 - ☐ Time
 - ☐ Step
- Value Type:**
 - ☒ Integer
 - ☐ Real
- Value Range:**
 - x: Min 0, Max 100
 - y: Min 0, Max 100
- Origin:**
 - x: 0
 - y: 0
- Overflow:**
 - ☒ Rescale Axis
 - ☐ Move Axis

Buttons at the bottom: Save..., Load, Reset, Cancel, OK.

Defines the appearance, initialization, method of updating, and value range for the chart. The defaults are as shown in the dialog above. They produce a chart that looks like:



Design/CPN Menu Reference

Name

Specifies the chart identifier, which is used by all the ML functions that reference this chart. The name must be unique within the diagram. Otherwise the system displays an error message when you try to click the **OK** button.

Lines

No of lines

Specifies the number of graph lines in the chart.

Box Size

Specifies the size of the line terminators. The value is in pixels.

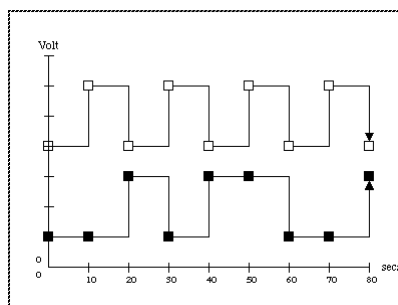
Start at Origin

Causes all lines to start from the origin.

Horiz/Vert

Specifies how the data points on the graph lines are connected.

Checked causes lines to be drawn with right angles, such as in a time series for hardware circuits seen below:



Non-checked causes straight lines to be drawn between points.

Grid Lines

The **x** and **y** entries have the same meaning:

Number

Causes the distance between two grid lines to be fixed while the numeric range between the grid lines varies. The number should be of the type specified in the **Value Type** option.

Distance

Causes the numeric range between two grid lines to be fixed while the number of grid lines varies.

Tics Only

Specifies whether the grid lines are visible or not. If checked, only small tics can be seen where there are value regions.

Regions

Title

Causes the system to generate a title region for the chart. The initial title is the string "Title".

Legend

Causes the system to generate a legend describing the patterns associated with the lines.

Axis Names

Causes the system to generate axis names.

Initialization

Save Copy

Causes the chart to be copied to a new page as an Aux node when the **Initial State** (**Sim** menu) command is invoked.

Design/CPN Menu Reference

Keep Contents

Specifies whether or not the chart will be initialized when invoking the **Initial State** (**Sim** menu) command. If checked, the chart will not be initialized when invoking **Initial State**.

Update Period

End of Run

Specifies that the chart is to be updated at the end of the simulation run.

Time

Specifies the number of time units between chart updates for timed models being simulated with time.

Step

Specifies the number of steps between chart updates.

Note: If both Time and Step are checked, the chart may be updated with both time and step intervals.

Value Type

Specifies whether values displayed in a chart are **integer** or **real**.

Value Range

The **x** and **y** entries have the same meaning:

Specifies the minimum to maximum range of values which are initially displayed in the plot area.

Real values must be entered for **min** and **max** if real is the **value type**.

Origin

Specifies where the **x** and **y** axes intersect initially. **x** and **y** must be located within the **Min** and **Max** range. Checking the **Move axis** option can change the location of the origin.

Overflow

Specifies methods to handle data that exceeds the limits specified when the chart was created. **Rescale axis** compresses the data; **Move axis** is analogous to the history chart update method.

Rescale Axis

Causes the system to automatically adjust the chart's display scale to accommodate changes in the data.

Move Axis

Causes the system to move the display scale to accommodate changes in the data. The display scale is fixed. If a chart will be updated in both the negative and positive direction, this feature should *not* be used.

Move to Subpage

Reduces the graphic complexity on a page by moving detail to a subpage, one level lower in the page hierarchy.

A selection nodes and connectors is moved to the subpage and replaced on the superpage with a substitution transition. The selected nodes are allowed to contain fusion places.

If the command is invoked when a region is selected, it works as if the corresponding ancestor node were selected.

Terms

External Arcs

Those arcs that connect transitions in the group of selected nodes to places outside of the selection.

Perimeter Transitions

Those transitions in the selection that have an external arc. The perimeter must be non-empty.

Socket Places

Those places outside of the selection that are connected to perimeter transitions through external arcs. These are the places that define the external arcs.

Port Places

The copies of the socket places that are created on the subpage.

Substitution Transition

The system replaces the selection on the superpage with a substitution transition. The position and size of the new substitution transition is determined in the same way as for ordinary transitions. The other attributes are determined by the diagram defaults for transitions.

Creating the Subpage

The substitution transition replaces all the selected nodes, which are moved to the subpage. All arcs between socket places and the perimeter transitions are reattached, so that they now have the new node as source/target (instead of the perimeter nodes). Auxiliary connectors between selected and non-selected nodes are deleted.

Subpage Contents

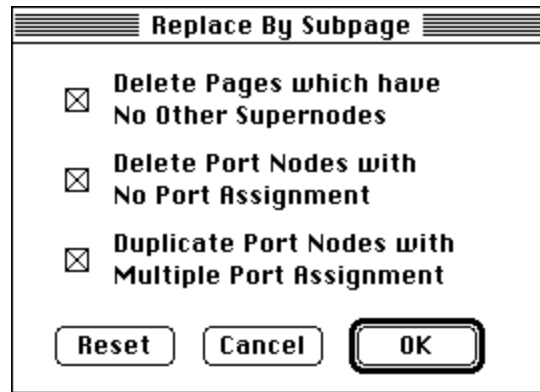
The subpage contains a set of port places and the selected nodes moved from the superpage. The subpage nodes have the same position as the superpage nodes. The subpage nodes also have the same set of interconnecting arcs and auxiliary connectors as they had on the superpage. Moreover, all arcs and auxiliary connectors are drawn in the same way as on the superpage.

Hierarchy Regions

The system automatically creates the hierarchy region for the substitution transition, assigns each socket place to the corresponding port place, and creates the port regions for the port places. The port type of a port place is determined relative to the corresponding socket place and the new supernode. It will be either **In**, **Out** or **I/O**.

Replace by Subpage

Replaces an existing substitution transition with the objects on its corresponding subpage:



The substitution transition and its surrounding arcs are replaced by a copy of the objects on the subpage.

Each port place is merged with the corresponding socket places without merging any of the arcs, connectors, or texts. This convention guarantees that a **Move to Subpage** (CPN menu), if immediately followed by a **Replace by Subpage**, will not result in any changes to the net. In effect, **Replace by Subpage** undoes the **Move to Subpage**.

However, if **Replace by Subpage** does not follow directly after **Move to Subpage**, changes made to the subpage are carried over.

The command can also be invoked when a region is selected and works as if the corresponding ancestor node were selected.

Substitution Transition

Turns transitions into substitution transitions by relating them to an existing subpage.

The hierarchy page window appears, and the status bar prompts: Click on Page Node to Select a Subpage.

Clicking on the page node returns to the original node, which is now labeled in the status bar as a substitution transition. When more than one transition is selected, each of them is turned into a substitution transition. The system automatically creates the hierarchy region for the substitution transition.

The actual subpage is active after the command is executed.

Design/CPN Menu Reference

Double-clicking on a substitution transition goes from the original page to the related page.

Invoking **Substitution Transition** when a region is selected works as if the corresponding ancestor node were selected.

Restrictions

Neither pages which are substitution-ancestors of the page containing the substitution transition nor the page itself can be selected for this would introduce a cycle in the substitution hierarchy.

Fusion Place

Turns non-fusion places into fusion places. The places are either inserted in a new fusion set or added to an existing one.

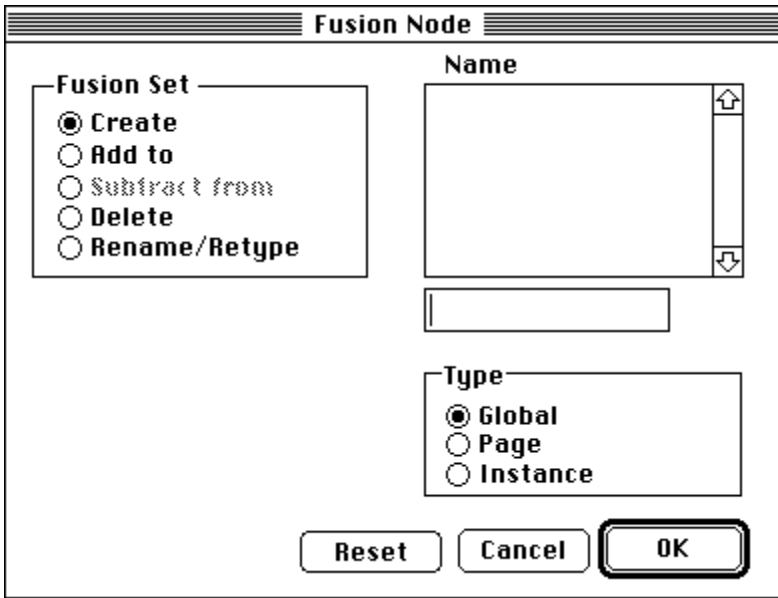
Fusion Place displays a dialog with one of three different forms. Clicking the left-hand set of radio buttons changes the right-hand side of the dialog and thus changes the form of the dialog.

When the current selection contains fusion places or port places, **Create** and **Add To** are disabled. When the current selection contains non-fusion places, **Subtract From** is disabled.

Invoking **Fusion Place** when a region is selected works as if the corresponding ancestor places were selected. The system automatically creates/deletes/updates the corresponding fusion regions.

The same dialog appears for both **Create** and **Rename/Retype**.

Create and Rename/Retype



Create creates a new fusion set:

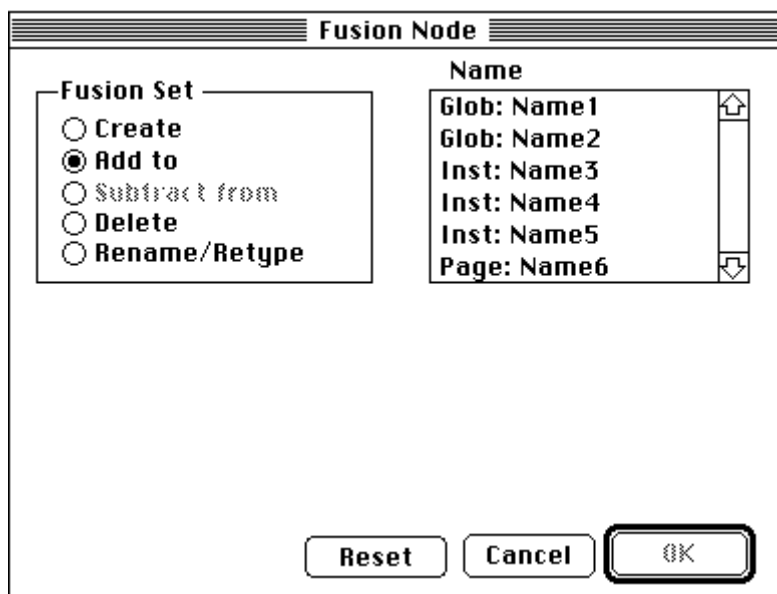
- Specify the name in the edit box, immediately below the list box.
- Select the type from the **Type** section.
- Click OK.

Rename/Retype renames or changes the type of a fusion set:

- Click on a list element. The fusion name is copied to the edit box.
- Select a fusion type.
- Make the changes .
- Click **OK**.

Design/CPN Menu Reference

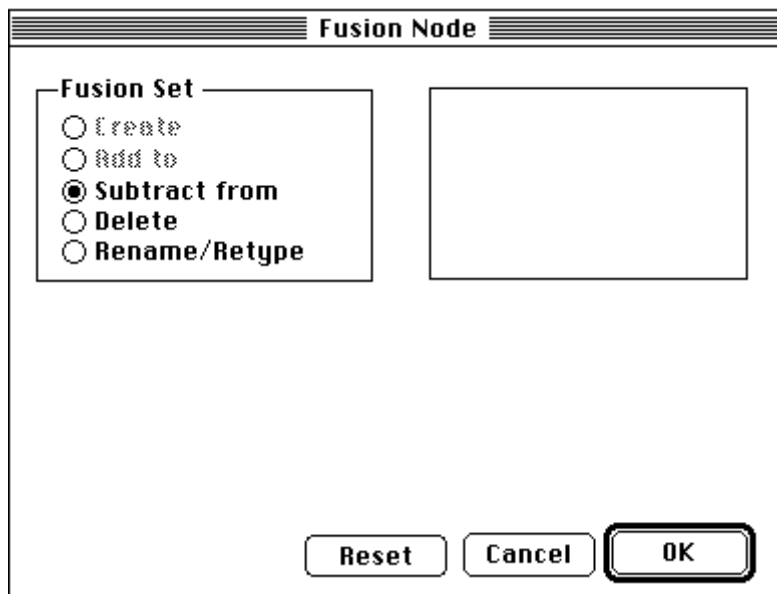
Add to



Add To adds selected places to a fusion set:

- Click on the list element representing the fusion set to which the selected places are added.
- Click **OK**.

Subtract from and Delete



Subtract From subtracts the selected places by clicking **OK**.

Delete deletes an entire fusion set.

Subtracting all places from a fusion set deletes the fusion set.

Naming Conventions

Fusion set names follow the syntax defined in Names and Numbers.

Moving Places Between Fusion Sets

To move places from one fusion set to another, subtract them before adding them.

Turning Fusion Places Into Non-Fusion Places

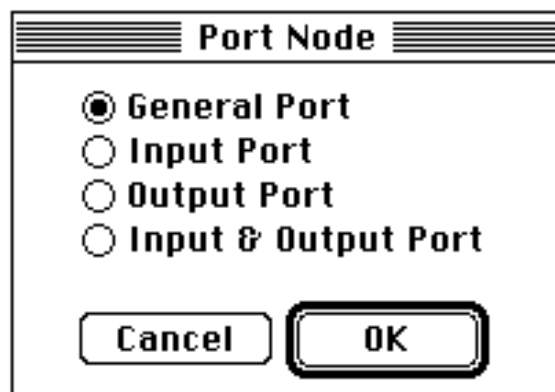
Turn fusion places into non-fusion places by deleting the corresponding fusion region.

Retyping Global Fusion Sets

The global fusion set is split into a number of fusion sets — one for each page that had places in the global fusion set.

Port Place

Turns non-port places into port places:



Port Types

General Port

Allows any set of arcs between the assigned socket place and the substitution transition. However, in order to be a socket place there must be at least one arc between the place and the substitution transition.

Input Port

The assigned socket place must have an input arc to the substitution transition and no output arc from the substitution transition.

Output Port

The assigned socket place must have an output arc from the substitution transition and no input arc to the substitution transition.

Input & Output Port

The assigned socket place must have an input arc to the substitution transition and an output arc from the substitution transition.

Using Port Place

When **Port Place** is invoked for a region it works as if the corresponding ancestor node were selected.

Port Place may be used to change the port type of a port place.

To turn port places into non-port places, delete the corresponding port region.

The port type information is used by **Port Assignment**. During a syntax check, the arcs for each socket place are checked for a match with the port types of the assigned port places.

Port Assignment

Establishes the port assignment between socket places and port places.

When invoked with a selected socket place, the subpage is made active before port selection.

Port Assignment can be invoked when a region is selected and works as if the corresponding ancestor place were selected.

Selecting Ports

Select normally to select ports with a matching port type.

Press **OPTION** to select ports with a non-matching port type.

Restrictions

Input Place

The port must be an input port or a general port.

Output Place

The port must be an output port or a general port.

Both an Input Place and an Output Place

The port must be an input/output port or a general port.

Place of More Than One Substitution Transition

One of the substitution transitions must be selected.

Port Assignment Exists

The old port assignment is overwritten.

Deleting Port Assignments

Remove a port assignment by overwriting it.

To remove the port assignment without creating a new one, do one of the following:

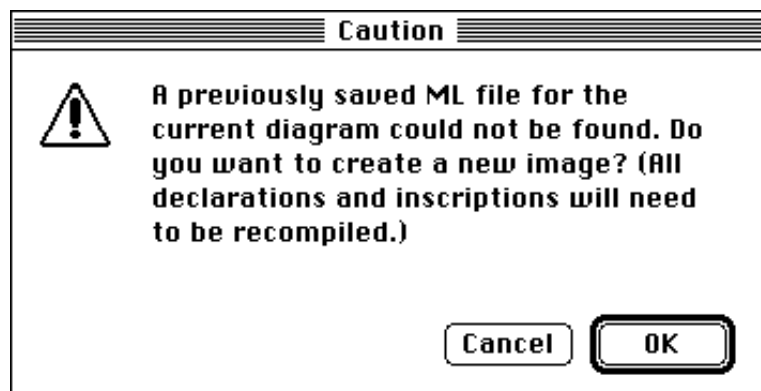
- Delete the corresponding hierarchy region.
- Delete the arcs between the socket place and the substitution transition.
- Make the port place a non-port.

Syntax Check

Checks declarations, inscriptions, substitution transitions, and fusion sets. **Syntax Check** also checks whether the compulsory restrictions are satisfied or not.

If you see a dialog that mentions a problem of any kind, see Appendix B, “Troubleshooting.”

If the diagram previously had an ML file, but the file no longer exists, the system prompts:



Error Reporting

When an error is located, an auxiliary node is created on the hierarchy page. This error node contains the name and number of each page showing an error and a text pointer to an auxiliary node on the erroneous page. Each of these error nodes contains a description of the errors found on the page and a text pointer to the error object.

Following the Text Pointers

In text mode, with the cursor in the text pointer (<<...>>),

- Select **Child Object** to move to the object with the error. Return to the error node by means of **Parent Object**.
- Use SHIFT-DOWN-ARROW.

Error Nodes

The error nodes are created with the default attributes of auxiliary boxes and are positioned in the upper left corner of the corresponding page.

A deleted error node is automatically recreated when needed.

Error nodes in the diagram are reused when new errors are reported. The error nodes are emptied before a syntax check.

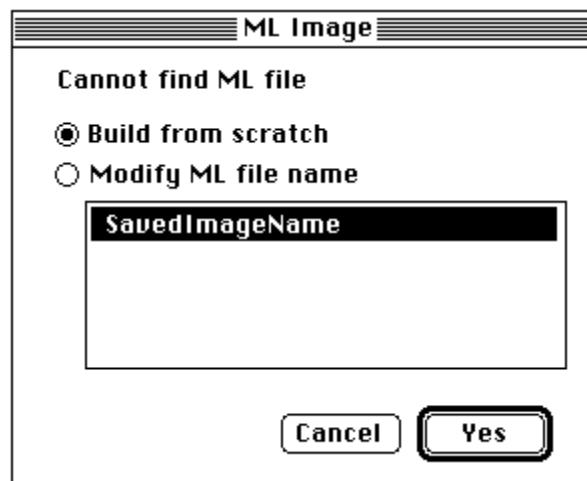
Status Bar Information

The status bar displays the number of objects checked. The syntax check can be stopped by typing ESC.

If the ML interpreter is not already running, **Start ML** will automatically be invoked before the syntax check.

Specifying the ML File Name

Design/CPN saves the ML file name, including path or file name, respectively, with the diagram. If that file cannot be found, the following dialog appears:



Use this dialog to specify:

- Start from the default image, resulting in a total syntax check.
- Let the system try again with another name.

If more than one ML image exists for the same diagram, be careful to select the right one; otherwise there may be a mismatch between the image and the database. Such a mismatch can always be eliminated by doing a complete syntax check.

Remove Sim Regions

Deletes simulator regions:



Use **Remove Sim Regions** on returning to the Editor after simulation to make significant changes in the diagram.

Remove Sim Regions works on the selected set of pages in the page hierarchy window, or on the active page.

Chapter 22

Sim Menu Commands

Bind Occurrence Set Start Step	Alt+B
Interactive Run Automatic Run	Alt+E Alt+R
Continue Stop	Alt+K Alt+D
Change Marking... Initial State...	
Select Instance	
Save Report... Clear Report...	
Update Chart	
Reswitch	Alt+ '

The commands in this menu prepare the diagram for simulation. They also allow you to regulate the simulation cycle, switch between page instances, or return the diagram to its initial state.

Bind

Manipulates bindings by:

- Manually including new bindings into the occurrence set
- Removing some of the already included bindings
- Inspecting the included bindings
- Calculating potential bindings.

Bind can be invoked only when all of the following are true:

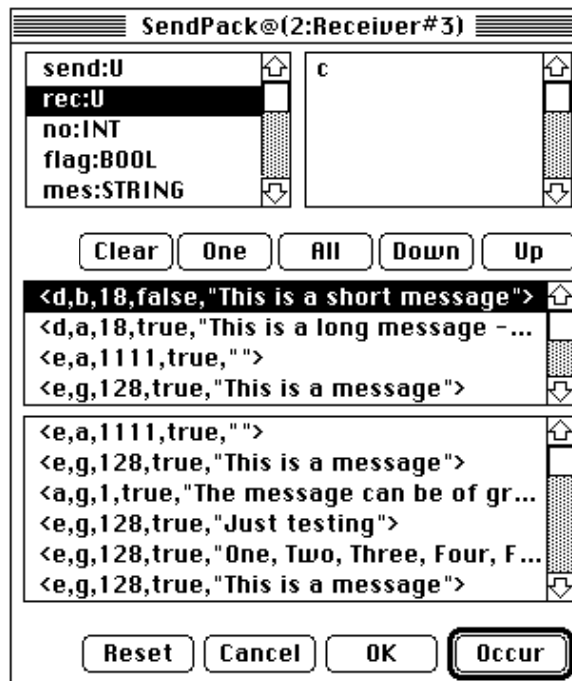
- Fair Interactive simulation is specified in the **General Simulation Options** dialog (**Set** menu).
- A single transition (or a single region of a transition) is selected.
- The selected transition is enabled, or has bindings in the current occurrence set.

Bind Dialog Format

The **Bind** dialog has three sections:

- The *editing area* for the CPN variables.
- The *included bindings* list box.
- The *potential bindings* list box.

The dialog also contains two bars of control buttons, the upper control button bar and lower control button bar.



Editing Area

The editing area is where values may be entered for a variable. It consists of:

- A list box (left) containing one entry for each variable for the transition
- An edit box (right) containing the value of the variable selected on the left.

Selecting a Variable

Scroll the list box using the scroll bars and select a variable by clicking on it. Only one variable's value is accessible at any given time.

Editing the Contents

Edit in the usual way by typing in the data.

Each time a new variable or a binding in the included or potential bindings list box is selected, the system checks that the current value belongs to the corresponding colorset. A beep occurs if the value is not of the appropriate type.

Design/CPN Menu Reference

Pressing the **One** button causes the system to assign a value to each unassigned variable.

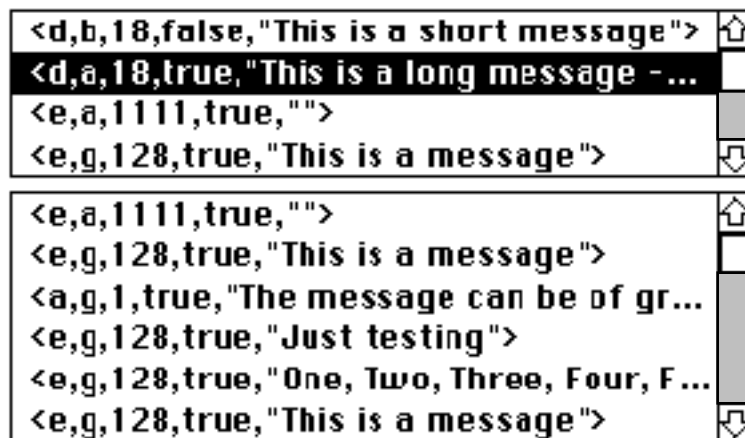
Values for the Variables

The editing area may contain *partial bindings*, that is, when some of the CPN variables are calculated by the code segment or have missing values.

The editing area is said to be *consistent* if its current contents can be extended to a binding for inclusion in the occurrence set. This means that the extended binding must be enabled in the current system state, concurrent with the binding elements that are already in the current occurrence set.

Included and Potential Bindings

There are two list boxes between the two rows of control buttons. The first list box consists of *included bindings*; the second list box has *potential bindings*. They appear as shown:



Syntax

To save space in the included and potential bindings list boxes, each binding <x=a,y=b,...,z=c> is abbreviated to: <a,b,...,c>, where the order is defined by the order of the variables in the editing area.

Included Bindings List Box:

Contains the letter “©” for calculated CPN variables.

Potential Bindings List Box:

Contains the letter “©” for calculated CPN variables.

The bindings contain a single space for CPN variables with missing values. For example, <b,a, ,©,c, ,g> represents a binding where the third and sixth CPN variable have missing values, while the fourth is calculated.

Included Bindings List Box

Included bindings are those bindings that have been included in the occurrence set for the transition (and page instance) in question.

Each time a binding is added to or deleted from this list, the following items are recalculated and redisplayed:

- The enabled transitions (shown by means of feedback regions)
- The non-committed part of the marking (shown in the current marking region)
- The input tokens (shown in the optional input token region)
- The output tokens (shown in the optional output token region)

Enter bindings in the included bindings by means of the **Up** and **Down** buttons in the upper control button bar.

Potential Bindings List Box

Potential bindings are bindings that are stored for later manipulation or display.

Clicking the **All** button causes the bindings to be calculated and put into the potential bindings.

Enter bindings in the potential bindings list box by means of the **Up** and **Down** buttons.

Control Buttons

The control buttons work differently depending on which of the three sections is selected. Selecting a binding in the included or potential bindings list boxes highlights the binding.

The default selection is the editing area.

Upper Control Button Bar

Clear Button

Works on the selected section of the **Bind** dialog: when the editing area is selected, **Clear** empties the large edit box for the current variable. When an included or potential binding is selected, **Clear** removes the selected binding.

One Button

Calculates *one* binding, consistent with the current contents of the editing area. The calculated binding is inserted into the editing area. It may be a partial binding, that is, some of the values may be missing.

If the editing area is inconsistent, a beep occurs. If the editing area can be extended in several different ways, the choice is random (total bindings are preferred to partial bindings).

If all edit boxes contain a color, before **One** is clicked, **One** checks the consistency of the binding.

All Button

Calculates all bindings that are consistent with the current contents of the editing area and shows them in a potential bindings as potential bindings. The editing area is not changed. Some of the calculated bindings may be partial.

Down and Up Buttons

Transfer bindings from the selected section of the **Bind** dialog to another. The buttons' names indicate the direction of the transfer.

The buttons work in a cyclic manner at the two ends; **Down** transfers from the potential bindings to editing area while **Up** transfers from the editing area to the potential bindings.

When a target is not present, it is created. When the source section is the editing area, the transfer is a move ; otherwise it is a copy. The following section describes in detail the workings of **Down** and **Up**:

- **Editing area —> included bindings**

When the editing area is selected, **Down** tries to *move* the contents of the editing area to the list box for included bindings. However, before this can be done, the contents are ex-

tended to a full binding; this is only possible when the editing area is consistent.

If the editing area contains a full consistent binding, this binding is simply added to the included bindings. If the editing area does not contain a full binding, the system extends the partial binding. This is done in the same way as when **One** is clicked, except that it also applies random drawing to find values for unbound CPN variables.

In both cases, the editing area is cleared. A beep occurs if the operation is unsuccessful; for instance, if the binding cannot be added to the occurrence set.

- **Editing area —> potential bindings**

When the editing area is selected, **Up** *moves* the binding in the editing area to the potential bindings (without any modifications). This is always possible, even when the binding is partial.

- **Included bindings —> potential bindings**

When a single included binding is selected, **Down** *copies* the binding to the list box for potential bindings.

- **Included bindings —> editing area**

When a single included binding is selected, **Up** *copies* the binding to the editing area. This operation is useful when you want to inspect an included binding in more detail than what is available for review in the included bindings.

- **Potential bindings —> editing area**

When a single potential binding is selected, **Down** *copies* the binding to the editing area. This operation is useful when you want to inspect a potential binding in more detail than what is available for review in the potential bindings.

- **Potential bindings —> included bindings**

When a single potential binding is selected, **Up** *tries to copy* the binding to the list box for included bindings. However, before this can be accomplished, the binding is extended to full bindings (in the same way as described in **One** button). The full bindings must be checked for conflict with each other. A beep occurs if the operation is unsuccessful.

Design/CPN Menu Reference

Lower Control Button Bar

Reset Button

Resets the bindings to the initial state.

Cancel Button

Cancels the dialog.

OK Button

Retains the occurrence set for a transition in the included bindings and closes the **Bind** dialog. The contents of the potential bindings are lost in this process.

Occur Button

Speeds up the simple cases. This button is a shortcut for the following sequence:

- **Down**
- **OK**
- **Start Step**

To define and invoke an occurrence set with a single binding, do the following:

- Invoke **Bind** (for example, by means of the key equivalent).
- Type the binding (unless you want a random binding).
- Invoke **Occur** (for example, by means of RETURN).

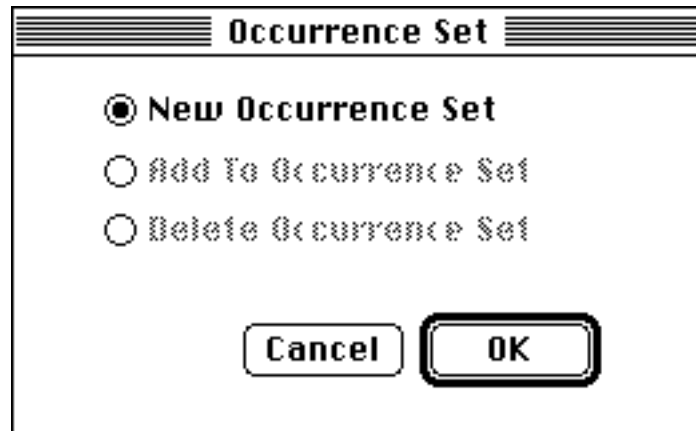
Invoking Bind Without the Bind Dialog

Invoking **Bind** with SHIFT and OPTION depressed invokes Occur directly without displaying a dialog. A single random binding is added to the occurrence set, which is immediately executed.

Invoking **Bind** with SHIFT depressed invokes **Down** directly without displaying a dialog. A single random binding is added to the occurrence set without being executed.

Occurrence Set

Creates and manipulates occurrence sets. **Occurrence Set** can be invoked only when Fair Interactive simulation is specified in the **General Simulation Options** dialog (**Set** menu).



New Occurrence Set

Creates a new randomly generated occurrence set.

Add To Occurrence Set

Adds a randomly generated occurrence set to the existing occurrence set. The new bindings are from page instances that are members of the proposed occurrence set.

Delete Occurrence Set

Deletes the existing occurrence set.

Start Step

Start a simulation step. **Start Step** can be invoked only when Fair Interactive simulation is specified in the **General Simulation Options** dialog (**Set** menu), and the occurrence set is not empty.

Interactive Run

Initiates an interactive run. **Interactive Run** can be invoked only when Fair Interactive simulation is specified in the **General Simulation Options** dialog (**Set** menu).

Automatic Run

Initiates an automatic run. **Automatic Run** can be invoked only when **Fast Automatic** or **Fair Automatic** simulation is specified in the **General Simulation Options** dialog (**Set** menu).

Continue

Restarts simulation after a breakpoint, or when it has been interrupted via the **Stop** command (**Sim** menu).

If simulation stops at the following breakpoints:

- Beginning of substep
- End of substep

then simulation is considered paused, and only **Continue** is available.

If simulation stops at the following breakpoint:

- Between steps

then the following dialog is presented:



Stop

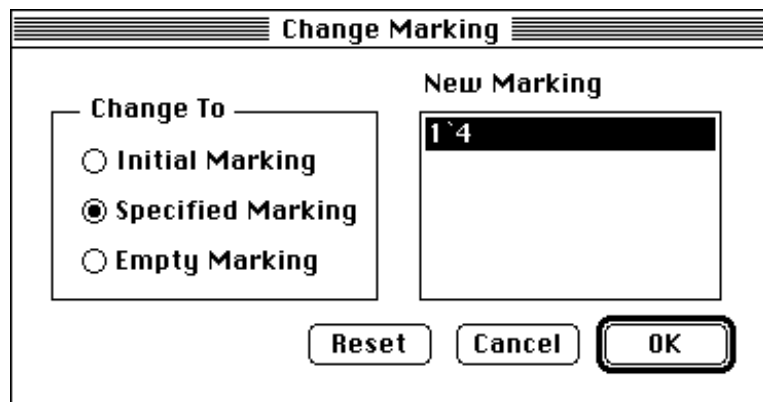
Causes simulation to stop at the end of the current step. It may also stop before that point if breakpoints are set.

Pressing the **OPTION** key with **Stop** causes the same behavior as if all breakpoints had been set.

Change Marking

Changes the marking of a selected place or a group of places when the occurrence set is empty.

The marking is only explicitly changed for the current page instance of the active window. The change may, however, imply that the markings of other places are changed because of fusion sets and port assignments.



Initial Marking

Specify the new marking by typing it at the cursor.

Specified Marking

Specify the marking to be identical to the initial marking of each of the selected places.

Empty Marking

Clear the marking.

Initial State

Generates a new initial system state by applying the current mode attributes. This means that the number of instances of the individual

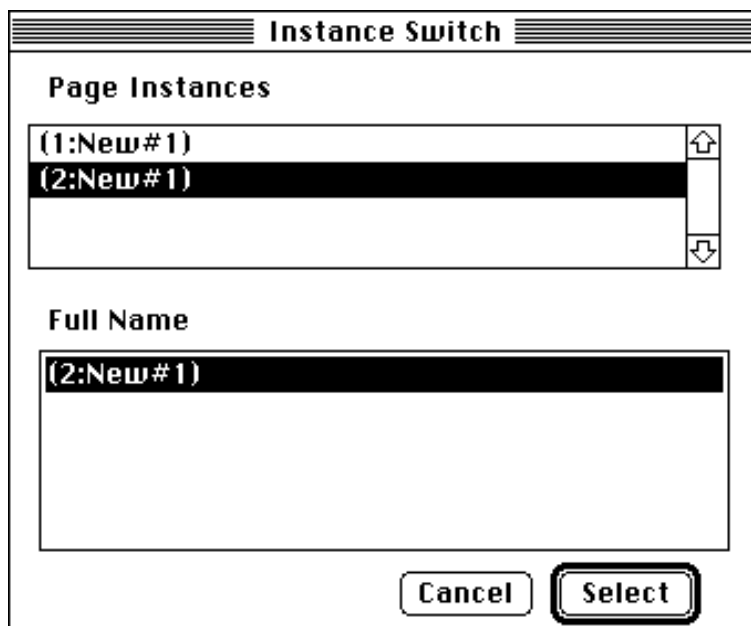
Design/CPN Menu Reference

pages may change if prime, multiplicity or include has been changed for some pages or compound nodes.

When you invoke **Initial State**, the **Save State** dialog appears, allowing you to save the current state.

Select Instance

Switches between instances of the current page.



You can also invoke **Select Instance** by depressing the SHIFT key and clicking the mouse in the current page's title bar.

Page Instances

Each line corresponds to a page instance. The line displays a short form of the instance's name. Select the page instance that you would like to switch to.

Full Name

Displays the full name of the page instance currently selected in the **Page Instances** section.

Select

Clicking **Select** causes the page instance selected in the **Page Instances** section to become the current instance.

Save Report

Save a copy of the current simulation report. Invoking Save Report displays the **Save As** dialog, allowing you to name the reports and indicate where it is to be saved.

The **Record** subdialog of **General Simulation Options (Set menu)** specifies the information included in the report.

Clear Report

Deletes all stored information on the course of simulation. See **Save Report (Set menu)**.

Update Chart

Available only when a chart or group of charts is currently selected. Causes an immediate update of the selected chart(s).

Reswitch

Reswitches the current diagram. **Initial State (Sim menu)** requires reswitching whenever inscriptions are changed in the simulator. Using **Simulation Code Options (Set menu)** to change any code generation parameter also requires reswitching.

Chapter 23

Aux Menu Commands

Connector Box Rounded Box Ellipse Polygon Regular Polygon Wedge Picture Line Label	Alt+L
Make Region Make Node	Alt+M
Convert to CPN... Convert to Aux	
Start ML/Stop ML ML Evaluate	Alt+;

The **Aux** menu commands draw non-formal graphic objects (nodes and connectors).

They can also start, stop, and evaluate ML with commands in this menu.

Connector

Draws auxiliary connectors between two nodes. The diagram defaults for connectors determine the attributes.

The cursor, a light arrow \rightarrow , indicates that the system is in auxiliary connector creation mode.

Drawing Single-Segment Connectors

Click inside one node (the source node) and drag the connector tool inside another node (the destination node). Design/CPN always draws connectors from border to border, along a line from the center of the source node to the center of the destination node.

Delete a connector by double-clicking outside a node during a drag. Connector length and placement are determined by the nodes they connect.

Drawing Multi-Segment Connectors

To create a segmented connector, follow the above process, with one difference: release the mouse at the location of the first bend point, outside the destination node. Subsequent clicks outside the destination node create other bend points. Click in the destination node to complete the connector.

Pressing the mouse on a bend point and pressing the COMMAND key snaps the two segments of the bend point into a right angle.

Delete a bend point by placing the pointer tool on the bend point and pressing the mouse button along with SPACEBAR.

Add a bend point by placing the pointer tool on a midpoint selection handle and dragging it to a location.

Effect of Moving Source/Destination Nodes

Design/CPN automatically redraws connectors when their source and destination nodes are moved or resized.

Endpoint Handle Use

Reattach a connector by selecting the endpoint handle and dragging it to another object. After the connector has been created, use the endpoint regions to change the position of the source or destination endpoint.


Terminating Connector Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Box

Creates auxiliary nodes with a box shape. The diagram defaults for boxes determine the attributes. For instance, to change the default size of the rectangle, choose **Shape Attributes (Set menu)** with a box selected.

The cursor, a box , indicates that the system is in box creation mode.

Text Use

Clicking the mouse creates a box and turns on text mode with the cursor inside the box. This provides the opportunity to immediately identify the box.

Changing Size and Position

Mouse clicks create default size boxes. Holding down the mouse and dragging the cursor also creates a box, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the box.

Creating Multiple Boxes

While in box creation mode, each mouse click creates a new box.

Terminating Box Creation Mode


The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating box creation mode turns off text mode.

Rounded Box

Creates auxiliary nodes with a rounded box shape. The diagram defaults for rounded boxes determine the attributes. For instance, to change the default size of the rectangle, choose **Shape Attributes (Set menu)** with a rounded box selected.

The cursor, a rounded box , indicates that the system is in rounded box creation mode.

Text Use

Clicking the mouse creates a rounded box and turns on text mode with the cursor inside the rounded box. This provides the opportunity to immediately identify the rounded box.

Changing Size and Position

Mouse clicks create default size rounded boxes. Holding down the mouse and dragging the cursor also creates a rounded box, the size of which is determined by the extent of the drag. Pressing **SHIFT** while dragging the cursor moves the rounded box.

Creating Multiple Rounded Boxes

While in rounded box creation mode, each mouse click creates a new rounded box.

Terminating Rounded Box Creation Mode


The mode is terminated by:

- Pressing **ESC**.
- Selecting another command.

Terminating rounded box creation mode turns off text mode.

Ellipse

Creates auxiliary nodes with an ellipse shape. The diagram defaults for ellipses determine the attributes. For instance, to change the default size of the ellipse, choose **Shape Attributes (Set menu)** with an ellipse selected.

The cursor, an ellipse , indicates that the system is in ellipse creation mode.

Text Use

Clicking the mouse creates an ellipse and turns on text mode with the cursor inside the ellipse. This provides the opportunity to immediately identify the ellipse.

Changing Size and Position

Mouse clicks create default size ellipses. Holding down the mouse and dragging the cursor also creates an ellipse, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the ellipse.

Creating Multiple Ellipses

While in ellipse creation mode, each mouse click creates a new ellipse.

Terminating Ellipse Creation Mode


The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating ellipse creation mode turns off text mode.

Polygon

Creates an auxiliary node, shaped as a polygon. The diagram defaults for boxes determine the attributes. For instance, to change the default size of the rectangle, choose **Shape Attributes (Set menu)** with a polygon selected.

The cursor, a polygon , indicates that the system is in polygon creation mode.

Design/CPN Menu Reference

Text Use

Clicking the mouse creates a polygon and turns on text mode with the cursor inside the polygon. This provides the opportunity to immediately identify the polygon.

Changing Size and Position

Mouse clicks create default size polygons. Holding down the mouse and dragging the cursor also creates a polygon, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the polygon.

Creating Multiple Polygons

While in polygon creation mode, each mouse click creates a new polygon.

Drawing Polygons

The command sets the first vertex or bend point. Subsequent vertices are created by clicking the mouse. To complete the polygon, double-click on the last vertex. A polygon must have at least three vertices or it will be removed when double-clicked.

Pressing the mouse on a bend point and pressing the COMMAND key snaps the two segments of the bend point into a right angle.

Delete a bend point by placing the pointer tool on the bend point and pressing the mouse button along with SPACEBAR.

Add a bend point by placing the pointer tool on a midpoint selection handle and dragging it to a location.

Terminating Polygon Creation Mode


The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating polygon creation mode turns off text mode.

Regular Polygon

Creates auxiliary nodes with a regular polygon shape. The diagram defaults for regular polygons determine the attributes. For instance, to change the default size of the regular polygon, choose **Shape Attributes** (**Set** menu) with a regular polygon selected.

The cursor, a regular polygon , indicates that the system is in regular polygon creation mode.

Text Use

Clicking the mouse creates a regular polygon and turns on text mode with the cursor inside the regular polygon. This provides the opportunity to immediately identify the regular polygon.

Changing Size and Position

Mouse clicks create default size regular polygons. Holding down the mouse and dragging the cursor also creates a regular polygon, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the regular polygon.

Creating Multiple Rounded Boxes

While in regular polygon creation mode, each mouse click creates a new regular polygon.

Terminating Regular Polygon Creation Mode

The mode is terminated by:


- Pressing ESC.
- Selecting another command.

Terminating regular polygon creation mode turns off text mode.

Wedge

Creates auxiliary nodes with a wedge shape. The diagram defaults for wedges determine the attributes. For instance, to change the default size, choose **Shape Attributes** (**Set** menu) with a wedge selected.

Design/CPN Menu Reference

The cursor, , indicates that the system is in wedge creation mode.

Text Use

Clicking the mouse creates a wedge and turns on text mode with the cursor inside the wedge. This provides the opportunity to immediately identify the wedge.

Changing Size and Position

Mouse clicks create default size wedges. Holding down the mouse and dragging the cursor also creates a wedge, the size of which is determined by the extent of the drag. Pressing SHIFT while dragging the cursor moves the wedge.

Creating Multiple Wedges

While in wedge creation mode, each mouse click creates a new wedge.

Terminating Wedge Creation Mode


The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Terminating wedge creation mode turns off text mode.

Line

Creates multi-point lines classified as auxiliary nodes.

The cursor, a cross , indicates that the system is in line creation mode.

Terminating Line Creation Mode


The mode is terminated by:

- Pressing ESC.

- Selecting another command.

Label

Creates auxiliary nodes shaped as labels.

The cursor, a label , indicates that the system is in label creation mode.

Label creates a special borderless node to contain text. Text is always turned on when the label tool is active. A click of the label tool places an insertion point on the page.

Dimensions

A label's dimensions are determined by the text typed into it. Design/CPN does not word-wrap text typed in a label. Press the RETURN key to break lines.

Restrictions

A label's dimensions may not be changed by dragging or by any other graphic operation. For example, a label that is a group member does not change with the group, and a label that is a region does not change with its parent.

Labels cannot have fill added nor can the layering logic be altered.

Moving Labels

Move a label around the diagram with the graphic tool. The label's boundary is indicated by selection handles.

Terminating Label Creation Mode

The mode is terminated by:

- Pressing ESC.
- Selecting another command.

Make Region

Turns auxiliary nodes into auxiliary regions or selects a new parent for auxiliary regions. The command subordinates the current node to another object on the page.

Creating a Region

1. The status bar prompts: Select a Node, Region, or Connector to contain the new Region.
2. Place the pointer within the region's parent.
3. When the object's borders flash, click the mouse and the action is complete.
4. Selection handles reappear on the boundary of the new region and the status bar displays: Type: Region.

Effect on Parents

Some operations performed on a parent are also be performed on its regions (with the exception of a label), including moving and resizing.

Operations performed on a region do not affect its parent.

Restrictions

All the selected objects must be auxiliary nodes or auxiliary regions; all selected objects become auxiliary regions. If some of the selected objects are descendants of places, it is only possible to select places. The same applies to transitions.

SHIFT is no longer a modifier key.

Make Region does not reposition a region when it gets a new parent. To obtain this result, apply **Cut + Paste**.

Make Node

Turns auxiliary regions into auxiliary nodes.

Convert to CPN

Converts auxiliary objects into CPN objects.

Nodes

Auxiliary nodes may be converted into places, transitions, or declaration nodes. Nothing happens to their regions or their connectors.

Regions

Auxiliary regions may be converted into CPN regions of a specified kind. This can only be done when all the corresponding parent nodes are already CPN nodes.

Connectors

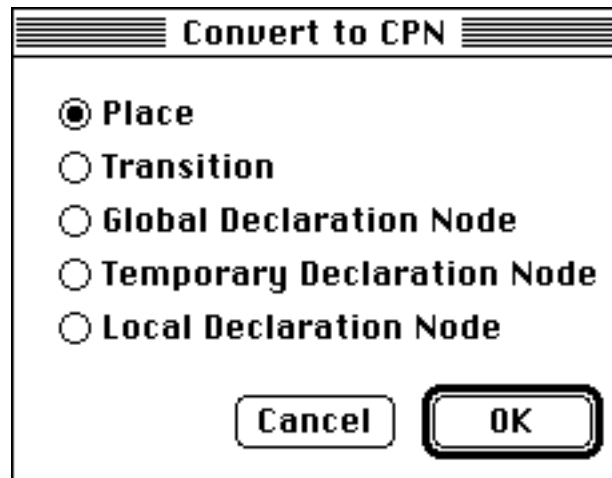
Auxiliary connectors are converted into arcs (nothing happens to their regions). This can only be done, when all the corresponding source/destination nodes are already CPN nodes.

Dialogs

The command evokes three different dialogs. Auxiliary connectors can only be converted into arcs, while regions of arcs can only be converted into arc inscriptions. These cases do not evoke dialogs.

Design/CPN Menu Reference

Nodes Selected:



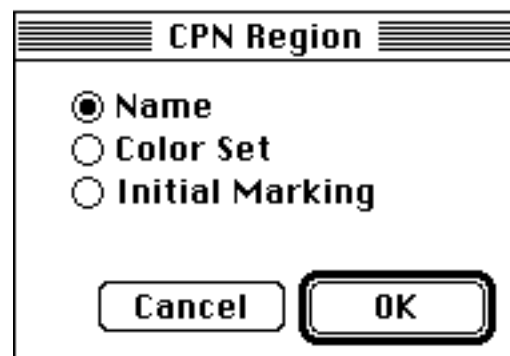
A dialog box titled "Convert to CPN" with a standard window border. Inside, there is a list of five radio button options: "Place" (selected), "Transition", "Global Declaration Node", "Temporary Declaration Node", and "Local Declaration Node". At the bottom right, there are two buttons: "Cancel" and "OK".

Convert to CPN

- ☒ **Place**
- ☐ **Transition**
- ☐ **Global Declaration Node**
- ☐ **Temporary Declaration Node**
- ☐ **Local Declaration Node**

Cancel **OK**

Place Regions Selected:



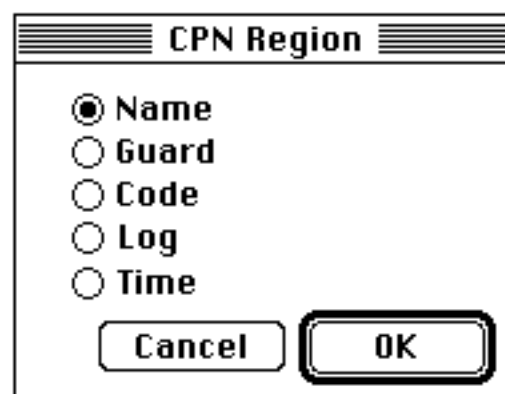
A dialog box titled "CPN Region" with a standard window border. Inside, there is a list of three radio button options: "Name" (selected), "Color Set", and "Initial Marking". At the bottom right, there are two buttons: "Cancel" and "OK".

CPN Region

- ☒ **Name**
- ☐ **Color Set**
- ☐ **Initial Marking**

Cancel **OK**

Transition Regions Selected:



A dialog box titled "CPN Region" with a standard window border. Inside, there is a list of five radio button options: "Name" (selected), "Guard", "Code", "Log", and "Time". At the bottom right, there are two buttons: "Cancel" and "OK".

CPN Region

- ☒ **Name**
- ☐ **Guard**
- ☐ **Code**
- ☐ **Log**
- ☐ **Time**

Cancel **OK**

Convert to Aux

Converts objects into auxiliary objects.

Nodes

Nodes are converted into auxiliary nodes; simultaneously, all their regions and surrounding connectors are converted into auxiliary regions and auxiliary connectors.

Regions

Regions are converted into auxiliary regions.

Arcs

Arcs are converted into auxiliary connectors. Simultaneously, the existing arc inscription region is converted into an auxiliary region.

Restrictions

Note that page objects are not converted.

Start ML

Starts up the ML interpreter. **Start ML** is available only in the Editor when the ML Interpreter is not running, in which case **Start ML** replaces **Stop ML** in the **Aux** menu. The interpreter always runs in the simulator, so **Start ML** is never available there.

If you see a dialog that mentions a problem of any kind, see Appendix B, “Troubleshooting.”

Stop ML

Shuts down the ML interpreter. **Stop ML** is available only in the Editor when the ML Interpreter is running, in which case **Stop ML** replaces **Start ML** in the **Aux** menu. The interpreter always runs in the simulator, so **Stop ML** is never available there.

Stop ML displays the **Save As** dialog, allowing you to save the diagram.

ML Evaluate

Evaluates the text of the current object as ML code. The result of this evaluation appears in an auxiliary region which the system creates next to the selected object.

If part of the current object's text is selected in text mode, only the selected part will be evaluated.

The auxiliary regions are created with the default attributes of auxiliary rounded boxes. If an auxiliary region already exists (from an earlier evaluation), this region is re-used; hence its old contents are overwritten.

Chapter 24

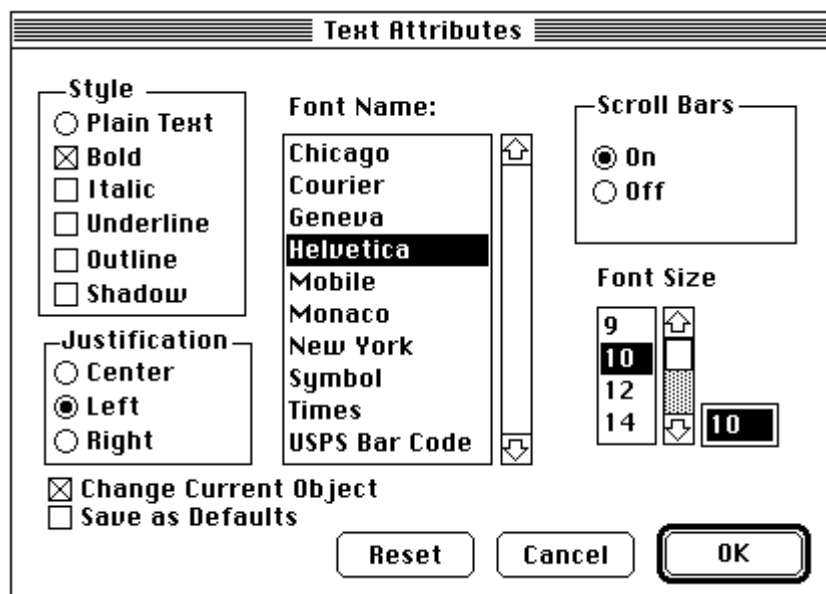
Set Menu Commands

Text Attributes...	Alt+6
Graphic Attributes...	Alt+7
Shape Attributes...	Alt+8
Region Attributes...	Alt+9
Page Attributes...	Alt+0
Mode Attributes...	
Chart Attributes...	
Hierarchy Page Options...	
Interaction Options...	
Merge Options...	
Text Options...	
Syntax Options...	
Simulation Code Options...	
General Simulation Options...	Alt+U
Interactive Simulation Options...	
Occurrence Set Options...	
Transition Feedback Options...	
ML Configuration Options...	
Copy Defaults...	

The **Set** menu command specify object attributes, diagram options, and simulation options.

Text Attributes

Changes the text attributes of the selected objects. The command changes the font, size, style, and justification; it also removes and adds text scroll bars globally.



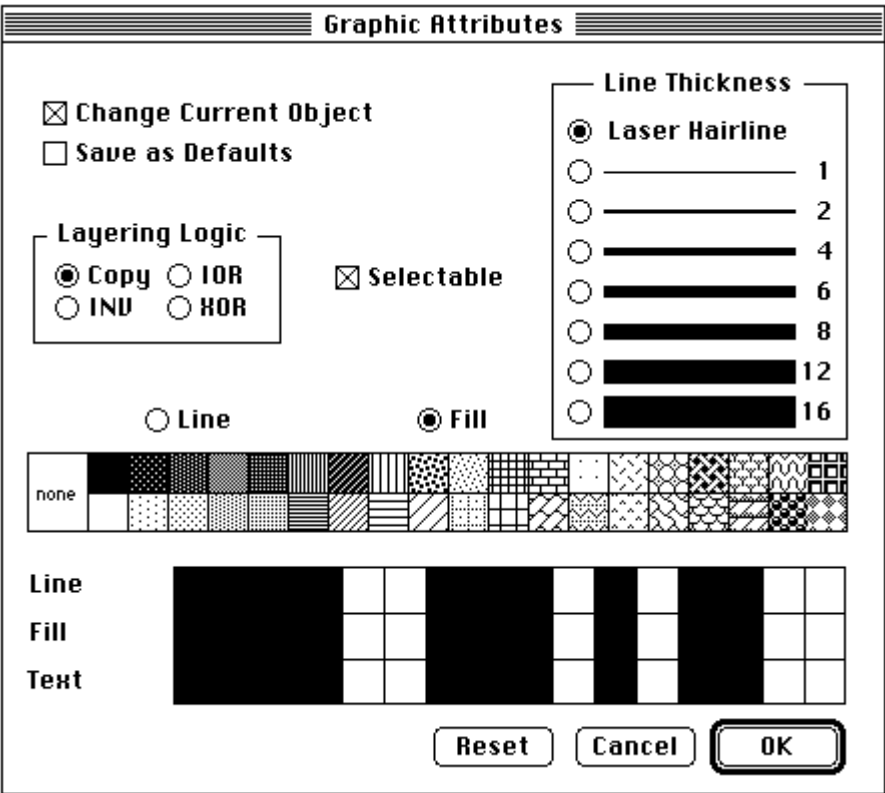
Saving Settings

Save the current settings as either system or diagram defaults by checking **Save as Defaults** and clicking **OK**. This brings up the following dialog:



Graphic Attributes

Changes graphic attributes for an object. These attributes include object fill, border line thickness, line type, and layering logic.



Under X-Windows, color is not supported.

Saving Settings

Save the current settings as either system or diagram defaults by checking **Save as Defaults** and clicking **OK**. This brings up the following dialog:



Layering Logic

These options affect how the fill and border of overlapping objects are displayed on the screen and printed on the ImageWriter. They affect an object in relation to the object that underlies it (its background).

Note: The LaserWriter supports only copy logic.

Copy

The **Copy** setting is most often used. The border and fill of the current object is not affected by the attributes of the underlying object. However, if the top layer has a fill of None, the fill of the object below it will show through.

Selectable

To avoid accidentally altering an object, remove the check to make the object non-selectable. Use this feature to lock a region in place.

To manipulate the object again use:

- **Next** (**Makeup** menu) or **Previous** (**Makeup** menu)
- **Parent** (**Makeup** menu) or **Child** (**Makeup** menu)
- Arrow keys and recheck the selectable box.

Line Thickness

Sets the line thickness of a node's border or a connector line.

Line and Fill Pattern

To set an object's or an arrowhead's fill, click on the **Fill** button and click in a pattern box.

To set the border or connector line type, click on the **Line** button and click in a pattern box to fill it.

A line type of **None** and a fill of **None** make objects invisible. Clicking within an invisible object's boundaries, selection handles appear (unless the object is non-selectable).

Shape Attributes

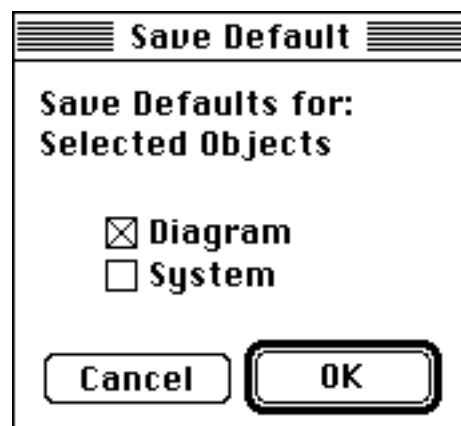
Changes the shape attributes of selected objects. The shape category of the current object — not its object type — determines which dialog appears on the screen.

The command applies a number of different dialogs, one for each shape:

- Nodes and regions can have these shapes: box, rounded box, ellipse, polygon, regular polygon, wedge, picture, and label.
- Connectors can have these shapes: segmented and curved.

Saving Settings

The method of saving the current settings as either system or diagram defaults varies with the particular dialog: either check **Save as Defaults** and click **OK**, or click the **Save** button. Both methods bring up the following dialog:



A **Load** button is available in certain dialogs.

Design/CPN Menu Reference

Dialog Determination

The shape of the selected object — or the first object in the selected group — determines which dialog is invoked.

The selected object also determines the attribute values displayed in the dialog. Attributes not applicable to some group members because of their shape do not affect those members.

When a label or a key region is selected, the command does not affect the object itself, but all descendants get a new size and a new position.

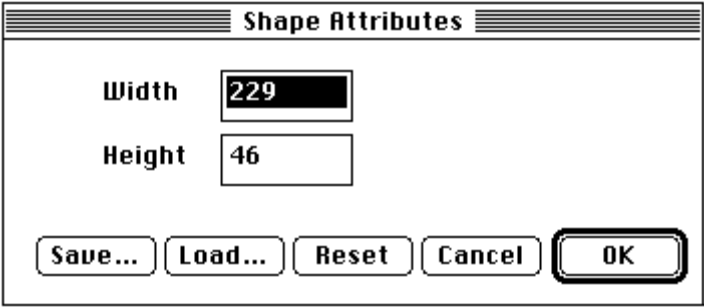
When a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Initial State of the Dialog

Defined by selected objects. If objects in the current selection are of the same shape, the corresponding dialog will appear for the selected shape. If the objects in the current selection are of different shapes, only the common attributes are shown — typically, width and height (for nodes and regions) or arrow orientation and width/height of heads/text (for connectors).

Dialogs

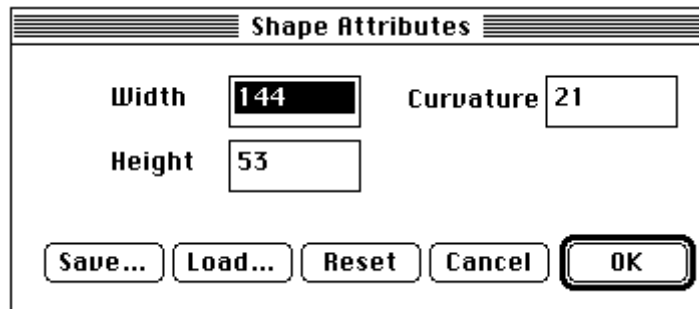
Box Shape, Ellipse Shape, Picture Shape



The image shows a dialog box titled "Shape Attributes". It contains two input fields: "Width" with the value "229" and "Height" with the value "46". At the bottom, there are five buttons: "Save...", "Load...", "Reset", "Cancel", and "OK". The "OK" button is highlighted with a thick border.

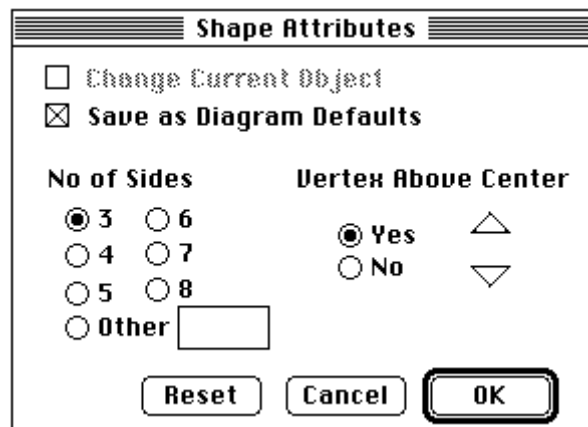
Shape Attributes	
Width	229
Height	46
Save... Load... Reset Cancel OK	

Rounded Box Shape



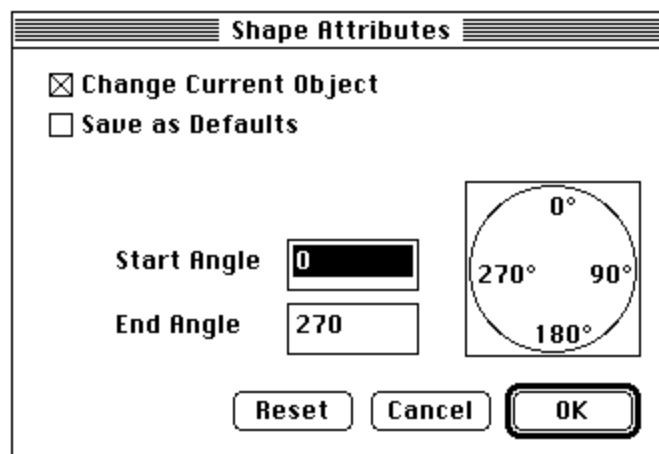
The dialog box is titled "Shape Attributes". It contains two input fields: "Width" with the value "144" and "Height" with the value "53". To the right of these is a "Curvature" input field with the value "21". At the bottom, there are five buttons: "Save...", "Load...", "Reset", "Cancel", and "OK". The "OK" button is highlighted with a double border.

Regular Polygon Shape



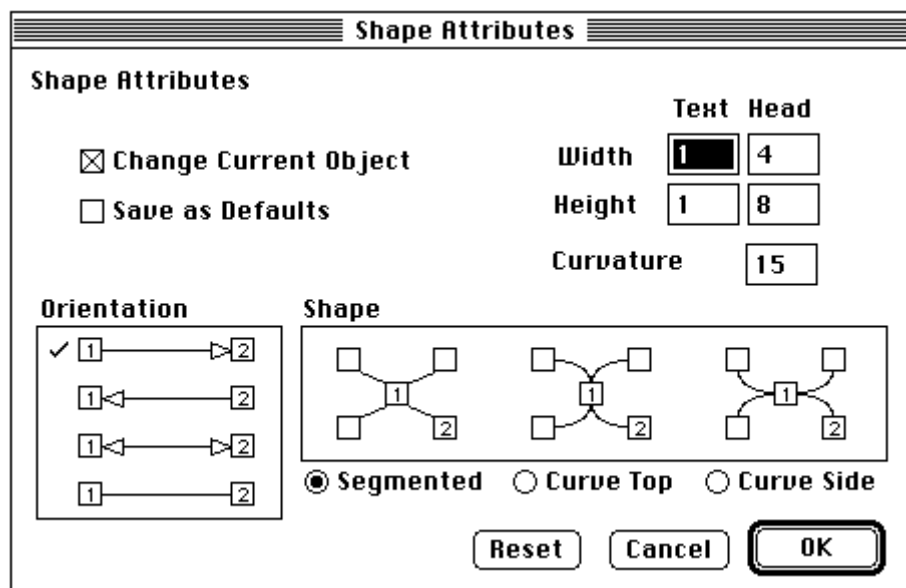
The dialog box is titled "Shape Attributes". It has two checkboxes: "Change Current Object" (unchecked) and "Save as Diagram Defaults" (checked). Below these are two sections. The "No of Sides" section has radio buttons for 3, 4, 5, 6, 7, 8, and "Other" with an adjacent input field. The "Vertex Above Center" section has radio buttons for "Yes" (selected) and "No", with small triangle icons next to them. At the bottom are "Reset", "Cancel", and "OK" buttons. The "OK" button is highlighted with a double border.

Wedge Shape



The dialog box is titled "Shape Attributes". It has two checkboxes: "Change Current Object" (checked) and "Save as Defaults" (unchecked). Below these are two input fields: "Start Angle" with the value "0" and "End Angle" with the value "270". To the right of these is a circular diagram showing angles at 0°, 90°, 180°, and 270°. At the bottom are "Reset", "Cancel", and "OK" buttons. The "OK" button is highlighted with a double border.

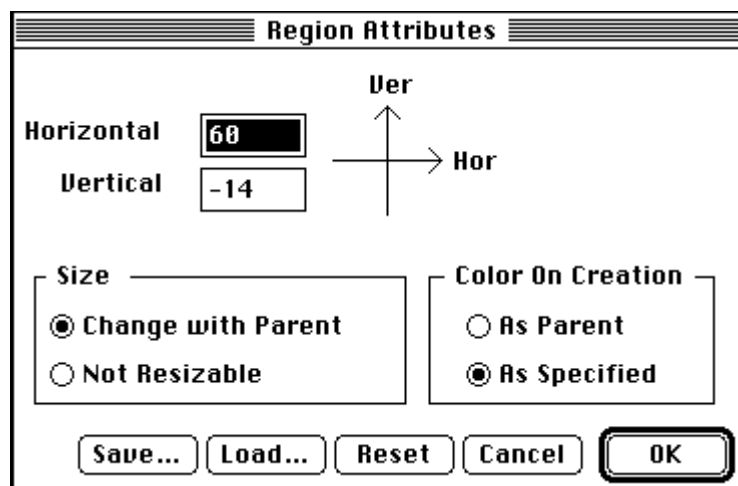
Connector Shape



The **Shape Attributes** dialog box is used to configure connector shapes. It includes a title bar, a 'Shape Attributes' section with checkboxes for 'Change Current Object' and 'Save as Defaults', and input fields for 'Text Head' (Width: 1, Height: 1, Curvature: 15). The 'Orientation' section shows four arrow styles between boxes labeled 1 and 2, with the first style selected. The 'Shape' section displays three connector types: Segmented (selected), Curve Top, and Curve Side, each with a diagram showing the connection between two nodes. At the bottom are 'Reset', 'Cancel', and 'OK' buttons.

Region Attributes

Changes position, size, and color relationships between parent and child objects:



The **Region Attributes** dialog box is used to configure region attributes. It includes a title bar, a coordinate system diagram with 'Hor' and 'Ver' axes, and input fields for 'Horizontal' (60) and 'Vertical' (-14). The 'Size' section has radio buttons for 'Change with Parent' (selected) and 'Not Resizable'. The 'Color On Creation' section has radio buttons for 'As Parent' and 'As Specified' (selected). At the bottom are 'Save...', 'Load...', 'Reset', 'Cancel', and 'OK' buttons.

Initial State of the Dialog

The selected object — or the first object in the selected group — determines the attribute values displayed in the dialog. When a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Position

Determined relative to the corresponding parent object:

Nodes

It is the center.

Arcs

It is the intersecting point between the arc and the node.

Size

Determines whether the size of the child changes with the size of the parent.

Color

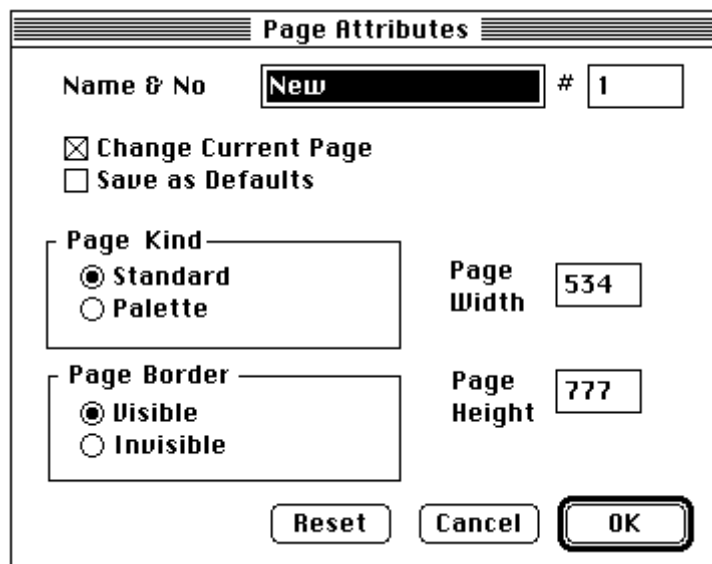
Specifies whether the color of a region follows the parent's color or those set in the defaults at creation time.

Popup

Only applicable for simulator key regions.

Page Attributes

Changes the page attributes.



The screenshot shows the 'Page Attributes' dialog box. It has a title bar with the text 'Page Attributes'. Inside, there are several fields and checkboxes. At the top, 'Name & No' is followed by a text box containing 'New' and a '#' followed by a text box containing '1'. Below this are two checkboxes: 'Change Current Page' (checked) and 'Save as Defaults' (unchecked). There are two groups of radio buttons. The first group is labeled 'Page Kind' and contains 'Standard' (selected) and 'Palette'. The second group is labeled 'Page Border' and contains 'Visible' (selected) and 'Invisible'. To the right of these groups are two text boxes: 'Page Width' containing '534' and 'Page Height' containing '777'. At the bottom are three buttons: 'Reset', 'Cancel', and 'OK'.

Name & No

Names and numbers the current page.

Master Page

Create a master page by typing “10000” in the page number box. Master page contents are printed on every page in the diagram when print output is set to **Normal** in the **Page Setup** dialog.

See **Page Setup** (**File** menu) for a complete description of master page creation, and **Print** (**File** menu) for the effects of the master page on the diagram.

Page Kind


Palette Page

Creates a custom palette of objects and thus greatly simplify the diagramming task.

Custom palette objects can include any collection of objects that can be created or pasted into Design/CPN, including objects with curved connectors, picture nodes, filled wedges, and irregular polygons.

Objects drawn with the custom palette can be manipulated in the same way as objects created with the Design/CPN menu commands.

Using the palette page: The palette page functions as a palette when it is not the active page and when another page is active:

1. Click on the desired palette object to highlight it.
2. Move the cursor to the active page. The pointer changes to the palette tool .
3. Position the pointer tool where the center of the object is to be drawn.
4. Click to draw the selected object.

The palette object is terminated by:

- Selecting another palette object.
- Choosing another command.
- Pressing ESC.

Change On-Screen Page Border Size

The page size is measured in points (72 points = 1 inch, 25.4 mm). Changes in the on-screen page size made in this dialog do not affect the printed page size, which is set in **Page Setup (File menu)**.

To restore the page size from **Page Setup**, click the **Reset** button in this dialog. The system default is set in **Page Setup**.

Make Page Borders Visible or Invisible

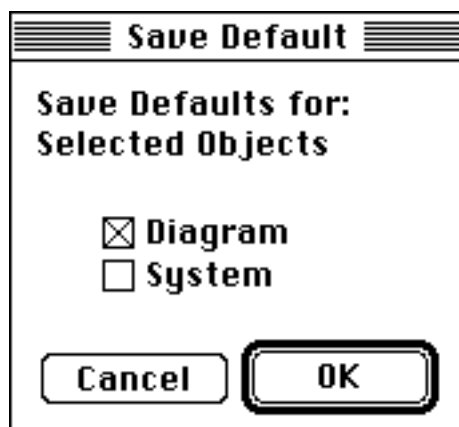
The standard default setting is **Visible**.

Invisible

The page borders will also be invisible when the diagrams printed. To see the page borders on the screen but not on the printed diagram, check the **Omit Borders** section in the **Page Setup** dialog.

Saving Settings

Save the current settings as either system or diagram defaults by checking **Save as Defaults** and clicking **OK**. This brings up the following dialog:



Restrictions

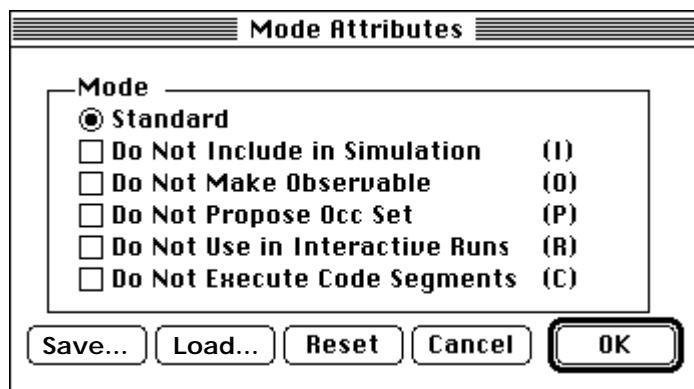
The page name may have defaults but not the page number.

The command cannot be applied to a group of page nodes.

Mode Attributes

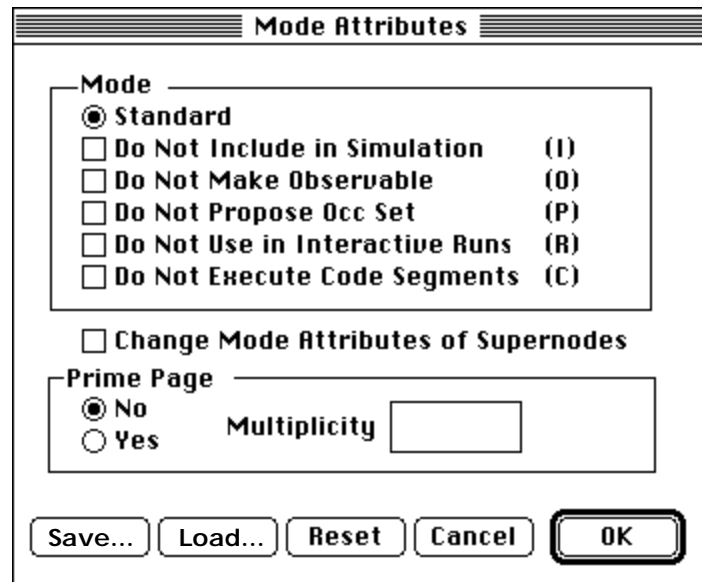
Changes the mode attributes. **Mode Attributes** applies two dialogs.

Substitution Transitions Selected



When the current selection is a substitution transition (or a group of substitution transitions), **Mode Attributes** changes the mode attributes for these transitions.

Page Nodes Selected



The dialog box is titled "Mode Attributes". It contains a "Mode" section with a radio button for "Standard" (selected) and five checkboxes: "Do Not Include in Simulation (I)", "Do Not Make Observable (O)", "Do Not Propose Occ Set (P)", "Do Not Use in Interactive Runs (R)", and "Do Not Execute Code Segments (C)". Below this is a checkbox for "Change Mode Attributes of Supernodes". The "Prime Page" section has a radio button for "No" (selected) and a radio button for "Yes", followed by a "Multiplicity" text field. At the bottom are buttons for "Save...", "Load...", "Reset", "Cancel", and "OK".

Change Mode Attributes of Supernodes

Determines the application of the specified mode attributes (minus the prime page information) for all substitution transitions that have the selected page as subpage.

Prime

Specifies whether or not the current page is to be a prime page.

Multiplicity

Controls the number of page instances.

Chart Attributes

Changes chart attributes.

Bar Charts

When the current selection is a bar chart (or a group of bar charts), the command changes the chart attributes for the corresponding bar charts.

Design/CPN Menu Reference

Line Charts

When the current selection is a line chart (or a group of line charts), the command changes the mode attributes for the corresponding line charts.

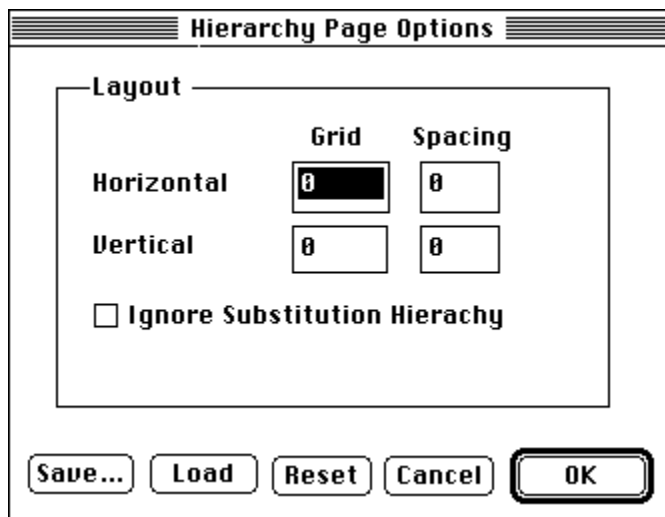
Dialogs

The dialogs displayed by the system are the same as those displayed by the **Chart** (CPN menu) command. If one chart is selected the values will be those of the chart. If a group of charts is selected there are no defaults; all the options must be specified and the charts will be redrawn according to the new specifications.

See the **Chart** (CPN menu) command for a complete description of each dialog and its options.

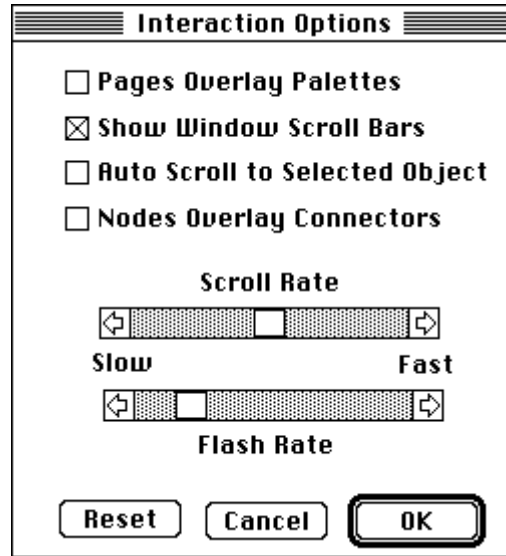
Hierarchy Page Options

Specifies the automatic layout and redraw of the hierarchy page.



Interaction Options

Specifies system interaction:



Pages Overlay Palettes

Causes new pages to overlay an open palette page.

Show Window Scroll Bars

Shows window scroll bars.

Auto Scroll to Selected Object

Scrolls when necessary to make the current object visible on an active page.

Nodes Overlay Connector

Specifies whether connectors are drawn on top of nodes, or vice versa.

Scroll Rate

Adjusts the window scrolling speed.

Design/CPN Menu Reference

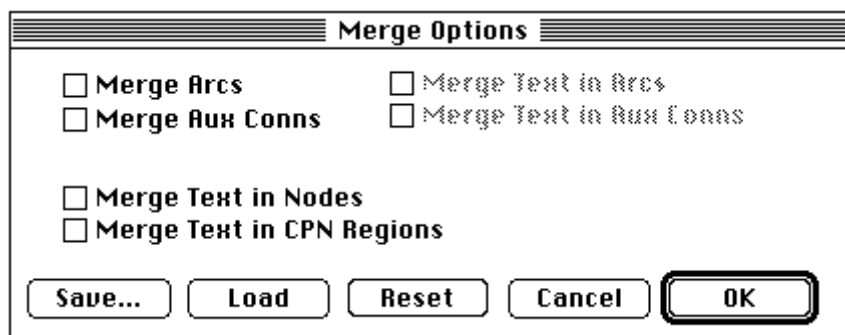
Flash Rate

Adjusts the speed at which the object borders flash when prompted to select an object.

When a new diagram is created, the interaction options are copied from the previous diagram (or from the system defaults, if no previous diagram exists).

Merge Options

Specifies the operations of the merge node when using **Merge Node** (**Makeup** menu).



Merge Arcs

Specifies whether arcs with the same source node and the same target node are merged or not. If the box is checked, the corresponding **Merge Text in Arcs** is made active.

Merge Aux Connectors

Specifies whether connectors with the same source node and the same target node are merged or not. If the box is checked, the corresponding **Merge Text in Aux Conns** is made active.

Merge Text in Nodes

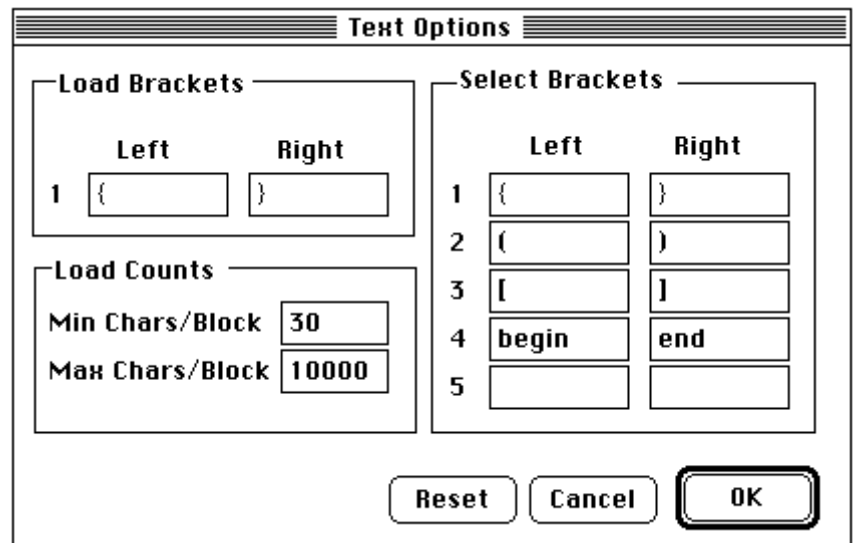
Specifies whether the texts of the nodes are merged or not.

Merge Text in CPN Regions

Specifies whether the texts of the CPN regions are merged or not. Only regions of the same kind are merged. Arc inscription regions are only merged **Merge Arcs** is checked.

Text Options

Defines the text brackets and character counts used by **Load Text** and **Select to Bracket**.



The **Text Options** dialog box is divided into three main sections: **Load Brackets**, **Load Counts**, and **Select Brackets**. The **Load Brackets** section contains a table with two columns, **Left** and **Right**, and one row for index 1, showing curly braces { and }. The **Load Counts** section contains two labeled input fields: **Min Chars/Block** with the value 30, and **Max Chars/Block** with the value 10000. The **Select Brackets** section contains a table with two columns, **Left** and **Right**, and five rows for indices 1 through 5. The values are: 1: { and }, 2: (and), 3: [and], 4: begin and end, 5: empty boxes. At the bottom right are three buttons: **Reset**, **Cancel**, and **OK**.

	Left	Right
1	{	}

Load Counts	
Min Chars/Block	30
Max Chars/Block	10000

	Left	Right
1	{	}
2	()
3	[]
4	begin	end
5		

Load Brackets and Load Counts

Use these options to set the parameters for **Load Text** (**File** menu). See **Load Text** for a description of the brackets' functions.

Min Chars/Block only influences a **Load Text** with delimiter blocking.

Max Chars/Block only influences a **Load Text** with character count blocking.

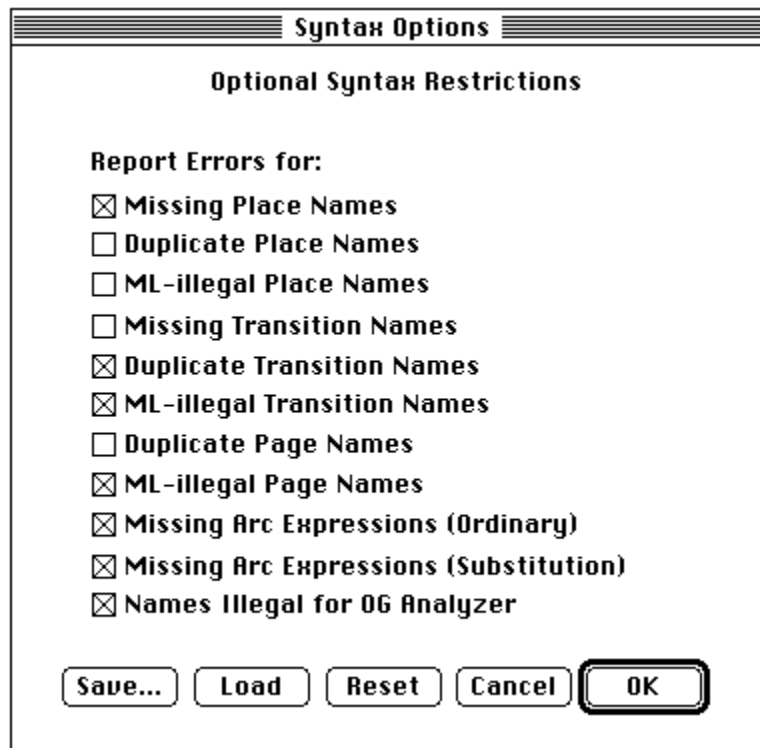
Select Brackets

Specifies the delimiters used by **Select to Brackets** (**Text** menu). Use these pairs of boxes to specify custom sets of delimiters.

Syntax Options

Specifies which optional syntax errors are to be reported during syntax check. The settings chosen in this dialog affect what happens when either **Syntax Check** or **Enter Simulator** is invoked.

Syntax errors found as a result of optional restrictions are reported in exactly the same way as all other syntax errors, i.e. with error boxes and hypertext pointers to erroneous objects.



Missing Place Names

The system generates a syntax error when it encounters a place with an empty name.

Duplicate Place Names

The system generates a syntax error when it encounters a place with a name that has been used before for a place on the same page. Places within the same fusion group are not subject to this restriction.

ML-illegal Place Names

The system generates a syntax error when it encounters a place with a name that is not a legal ML identifier.

Note: This restriction applies to ML reserved words since they are not legal ML identifiers.

Missing Transition Names

The system generates a syntax error when it encounters a transition with an empty name.

Duplicate Transition Names

The system generates a syntax error when it encounters a transition with a name that has been used before for a transition on the same page.

ML-Illegal Transition Names

The system generates a syntax error when it encounters a transition with a name that is not a legal ML identifier.

Note: This restriction applies to ML reserved words since they are not legal ML identifiers.

Duplicate Page Names

The system generates a syntax error when it encounters a page with a name that has been used before for another page.

ML-Illegal Page Names

The system generates a syntax error when it encounters a page with a name that is not a legal ML identifier.

Note: This restriction applies to ML reserved words since they are not legal ML identifiers.

Missing Arc Expressions (Ordinary)

The system generates a syntax error when it encounters an ordinary arc with an empty arc inscription.

Design/CPN Menu Reference

Missing Arc Expressions (Substitutions)

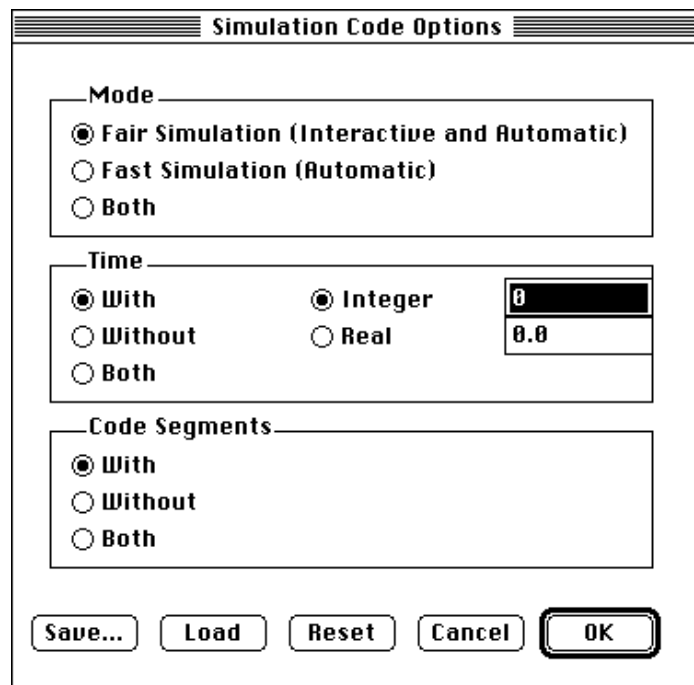
The system generates a syntax error when it encounters a substitution arc with an empty arc inscription.

Names Illegal for OG Analyzer

The system generates a syntax error when it encounters a name that is not acceptable to the Occurrence Graph Analyzer.

Simulation Code Options

Determines what features of the simulator will be available after the diagram is switched.



The dialog box titled "Simulation Code Options" contains three sections: "Mode", "Time", and "Code Segments". The "Mode" section has three radio buttons: "Fair Simulation (Interactive and Automatic)" (selected), "Fast Simulation (Automatic)", and "Both". The "Time" section has two columns of radio buttons: "With" (selected) and "Without", and "Integer" (selected) and "Real". To the right of these are two input fields with values "0" and "0.0". The "Code Segments" section has three radio buttons: "With" (selected), "Without", and "Both". At the bottom are five buttons: "Save...", "Load", "Reset", "Cancel", and "OK" (highlighted).

This dialog allows you to speed up the process of switching to the simulator by omitting the creation of executable code that will never be used. The choices made in this dialog also affect what options are available in the **General Simulation Options** dialog (**Set** menu), and in the **Sim** menu.

Mode

Fair Simulation (Interactive and Automatic)

Enables fair simulation. Fair simulation specifies a random selection from among potential enabling bindings. Each possible binding is equally likely to be used. This option results in intermediate switching time and slowest execution speed.

Fast Simulation (Automatic)

Enables fast simulation. Fast simulation specifies that selections from among potential enabling tokens will be made as rapidly as possible, without consideration for fairness. This option results in minimum switching time and fastest execution speed.

Both

Enables both fair and fast simulation. You can then do either type of simulation without having to reswitch the diagram by making appropriate choices in the **General Simulation Options** dialog (**Set** menu). This option results in maximum switching time. Execution speed depends on the type of execution selected in the **General Simulation Options** dialog.

Time

With

Enables timed simulation. Any time regions, inscriptions, etc. will affect the course of simulation. This option results in intermediate switching time.

Without

Disables timed simulation. The model may not contain any time regions, inscriptions, etc. This option results in minimum switching time.

Both

Enables both timed and untimed simulation. You can then simulate with or without time by making appropriate choices in the **General Simulation Options** dialog (**Set** menu). This option results in the maximum switching time. Execution speed depends on the type of execution selected in the **General Simulation Options** dialog.

Design/CPN Menu Reference

Integer

Available only when **With** or **Both** is chosen. Specifies that time is measured as an integer. The number in the accompanying edit box specifies the clock setting at the start of simulation. This number cannot be changed in the simulator.

Real

Available only when **With** or **Both** is chosen. Specifies that time is measured as a real number. The number in the accompanying edit box specifies the clock setting at the start of simulation. This number cannot be changed in the simulator.

Code Segments

With

Causes code segments to be compiled in a manner that requires them to be executed during simulation. This option results in intermediate switching time and slowest execution speed.

Without

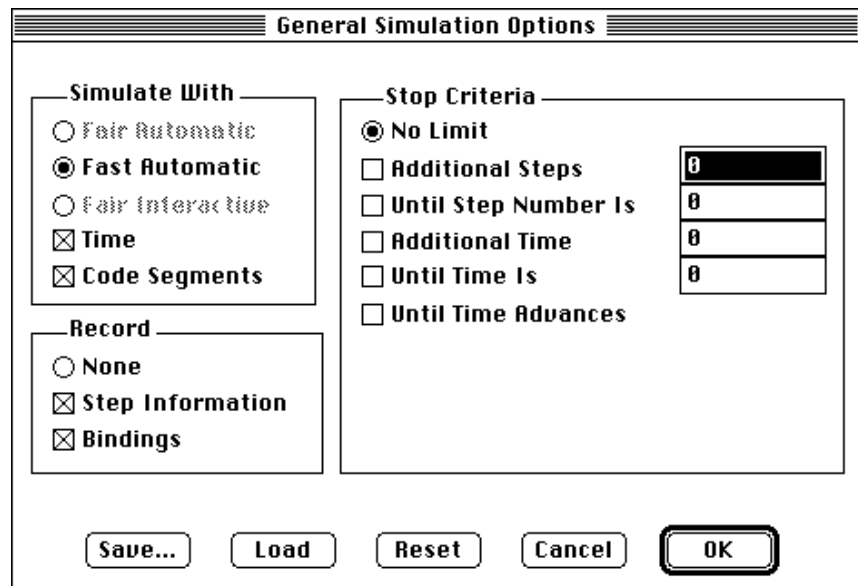
Causes code segments to not be compiled. They therefore cannot be executed during simulation. This option results in minimum switching time and fastest execution speed.

Both

Causes code segments to be compiled, but allows simulation to occur with or without executing them, depending on choices made in the **General Simulation Options** dialog (**Set** menu). This option results in maximum switching time. Execution speed depends on the type of execution selected in the **General Simulation Options** dialog.

General Simulation Options

Specifies the parameters for a simulation run. The settings in this dialog can be changed whenever simulation is not in progress.



The dialog box is titled "General Simulation Options". It contains three main sections: "Simulate With", "Record", and "Stop Criteria".

- Simulate With:**
 - ☐ Fair Automatic
 - ☒ **Fast Automatic**
 - ☐ Fair Interactive
 - ☒ **Time**
 - ☒ **Code Segments**
- Record:**
 - ☐ None
 - ☒ **Step Information**
 - ☒ **Bindings**
- Stop Criteria:**
 - ☒ **No Limit**
 - ☐ Additional Steps: 0
 - ☐ Until Step Number Is: 0
 - ☐ Additional Time: 0
 - ☐ Until Time Is: 0
 - ☐ Until Time Advances

At the bottom are five buttons: "Save...", "Load", "Reset", "Cancel", and "OK".

Simulate With

Specifies the type of simulation. The choices available depend on the options that were in effect in the **Simulation Code Options** dialog (**Set** menu) when the diagram was switched.

Fair Interactive

Specifies fair interactive simulation. This option is available only if **Fair Simulation** or **Both** was chosen in the **Mode** section of the **Simulation Code Options** dialog (**Set** menu).

Fair Automatic

Specifies fair automatic simulation. This option is available only if **Fair Simulation** or **Both** was chosen in the **Mode** section of the **Simulation Code Options** dialog (**Set** menu).

Fast Automatic

Specifies fast automatic simulation. This option is available only if **Fast Simulation** or **Both** was chosen in the **Mode** section of the **Simulation Code Options** dialog (**Set** menu).

Time

If **Both** was chosen in the **Time** section of the **Simulation Code Options** dialog (**Set** menu), choosing this option specifies that simulation will be with time. Otherwise, simulation will be without time.

If **With** was chosen in the **Time** section, this option is automatically chosen and cannot be toggled. If **Without** was chosen in the **Time** section, this option is automatically unchosen and cannot be toggled.

Code Segments

If **Both** was chosen in the **Code Segments** section of the **Simulation Code Options** dialog (**Set** menu), choosing this option specifies that code segments will be executed during simulation. Otherwise, code segments will not be executed during simulation.

If **With** was chosen in the **Code Segments** section, this option is automatically chosen and cannot be toggled. If **Without** was chosen in the **Code Segments** section, this option is automatically unchosen and cannot be toggled.

Stop Criteria

Specifies how simulation is to terminate.

No Limit

Simulation terminates when there are no enabled transitions. If there is always at least one enabled transition, simulation will continue indefinitely. You should therefore be extremely careful in using **No Limit**.

Additional Steps

Simulation stops after execution of the number of steps specified in the value box.

Until Step Number Is

Simulation stops when the step number specified in the value box is reached.

Additional Time

Simulation stops as soon as the number of time units specified in the value box have elapsed. This option is available only if **Time** is chosen in the **Simulate With** section.

Until Time is

Simulation stops as soon as the clock value equals or exceeds the time specified in the value box. This option is available only if **Time** is chosen in the **Simulate With** section.

Until Time Advances

Simulation stops as soon as the clock advances. This option is available only if **Time** is chosen in the **Simulate With** section.

Record

Specifies the recording of simulation data in a report. Recorded data can be written to a file using the **Save Report** dialog (**Sim** menu).

None

No data is recorded. This results in slightly faster simulation.

Step Information

Step information is recorded. For each simulation step, a line is stored whose syntax is:

Step M Transition@(Instance:Page)

For example:

1 I Process@(1:Fnet#1)

This line specifies that in Step 1, while executing in an Interactive mode, a transition named `Process` occurred on Instance 1 of page Fnet#1. If the transition had occurred in Automatic mode, the line would have read:

1 A Process@(1:Fnet#1)

Design/CPN Menu Reference

Bindings

Bindings are recorded. For each simulation step, a line is stored whose syntax is:

```
{variable = value [,variable = value]...}
```

For example:

```
{ordent = Big, staff = Expert}
```

This line specifies that a step occurred in which a CPN variable named `ordent` was bound to the value `Big`, and a CPN variable named `Staff` was bound to the value `Expert`.

Simulation Report Example

The following report shows both step information and bindings for a net in which a transition named `T1` fired alone in three consecutive steps:

```
Simulation Report
1 I T1@(1:New#1)
  {x = 1, y = 3}
2 I T1@(1:New#1)
  {x = 2, y = 1}
3 I T1@(1:New#1)
  {x = 1, y = 3}
```

The first line is a standard header; it is included in every simulation report.

Adding Information to the Report

It is also possible to include information in a report by calling an ML function from the code segment of a transition. The function is:

```
write_report: string -> unit
```

The information contained in `string` will appear immediately before the automatically generated information that describes the firing of the transition. For example, if the code segment for transition `T1` contained the statement:

```
write_report
  ("The time is: " ^^
   (makestring (time())));
```

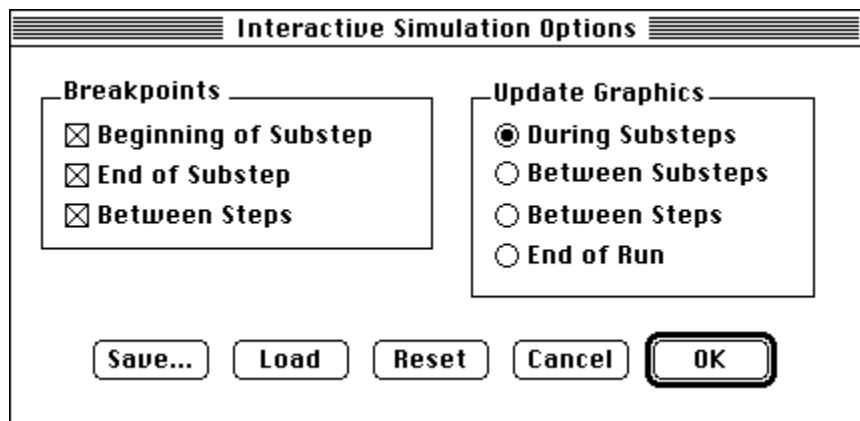
the example report above might look like this:

```
Simulation Report
The time is: 0
```

```
1 I T1@(1:New#1)
   {x = 1, y = 3}
The time is: 5
2 I T1@(1:New#1)
   {x = 2, y = 1}
The time is: 7
3 I T1@(1:New#1)
   {x = 1, y = 3}
```

Interactive Simulation Options

Specifies the observation methods for a step. **Interactive Simulation Options** affects only Fair Interactive simulation. The settings can be changed whenever simulation is not in progress.



Breakpoints

Specifies when the simulation is to pause.

Update Graphics

Specifies how the graphics will be updated during a simulation

During Substeps

Updates all existing simulation regions. These simulation regions include:

- Input and output tokens
- Enabled and occurring transitions
- Markings

Design/CPN Menu Reference

Between Substeps

Updates simulation regions only after each substep. This eliminates the display of input and output tokens. Simulation regions displayed include:

- Enabled and occurring transitions
- Markings

Between Steps

Updates simulation regions only after each step. This eliminates the display of input and output tokens and well as occurring transitions. Simulation regions displayed include:

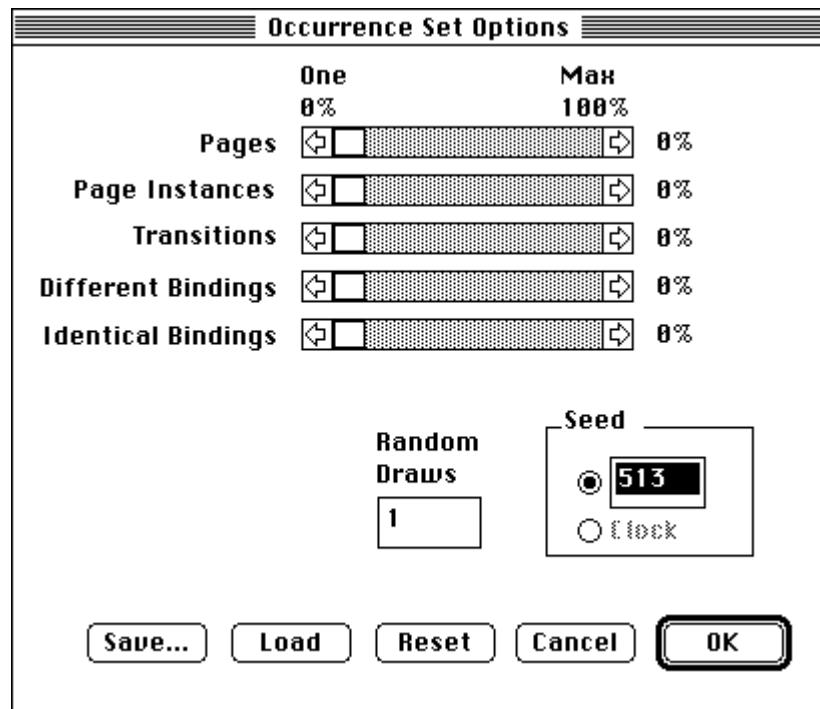
- Enabled transitions
- Markings

End of Run

Updates simulation regions only the end of a simulation run. This eliminates the display of simulation regions.

Occurrence Set Options

Specifies the way how the system is to calculate occurrence sets.



Under X-Windows, edit boxes appear instead of scroll bars.

Settings

Pages

Each calculated occurrence set contains binding elements from at least one page. The first contributing page is found by a random drawing among all pages that have enabled transitions.

When the binding elements for the first page have been calculated, a random number between 1 and 100 is generated. If this number is smaller than or equal to the specified percentage for pages (in **Occurrence Set Options**), a second page is allowed to contribute to the occurrence set (if a page with enabled bindings can be found). The second contributing page is found by a random drawing among all pages that have enabled transitions and have not yet contributed. All pages undergo the same process, until simulation comes to a halt.

A specified percentage of 0% means that you only want a single page to contribute to the occurrence set. A specified percentage of 100% means that you want a maximal set of pages to contribute to the occurrence set (in this context, maximal means that the set cannot be extended, not necessarily that it is the largest one).

Page Instances

Each contributing page has binding elements from at least one page instance. The number and identities of the contributing page instances are determined by a sequence of numbers that are randomly drawn and analogous to the one for pages. However, the drawn numbers are among the enabled page instances of the contributing page and the applied percentage is the one for page instances.

Transitions

Each contributing page instance has binding elements from at least one transition. The number and identities of the contributing transitions are determined by a sequence of numbers that are randomly drawn and analogous to the one for pages. However, the drawn numbers are among the enabled transitions of the contributing page instance and the applied percentage is the one for transitions.

Different Bindings

Each contributing transition has at least one contributing binding. The number and identities of the contributing bindings are determined by a sequence of numbers that are randomly drawn and analogous to the one for pages. However, the drawn numbers are among the potential bindings of the contributing transition and the applied percentage is the one for different bindings.

Identical Bindings

Each contributing binding is always included at least once in the occurrence set. The number of times it is included is determined by a sequence of numbers that are randomly drawn and analogous to the one for pages. However, the drawn numbers are between 1 and 100 and the applied percentage is the one for identical bindings.

Occurrence Rule

Some applications of CP-nets apply the *maximal occurrence rule*. To achieve this, set all five percentages to 100%.

Some applications of CP-nets do not allow bindings to occur concurrently with themselves. To achieve this, set the percentage for identical bindings to 0%.

Mutual Influence

It should be noted, that the drawing sequences described above influence each other because they all compete for the same set of to-

kens. Specifying high-percentages for identical and different bindings makes it easier for the first transitions to get many bindings but it may make it difficult for the following ones because many of the tokens have already become committed. A similar relationship exists between the other percentages.

Random Generator

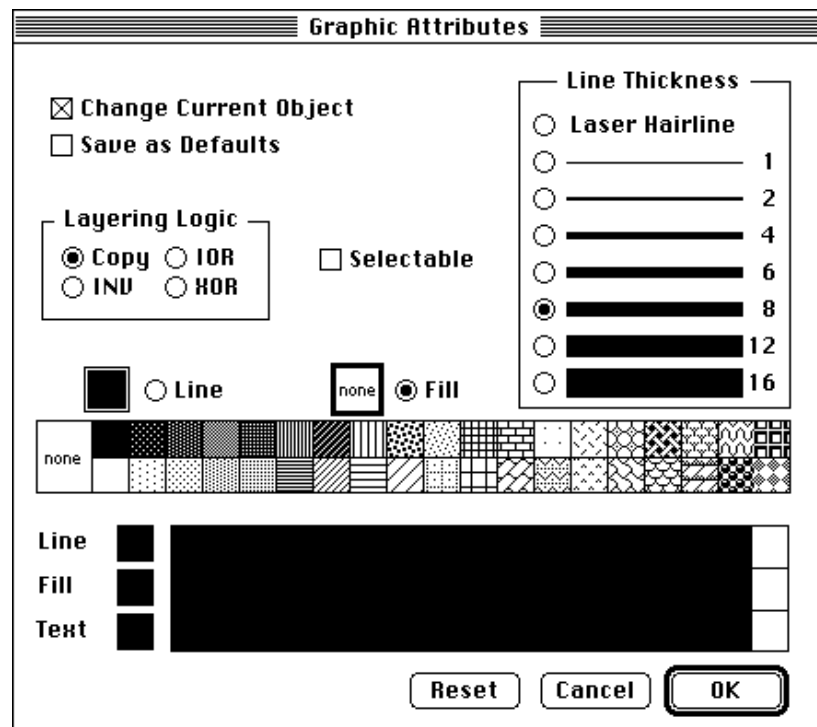
The numbers are randomly drawn by means of a conventional arithmetic random number generator, which has a seed value determining the generated sequence of numbers. In **Occurrence Set Options** you specify the seed value directly.

The seed is set each time the CPN Simulator is entered and each time **Initial State** or **Occurrence Set Options** is invoked. This means that you can repeat a simulation by choosing the same seed and the same five percentages.

Transition Feedback Options

Specifies the properties of the transition feedback region that the simulator uses to highlight an enabled or occurring transition.

The dialog is the same used for **Graphic Attributes (Set menu)**:

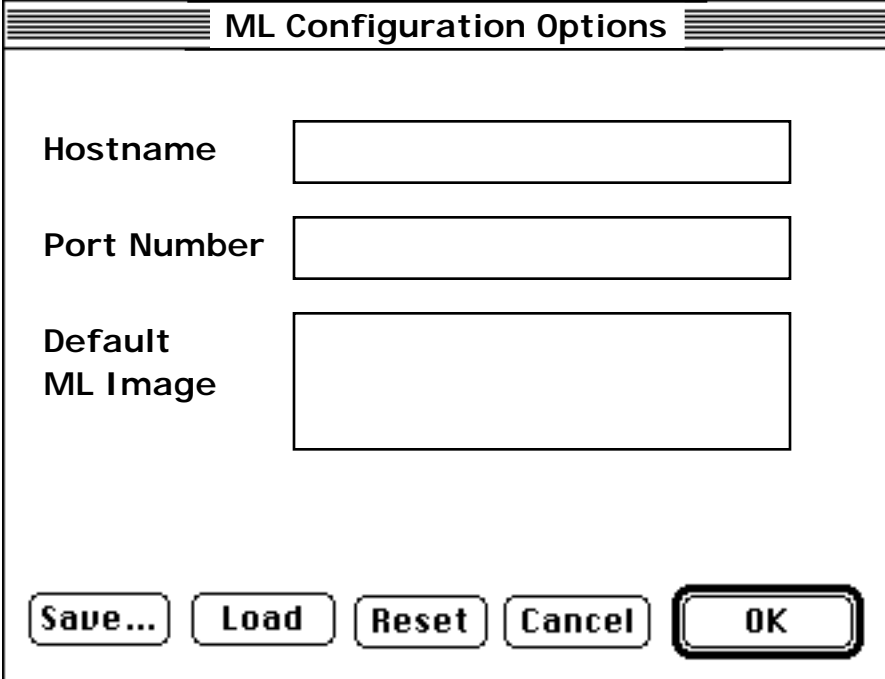


Design/CPN Menu Reference

The line thickness specified will be used to highlight enabled transitions. This thickness will be doubled to highlight transitions that are occurring.

ML Configuration Options

Design/CPN needs some information about the ML interpreter it is to use. Without this information, it cannot start the interpreter, and hence cannot syntax check or execute a diagram. Use **ML Configuration Options** to specify the necessary information.

The image shows a dialog box titled "ML Configuration Options". It has a title bar with a standard window icon on the left and a close button on the right. The main area contains three labels with corresponding input fields: "Hostname" with a single-line text box, "Port Number" with a single-line text box, and "Default ML Image" with a larger multi-line text box. At the bottom of the dialog, there are five buttons: "Save...", "Load", "Reset", "Cancel", and "OK". The "OK" button is highlighted with a double border.

The information needed is:

- **Hostname:** Where the ML interpreter is to be run.
- **Port number:** Which port is used by the daemon is to run the interpreter.
- **ML image:** Where the interpreter is located in the file system.

Design/CPN keeps this information as system defaults. In order for a particular diagram to use it, the information must be present in its diagram defaults. If any of the information is missing or incorrect:

Enter correct values in the **Hostname**, **Port number**, and **ML image** sections.

See the Design/CPN Installation Notes for information on determining these values.

Setting the System Defaults

If you want the new values to become system defaults:

Click **Save**.

A confirmation dialog appears.

Click **OK** in the confirmation dialog.

The values in the **ML Configuration Options** dialog are now the system defaults.

Setting the Diagram Defaults

To make the new values the diagram defaults:

Click **OK** in the **ML Configuration Options** dialog.

The values in the **ML Configuration Options** dialog are now the diagram defaults. The dialog closes.

Copying the System Defaults to the Diagram

To copy the system defaults to the diagram defaults:

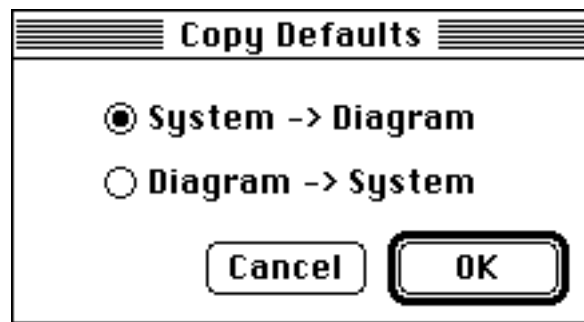
Click **Load**.

Click **OK**.

The system defaults are now the diagram defaults also. This step is necessary whenever a diagram that was created on some other system is syntax checked or executed for the first time.

Copy Defaults

This command allows you to copy the entire set of system defaults into the diagram defaults of the current diagram, and vice versa. Each system attribute is copied into the corresponding diagram attribute, or vice versa. Each system option is copied into the corresponding option value, or vice versa.



Chapter 25

Makeup Menu Commands

Select	Alt+1
Drag	Alt+2
Displace	Alt+3
Adjust	Alt+4
Fit to Text	Alt+5
Change Shape	
Duplicate Node	Alt+ /
Move Node	
Merge Node	
Hide Regions...	Alt+-
Show Regions...	Alt+=
Bring Forward	
Parent Object	
Child Object	
Next Object	
Previous Object	

The **Makeup** menu commands manipulate and alter graphic objects.

Select

Selects objects by temporarily hiding other objects.

When objects are layered, closely spaced, or contained within other objects, they may be difficult to select with the graphic tool.

Selecting Visible Objects

Moving the pointer tool across an object causes its borders flash. Clicking on a flashing object makes it become the current object.

Selecting Hidden Objects

While the obscuring object is flashing, press SPACEBAR to make it invisible. Continue to hide objects until the desired object is visible. Select the object by clicking the mouse. All the hidden objects are restored.

Clicking anywhere in the active window restores all the temporarily hidden objects.

While **Select** is active, automatic scrolling is in effect.

Note: The command cannot be applied when the system is in group mode.

Drag

Repositions objects.

The drag tool  indicates that the system is in drag mode.

Use this tool to move the current object or group without placing the pointer inside a selected object. This is especially useful for moving small objects and objects inside other objects.

Dragging Text

If text is turned on, text can be moved as a block within its object. To begin a drag text, the drag tool must be inside the graphic object containing the text.

Dragging the Endpoints of a Connector

Endpoint Source

When the source object of the selected connector is an endpoint region, the command selects this region and enters drag mode.

Node Source

When the source object is a node, **Drag:**

- Creates a new endpoint region of the node.
- Positions the region at the connector endpoint.
- Attaches the connector to the region.
- Selects the region.
- Enters drag mode.

The new endpoint region is a small box, has None as line and fill patterns, and is selectable.

Target

When **OPTION** is pressed, the command works on the target instead of the source.

Endpoint Target

Pressing one of the endpoint handles of a selected connector with **SHIFT** depressed has the same effect as described above in permitting the selection and drag of the source/target object if it is a region.

Node Target

Otherwise, it creates an endpoint region which can be dragged.

Terminating Drag Mode

The mode is terminated by:

- Pressing **ESC**.
- Selecting another command.

Displace

Repositions objects by moving the current node or group the same distance (horizontally and vertically) as the last drag or alignment of an object or group.


SHIFT + menu command reverses the last displacement.

Note: The command cannot be applied to groups of regions.

Adjust

Resizes the current node without moving its center.

Resizing in Two Dimensions

The hand pointer  appears at the lower right corner of the current object. Drag the hand to resize the object in either dimension. Click the mouse to complete the operation.

Resizing in One Dimension

To resize one dimension while retaining the object's other dimension and its center, hold down the OPTION key.

Drag horizontally, with the vertical dimension locked.

Drag vertically and the horizontal dimension is locked.

Click the mouse to complete the operation.

Resizing Regions and Labels

When a label or a key region is selected, the command has no effect unless the object is part of a group. In such a case, the command does not affect the object itself but only its descendants, which get a new size and position.

When a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Fit to Text

Adjusts the boundary of the selected graphic object to fit its text.

Width

The new width is determined by the longest line of text plus a margin. Text is justified according to the new width.

Height

The height of the object is adjusted so that all text is visible with a minimum of empty space above and below. This means that it has no effect on objects that have no text.

Word Wrap

The box width set before choosing this command determines word wraparound. Resize the width to fit more characters on the line.

Labels and Key Regions

When a label or a key region is selected, the command cannot be applied, unless the object is part of a group. In such a case, the command does not affect the object itself, but its descendants change size and position slightly.

Change Shape

Changes the size and shape of objects.

Change Shape changes the shape and size of the current node or group to match that of any other node on the page.

Shape Changing

The status bar prompts the selection of a node or region as a model for the shape and size change.

Placing the pointer within the selected object causes its borders flash.

Clicking on the flashing object completes the change.

Design/CPN Menu Reference

Region Changing

Note that when a selected group of regions contains ancestors/descendants of each other, some regions may change more than once.

Duplicate Node

Duplicates objects on the current page without altering their object type.

Duplicate Node places a copy of the group or current node on the page slightly below and to the right of the original. The duplicate becomes the current node.

Duplicating Regions and Connectors

Regions and connectors are duplicated, along with the corresponding nodes as follows:

- A region is duplicated if the corresponding object is duplicated.
- A connector is duplicated if either the source or the destination node is duplicated.

This is unlike the Copy and Paste commands, which only handle internal connectors.

Text Mode Impact on Duplication

Text mode may be turned on or off without affecting duplication.

Effect on Hierarchy Objects

Duplicate Node makes it possible to reproduce:

- Substitution transitions without losing their hierarchy region. If some of the socket places are not duplicated along with the substitution transition, the corresponding lines describing the port assignment will be removed from the hierarchy region.
- Fusion places, without losing the fusion region.
- Port/exit places, without losing the port region.

Move Node

Moves the current node or group to another page in the diagram.

Moving Nodes

The hierarchy page window appears and the status bar prompts:
Click on page to receive object.

Move Node returns to the current page after the target page has been selected.

Effect on Regions and Connectors

Regions and connectors are moved, along with the corresponding nodes, in the usual way:

- A region is moved if the corresponding node is moved.
- A connector is moved if both the source and destination nodes are moved. If only one of these nodes is moved, the connector is deleted.

Effect on Hierarchy Objects

Substitution Transitions

Moving substitution transitions rearranges the hierarchical structure of pages in a diagram. The hierarchy region is retained.

If some of the socket places are not moved along with the substitution transition, the corresponding lines will be removed from the hierarchy region.

The system checks whether this move introduces cycles into the substitution hierarchy.

Fusion Places

The fusion region is retained.

This may create new fusion sets at the target page and delete fusion sets at the source page.

Moreover, the fusion type may change (from page to instance, or vice versa).

Design/CPN Menu Reference

Port Places

If a port place is moved to another page, the system checks to see if there are any socket places attached to it. If there are, the corresponding lines are removed from the port assignment information (if any) in the hierarchy region.

Restrictions

Page nodes cannot be moved.

Merge Node

Merges nodes into a single node by combining a node or group into a single target node.

Specify options for the merge with **Merge Options** (**Set** menu).

Restrictions

Each kind of CPN node — places, transitions and declaration nodes — can only be merged with objects of the same kind.

Merge cannot be applied when the current selection contains substitution transition, or fusion, socket, or port/exit places, or page nodes.

Regions

Regions of merged nodes become regions of the target node.

When some of the source nodes have CPN regions of a kind already existing at the target node, the CPN regions are merged or deleted with the target region, depending on the settings in **Merge Options**.

Connectors

Any connectors attached to the node or group to be merged are attached to the target node unless they are duplicate connectors. A duplicate connector is one identical in orientation and node endpoints to a target node connector.

Targets

A member of a group to be merged may also be the target node.

CPN nodes can be target nodes, where some of the source nodes are auxiliary nodes.

Substitution transition, or fusion, socket, and port/exit places, and page nodes can be used as target nodes.

Hide Regions

Hides descendants of selected objects , making them non-visible and non-selectable.

Nodes and Connectors

Hides all the descendants of the selected objects.

Regions

Hides the selected regions, together with all their descendants.

The hidden regions can, however, still be selected by means of the arrow keys; clicking one of the selection handles, the region and its descendants reappear.

Show Regions

Shows descendants of selected objects, making the descendants visible and selectable.

Bring Forward

Changes the object order of objects on a page.

Reordering Object Layering

1. Select the object to bring forward.
2. Choose **Bring Forward**.

Design/CPN Menu Reference

3. Place the pointer on the object to put one layer below the selected object. The object chosen to underlie the selected object flashes.
4. Click to bring the selected object forward .

To travel through the layering order sequentially, use **Next Object** (**Makeup** menu) and **Previous Object** (**Makeup** menu).

Restrictions

The command is not yet implemented for groups of regions.

Parent Object

Selects the parent of an object.

Parent Object moves one level higher in the diagram hierarchy to the parent of the current page, node, or text, making the parent active. If the selection has no parent, **Parent Object** is dimmed.

Text Pointers

If text is turned on and **Parent Object** is highlighted in the menu, this command goes to the node containing the text pointer in an error mode.

If the tracer is on another page, that page becomes active and the text tracer is highlighted.

Child Object

Selects a child of an object.

If the current object, text, or page is a parent, this command moves one level lower in the hierarchy and selects the child. **Child Object** is dimmed if the current selection is not a parent.

Substitution Transitions

Goes to the node's subpage. Double-clicking in the substitution transition has the same effect.

Regions

Goes to the object's region.

If an object has more than one region, **Child Object** goes to the first created region of this parent, unless the **Bring Forward** (**Makeup** menu) command has changed the layering order.

Double-clicking in the parent node has the same effect.

If the selected object is both a substitution transition and a parent of a region, **Child Object** goes to the subpage.

Text Pointers

If text is turned on and the insertion point is in a text pointer in the error box, choosing the **Child Object** command goes to the object with the error.

Next Object

Selects the object one layer above the current object.

This process is repeated with each selection of **Next Object** until the top layer of the stack of objects is reached. **Previous Object** (**Makeup** menu) reverses the process.

Previous Object

Selects the previous object (or text pointer).

Previous Object selects the object one layer below the current object. This process is repeated with each selection of **Previous Object** until the bottom layer of the stack is reached. **Next Object** (**Makeup** menu) reverses the process.

Chapter 26

Page Menu Commands

New Page Open Page Close Page	Alt+Y
Scroll Blowup Reduce Cleanup	Alt+ \ Alt+ ` Alt+ ,
Redraw Hierarchy...	

The **Page** menu commands affect the appearance, labeling, and page numbering of the diagram. Other commands move around the pages and hierarchies of the diagram by means of the page hierarchy window.

New Page

Creates new pages.

- The attributes of the page (size and name) are determined by the diagram defaults for pages.
- **New Page** places a page node, which represents the new page, on the hierarchy page .

Open Page

Opens pages.

- When the page hierarchy window is active, **Open Page** opens the selected set of pages and activates one of them.
- When the page hierarchy window is not active, **Open Page** makes the page hierarchy window open and active.

Close Page

Closes the selected set of pages.

If all pages are closed, **Close Page** automatically opens the hierarchy page.

Scroll

Scrolls the selected set of pages so that the upper left corner of the page border coincides with the upper left border of the window.

It is also possible to **Scroll** closed pages.

Blowup

Enlarges the page scale for a selected set of pages.

Each time **Blowup** is used, it enlarges the active page to twice its present size up to the maximum of 400 percent of normal. The page scale is indicated in the status bar.

Some degrading of the screen image may occur. The actual diagram and the printed diagram are not degraded by **Blowup** or **Reduce**. Use the **Cleanup** (**Page** menu) command to restore on-screen resolution after using **Blowup**.

Blowup can also be used to enlarge closed pages.

Reduce

Reduces the page scale for the selected set of pages.

Each time **Reduce** is used, the active page is reduced to one half its present size to the minimum of 25 percent of normal. **Reduce** reverses the effect of **Blowup**, and like **Blowup**, may affect screen resolution.

Reduce can be used to reduce the page scale of closed pages.

Cleanup

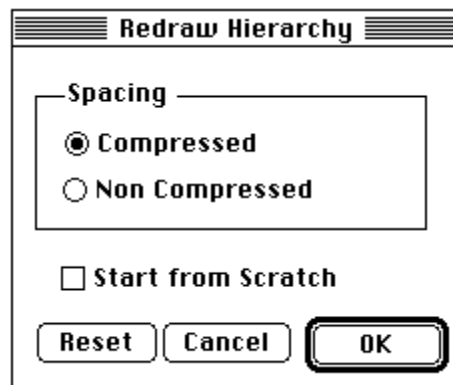
Cleans up a selected set of pages.

Cleanup redraws the diagram on the active page, restoring the on-screen resolution of objects and text distorted by operations such as **Blowup** and **Reduce**.

Cleanup can be used on closed pages.

Redraw Hierarchy

Redraws the hierarchy page. If necessary, the hierarchy page is made active and remains active.



Chapter 27

Group Menu Commands

Enter Group Mode	Alt+G
Select All Nodes Select All Regions Select All Connectors Select Fusion Set	Alt+A


The **Group** menu commands group objects according to their object types. The objects within the group can be changed or manipulated as a single node. A group is indicated by a gray outline around all its member nodes that are not regions.

Enter/Leave Group Mode

Enters or leaves group mode.



Enter Group Mode

Restores the last group that existed on a page if the page has not changed.

The pointer changes to the group pointer, .

Leave Group Mode

Leaves the current group and makes current the last object selected before the group was formed.

The pointer changes to the graphic tool, , or to the text tool, , if text is turned on.

Clicking outside a group member also leaves group mode.

Select All Nodes

Selects all nodes on the active page whether the system is in text or group mode.

Select All Nodes is available when text is turned off and no group is active. It forms a group of all the nodes on the current page.

Select All Regions

Selects all regions on the active page.

Select All Connectors

Selects all connectors on the active page.

Select Fusion Set

Selects all members on the active page of the fusion set to which the selected object belongs.

Select Fusion Set can only be invoked when a fusion place, or a descendant of a fusion place, is selected. In the latter case, the command works as if the corresponding ancestor node were selected.

Chapter 28

Text Menu Commands

Enter Text Mode	Alt+T
Find... Find Next Find Beginning	Alt+F Alt+N
Select All Text Select to... Select to Bracket	Alt+W

The **Text** menu commands manipulate text objects, permitting the creation, editing, formatting, selecting, and searching of text.


Graphical objects can have text associated with them.

Editing text in Design/CPN follows the standard editing procedures. Text attributes (font, style, size, and justification) can not be mixed within a single graphic object.

Enter/Leave Text Mode

Enters or exits text mode.

Enter Text Mode

Changes the pointer to the text tool, . An insertion point appears in the current object.


Use text mode to create, edit, select, and search text.

Leave Text Mode

Leaves text mode. The current object is now indicated by selection handles and the pointer changes to the graphic tool. Text can no longer be edited, selected, or searched.

The status bar indicates whether text is on or off.

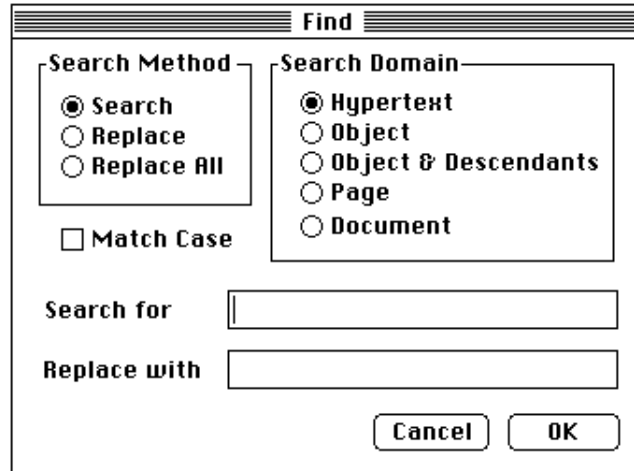
Group Text Mode

When the system is both in group mode and text mode, the cursor contains “GT” rather than only “G”: .

In group mode, the menu item does not toggle; thus, it always reads **Enter Text Mode**, even when it invokes a **Leave Text Mode**. However, the status bar and the group cursor indicate the current state.

Find

Performs searches and replacements in texts:



Up to 255 characters may be searched for at each invocation of **Find**. Text matches are highlighted in the current object and text mode is turned on.

Search Method

Search

Searches for the first occurrence of the search text in the search domain.

Replace

Replaces the first occurrence of the search text in the search domain.

Replace All

Replaces all occurrences of the search text in the search domain.

Match Case

Causes the search to be case-sensitive.

Design/CPN Menu Reference

Search Domain

Hypertext

An object and its substructure (regions and sub-regions) down the chain of text pointers.

Current Object

The current object or group of objects.

Pages

All open pages.

Document

The entire model.

Searching and Replacing in a Group

Text is searched from the insertion point, scrolled, and replaced in every group member.

Choosing **Leave Text Mode** and selecting a searched object leaves matched selections highlighted.

Find Next

Repeats the last invocation of **Find**.

The command continues to search for matches of text entered in the last **Find** operation. This search begins at the insertion point.

Find Beginning

Moves text cursors to the beginning of the corresponding texts.

The command scrolls to the beginning of the current object's text and places the insertion point in front of the first character.

If a group is active, **Find Beginning** scrolls to the beginning of all group members.

Select All Text

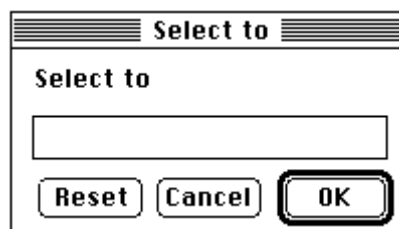
Selects all text in an object and enters text mode.

Select All Text is available whether text is turned on or off.

All the text in the current node is highlighted and can be edited, cut, copied, pasted, or cleared.

Select to

Extends the text selection, until a specified text string is found:



Select to Bracket

Extends the text selection until a bracket is found.

When the insertion point is placed immediately before an open bracket, this command finds and selects text up to the matching close bracket. Design/CPN recognizes the delimiters listed in the **Text Options** dialog.

Select to Bracket is especially useful for selecting and editing text that is hidden from view.

Case Sensitivity

When using **Select to Bracket**, the text entries must match the case (upper/lower) of the searched text.

Chapter 29

Align Menu Commands

Horizontal Vertical Center Position...	Alt+H Alt+J
Horizontal Spread Vertical Spread Circular Spread Between Projection	Alt+[Alt+]
Left to Left Left to Right Right to Left Right to Right Top to Top Top to Bottom Bottom to Top Bottom to Bottom	

The **Align** menu commands reposition the currently selected node or group by aligning it to one or two reference objects.

Aligning is a two-step process: first select the command and then the reference object(s). A click of the mouse on the reference object accomplishes the alignment.

All alignments can be undone or redone with **Undo/Redo** (**Edit** menu).

Horizontal

Performs a horizontal alignment.

Horizontal aligns the center of the current object along a horizontal axis drawn through the center of the reference object.

Group Alignment

Shift

With SHIFT pressed in group mode, the first element in the group is aligned in the usual way.

All the other objects in the group remain in the same position relative to the first object in the group.

Option

With OPTION pressed in group mode, the group is aligned as a whole: the smallest box covering all objects in the group is aligned.

All the objects in the group keep their positions relative to this imaginary box.

Vertical

Performs a vertical alignment.

Vertical aligns the center of the current object along a vertical axis drawn through the center of the reference object.

Group Alignment

See **Horizontal** (**Align** menu).

Center

Performs a center alignment.

Center places the center of the current object over the center of the reference object.

Group Alignment

See **Horizontal** (**Align** menu).

Position

Repositions objects by specifying their relative position, compared to another object.

The position can be given either in Cartesian or polar coordinates.

Group

Only non-conflicting values are shown in a group.

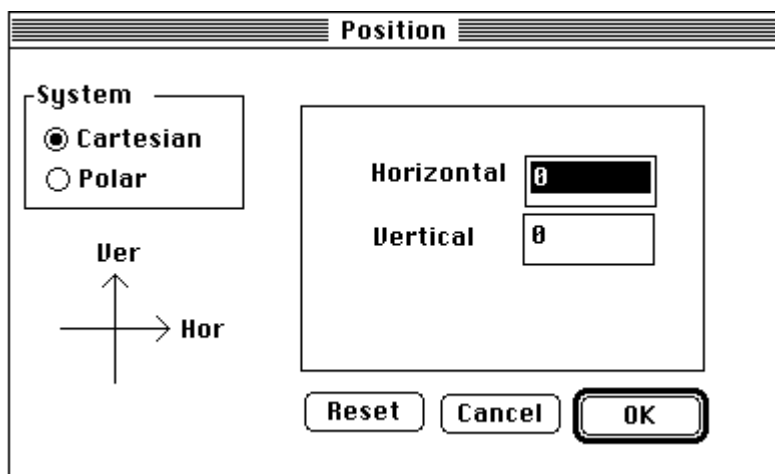
Group Alignment

See **Horizontal** (**Align** menu).

Dialogs

The **Cartesian Coordinates** dialog appears when **Position** is invoked. Click the **Polar** button to select the **Polar Coordinates** dialog.

Cartesian Coordinates



Polar Coordinates

The image shows a dialog box titled "Position". It contains two main sections: "System" and "Angle". In the "System" section, there are two radio buttons: "Cartesian" and "Polar", with "Polar" being selected. Below this is a circular diagram representing a polar coordinate system with axes and labels for 0°, 90°, 180°, and 270°. In the "Angle" section, there are two radio buttons: "Absolute" and "Relative", with "Absolute" being selected. Next to the "Angle" section is a small input field containing the number "0". Below the "Angle" section is a "Distance" label followed by a larger input field also containing the number "0". At the bottom of the dialog box are three buttons: "Reset", "Cancel", and "OK".

Horizontal Spread

Performs a horizontal spread of a selected group of nodes.

Targets

Horizontal Spread can be applied to groups of nodes, single nodes or single regions, and to a group of regions.

To Perform the Horizontal Spread

Choose two reference objects (which may or may not be part of the current selection and which may or may not be identical).

Effect of the Horizontal Spread

The spread changes the horizontal position of the currently selected objects in such a way that the:

- Leftmost object becomes vertically aligned with the leftmost reference object.
- Rightmost object becomes vertically aligned with the rightmost reference object.

The spread is always performed horizontally, even when the reference objects are close to each other.

Reference Objects Far Apart

Positions the selected objects evenly, without overlapping.

Reference Objects Close Together

Positions the selected objects such that their centers become (horizontally) equidistant, but the objects may overlap.

Vertical Spread

Similar to the **Horizontal Spread** in features and functions.

Circular Spread

Spreads a group of objects equidistantly on a circle.

To spread the objects:

1. Select an object indicating the center of the circle
2. Select another object to specify the circle radius, which is the distance between the two objects' centers.

Between

Places the center of the current object at the midpoint of an imaginary line drawn between the centers of the two reference objects.

Group Alignment

See **Horizontal** (**Align** menu).

Projection

Places the current object at one end of an imaginary line, whose first endpoint is the center of the first reference object, and whose midpoint is the center of the second reference point.

Group Alignment

See **Horizontal (Align menu)**.

Left to Left

Performs a left-to-left alignment.

Group Alignment

See **Horizontal (Align menu)**.

Left to Right

Performs a left-to-right alignment.

Group Alignment

See **Horizontal (Align menu)**.

Right to Left

Performs a right-to-left alignment.

Group Alignment

See **Horizontal (Align menu)**.

Right to Right

Performs a right-to-right alignment.

Group Alignment

See **Horizontal (Align menu)**.

Top to Top

Performs a top-to-top alignment.

Group Alignment

See **Horizontal (Align menu)**.

Top to Bottom

Performs a top-to-bottom alignment.

Group Alignment

See **Horizontal (Align menu)**.

Bottom to Top

Performs a bottom-to-top alignment.

Group Alignment

See **Horizontal (Align menu)**.

Bottom to Bottom

Performs a bottom-to-bottom alignment.

Group Alignment

See **Horizontal (Align menu)**.

PART 3

CPN ML Reference

Chapter 30

Introduction to CPN ML

Design/CPN uses an extension of Standard ML called *CPN ML* for:

- CP net inscriptions - initial markings, arc expressions and guards.
- Internal algorithms that calculate the enabling and occurrence of bindings.
- Transition code segments.

Standard ML Features

Standard ML includes the following features that support the execution of CP nets:

- **Functional language** - provides a natural and general way to specify the arc expressions and guards.

Moreover, Standard ML makes it very easy to translate a CP graph into a CP matrix because it supports lambda notation.

Finally, it simplifies the calculation of place invariants for CP nets by applying a set of reduction rules formulated in terms of standard operations on functions.

- **Interactive language** - enables users to change individual net inscriptions without having to reevaluate all of them.
- **Strong typing** - allows Design/CPN to detect colorset errors during syntax checking.
- **Polymorphic type system** - permits operations and functions to be used by different colorsets instead of having to declare a version for each colorset.
- **Overloading** - permits operations and functions to be used for several different purposes. For example, the symbol "+"

is used for addition in an integer or real colorset as well as for the addition of multisets and function.

The polymorphic type system and operator/function overloading facilitates the use of standard mathematical notation for net inscriptions.

- **Static scoping** - provides a consistent and stable runtime environment. Since the environment is fixed at runtime, it is easier to understand the functionality of large code segments.

In addition, internal routines are more efficient because identifiers can be bound when the functions are declared instead of when they are evaluated.

- **Exception mechanism** - handles runtime errors. When an error occurs, an exception is raised and the ML system looks for an exception handler. If no system or user-defined exception handler is found, the exception is reported as an error.

CPN ML Extensions to Standard ML

CPN ML extends Standard ML by the addition of three capabilities:

- CPN datatypes, called *colorsets*.
- CPN variables.
- Reference variables with a specified scope.

Colorsets

Colorsets are extensible. This allows CP nets to more accurately reflect the system structures being modeled. Colorsets are described in Chapter 32.

CPN Variables

CPN variables provide CP net components, such as arc inscriptions and guards, with the ability to reference different values. They are characterized by:

- Scope is local to transitions.

- Multiple possible simultaneous bindings on different transitions.
- Extent is the firing of a particular transition.

CPN Variables are described in Chapter 34.

Reference Variables

A page may have many instances. These instances have identical code segments and thus they usually work on the same set of variables.

However, sometimes it is necessary (or at least convenient) to declare local variables where each page instance has its own private version of the variable called a *reference variable*.

CPN ML allows you to declare global, page, and instance reference variables. Chapter 35, “Reference Variables,” describes both CPN reference variables and `val`-defined reference variables.

Declaration Nodes

All colorset, CPN variable, and value declarations (with the exception of value declarations within `let` structures) must be declared in a declaration node.

This is a departure from Standard ML requirements. It means that a value identifier cannot be redefined during the execution of the CP net.

Each syntactically correct CPN diagram can contain exactly one global declaration node and at most one temporary declaration node. In addition, each page can contain at most one local declaration node. Each of the three types of nodes contains *declarations* of items that can be used as *inscriptions* on the diagram.

CPN ML Restrictions and Modifications

Standard ML provides many more capabilities than are actually needed to support CPN modeling, and a few conventions that are not appropriate to the needs of Design/CPN. CPN ML does not implement the unnecessary capabilities, and changes the inappropriate conventions, as described in this section.

CPN ML Reference

Abstract Datatypes

Colorsets cannot be defined in terms of abstract datatypes.

Wildcards

Wildcards are not supported.

Modules

Modules are not relevant to CP Nets and are not supported.

Side Effects

Side effects are only allowed in code segments.

@ Operator (List Concatenation Operator)

In timed simulation CPN ML redefines the @ operator to append time stamps. The list concatenation operator becomes ^^ (double hat).

Conventions

Syntax descriptions

Format

- 1) Format line - The actual form to use when writing a specific operator, function, or constant.

The font is courier.

Words and symbols that must be entered literally are shown in **bold**.

Optional components are shown with angle brackets. As with the required portion, literal words to be entered are shown in **bold**.

In all cases the form terminates in a semicolon, even though sometimes the Design/CPN context does not require it. For example, initial markings do not terminate in a semicolon.

- 2) Description line - Shows the colorset of the constant, or the domain and range colorset for the function or operator.

If any colorset may be used, then *type variables* are used to indicate this fact. Each type variable, usually a lowercase a, is preceded by either a single quote or a double quote.

The *single quote* indicates that any colorset may be used.

The *double quote* indicates that any colorset for which the equality operation is defined may be used. In practice this means that any colorset may be used since currently all colorsets have an equality operation defined for them.

If either the domain or the range is the Cartesian product of two colorsets, then it is shown as:

$$(\text{colorset}_1 \bullet \text{colorset}_2)$$

Examples

Examples provide one or all of the following:

- 1) Declarations, if any, required to understand the example
- 2) Expression(s) actually entered, preceded by an ML> prompt.
- 3) The value returned by the system is in the form returned by Design/CPN after evaluation, preceded by a single >.

Testing Examples

To test non-CPN inscription examples:

- 1) If there are any declarations, create a global or temporary declarations box with the declarations required and then syntax check it with the **Syntax Check** command.
- 2) Create an aux box with the **box** command (**Aux** menu).
- 3) Type in the expressions exactly as shown in the example.
- 4) Select the expressions and evaluate it with the **ML Evaluate** command (**Aux** menu).
- 5) The result is shown in a separate evaluate box and should match the results shown in the example.

To test CPN inscription examples:

See the User Guide for CP net creation techniques.

Chapter 31

Identifiers

CPN ML identifiers are alphanumeric sequences of letters, digits, primes (apostrophe), and underscores starting with a letter.

They are used as the names for:

- Colorsets (Chapter 32)
- Record colorset field labels (Chapter 32)
- Value constructors (Chapter 33)
- CPN variables (Chapter 34)
- Reference variables (Chapter 35)
- Operators and function symbols (Chapters 36 & 38)

Examples

These identifiers are legal:

SmallOrder

e3'f4

big_bertha

These are illegal, because they do not begin with a letter:

3e4f

_New_Id

Reserved Words

Reserved words are words that have a predefined meaning for CPN ML. They may not be used as identifiers:

```
abstype action all and andalso as bool
by case color colorset datatype declare do
else end exception fn fun globref guard
handle if in index infix infixr input
instref int let list local ms nonfix of
op open orelse output pageref product
raise real rec record ref same start
string subset then time TIME tms type unit
untimed val value var while with ( ) [
] { } , : := ; .. ... ` | || ? @ ! = => -> _
#
```

Identifier Duplication

The syntax check in Design/CPN will usually not allow CPN ML identifiers to be identical to each other. However, there are exceptions:

- **Record labels** - allowed to be identical to each other if they appear in different colorset declarations. Record labels are also allowed to be identical to other kinds of CPN ML identifiers, with the exception of CPN variables. See Chapter 32 for a discussion of colorsets.
- **Page/instance reference variables** - allowed to have identical identifiers if their scopes do not overlap (they are declared on different pages).

Page/instance reference variables are also allowed to have the same name as global reference variables: the page/instance reference variable overwrites the global reference variable at the page where the page/instance reference variable is declared. See Chapter 35 for a discussion of reference variables.

Predeclared Identifiers

In addition to the reserved words which may not be used as identifiers, Design/CPN uses a number of predeclared identifiers:

- Predeclared constants, operations and functions.

- Predeclared exceptions (all of these begin with “CPN' ” or end with an apostrophe ('), also called a prime, and the name of the colorset).
- Constants, variables and functions in the internal algorithms (all of these begin with “CPN' ”).
- Internal representation of multisets and bindings (applies the identifiers “!!”, “!!!”, “\$”, “\$\$” and “\$\$\$”).

All the predeclared identifiers are described in Chapter 41. You can safely reuse these identifiers unless they are marked for internal use by a star in tables and descriptions in that chapter.

Comments

Comments are identified by an initial (*) and a terminating (*). Comments may be inserted anywhere that white space can appear in CPN net inscriptions. For example,

```
(* Enumerated colorset *)

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

(* Indexed colorset *)

color BankBranch =
    index branch_number with 1...8;

color integer = (* Integer colorset *) int;
```

Since comments are treated as white space, inserting them in the middle of identifiers causes a syntax error. For example,

```
var New_int : inte(* comment *)ger;
```

results in the following error message:

```
Error:
Syntax error in variable declaration:
inte ger;
```


Chapter 32

Colorsets

Design/CPN provides a mechanism for user-defined datatypes. For historic reasons these user-defined datatypes are called *colorsets*.

Each colorset represents a set of values and is identified by an alphanumeric identifier.

The colorsets must be declared in global, local, and temporary declaration nodes. Design/CPN automatically generates a set of pre-declared constants, operations, and functions for each colorset declared.

Classifying Colorsets

Colorsets are classified by size and by complexity.

Size

Colorsets can be classified as large or small. This distinction determines which optional system-defined constants, operations, functions may be declared for a particular colorset.

A colorset is large if it contains too many elements to enumerate. Otherwise, it is small:

- Unit, bool, indexed and enumerated colorsets are small.
- Int, and string colorsets are small if and only if they include the appropriate **with** specification. Note that the real colorset is always large, even with the **with** specification.
- Product, record, and union colorsets are small if and only if all their component colorsets are small.
- List colorsets are small if and only if their component colorset is small and they are declared by means of a **with** clause.

- Subset colorsets are small if and only if either their component colorset is small or their subset specification is a list.

Complexity

A colorset is *compound* if it is constructed from other colorsets. Otherwise, it is *simple*.

Simple Colorsets

Unit

The unit colorset comprises a single element, denoted (). It has no predefined operations that use it, but is very useful as a placeholder. See examples in Chapter 40, “Inscriptions.”

Declaration Syntax

```
color colorset_name = unit <<with
    new_unit_id>>;
```

Optional With Clause

Renames the value, that is, defines the identifier representing (). The new value name must be an alphanumeric identifier.

Declaration Example

```
color terminator = unit with nothing_here;
```

Value Representation Example

```
ML> nothing_here;
> () : terminator
```

Boolean

The boolean values are the constants **true** and **false**.

Declaration Syntax

```
color colorset_name = bool <<with
    (new_false, new_true)>>;
```

Optional With Clause

Renames the values, that is, defines the identifiers representing “true” and “false”. The new value names must be alphanumeric identifiers.

Declaration Example

```
color answer = bool with
    (no, yes);
```

Value Representation Example

```
ML> no;

> false : answer
```

Boolean Operations

The following operations are predefined for booleans:

not The unary negation operation.

For example:

```
ml> not true;

      false : bool
```

This example returns false, which is the negation of the boolean value true.

andalso Boolean conjunction. And.

orelse Boolean disjunction. Inclusive or.

relational

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal

= The equality operator. All the colorsets, with the exception of functions have their own infix equality operators that return boolean values.

if, then, else

The conditional expression. See Chapter 38, “Functions,” for a discussion of its use.

Boolean Selectors

See Chapter 38, “Functions,” for a discussion of their meanings and use.

Integers

Integers are numerals without a decimal point. The integer colorset is large unless restricted by the **with** clause, in which case it is small.

Declaration Syntax

```
color colorset_name = int <<with  
                        int-expstart..int-expend>>;
```

Optional With Clause

Restricts the integer colorset to an interval determined by the two expressions `int-expstart` and `int-expend`.

The expression `int-expstart` must be `int-expend`.

Declaration and Use Example

```
color small_integer = int with 1..3;  
  
var SmallInt : small_integer;
```

The CPN variable `SmallInt` may have a range of integer values from 1 to 3. For example, 2 is a legal value, but 4 is not and 2.9 is not. See Chapter 34 for a discussion of the creation and use of CPN Variables.

Integer Operations

The following operations are predefined for integers:

abs	The unary absolute value operator.
~	The unary negation operator.
+	The infix addition operator.
-	The infix subtraction operator.
*	The infix multiplication operator.

Division is handled by two operations:

div	Quotient
mod	Remainder

For example:

```
ML> 37 div 5;
      7 : int
ML> 37 mod 5;
      2 : int
```

Real Numbers

Real numbers are distinguished from integers by the decimal point. The period separates the integer part from the fractional part. One or more digits must follow after the decimal point

The number can be written in scientific notation where the power of 10 is given after the exponent symbol E. The fraction part is optional if the exponent part is present. The real colorset is large.

Declaration Syntax

```
color colorset_name = real <<with
                        real-exp_start..real-exp_end>>;
```

Optional With Clause

Restricts the real colorset to an interval determined by the two expressions `real-expstart` and `real-expend`.

The expression `real-expstart` must be `real-expend`.

Declaration and Use Example

```
color small_real = real with 1.0..3.0;

var SmallReal : small_real;
```

The CPN variable `SmallReal` may have a range of real values from 1.0 to 3.0. For example, 1.0 and 2.9 are legal values but 1 and 4.9 are not.

Real Operations

The following operations are predefined for reals:

<code>abs</code>	Absolute value (infix)
<code>~</code>	Negation (prefix)
<code>+</code>	Addition (infix)
<code>-</code>	Subtraction (infix)
<code>*</code>	Multiplication (infix)
<code>/</code>	Division (infix)
<code>sqrt</code>	Square root (unary)
<code>ln</code>	Natural logarithm (unary)
<code>exp</code>	Exponential (unary)
<code>sin</code>	Sine (unary)
<code>cos</code>	Cosine (unary)
<code>tan</code>	Tangent (unary)
<code>arctan</code>	Arc tangent (prefix)

The trigonometric functions all work in units of radians.

Strings

Strings are specified by sequences of printable ASCII characters surrounded with double quotes. The string colorset is large unless restricted by the **with** clause, in which case it is small.

Characters are strings of length one.

Declaration Syntax

```
color colorset_name = string <<with
    string-expb .. string-expt and
    int-expb .. int-expt >>;
```

Optional With Clause

Restricts the character set of string colorsets.

The character set is specified by the string expressions
string-exp_{start} .. string-exp_{end}.

Each string expression must evaluate to a single character
(characters are defined as strings of length one) and
string-exp_{start} must be string-exp_{end}.

Optional And Clause

Restricts the length of string colorsets.

The minimum and maximum length of the string is specified by the
integer expressions int-exp_{start} .. int-exp_{end}.

The integer expressions must be:

```
0 int-expb int-expt.
```

Declaration and Use Example

```
color small_string = string
    with "a".. "d" and 3..9;
var SmallString : small_string;
```

The CPN variable SmallString may contain only the letters a, b, c, and d. Its length must be 3 and 9. For example, "abcd" and "bdbdbbdb" are legal values but "bcde" and "ab" are not.

```
ML> "abc"
> "abc" : string
```

String Operations

size	length of a string (prefix)
^	concatenate strings (infix)
ord	octal code word corresponding to a character (prefix)
chr	character corresponding to an octal code word (prefix)
mkst_ms	string representation for a multiset
mkst_col	string representation for a colorset

String Compare

Lexigraphic comparison based on the ASCII code representation of the string. The comparison is based on the first character pair that differs in the string.

>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
=	equal
<>	not equal

Escape Sequences

The backslash operator provides a way to insert control characters in strings.

\n	line feed
\t	tab

Enumerated Values

Enumerated values are explicitly named as identifiers in the declaration. These values must be alphanumeric identifiers. This is a small colorset.

Declaration Syntax

```
color colorset_name =  
    with id0 | id1 | ..... | idn;
```

Declaration Example

To represent several banks operating in Massachusetts:

```
color Banks = with HarvardTrust |  
    Shawmut | NationalGrand;
```

Value Representation Example

A particular bank is then simply entered:

```
ML> HarvardTrust ;  
  
    > HarvardTrust : Banks
```

Suggested Uses

Typically used for a limited set of predefined values.

Indexed Values

Indexed values are sequences of values comprised of an identifier and an index-specifier. The index-specifier range is:

$$\text{int-exp}_{\text{start}} .. \text{int-exp}_{\text{end}}$$

When:

$\text{int-exp}_{\text{start}}$ and $\text{int-exp}_{\text{end}}$ are integers,

and:

$$\text{int-exp}_{\text{end}} \text{ is } 0$$

and:

$$\text{int-exp}_{\text{start}} \text{ is } \text{int-exp}_{\text{end}}$$

the colorset is small.

Declaration Syntax

```
color colorset_name = index identifier
                      with int-expstart .. int-expend;
```

Declaration Example

To represent the different branches in a bank:

```
color HarvardTrust_bank = index branch_number
                          with 1..8;
```

Value Representation Example

```
ML> branch_number(3);

> branch_number 3 : HarvardTrust_bank
```

Suggested Uses

Typically used for naturally indexed values.

Compound CPN Colorsets

The following colorsets use previously declared colorsets.

Tuples

A fixed-length, positional colorset whose set of values is identical to the cartesian product of the values in previously declared colorsets. Each of the component colorsets may be a different type.

Declaration Syntax

```
color colorset_name= product
                      colorset_name1*colorset_name2*
                      ...*colorset_namen;
```

n must be ≥ 2 for colorset_name₁, colorset_name₂, ..., colorset_name_n.

The values in the set are tuples. Individual values are positionally accessed.

Declaration Example

A business rule that large customer orders must be processed by expert staff members, where the customer order colorset is declared by:

```
color customer_order_size = with Big | Medium
                             | Small;
```

and the staff expertise is declared by:

```
color staff_type = with Expert | Journeyman |
                   Novice
```

can be represented by a tuple colorset:

```
color order_process_requirement =
  product customer_order_size * staff_type;
```

Representation Examples

```
ML> (Big,Expert);

> (Big,Expert) : customer_order_size
  * staff_type
```

Note: This example uses explicit values. If the tuple had contained CPN variables it could have been used as a pattern. See Chapter 36, “Expressions,” for a discussion of tuple patterns.

Suggested Uses

Used in the collecting of data where the values are tuples.

Records

A fixed-length colorset whose set of values is identical to the cartesian product of the values in previously declared colorsets. Each of the component colorsets may be a different type and each is identified by a unique label so that each field is position-independent.

Declaration Syntax

```
color colorset_name= record
  id1:colorset_name1 * id2:colorset_name2 *
  ...* idn:colorset_namen;
```

n must be 2 for colorset_name₁, colorset_name₂, ..., colorset_name_n.

The values in the set are labeled records and their contents can be recognized by the fieldname.

Selector Functions

For records, CPN ML contains a set of predeclared selector functions that access components of the tuple:

#id₁, #id₂, ..., #id_n.

Declaration Example

A business rule that large customer orders must be processed by expert staff members, where the customer order colorset is declared by:

```
color customer_order_size = with Big | Medium
                             | Small;
```

and the staff expertise is declared by:

```
color staff_type = with Expert | Journeyman |
                   Novice
```

can be represented by a record colorset:

```
color order_process_requirement =
    record Order:customer_order_size
    * Staff:staff_type;
```

Value Representation Example

```
ML> {Order=Big, Staff=Expert};

> {Order=Big, Staff=Expert} :
  { Order:customer_order_size,
    Staff:staff_type }
```

Note: This example uses explicit values. If the record had contained CPN variables it could have been used as a pattern. See Chapter 36, “Expressions,” for a discussion of record patterns.

Selector Example

```
ML> #Order{Order=Big, Staff=Expert};  
    > Big : customer_order_size
```

Suggested Uses

Used in the collecting of data where the values are records.

Lists

A variable-length colorset. The values are a sequence whose colorset must be the same type.

Declaration Syntax

```
color colorset_name = list colorset_name0  
    <<with int-expb .. int-expt>>;
```

Optional With Clause

The **with** clause specifies the minimum and maximum length of the lists.

Declaration Example

The asset types for a bank could be maintained in a list where the assets have been declared as strings:

```
color asset_list = list assets;
```

Value Representation Example

```
ML> ["real estate", "treasury bonds"] ;  
    > ["real estate","treasury bonds"] :  
      string list
```

Note: This example uses explicit values. If the list had contained CPN variables it could have been used as a pattern. See Chapter 36, “Expressions,” for a discussion of list patterns.

List Operators and Functions

Prepend: The `::` operator creates a list from an element and a list by placing the element at the head of the list:

```
ML> 1::[2,3,4] ;  
      > [1,2,3,4] : int list
```

If the list is the empty list, the result is a list of one member. For example:

```
ML> 1::[] ;  
      > [1] : int list
```

This is one way to get list construction started.

Concatenate: The `^^` operator creates a list from two source lists. The elements of each list are extracted and combined to form the new list:

```
ML> [1,2,3]^^[4,5,6] ;  
      > [1,2,3,4,5,6] : int list
```

To create a new list whose elements are the source lists, not the elements of the source list, put an extra set of brackets around each list:

```
ML> [[1,2,3]]^^[4,5,6]] ;  
      > [[1,2,3],[4,5,6]] : (int list) list
```

This is the same as if you had said:

```
ML> [1,2,3]::[[4,5,6]] ;  
      > [[1,2,3],[4,5,6]] : (int list) list
```

because the element colorset in this example is an integer list.

Suggested Uses

Using one token with a list of values gives access to all the values at once and makes ordering and other behavior "based on all tokens" possible.

Subsets

Subsets are a selection of values in a previously declared colorset.

Declaration Syntax

The subset specification can take two different forms, function and list:

Function Form

```
color colorset_name = subset colorset_name0
    by subset-function;
```

By Clause

Specifies a function whose return value is a boolean. `colorset_name` will contain exactly those values from `colorset_name0` that are mapped into the boolean value true.

List Form

```
color colorset_name = subset colorset_name0
    with subset-list;
```

With Clause

Specifies a list with elements from `colorset_name0`. `colorset_name` will contain exactly those values that are listed.

Declaration and Use Example - List Form

```
color Banks = with HarvardTrust |
    Shawmut | NationalGrand;

color Banksub = subset Banks
    with [Shawmut, NationalGrand];

ML> [Shawmut, NationalGrand];

> [Shawmut, NationalGrand] : Banks list
```

Notice that when simply given as a value list CPN ML responds with a combination colorset - Banks list. CPN ML would have similarly accepted the following:

```
ML> [HarvardTrust, NationalGrand];  
  
> [HarvardTrust, NationalGrand] :  
    Banks list
```

However, if Banks_{sub} is the colorset for a place, Design/CPN would enforce the subset and would reject an inscription of [HarvardTrust, NationalGrand] as a syntax error.

Suggested Uses

Restriction of an already defined colorset. Implies runtime checks, initial, and current markings.

Union

union is a (disjoint) union of previously declared colorsets.

Declaration Syntax

```
color colorset_name =  
    union id1<<:colorset_name1>> +  
        id2<<:colorset_name2>>  
    + ..... + idn<<:colorset_namen>>;
```

For each component colorset, provide a unique selector. This construction is similar to variant records in other programming languages.

If colorset_name_i is omitted, then id_i is treated as a new value, and it can be referred to simply as id_i.

The sequence of selectors in the declaration defines the order of elements in the new colorset. For the predeclared function **lt'**colorset_name, this implies that any value with selector id_i is less than any value with selector id_j, provided that i < j.

For a given selector, id_i, the ordering of values is defined by **lt'**colorset_name_i.

The predeclared function **ran'**colorset_name works as follows:

- first a selector id_i is drawn randomly,
- then the function **ran'**colorset_name_i for the corresponding component colorset (if any) is applied.

Declaration Example

```
color Banks = with HarvardTrust | Chase
               | MarbleheadSavings | Shawmut
               | NationalGrand;

color No_of_Branches = int with 0..10;

color No_of_Customers = int with 0..99999;

color RegionBanksSize = union Name:Banks +
                             CustomerBase : No_of_Customers +
                             BranchNumbers : No_of_Branches;
```

See Chapter 40, “Inscriptions,” for an example showing arc inscriptions based on this colorset.

Suggested Uses

A union colorset is similar to a variant record in other programming languages with selectors to signify variant type.

Alias CPN Colorset

A colorset that provides an alias construct. The colorset has exactly the same values and properties as a previously declared colorset. However, each of the two colorsets has its own, possibly different, set of predeclared constants, operations, and functions as specified by the optional **declare** clause. Thus, to obtain the same declare set as the source colorset, specify `declare same`.

Declaration Syntax

```
color colorset_name = colorset_name0
                        declare same;
```

The type checking mechanism in Design/CPN does not distinguish between the original colorset and its duplicates. Hence colorsets for all members in a place fusion set must have the same prime colorset. The same rule applies for port nodes and their assigned socket/parameter nodes.

Declaration Example

If two banks are merging and want to describe the same internal structure but distinguish their origin, they could define it:

```
color BayBank_bank = HarvardTrust_bank;
```

Suggested Uses

To prevent inconvenience when colorsets are needed that differ only in name.

Function

See Chapter 38, “Functions.”

Declare Clause

The **declare** clause makes the system constants, operations, and functions described below available to the colorset.

The syntax is:

```
color colorset declare id1, ..., idk;
```

id₁, ..., id_k is a comma-separated sequence of the desired names. Two special keywords have been defined to ease the declaration:

- **declare all** - yields all possible constants, operations, and functions for the actual kind of colorset.
- **declare same** - declares for a duplicate colorset the same functions that were requested for the original colorset.

Random Value

Returns a random value. The colorset of the returned value is that specified in the function call.

Syntax

```
ran'colorset ();
```

Function from: unit -> colorset.

The function cannot not be used in net inscriptions such as arc inscriptions and guards. It can, however, be used in time regions and in code segments.

Example

Given the following colorset declaration:

```
color Banks =
  with HarvardTrust | Shawmut |
  NationalGrand;
```

then the following function call:

```
ran'Banks  ();
```

returns one of the enumerated values. One response might be:

```
> Shawmut : Banks
```

Another time the response might be:

```
> NationalGrand: Banks
```

Less Than

Tests whether a value in the specified colorset is less than another value in same colorset with respect to the colorset ordering. (See the section on colorset ordering at the end of this chapter).

Syntax

```
lt'colorset (value-identifier1,
             value-identifier2);
```

Function from: (colorset * colorset) -> bool.

Example

Given the following colorset declaration:

```
color Banks =
  with HarvardTrust | Shawmut |
  NationalGrand;
```

then the following function call:

```
lt'Banks (Shawmut, NationalGrand);
```

returns:

```
> true : bool
```

String Representation of a Value

Returns the string representation for a value of a particular colorset.

Syntax

```
mkst_col'colorset (value-identifier);
```

Function from: colorset -> string.

To get around the limited input line length in the Edinburgh interpreter (256 characters), a line feed is inserted after each element in a list and in a record by `mkst_col` for list and record colorsets, respectively.

Example

Given the following colorset declaration:

```
color Banks =  
  with HarvardTrust | Shawmut |  
  NationalGrand;
```

then the following function call:

```
mkst_col'Banks (Shawmut);
```

returns:

```
> "Shawmut" : string
```

String Representation of a Multiset

Returns the normalized string representation for a multiset of a particular colorset.

Syntax

```
mkst_ms'colorset (multiset-specifier);
```

Function from: colorset ms -> string.

Example

Given the following colorset declaration:

```
color Banks =  
  with HarvardTrust | Shawmut |  
  NationalGrand;
```

then the following function call:

```
mkst_ms'Banks (1`Shawmut + 2`HarvardTrust);
```

returns:

```
> "2`HarvardTrust + 1`Shawmut" : string
```

Multiset, Size, First, and Last Constants

These constants can only be declared for small colorsets. They define:

- 1) The full multiset over the `colorset`. The multiset contains exactly one occurrence of each colorset value.
- 2) The number of values in `colorset`.
- 3) The first value in `colorset`.
- 4) The first value in `colorset`.

ms

Multiset constant declaration

```
<<colorset declaration >> declare ms;
```

Multiset constant use

To create a multiset for the colorset with **ms** declared, either:

Specify the colorset name to obtain the internal representation of the multiset, or

Use the function **mkst_ms** to obtain a string representation of the multiset.

size

```
size'colorset;
```

Function from: `colorset` ->integer-value

first

```
first'colorset;
```

Function from: `colorset` ->identifier-value

last

```
last 'colorset;
```

Function from: colorset -> identifier-value

Examples

Given the following colorset declaration:

```
color Banks = with HarvardTrust | Chase
               | MarbleheadSavings | Shawmut
               | NationalGrand
declare ms, size, first, last;
```

then evaluation of the following :

```
Banks;
mkst_ms'Banks (Banks);
size'Banks;
first'Banks;
last'Banks;
```

displays:

```
> !! ((1,HarvardTrust),!! ((1,Chase),!!
    ((1,MarbleheadSavings),!! ((1,Shawmut),!!
    ((1,NationalGrand),empty)))) : Banks ms

> "1`HarvardTrust + 1`Chase +
  1`MarbleheadSavings +
  1`Shawmut + 1`NationalGrand" : string

> 3 : int

> HarvardTrust : Banks

> NationalGrand : Banks
```

Ordinal Number and Value

ord and **col** can only be declared for enumerated and indexed colorsets. They

- 1) Convert from a value to a number representing its position in the colorset definition.
- 2) Convert from a number representing a value's position in the colorset definition to the value.

The result of **ord** 'colorset is always in the interval:
0..(size'colorset-1).

The argument of **col**'colorset should be in the same interval; otherwise, an exception is raised.

Syntax

```
ord'colorset (identifier_value);
```

Function from: colorset -> int

```
col'colorset (position_value);
```

Function from: int -> colorset

Examples

Given the following colorset declaration:

```
color Banks =  
  with HarvardTrust | Shawmut | NationalGrand  
  declare ord, col;
```

then evaluation of the following :

```
ord'Banks (HarvardTrust);  
col'Banks (2);
```

displays:

```
> 1 : int  
> Shawmut : Banks
```

Index Number and Value

index and **clr** convert from index number to identifier-value, and vice versa. They can only be declared for indexed colorsets.

The argument to **index** is the identifier-value specified in the colorset definition followed by an index number in parentheses. The value returned is the integer index number.

The range of the argument to **clr** must be that of the index numbers specified in the colorset definition; otherwise, an exception is raised.

Syntax

```
index'colorset (identifier_value  
  (index_number));
```

Function from: colorset -> int

```
clr'colorset (index_number);
```

Function from: int -> colorset

Examples

Given the following colorset declaration:

```
color FleetBank_branch = index branches with  
1..8 declare index, clr;
```

then evaluation of the following :

```
index'FleetBank_branch (branches(1));  
clr'FleetBank_branch (3);
```

displays:

```
> 1 : int  
> branches 3 : FleetBank_branch
```

Distance and Rotation for Values

dist and **rot** provide cyclic manipulation of the values of colorset. They can only be declared for enumerated and indexed colorsets.

dist returns the distance from the first value to the second in the forward direction. The result is always in the interval:
`0..(size'colorset-1)`.

rot returns a value, which is reached from a given value by rotating a specified distance, positive or negative. There is no restriction upon the integer argument of `rot'colorset`.

Syntax

```
dist'colorset (identifier_value1,  
               identifier_value1);
```

Function from: (colorset * colorset) -> int

```
rot'colorset distance-to-rotate  
identifier_value;
```

Function from: int -> (colorset -> colorset)

Examples

dist example

Given the following colorset declaration:

```
color Banks =
  with HarvardTrust | Shawmut | NationalGrand
  declare dist, rot;
```

then evaluation of the following :

```
dist'Banks (HarvardTrust, NationalGrand);
```

displays:

```
> 2 : int
```

rot example

Given the following colorset declaration:

```
color Banks =
  with HarvardTrust | Shawmut | NationalGrand
  declare dist, rot;
```

then evaluation of the following :

```
rot'Banks 2 Shawmut;
```

displays:

```
> HarvardTrust : Banks
```

Multiplication of Multisets

mult constructs a multiset for a tuple or record colorset by multiplying a set of multisets constructed from its component colorsets.

Syntax

```
mult'colorset (colorset1 ms * colorset2 ms *
..... * colorsetn ms);
```

Function from: (multiset*multiset...)-> multiset

Example

Given the following colorset declarations:

```
color alpha = with a | b | c;  
color number = with num1 | num2 | num3;  
color tuple = product alpha * number declare  
    mult;
```

then evaluation of the following :

```
mult'tuple ((1`a) + (3`c), (3`num1) +  
    (4`num3));
```

displays the internal multiset representation:

```
> !! ((3,(a,num1)),!! ((4,(a,num3))),  
    !! ((9,(c,num1))),  
    !! ((12,(c,num3)),empty))) : tuple ms
```

To display the multiset as a string:

```
mkst_ms'tuple (mult'tuple ((1`a) + (3`c),  
    (3`num1) + (4`num3)));
```

displays:

```
> "3`(a,num1) + 4`(a,num3) + 9`(c,num1) +  
    12`(c,num3)" : string
```

Membership of a Subset CPN Colorset

in tests whether a given value is a member of the colorset.

This function can only be declared for subset colorsets or for `int`, `real` or `string` colorsets defined using a **with**-clause specification.

Syntax

```
in'colorset (value);
```

Function from: colorset -> bool

Example

```
color Banks = with HarvardTrust | Chase  
    | MarbleheadSavings | Shawmut  
    | NationalGrand;
```

```
color LocalBanks = subset Banks
                    with [NationalGrand,
                        MarbleheadSavings];

ML> in'LocalBanks (NationalGrand);

> true : bool
```

Membership of Union Base CPN Colorset

of_id_i tests whether a value from a union colorset belongs to a specific component colorset (with the selector id_i).

Syntax

```
of_idi 'colorset (component_colorset);
```

Function from: colorset -> bool

Example

```
color Banks = with HarvardTrust | Chase
                | MarbleheadSavings | Shawmut
                | NationalGrand;

color No_of_Branches = int with 0..10;

color No_of_Customers = int with 0..99999;

color RegionBanksSize = union Name:Banks +
    CustomerBase : No_of_Customers +
    BranchNumbers : No_of_Branches
    declare of_CustomerBase,
            of_BranchNumbers;

ML> of_CustomerBase'RegionBanksSize
    (CustomerBase 8);

> true : bool

ML> of_CustomerBase'RegionBanksSize
    (CustomerBase 50000000);

> false : bool

ML> of_BranchNumbers'RegionBanksSize
    (BranchNumbers 50000000);

> false : bool
```

Ordering of CPN Colorsets

All colorsets are ordered. The order is defined in the following way:

enumerated	Determined by the order in which the value names appear in the colorset declaration.
indexed	Determined by the usual integer ordering of the indexes.
product	Lexicographic (with the ordering of the base colorsets).
record	Lexicographic (with the ordering of the base colorsets).
list	Lexicographic (with the ordering of the base colorset).
subset	Determined by the order of the base colorset.
unit	Trivial order.
bool	False before true.
int, real	Usual ordering of numbers.
string	Lexicographic (with the ASCII ordering).

For non-predefined ML colorsets, the order is determined from the `lt` function in the additional specification.

Chapter 33

Values

Overview

A value declaration connects an identifier to a value. The reserved word **val** is used to declare an identifier and an arbitrary expression may be used as the value.

Values are not just constants - functions are declared as values (the reserved word **fun**, used to declare functions, returns a value declaration). Function values are discussed in Chapter 38, “Functions.”

In CPN ML values must be declared in a declaration node. They may be used but not redefined in inscriptions and in code segments. The exception is values that are declared in a `let` structure.

They are frequently used to define constants used in multiple locations as well as for specifying time delays.

Declaration

Syntax

The syntax is:

```
val identifier = expression;
```

where `identifier` is an identifier and `expression` is a CPN ML expression, including multiset expressions. The expression represents the value to be associated with the identifier.

Value Representation

Val declaration expressions may use any value representation that is syntactically unique, without a prior colorset declaration. These include:

- unit
- bool
- int
- real
- string
- tuple
- list
- record

Examples:

These examples all use the conventions described in Chapter 30, “Introduction to CPN ML,” in order to show the result of the declaration. In Design/CPN practice, however, they would be declared in a declaration node and no value would be displayed.

Integer value

```
ML> val intval = 3;  
      > val intval = 3 : int
```

Boolean value

```
ML> val booleanval = true;  
      > val booleanval = true : bool
```

Real value

```
ML> val realval = 3.0;  
      > val realval = 3.0 : real
```

String value

```
ML> val stringval = "string 3.0";  
      > val stringval = "string 3.0" : string
```

Tuple value

```
ML> val tupleval = (3,4.0);  
      > val tupleval = (3,4.0) : int * real
```

List value

```
ML> val listval = [3,4];
    > val listval = [3,4] : int list
```

Colorsets

Value declarations may use expressions of previously declared colorsets. For example, the following declaration is acceptable because the colorset of the value of `bigjob` has been previously declared:

```
color Staff =
    with Expert | Journeyman | Novice;

color ProcessOrder = product Order * Staff;

val bigjob = (Big, Expert);
```

The following example is also acceptable:

```
val integer_tuple = (1, 4);
```

because the `(1, 4)` specification is one of the syntactically unique value representations known to Design/CPN.

The following example, however:

```
val alpha_tuple = (ss, aa);
```

produces the following error message:

Errors

C.4 Error in global/temporary declaration node

Type checking error in: ss
 Unbound value identifier: ss
 [Closing <string>]
 [Closing <string>]
 « «18» »

because the system doesn't recognize the value `(ss, aa)`. If `ss` and `aa` were specified as strings:

```
val alpha_tuple = ("aa", "ss");
```

the declaration would be acceptable.

Other types of specifiers

Multiset Expression

Multiset expressions may be used as value specifiers. For example:

```
ML> val baz = 1`3 + 3`4;

> val baz = !! ((3,4),!! ((1,3),empty)):int
ms
```

See Chapter 37, “Multisets,” for a complete description.

Function expression

Function expressions may be used as value specifiers. For example:

```
ML> val baz = fn x => 3 + x;

> val baz = fn : int -> int
```

See Chapter 38, “Functions,” for a complete description.

Value Uses in CPN Inscriptions

Chapter 40, “Inscriptions,” describes the use of constants in CPN inscriptions.

Chapter 34, “CPN Variables,” describes the Design/CPN mechanism for representing changing values in CPN inscriptions.

Chapter 34

CPN Variables

Term Definitions

Variables

A *variable* is an identifier whose value can be changed during the execution of the model.

Binding

Binding is the association of a value with a variable. A binding has both scope and extent.

Scope

Scope is the locations in a model in which a particular binding can be referenced.

Extent

Extent is the interval during which a particular binding is in effect.

CPN Variables

CPN variables are variables that are used in CP net inscriptions. They have the following characteristics:

- They are declared using the reserved word **var** and the name of a previously declared colorset in a global or temporary declaration node.
- They are bound to a variety of different values from their colorset by the simulator as it attempts to determine if a transition is enabled.

- There can be multiple bindings simultaneously active on different transitions. These bindings can exist simultaneously because they have different scopes.
- The extent of a CPN variable binding is the firing of a particular transition.
- They provide arc inscriptions with the ability to reference different values. See Chapter 33, “Values,” for a discussion of constants and their use in arc inscriptions.

Declaration syntax

```
var id1, id2, ....., idn : colorset_name;
```

where:

id is an alphanumeric identifier,

: is the syntactic separator between the identifier(s)

colorset_name

is the name of a previously declared CPN colorset.

The system checks whether any of the CPN variable names have been used for type constructors, for example, in a colorset declaration. If an error occurs, a warning is given.

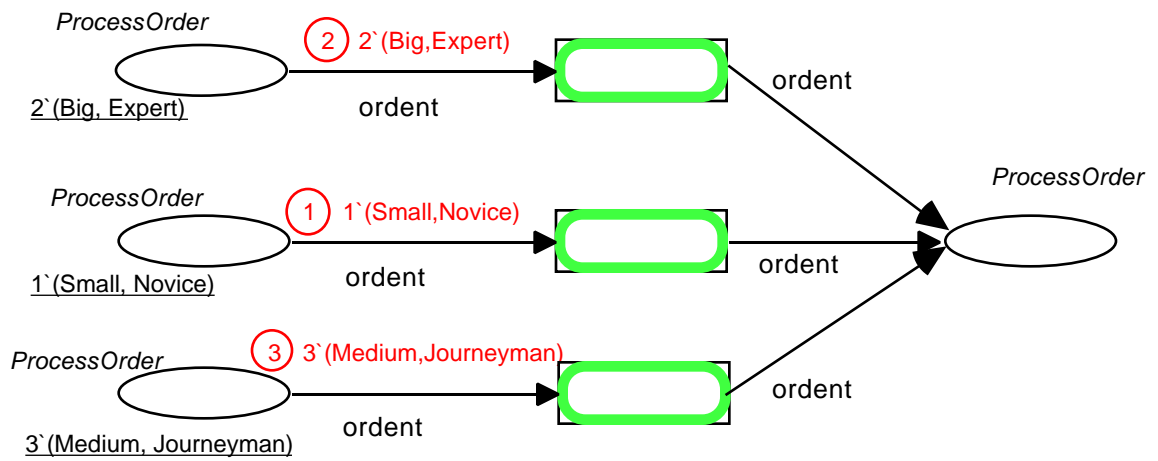
See Chapter 32, “Colorsets,” for a discussion of the declaration and use of CPN colorsets.

Example of Declaration and Use

This example illustrates the significance of the scope characteristics of CPN variables. The following CPN colorset and CPN variables are declared in the global declaration node:

```
color Order = with Big | Medium | Small;  
color Staff = with Expert | Journeyman | Novice;  
color ProcessOrder = product Order * Staff;  
var ordent : ProcessOrder;
```

The following three transitions are all enabled with different values for the CPN variable ordent :



Chapter 35

Reference Variables

Reference Variables

ML reference variables are the same thing as pointer variables in imperative languages such as C. They may be used only in code segments.

Reference variables must never be used in any way that affects transition enablement. To illustrate the problems that could arise if this rule were broken, consider the following example:

A reference variable, `refvar`, is used in the arc inscription of two different transitions, transition A and transition B. Both are enabled, and transition A fires. Transition A changes the value of `refvar` in such a way that transition B is no longer actually enabled. But the system has no way of knowing this, since enabled transitions are not rechecked before they are fired (to recheck each time a transition is chosen for firing would degrade performance to the point of unusability). Attempting to fire transition B results in unpredictable system errors.

Reference variables are necessary for accessing information outside the model, such as files and databases, and for providing information about the model. They are used in this manual to handle statistical variables and I/O access.

Val-defined Reference Variables

Val-defined reference variables are defined in declaration nodes. The value stored at the location named by the identifier is accessed a syntax that tells the system to obtain a value pointed to be a name, not to change the value of the name itself.

Caution

Design/CPN is unable to check whether a reference variable has been declared with **val** instead of **globref**. If a reference variable declared by means of **ref** has the same name as another reference

variable declared by means of **globref**, **pageref**, or **instref**, the CPN Simulator will apply the latter.

Declaration Syntax

To declare a val-defined reference variable and provide an initial value:

```
val id = ref data_value;
```

where <id> is an alphanumeric identifier, and the value following **ref** is the initial value. The colorset of the initial value is either predefined (See Chapter 33, “Values,” for a discussion of predefined identifiers) or has been previously declared with a **color** declaration.

For example,

```
val foo = ref (3,4);
```

If this form were interactively evaluated by ML, the system would return:

```
> val foo = ref (3,4) : (int * int) ref
```

which declares the identifier `foo` as tuple - (int * int) - reference variable with the value of (3,4).

Reference Syntax

To update the value of `foo` by 1, type:

```
foo := !foo + 1;
```

accesses the value (which is 0) at the location `foo` and updates it by 1. The `:=` and `!` syntax specifiers tell the system to access the value at the location identified by the name `foo`.

Value Syntax

Entering the reference variable name by itself:

```
foo;
```

results in:

```
> ref 2 : int ref
```

which says that the identifier is a reference variable whose value is two and whose colorset is integer reference.

CPN Reference Variables

Global Reference Variables

CPN global reference variables provide the same capabilities as do the val-ref defined reference variables: they define reference variables with a global scope.

Declaration syntax

```
globref id = exp;
```

Global reference variables must be declared in a global or temporary declaration node and can be used in all code segments in the entire diagram.

The colorset of the global reference variable is determined by the colorset of the expression, which also determines the initial value.

Page Reference Variables

Defines reference variables with a page scope.

Declaration syntax

```
pageref id = exp;
```

Page variables must be declared in a local declaration node and can only be used in code segments on the corresponding page. All page instances of the page with the local declaration node refer to the same reference variable.

The colorset of the expression determines the colorset of the reference variable, which in turn determines the initial value.

Instance Reference Variables

Defines reference variables with an instance scope.

Declaration syntax

```
instref id = exp;
```

Instance variables must be declared in a local declaration node and can only be used in code segments on the corresponding page. Each page instance of the page with the local declaration node refers to its own copy of the reference variable.

CPN ML Reference

The colorset of the expression determines the colorset of the instance reference variable, which in turn determines the initial value.

Chapter 36

Expressions

In this chapter we first talk about the expression syntax, then expression evaluation, and finally expression use as inscription patterns and constructors.

Multiset expressions are not discussed in this chapter but are described, along with those predefined functions that operate on multisets, in Chapter 37, “Multisets.”

Functional expressions are not discussed in this chapter but are described in Chapter 38, “Functions,” along with the **let** and **case** forms.

Term Definitions

An *expression* is a sequence of values, CPN variables, type constructors, and operators that can be evaluated. The result of expression evaluation depends on both the types of operators and the CP net use of the expression.

A *canonical expression* is an expression that has itself as a value; it has been reduced to its lowest, or most fundamental form.

A *simple expression* is an expression whose components are all the same colorset.

Compound expressions are expressions whose components are simple expressions of different colorsets or other compound expressions.

Inscription expressions are expressions used as CP net arc inscriptions and guards. They do not contain reserved words, keywords, or semicolons, with the exception of

```
( ) [ ] { } , : = andalso orelse not  
if/then/else let/in/end case/of fn
```

Expression Syntax

Syntactic Specifiers and Reserved Words

The syntactic specifiers and reserved words that may be used in inscription expressions include:

```
( ) [ ] { } , : = andalso orelse  
if/then/else let/in/end case/of fn
```

Their meanings are:

Specifier	Meaning	Where Described
()	tuple constructor expression associator	
[]	list constructor	
{}	record constructor	
,	tuple, list, or record component, or subexpression separator	
:		
=	equality operator	
andalso	boolean conjunction	
orelse	boolean disjunction	
if/then/else	boolean conditional	
let/in/end	local	
case/of	case	
fn	function	

Variables

Expressions used in code segments may contain CPN variables and reference variables, but inscriptions may contain only CPN variables.

Values

Val-defined identifiers may be anything legally definable by a `val` construct, including functions. See Chapter 33, “Values,” for a complete discussion of the `val` construct, and Chapter 38, “Functions,” for a discussion of val-defined identifiers whose value is a function.

Constructors

Constructors include:

- Tuple constructors:

`(id1, ..., idn)`

- Record constructors:

`{id1name = id1, ..., idnname = idn}`

- List constructors and operators:

`[id1, ..., idn]`

Chapter 32, “Colorsets,” provides complete descriptions of the constructors and their use.

Operators

Operators include:

- Arithmetic:

`~ * / div mod + -`

- String:

`^`

- List:

`:: ^^`

- Relational:

`= <> < > <= >=`

- Boolean:

`andalso orelse % \`

- Selector

`#`

Chapter 32, “Colorsets,” provides complete descriptions of the operators and their use.

Expression Evaluation

Expressions are evaluated within the context of their CP net location. The result of the evaluation depends on:

- 1) Component colorsets
- 2) Operator precedence
- 3) Associativity rules
- 4) CP net context

Evaluation Order

The default evaluation proceeds from left to right under the control of operator precedence and operator association.

The default evaluation order can be changed by the use of parenthesis grouping. For example, multiplication has a higher precedence than addition, so that evaluation of the following expression:

```
ML> 3 + 4*5;
```

yields:

```
> 23 : int
```

but

```
ML> (3 + 4)*5;
```

yields:

```
> 35 : int
```

Operator Precedence

Precedence is the default order in which different operators are evaluated. The Operator Precedence and Association Table below gives the default precedence for all the operators that are used in CPN ML expressions.

Association

Association is the direction of evaluation. An expression containing sequential occurrences of the same operator can be interpreted in several ways. For example, the expression:

5 - 4 - 3

can be interpreted as

(5-4) -3

or as

5 - (4 - 3) .

In the first case the minus associates to the left and in the second the minus associates to the left.

The default association for operators of the same precedence is from left to right (left association).

Binary operators

All standard binary operators associate to the left.

Function Type Operator

The function type operator associates to the right. For example, the type:

int -> real -> bool

is to be interpreted as the type:

int -> (real -> bool)

and not as

(int -> real) -> bool.

Operator Precedence and Association Table

Operator	Meaning	Precedence	Association	Inscription
~	arithmetic negation	1	right	yes
*	arithmetic multiplication	7	left	yes
/	arithmetic real division	7	left	yes
div	arithmetic integer division	7	left	yes
mod	arithmetic modulo	7	left	yes
^	string concatenation	6	left	yes
+	arithmetic addition	6	left	yes
-	arithmetic subtraction	6	left	yes
::	list constructor	5	right	yes
^^	list concatenation	5	left	yes
=	equality	4	left	yes
<>	relational not equal	4	left	yes

CPN ML Reference

<	relational less than	4	left	yes
>	relational greater than	4	left	yes
<=	relational less than or equal to	4	left	yes
>=	relational greater than or equal to	4	left	yes
:=	reference access	3	left	no
o	function composition	3	left	no
andalso	boolean conjunction	2	left	yes
%	boolean selector			
\	boolean selector			
#	selector			
orelse	boolean disjunction	2	left	yes
case/of			left	yes
!	reference access			no
->	function type		right	no

Expression Uses

Patterns

Frequently the elements of a compound data object must be accessed independently of one another. Often the most convenient way is to set variables to the values of the separate elements, then use the variables as needed.

ML facilitates this type of access via pattern matching. A *pattern* is an ML expression that consists only of CPN variables and type constructors. It is used in input arc inscriptions to access and bind components of compound CPN data objects.

For example, given the following colorset and variable declarations:

```
color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

color No_of_Branches = int with 0..10;

color BankImportance = product
    No_of_Branches * Banks;

var national_banks,
    regional_banks, banks:Banks;

var branch_no:No_of_Branches;

var bank_size:BankImportance;
```

The following expressions are patterns:

```
(branch_no, regional_banks)
```

```
(branch_no, banks)
```

The following expression is not a pattern:

```
(9, Chase)
```

Tuple Pattern Matching

A tuple pattern is a sequence of variables that have the same type(s) as the elements in the tuple to be matched. The variables are enclosed in parentheses and separated by commas. For example:

```
(intvar, realvar, stringvar)
```

matches a three-element tuple in which the first element is an integer, the second is a real, and the third is a string.

List Pattern Matching

A pattern to match each element in a list is a sequence of variables of the type characteristic of the list. There must be as many variables as there are list elements. The variables are enclosed in brackets and separated by commas. For example:

```
[intvar1, intvar2, intvar3, intvar4]
```

matches a four-member list of integers.

Often the length of a list is unimportant or unknown, or only the first member is of specific interest. The double-colon pattern can be used in such cases: For example, a list of integers could be matched with the pattern:

```
intvar::intlistvar
```

The first variable will be set to the first element of the list, and the second will be set to the rest of the list. If the list contained only one member, `intlistvar` would be set to the empty list.

Constructors

A constructor is an expression that when evaluated yields a compound data object. They are used in guards and output arc inscriptions to generate and bind output variables.

It is often useful to construct a compound object whose values are given by expressions. CPN ML accomplishes this through the use

of constructors. There are appropriate constructors for each compound data type.

Constructor expressions may include any combination of identifiers, operators, and type constructors. They are distinguished from other expressions in that they evaluate to a compound data object. Thus, a pattern such as (a, b, c) when used as an output arc inscription is a constructor, whereas an expression such as:

```
(abs (intvar1 + intvar2),  
  stringvar ^^ "New York",  
  realvar1 <= realvar2, 4.4)
```

is not a pattern, but is a tuple constructor.

Tuple Constructors

A tuple constructor is a sequence of expressions that evaluate to data objects of same types as the elements of the type of tuple to be constructed. The sequence is enclosed in parentheses and the expressions are separated by commas, as with tuples generally. For example:

```
(abs (intvar1 + intvar2),  
  stringvar ^^ "New York",  
  realvar1 <= realvar2, 4.4)
```

constructs a tuple whose elements are an integer, a string, a boolean, and a real.

List Constructors

A list constructor is a sequence of expressions that evaluate to the same colorset. The sequence is enclosed in square brackets and the expressions are separated by commas. For example:

```
[intvar1* intvar2, if boolvar1  
  then intvar3  
  else intvar4, 46]
```

constructs a list consisting of three integers.

Expression Evaluation in CP Nets

Arcs

Expressions used as input and output arc inscriptions yield a multi-set when evaluated.

Guards

Expressions used as guards yield a boolean value when evaluated. They may both constrain enablement and bind a CPN variable in at the same time.

Code Segments

Expressions used in code segments yield values depending on the colorset and operators of the expression components. The only restriction is that code segments must not rebind CPN variables bound on the input arcs or guards.

Further Discussion

See Chapter 40, “Inscriptions,” for a complete discussion and copious examples of their use.

Chapter 37

Multisets

Term Definitions

A *token* is a typed data object. Tokens are not called objects to avoid confusion with graphical objects.

A *multiset* is a collection of tokens. Unlike ordinary sets each token may have duplicate values, hence the name multiset.

Multiset Variables

A *multiset variable* is a CPN variable that yields a multiset when evaluated.

An ordinary CPN variable can have any value of the colorset for which the variable is defined. The value of a multiset variable is a multiset of the colorset for which the variable is defined. See Chapter 34, “CPN Variables,” for further information about CPN variables.

Syntax

The syntax for declaring a multiset variable is:

```
var id1, id2, ....., idn : colorset ms;
```

This is the same as an ordinary CPN variable declaration, except for the addition of **ms** after the colorset name.

Use

Multiset variables can be referenced in a code segment and used in output arc inscriptions. They can be used to read in multiset expressions from an input file. This expression can then be evaluated and set as the value of the variable.

Multiset Creation

The *multiset creation operator*, backquote (```), creates a multiset containing a given value, a given number of times.

Syntax

```
int` colorset-value;
```

Operator from: (int * colorset_value) -> multiset

The integer argument must be non-negative. If this is not the case, an exception is raised.

Use

The multiset operator combined with multiset addition and subtraction provides a succinct method for specifying multisets. For example:

```
2`a + 1`c + 4`f
```

is a multiset consisting of seven CPN variables: two of variable a, one of variable c, and four of variable f.

Multiset expressions

A *multiset expression* is an ML expression that when evaluated yields a multiset. Multiset expressions are used as arc inscriptions. Input arc inscriptions may consist of regular multiset expressions, including functions.

Syntax

Examples

Two integer tokens each of whose value is the integer 3:

```
2`3
```

One integer variable named

```
(1`intvar1 + 4`intvar2)
```

A tuple pattern

```
(staff_type, order_type)
```

Multiset Constants, Operations, and Functions

Constant Definitions

Empty Multiset

The **empty** constant constructs an empty multiset that is polymorphic, e.g. identical for all kinds of multisets.

```
empty;
```

Constant: -> "a ms

For example:

```
ML> empty;

> empty : "a ms
```

Full Multiset

The **colorset** constant defines a constant which contains exactly one occurrence of each value in **colorset** for the full multiset over **colorset**. The **colorset** must have been declared with **ms** and must be small.

```
colorset_name;
```

Constant: -> <colorset> ms

For example:

```
color small_int = int with 1..10 declare ms;

ML> small_int;

> !! ((1,1),!! ((1,2),!! ((1,3),!! ((1,4),
    !! ((1,5),!! ((1,6),!! ((1,7),
    !! ((1,8),!! ((1,9),
    !! ((1,10),empty)))))))) : small_int ms
```

Addition, Subtraction, and Scalar Multiplication, of Multisets

Addition (+ operator)

```
"a ms + "a ms;
```

Operator from: ("a ms * "a ms)-> "a ms

For example:

```
ML> 3`5 + 5`5;

> !! ((8,5),empty) : int ms

ML> 6`5 + 3`6;

> !! ((3,6),!! ((6,5),empty)) : int ms
```

Subtraction (- operator)

```
"a ms - "a ms;
```

Operator from: ("a ms * "a ms)-> "a ms

The first multiset must be greater than or equal to the second. If this is not the case, an exception is raised.

The second multiset must be a subset of the first multiset. If this is not the case, an exception is raised.

For example:

```
ML> 6`5 - 3`6;

> Failure: CPN'IllegalMultiSetSubtr

ML> 6`5 - 3`5;

> !! ((3,5),empty) : int ms
```

Scalar Multiplication (* operator)

For scalar multiplication the integer must be non-negative.

```
int * "a ms;
```

Operator from: (int * "a ms)-> "a ms

For example:

```
ML> 3*2`5;

> !! ((6,5),empty) : int ms

ML> 4`5 - 3`5;

> !! ((1,5),empty) : int ms
```

Multiplication of Multisets

`mult'colorset` can only be declared for tuple and record colorsets. It constructs a multiset over `colorset` by multiplying a set of multisets (one for each base colorset).

```
mult'colorset (colorset1 ms, colorset2 ms,
....., colorsetn ms);
```

Function from: $(\text{colorset}_1 \text{ ms} * \text{colorset}_2 \text{ ms} * \dots * \text{colorset}_n \text{ ms}) \rightarrow \text{colorset ms}$

For example:

```
color integer = int;
color realno = real;
color tupleno = product integer * realno;

ML> mult'tupleno
      ((1`3 + 4`6), (3`3.0 + 4`6.0));

> !! ((3,(3,3.0)),!! ((4,(3,6.0)),!!
      ((12,(6,3.0)),!!
      ((16,(6,6.0)),empty)))) : baz ms
```

Comparing Multisets

These operations compare two multisets.

Equality (== operator)

```
"a ms == "a ms;
```

Operator from: $(\text{"a ms} * \text{"a ms}) \rightarrow \text{bool}$

For example:

```
ML> 6`5 == 8`5;

> false : bool
```

Not Equal (<><> operator)

```
"a ms <><> "a ms;
```

Operator from: $(\text{"a ms} * \text{"a ms}) \rightarrow \text{bool}$

For example:

```
ML> 6`5 <><> 8`5;
```

```
> true : bool
```

Less Than or Equal (<= operator)

```
"a ms <= "a ms;
```

Operator from: ("a ms * "a ms) -> bool

For example:

```
ML> 8`5 <= 8`5
```

```
> true : bool
```

Less Than (<< operator)

```
"a ms << "a ms;
```

Operator from: ("a ms * "a ms) -> bool

For example:

```
ML> 8`5 << 8`5
```

```
> false : bool
```

```
ML> 6`5 << 8`5
```

```
> true : bool
```

Greater Than or Equal (>= operator)

```
"a ms >= "a ms;
```

Operator: ("a ms * "a ms) -> bool

For example:

```
6`5 >= 8`5;
```

```
> false : bool
```

Greater Than (>> operator)

```
"a ms >> "a ms;
```

Operator: ("a ms * "a ms) -> bool

For example:

```
ML> 6`5 >> 8`5;
> false : bool
```

Coefficient

cf returns the coefficient of a given value in a given multiset.

```
cf ("a, "a ms);
```

Function from: ("a * "a ms)-> int

For example:

```
ML> cf (3, (5`6 + 3`6));
> 0 : int
```

Size

size returns the number of elements in a multiset.

```
size ("a ms);
```

Function from: "a ms-> int

For example:

```
ML> size (3`6 + 4`7);
> 7 : int
```

Random Value From a Multiset

random returns a random value from a given multiset. The probability for the selection of a value is proportional to its coefficient in the multiset.

random cannot be applied in net inscriptions such as arc inscriptions and guards. It can, however, be applied in code segments.

```
random ("a ms);
```

Function from: "a ms -> "a

For example:

```
ML> random (3`6 + 4`7);  
      > 6 : int
```

Conversion Between Lists and Multisets

list_to_ms and **ms_to_list** provide the conversion between the two types.

```
list_to_ms ["a, "b, ..., "c];
```

Function from: "a list -> "a ms

For example:

```
ML> list_to_ms [1,3,4];  
      > !! ((1,1),!! ((1,3),!! ((1,4),empty)))  
          : int ms
```

```
ms_to_list ("a ms);
```

Function from: "a ms -> "a list

For example:

```
ML> ms_to_list (1`3 + 4`6);  
      > [6,6,6,6,3] : int list
```

Filter Function

filter constructs a filter (a function mapping multisets into smaller or identical multisets) from a predicate (a function mapping values into booleans).

```
filter ('a bool);
```

Function from: ('a -> bool) -> ("a ms -> "a ms)

Linear Extension of Functions

ext_col and **ext_ms** make a *linear extension* of a function.

```
ext_col ('a ''b);
```

Function from: ('a -> "b) -> ("a ms -> "b ms)

```
ext_ms ('a ' 'b ms);
```

Function from: ('a -> "b ms) -> ('a ms -> "b ms)

Multiset Input and Output

input_ms and **output_ms** allow code segments to read and write multisets.

```
input_ms (instream);
```

Function from: instream -> colorset ms

```
output_ms (outstream, <colorset> ms);
```

Function from: (outstream * colorset ms)-> unit

Internal Representation of Multisets

Design/CPN represents each multiset as a nested sequence of pairs, separated by double exclamation points (!!) where each pair contains a coefficient followed by a value. The constant `empty` denotes the empty multiset and terminates the sequence.

For example:

```
ML> 2`3 + 1`4 + 1`5 + 3`6;

> !! ((3,6),!! ((1,5),!! ((1,4),
    !! ((2,3),empty)))) : int ms
```

A multiset representation is said to be:

- Compressed, if each color occurs in at most one pair, with a positive coefficient.
- Sorted, if the colors occur in increasing order with respect to the ordering of the colorset.
- Normalized, if it is both compressed and sorted.

Design/CPN assumes all multiset representations to be compressed, but it does not assume them to be sorted.

When you apply the predeclared constants, operations, and functions, all multiset representations automatically become compressed.

When you define your own multisets or functions (between multisets) — by means of the internal representation of multisets — you *must* define them in such a way that all multiset representations become compressed. An easy way to do this is to apply the predeclared **compress** function.

Construct, Compress, and Sort

The following operations/functions are only necessary for users who want to work with the internal representation of multisets. The **!!** operation allows you to construct — and pattern match — the internal representation of multisets. The **compress** function allows you to compress a multiset representation.

Multiset constructor (!! operator)

```
!! ((int * "a) * "a ms);
```

Operator from : ((int * "a) * "a ms) -> "a ms

```
compress    "a ms
```

Function from: "a ms -> "a ms

Timed Multisets

To support simulation with time, a new colorset is declared, when a time declaration is given:

```
colorset "a tms = !!! of
```

```
((int * "a * (TIMElist)) * ("a tms)) | empty;
```

Furthermore, **+**, **-**, **size**, **cf** and **compress** are overloaded to support timed multisets, when a time declaration is given.

Chapter 38

Functions

Declarations

Syntax

The syntax is:

```
fun identifier (parameter-tuple)  
    = function-body;
```

identifier is associated with the **function-body**.

Datatype

The datatype of a function is the combined datatype of the domain (parameter) and the range (result). It is indicated by the reserved word **fn** followed by an expression whose syntax is:

```
fn parameter-type -> result-type
```

The function operator, **->**, indicates the mapping between the domain and range of the function.

Example

The following example increments its parameter by 1:

```
fun sum1 (n) = n + 1;
```

Local Declarations - Let

The **let** construct permits the declaration of locally-scoped variables within a function definition. In addition, **let** may be used within a code segment. This use provides the only case in CPN ML where **val** may be used outside of a declaration node.

Syntax

```
let val_declaration in expression end;
```

where `val_declaration` is a value declaration using `val`, and `expression`, is a CPN ML expression. See Chapter 33, “Values,” for a discussion of `val`, and Chapter 36, “Expressions,” for a discussion of CPN ML expressions.

Control Structures

if-then-else

if-then-else provides a conditional control structure that may be used in function declarations, inscriptions, and code segments.

Syntax

```
if boolean-expression  
then expression  
else expression;
```

The uses of the if-then-else construct determine which expressions are legal in the then and else clauses:

Input Arc Inscription: CPN variables may not be bound by a conditional input arc inscriptions; they must be bound somewhere else on the transition - either on a different input arc or in the guard.

Guard: Each expression must be a boolean expression, evaluating to true or false. Thus the following example is legal because each variable binding is either true or not true:

```
[if a = 3 then b = 4 else c = 5]
```

The following example is not legal because the then and else clauses evaluate to integers:

```
[if a = 3 then 4 else 5]
```

Output Arc Inscription: CPN variables may be bound by a conditional output arc inscription.

Boolean Selectors

These operations allow you to apply a *boolean selector* to a value, a multiset, a pair of values or a pair of multisets. The first two op-

erations work as an if-then command, while the last two work as an if-then-else.

```
% (bool list * 'a) -> 'a ms
/ (bool list * 'a ms) -> 'a ms
```

Case

The `case` structure is a conditional expression generators that provides an if-then-else, if-then-else,... capability.

Syntax

```
case identifier of
  boolean_test1 |
  boolean_test2 |
  ...
end;
```

The evaluation proceeds until there is a match. If none of the tests succeed then the result of evaluation is false.

Function Invocation

To use a function, apply it to its argument, which must be in the domain of the function. For example, to add 1 to the number 100: **sum1**to the argument 100:

```
ML> sum1 (100);
> 101 : int
```


Chapter 39

Timing

Time Stamps

For simulation with time, each token may, in addition to its token value, carry a *time stamp*, that is, an integer or real expression. The time stamps are calculated in the CPN Simulator by adding the transition delay, if any, and the arc delay, if any, to the model time at which the token is created.

Declaration

Each colorset that is to be timed must append the keyword `timed` to the colorset declaration. For example:

```
color A = with id1 | id2 | id3 | id4 timed;
```

Defaults for Time Stamps

Index, enumeration, and simple colorsets are untimed by default.

Compound colorsets (that is, unions, products, records, and lists) are time if and only if at least one of the base colorsets is timed. In the case where more than one base colorset is timed, the compound colorset will still only have a single time stamp for each token.

Duplicate and subset colorsets are timed by default if and only if the base colorset is timed.

To make a default timed colorset non-timed, append the keyword **untimed** to the colorset declaration. For example:

```
color colorset = colorset0 untimed;
```

Timed Arc Inscriptions

Input Arc Inscriptions

To ignore the time stamps of tokens from an input place whose colorset is timed, append the keywords `@ignore` to the input arc inscription. For example:

The following declaration creates a timed colorset:

```
color Banks = with HarvardTrust |  
              Shawmut | NationalGrand timed;
```

so that an input place with `Banks` as a colorset is timed. The following input arc inscription:

```
2`Banks@ignore
```

causes the simulator to ignore that specification.

Input arc inscriptions may not specify time delays.

Output Arc Inscriptions

Output arc inscriptions may specify time delays by appending a time delay expression to the output arc inscription. The time delay expression consist of the keyword `@+` followed by an integer or real expression. For example, the following output arc inscription:

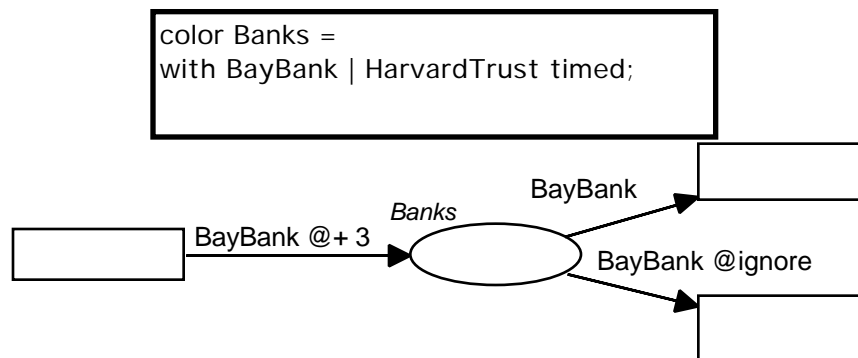
```
2`Banks @+5
```

causes the simulator to put a time delay on the tokens going into the output place. A missing arc delay is equivalent to a zero time delay.

Output arc delays can use CPN variables and the function `time`.

Combined Use

The CP net below shows an example of how the input and output arc time delays can interact:

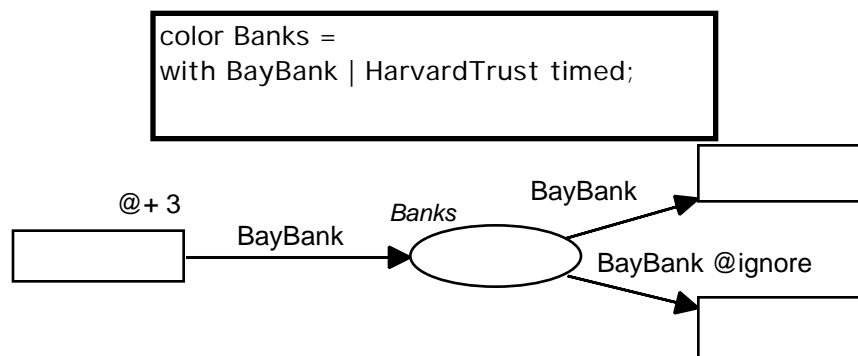


Time region

The time region is a region of a transition that specifies a time delay. Transitions may specify time delays by specifying an integer or real expression preceded by `@+`. For example, the following transition time delay:

`@+5`

causes the simulator to put a time delay of + 5 time units on all tokens leaving the transition. The example CP net above could be expressed with a transition time region instead of a timed output arc inscription:



A missing arc delay is equivalent to a zero time delay.

Time regions can use all the CPN variables of the corresponding transition. This means that the time delay may depend upon the token values for input and output tokens.

Moreover, the expression may, via calculated CPN variables, depend upon reference variables and input files. (See the discussion of calculated CPN variables in the code segment discussion in Chapter 40, "Inscriptions.")

Finally, the time region is also allowed to use `time()`.

Time-Related Functions

Append time list to one-element multiset (@ operator)

The @ operator takes a one-value multiset and appends a list of time stamps. The length of the list must be equal to the number of occurrences of the value in the multiset. The function can only be used if time has been declared. If the multiset has more than one value, an exception is raised.

```
multiset_specification @  
  [integer_list_specification];
```

Function from: ("a ms * int list) -> "a tms

For example:

```
ML> 2`6 @[1,2];  
  
> !!! ((2,6,[1,2]),empty) : int tms
```

Time and step

The current model time and the step number can be inspected by means of the functions:

```
time ();
```

Function from: unit -> integer or real

```
step ();
```

Function from: unit -> integer

Time and **step** can be used in initial marking, arc inscription, guard, code and time regions, with the following exceptions: **Time** cannot be used in input arc inscriptions, guards and code guards, while **step** may only be used for reporting purposes in code segment action clauses.

Simulation options

with_time tests whether or not the simulation option *Simulation with time* is currently in force.

```
with_time ();
```

Function from: unit -> bool

Chapter 40

Inscriptions

This section describes the inscriptions associated with CPN net components. An *inscription* is a CPN ML construct that affects the behavior of the net. Inscriptions vary in their syntactic requirements depending on the type of inscriptions. The inscription types include:

- Colorset region (place)
- Initial marking region (place)
- Arc inscription region (arcs)
- Time region (transition)
- Guard region (transition)
- Code segment (transition)

We present the inscriptions in the context of a sample model in which the regions associated with each place, arc, and transition are shown and briefly described. Each inscription is defined along with its syntax and one or more examples.

Example Conventions

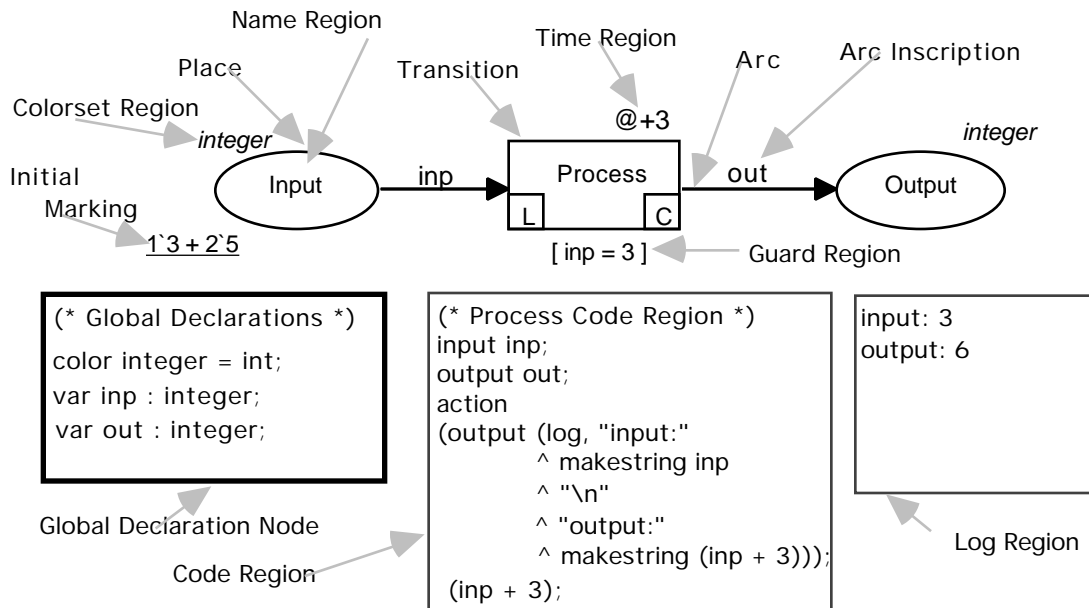
In the examples in this chapter we use the default typographic conventions used by CPN:

- Initial markings are underlined
- Colorset regions are italic

In addition, guards are always enclosed in square brackets to distinguish them from other inscriptions. Square brackets are only required for guards when they consist of a list of expressions.

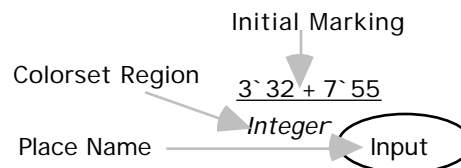
Example Diagram

The diagram below has an example of each inscription and region we will discuss in this chapter. The global declaration node is not an inscription, but is included because it is an essential part of any model.



Places

There are three regions that may be associated with a place. Two are optional and one is required:



- Name region - optional
- Colorset region - required
- Initial marking region - optional

Place Name Region

The name region is a label that identifies the place. It is not a CPN ML inscription and may contain any sequence of characters. The

Syntax Options dialog (**Set** menu) provides the additional validation criteria:



Syntax Options

Optional Syntax Restrictions

Report Errors for:

- ☒ **Missing Place Names**
- ☐ **Duplicate Place Names**
- ☐ **ML-illegal Place Names**

Colorset Region

The colorset region contains the identifier of a colorset declared in a global or temporary declaration node. The colorset determines the colorset of all the tokens that can be put in the place.

Initial Marking Region

The *initial marking* is a multiset expression that specifies the initial tokens for a place. The colorset for the expression must match the colorset of the place. If the colorset is timed, the initial marking can specify a time delay.

The initial marking is optional. When an initial marking is absent, it is equivalent to empty, the empty multiset.

Optional Time Delays

An initial marking time delay is an expression of type **TIME** that is appended to the initial marking expression with **@+** as a separator. A missing initial marking time delay is equivalent to a zero delay.

When an initial state is generated, each timed token gets a time stamp which is equal to the start value of model time plus the initial marking delay of the corresponding place. This ensures that all the initial tokens get the same time stamp.

Syntax

The initial marking region has the form:

```
initial-marking <<@+ init-mark-delay>>
```

where *initial-marking* is a multiset expression and *init-mark-delay* is an expression whose SML datatype is integer or real.

Examples

Example 1: In the example below *bank*, the colorset for the place *bank_input*, is declared:

```
color bank =  
  with Fleet | BayBank | Shawmut | NationalGrand;  
  
bank  
  (bank_input)  
  2`Fleet
```

2`Fleet specifies two initial tokens each of whose value is the constant *Fleet*.

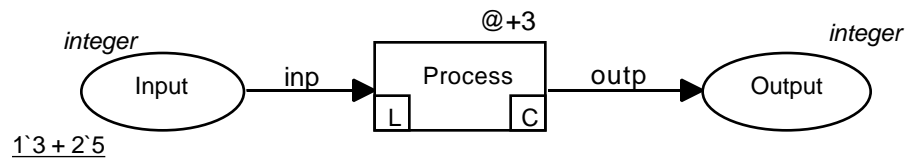
Example 2: In the example below *bank*, the colorset for the place *bank_input*, is declared:

```
color bank =  
  with Fleet | BayBank | Shawmut | NationalGrand  
  timed;  
  
bank  
  (bank_input)  
  2`Fleet @+ 10
```

2`Fleet specifies two initial tokens each of whose value is the constant *Fleet*. *@+10* specifies a time delay of 10 time units.

Arcs

Arcs have one region associated with them - the arc inscription region. The following figure shows two arc inscriptions - one input arc inscription and one output arc inscription. Each arc inscription has a CPN variable, *inp*, of colorset integer.



Arc Inscription Region

An arc inscription is a CPN ML expression that evaluates to a multiset.

A missing arc inscription is equivalent to `empty (tempty)`, the empty (timed) multiset.

Patterns and Constructors

See Chapter 36, “Expressions,” for a discussion of patterns and constructors as arc inscriptions.

Side Effects

Arc inscriptions are *not* allowed to have side effects and cannot:

- Contain input/output or `use` commands.
- Update or use reference variables.
- Use the `random` function.

This restriction is currently not checked by Design/CPN and the system may malfunction if the restriction is violated.

Free variables are output arc variables that have not been bound on an input arc or in the guard. They are assigned random values when executing in interactive mode.

Caution: Do not use free variables when executing in automatic mode, since that will stop the simulator.

Time Stamps

An input arc inscription can override a time stamp on an input place by specifying `ignore`. The syntax is:

```
arc-inscription@ignore.
```

Specifying `ignore` causes time stamps not to be taken into account when the time enabling of the transition is calculated.

An output arc delay must be an expression of type `TIME` and it must be appended to an output arc inscription, using `@+` as a separator. The syntax is:

```
arc-exp @+arc-delay.
```

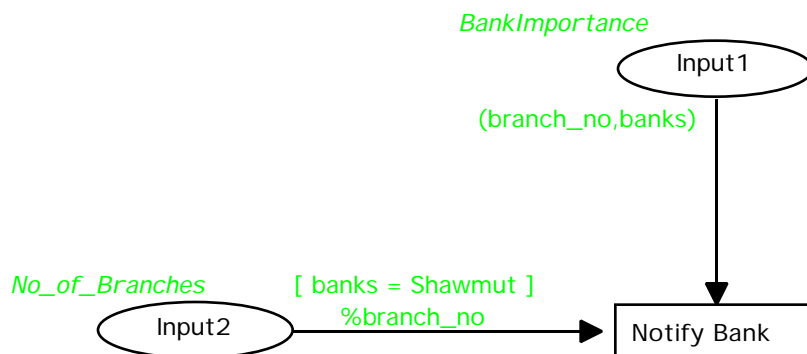
An omitted output arc delay is equivalent to a zero delay.

Arc delays can, in exactly the same way as transition delays, use CPN variables and `time()`.

Conditional arc inscriptions

Conditional arc inscriptions may be used on input arcs, if all the variables are bound in the guard or in another input arc.

In the example below, the input arc inscription from `Input2` can use conditional boolean selectors (`%`, which means "if then") because the variables `branch_no` and `banks` are bound on the input arc from `Input1`. See Chapter 38, "Functions," for a discussion of boolean selectors and the `if-then-else` structure.



Conditional arc inscriptions on output arcs may bind output variables mentioned in the output clause of the code segment.

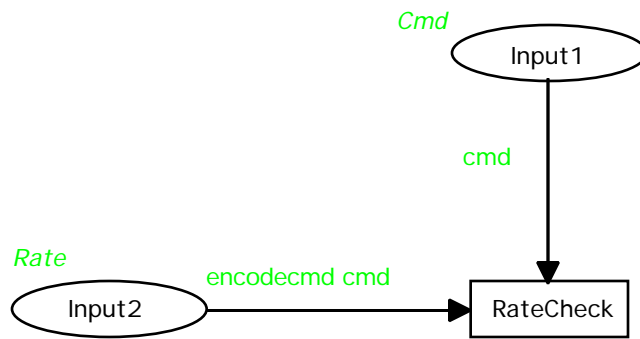
Function application

Functions used in arc inscriptions must not have side effects.

Functions may be used on input arcs only if the function parameters are bound elsewhere at that transition.

Functions on output arc inscriptions may bind output variables.

In the example below, the input arc inscription from `Input2` can use a function because the variable `cmd` is bound on the input arc from `Input1`:



Input Arc Inscription Examples - Declarations

Colorset Declarations

```

(* Enumerated colorset *)

color Banks = with HarvardTrust | Chase
               | MarbleheadSavings | Shawmut
               | NationalGrand;

(* Indexed colorset *)

color BankBranch =
  index branch_number with 1...8;

(* Constrained integer colorsets *)

color No_of_Branches = int with 0..10;

color No_of_Customers = int with 0..99999;

(* Tuple colorset *)

color BankImportance = product
  No_of_Branches * Banks;

(* Record colorset *)

color BankImportanceRec = record
  Size:No_of_Branches *
  Name:Banks;

(* Union colorset *)

color RegionBanksSize = union Name:Banks +
  CustomerBase : No_of_Customers +
  BranchNumbers : No_of_Branches;

(* List colorset *)

color BankList = list Banks;
  
```

```
(* Subset colorset *)

color LocalBanks = subset Banks
                  with [NationalGrand,
                      MarbleheadSavings];
```

Variable declarations

```
(* enumerated colorset *)

var national_banks,
    regional_banks, banks:Banks;

(* indexed colorset *)

var branch:Bank_Branch;

(* integer colorset *)

var branch_no:No_of_Branches;

(* integer colorset *)

var customers : No_of_Customers;

(* tuple colorset *)

var bank_size:BankImportance;

(* record colorset *)

var bank_size_rec :BankImportanceRec;

(* union colorset *)

var regionbank:RegionBanksSize;

(* list colorset *)

var banklist:BankList;

(* subset colorset *)

var local_bank : LocalBanks;
```

Input Arc Inscription Examples

Constant:

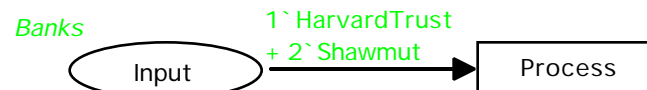
```
color Banks = with HarvardTrust | Chase
                | MarbleheadSavings | Shawmut
                | NationalGrand;
```



Multiset expression with enumerated values:

```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;
    
```



Variable:

```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

var banks:Banks;
    
```

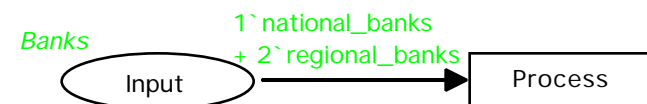


Multiset expression with variables:

```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

var national_banks,
    regional_banks, banks:Banks;
    
```

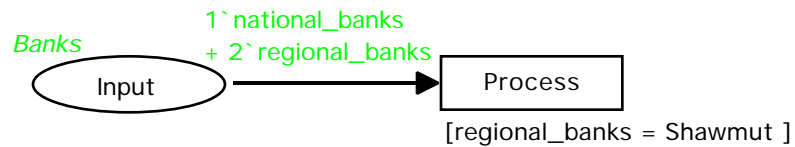


Multiset expression using variables constrained by a guard:

```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

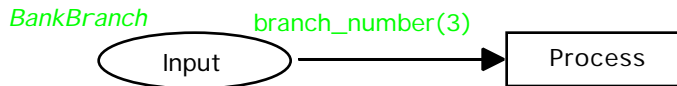
var national_banks,
    regional_banks, banks:Banks;
    
```



Indexed values:

```
color BankBranch =
    index branch_number with 1...8;

var branch_no:No_of_Branches;
```



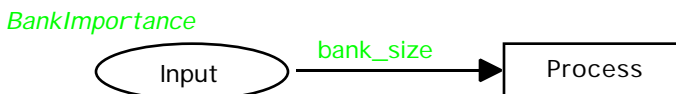
Tuple colorset, CPN variable arc inscription:

```
color No_of_Branches = int with 0..10;

color Banks = with HarvardTrust | Chase
               | MarbleheadSavings | Shawmut
               | NationalGrand;

color BankImportance = product
    No_of_Branches * Banks;

var bank_size:BankImportance;
```



Tuple colorset, pattern arc inscription:

```
color No_of_Branches = int with 0..10;

color Banks = with HarvardTrust | Chase
               | MarbleheadSavings | Shawmut
               | NationalGrand;

color BankImportance = product
    No_of_Branches * Banks;
```



Record colorset, pattern arc inscription with variables:

```

color No_of_Branches = int with 0..10;

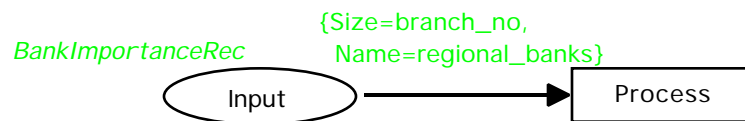
color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

color BankImportanceRec = record
    Size:No_of_Branches *
    Name:Banks;

var branch_no:No_of_Branches;

var regional_banks:Banks;

```



List colorset, list constructor arc inscription:

```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

color BankList = list Banks;

var banks:Banks;

var bank_list:BankList;

```



List colorset, list pattern arc inscription:

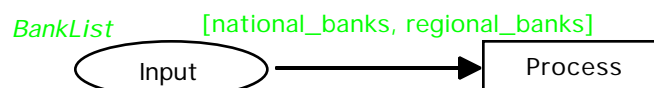
```

color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

color BankList = list Banks;

var national_banks,
    regional_banks:Banks;

```



Union colorset:

In this example, the union colorset `RegionBanksSize` is used in an input place that sends different components of the union to different transitions for different kinds of processing. The output places of those transitions have colorsets corresponding to the inputs that came into the transitions.

The colorset of the first place, `Output1`, is simply the colorset of one of the union colorset components - `No_of_Customers`. The colorset of the second place, `Output2`, is a tuple, `BankImportance`, that combines two of the union colorset components - `No_of_Branches`, and `Banks`.

```
color Banks = with HarvardTrust | Chase
              | MarbleheadSavings | Shawmut
              | NationalGrand;

color No_of_Customers = int with 0..99999;

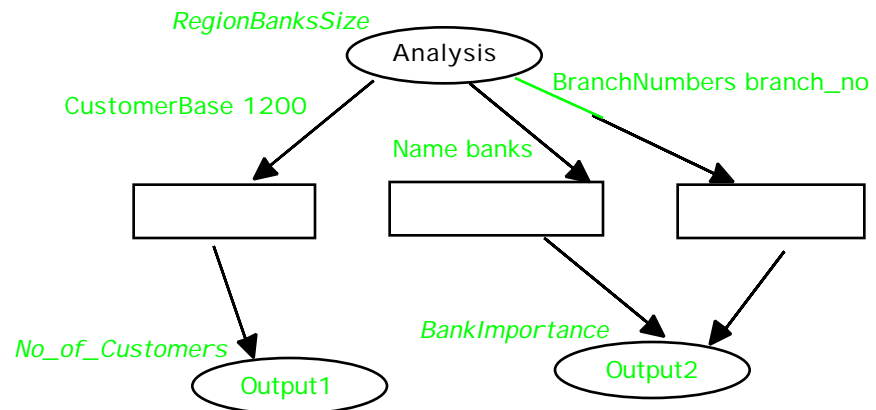
color No_of_Branches = int with 0..10;

color BankImportance = product
    No_of_Branches * Banks;

color RegionBanksSize = union Name:Banks +
    CustomerBase : No_of_Customers +
    BranchNumbers : No_of_Branches;

var banks:Banks;

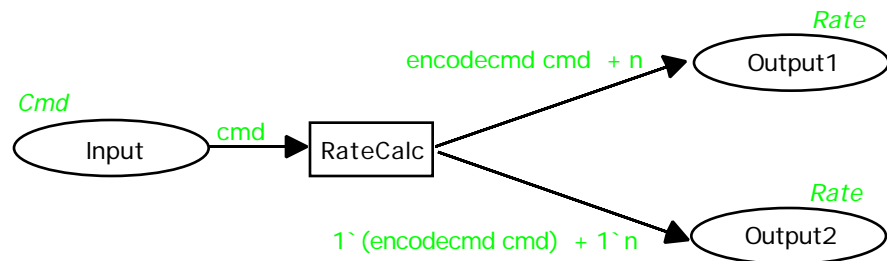
var branch_no:No_of_Branches;
```



Output Arc Inscription Examples

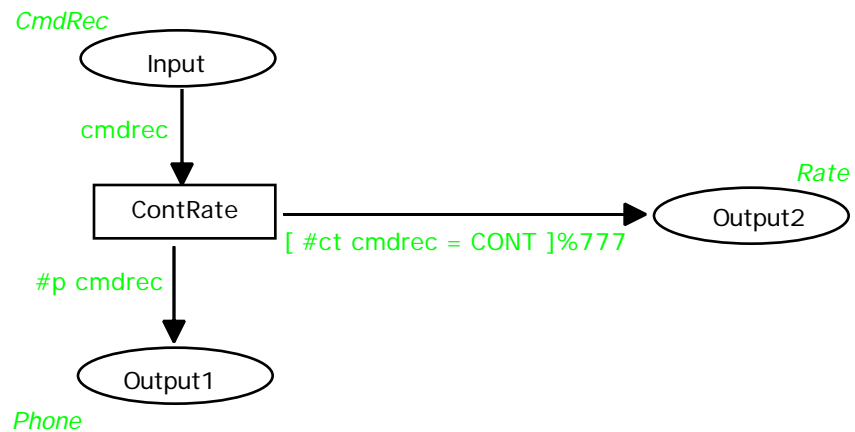
Function application

In this example the first output arc has one token, the second output arc has two. Note the use of free variables.

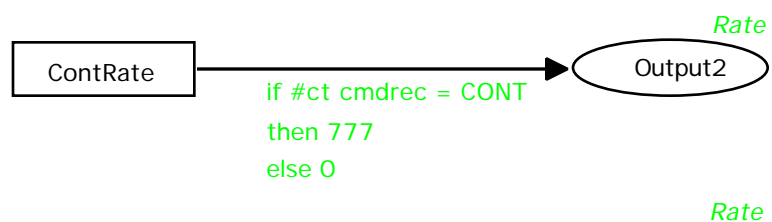


Conditional output arc inscriptions

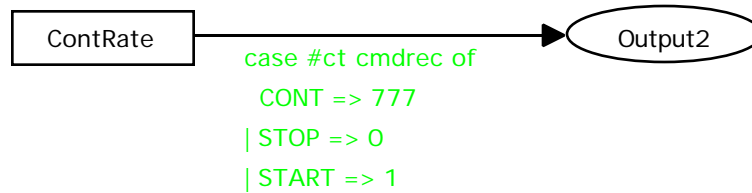
Example 1



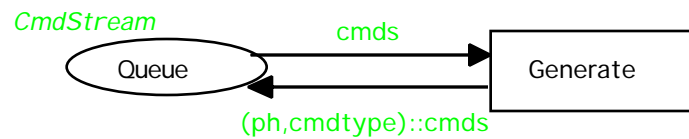
Example 2



Example 3

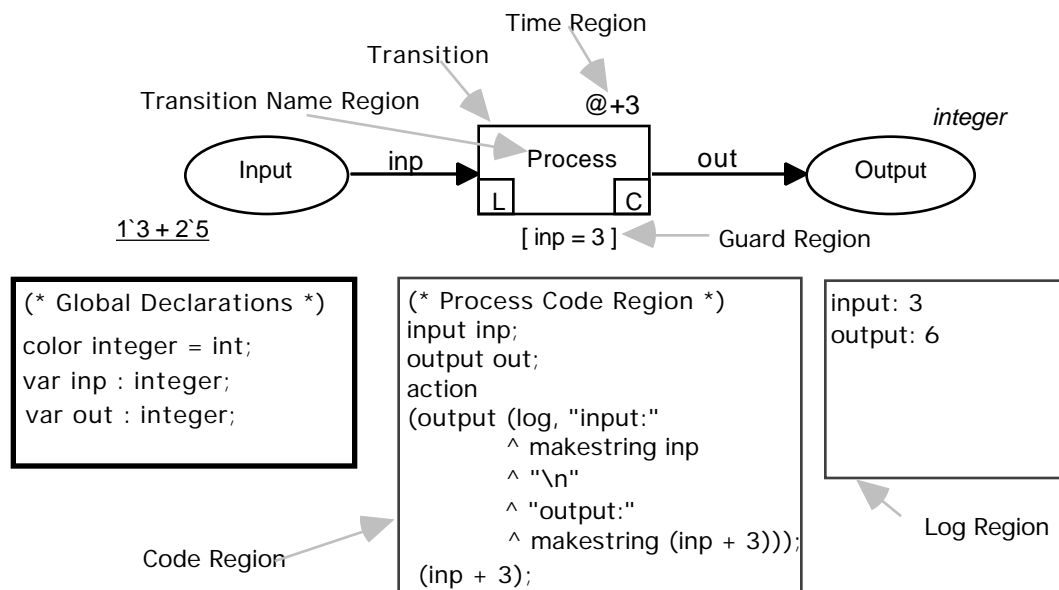


Unbound variables on an output arc



Transition Regions

There are five regions that may be associated with a transition. All are optional:



Transition Name Region

The name region is a label that identifies the transition. It is not a CPN ML inscription and may contain any sequence of characters. The **Syntax Options** dialog (**Set** menu) provides the additional validation criteria.

Time Region

A transition delay must be a positive integer or real expression. The expression is preceded by @+ and this means that the time region has the form @+ delay-expr.

Time delay is always added relative to the current time. for example, if current time is 10 and the time delay is @+2 then the time stamp of tokens sent to the output place will be 12.

A missing time region is equivalent to a zero delay.

The time delay expression can use all the CPN variables of the corresponding transition. This means that the time delay may depend upon the token values for input and output tokens.

Moreover, the expression may, via calculated CPN variables, depend upon reference variables and input files.

The time region may use the random function and the time function.

Guard Region

An guard is a CPN ML boolean expression that evaluates to a true or false.

Syntax:

- a boolean expression or
- a list of boolean expressions: [b-exp₁, b-exp₂, ...]

Use

Guards are used for *tests* on input arc inscription variables (enabling restrictions), typically with the following syntactical elements:

= <> <= >= < >
andalso orelse

Guards are also used to restrict values of output arc inscription variables, based on the values of the input arc inscription variables.

Side Effects

Guards are *not* allowed to have side effects and cannot:

- Contain input/output or use commands.
- Update or use reference variables.
- Use the random function.

This restriction is currently *not* checked by Design/CPN and the system may malfunction if the restriction is violated.

Conditional Expressions

Each expression used in a guard must be a boolean expression, evaluating to true or false. Thus the following example is legal because each variable binding is either true or not true:

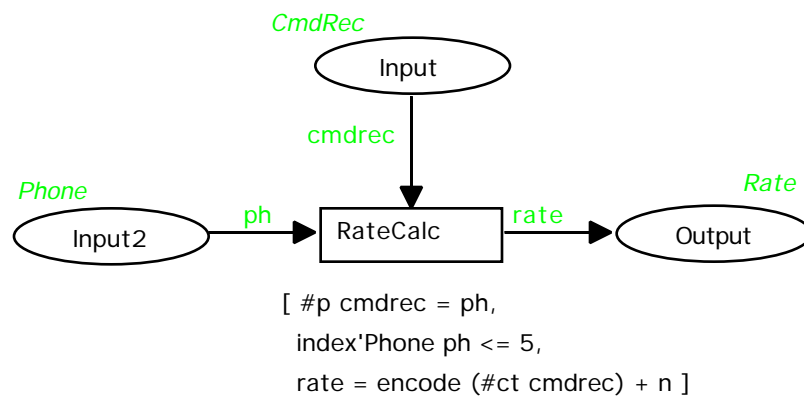
```
[if a = 3 then b = 4 else c = 5]
```

The following example is not legal because the then and else clauses evaluate to integers:

```
[if a = 3 then 4 else 5]
```

Example

A transition and a guard with two tests, providing a value to the output variable rate.



Code Segment

Each transition may have a code segment, which is a sequential piece of CPN ML code, executed each time the transition occurs.

Typical uses of transition code segments:

- Instrumentation (for example, statistics, tests, reports)
- Communication (for example, with the user through dialogs and the files)
- Complex calculations
- Graphic animation

CPN Variable Restrictions

Code segments may use CPN variables that appear on the input arc and in the guard. However, code segments may not change the value of input arc and guard variables since the input arc and guard variables are already bound by the time the code segment is executed.

Code segments may calculate the value of CPN variables that appear on the output arc since these are not bound at the time the code segment is executed.

Syntax

The code segments have syntax restrictions. Each code segment may contain an ordered list of one each of the following:

- Input clause (optional)
- Output clause (optional)
- Code action clause (mandatory)

Input Clause

The input clause is identified by the keyword `input` and is a CPN variable or a tuple of CPN variables without repetitions.

The input pattern lists the CPN variables that can be used in the code action clause. If the input clause is omitted, it implies that no CPN variables can be applied in the code action.

As described above, under **CPN Variable Restrictions**, the code action can apply the values of these CPN variables but it cannot change them.

The CPN variables listed in the input pattern can be used in the code action even if you have declared an SML identifier with the same name in the declaration node.

Output Clause

The output clause is identified by the keyword `output` and is a CPN variable or a tuple of CPN variables without repetitions.

The output clause lists the CPN variables that are to be calculated in the code action clause; these variables are said to be calculated CPN variables. If the output clause is omitted, no CPN variables may be calculated.

The output clause is a tuple whose type is the cartesian product of the types of the variables. The result of evaluating the action expression is bound to the output variables in the order listed in the output clause.

Code Action Clause

The code action clause is identified by the keyword `action` and is a CPN ML expression.

The code action cannot contain any colorset, CPN variable, or reference variable declarations. However, new functions and constants can be defined for local use by means of the `let/in/end` construct, as shown in Example 1. See Chapter 38, “Functions,” for a discussion of the `let/in/end` construct.

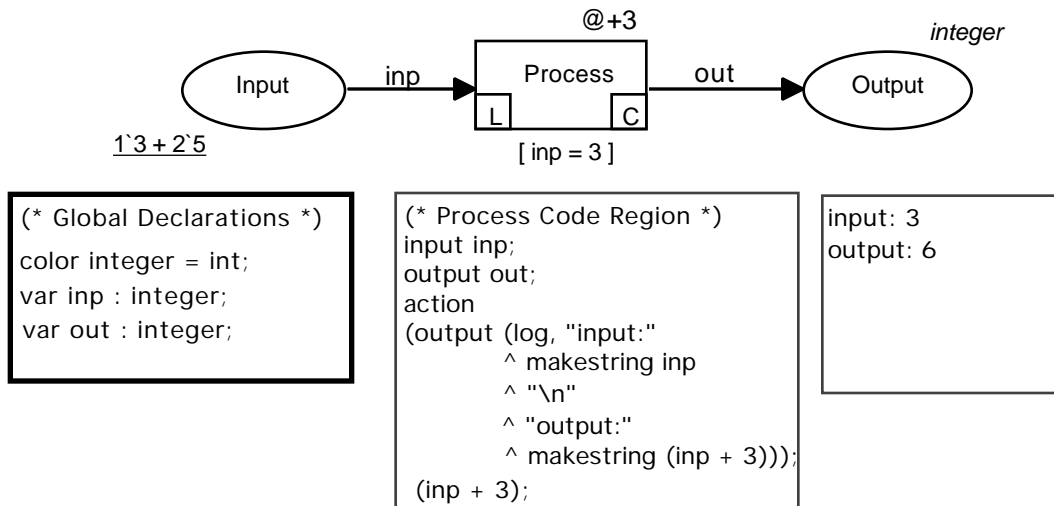
In addition, it can apply user-declared and predeclared constants, operations, and functions.

The code action is executed as a local declaration in an environment enriched by the input arc and guard CPN variables. This guarantees that the code action cannot directly change any CPN variables but only local copies of them. When the code action has been executed, its result is applied to bind the output arc CPN variables.

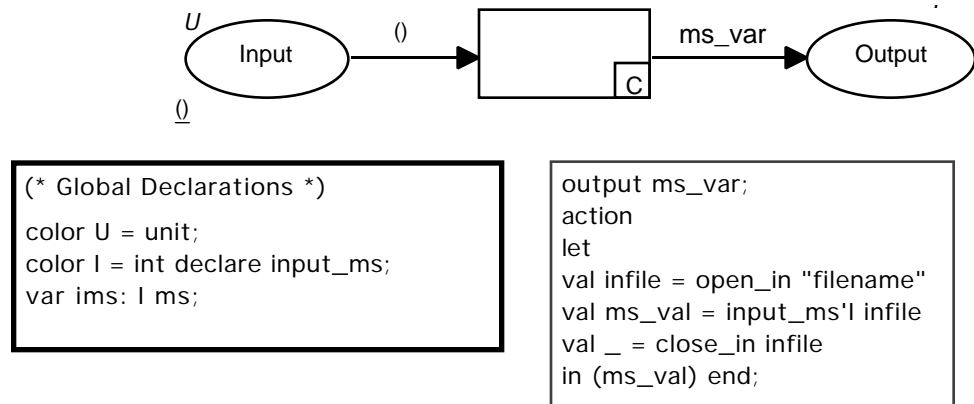
If no output clause is given, its type is assumed to be unit.

Examples

Example 1:



Example 2:



Chapter 41

The CPN ML Runtime Environment

Predeclared Environment

Design/CPN automatically generates a runtime environment containing a set of predeclared constants, operations and functions which you can apply to manipulate values, multisets and functions:

- A *constant* is an identifier with a fixed value.
- A *function* always takes one argument (at a time), and its name is written before the argument. The argument may be a tuple, for example: `dist 'A (c,f)`.
- An *operation* always takes exactly two arguments, and the name of the operation is written between the arguments, for example: `3 + 7`.

Predefined Constants, Functions, and Operators

Empty Multiset

The **empty** constant constructs an empty multiset that is polymorphic, e.g. identical for all kinds of multisets.

```
empty;
```

Constant: `-> "a ms`

Addition, Subtraction, and Scalar Multiplication, of Multisets

Addition (+ operator)

```
"a ms + "a ms;
```

Operator from: `("a ms * "a ms)-> "a ms`

Subtraction (- operator)

```
"a ms - "a ms;
```

Operator from: ("a ms * "a ms)-> "a ms

The first multiset must be greater than or equal to the second. If this is not the case, an exception is raised.

The second multiset must be a subset of the first multiset. If this is not the case, an exception is raised.

Scalar Multiplication (* operator)

For scalar multiplication the integer must be non-negative.

```
int * "a ms;
```

Operator from: (int * "a ms)-> "a ms

Multiplication of Multisets

mult'colorset can only be declared for tuple and record colorsets. It constructs a multiset over colorset by multiplying a set of multisets (one for each base colorset).

```
mult'colorset (colorset1 ms, colorset2 ms,  
....., colorsetn ms);
```

Function from: (colorset₁ ms * colorset₂ ms
* * colorset_n ms)-> colorset ms

Comparing Multisets

These operations compare two multisets.

Equality (== operator)

```
"a ms == "a ms;
```

Operator from: ("a ms * "a ms) -> bool

Not Equal (<><> operator)

```
"a ms <><> "a ms;
```

Operator from: ("a ms * "a ms) -> bool

Less Than or Equal (<= operator)

```
"a ms <= "a ms ;
```

Operator from: ("a ms * "a ms) -> bool

Less Than (<< operator)

```
"a ms << "a ms ;
```

Operator from: ("a ms * "a ms) -> bool

Greater Than or Equal (>= operator)

```
"a ms >= "a ms ;
```

Operator: ("a ms * "a ms) -> bool

Greater Than (>> operator)

```
"a ms >> "a ms ;
```

Operator: ("a ms * "a ms) -> bool

Multiset Creation

The *multiset creation operator*, backquote (`), creates a multiset containing a given value, a given number of times.

```
int` colorset-value ;
```

Operator from: (int * colorset_value) -> multiset

The integer argument must be non-negative. If this is not the case, an exception is raised.

Coefficient

cf returns the coefficient of a given value in a given multiset.

```
cf ("a, "a ms) ;
```

Function from: ("a * "a ms) -> int

Size

size returns the number of elements in a multiset.

```
size ("a ms);
```

Function from: "a ms-> int

Random Value From a Multiset

random returns a random value from a given multiset. The probability for the selection of a value is proportional to its coefficient in the multiset.

random cannot be applied in net inscriptions such as arc inscriptions and guards. It can, however, be applied in code segments.

```
random ("a ms);
```

Function from: "a ms -> "a

Conversion Between Lists and Multisets

list_to_ms and **ms_to_list** provide the conversion between the two types.

```
list_to_ms ["a, "b, ..., "c];
```

Function from: "a list -> "a ms

Filter Function

filter constructs a filter (a function mapping multisets into smaller or identical multisets) from a predicate (a function mapping values into booleans).

```
filter ('a bool);
```

Function from: ('a -> bool) -> ("a ms -> "a ms)

Linear Extension of Functions

ext_col and **ext_ms** make a *linear extension* of a function.

```
ext_col ('a "b);
```

Function from: ('a -> "b) -> ("a ms -> "b ms)

```
ext_ms ('a "b ms);
```

Function from: ('a -> "b ms) -> ("a ms -> "b ms)

Multiset Input and Output

`input_ms` and `output_ms` allow code segments to read and write multisets.

```
input_ms    (instream);
```

Function from: `instream` -> `colorset ms`

```
output_ms (outstream, <colorset> ms);
```

Function from: `(outstream * colorset ms)`-> `unit`

Append time list to one-element multiset (@ operator)

The @ operator takes a one-value multiset and appends a list of time stamps. The length of the list must be equal to the number of occurrences of the value in the multiset. The function can only be used if time has been declared. If the multiset has more than one value, an exception is raised.

```
multiset_specification @
    [integer_list_specification];
```

Function from: `("a ms * int list)` -> `"a tms`

For example:

```
ML> 2`6 @[1,2];
> !!! ((2,6,[1,2]),empty) : int tms
```

Time and step

The current model time and the step number can be inspected by means of the functions:

```
time ();
```

Function from: `unit` -> `integer or real`

```
step ();
```

Function from: `unit` -> `integer`

Time and **step** can be used in initial marking, arc inscription, guard, code and time regions, with the following exceptions: **Time** cannot be used in input arc inscriptions, guards and code guards, while **step** may only be used for reporting purposes in code segment action clauses.

Simulation options

with_time and **with_code** test whether or not the simulation options *Simulation with time* and *simulation with code* are currently in force.

```
with_time ();
```

Function from: unit -> bool

```
with_code ();
```

Function from: unit -> bool

These functions can be used in initial marking, output arc inscription, and code segments. They may not be used in input arc inscriptions and guards.

Simulation management

write_report manipulates the simulation reports given by CPN. It is possible to turn off the simulator's use of the facility and use the facility for other purposes.

```
write_report (string);
```

Function from: string -> unit

stop_simulation permits the simulation to be stopped from within a code segment. This provides a way of handling exceptions from within the model.

```
stop_simulation ();
```

Function from: unit -> unit

Boolean Selector

% and **/** apply a *boolean selector* to a value, a multiset, a pair of values or a pair of multisets.

% is an if-then command, while **/** is an if-then-else.

```
[boolean list] % 'a;
```

Operator from: (bool list * 'a) -> 'a ms

```
[boolean list] / 'a ms;
```

Operator from: (bool list * 'a ms) -> 'a ms

Addition, Subtraction, and Scalar Multiplication of Functions

Addition (+ operator)

```
("a -> "b ms) + ("a -> "b ms);
```

```
Operator from: (("a -> "b ms) *
("a -> "b ms)) -> ("a -> "b ms)
```

Subtraction (- operator)

```
("a "b ms) - ("a "b ms);
```

```
Operator from: (("a -> "b ms) *
("a -> "b ms)) -> ("a -> "b ms)
```

Function subtraction applies multiset subtraction.

The first multiset must be greater than or equal to the second. If this is not the case, an exception is raised.

The second multiset must be a subset of the first multiset. If this is not the case, an exception is raised.

Scalar Multiplication (* operator)

For scalar multiplication the integer must be non-negative.

```
(int * ('a 'b ms));
```

```
Operator from: ((int * ('a -> 'b ms)) ->
('a -> 'b ms))
```

Identity, Zero, and Ignore

id, **zero**, and **ign** can be applied to all kinds of arguments, for example: values and multisets. They can even be applied to functions.

Identity

Id maps everything into itself.

```
id ("a);
```

```
Function from: 'a -> 'a
```

Zero

Zero maps everything into the empty multiset, which is polymorphic (that is, the same for all kinds of multisets).

```
zero ("a");
```

Function from: "a -> "b ms

Ignore

Ign maps everything into () — the only element in the primitive ML type unit.

```
ign ("a");
```

Function from: "a -> unit

Graphical Functions

You can use ML functions to access most of the functions in the graphic library of Design/OA. Users of Design/OA should note that the functions have the same names as their C counterparts.

Using these functions you can write animation code for code segments of transitions. The functions should only be used to manipulate auxiliary objects.

ColorIO Functions

These functions allow code segments to read and write colorset and multiset definitions.

```
input_col (instream);
```

Function from: instream -> <colorset>

```
output_col (instream, colorset);
```

Function from: (outstream * <colorset>) -> unit

```
input_ms (instream);
```

Function from: instream -> <colorset> ms

```
output_ms (outstream, colorset ms);
```

Function from: (outstream * <colorset> ms) -> unit

```
input_tms (instream);
```


Function from: instream ->
 (<colorset> ms * TIME list)

```
output_tms (outstream, colorset tms);
```

Function from: (outstream * <colorset> tms) -> unit

NOTE: The timed multiset functions (`input_tms` and `output_tms`) are implicitly defined when the colorset is timed. They can not be explicitly declared.

ML Library

A number of declarations can be grouped in an ML structure. They can then be referred to by prefixing the name with `StructureName` followed by period. You can open the structure by writing `open StructureName;` and then referring to the items simply by means of their names.

When a text file contains a number of ML structures, the corresponding items are made accessible by writing

```
use "FileName"; open Struct1;.....; open StructN;
```

in the global or temporary declaration node.

The file `MLlib` contains a library of ML structures with additional constants, operations, and functions. These structures can be applied if you include the “use” command, described above, in a global or temporary declaration node. Please consult the file to see the items available.

Standard Constants, Operations, and Functions

The following constants, operations, and functions are part of the standard image called `cpn.ML`. Thus they may be used in inscriptions and code segments.

Since these are standard SML constructs, they are not defined in detail; the description simply identifies whether a particular construct is a constant, operator, or function and gives the SML datatype for the domain and range. In the case of constants it is the SML datatype of the constant. Minor differences for the two interpreters are specified as well.

General Functions

Fun	o	(('a -> 'b) * ('d -> 'a)) -> ('d -> 'b)
Fun	use	string -> unit
Fun	usestring	(string list) -> unit
Fun	use_stream	instream -> unit (New Jersey)
Fun	open_string	string -> instream (New Jersey)

Boolean Functions

datatype **bool**

con	false	bool
con	true	bool
Fun	=	"a * "a -> bool
Fun	<>	"a * "a -> bool
Fun	makestring	bool -> string
Fun	not	bool -> bool

Integer Functions

datatype **int**

Fun	*	(int * int) -> int
Fun	+	(int * int) -> int
Fun	-	(int * int) -> int
Fun	<	(int * int) -> bool
Fun	<=	(int * int) -> bool
Fun	>	(int * int) -> bool
Fun	>=	(int * int) -> bool
Fun	abs	int -> int
Fun	div	(int * int) -> int
Fun	makestring	int -> string
Fun	max	(int * int) -> int
Fun	min	(int * int) -> int
Fun	mod	(int * int) -> int
Fun	~	int -> int

Real Functions

datatype **real**

Fun	*	(real * real) -> real
Fun	+	(real * real) -> real
Fun	-	(real * real) -> real
Fun	/	(real * real) -> real
Fun	<	(real * real) -> bool
Fun	<=	(real * real) -> bool
Fun	>	(real * real) -> bool
Fun	>=	(real * real) -> bool

Fun	abs	real -> real
Fun	arctan	real -> real
Fun	cos	real -> real
Fun	exp	real -> real
Fun	floor	real -> int
Fun	ln	real -> real
Fun	makestring	real -> string
Fun	min	(real * real) -> real
Fun	max	(real * real) -> real
Fun	real	int -> real
Fun	sin	real -> real
Fun	sqrt	real -> real
Fun	~	real -> real

String Functions

datatype **string**

Fun	<	(string * string) -> bool
Fun	<=	(string * string) -> bool
Fun	>	(string * string) -> bool
Fun	>=	(string * string) -> bool
Fun	^	(string * string) -> string
Fun	chr	int -> string
Fun	explode	string -> (string list)
Fun	implode	(string list) -> string
Fun	length	string -> int (Edinburgh)
Fun	ord	string -> int
Fun	ordof	(string * int) -> int
Fun	size	string -> int
Fun	substring	(string * int * int) -> string

List Functions

datatype 'a **list**

con	::	('a * ('a list)) -> ('a list)
con	nil	'a list
Fun	^^	((('a list) * ('a list)) -> ('a list))
Fun	app	('a -> 'b) -> (('a list) -> unit)
Fun	exists	((('a -> bool) * ('a list)) -> bool)
Fun	fold	((('a * 'b) -> 'b) -> (('a list) -> ('b -> 'b)))
Fun	hd	('a list) -> 'a
Fun	length	('a list) -> int
Fun	map	('a -> 'b) -> (('a list) -> ('b list))
Fun	nth	((('a list) * int) -> 'a)
Fun	null	('a list) -> bool
Fun	rev	('a list) -> ('a list)
Fun	revapp	('a -> unit) -> (('a list) -> unit) (Edinburgh)
Fun	revapp	('a -> 'b) -> (('a list) -> unit) (New Jersey)
Fun	revfold	((('a * 'b) -> 'b) -> (('a list) -> ('b -> 'b)))

Fun **tl** ('a list) -> ('a list)

CPN ML uses ^^ as the operator symbol for list concatenation (instead of the standard @). If you are not using the time clause, you may reinstall the Standard name by typing "val @=^^;"

Reference Functions

datatype '**_a ref**

con **ref** ('_a -> ('_a ref)
Fun **!** ('a ref) -> 'a
Fun **:=** (('a ref) * 'a) -> unit
Fun **dec** (int ref) -> unit
Fun **inc** (int ref) -> unit

I/O Functions

datatype **instream**
datatype **outstream**

exception **io_failure** string

Fun **close_in** instream -> unit
Fun **close_out** outstream -> unit
Fun **end_of_stream** instream -> bool
Fun **file_exists** string -> bool
Fun **input** (instream * int) -> string
Fun **input_line** instream -> string
Fun **lookahead** instream -> string
Fun **open_in** string -> instream
Fun **open_out** string -> outstream
Fun **open_append** string -> outstream
Fun **output** (outstream * string) -> unit
Fun **print** 'a -> 'a (* Writes to std_out *)
Fun **std_in** instream
Fun **std_out** outstream
Fun **log** outstream

Never call `close_in` with `std_in`. It will cause the system to malfunction.

Real Time Functions

The following function makes it possible to test with respect to the time shown by the operating system clock. The function returns the number of seconds elapsed since 00.00 GMT January 1, 1970.

Fun **tod** unit -> int

APPENDIXES

Appendix A

Keys and Shortcuts

Design/CPN provides a range of “power commands” for quick and efficient performance. The modifier keys are used with a command, during an operation invoked by a command, or during direct manipulation with the mouse.

Design/CPN Use of the Alt Key

Design/CPN documentation specifies the use of the ALT key in various keystroke shortcuts. Some keyboards use a DIAMOND () key instead; occasionally other keys are used. If ALT does not produce the described results, check your terminal configuration.

Keystroke Shortcuts

You can invoke some of the most commonly applied commands by using *keystroke shortcuts* instead of the items in the menus. Some keystroke shortcuts have different meanings in different contexts. The following table lists all shortcuts.

Alt+A	Select All Nodes
Alt+B	Transition, Bind
Alt+C	Copy
Alt+D	Stop
Alt+E	Place, Interactive Run
Alt+F	Find
Alt+G	Enter/Leave Group Mode
Alt+H	Horizontal
Alt+I	Get Info
Alt+J	Vertical
Alt+K	Arc, Continue
Alt+L	Label
Alt+M	Make Region
Alt+N	Find Next

Appendix A

Alt+O	Open
Alt+P	Print
Alt+Q	Quit
Alt+R	CPN Region, Automatic Run
Alt+S	Save
Alt+T	Enter/Leave Text Mode
Alt+U	General Simulation Options
Alt+V	Paste
Alt+W	Select All Text
Alt+X	Cut
Alt+Y	Open Page
Alt+Z	Undo/Redo
Alt+1	Select
Alt+2	Drag
Alt+3	Displace
Alt+4	Adjust
Alt+5	Fit to Text
Alt+6	Text Attributes
Alt+7	Graphic Attributes
Alt+8	Shape Attributes
Alt+9	Region Attributes
Alt+0	Page Attributes
Alt+-	Hide Regions
Alt+=	Show Regions
Alt+[Between
Alt+]	Projection
Alt+;	ML Evaluate
Alt+'	Syntax Check, Reswitch
Alt+,	Cleanup
Alt+.	Terminates modal commands
Alt+/	Duplicate Node
Alt+`	Reduce
Alt+\	Blowup

Modifier keys

You can change the way a command works by pressing certain keys:

Option key

Pressing the **OPTION** key with the relevant command produces the following results:

- Makes port assignment to ports with a non-matching port type.
- Constrains repositioning to horizontal or vertical for **Drag**.
- Constrains resizing with the mouse to horizontal or vertical.
- Invokes the instance menu.
- Modifies all **Align** menu commands in group mode.
- Selects target node for connector drag.

Shift key

Pressing the **SHIFT** key with the relevant command produces the following results:

- Arrow keys scroll or move in text pointer hierarchy .
- Adds and subtracts to/from current selection.
- Repositions during adjust/creation.
- Creates a single binding for the selected transition.
- Invokes the page instance dialog box.
- Modifies all **Align** menu commands in group mode.

Option + Shift keys

Pressing the **OPTION + SHIFT** keys:

- Creates a single binding for the selected transition and starts an execution.

Appendix A

Caps Lock key

Maintains object proportions.

Space bar

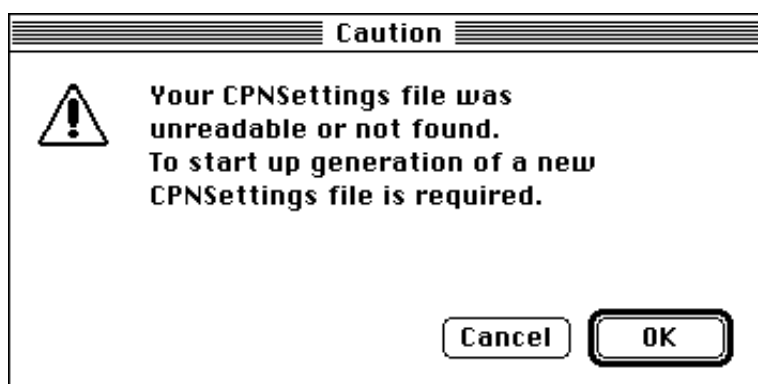
Pressing the space bar:

- Removes a point from object when the graphic tool is placed on it.
- Hides an object during selection.

Appendix B

Troubleshooting

When you try to start Design/CPN, and the CPN Settings file is missing or obsolete, Design/CPN displays:



This appendix describes various problems that you may encounter when you attempt to run Design/CPN, and tells you how to solve them. All of the problems relate in some way to the interface between Design/CPN and the computer on which it runs. The problems described are:

- CPN Settings file is missing or obsolete.
- ML configuration is unspecified or incorrect.
- ML Interpreter cannot be started.

When one of these problems occurs, Design/CPN displays a descriptive dialog box. These boxes, the problems they indicate, and the solutions to those problems, are described in this appendix.

CPN Settings File Missing or Obsolete

When you try to start Design/CPN, and the .CPNSettings file is missing or obsolete, Design/CPN displays a dialog that states:

Your .CPNSettings file was unreadable or not found. To start up, generation of a new .CPNSettings file is required.

The choices offered are **OK** and **Cancel**.

- Click **Cancel**.

Design/CPN quits.

Problem Description

When Design/CPN is installed, a directory called Design/CPN is created. This directory contains a file called .CPNSettings. In order for Design/CPN to run, this file must be copied to your home directory. If the file was not copied there, or was subsequently renamed or removed, Design/CPN cannot find the settings it needs in order to run correctly. It therefore displays the above dialog.

Problem Solution

If you have a copy of .CPNSettings in your Design/CPN directory:

- Copy .CPNSettings to your home directory.

If you do not have a copy of .CPNSettings in your Design/CPN directory:

- Reinstall Design/CPN from the source tape.
- Copy .CPNSettings to your home directory.

After you have copied the settings file:

- Start Design/CPN.

The application should now start without problems relating to CPN settings.

ML Configuration Unspecified or Incorrect.

Before you can run the ML interpreter to syntax check or execute a diagram, you must use the **ML Configuration Options** command (**Set** menu) to tell Design/CPN:

- **Hostname:** Where the ML interpreter is to be run.

- **Port number:** Which port is used by the daemon is to run the interpreter.
- **ML image:** Where the interpreter is located in the file system.

Design/CPN keeps this information as system defaults. In order for a particular diagram to use it, the information must be present in its diagram defaults.

Identifying the Problem

If any ML configuration information is missing or incorrect, Design/CPN will be unable to start the ML interpreter, and will display a dialog box. The dialog displayed depends on which information is faulty. If more than one of the parameters is faulty, the dialog that appears will describe the first erroneous parameter encountered.

If the **Hostname** is missing or wrong, the dialog offers an opportunity to log in, with **OK** and **Cancel** buttons. Attempting to log in will fail, so:

- Click **Cancel**.

If the **Port number** is missing or wrong, the dialog states:

Network not responding. Connection with host could not be established.

The dialog contains an **OK** button.

- Click **OK**.

If the ML image is missing or wrong, the dialog states:

ML could not be started. Check available memory and presence of default ML file.

The dialog contains an **OK** button.

- Click **OK**.

You must now supply the necessary information. You may be able to do this by copying the system defaults, or you may have to supply correct information by typing it into the **ML Configuration Options** dialog.

Copying Diagram Default ML Configuration Options

When the system defaults are correct, but the diagram defaults are not, the system defaults must be copied to the diagram defaults. This is typically necessary when the diagram was created on some other computer. It is also necessary if the system defaults have changed since the diagram was created. To update the diagram defaults:

- Choose **ML Configuration Options** from the **Set** menu.

The **ML Configuration Options** dialog appears.

- Click **Load**.
- Click **OK**.

The necessary information about the ML process is copied from the system defaults to the diagram defaults. Design/CPN can now start the ML process and use it to syntax check and execute the diagram.

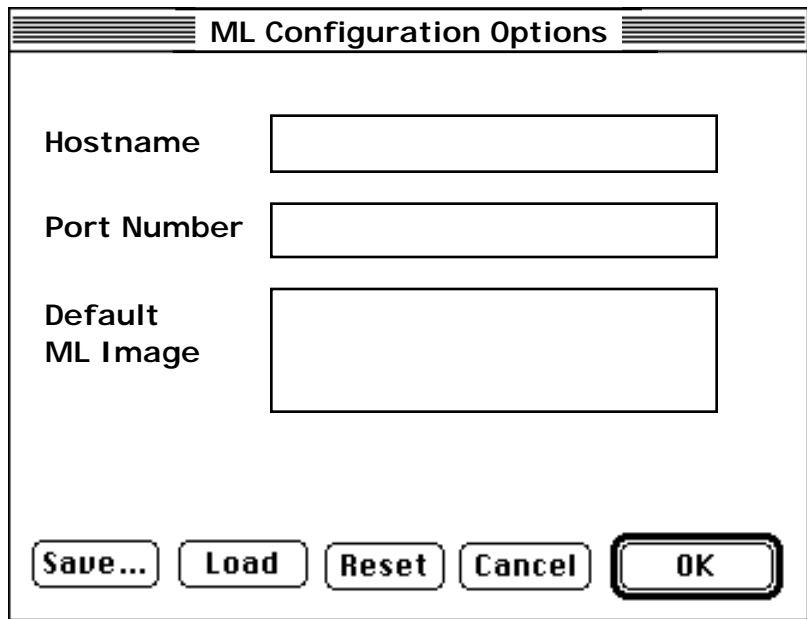
When the ML process fails to start, loading the system defaults is the thing to try first. If Design/CPN is correctly installed on your system, it will always work, and it can never do any harm since the diagram defaults were wrong in any case. If it does not work, the system default ML configuration options must be established or corrected.

Setting ML Configuration Options

To establish new values for any or all ML configuration options:

- Choose **ML Configuration Options** from the **Set** menu.

The **ML Configuration Options** dialog appears:



The image shows a dialog box titled "ML Configuration Options". It contains three input fields: "Hostname", "Port Number", and "Default ML Image". At the bottom, there are five buttons: "Save...", "Load", "Reset", "Cancel", and "OK".

- Enter correct values as needed for the **Hostname**, **Port number**, and **ML image** fields.

See the Design/CPN Installation Notes for information on determining these values.

If you want the new values to become system defaults:

- Click **Save**.

A confirmation dialog appears.

- Click **OK** in the confirmation dialog.

The values in the **ML Configuration Options** dialog are now the system defaults.

To make the new values the diagram defaults:

- Click **OK** in the **ML Configuration Options** dialog .

The values in the **ML Configuration Options** dialog are now the diagram defaults. The dialog closes. If the values are correct, you will now be able to use the ML interpreter to syntax check and execute the diagram.

ML Interpreter Cannot Be Started

In order to run, the ML interpreter needs:

- A least 32 meg of RAM.
- At least twice as much swap space as RAM.

Larger allocations will result in better performance.

If the ML interpreter does not have enough RAM or swap space to run in, attempting to start it will display a dialog that mentions a lack of available memory. There are two possible solutions:

- Terminate other processes that are running in your memory space, freeing their memory allocations for use by the ML interpreter.

If there are no such processes, or terminating them does not free enough memory:

- Ask your system administrator to increase your memory allocation.

