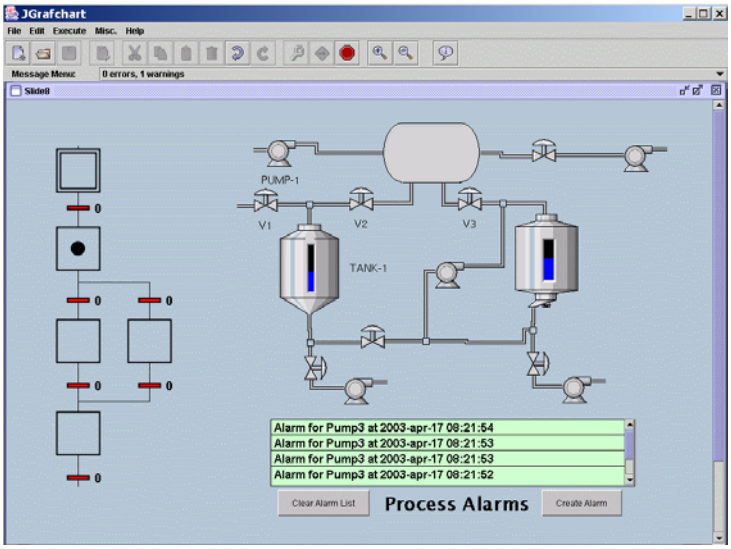
JGrafchart User Manual
Version 1.5

2004-02-13



Karl-Erik Årzén
Department of Automatic Control
Lund University

Contents

1 INTRODUCTION 6

1.1 STEPS AND TRANSITIONS 6
1.2 PARALLEL AND ALTERNATIVE PATHS 7
1.3 MACRO STEPS 7
1.4 PROCEDURES 8
1.5 TYPES AND VARIABLES 9
1.6 JGRAFCHART ACTION LANGUAGE 10
1.7 FUNCTION CHART CREATION 11
1.8 EXECUTION MODEL AND SCOPING RULES 11

2 JGRAFCHART EDITOR 13

2.1 OVERVIEW 13
2.2 WORKSPACES 15
2.3 MOUSE INTERACTION 16
2.4 FILE MENU 18
2.5 EDIT MENU 21
2.6 EXECUTE MENU 22
2.7 MISC MENU 23
2.8 HELP MENU 24
2.9 COMMAND LINE ARGUMENTS 24

3 LANGUAGE REFERENCE 26

3.1 GRAPHICAL FUNCTION CHART ELEMENTS 26
3.1.1 Step 26
3.1.2 Initial Step 27
3.1.3 Transition 28
3.1.4 Parallel Split 29
3.1.5 Parallel Join 30
3.1.6 Macro Step 30
3.1.7 Enter Step 32
3.1.8 Exit Step 33
3.1.9 Exception Transition 34
3.1.10 Procedure 35
3.1.11 Procedure Step 39
3.1.12 Process Step 41
3.1.13 Workspace Object 42
3.1.14 Step Fusion Set 45
3.1.15 Connection Post 46
3.2 VARIABLES AND LISTS 47
3.2.1 Boolean Variable 47
3.2.2 Integer Variable 49
3.2.3 Real Variable 51
3.2.4 String Variable 53
3.2.5 Boolean List 55
3.2.6 Integer List 57
3.2.7 Real List 59
3.2.8 String List 61
3.3 INPUTS AND OUTPUTS 63
3.3.1 Digital Input 63
3.3.2 Digital Input (Inverse Logic) 64
3.3.3 Digital Output 65
3.3.4 Digital Output (Inverse Logic) 66
3.3.5 Analog Input 67
3.3.6 Analog Output 68
3.3.7 Socket IO 69
3.3.7.1 Boolean Socket Input 70

3.3.7.2 Integer Socket Input ... 71
3.3.7.3 Real Socket Input ... 72
3.3.7.4 String Socket Input ... 73
3.3.7.5 Boolean Socket Output ... 74
3.3.7.6 Integer Socket Output ... 75
3.3.7.7 Real Socket Output ... 76
3.3.7.8 String Socket Output ... 77
3.4 CHEM AND CCOM ... 78
3.4.1 Representing XML structures ... 78
3.4.2 Connecting to CCOM ... 81
3.4.3 Publishing Data Messages ... 82
3.4.4 Sending Data Messages ... 82
3.4.5 Subscriptions ... 82
3.4.6 Receiving Data Messages ... 82
3.4.7 Detecting Message Arrival ... 83
3.4.8 Sending Reply Messages ... 84
3.4.9 Receiving Message Replies ... 85
3.4.10 Remote Procedure Calls ... 85
3.4.11 External Calls to JGrafchart Procedures ... 86
3.4.12 Calling External RPCs ... 86
3.4.13 CCOM RPC Libraries ... 87
3.5 CCOM XML OBJECT REFERENCE ... 88
3.5.1 XML Message In ... 88
3.5.2 XML Message Out ... 92
3.5.3 Integer Attribute ... 96
3.5.4 Real Attribute ... 97
3.5.5 String Attribute ... 98
3.5.6 Index Variable ... 99
3.6 JGRAFCHART TEXTUAL LANGUAGE ... 100
3.6.1 Action Language ... 100
3.6.2 Condition Language ... 101
3.6.3 Functions ... 102
3.6.3.1 Mathematical functions ... 102
3.6.3.2 Text and time functions ... 102
3.6.3.3 Colour functions ... 103
3.6.3.4 Debug functions ... 103
3.6.3.5 Dynamic Loading and Creation Functions ... 103
3.6.3.6 Misc. functions ... 104
3.6.3.7 CCOM functions ... 104
3.6.4 Evaluation Rules ... 105
3.7 JGRAFCHART EXECUTION MODEL ... 106
3.7.1 Compilation ... 106
3.7.2 Execution ... 106

4 GRAPHICAL USER INTERFACE ... 108

4.1 GRAPHICAL USER INTERFACE ELEMENTS ... 108
4.1.1 Free Text ... 108
4.1.2 Rectangle ... 112
4.1.3 Ellipse ... 115
4.1.4 Polygon ... 117
4.1.5 3D Rectangle ... 119
4.1.6 Line ... 120
4.1.7 Spline ... 123
4.1.8 Icon ... 125
4.1.9 Action Button ... 126
4.1.10 Graphical Action Button ... 127
4.1.11 Plotter ... 128
4.1.12 Browser ... 130
4.1.13 Show Workspace Button ... 133
4.1.14 CHEM Icon ... 134
4.1.15 LUCAS Icon ... 135
4.1.16 Group ... 136

5 CUSTOM IO 137
5.1 INTERFACE LOCALIO 137
5.2 INTERFACE ANALOGINPUT 137
5.3 INTERFACE ANALOGOUTPUT 137
5.4 INTERFACE DIGITALINPUT 137
5.5 INTERFACE DIGITALOUTPUT 138
5.6 EXAMPLE: LUNDPRINTIO 138

1 Introduction

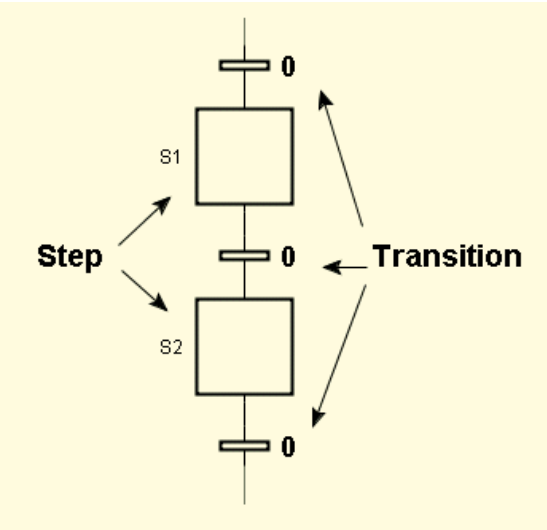
JGrafchart provides graphical representation and execution of operation sequences, procedures, and state machines. It integrates features from Grafcet/Sequential Function Charts (SFC) in IEC61131-3, and Statecharts with concepts from procedural and object-oriented programming. JGrafchart is implemented in Java/Swing and runs on all platforms that support Java. The tool combines an interactive graphical programming environment with a run-time engine for execution of JGrafchart function charts. The tool also makes it possible to implement animated graphical user interfaces with message browsers, plotters, bitmap graphics, and animated icons

JGrafchart can be used for all types of discrete-event based applications, e.g., logical control, operating procedure management, recipe-based batch control, and workflow modelling. The high generality of JGrafchart makes it applicable in a wide variety of process control and operator support situations. For example, JGrafchart can be used to coordinate the execution of the different fault detection and diagnosis components and also implement the necessary fault-recovery actions. Another possibility is to use JGrafchart to automate process operating procedure execution, either fully automatically or with the involvement of the human process operator.

JGrafchart comes with an integrated HTML-based on-line help system. It also contains an interactive dynamic tutorial.

1.1 Steps and Transitions

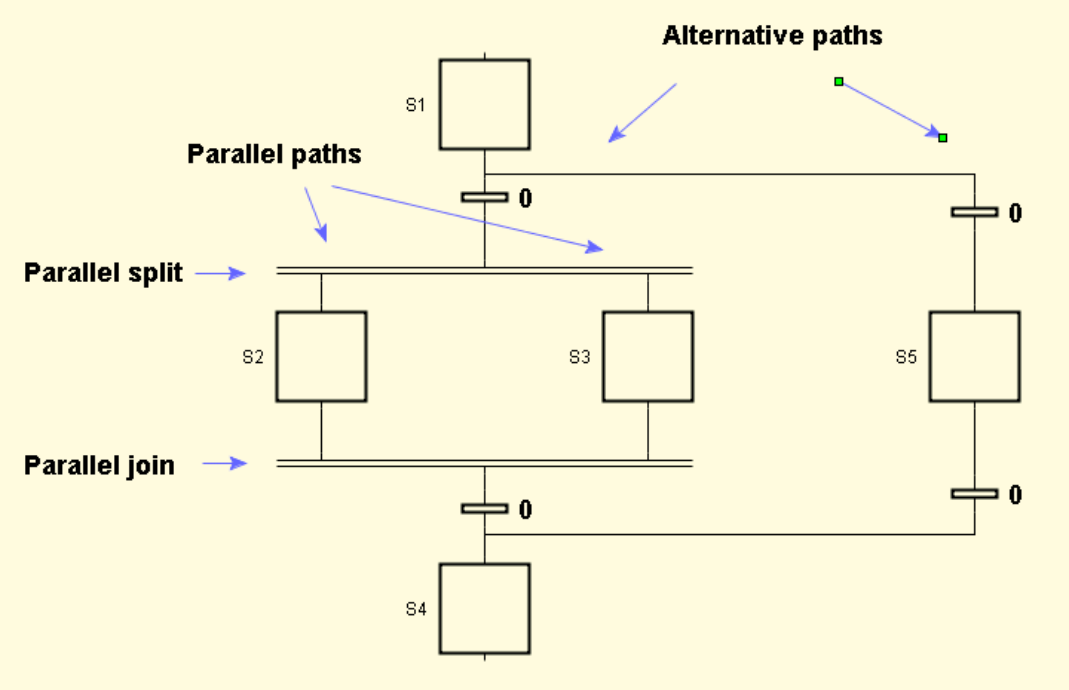
The graphical language of JGrafchart is based on Grafcet/Sequential Function Charts (SFC). The language is state-transition oriented. A JGrafchart program consists of a function chart that represents an activity flow. By convention the flow goes from the top of page to the bottom of page. More formally a Grafcet function chart is a bipartite directed graph consisting of **steps** and **transitions**.



A step can be inactive or active. An active step is indicated by a filled token. **Initial steps** are special steps that are automatically activated when execution starts. Initial steps are shown graphically by a double rectangle. Associated with the steps are actions. The actions are expressed in a textual action language. The actions are performed when the step is activated, deactivated, aborted, or periodically, while the step is active. The transitions have an associated boolean condition that acts as a guard. In order for a transition to be enabled, all the steps immediately preceding the transition must be active. A transition that is enabled fires if its condition is true.

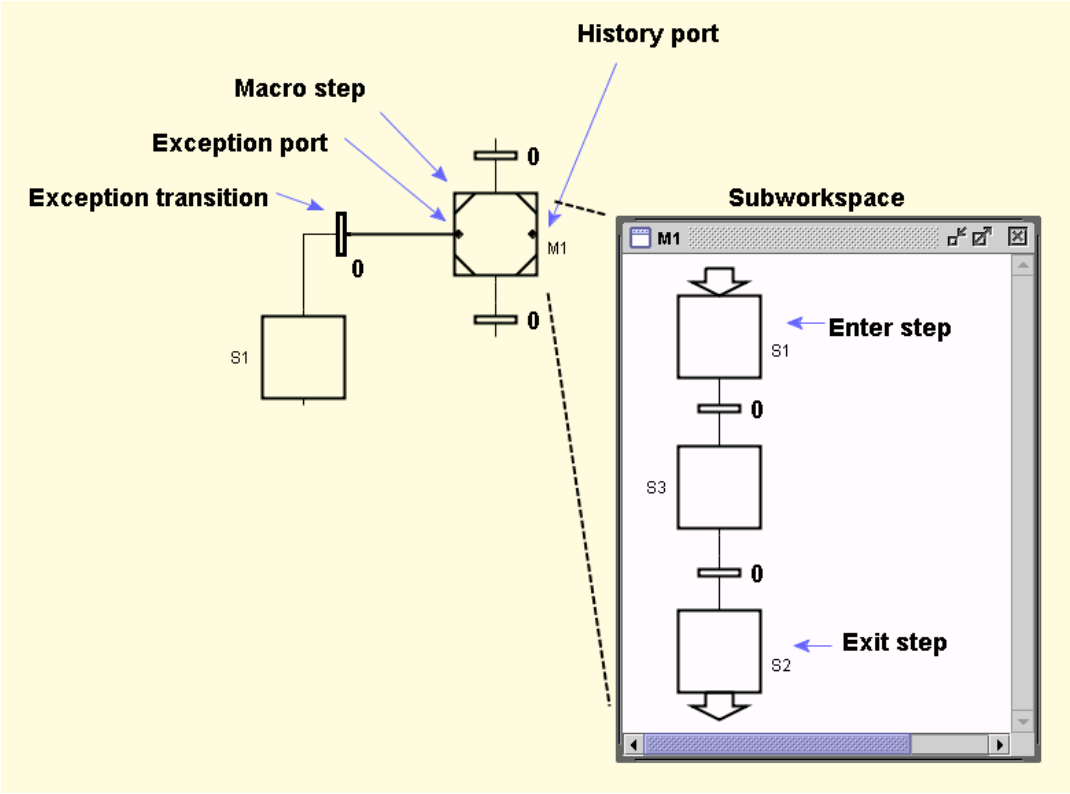
1.2 Parallel and Alternative Paths

In JGrafchart it is possible to express **parallel** and **alternative paths** as shown below.



1.3 Macro Steps

Macro steps are steps that contain an internal function chart. The internal function chart is located on the subworkspace of the macro step. When the macro step is activated the **enter step** of the internal function chart is activated. When the execution in the internal function chart has reached the **exit step**, the transition after the macro step is enabled. An **exception transition** is a special type of transition that only may be connected to a macro step. The exception transition if connected to the **exception port** on the left-hand side of the macro step. An ordinary transition connected to a macro step does not become enabled until the execution of the macro step has reached the exit step. An exception transition, however, is enabled all the time while the macro step is active. If the transition is fired the execution inside the macro step is terminated and the step succeeding the exception transition becomes activated. Exception transitions have priority over ordinary transitions in cases where both are fireable. Macro steps have history. When an exception transition aborts the execution of a macro step, the execution state is saved. The macro step can be resumed in this saved state if a transition connected to the special **history port** of the macro step is fired. The history-input port is located on the right hand side of the macro step.



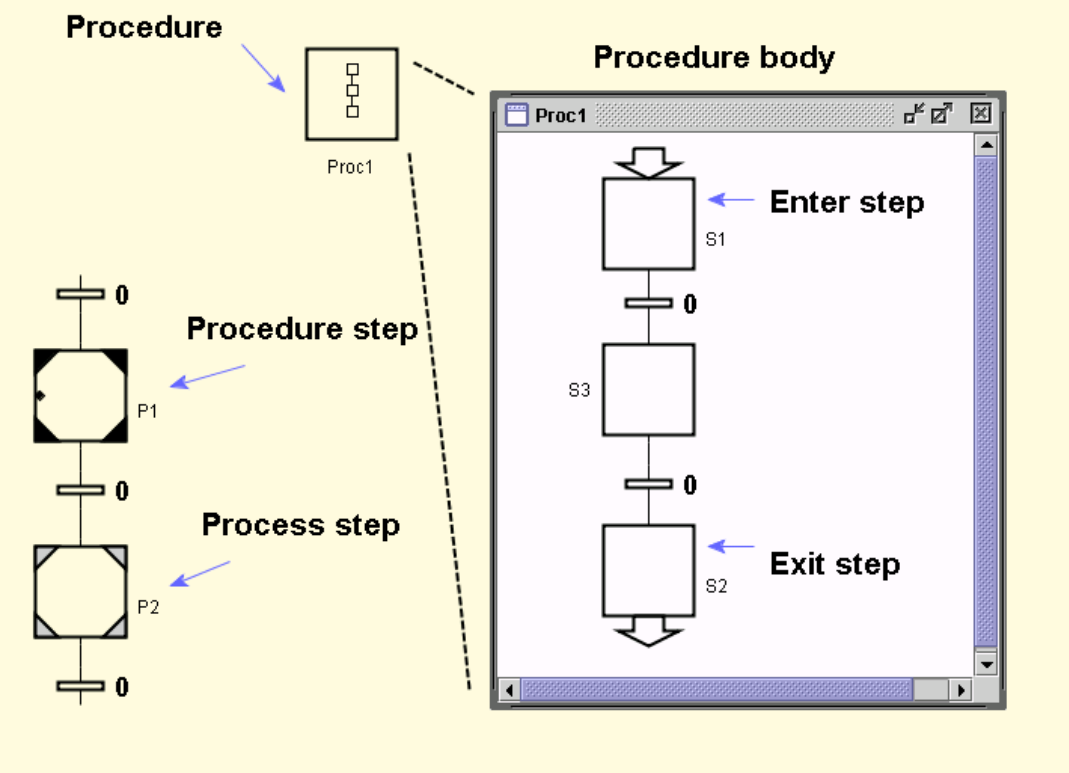
Macro steps may have multiple inputs and multiple outputs. In this case the enter and exit steps are also multiplied. Macro steps with multiple inputs and/or outputs resemble the hierarchical states (superstates) in Statecharts. They are useful for implementing hierarchical state machines.

1.4 Procedures

Step-transition sequences that are used in several different contexts can be represented as **procedures.** A procedure has a procedure body stored on the subworkspace of the procedure. The body begins with an enter step and ends with an exit step in the same way as for macro steps. Procedures are re-entrant and may be recursive. Procedures may have parameters. All internal variables within the procedure can be used as parameters. Parameters can be assigned values both using call-by-value and call-by-reference. A procedure can be called in three different ways:

- Directly from the procedure object.
- Through a **procedure step**.
- Through a **process step**.
- Through an action within a step or a button.

When a procedure is called from a procedure step, the transition(s) after the procedure step does not become enabled until the execution has reached the exit step of the procedure. Hence, this corresponds to an ordinary procedure call. When a procedure is called from a process step the transition(s) after the procedure step is enabled immediately. Conceptually, the procedure call is spawned and executed in a separate execution thread. This is, however, not the way it is implemented. The same thing happens when the procedure is called (started) from an action.

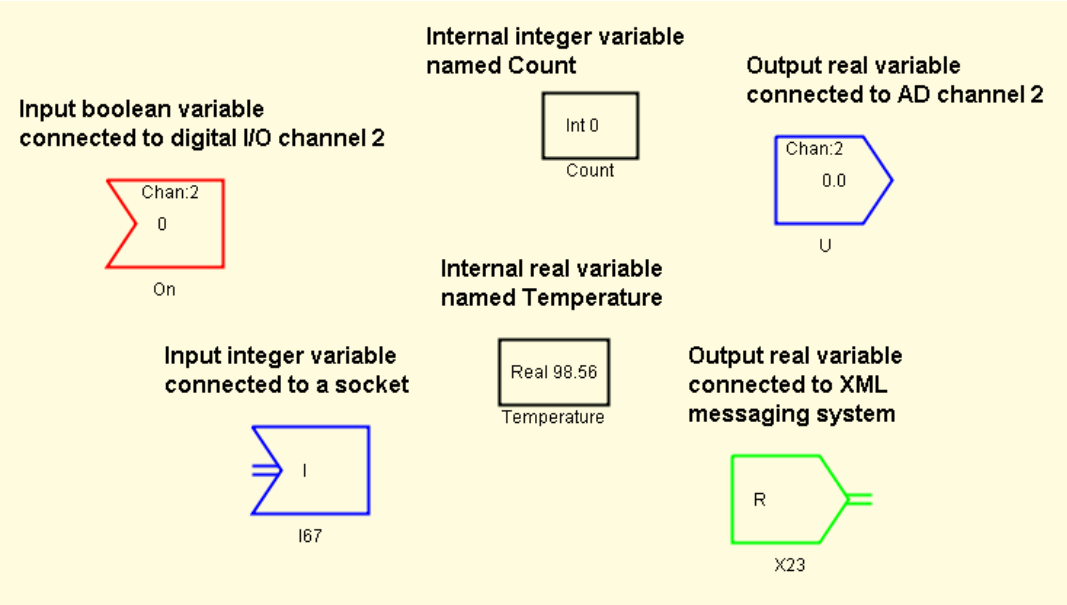


1.5 Types and Variables

JGrafchart supports four basic types: **boolean, integer, real,** and **string**. For each basic type there is a corresponding variable. The variables are either input variables: internal variables, or output variables.

- Input variables receive their values from the external environment. They can be connected to I/O cards, input sockets, etc. Input variables can be read from but not written to from the JGrafchart application.
- Internal variables receive their values from the JGrafchart application. Internal variables can both be read from and written to, unless they have been specified as constant, in which case they only can be read from.
- Output variables receive their values from the JGrafchart application and transmit these values to the external environment. They can be connected to I/O cards, output sockets, etc. Output variables are normally only written to. However, it is possible to also read from them, in which case the last written value is returned.

Some examples of different types of variables are shown below:



For each data type there is also a corresponding list type.

1.6 JGrafchart Action Language

JGrafchart contains both graphical and textual language elements. The textual format is primarily used for expressing step actions and transition conditions. Step actions can be associated with steps (ordinary steps, initial steps, enter steps, exit steps), macro steps, procedure steps, and buttons. A step action consists of two parts: an action qualifier and the actual action. The action qualifier decides when the action should be executed. Five different types are available:

- **Enter action** (Stored action). An enter action is executed once when a step becomes active or when a button is pressed. The syntax for enter actions is

  ```
  S "action";
  ```

- **Exit action.** An exit action is executed once immediately before the step is deactivated, The syntax for exit actions is

  ```
  X "action";
  ```

- **Periodic action.** A periodic action is executed periodically while a step is active. The period is given by the scan-cycle at which the step is executing, The syntax for periodic actions is

  ```
  P "action";
  ```

- **Abort action** An abort action is executed once when a step is aborted due to the firing of an exception transition. The syntax for an abort action is

  ```
  A "action";
  ```

- **Normal action** (Level action). A normal action is used to associate the truth-value of a boolean variable with the activation status of the corresponding step. The boolean variable becomes true when the step become active and becomes false when the step becomes deactivated. If the step becomes deactivated and activated in the same scan-cycle the boolean variable remains true. In order to work properly normal actions may only refer to boolean variables that reside within the same execution context as that of the step or button containing the normal action. For example, the boolean variable may not reside in another top-level workspace. The syntax for a normal action is

  ```
  N "boolean variable";
  ```

The "action" in the syntax definitions above can be of two different types:

- **Assignment**

  In this case the syntax is

  ```
  "variable expression" = "expression"
  ```

  The "variable expression" is either a direct variable reference or an expression that evaluates to a variable reference, The way the "expression" is evaluated is determined by the type of the variable. For example, if the variable is a real variable the expression will be evaluated as a real-valued expression. JGrafchart uses automatic type casting.

- **Method call**
  In this case the syntax is

  ```
  "object expression"."methodname"(arg1,...argn)
  ```

  Method calls are used to call a built-in method of an object. Methods are available for changing size, position, etc of objects, showing and hiding workspaces, enabling and disabling workspaces, doing printouts, etc. In many cases the methods are implemented by calls to underlying Java methods.

The action language is also used to express transition conditions. A transition condition is represented by an expression, which is evaluated as a boolean expression.

1.7 Function Chart Creation

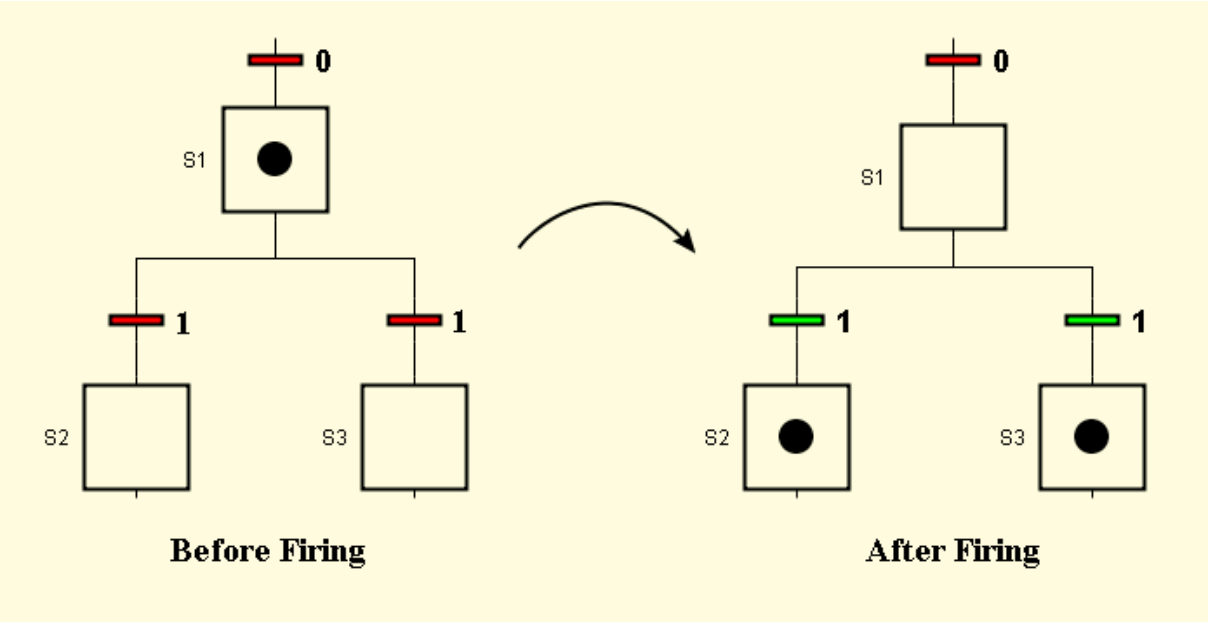
By dragging and dropping language elements from the palette to the left in the editor window into a workspace and connecting them together the developer creates function charts. Clicking and dragging the connection stubs at the ports of the different language elements creates connections. The connection creation is syntax-driven. It is not possible to create any illegal connections, for example connecting a step directly to another step.

1.8 Execution model and scoping rules

JGrafchart uses the traditional PLC execution model. A separate periodic thread executes each top-level workspace. Every period (scan-cycle) the thread performs three operations:

1. Read inputs
2. Execute one scan of the application. Outputs of enter, exit, periodic, and abort actions are written.
3. Write outputs of normal actions

The execution of the application is performed in two steps. In the first step every transition is evaluated, If the preceding steps of the transition are active and the transition condition is true then the transition is fired. Firing the transition involves setting the activation status of the preceding steps for the next scan to false and setting the activation status of the succeeding steps for the next scan to be true. In the second step the activation status of the step is updated. This evaluation strategy eliminates the risk for chains of transition firings within the same scan-cycle. The strategy also effects how alternative transitions with non-mutually exclusive conditions are handled. In JGrafchart all of these alternative transitions will be fired, as shown below. This conforms to the Grafcet standard. It is also gives a fully deterministic behavior. It is good programming practice to make sure that alternative transitions always are mutually exclusive.



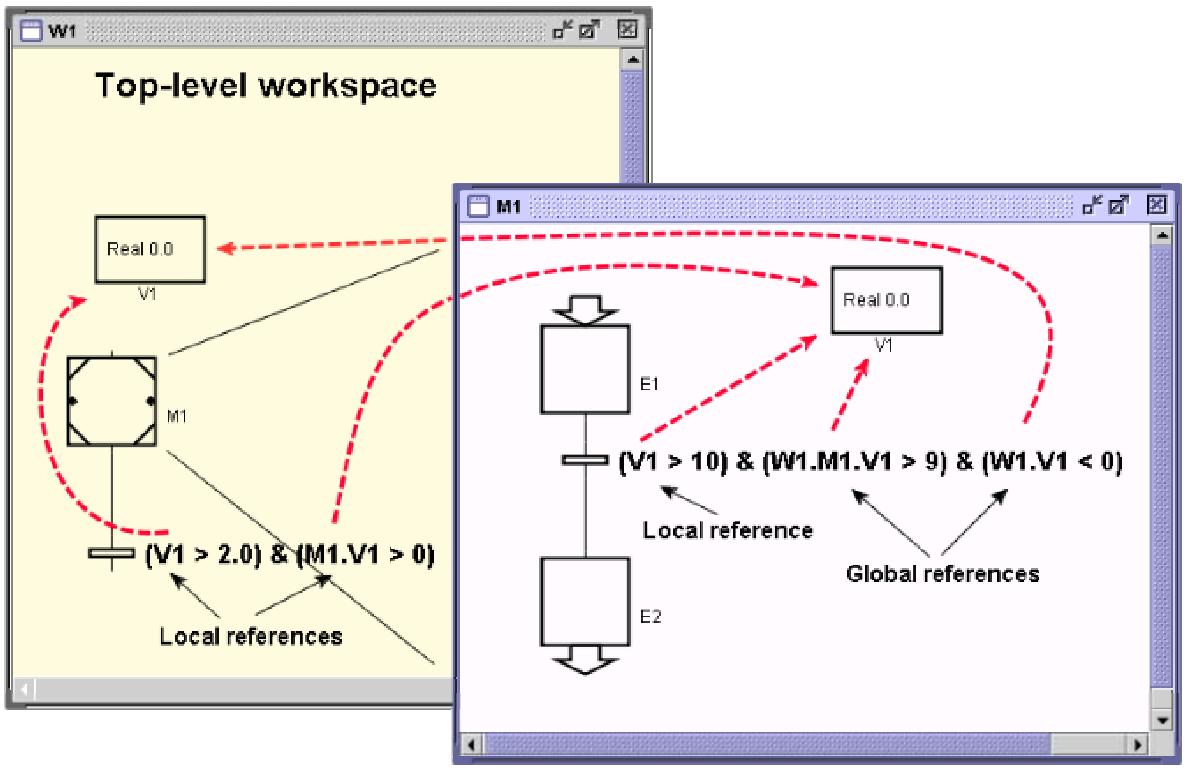
Before a function chart can be executed it must be compiled. During compilation the following operations are performed:

- For every transition, references are set up between the transition and its preceding steps and between the transition and its succeeding steps.
- For every step, the step actions are parsed and an abstract syntax tree is created.
- For every transition, the transition condition is parsed and an abstract syntax tree is created.
- The abstract syntax trees are traversed and a symbol lookup is performed where references are created between the variable nodes and the objects, e.g., variables that they refer to,

JGrafchart uses lexical name scoping. A variable or object reference can be either **global** or **local**. A global reference starts with a top-level workspace name and then uses dot-notation to access lower lexical layers. For example, the reference

W1.M1.V1

could refer to a variable named V1 contained in the macro step M1 that in turn is found in the top-level workspace W1. A local reference is local to the lexical context in which the reference is declared. For example, a reference to V1 from a step action of a step located at the same workspace as V1 is simply written as "V1". During symbol lookup the search begins at the lexical context of the expression. If a matching object is found, the lookup is completed, If not, the search continues at the next higher lexical level, in this example the top-level workspace W1. The effect of this is that names on a lower lexical level hide the same names on higher lexical levels, in the same way as in ordinary lexically scoped programming languages. It is also possible to use dot-notation to reference the internal parts of object groups.



The symbol lookup is currently not always type-sensitive. For example, when looking up the reference for a variable node named "S1", a reference is created to the *first* object with the name "S1", irrespectively of if the type of the object matches the context in which the variable node occurs. For example, assume that "S1" is the name of a step. The transition condition "S1 > 100" would not generate any compilation errors. This behavior is something that is subject to change in future releases.

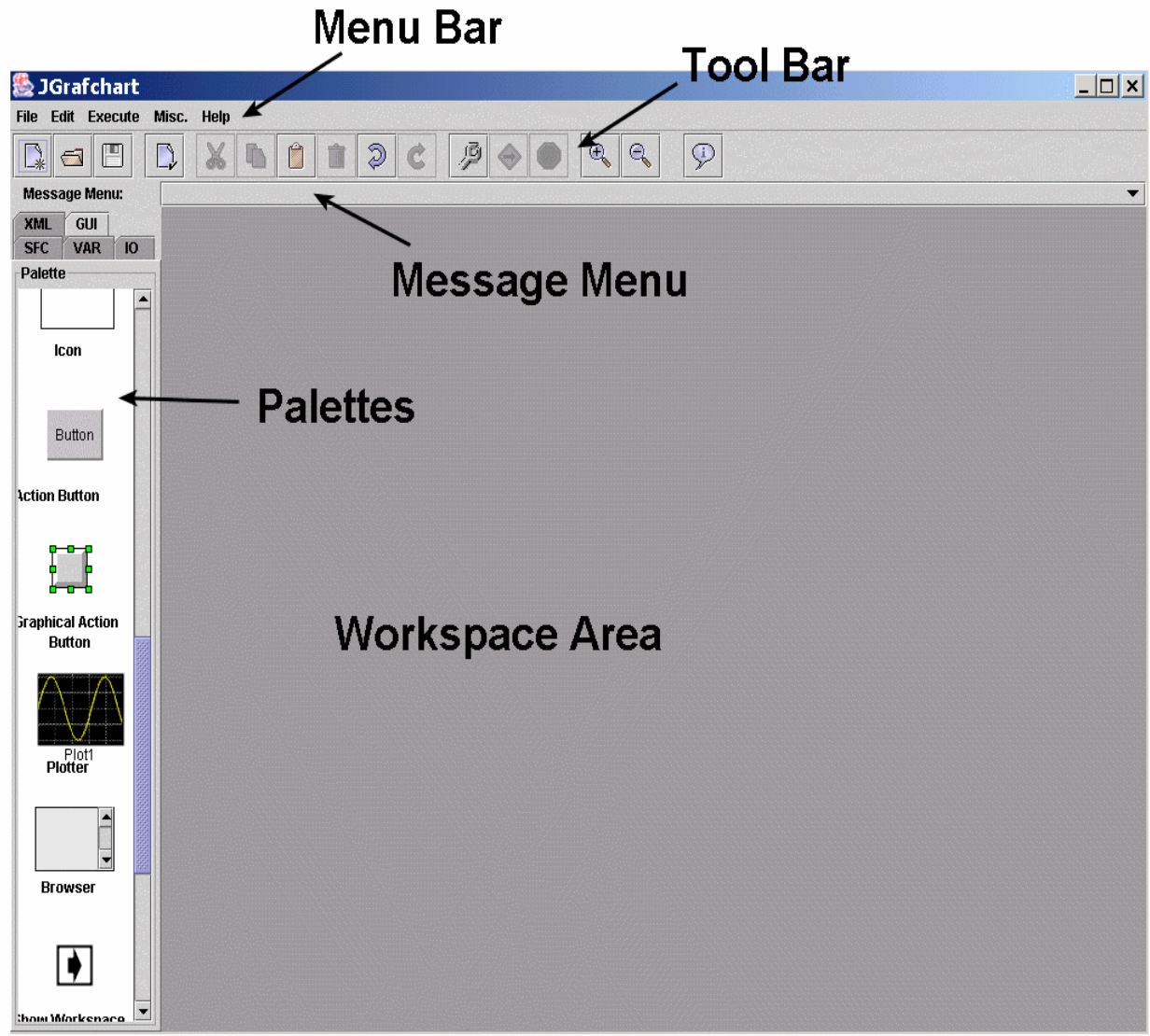
Compilation errors and warnings are shown in the pull-down menu in the toolbar at the top of the editor window, If the compilation was successful the function chart can be executed. During execution it is not possible to drag and drop elements from the palette or to add elements from or to the clipboard. It is also impossible to delete language elements.

2 JGrafchart Editor

2.1 Overview

The JGrafchart editor consists of a graphical user-interface through which the user creates, compiles, executes, and stores function charts. The basic layout of the editor is shown below.

1



The editor consists of the following main parts:

- **Menu bar**

The menu bar contains pull-down menus containing menu choices by which the user controls the editor.

- **Tool bar**

The toolbar contains tool buttons that provide shortcuts to some of the menu choices available from the menus in the menu bar. The toolbar also contains a message menu. The message menu contains compilation error and warning messages. It is also possible for an application to write messages to the message menu and to clear the message menu. The message menu is implemented as a pull-down menu. It works as a stack where new messages are pushed on top of the stack and shown at the top of the menu. The toolbar can be removed and added using menu choices,

- **Palette**

The palette is a five-tabbed pane containing the different language elements in JGrafchart. The user creates an application by drag-and-drop from the palette into a workspace. The five tabs are named: SFC (the default palette), VAR, IO, XML, and GUI. The SFC palette contains the basic JGrafchart language elements, e.g, steps, transitions, procedures, etc. The VAR palette contains JGrafchart variables and lists. The IO menu contains input and output objects. The XML palette contains objects for communication using XML. The objects in this menu assume the availability of the CCOM

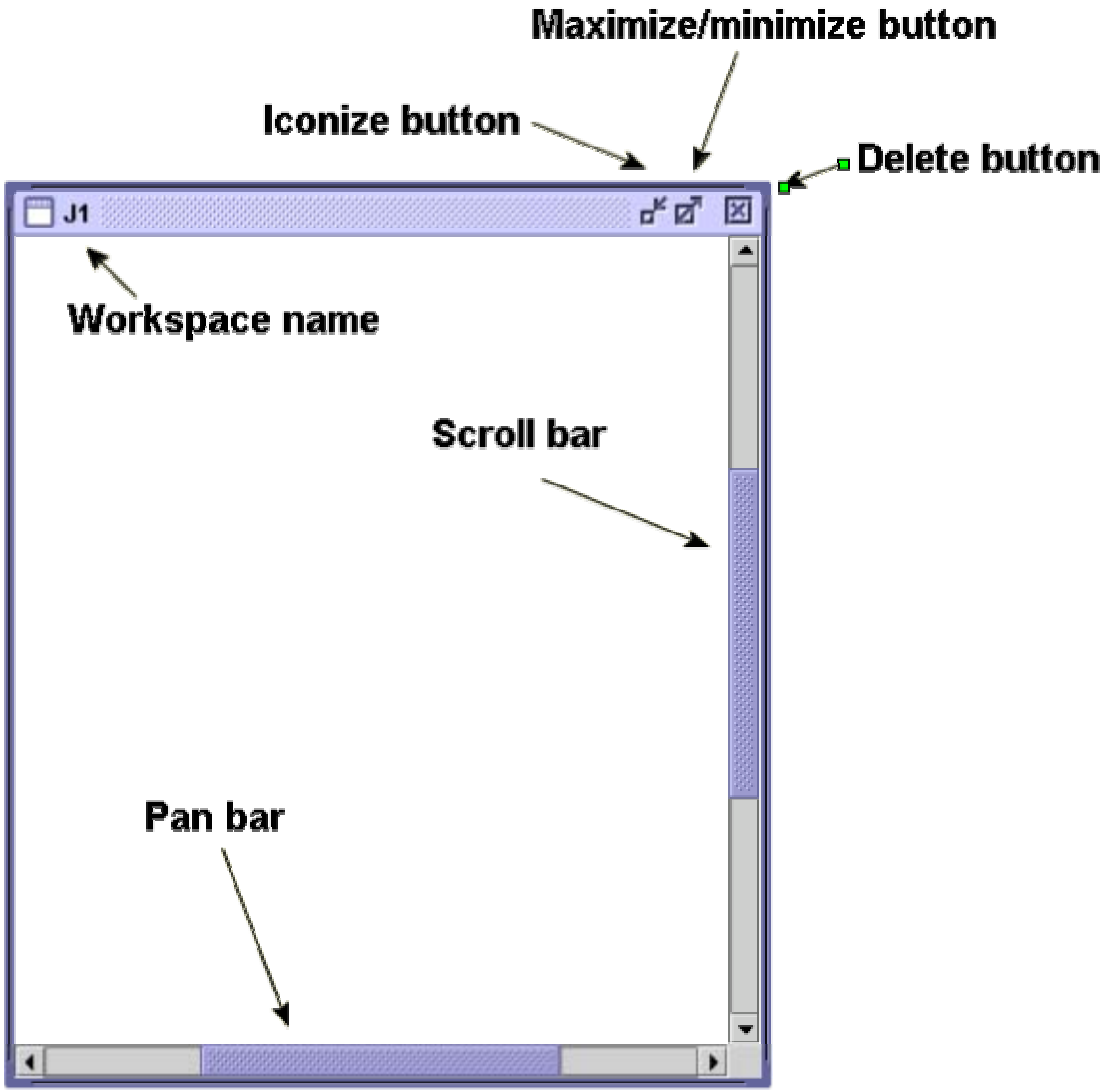
infrastructure. This is not available in all distributions. The GUI palette contains graphical objects, e.g., texts, rectangles, icons, etc. It also contains plotters, browsers, and buttons. In addition to the drag-and-droppable objects the GUI palette also contains two mode buttons: the line-mode button and the spline-mode button. By clicking on these buttons the mouse-behavior changes from the standard select-object mode to a line-drawing mode. The palette can be removed and added.

- **Workspace area**

The workspace area is the area where the user application workspaces are shown. The workspaces are implemented as Swing internal frames. They can be maximized/minimized, iconized, deleted, scrolled, and panned using standard window operations.

2.2 Workspaces

Top-level workspaces and sub-workspaces are implemented as internal frames (windows). The windows can be moved within the workspace area of the editor. The windows have a border, a scroll bar and a pan bar. The border at the top of the window contains the name of the workspace, an iconize button, a maximize/minimize button, and, for non-top level workspaces, a delete button. Top-level workspaces can only be deleted using the **Delete top-level workspace** menu option in the **Edit menu**. Deletion of top-level workspaces is not undoable. For top-level workspaces, the workspace name is preceded by the text string "**Top:** ".



Top-level workspaces are given the default names J# where # is an integer number. The first top-level workspace created gets the name J1, the second one J2, etc. The name of a subworkspace is the same as the name of the language element that contains the subworkspace. The workspace which has the current focus is indicated by a highlighted border. The focus is changed by clicking on the workspace. Workspaces have an associated level number. The level of a top-level workspace is 1. The level of the immediate subworkspaces of this top-level workspace is 2 etc. Workspaces with a high level number are always shown on top of workspaces with a lower level number. When a workspace receives focus it is shown on top of all workspaces with the same or lower level number.

A workspace has an associated coordinate system. The origin of the coordinate system is the top-left corner of the workspace. Both positive and negative coordinates are supported. The window shows a view of the document associated with the window. By moving an object slowly towards the left/right or top/bottom border of the window the view is moved (equivalent to panning and scrolling). The horizontal and vertical scrollbars may be removed.

As a default each workspace has a rectangular grid where the width and the height are ten pixels. Objets that are created or moved will automatically snap to the grid. The default grid snap point of an object that contains a connection stub (steps, transitions, etc) is the top center of the main rectangle of the object. The default grid snap point of an object without any connection stubs is the top left corner of the object.

A top-level workspace can be locked. In a locked workspace it is not possible to move objects, delete objects, create objects, make connections etc. However, it is still possible to access the object-specific menus and it is also possible to execute the workspace. A locked workspace cannot be resized or maximized. A locked workspace is indicated by a lock icon in the top-left corner of the window border. When a top-level workspace is locked the locking applies to all subworkspaces within the top-level workspace.

A workspace that is executing is indicated by an arrow icon in the top-left corner of the window border. The arrow icon overrides the lock icon.

Locked workspaces can be used in two main ways:

- To restrict the operations that an end-user can perform on the application. This is particularly useful in graphical user interfaces.
- To create application-specific libraries from which the user can drag-and-drop objects in the same as from the built-in palettes.

2.3 Mouse Interaction

The primary mean for interaction with the editor is the mouse. Using the mouse the user selects objects, moves objects, connects objects together, brings up object specific pop-up menus, etc.

Select

- An object is selected by single-clicking on the object. When an object has been selected its resize handles are highlighted.
- A group of objects is selected by drawing an enclosing rectangle with the mouse.
- An object is added to the current selection by single-clicking on the object while pressing down the Shift key.
- An object is removed from the current selection by single-clicking on the object while pressing down the Shift key.

Move

- A selection is moved using the mouse in the standard way.
- Moving a selection while simultaneously pressing the Ctrl key will create a copy of the the selection,
- Moving a selection slowly against the border of the window of a workspace will cause scrolling or panning.
- Moving a selection rapidly towards another workspace will create a copy of the selection in the other workspace

Resize

- An object is resized through its resize handles.

Connect

- Two objects are connected by clicking on a port stub of one of the objects and moving the mouse while keeping the mouse button depressed to the port stub of the other object.
- A connection between two ports that are sufficiently close together is created simply by clicking on one of the ports.
- It is only possible to create syntactically legal connections.
- A connection is redrawn whenever any of the interconnected objects is moved.
- The corner points of a connection can be moved manually using the mouse. Once a connection segment has been placed manually it will remain at the new position also if the any of the interconnected objects is moved.

Show Object Menu

- Object-specific pop-up menus are shown by double-clicking on the object or clicking on the object using the right mouse button.

Object Help

- Object-specific help is available by selecting the **Object Help** menu option in the **Help menu**. This causes the editor to change mode into object-help mode. The object-help mode is indicated by a special "information cursor".
- When in object-help mode, object-specific help is obtained by clicking on an object, on a menu option, or on a toolbar button.
- To return to the standard editing mode, the **Object Help** menu option is selected again.

Drawing Lines

- When clicking on the Line or Spline buttons in the GUI palette, the editor changes mode to line-drawing mode.
- A line is is created by clicking in a workspace.
- For every subsequent click within the same workspace a new point of the line is created.
- The line-drawing is terminated by a double-click.
- For a line, the points of the line are interconnected by straight lines.
- For a spline, the points of the line are interconnected by cubic splines. At least three points are needed in order to make a cubic spline.
- If the start point and the end point are sufficiently close to each other a polygone (line) or "curved polygone" (spline) is created rather than a line/spline.
- The start and end points are sufficiently close if they have the same nearest grid point.

2.4 File Menu

The file menu contains the following menu choices.

- **New** (Keyboard accelerator: Ctrl-N Toolbar button:)

The new menu choice is used to create a new top-level workspace. The new workspace receives a default name and automatically receives focus.

- **Open** (Keyboard accelerator: Ctrl-O Toolbar button:)

A saved application is opened. A file chooser menu is used to select the file to be opened.

- **Save** (Keyboard accelerator: Ctrl-S Toolbar button:)

The top-level workspace together with all its subworkspaces are saved on a file using XML-format. If the workspace has not been saved before a file chooser menu is obtained. Selecting **Save** for a subworkspace is equivalent to saving the top-level workspace that the subworkspace belongs to. It is not possible to save if the workspace is executing.

- **Save As**

Equivalent to **Save**, but a file chooser menu is always shown. Using **Save As** it is possible to save a top-level workspace under a new file name.

- **OpenAll**

Open a set of top-level workspaces that have been saved in a single XML file using **SaveAll** or **SaveAllAs**. This operation is currently error-prone and should be avoided.

- **SaveAll**

Equivalent to **Save**, but saves all top-level workspaces currently in the editor as a single file. This operation is currently error-prone and should be avoided.

- **SaveAllAs**

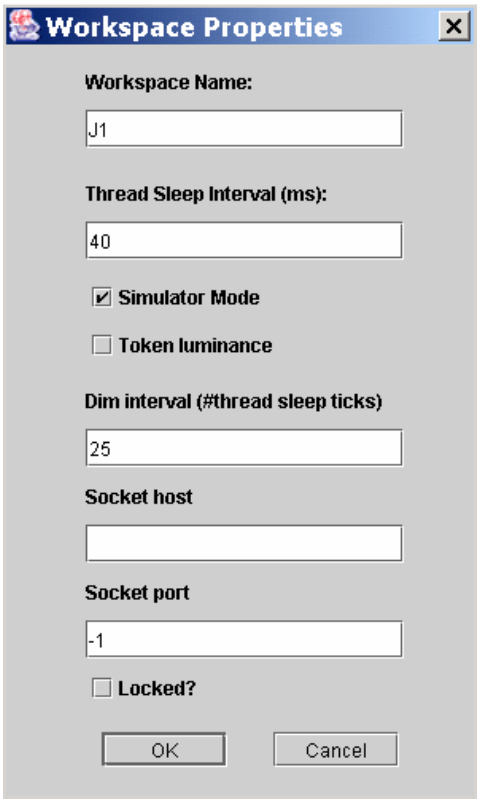
Equivalent to **Save As**, but saves all top-level workspaces currently in the editor as a single file. This operation is currently error-prone and should be avoided.

- **Print**

Prints the workspace cuurently in focus. The operation calls the Java printing facilities and its behavior is dependent on the print support available on the current execution platform.

- **Properties** (Tool bar button:)

Bring up a dialog that allows the user to change different properties related to the Grafchart execution thread associated with the current top-level workspace. Only accessible from top-level workspaces. The dialog is shown below.



Workspace Name: The name of the top-level workspace.

Thread Sleep Interval: The period of the execution thread associated with the top-level workspace in milliseconds. Default value is 40.

Simulator Mode: In simulator mode the digital and analog input variables are disconnected from their associated digital and analog I/O. Instead the user may change their values from the editor. In non-simulator mode the digital and analog input variables are connected to the I/O. Simulator mode is the default.

Token Luminance & Dim interval: Token luminance is useful for animating the execution of a function chart at high execution speeds when the token indicating the currently active step(s) moves rapidly between steps. Token luminance creates a trail of outlined and shaded tokens that follows the normal token. The dim interval decides how many scan-cycles that the outlined token should be visible after the deactivation of a step. Token luminance is only supported for simple function charts containing steps, transitions, and macro steps.

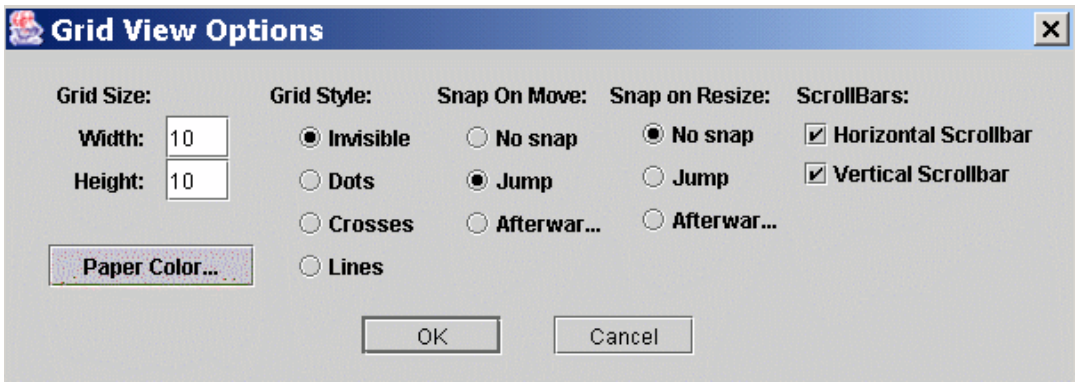
Socket Host: The IP address of the TCP socket server connected to the top-level workspace, An IP address number or localhost. If the string is empty no socket reader thread is created for the top-level workspace.

Socket Port: The port number of the TCP socket server. Default value is -1, indicating that no socket communication is used for the current top-level workspace.

Locked: Determines if the workspace should be locked or not. Default mode is not locked.

- **Grid**

Bring up a dialog that makes it possible to change different graphical properties of the workspace.



Grid Size: Non-negative integers determining the width and height of the grid of the workspace.

Grid Style: Decides the style of the grid.

Snap On Move: Determines how moved objects should be related to the grid. **No snap** does not snap the object to the grid. **Jump** snaps to the grid continuously during the move. **Afterwards** does not perform the snap until the object is released.

Snap on Resize: Determines how resized objects should be related to the grid.

Horizontal Scrollbar: Determines if the workspace should have a horizontal scrollbar.

Vertical Scrollbar: Determines if the workspace should have a vertical scrollbar.

Paper Color: Makes it possible to change the background color of the workspace. Brings up a Java color choser menu.

- **Remove Palette**

Remove the palette from the editor. Can be useful during execution when the screen size is limiting.

- **Add Palette**

Add the palette to the editor.

- **Remove Toolbar**

Remove the toolbar from the editor.

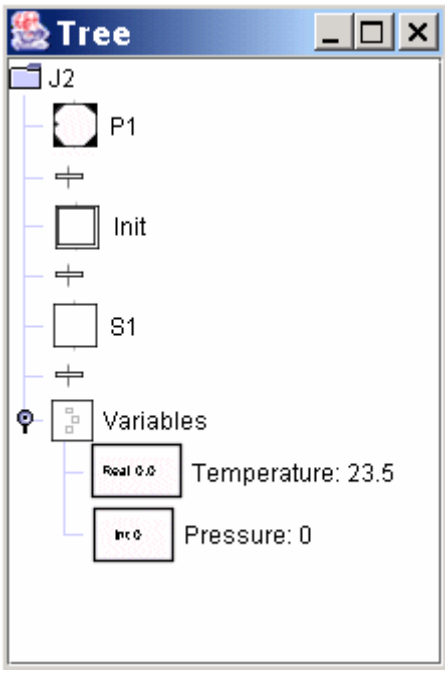
- **Add Toolbar**

Add the toolbar to the editor.

- **Show Tree** (Toolbar button:)

Displays a hierarchical tree view navigation window of the Windows "explorer" type. The tree view shows a snapshot of all the objects in the current workspace and its subworkspaces. By clicking with the left mouse button on a node in the tree, the node becomes selected in its corresponding workspace view. Clicking with the right mouse button on a tree node has the same effect as clicking on the corresponding object in the workspace view, e.g., an object-specific pop-up menu is shown. All hierarchical objects, e.g., workspace objects and macro steps, can be expanded and collapsed. It is possible to have multiple tree display open at the same time. It is important to understand that the tree only displays a snapshot of the application with the state it was in when the menu choice was selected, i.e., it is not updated if something changes. An

example of the tree navigation window is shown below.



- **Exit**

Exit JGrafchart. No saving of workspaces is performed. JGrafchart can also be exited using the delete button on the JGrafchart window. Before JGrafchart is closed down, the user must explicitly confirm the operation.

2.5 Edit Menu

The edit menu contains the following menu choices.

- **Cut** (Keyboard accelerator: Ctrl-X Toolbar button:.)

Cut the current selection to the clipboard.

- **Copy** (Keyboard accelerator: Ctrl-C Toolbar button.)

Copy the current selection to the clipboard.

- **Paste** (Keyboard accelerator: Ctrl-V Toolbar button:)

Paste the content of the clipboard to the workspace. The newly pasted objects will be automatically selected. The default position where the pasting takes place is the original location of the objects offset one grid-width to the right and one grid-height down. The pasting position can be changed by clicking on the workspace at the desired position before performing the paste.

- **Delete** (Toolbar button:)

Delete the current selection.

- **Delete top-level workspace**

Delete the current top-level workspace. Before the workspace is deleted, the user must confirm the deletion.

- **Select All** (Keyboard accelerator: Ctrl-A)

Select all objects in the current workspace.

- **Group**

Group the objects in the current selection into a single object. Grouping is still quite error-prone.

- **Ungroup**

Ungroup the currently selected previously grouped object.

- **Undo** (Keyboard accelerator: Ctrl-Z Toolbar button:)

Undo the last editor operation. Only operations that are of a graphical nature can be undone. Every workspace has its own undo history. The undo history is maximized to 30 past operations. Recording of undoable operations is turned off during execution. It is not possible to undo the deletion of an entire workspace. It is not possible to undo all operations.

- **Redo** (Keyboard accelerator: Ctrl-Y Toolbar button.)

Redo the last undone operation. It is not possible to redo all operations.

- **Move to front**

Move the currently selected objects in front of any other objects in the workspace.

- **Move to back**

Move the currently selected objects behind any other objects in the workspace.

- **Zoom in** (Keyboard accelerator. Ctrl-B Toolbar button:)

Zoom in on the current workspace with 10%.

- **Zoom out** (Keyboard accelerator: Ctrl-D Toolbar button.)

Zoom out on the current workspace with 10%.

- **Zoom normal size** (Keyboard accelerator: Ctrl-F)

Zoom back to the normal size.

- **Zoom to fit**

Zoom in on a workspace so that all objects fit in the current workspace window.

- **Overview**

Bring up a separate top-level window containing a miniature of the contents of the document showed by the current workspace. The overview window also contains an outlined rectangle showing the current position of the view of the document shown in the workspace. When the rectangle is moved the current workspace view is panned and scrolled accordingly.

2.6 Execute Menu

The execute menu contains the following menu choices.

- **Compile** (Toolbar button:)

Compile the current top-level workspace and all its subworkspaces. Applying the menu choice to a subworkspace is equivalent to applying it to the top-level workspace of the subworkspace. Compilation error and warnings are shown in the message menu.

- **Compile all**

Apply compile to all top-level workspaces in the system.

- **Execute** (Toolbar button:)

Start the execution of the current top-level workspace context. Applying the menu choice to a subworkspace is equivalent to applying it to the top-level workspace of the workspace. It is only possible to start execution of a top-level workspace if the top-level workspace has been previously compiled without any compilation errors. During execution it is not possible to drag-and-drop from the palette onto the workspace. During execution the undo history is switched off. During execution the top-level frame icon is changed to an arrow icon .

- **Stop** (Toolbar button:)

Stop the execution of the current top-level workspace context. Applying the menu choice to a subworkspace is equivalent to applying it to the top-level workspace of the workspace.

- **Shutdown client**

Stop the socket reader thread associated with the workspace and close the socket connection.

2.7 Misc Menu

The menu choices available in the misc. menu are the following:

- **Connect CCOM**

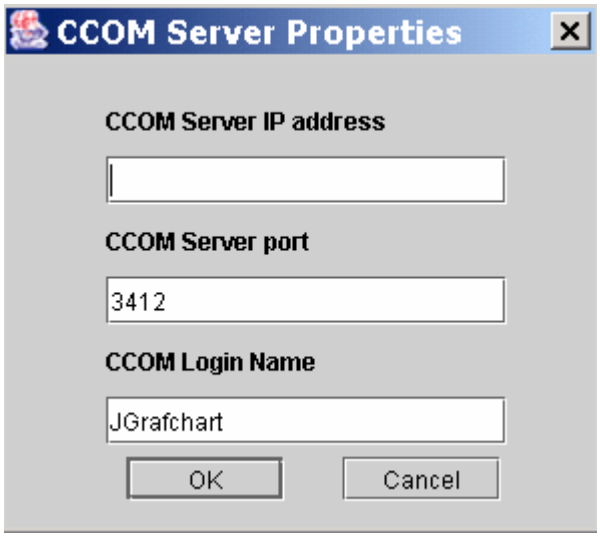
Connects to the CCOM server (Requires that the CCOM infrastructure is available).

- **Disconnect CCOM**

Disconnects from the CCOM server (Requires that the CCOM infrastructure is available).

- **CCOM Server**

Brings up a dialog from which the CCOM server properties can be edited.



CCOM Server IP address:
The IP address to the CCOM/xmlBlaster node.

CCOM Server Port:
The port of the CCOM server. (Default = 3412)

CCOM Login Name:
The login name to CCOM. (Default = JGrafchart)

- **Enable slide mode**

Change the editor into slide show mode.

- **Disable slide mode**

Change the editor into its normal mode.

2.8 Help Menu

The menu choice available in the help menu is the following.

- **On-Line Help**

Show the interactive help system browser.

- **Object Help** (Toolbar button:)

A toggle menu choice that changes the editor mode into object-help mode. In object-help mode, specific help information is obtained by clicking on an object, a menu option, or a toolbar button.

- **Open On-Line Tutorial**

Brings up the JGrafchart interactive on-line tutorial containing examples of how to use the different elements of JGrafchart. The tutorial is an ordinary JGrafchart application residing in the file ../grafchart/doc/tutorial/tutorial.xml. The tutorial can be changed and extended by the user.

- **Close On-Line Tutorial**

Closes the on-line tutorial.

2.9 Command Line Arguments

JGrafchart supports the following command line arguments:

- -geometry 1024x768

The initial size of the JGrafchart window (default 1024x768)

- -windows 1

Use the Windows look-and-feel UI rather than the standard Java look-and-feel.

- -name JGrafchart

The name of the JGrafchart application. This will also be the identifier used in CCOM communication. Default is JGrafchart.

- -localIO "class_name"

The implementation class for the custom IO.

- -connect 1

Make an automatic connect to the CCOM server.

- -serverIP "IP address"

The CCOM server IP address. Localhost is not a valid value.

- -port "number"

The port number that the CCOM server listens to.

- -Xincgc

Enables incremental Java GC. This option should be given to the JVM and not to JGrafchart.

- -load "file_name"

Loads the application named by file when JGrafchart has started. The file name should be an absolute reference.

- -loadcompile "file_name"

Loads and compiles the application named by file when JGrafchart has started. The file name should be an absolute reference.

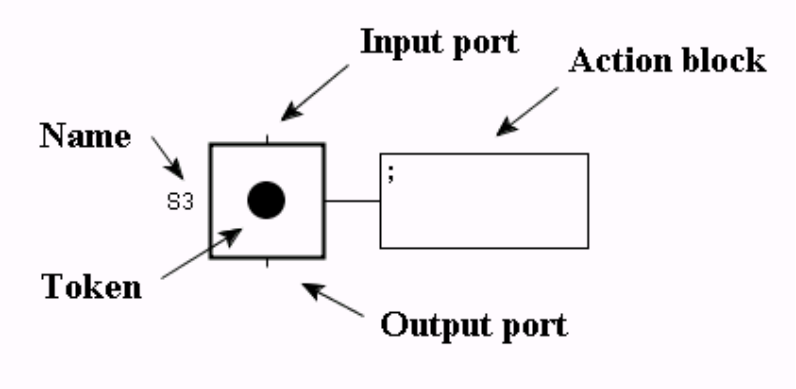
- -loadcompilestart "file_name"

Loads, compiles, and, if the compilation was successful, starts the application named by file when JGrafchart has started. The file name should be an absolute reference.

3 Language Reference

3.1 Graphical Function Chart Elements

3.1.1 Step



A step represents a state of the application. The step can be active or inactive. An active step is indicated by a **token.** The step has one **input port** and one **output port**. The input port can be connected to transitions, exception transitions, and parallel split objects. The output port can be connected to transitions and parallel join objects. Step actions are shown in the **action block** associated with the step. The action block can be hidden (default) or shown.

A step may have a **name**. By default the step has no name. The user can give the step a name by clicking on the (invisible) name text-edit field to the left of the step. Steps names are needed in references to the .s and .t attributes of the step.

Changes to the step actions will not take effect until the next time the step is compiled.

Menu Choices:

- **Show Action Block**

  Shows the action block of the step.

- **Hide Action Block**

  Hides the action block of the step.

- **Edit**

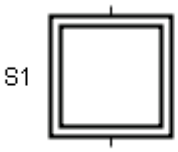
  Brings up a text-editor window in which the user can enter and edit step actions. The editor supports cut, copy, and paste using the normal keyboard commands. Individual actions should be separated by a semi-colon. The editor can also be brought up by clicking on the action text in the action block. If the step contains actions, the action text is shown as a tool-tip text when the action block is hidden.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

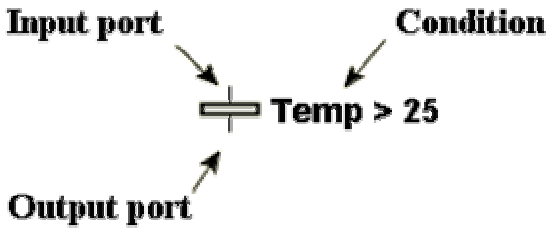
3.1.2 Initial Step

The initial step is an ordinary step that is automatically activated when the application is started. Initial steps may be located on top-level workspaces or within workspace objects. Initial steps within macro steps will currently not be initialised properly.



Initial steps have the same menu choices and text methods as ordinary steps.

3.1.3 Transition



A transition represents a condition for changing from one state to another state. The transition is enabled when all the steps connected to the **input port** (either directly or through a parallel join object) are active. The transition fires if it is enabled and its associated **condition** is true. When the transition fires it deactivates all the steps connected at the input port and activates all the steps connected at the **output port** (either directly or through a parallel split object).

The input port can be connected to all types of steps and parallel join objects. The output port can be connected to the input port of all types of steps, to parallel split objects, and to the history port of macro steps.

During execution the truth-value of the condition of the transition is indicated using colors. A true condition is indicated by a green transition. A false condition is indicated by a red transition. The condition text can be shown (default) or hidden. With one exception, changes to the transition condition while executing will not take effect until the next time the transition is compiled. The exception is that changes of conditions involving only boolean literals will take effect immediately. For example, it is possible to change the condition from "1" to "0" at run-time and vice versa.

Transitions currently have no name.

Menu Choices:

- **Show Condition**

  Shows the condition of the transition.

- **Hide Condition**

  Hides the condition of the transition. When the condition is hidden it is shown as a tool-tip text.

- **Edit**

  Brings up a text-editor through which the transition condition can be changed. The same editor can also be brought up by simply clicking on the condition text.

- **Force**

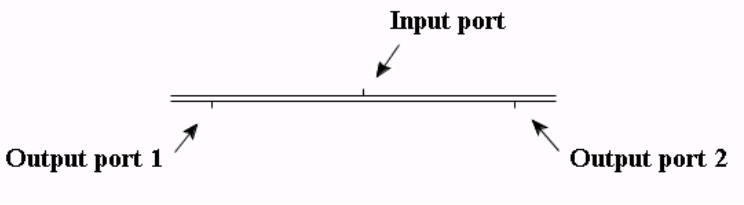
  Forces an enabled transition to be fired by overriding the transition condition.

Text Methods:

Since transitions have no name there is no way to create references to them and, hence, transitions have no text methods. This is subject to change in future releases.

3.1.4 Parallel Split

The parallel split is used to split up the execution in two parallel branches. If more than two branches are needed the parallel split objects can be connected in series (an output port of the first parallel split object is connected to the input port of the second parallel split).

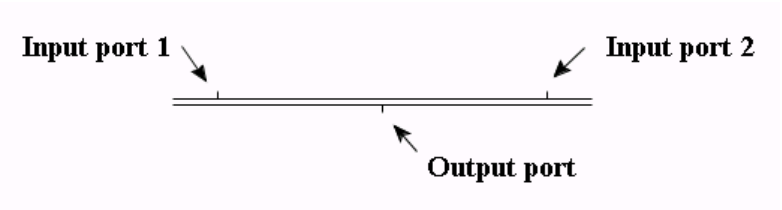


The **input port** of a parallel split may be connected to the output ports of transitions and exception transitions and to a output port of other parallel split objects. The **output port** of a parallel split object may be connected to the input port of an arbitrary step and to the input port of another parallel split object. When a transition preceding the parallel split is fired all steps connected directly or indirectly to the output ports of the parallel split are activated.

Parallel splits have no menu choices and no text methods

3.1.5 Parallel Join

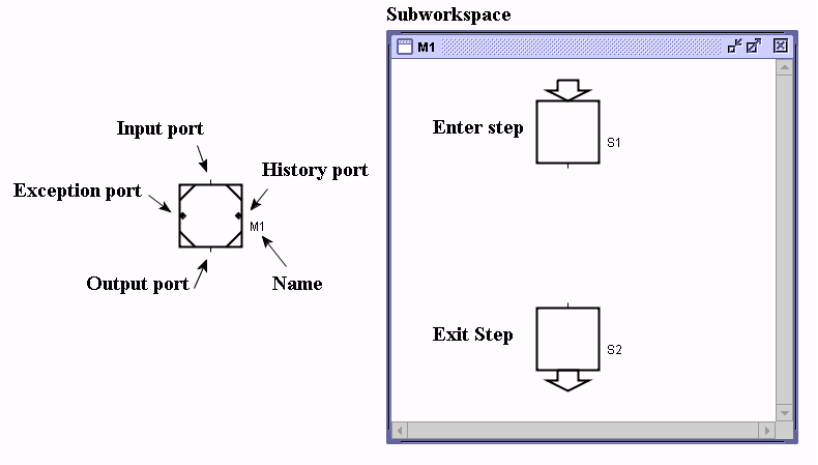
The parallel join is used to merge together the execution in two parallel paths into a single path. If more than two branches need to be joined the parallel join objects can be connected in series (the output port of the first parallel join is connected to an input port of the second parallel join).



The **input port** of a parallel join may be connected to the output ports of arbitrary steps and to the output port of another parallel join objects. The **output port** of a parallel join may be connected to the input port of a transition. The transitions connected to the output port of a parallel join are not enabled unless all the steps connected directly or indirectly to the input ports of the parallel join are active. When the transition fires all the preceding steps are deactivated.

Parallel joins have no menu choices and no text methods.

3.1.6 Macro Step



A macro step represents a hierarchical step containing a substructure of steps and transition on the **subworkspace** of the macro step. A macro step may have one or several **input ports** and one or several **output ports.** By default a macro step has one input port and one output port. An **enter step** located on the subworkspace of the macro step is associated with each input port. The association is based upon the x-position of the enter step. The leftmost enter step is associated with the leftmost input port et cetera. An **exit step** located on the subworkspace of the macro step is associated with each output port, Also in this case it is the geographical location of the exit step that decides the association. The input ports may be connected to the output ports of transitions, exception transitions, and parallel splits. The output ports may be connected to the input ports of transition, exception transitions, and parallel joins.

When a macro step is activated through a certain input port, the corresponding enter step is activated. When an exit step is activated the transitions connected to the output port associated with the exit step are enabled. When any of these transitions fires both the exit step and the entire macro step are deactivated.

On the left-hand side of the macro step is a special **exception output port**. To this port it is only possible to connect the input ports of exception transitions. The exception transitions are enabled all the time while the macro step is active. If fired they will abort the execution of the macro step. Exception transitions have priority over "ordinary" transitions. Macro steps remember the state they were in when they last were aborted. By activating the macro step through the **history input port** on the right hand side of the macro step the macro step is resumed in the state it was in when it last was aborted. Resuming a macro step that never has been aborted before is considered an error.

A macro step may have actions in the same way as ordinary steps. The stored actions of the macro step are executed *before* the stored actions of the enter step. Similarly, the exit actions of the macro step are executed *after* the execution of the exit actions of the exit step. The periodic actions of the macro step are executed while the macro step is active, independently of which internal step that is active. The abort actions of the macro step are executed when the macro step has been aborted due to the firing of an exception transition. The abort actions of the macro step are executed after the abort actions of the internal steps of the macro steps.

A macro step may have a **name**. By default the macro step has no name. The user can give the macro step a name by clicking on the (invisible) name text-edit field to the right of the step (see the figure above). Macro steps names are needed in references to the .s, .x and .t attributes of the macro step.

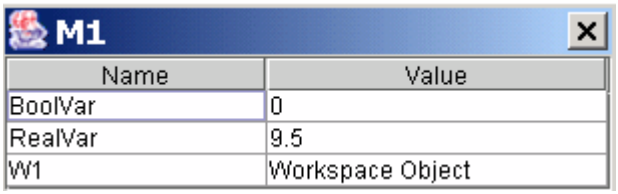
Menu Choices:

- **Show/Hide Body**

  A toggle menu choice that shows the subworkspace of the macro step if it is hidden and shows the subworkspace if it is shown. The subworkspace remembers its actual position, size, and scale factor when it is hidden.

- **Table**

Brings up a table containing all internal variables, analog inputs and outputs, digital inputs and outputs, and all hierarchical elements (macro steps, workspace objects, XML Message In, XML Message Out, procedures) found inside the macro step. By double clicking on the value of an internal variable or input/output it is possible to change the current value of the object. This is, however, only possible if the workspace that the object resides in is not locked, and, in the case the object is an internal variable, the internal variable has not been declared constant. By double-clicking on the value column for a hierarchical object, a new table is shown contain the internals of the hierarchical object. The variable values in the table are dynamically updated. The table is shown below:



- **Edit**

Brings up the step action editor.

- **Update Stubs**

Adds or removes input and output port stubs in such a way that the number of port stubs corresponds to the number of enter and exit steps on the subworkspace of the macro step. To create a macro step with multiple input port and/or output ports the user adds new enter steps and/or exit steps to the subworkspace and then updates the stubs. Enter steps and exit steps are not available in the palette. Therefore, they must be created using copy and paste.

Text Methods:

- setWorkspaceColor(int color)

Sets the color of the subworkspace to the given integer RGB color.

- int getWorkspaceColor()

Returns the color of the subworkspace as an integer.

- showWorkspace()

Shows the subworkspace of the macro step.

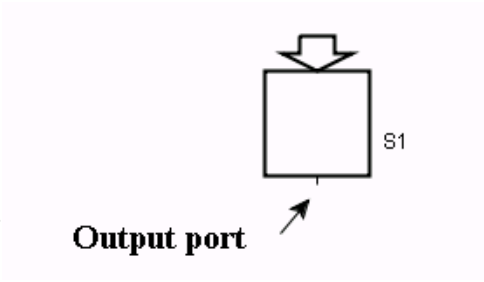
- hideWorkspace()

Hides the subworkspace of the macro step.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

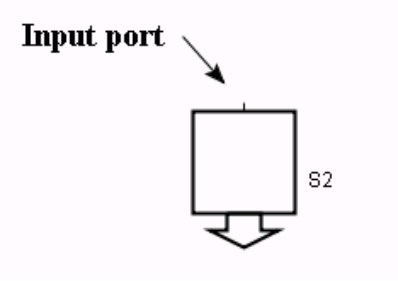
3.1.7 Enter Step

The enter step is an ordinary step that is activated when a macro step is activated through an input port or when a procedure call is made. The enter step has an action block and may have actions in the same way as ordinary steps. The connection rules for the **output port** are the same as for ordinary steps. A macro step may contain multiple enter steps, but a procedure may only contain one enter step.



Enter steps have the same menu choices and text methods as ordinary steps.

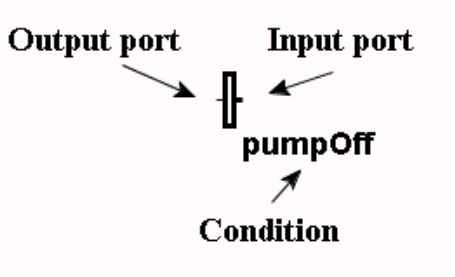
3.1.8 Exit Step



The exit step is an ordinary step that marks the last step within a macro step or within a procedure. When the exit step of a macro step is activated the transitions connected at the corresponding output port of the macro step are enabled. When the exit step of a procedure is activated the behavior differs depending in which way the procedure was invoked. If the procedure was invoked from a procedure step then the transitions connected at the output port of the procedure step is enabled. When it fires the exit step and the procedure step are deactivated and the actual procedure call is terminated. If the procedure is invoked directly from the procedure object or if it is invoked from a process step then the procedure call is immediately terminated without executing any of the actions of the exit step. A macro step may contain multiple exit steps, but a procedure may only contain one exit step. The connections rules for the **input port** are the same as for ordinary steps.

Exit steps have the same menu choices and text methods as ordinary steps.

3.1.9 Exception Transition

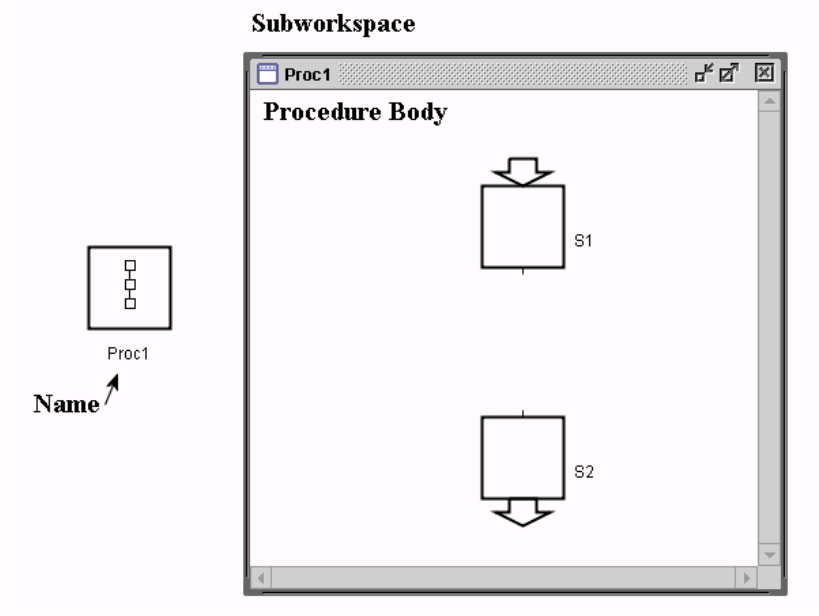


An exception transitions is a special transition for which the input port only may be connected to the exception output port of macro steps and of procedure steps. When the exception transition is fired the macro step or procedure step is aborted and the corresponding abort actions are executed. Exception transitions have priority over ordinary transitions. When connected to a macro step it is possible to refer to the internal state of the macro step from the condition. For example, it is possible to express that an exception only may be fired if the macro step is in a certain internal state. The connection rules for the output port are the same as for ordinary transitions.

Exception transitions have the same menu choices as ordinary transitions and it has currently no text methods.

3.1.10 Procedure

Step-transition sequences that are used in several different contexts can be represented as procedures. A procedure has a **procedure body** stored on the subworkspace of the procedure. The body begins with one enter step and ends with one exit step. Procedures are re-entrant and may be recursive.



A procedure can be called in four different ways:

- Directly from the procedure object.
- Through a **procedure step**.
- Through a **process step**
- Through the spawn action..

When a procedure is called from a procedure step, the transition(s) after the procedure step does not become enabled until the execution has reached the exit step of the procedure. Hence, this corresponds to an ordinary procedure call. When a procedure is called from a process step the transition(s) after the procedure step is enabled immediately. Conceptually, the procedure call is spawned and executed in a separate execution thread.

The procedure step may contain arbitrary language elements, except other procedures, input variables, and output variables.

Procedures may have parameters. All internal variables in the procedure body can be used as parameters. In addition it is also possible to use workspace objects as parameters if the workspace objects only contain internal variables and other workspace objects with the same restriction. Workspace objects of this type represent structured variables. Internal variable parameters can be assigned values both using call-by-value and call-by-reference. Workspace object parameters can only be assigned values using call-by value.

Menu Choices:

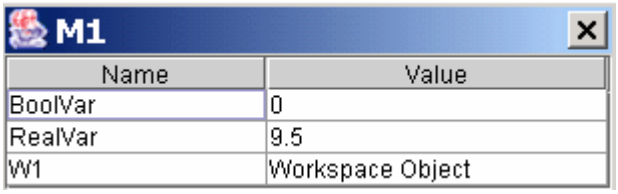
- **Show/Hide Body**

  A toggle menu choice that shows the subworkspace of the procedure if it is hidden and shows the subworkspace if it is shown. The subworkspace remembers its actual position, size, and scale factor when it is hidden.

- **Table**

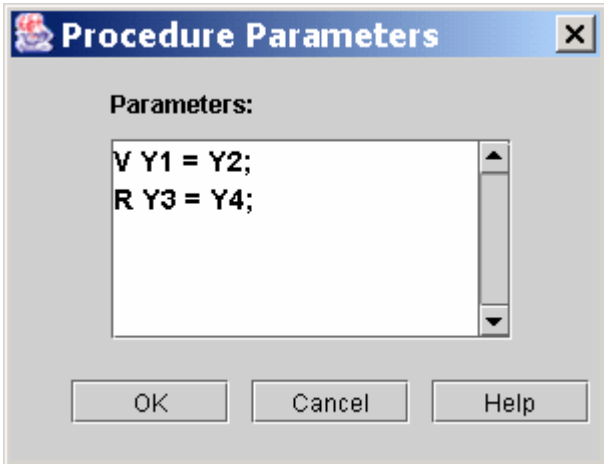
  Brings up a table containing all internal variables, analog inputs and outputs, digital inputs and outputs, and all hierarchical elements (macro steps, workspace objects, XML Message In, XML Message Out, procedures) found inside the procedure. By double clicking on the value of an internal variable or input/output it is possible to change the current value of the object. This is,

however, only possible if the workspace that the object resides in is not locked, and, in the case the object is an internal variable, the internal variable has not been declared constant. By double-clicking on the value column for a hierarchical object, a new table is shown contain the internals of the hierarchical object. The table is shown below:



- **Parameters**

Brings up a parameter text editor. This editor is used to set the parameter values when the procedure is called directly from the procedure object.



The syntax for call-by-value parameter assignments is

V <internal_variable> = <expression>;

In the example above the internal variable Y1 in the procedure body will be assigned the value of variable Y2 when the procedure is called.

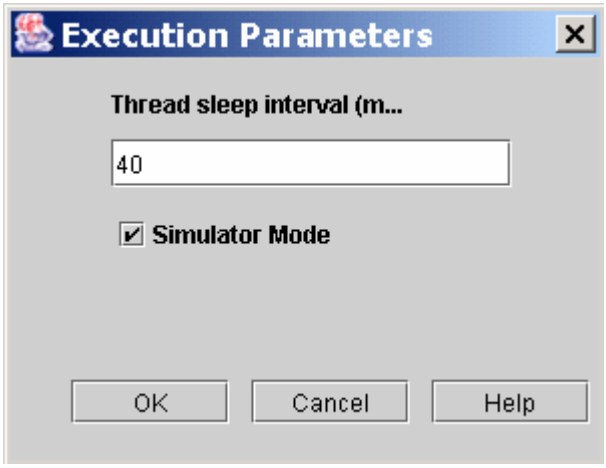
The syntax for call-by-reference parameter assignments is

R <internal_variable> = <variable>;

In the example above a redirection link is set up between the internal variable Y3 in the procedure body and the variable Y4. Every reference and assignment to Y3 within the procedure body will be redirected to Y4. In this way it is possible to pass out values from the call to a procedure.

- **Start**

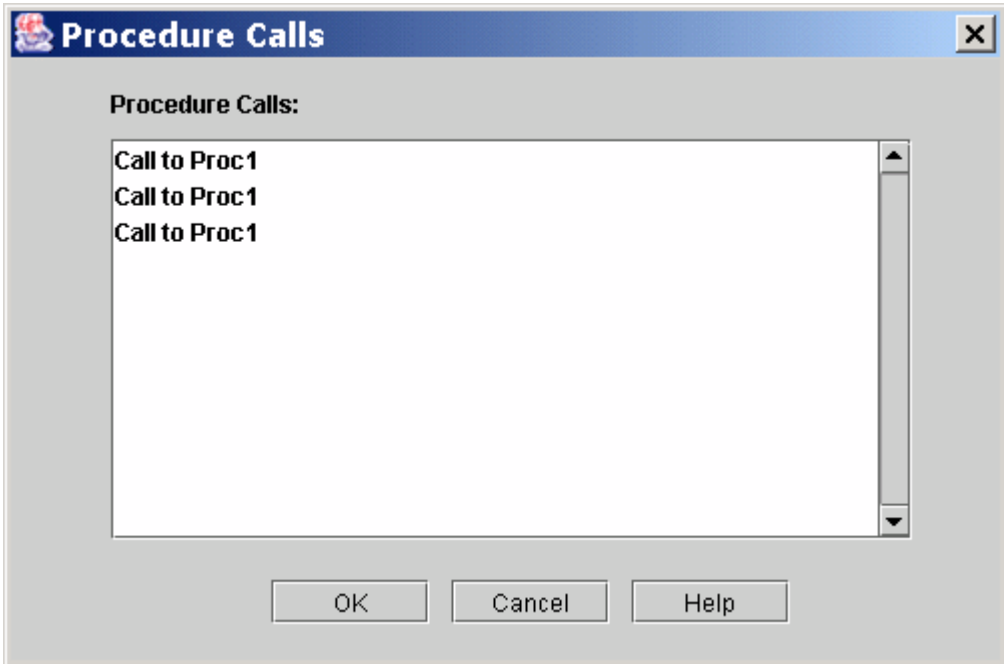
Brings up a parameter from which it is possible to call the procedure.



When invoked in this way each procedure call will be executed by a separate execution thread. The thread sleep interval decides the scan period of this thread. The simulator mode checkbox decides whether analog and digital input and output variables should be connected to the IO or only to the graphical user interface.(NOTE: Should probably be removed. Not relevant since procedures may not contain input and output variables.)

- **Calls**

Brings up a menu showing all currently executing invocations of the procedure. By selecting one of them the body of that particular procedure call is shown.



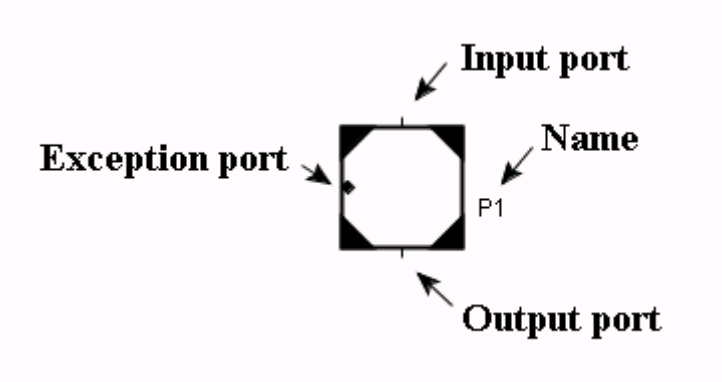
Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)

- setHeight(int height)
- setVisible(boolean visible)
- spawn(String parameters)

Starts a call to the procedures as a separate execution thread in the same way as if the procedure had been started with a Process Step. The parameter argument is a string containing a parameter list with same format as for a Procedure Step or Process Step. The parameters are evaluated in the context of the action that the method call is part of, i.e., within the context of the Step or Button that contains the corresponding action.

3.1.11 Procedure Step

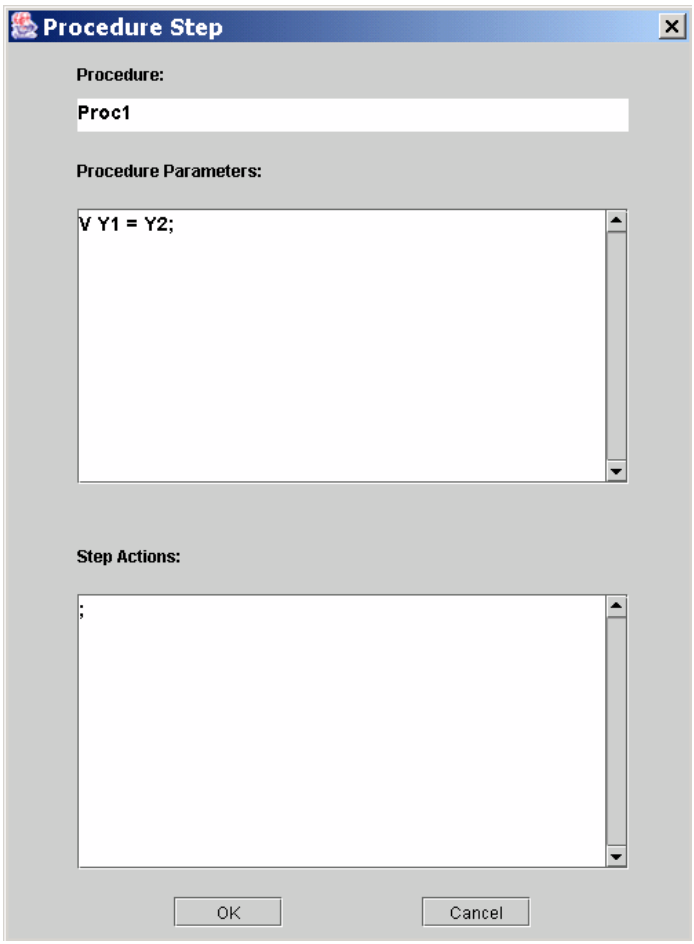


A procedure represents a call to a procedure. The **input port** and the **output port** can be connected in the same way as for an ordinary step. The **exception output port** behaves in the same way as the exception output port of a macro step. However, it is not possible to resume the execution of a procedure. The transitions connected to the output port are enabled when the enter step of the procedure is activated. When any of the transitions are fired the exit step is deactivated, the procedure call is terminated, and the procedure step is deactivated.

Menu Choices:

- **Edit**

  Brings up a editor from which the users sets the procedure to be called, the procedure parameters, and the procedure step actions.



The procedure to be called is entered in the Procedure field. The reference can be written in several ways:

- Proc1 - A reference to the procedure named Proc1.

- "Proc1" - A reference to the procedure named Proc1.
- StringVar - A reference to the variable named by the string variable StringVar.
- "Proc" + "1" - A reference to the procedure named Proc1.
- ObjectVar + "." + MethodVar - A reference to the procedure named by taking the value of the string variable ObjectVar and the value of the string variable MethodVar and concatenating them with a "." in between,

The procedure parameters are entered in the Procedure parameters field. The syntax for call-by-value parameter assignments is

V <internal_variable> = <expression>;

In the example above the internal variable Y1 in the procedure body will be assigned the value of variable Y2 when the procedure is called. The expression will be evaluated in the context of the procedure step.

The syntax for call-by-reference parameter assignments is

R <internal_variable> = <variable>;

In the example above a redirection link is set up between the internal variable Y3 in the procedure body and the variable Y4. Every reference and assignment to Y3 within the procedure body will be redirected to Y4. in this way it is possible to pass out values from the call to a procedure.

Procedure step actions are entered in the Step actions field. The normal step action syntax is used.

- **Call**

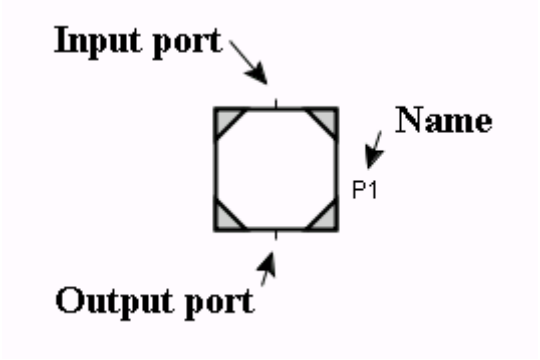
Brings up the body of the procedure call.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.1.12 Process Step

A process step is a special version of a procedure step where the call to the procedure is started as a separate execution thread. The transitions connected at the output port are enabled immediately when the procedure has been started. It is possible to have several executing procedure calls emanating from the same process step. When the execution within a procedure call reaches the exit step the procedure call is terminated without executing any actions of the exit step. Process steps have no exception output ports.



Menu Choices:

- **Edit**

  Brings up a editor from which the users sets the procedure to be started, the procedure parameters, and the process step actions. The editor is the same as for the procedure step.

- **Calls**

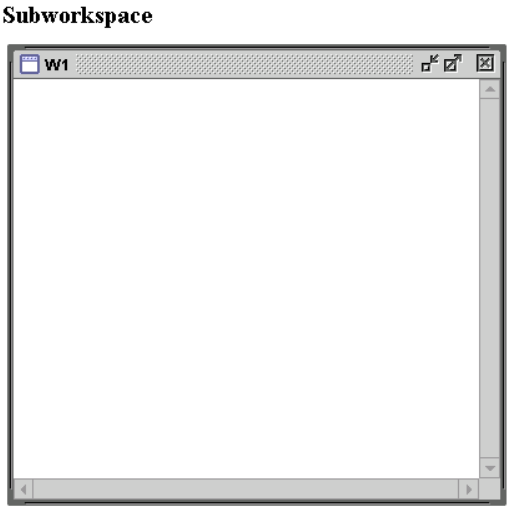
  Brings up a menu showing all currently executing invocations of the procedure. By selecting one of them the body of that particular procedure call is shown.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.1.13 Workspace Object

A workspace is an object that contains a subworkspace. On the subworkspace it is possible to place arbitrary JGrafchart elements. The workspace object can be used for several purposes, e.g.;



- As a structuring element for decomposing large applications.
- As a way of representing structured variables. In this case the workspace object should only contain variables and other workspace objects only containing variables. References to the variables are made using dot-notation.
- As a way of representing objects. In this case the workspace should only contain variables (attributes/state) and procedures (methods/behaviour)

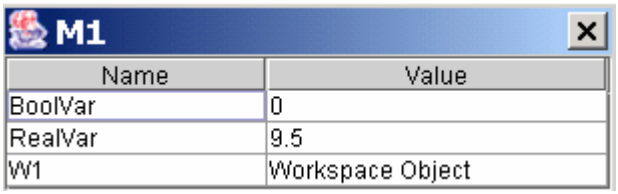
Menu Choices:

- **Show/Hide Body**

A toggle menu choice that shows the subworkspace of the workspace object if it is hidden and hidess the subworkspace if it is shown. The subworkspace remembers its actual position, size, and scale factor when it is hidden.

- **Table**

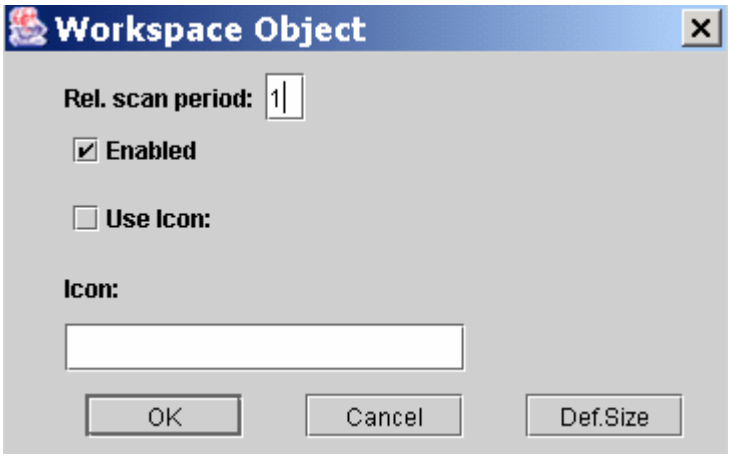
Brings up a table containing all internal variables, analog inputs and outputs, digital inputs and outputs, and all hierarchical elements (macro steps, workspace objects, XML Message In, XML Message Out, procedures) found inside the workspace object. By double clicking on the value of an internal variable or input/output it is possible to change the current value of the object. This is, however, only possible if the workspace that the object resides in is not locked, and, in the case the object is an internal variable, the internal variable has not been declared constant. By double-clicking on the value column for a hierarchical object, a new table is shown contain the internals of the hierarchical object. The variable values in the table are dynamically updated. The table is shown below:



Name	Value
BoolVar	0
RealVar	9.5
W1	Workspace Object

- **Edit**

Brings up an edit dialog



Relative scan period:
By default the contents of a workspace object execute at the same rate as the top-level workspace that it is contained in. However by setting the relative scan period it is possible to have the workspace object execute at a slower rate. If the value is set to 2 the contents of the workspace object will only be executed every second time that that the workspace object is executed, etc. This is useful if the application contains parts with different execution rate demands.

Enabled:
When a workspace object is disabled the contents of the workspace object will not be executed. This only effects the execution of transitions and exception transitions. For, example, it is still possible to call a procedure that is located within a disabled workspace object. Input variables located within the workspace object will also still receive new values.

Use Icon:
Determines if the workspace object should use its default graphical presentation or if it should replace this by the icon given by the **Icon** text field.

Icon:
The file name to a .gif or .jpg file that may be used as an alternative graphical presentation of the workspace object.

Def.Size:
The size of the icon is modified to the default size of the icon associated with the workspace object.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- setWorkspaceColor(int color)

  Set the color of the subworkspace of the workspace object.

- real getWorkspaceColor()

Returns the color of the subworkspace of the workspace object.

- showWorkspace()

Shows the subworkspace.

- hideWorkspace()

Hides the workspace.

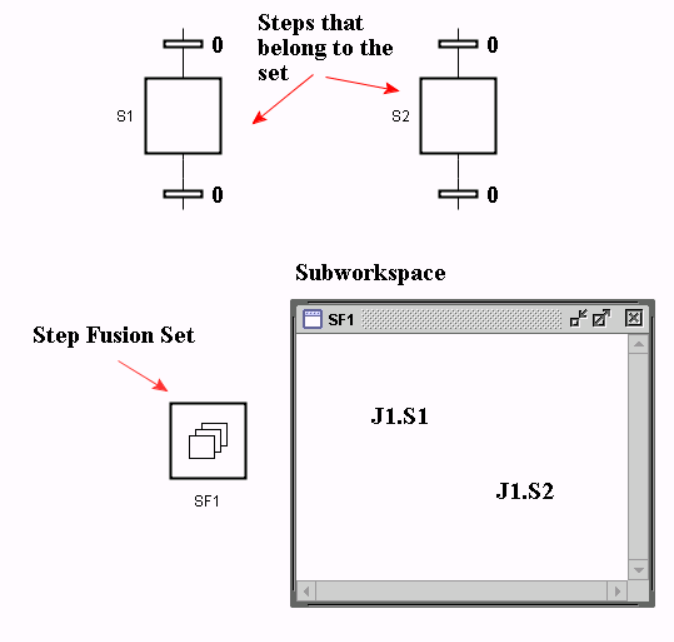
- setEnabled(boolean value)

Determines whether the workspace object should be enabled or disabled.

- int getTickTime()

Returns the tick length in milliseconds for the subworkspace context of the workspace. Takes the relative scan period into account. Can also be applied to top-level workspaces.

3.1.14 Step Fusion Set



A step fusion set makes it possible to have steps with multiple views. Each step that belongs to a step fusion set constitutes one view of the same conceptual step. When one of the steps in the set become active, all the steps in the set are activated, and when one of the steps is deactivated, all are deactivated. The steps that should belong to the same set are specified by **Free Text** objects on the subworkspace of the step fusion set. Global (qualified) references must be used. Step fusion sets may contain steps, macro steps, and procedure steps. Neither the step fusion set itself nor any of its views may be contained within a procedure. In principle it should be possible for a step to be a member of multiple step fusion sets, although this has not been tested.

A step fusion set may be non-abortive or abortive.

1. **Non-abortive step fusion sets:**

In a non-abortive fusion set, all the transitions connected to the output of any of the step fusion set views must be enabled in order for one of the transitions to be enabled. When an enabled transition is fired, all the steps in the set are deactivated and their exit actions are executed. The different steps of a non-abortive fusion set behave as if they all are connected in parallel.

2. **Abortive step fusion sets:**

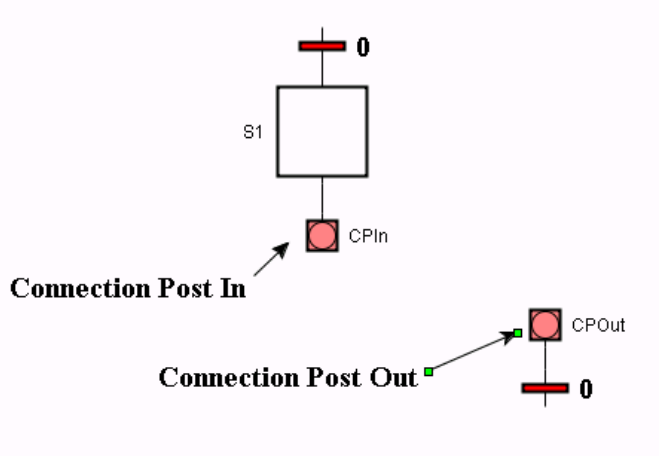
In an abortive fusion set, a transition connected to the output of one of the steps in the set is enabled independently of the status of the other steps in the set. When an enabled transition is fired, the step connected to the transition is terminated in the normal way and its exit actions are executed. The other steps in the fusion set are aborted and their abortive actions are executed.

Double clicking on the step fusion works as a Show/Hide Body menu choice on a workspace object.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.1.15 Connection Post



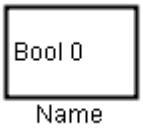
Connection posts are used to connect JGrafchart language elements without graphical connections. They can be used to split up a function chart and separate it on multiple workspaces. It can also conveniently be used to avoid "spaghetti function charts". A connection post connection consists of one **Connection Post In** object and one **Connection Post Out** object.

Double-clicking on a connection post brings up a dialog window where the user should enter the name of the remote connection post to which the connection should be created. Similarly, the corresponding name has to be entered in the remote connection post. **Note: Global (qualified) references must be used.**

When moving the mouse over a connection post the two connected connection posts are highlighted in red. The name of the remote connection posts is shown as a tool tip.

3.2 Variables and Lists

3.2.1 Boolean Variable

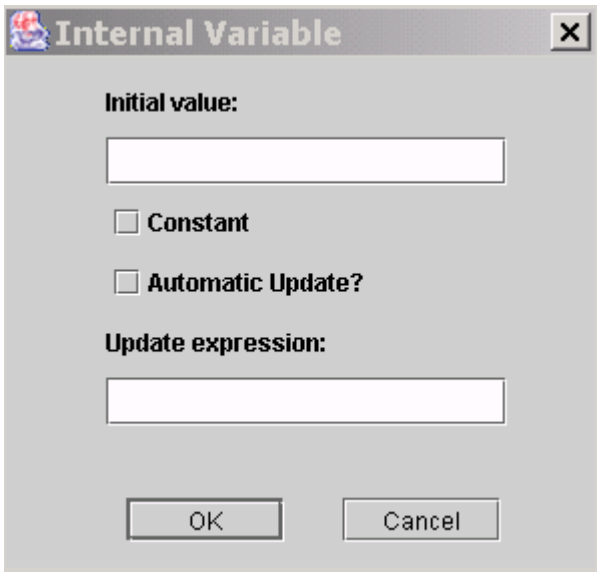


A boolean variable has the value true (represented by the literal 1) or false (represented by the literal 0). A boolean variable can be used in a transition condition and in an action. By default, a variable retains its value when the application is stopped and later restarted. Optionally, the user may give a variable an initial value. If the variable has an initial value its value will be set to the initial value when the application is started.

A boolean variable can be declared as being constant. A constant variable may not be changed at run-time, neither from a JGrafchart application nor by the user. A variable that is declared constant is shown with a blue icon.

A boolean variable may have an associated update expression. In this case the variable automatically receives a new value in the beginning of every scan cycle through an evaluation of the update expression. The update expression will be evaluated as a boolean expression. Boolean variables that receive their values from an update expression may be used in falling edge and raising edge event expression in the same way as Digital Inputs.

Double-clicking on the variable brings up an edit menu.



Initial value:
The initial value of the variable. An empty initial value field means that no initial value should be used.

Constant:
If selected the variable will be considered a constant which may not be written to.

Automatic Update?:
If selected the variable will automatically receive a new value in the beginning of every scan cycle through an evaluation of the update expression.

Update expression:
The automatic update expression for the variable.

The user can edit the value of the variable during execution.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- boolean isUpdated()

  Returns true if the variable receives it value from the update expression.

- setUpdated(boolean value)

  Decides if the value should receive its value from the update expression or not. May only be used to turn the updating on or off, if the updating was on when the variable was last compiled.

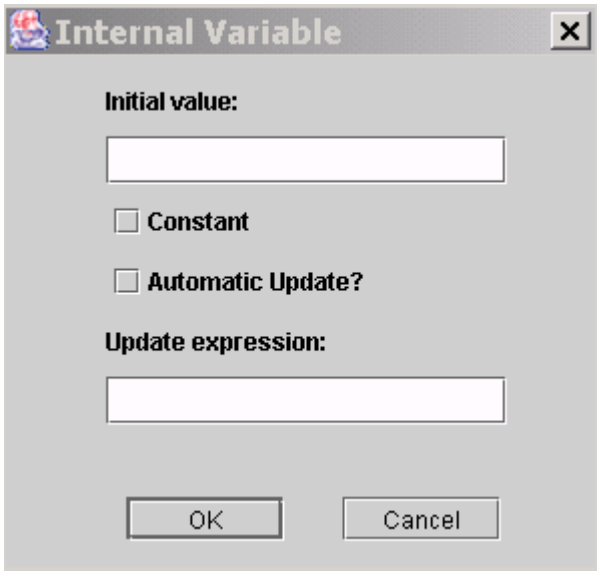
3.2.2 Integer Variable

An integer variable has a value that is represented by a Java int. An integer variable can be used in a transition condition and in an action. By default, a variable retains its value when the application is stopped and later restarted. Optionally, the user may give a variable an initial value. If the variable has an initial value its value will be set to the initial value when the application is started.

An integer variable can be declared as being constant. A constant variable may not be changed at run-time, neither from a JGrafchart application nor by the user. A variable that is declared constant is shown with a blue icon.

An integer variable may have an associated update expression. In this case the variable automatically receives a new value in the beginning of every scan cycle through an evaluation of the update expression. The update expression will be evaluated as an integer expression.

Double-clicking on the variable brings up an edit menu.



Initial value:
The initial value of the variable. An empty initial value field means that no initial value should be used.

Constant:
If selected the variable will be considered a constant which may not be written to.

Automatic Update?:
If selected the variable will automatically receive a new value in the beginning of every scan cycle through an evaluation of the update expression.

Update expression:
The automatic update expression for the variable.

The user can edit the value of the variable during execution.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- boolean isUpdated()

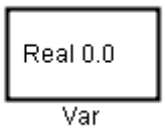
Returns true if the variable receives it value from the update expression.

- setUpdated(boolean value)

Decides if the value should receive its value from the update expression or not. May only be used to turn the updating on or off, if the updating was on when the variable was last compiled.

3.2.3 Real Variable

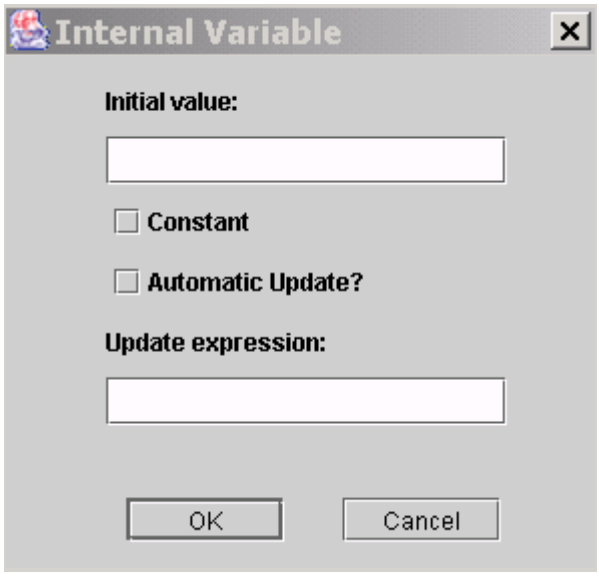
A real variable has a value that is represented by a Java double. A real variable can be used in a transition condition and in an action. By default, a variable retains its value when the application is stopped and later restarted. Optionally, the user may give a variable an initial value. If the variable has an initial value its value will be set to the initial value when the application is started,



A real variable can be declared as being constant. A constant variable may not be changed at run-time, neither from a JGrafchart application nor by the user. A variable that is declared constant is shown with a blue icon.

A real variable may have an associated update expression. In this case the variable automatically receives a new value in the beginning of every scan cycle through an evaluation of the update expression. The update expression will be evaluated as a real expression.

Double-clicking on the variable brings up an edit menu.



Initial value:
The initial value of the variable. An empty initial value field means that no initial value should be used.

Constant:
If selected the variable will be considered a constant which may not be written to.

Automatic Update?:
If selected the variable will automatically receive a new value in the beginning of every scan cycle through an evaluation of the update expression.

Update expression:
The automatic update expression for the variable.

The user can edit the value of the variable during execution.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- boolean isUpdated()

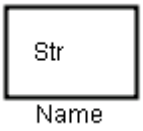
Returns true if the variable receives it value from the update expression.

- setUpdated(boolean value)

Decides if the value should receive its value from the update expression or not. May only be used to turn the updating on or off if the updating was on when the variable last was compiled.

3.2.4 String Variable

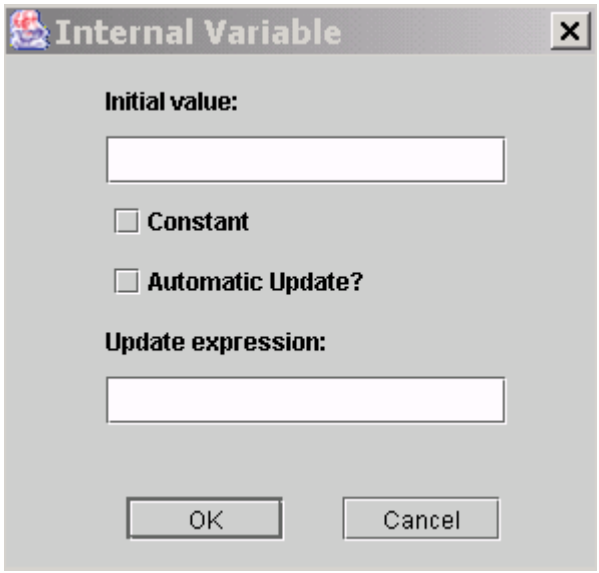
A string variable has a value that is represented by a Java String. A string variable can be used in a transition condition and in an action. By default, a variable retains its value when the application is stopped and later restarted. Optionally, the user may give a variable an initial value. If the variable has an initial value its value will be set to the initial value when the application is started,



A string variable can be declared as being constant. A constant variable may not be changed at run-time, neither from a JGrafchart application nor by the user. A variable that is declared constant is shown with a blue icon.

A string variable may have an associated update expression. In this case the variable automatically receives a new value in the beginning of every scan cycle through an evaluation of the update expression. The update expression will be evaluated as a string expression.

Double-clicking on the variable brings up an edit menu.



Initial value:
The initial value of the variable. An empty initial value field means that no initial value should be used.

Constant:
If selected the variable will be considered a constant which may not be written to.

Automatic Update?:
If selected the variable will automatically receive a new value in the beginning of every scan cycle through an evaluation of the update expression.

Update expression:
The automatic update expression for the variable.

The user can edit the value of the variable during execution.

Text Methods:

- setLocation(int x, int y)

- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- boolean isUpdated()

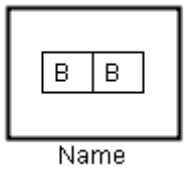
Returns true if the variable receives it value from the update expression.

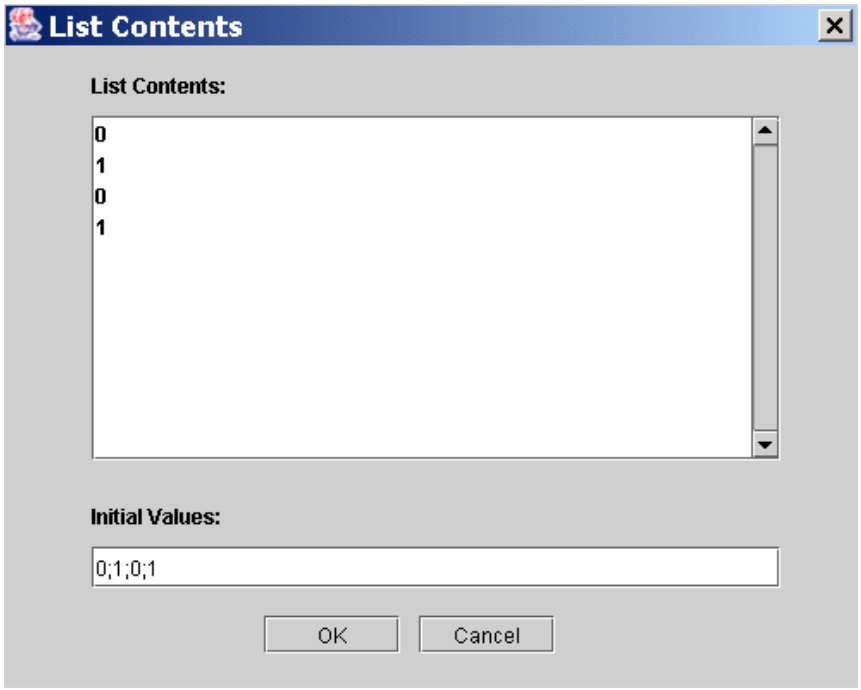
- setUpdated(boolean value)

Decides if the value shoul receive its value from the update expression or not. May only be used to turn the updating on or off, if the updating was on when the variable was last compiled.

3.2.5 Boolean List

A boolean list is a list that contains boolean values. When an application is started the list will be empty, unless an initial value has been defined for the list. Selecting the list brings up an edit window. In the edit window the current content of the list is shown and it is possible to enter the initial value for the list. The initial values are separated by semi-colons.





Text Methods:

- isEmpty()

Returns true if the list is empty and false if the list is non-empty.

- clear()

Empties the list.

- int size()

Returns the number of elements in the list.

- add(boolean val)

Adds an element at the end of the list.

- add(int position, boolean val)

Adds an element at the specified position in the list. The first element in the list corresponds to position 0.

- addFirst(boolean val)

Adds an element at the beginning of the list.

- boolean get(int position)

Returns the boolean value at the given position.

- int indexOf(boolean value)

Returns the index of the first occurrence of value in the list. Returns -1 if the value is not found.

- remove(int position)

Removes the value at the given position from the list.

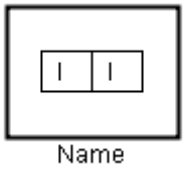
- set(int position, boolean value)

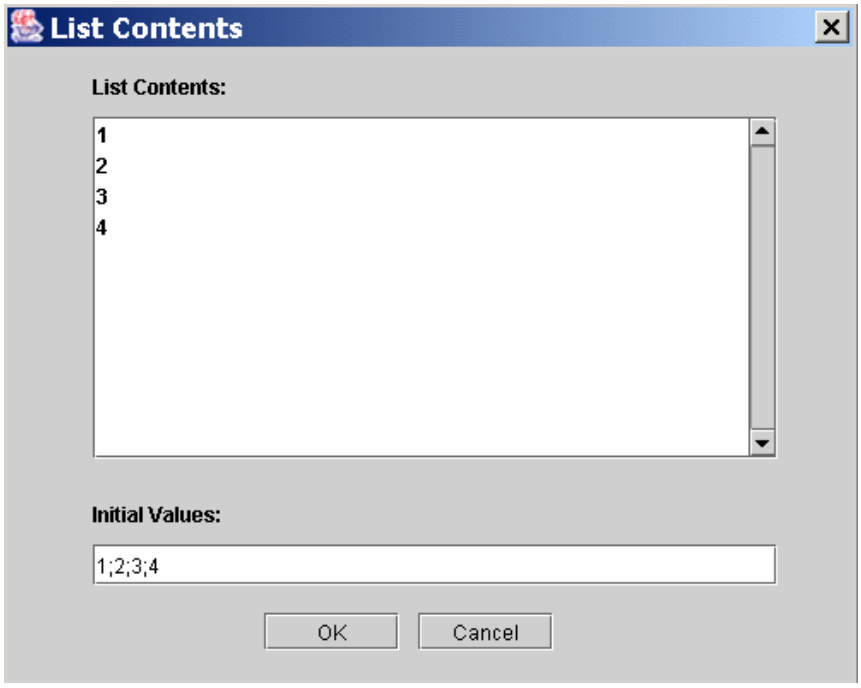
Sets a new value for the element at the given position.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.2.6 Integer List

An integer list is a list that contains integer values. When an application is started the list will be empty, unless an initial value has been defined for the list. Selecting the list brings up an edit window. In the edit window the current content of the list is shown and it is possible to enter the initial value for the list. The initial values are separated by semi-colons.





Text Methods:

- isEmpty()

Returns true if the list is empty and false if the list is non-empty.

- clear()

Empties the list.

- int size()

Returns the number of elements in the list.

- add(int val)

Adds an element at the end of the list.

- add(int position, int val)

Adds an element at the specified position in the list. The first element in the list corresponds to position 0.

- addFirst(int val)

Adds an element at the beginning of the list.

- int get(int position)

Returns the integer value at the given position.

- int indexOf(int value)

Returns the index of the first occurrence of value in the list. Returns -1 if the value is not found.

- remove(int position)

Removes the value at the given position from the list.

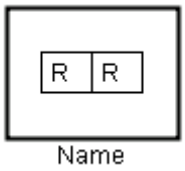
- set(int position, int value)

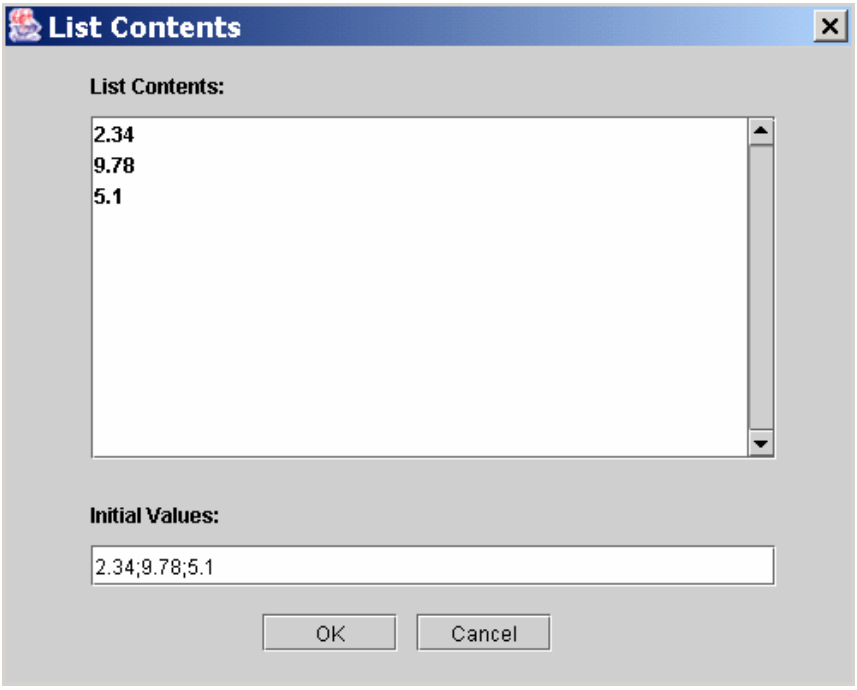
Sets a new value for the element at the given position.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.2.7 Real List

A real list is a list that contains real values. When an application is started the list will be empty, unless an initial value has been defined for the list. Selecting the list brings up an edit window. In the edit window the current content of the list is shown and it is possible to enter the initial value for the list. The initial values are separated by semi-colons.





Text Methods:

- isEmpty()

Returns true if the list is empty and false if the list is non-empty.

- clear()

Empties the list.

- int size()

Returns the number of elements in the list.

- add(real val)

Adds an element at the end of the list.

- add(int position, real val)

Adds an element at the specified position in the list. The first element in the list corresponds to position 0.

- addFirst(real val)

Adds an element at the beginning of the list.

- real get(int position)

Returns the integer value at the given position.

- int indexOf(real value)

Returns the index of the first occurrence of value in the list. Returns -1 if the value is not found.

- remove(int position)

Removes the value at the given position from the list.

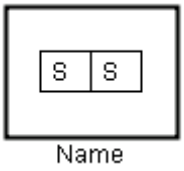
- set(int position, real value)

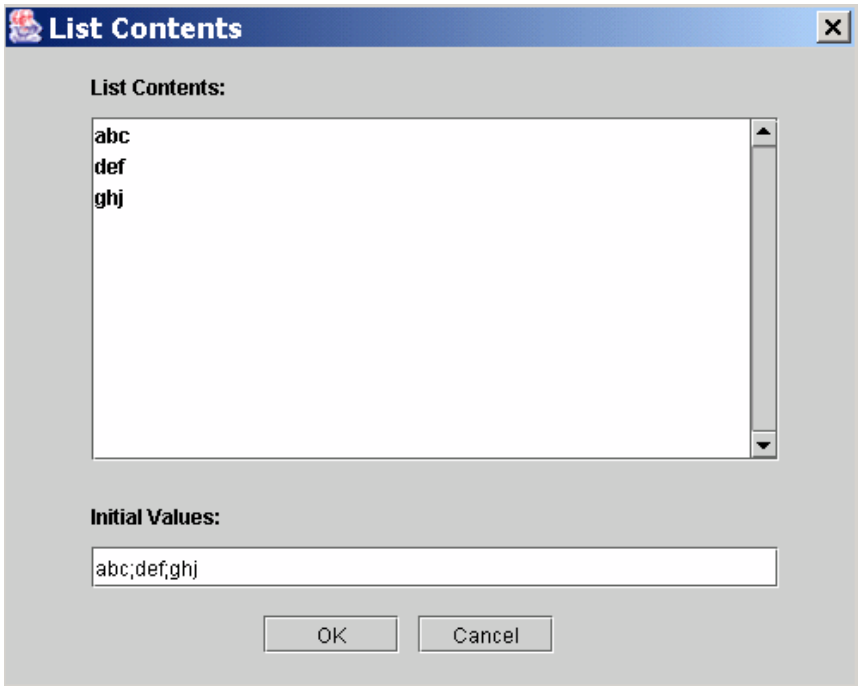
Sets a new value for the element at the given position.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.2.8 String List

A string list is a list that contains text strings. When an application is started the list will be empty, unless an initial value has been defined for the list. Selecting the list brings up an edit window. In the edit window the current content of the list is shown and it is possible to enter the initial value for the list. The initial values are separated by semi-colons.The values in the initial-value description may not themselves contain any semi-colons.





Text Methods:

- isEmpty()

Returns true if the list is empty and false if the list is non-empty.

- clear()

Empties the list.

- int size()

Returns the number of elements in the list.

- add(string val)

Adds an element at the end of the list.

- add(int position, string val)

Adds an element at the specified position in the list. The first element in the list corresponds to position 0.

- addFirst(string val)

Adds an element at the beginning of the list.

- string get(int position)

Returns the integer value at the given position.

- int indexOf(string value)

Returns the index of the first occurrence of value in the list. Returns -1 if the value is not found.

- remove(int position)

Removes the value at the given position from the list.

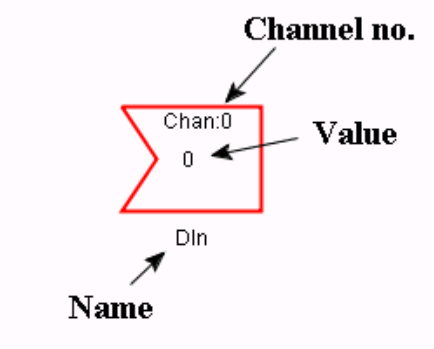
- set(int position, string value)

Sets a new value for the element at the given position.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3 Inputs and Outputs

3.3.1 Digital Input



A digital input is a boolean variable that is connected to a digital input channel. The digital input has a name and a channel number. The channel number should be an integer. The color of the icon is red when the value is false and green when the value is true. The value of the digital input can be changed on-line by the user using editing or by double-clicking on the icon (will cause a toggle of the binary value).

A digital input can be referenced from a transition or an action. It is not allowed to try to assign a new value to a digital input from an action. The digital input will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

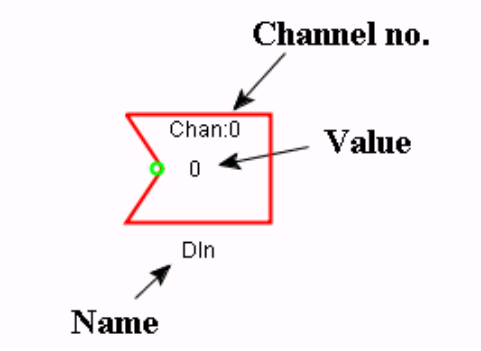
By default the digital input is associated with the Department of Automatic Control Java digital input channel class. It is possible for the user to attach the digital input object to a custom user-provided implementation of a digital input channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.2 Digital Input (Inverse Logic)

A digital input with inverse logic is a boolean variable that is connected to a digital input channel. The digital input has a name and a channel number. The channel number should be an integer. The digital input has the value 1 (0) when the value of the corresponding digital input channel is false (true). The color of the main part of the icon is red when the value is false and green when the value is true. The value of the digital input can be changed on-line by the user using editing or by double-clicking on the icon (will cause a toggle of the binary value).



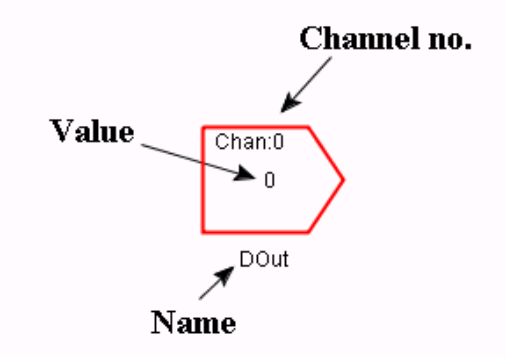
A digital input can be referenced from a transition or an action. It is not allowed to try to assign a new value to a digital input from an action. The digital input will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

By default the digital input is associated with the Department of Automatic Control Java digital input channel class. It is possible for the user to attach the digital input object to a custom user-provided implementation of a digital input channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.3 Digital Output



A digital output is a boolean variable that is connected to a digital output channel. The digital output has a name and a channel number. The channel number should be an integer. The color of the icon is red when the value is false, green when the value is true, and black if no value has been assigned to it. The value of the digital output may not be changed on-line by the user through editing or by double-clicking on the icon.

A digital output is assigned its value from an action. A digital output can also be referenced from a transition or an action. The digital output will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

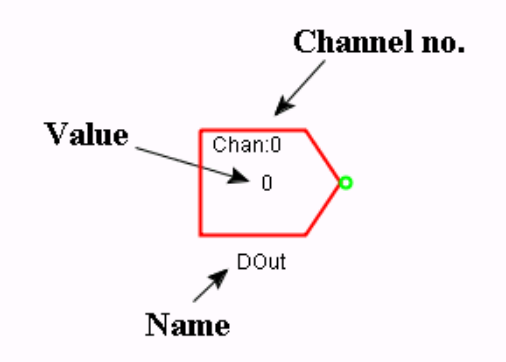
By default the digital output is associated with the Department of Automatic Control Java digital output channel class. It is possible for the user to attach the digital output object to a custom user-provided implementation of a digital output channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.4 Digital Output (Inverse Logic)

A digital output with inverse logic is a boolean variable that is connected to a digital output channel. The digital output has a name and a channel number. The channel number should be an integer. The digital output channel will be set to true (false) when the value of the digital output is 0 (1). The color of the main part of the icon is red when the value is false, green when the value is true, and black if no value has been assigned to it. The value of the digital output may not be changed on-line by the user through editing or by double-clicking on the icon.



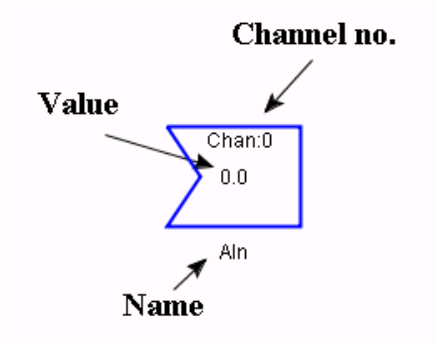
A digital output is assigned its value from an action. A digital output can also be referenced from a transition or an action. The digital output will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

By default the digital output is associated with the Department of Automatic Control Java digital output channel class. It is possible for the user to attach the digital output object to a custom user-provided implementation of a digital output channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.5 Analog Input



An analog input is a real variable that is connected to a analog input channel. The analog input has a name and a channel number. The channel number should be an integer. The value of the input can be changed on-line by the user using editing.

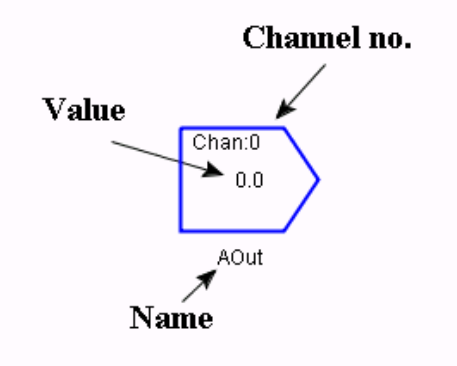
An analog input can be referenced from a transition or an action. It is not allowed to try to assign a new value to an analog input from an action. The analog input will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

By default the analog input is associated with the Department of Automatic Control Java analog input channel class. It is possible for the user to attach the analog input object to a custom user-provided implementation of a analog input channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.6 Analog Output



An analog output is a real variable that is connected to a digital output channel. The analog output has a name and a channel number. The channel number should be an integer. The value of the analog output may not be changed on-line by the user through editing or by double-clicking on the icon.

An analog output is assigned its value from an action. An analog output can also be referenced from a transition or an action. The analog output will only be connected to the external environment when the Simulator mode checkbox in the workspace properties is not set. By default this check box is set.

By default the analog output is associated with the Department of Automatic Control Java analog output channel class. It is possible for the user to attach the analog output object to a custom user-provided implementation of a analog output channel. In this way it is possible to connect JGrafchart to arbitrary IO cards, etc. The details for how to do this are found in Section 5.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7 Socket IO

Each top-level workspace can act as the client in a TCP socket connection with some user-defined TCP server. For each of the four basic data types there is one socket input variable type and one socket output variable type. For socket output variables a TCP message is being sent each time the variable receives a new value. Similarly, a socket input variable receives a new value when a matching TCP message is received.

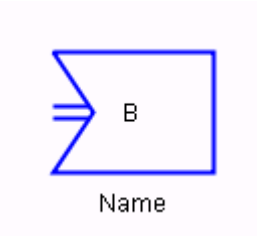
The TCP message uses a simple text protocol. A message consists of

"identifier" | "value"

The identifier is the local (non-qualified) name of the variable, unless a special identifier has been defined for the variable, in which case this identifier will be used. Since it is the local name that is used it is possible for several socket input variables to be updated as the result of one incoming TCP message.

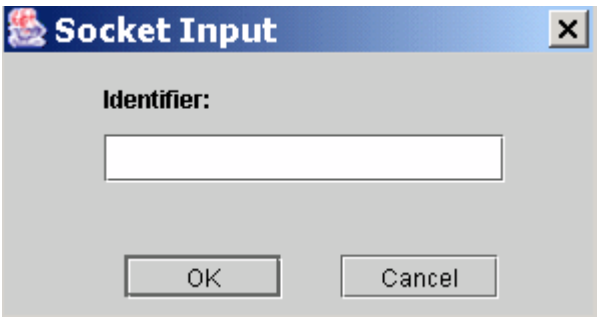
The host name (or URL) and the port of the TCP server is entered through the **Properties** menu of the top-level workspace. An empty socket name and a port number of -1 means that no socket connection will be established. An established connection remains open until the **Shutdown Client** menu choice is entered.

3.3.7.1 Boolean Socket Input



A boolean socket input variable is a boolean input variable that receives its value through a TCP message. The variable receives a new value when a matching TCP message is received. If the identifier of the variable has a value then the message identifier should equal this value. If not, the message identifier should equal the local name of the variable.

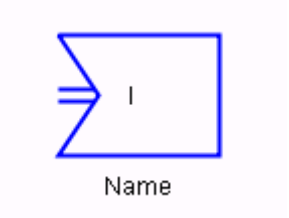
The identifier is set from the edit-menu of the variable.



Text Methods:

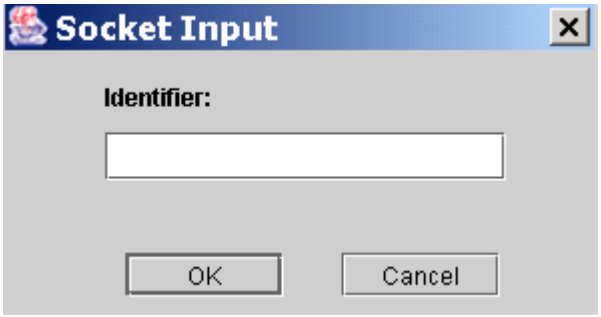
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.2 Integer Socket Input



An integer socket input variable is an integer input variable that receives its value through a TCP message. The variable receives a new value when a matching TCP message is received. If the identifier of the variable has a value then the message identifier should equal this value. If not, the message identifier should equal the local name of the variable.

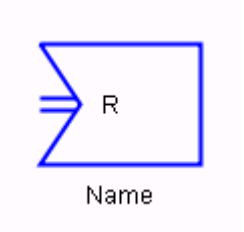
The identifier is set from the edit-menu of the variable.



Text Methods:

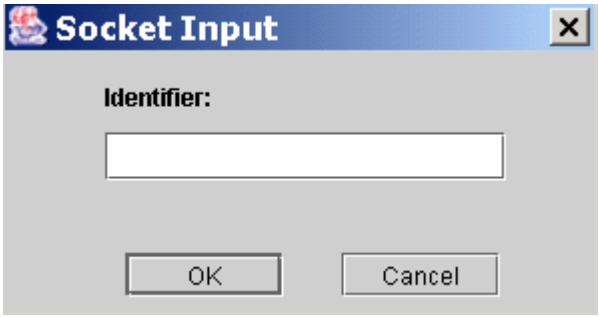
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.3 Real Socket Input



A real socket input variable is a real input variable that receives its value through a TCP message. The variable receives a new value when a matching TCP message is received. If the identifier of the variable has a value then the message identifier should equal this value. If not, the message identifier should equal the local name of the variable.

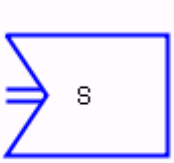
The identifier is set from the edit-menu of the variable.



Text Methods:

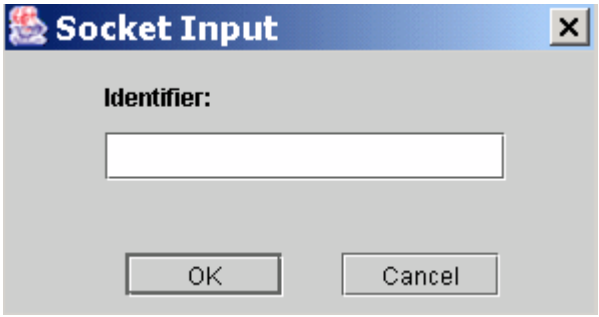
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.4 String Socket Input



A string socket input variable is a string input variable that receives its value through a TCP message. The variable receives a new value when a matching TCP message is received. If the identifier of the variable has a value then the message identifier should equal this value. If not, the message identifier should equal the local name of the variable.

The identifier is set from the edit-menu of the variable.

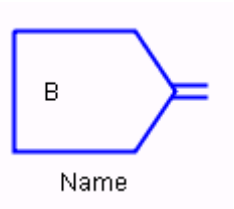


Text Methods:

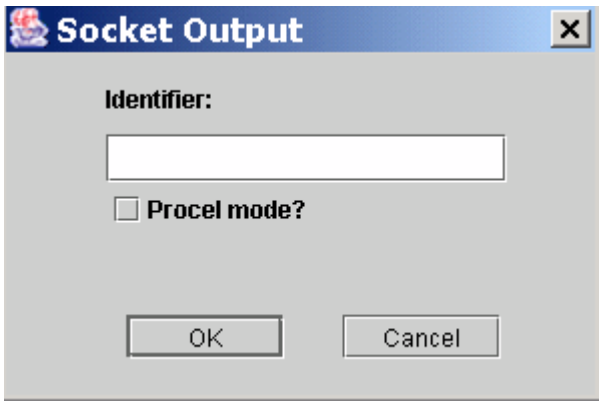
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.5 Boolean Socket Output

A boolean socket output variable is a boolean output variable that is connected to a TCP socket. A TCP message is being sent each time the variable receives a new value. If the identifier of the variable has a value then this value is used as message identifier. Otherwise, the local (unqualified) name of the variable is used as message identifier.



The identifier is set from the edit-menu of the variable.

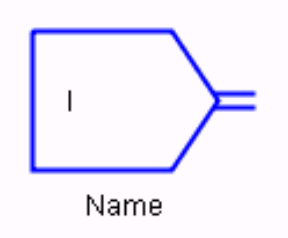


The Procel check box is for a special purpose IO and should be ignored.

Text Methods:

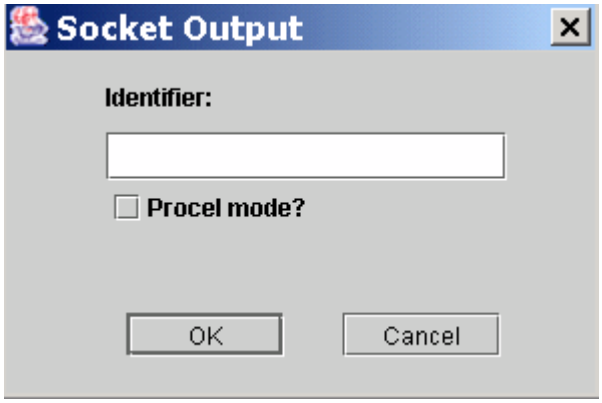
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.6 Integer Socket Output



An integer socket output variable is an integer output variable that is connected to a TCP socket. A TCP message is being sent each time the variable receives a new value. If the identifier of the variable has a value then this value is used as message identifier. Otherwise, the local (unqualified) name of the variable is used as message identifier.

The identifier is set from the edit-menu of the variable.

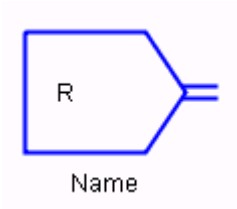


The Procel check box is for a special purpose IO and should be ignored.

Text Methods:

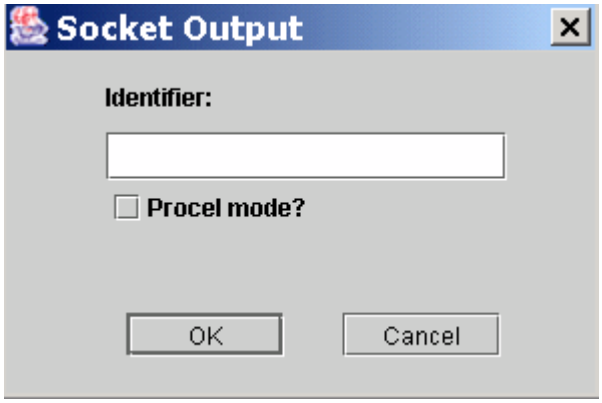
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.7 Real Socket Output



A real socket output variable is a real output variable that is connected to a TCP socket. A TCP message is being sent each time the variable receives a new value. If the identifier of the variable has a value then this value is used as message identifier. Otherwise, the local (unqualified) name of the variable is used as message identifier.

The identifier is set from the edit-menu of the variable.

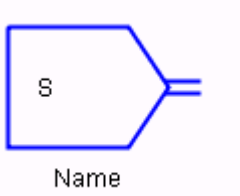


The Procel check box is for a special purpose IO and should be ignored.

Text Methods:

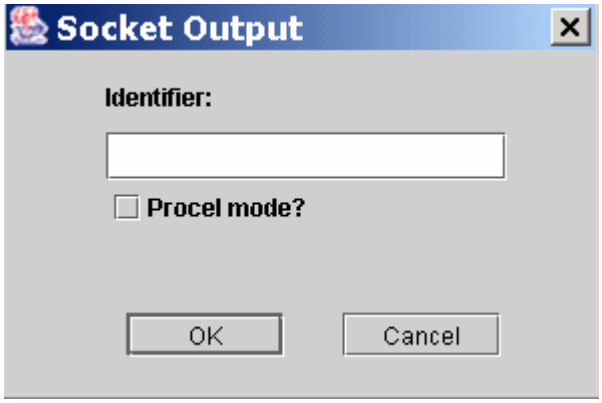
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.3.7.8 String Socket Output



A string socket output variable is a string output variable that is connected to a TCP socket. A TCP message is being sent each time the variable receives a new value. If the identifier of the variable has a value then this value is used as message identifier. Otherwise, the local (unqualified) name of the variable is used as message identifier.

The identifier is set from the edit-menu of the variable.



The Procel check box is for a special purpose IO and should be ignored.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

3.4 CHEM and CCOM

JGrafchart has been developed within the EU/GROWTH FP5 project CHEM - Advanced Decision Systems for the Chemical/Petrochemical Manufacturing Industries. Within CHEM a large number of operator support toolboxes for different tasks (e.g., alarm analysis, fault detection, diagnosis, planning) are being developed. The toolboxes are implemented in different languages and run on different platforms. They are integrated through the CCOM infrastructure.

CCOM (Chem Communications Manager) allows toolboxes to communicate using message passing. It is based on the message-oriented middleware xmlBlaster. The messages are represented by XML structures. CCOM supports publish/subscribe and point-to-point communication. The messages can be of different type, e.g., data messages, event messages, RPC invocation messages, and RPC response messages. The CHEM information model establishes a common syntax for data messages.

3.4.1 Representing XML structures

XML structures are represented by XML Message objects. XML Message In objects are used to receive XML messages and XML Message Out objects are used to send XML messages. The XML Message objects are hierarchically structured objects similar to workspace objects. They may only contain internal variables and workspace objects.

The identifier of the XML Message object determines the tag of the main XML element. If the identifier has no value the name of the XML Message object is used instead. Internal variables within the XML Message object represent either simple XML elements or XML attributes. The name of the internal variable maps to the element or attribute name and the value of the internal variable maps to the value of the element or attribute. Workspace objects within the XML Message object represent structured XML elements (elements that contain child elements and/or attributes). The name of the workspace objects maps to the name of the structured element.
The top-down order of the objects on the subworkspace of the XML Message object determines the order of the elements and attributes in the XML structure.

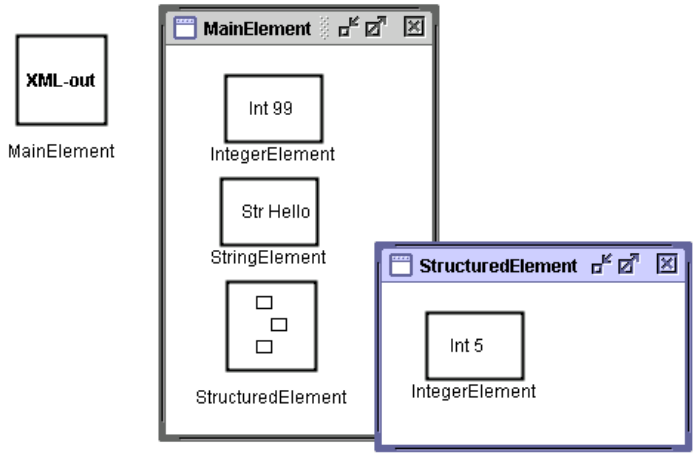
Example: *Representing XML Structures*

An example showing an XML structure and its corresponding XML Message represented is shown below. Here it is the name of the XML Message Out object that corresponds to the main element tag name.

XML Structure:

```
<MainElement>
  <IntegerElement>99</IntegerElement>
  <StringElement>Hello</StringElement>
  <StructuredElement>
    <IntegerElement>5</IntegerElement>
  </StructuredElement>
</MainElement>
```

XML Message representation:



In order to indicate that an internal variable should represent an XML attribute rather than a simple XML element, special Attribute Variable objects are used. The attribute variable objects are subclasses of the corresponding internal variable class.

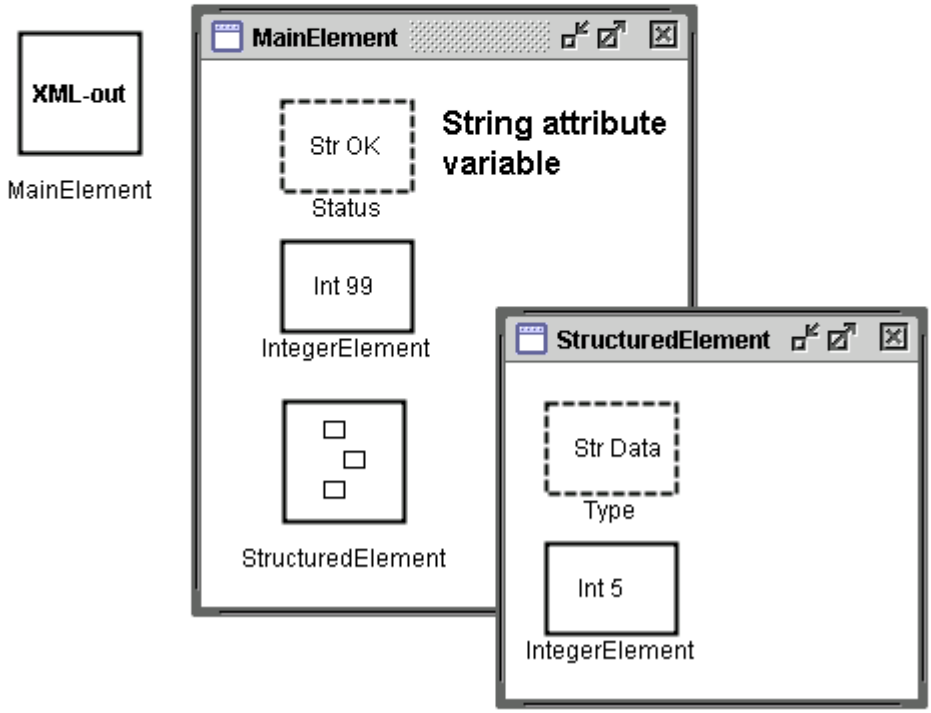
Example: *Representing XML Structures with Attributes*

This example contains both elements and attributes.

XML Structure:

```
<MainElement Status="OK">
  <IntegerElement>99</IntegerElement>
  <StructuredElement Type="Data">
    <IntegerElement>5</IntegerElement>
  </StructuredElement>
</MainElement>
```

XML Message representation:



XML structures that contain multiple occurrences of elements with the same tag name are represented in a special way. Special index variables (a subclass of integer variable) are used to represent the multiplicity of the element. The name of the index variable maps to the tag name of the multiple element. The value of the index variable determines the number of occurrences of the multiple element. The individual occurrences of the multiple element are represented as ordinary elements (using internal variables or workspace objects). They should be named "element name" + "integer index", where the integer index goes from 1 up to the number of occurrences of the multiple element.

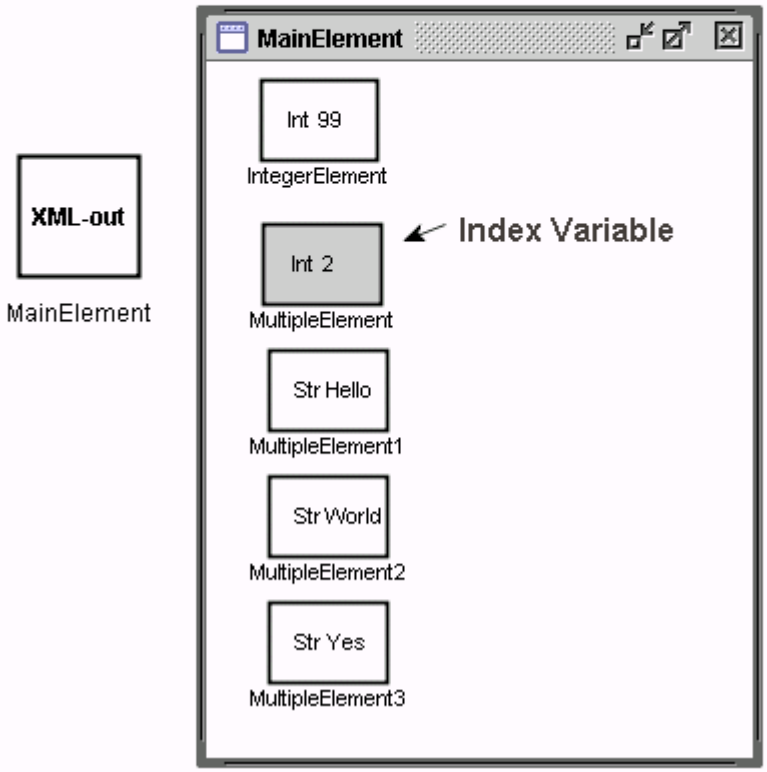
Example: *Representing XML structures containing Multiple Elements*

In the following example multiple occurrences of the element MultipleElement exist.

XML Structure:

```
<MainElement>
  <IntegerElement>99</IntegerElement>
  <MultipleElement>Hello</MultipleElement>
  <MultipleElement>World</MultipleElement>
</MainElement>
```

XML Message representation:



Since the index variable MultipleElement has the value 2 only two occurrences of the MultipleElement element will be present in the XML structure that is created. Similarly, if an XML structure is received that may contain multiple occurrences of MultipleElement, then the value of the index variable will indicate how many occurrences that were present in the actual message that was received. The number of individual instances of a multiple element (in this case MultipleElement1, MultipleElement2, MultipleElement3) must be equal to the maximum number of occurrences (worst-case).

3.4.2 Connecting to CCOM

In order to use the CCOM XML messaging service, JGrafchart must first be connected to CCOM. JGrafchart can be connected to CCOM in two different ways:

1. Manual CCOM Connection
   Here the server properties (Server IP address, server port, and login name) are entered manually using the CCOM Server menu action in the Misc menu. Once this is done the connection is established using the Connect CCOM menu action. JGrafchart is disconnected from the server using the menu action Disconnect CCOM.
2. Automatic CCOM Connection.
   JGrafchart is automatically connected to CCOM if it is started using the command line options

-connect 1 -serverIP "xyz.zyx.zyx.zyx" -port 3412 -name JGrafchart. The default value for port is 3412 and for name (the login name) is JGrafchart. JGrafchart is automatically disconnected from CCOM when it is terminated.

3.4.3 Publishing Data Messages

The message to be published is represented by an XML Message Out object. Attributes of the XML Message Out object are used to decide the topic under which the data message should be published, the CCOM type of the data message (Data or Event), the origin of the publication (i.e., JGrafchart), etc.

To publish a data message the following steps are performed:

1. The content of the message is built up using ordinary actions, For example, the values of the different elements and attributes within the XML Message out object are set using assignment actions.
2. The necessary XML Message out attributes are set either interactively or programmatically.
3. The message is published by calling the method publishMessage() of the XML Message Out object. When this is done an XML Message structure is created from the XML Message Out object. The XML Message is embedded in an CCOM envelope and then send to the CCOM server.

3.4.4 Sending Data Messages

The message to be sent is represented by an XML Message Out object. Attributes of the XML Message Out object are used to decide the destination of the message, the CCOM type of the data message (Data or Event), the origin of the publication (i.e., JGrafchart), etc.

To send a data message the following steps are performed:

1. The content of the message is built up using ordinary actions, For example, the values of the different elements and attributes within the XML Message out object are set using assignment actions.
2. The necessary XML Message out attributes are set either interactively or programmatically.
3. The message is sent by calling the method sendMessage() of the XML Message Out object. When this is done an XML Message structure is created from the XML Message Out object. The XML Message is embedded in an CCOM envelope and then send to the CCOM server.

3.4.5 Subscriptions

In order to receive messages that published under a certain topic the toolbox must first have subscribed to the topic. Subscription and unsubscription can be made in two different ways:

1. Implicit subscription
   When the application is started an automatic subscription is made to the topics of all the XML Message In objects of the application. Similarly, when the application is stopped an automatic unsubscription is made to the topics of all the XML Message Out objects in the application. Implicit subscription is always activated.
2. Explicit subscription.
   A topic is subscribed to using the function subscribe(aTopic). Similarly a topic is unsubscribed to using the function unsubscribe(aTopic)

3.4.6 Receiving Data Messages

All XML Messages are received using XML Message In objects. An XML Message In object will receive an XML Message if the application is executing and the identifier of the XML Message In object is equal to the main tag name of the message. (If the identifier has no value the local name of

the XML Message In is used instead). A special case exists for messages with the main tag name ChemContentMessage. If these messages contain a Subject element they will only be received in an XML Message In object if the value of the subject attribute of the XML Message In is equal to the value of the Subject element.

Multiple XML Message In objects may (and most likely will) receive the same message. When an XML Message In object is received two things occur:

1. All variables inside the XML Message In object are initialised to their initial values (if they have any initial values).
2. The XML structure in the message is decoded (using the DOM XML parser) and mapped to the corresponding element/attribute variables within the XML Message In object. The mapping is done according to best effort. This means that elements in the message for which there are no corresponding element/attribute variables are ignored. It also means that element/attribute variables for which there are no corresponding XML elements/attributes will not be assigned any new value. The initialisation of all variables is used to be able to detect if a variable really has been assigned a new value from the message.

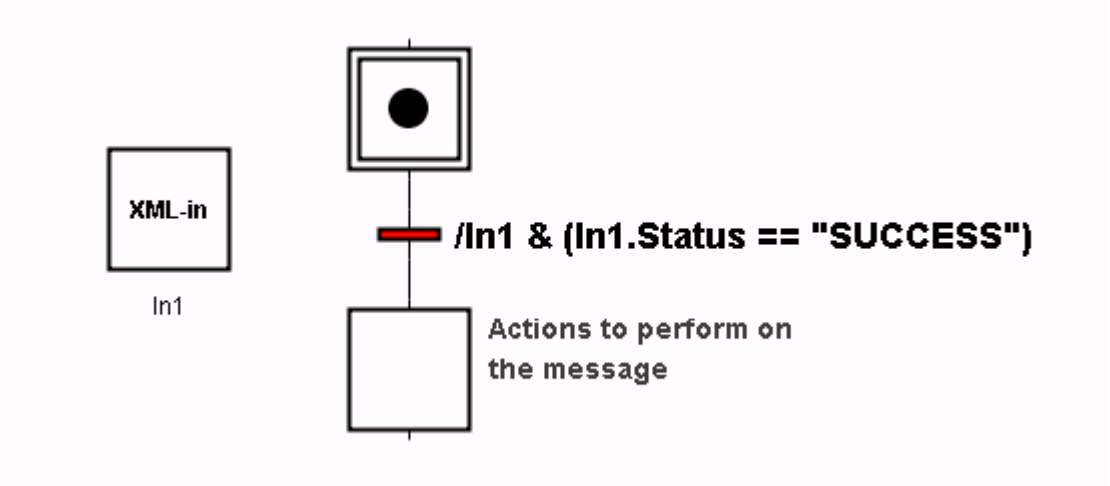
3.4.7 Detecting Message Arrival

The reception of an XML Message by an XML Message In object can be detected in two different ways:

1. Using an event transition expression
   The event expression /In1 is true whenever a new message is received by the XML Message In object named In1. The main advantage with is approach is that it is fast and simple. The disadvantage is that it only works if the transition is enabled. If this is not the case there is a risk that the message is lost due to overwriting by a new message. Also, the approach currently does not work for XML Message In objects that are contained within procedures.
2. Using a procedure.
   It is possible to associate a procedure with an XML Message In object. Whenever a message is received a new call to the procedure is spawned (similar to a process step). The entire contents of the message can be passed in as a call-by-value parameter to the procedure. The advantage of this approach is that no messages may be lost. The disadvantage is that the spawning of a procedure is quite time consuming.

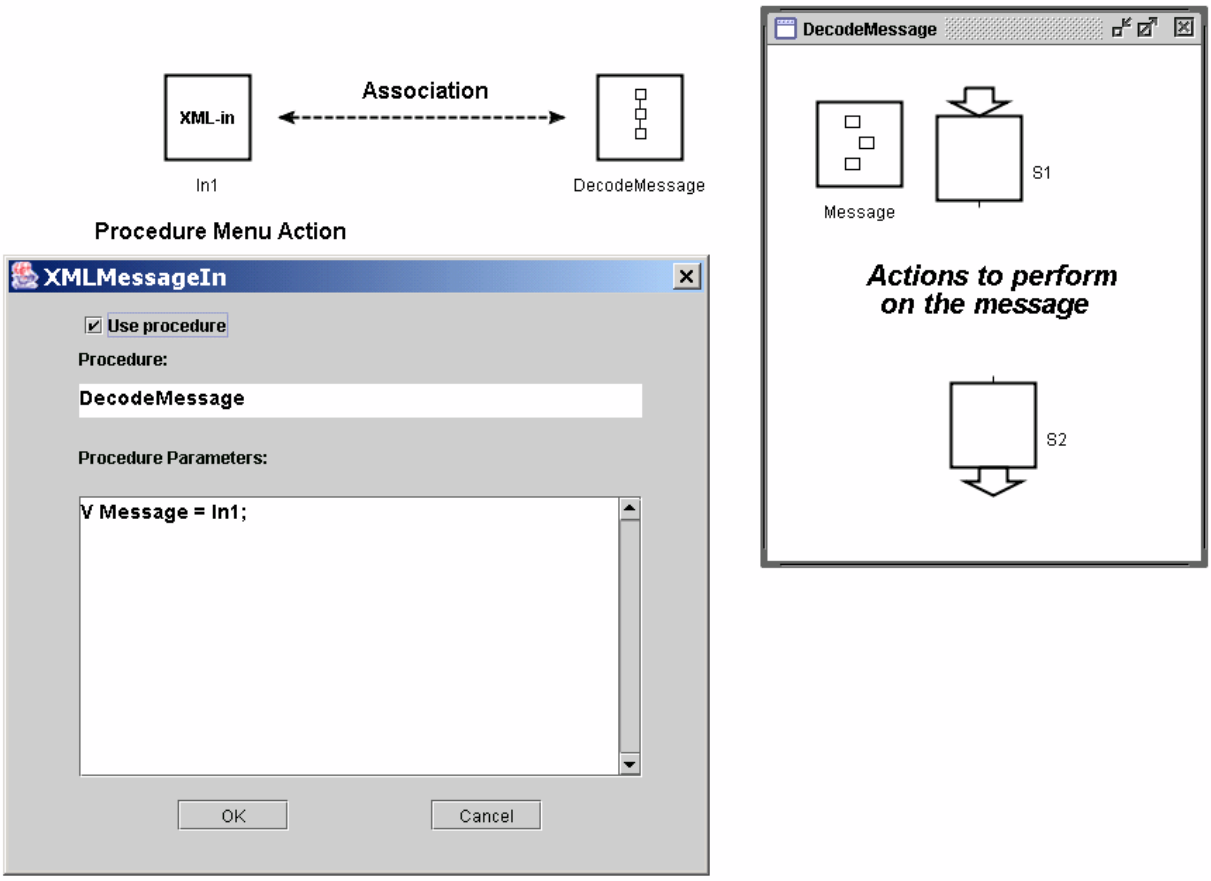
Example: *Detecting Message Arrival Using Transitions*

An example of how the arrival of a message is detected using an event expression is shown below.



Example: *Detecting Message Arrival Using a Procedure*

An example of how the arrival of a message is detected using a procedure is shown below:



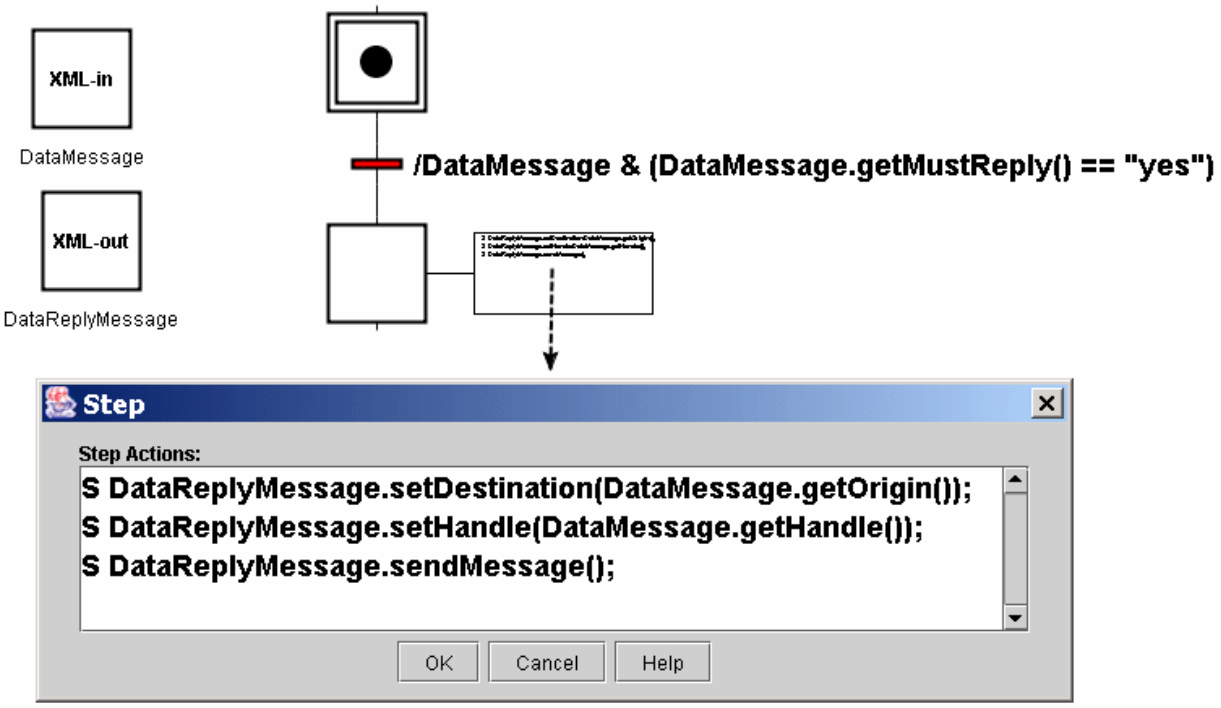
In the Procedure Menu action dialog the user indicates which procedure that should be associated with the XML Message In object. The entire object is passed in to the procedure invocation (V Message = In1;) Here it is important that the object at the left hand side in the assignment is a workspace object and not an XML Message In object.

3.4.8 Sending Reply Messages

When a message is received for which the MustReply attributes has the value "yes" then a reply message is required. The handle attribute of the reply message should be set to the value of the handle of the received message.

Example: *Sending a Reply Message*

An example of how a reply message is generated is shown below:

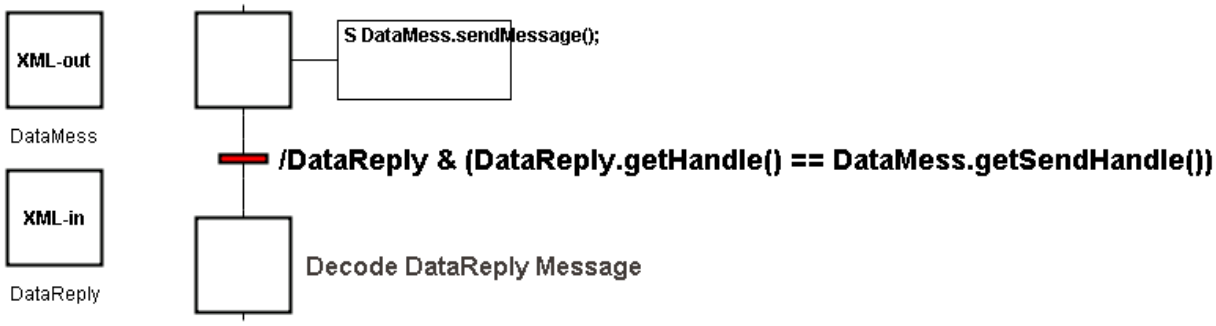


3.4.9 Receiving Message Replies

In many situations it is necessary to be able to match an incoming reply message with the original data request message. This can be done with the help of the getSendHandle() method of the XML Message Out object. The method returns the handle that was associated with the message being sent by CCOM. This handle will be a part of the reply message and can there be used to detect the correct reply message (the one that corresponds to the data request message).

Example: *Receiving a Reply Message*

An example of how a reply message can be replied and identified is shown below:



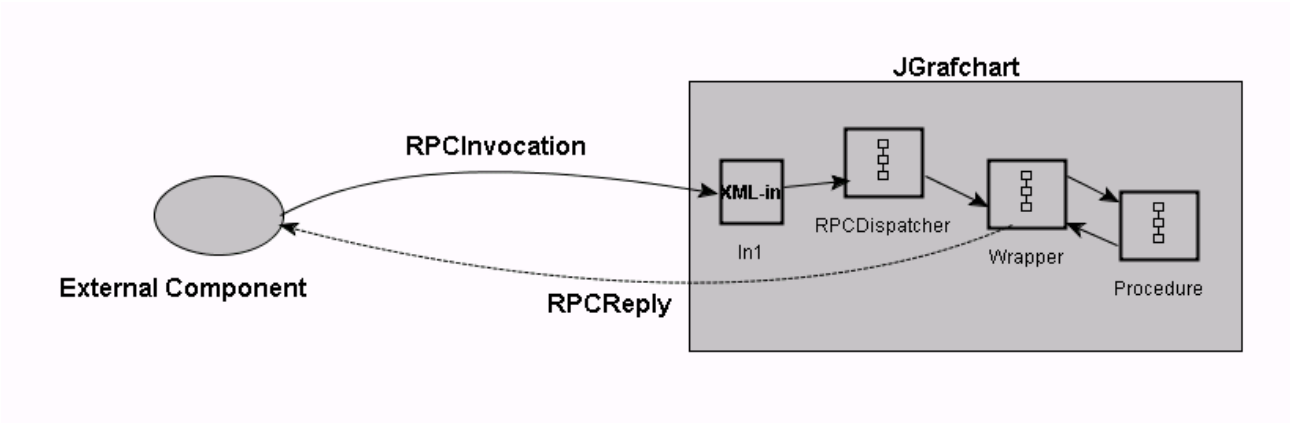
The DataReply message will only be decoded if it matches the DataMess message being sent.

3.4.10 Remote Procedure Calls

Sending RPC requests (invocations) and receiving RPC responses are done according to the same principles as for ordinary data messages.

3.4.11 External Calls to JGrafchart Procedures

Calls from external components or toolboxes to JGrafchart procedures are handled as in the figure below.

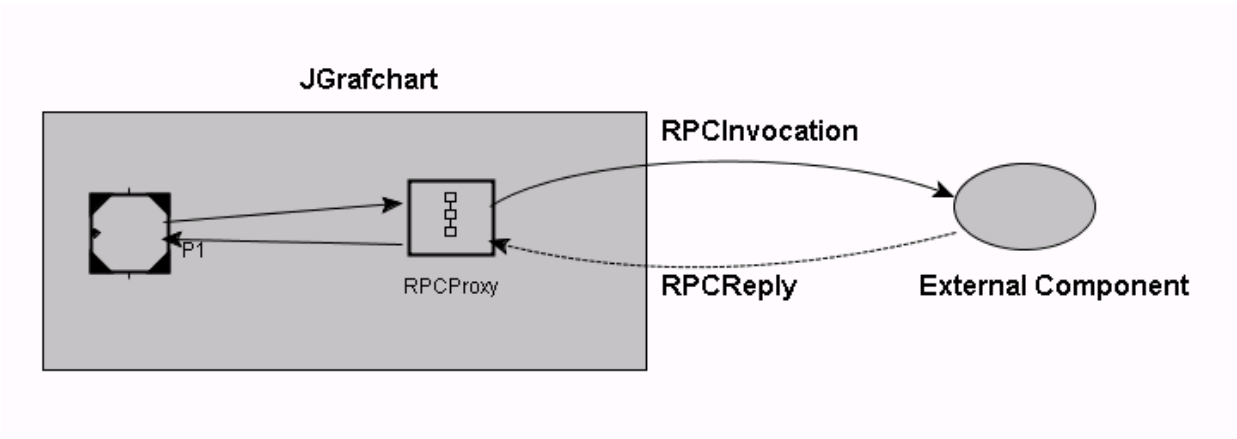


An XML Message In message is used to receive the RPC invocation message. The identifier/name of the XML Message In object should be Rpc. The arrival detection is implemented by a separate procedure associated with the XML Message In object (here named RPCDispatcher). The RPCDispatcher may be common to all RPCs. The RPCDispatcher calls a wrapper procedure associated with the JGrafchart procedure, e.g., by adding the "Wrapper" prefix to the procedure name.. The wrapper procedure extracts the procedure parameters from the message, calls the actual procedure, collects the out parameters, and generates an RPCResponse message that is sent back to the external caller.

Complete examples of external calls to JGrafchart procedures can be found in the on-line tutorial.

3.4.12 Calling External RPCs

When JGrafchart should call an RPC in another component/toolbox it is convenient to use a Grafchart procedure as a proxy for the remote procedure according to the example below.



The proxy procedure, which may have the same name as the external procedure, generates an RPCInvocation message using a local XML Message Out object, awaits the RPCResponse message using a local XML Message In object, extracts the reply parameters, and returns the reply to the context of the procedure step.

Complete examples of calling external procedures from JGrafchart can be found in the on-line tutorial.

3.4.13 CCOM RPC Libraries

RPC libraries with proxy procedures for the different CCOM components can be found in the grafchart/CCOM folder. To use them you need to load them and compile them. The procedures can then be called from ordinary procedure steps.

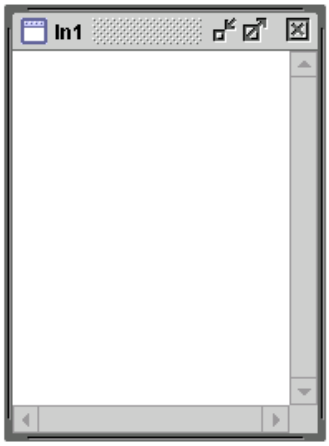
Currently the only libraries supported are the DTM Library and the Generic RTM Library.

3.5 CCOM XML Object Reference

3.5.1 XML Message In

In1

Subworkspace



An XML Message In object is an hierarchical object that is used to represent received XML Messages. It contains a subworkspace on which it is only allowed to place internal variables and workspace objects. The internal variables represent simple XML elements or XML attributes. The workspace objects represent structured XML elements.

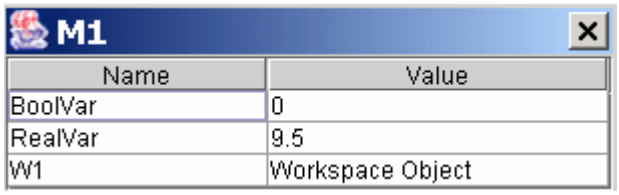
Menu Choices:

- **Show/Hide Body**

  A toggle menu choice that shows the subworkspace of the XML Message In object if it is hidden and hides the subworkspace if it is shown. The subworkspace remembers its actual position, size, and scale factor when it is hidden.

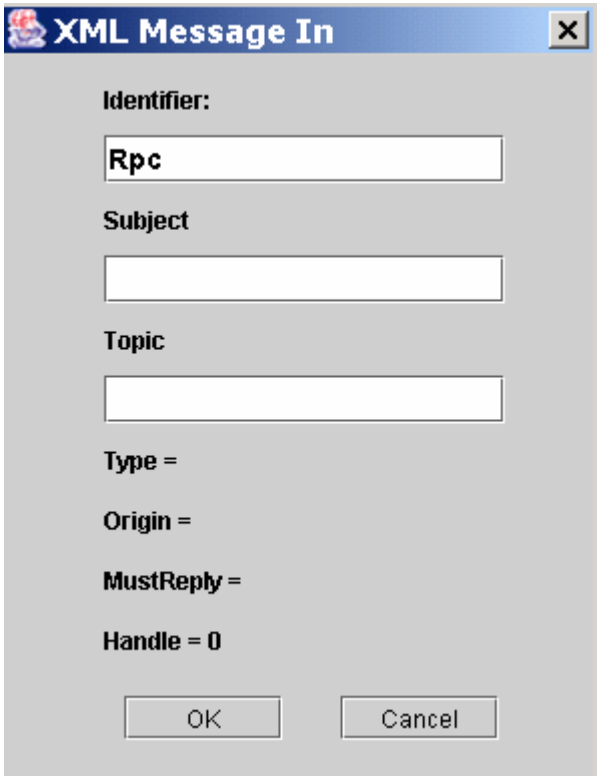
- **Table**

  Brings up a table containing all internal variables, analog inputs and outputs, digital inputs and outputs, and all hierarchical elements (macro steps, workspace objects, XML Message In, XML Message Out, procedures) found inside the XML Message In. By double clicking on the value of an internal variable or input/output it is possible to change the current value of the object. This is, however, only possible if the workspace that the object resides in is not locked, and, in the case the object is an internal variable, the internal variable has not been declared constant. By double-clicking on the value column for a hierarchical object, a new table is shown contain the internals of the hierarchical object. The variables values in the table are dynamically updated. The table is shown below:



- **Edit**

  Brings up an edit dialog



Identifier:
Determines which XML messages that will be received by the XML Message In object. If it has a value then only messages with this as its main element tag will be received. If the identifier has no value then the local name of the object will be used instead.

Subject:
If the value of the identifier is ChemContentMessage then Subject acts as a secondary reception filter for incoming messages. Only messages that contain a Subject element with a corresponding value will be received by the XML Message In object.

Topic:
If topic has a value then JGrafchart will automatically subscribe to this topic when the application is started. and unsubscribe to this topic when the application is stopped.

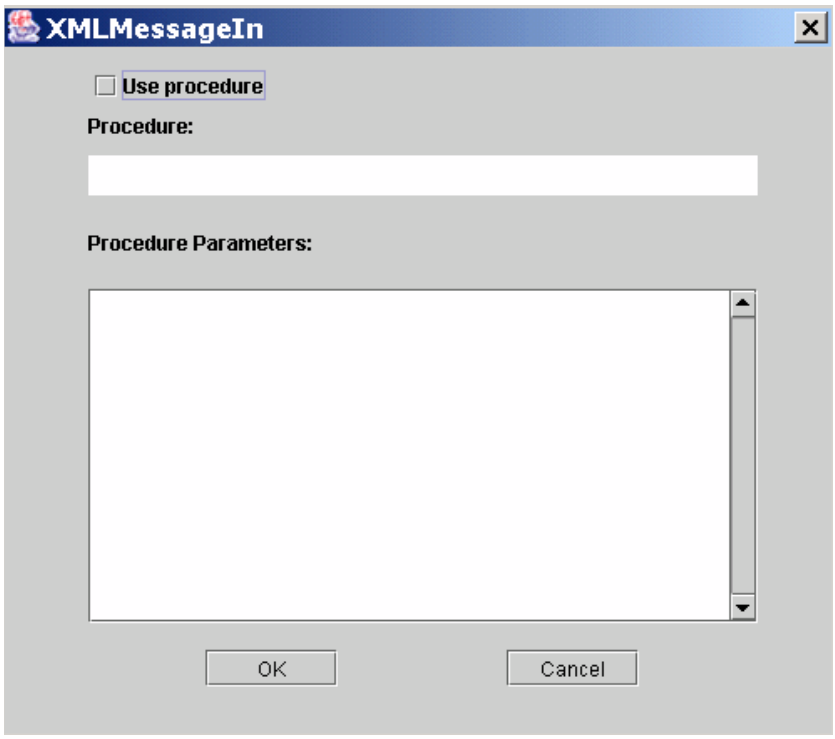
Type:
The CCOM type of the most recently received message.

Origin:
The origin (sender) of the most recently received message.

MustReply:
If this has the value "yes" then the most recently received message expects a reply message.

Handle:
The handle of the most recently received message.

- **Procedure**
  Brings up an edit dialog for information about the detection procedure associated with the XML; Message In object



Use procedure:
A checkbox that decides if a procedure call should be spawned each time a new message is received. Default = no.

Procedure:
The procedure that will be spawned when the message is received.

Procedure parameters:
The parameters to the procedure. It is common to pass in the entire XML Message In object as a parameter to a local workspace object within the procedure. In this way the content of the received message is saved and may not be overwritten by later messages.

- **Calls**

  Brings up a menu showing all currently executing invocations of the procedure. By selecting one of them the body of that particular procedure call is shown.

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- setWorkspaceColor(int color)

  Set the color of the subworkspace of the workspace object.

- real getWorkspaceColor()

Returns the color of the subworkspace of the workspace object.

- showWorkspace()

Shows the subworkspace.

- hideWorkspace()

Hides the workspace.

- String getType()

Returns the CCOM type of the most recently received message.

- String getOrigin()

Returns the origin (the sender) of the most recently received message.

- String getHandle()

Returns the handle of the most recently received message.

- String getMustReply()

Returns the value of the MustReply attribute of the most recently received message.

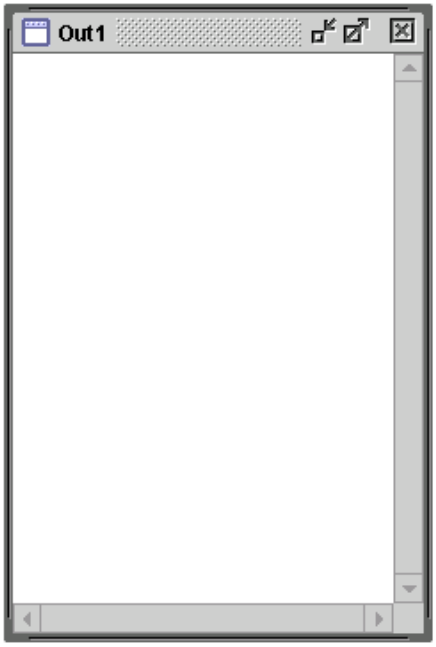
- String getXML()

Returns the XML string of the most recently received message including the CCOM envelope.

3.5.2 XML Message Out

Out1

Subworkspace



An XML Message Out object is an hierarchical object that is used to represent sent XML Messages. It contains a subworkspace on which it is only allowed to place internal variables and workspace objects. The internal variables represent simple XML elements or XML attributes. The workspace objects represent structured XML elements.

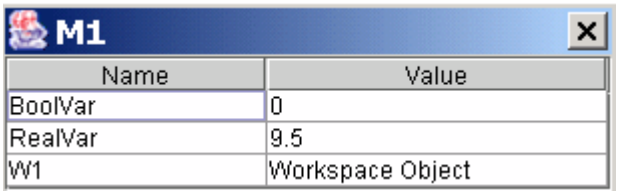
Menu Choices:

- **Show/Hide Body**

  A toggle menu choice that shows the subworkspace of the XML Message Out object if it is hidden and hides the subworkspace if it is shown. The subworkspace remembers its actual position, size, and scale factor when it is hidden.

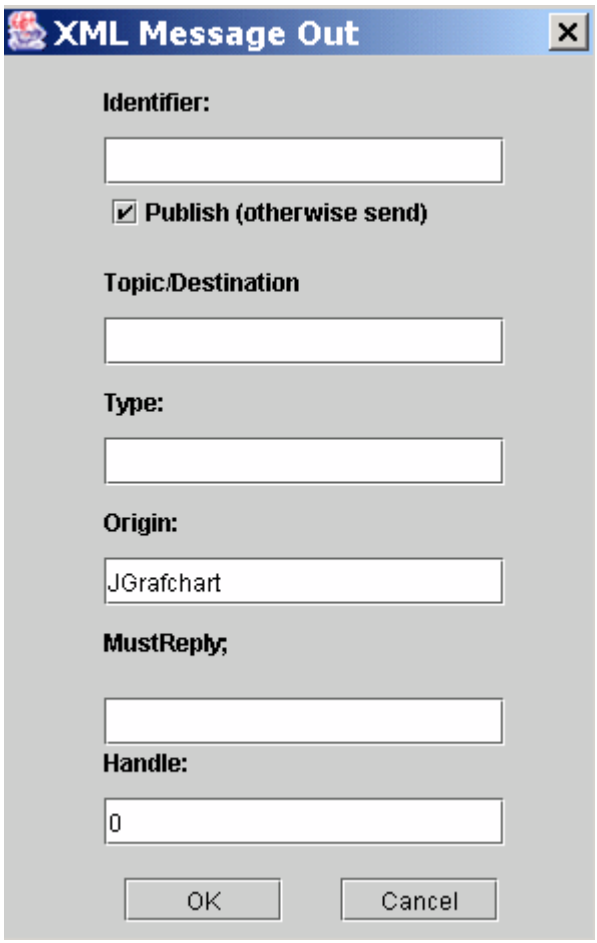
- **Table**

  Brings up a table containing all internal variables, analog inputs and outputs, digital inputs and outputs, and all hierarchical elements (macro steps, workspace objects, XML Message In, XML Message Out, procedures) found inside the XML Message Out. By double clicking on the value of an internal variable or input/output it is possible to change the current value of the object. This is, however, only possible if the workspace that the object resides in is not locked, and, in the case the object is an internal variable, the internal variable has not been declared constant. By double-clicking on the value column for a hierarchical object, a new table is shown contain the internals of the hierarchical object. The variables values in the table are dynamically updated. The table is shown below:



- **Edit**

  Brings up an edit dialog



Identifier:
Determines the tag of the main element. If it has no value then the local name is used instead.

Publish (otherwise send):
A checkbox that decides if the message should be published or sent when the method outputMessage() is called. Default = publish.

Topic/Destination:
The value of this field is used as the topic when the message is published and as the destination when the message is sent. A message can only be sent to a single destination at a time.

Type:
The CCOM type of the message.

Origin:
The origin of the message. Should normally be the same as the CCOM login name, i.e., JGrafchart.

MustReply:
If this has the value "yes" then the receiver of the message is required to send a response. Otherwise it should have the value "no".

Handle:
If the message is a reply message then this attribute should contain the handle of the original message. A value of 0 means that the message is not a reply message (Default = 0).

Text Methods:

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- setWorkspaceColor(int color)

Set the color of the subworkspace of the workspace object.

- real getWorkspaceColor()

Returns the color of the subworkspace of the workspace object.

- showWorkspace()

Shows the subworkspace.

- hideWorkspace()

Hides the workspace.

- String getType()

Returns value of the Type attribute.

- void setType(String value)

Sets the value of the Type attribute.

- int getHandle()

Returns the value of the Handle attribute.

- void setHandle(int value)

Sets the value of the Handle attribute.

- String getMustReply()

Returns the value of the MustReply attribute.

- void setMustReply(String value)

Sets the value of the MustReply attribute.

- String getOrigin()

Returns the value of the Origin attribute.

- void setOrigin(String value)

Sets the value of the Origin attribute.

- boolean getPublish()

Returns the value publish/send attribute. True means publish.

- void setPublish(boolean value)

Sets the value of the publish/send attribute.

- String getDestination()

Returns the value of the topic/destination attribute.

- void setDestination(String value)

Sets the value of the topic/destination attribute.

- String getTopic()

Returns the value of the topic/destination attribute.The same as getDestination()

- void setTopic(String value)

Sets the value of the topic/destination attribute. The same as setDestination(String value).

- void publishMessage()

Publishes the message under the topic given by the topic/destination attribute.

- void sendMessage()

Sends the message to the destination given by the topic/destination attribute.

- void outputMessage()

Equivalent either to sendMessage or publishMessage depending on the value of the publish attribute.

- String getXML()

Returns the XML message generated from the XML Message Out object excluding the CCOM envelope.

- int getSendHandle()

Returns the handle assigned to the most recent message being sent.

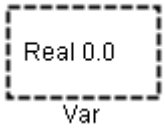
3.5.3 Integer Attribute

A subclass of integer variable that is used within XML Message In & Out objects to indicate that the variable should be represented as an XML attribute rather than an XML element.

Integer Attribute has the same menu choices and text methods as Integer Variable.

3.5.4 Real Attribute

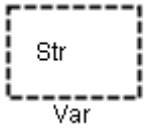
A subclass of real variable that is used within XML Message In & Out objects to indicate that the variable should be represented as an XML attribute rather than an XML element.



Real Attribute has the same menu choices and text methods as Real Variable

3.5.5 String Attribute

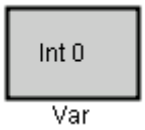
A subclass of string variable that is used within XML Message In & Out objects to indicate that the variable should be represented as an XML attribute rather than an XML element.



String Attribute has the same menu choices and text methods as String Variable.

3.5.6 Index Variable

A subclass of integer variable that is used within XML Message In & Out objects to count the number of occurrences of multiple elements.



Index Variable has the same menu choices and text methods as Integer Variable.

3.6 JGrafchart Textual Language

The JGrafchart textual language is used in actions and in transition conditions.

3.6.1 Action Language

The action language syntax is defined by the following abstract grammar:

action -> actionStmt ';' (actionStmt ';')*

actionStmt -> storeStmt | periodicStmt | exitStmt | normalStmt | abortStmt

storeStmt -> 'S' (assignment | methodCall)

periodicStmt -> 'P' (assignment | methodCall)

exitStmt -> 'X' (assignment | methodCall)

normalStmt -> 'N' id // id must reference a digital output or a boolean variable

abortStmt -> 'A' (assigment | methodCall)

assignment -> id '=' exp // id may not reference an input variable

methodCall -> id '(' [argumentList] ')' | id '.x' | id '.t' | id '.s'

argumentList -> exp (',' exp)*

```
exp -> exp '?' exp ':' exp |   // conditional expression
      exp '&' exp |          // boolean and
      exp '|' exp |          // boolean or
      exp '==' exp |         // equality
      exp '!=' exp |         // non-equality
      exp '<' exp |          // numerical less than
      exp '>' exp |          // numerical greater than
      exp '<=' exp |         // numerical less than or equal
      exp '>=' exp |         // numerical greater than or equal
      exp '+' exp |          // addition or string concatenation
      exp '-' exp |          // subtraction
      exp '*' exp |          // multiplication
      exp '/' exp |          // division
      '-' exp |              // numerical negation
      '!' exp |              // boolean negation
      string |
      number |
      methodCall |
      id |
      '(' exp ')'
```

id -> ('a' - 'z'| 'A' - 'Z') ('a' - 'z' | 'A' - 'Z' | '_' | '.' | '^' | '0' - '9')*

string -> '"' *arbitrary ASCII characters* '"'

number -> digit (digit)*

digit -> '0' - '9' | '.'

where

- **(...)** means one occurrence
- **[...]** means zero or one occurrence
- **(...)*** means zero or multiple occurrences
- | means OR
- **-** means an ASCII sequence
- // line comment

Comments:

- The identifier in a method call may consist of a object reference followed by a method reference, e.g., r1.getWidth(). It may also only consist of a function reference, e.g., max(b1,b2,b3)
- The identifier may only contain one evaluation of a string variable name-pointer. For example, the following expression is not allowed objectref^.attributeref^ (The same functionality can be obtained by using an intermediary help variable.
- There are three special method calls that do not take any arguments and do not use any argument list parentheses:
    - Stepname.x returns true if the corresponding step is active and false otherwise
    - Stepname.t returns the number of scan cycles that have elapsed since the step last was activated
    - Stepname.s returns the number of seconds that have elapsed since the step last was activated.
- The keywords **S, s, X, x, P, p, N, n, A, a, V, v,** and **R, r,** are reserved and may not be used as identifiers.

3.6.2 Condition Language

The textual language used in transition conditions is equivalent to the expression (exp) in the abstract syntax grammar for the action grammar with one exception. The additional / unary prefix operator is allowed. It may be used in the following ways:

- Raising edge event detection.

The expression **'/'id** is true if the value of id is true in the current scan cycle and if the value of id was false in the previous scan cycle. Id may only refer to a digital input variable. For example, the expression **/stop** returns true if stop was false in the previous scan and is true in the current scan.

- Falling edge event detection.

The expression **'\'id** is true if the value of id is false in the current scan cycle and if the value of id was true in the previous scan cycle. Id may only refer to a digital input variable. For example, the expression **\stop** returns true if stop was true in the previous scan and is false in the current scan.

- CCOM message arrival.

The expression **'/'id** is true if an incoming CCOM message has arrived and has been decoded into the XML Message In structure reference by id.

3.6.3 Functions

The textual language contains a number of predefined functions:

3.6.3.1 Mathematical functions

- real sin(real arg)

Returns the sine of the argument.

- real cos(real arg)

Returns the cosine of the argument.

- real max(real arg1, ... , real argn)

Returns the maximum value of its arguments.

- real min(real arg1, ... , real argn)

Returns the minimum value of its arguments.

- real random()

Returns a uniformly distributed value between 0.0 and 1.0.

- real sqrt(real arg)

Returns the square root of the argument.

3.6.3.2 Text and time functions

- void println(string arg)

Writes the string on the command window from which JGrafchart was started.

- void writeMessage(string arg)

Writes the string in the message menu on the toolbar.

- void clearMessages(string arg)

Removes all messages from the message menu on the toolbar.

- string format(real arg)

Returns a string containing the formatted value of the argument. The formatting is performed using the Java DecimalFormatSymbols with MaximumFractionDigits = 3 and MinimumFractionDigits = 1

- string currentTime()

Returns a string containing the current date and time.

- string getTimestamp()

Returns a string containing the CHEM timestamp format version of the current time. No millisecond or timezone information is used. An example of what the string might look like is "2003-06-06T11:29:31".

- int getTickTime()

Return the scan interval for a top-level workspace or for the subworkspace of a workspace object in milli seconds.

- returnType stringApply(string object, string method, string returnType, string arg1Type, arg1Type arg1, ..., string argNType, argNType argN)

Generic function that makes it possible to apply an arbitrary Java method to a string. The function is implemented using Java Reflection. The first argument is the string that the method should be applied to. The second argument is a string containing the name of the method to be invoked. The third argument is a string that decides the return argument type of the method. This type will also be the type of the returned value of stringApply. Legal values are "string", "real", "int", and "boolean". The remaining arguments consist of argument pairs where the first argument in the pair decides how the second argument should be evaluated. The legal values are the same as for the return type. For example, the following call to stringApply extracts the substring "Graf" from the argument "JGrafchart", stringApply("JGrafchart","substring","string","int",1,"int",5).

3.6.3.3 Colour functions

- int RGBToInt(real red, real green, real blue)

Converts a color represented by three real RGB components in the interval 0.0 - 1.0 to an integer RGB value.

- real intToRed(int arg)

Extracts the red color component as a value in the range 0.0 - 1.0 from an integer RGB value.

- real intToGreen(int arg)

Extracts the green color component as a value in the range 0.0 - 1.0 from an integer RGB value.

- real intToBlue(int arg)

Extracts the blue color component as a value in the range 0.0 - 1.0 from an integer RGB value.

3.6.3.4 Debug functions

- void freeMemory()

Print the amount of free memory of JGrafchart in the command window from which JGrafchart was started.

- void totalMemory()

Print the amount of total memory used by JGrafchart in the command window from which JGrafchart was started.

3.6.3.5 Dynamic Loading and Creation Functions

- string load(string absoluteFileName)

Loads the application named by absoluteFileName into JGrafchart. Returns the top-level workspace name of the application.

- boolean compile(string name)

Compiles the top-level workspace named by name. Returns true if the compilation was successful.

- void start(string name)

Starts the top-level workspace named by name. Requires that the workspace has been successfully compiled.

- void stop(string name)

Stops the top-level workspace named by name.

- void delete(string name)

Deletes and removes the top-level workspace named by name.

- void new(string class, string name, string workspace,int x, int y)

Creates a new object of the given class. The name of the object is set to the name argument. The object is added to the workspace denoted by workspace at the x and y position.
Currently only the classes Rectangle and Ellipse are supported.

3.6.3.6 Misc. functions

- void addPalette()

Adds the palette if it was previously hidden.

- void removePalette()

Hides the palette if it was previously visible.

- void exec(string command)

Executes the command in a separate operating system process.

- returnType applyStaticMethod(string class, string method, string returnType, string arg1Type, arg1Type arg1, ..., string argNType, argNType argN)

Generic function that makes it possible to call an arbitrary static method of any Java class. The function is implemented using Java Reflection. The first argument is the name of Java class. The second argument is a string containing the name of the method to be invoked. The third argument is a string that decides the return argument type of the method. This type will also be the type of the returned value of applyStaticMethod. Legal values are "string", "real", "int", and "boolean". The remaining arguments consist of argument pairs where the first argument in the pair decides how the second argument should be evaluated. The legal values are the same as for the return type. For example, the following call to applyStaticMethod calls the random() method of the Math class. applyStaticMethod("java.lang.Math","random","real").

3.6.3.7 CCOM functions

- void connectCCOM()

Programmatic connect to CCOM.

- void disconnectCCOM()

Programmatic disconnect from CCOM.

- void setCCOMServer(string IPaddress)

Sets the IP address of the CCOM server.

- string getCCOMServer()

Returns the IP address to the CCOM server.

- void setCCOMPort(string port)

Sets the CCOM port. (Default = 3412).

- string getCCOMPort()

Returns the CCOM port.

- void setCCOMLoginName(string name)

Sets the CCOM login name. (Default = JGrafchart)

- string getCCOMLoginName()

Returns the CCOM login name.

- void subscribe(string topic)

Subscribes to a CCOM topic.

- void unsubscribe(string topic)

Unsubscribes from a CCOM topic.

3.6.4 Evaluation Rules

The JGrafchart textual language uses loose typing and automatic type-casting whenever possible. It is the context that decides how an expression is evaluated. The following rules apply:

- A transition condition is evaluated as a boolean expression.
- In an assignment statement it is the type of the assigned variable that decides how the expression for the assigned value should be evaluated. For example, consider the following stored action:

S Variable = 0;

If Variable is of boolean type the expression will be evaluated as the boolean literal 0. If Variable is of real or integer type the expression will be evaluated numerically. Finally, if Variable is a string then the expression will be evaluated as a string. In this case this will give rise to the value "0".

- A action consisting only of a method call will be evaluated as a real expression.
- A boolean value will return the numerical values 0 or 1 when evaluated as an integer.
- A boolean value will return the numerical values 0.0 or 1.0 when evaluated as a real.
- A boolean value will return the strings "0" or "1" when evaluated as a string.
- An integer value will return the boolean value 1 if the value is 1 and the boolean value 0 otherwise when evaluated as a boolean.
- An integer value will return the corresponding real value when evaluated as a real (ordinary Java casting).
- An integer value will return the value as a string when evaluated as a string.
- A real value will return the boolean value 1 if the value is equal to 1 when cast to an integer, and the boolean value 0 otherwise when evaluated as a boolean.
- A real value will be cast to an integer when evaluated as an integer.
- A real value will return the value as a string when evaluated as a string.
- A string value will return the boolean value 1 if equal to "1", and the boolean value 0 otherwise when evaluated as a boolean.
- A string value will return the corresponding integer if it can successfully be parsed to an integer, and 0 otherwise, when evaluated as an integer.
- A string value will return the corresponding real if it can successfully be parsed to a double, and 0.0 otherwise, when evaluated as a real.

3.7 JGrafchart Execution Model

In order for a JGrafchart application to be started it must first have been successfully compiled.

3.7.1 Compilation

During the compilation the following things are performed:

- For each transition and exception transition two lists are built-up. The precedingList contains references to all steps that precede the transition. The succeedingList contains references to all steps that succeed the transition.
- The textual expressions are parsed. During the parsing syntax trees are generated. If the parsing is successful the syntax trees are interpreted and all name references are resolved.

3.7.2 Execution

If the compilation was successful the JGrafchart application can be started. The execution is performed as a three-phase operation. Each step has two attributes that controls the execution:

- boolean x - An attribute that is true if the step is active in the current scan.
- boolean newx - An attribute that is true if the step should be active in the next scan.

In every scan the following operations are performed:

1. Digital and analog inputs are read.
2. *Phase One:* The transitions are traversed and every transition for which the preceding steps are all active and for which the transition condition evaluates to true is marked for firing.
3. *Phase Two:* The transitions are traversed. For every transition that is marked for firing the following operations are performed:
    - The newx attribute is set to false for every step in the precedingList. The exit actions of these steps are executed.
    - The newx attribute is set to true for every step in the succeedingList. The enter actions of these steps are executed.
4. *Phase Three:* All steps are traversed. The following operations are performed for every step:
    - The state of the step is updated by assigning newx to x (x = newx;).
    - The timer of the step (accessed through stepname.t or stepname.s) is updated.
    - If the step is currently active and will remain so also in the next scan then the periodic actions of the step are executed.
    - If the activation status of the step has changed since the last scan the normal actions of the step are evaluated.
5. For every digital output that has been modified by the evaluation of normal actions, the total effect of the evaluation is effectuated.
6. The execution thread sleeps until the start of the next scan cycle. The period of the execution thread is determined by the thread sleep interval of the top-level workspace.

The above execution model has the following properties:

- In a serial chain of steps and transitions at most one transition will be fired in each scan cycle.

- The model effects how alternative transitions with non-mutually exclusive conditions are handled. In JGrafchart all of these alternative transitions will be fired. This conforms to the Grafcet standard. It is also gives a fully deterministic behavior. It is good programming practice to make sure that alternative transitions always are mutually exclusive.
- The order in which the transitions and steps are evaluated in the execution algorithm is irrelevant. This greatly simplifies the implementation.
- Digital outputs modified by normal actions will behave according to the Grafcet definition. For example, a digital output that is set by normal actions in two succeeding steps will remain true.

4 Graphical User Interface

4.1 Graphical User Interface Elements

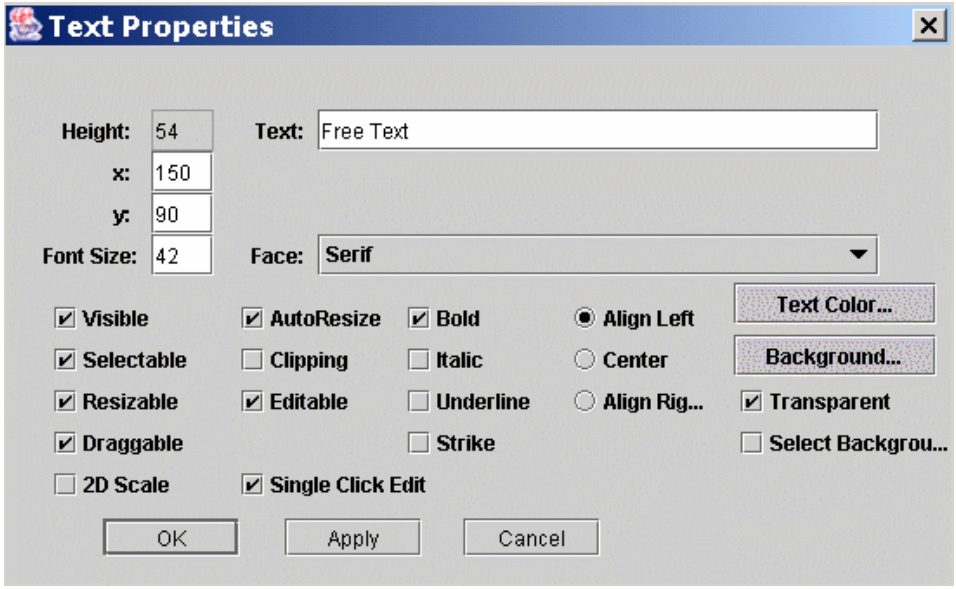
4.1.1 Free Text

A free text is a one-line editable text string. By single-clicking with the left mouse button the text can be edited. From the edit menu it is possible to change most of the text properties. The font size can also be changed with the mouse using the vertical resize handles.

Menu Choices:

- **Edit**

  Brings up a editor window from which the text properties can be changed.



x and y:
Changes the position of the text.

Font Size:
Changes the font size.

Text:
Changes the text content.

Visible:
Decides whether the text should be visible or hidden.

Selectable:
Decides whether it should be possible to select the text with mouse or not.

Resizable:
Decides whether it should be possible to resize the text with the mouse or not.

Draggable:
Decides whether it should be possible to drag the text with the mouse or not.

2D Scale:
Change whether the resizing of the text depends on both the width and the height, rather than just the height to determine the font size to use in displaying the string.

Text:
Change the text of the string.

Face:
Change the font face of the string. The available choices depend on which fonts that are available on the particular machine. In order to ensure portability between different platforms it is recommended to only use the following five symbolical face names: **Serif, SansSerif, Monospaced, Dialog, DialogInput.** These are guaranteed to be available in every Java system. However, they may map to different real fonts depending on the platform.

AutoResize:
Change whether the text should automatically adjust its length when the text string is changed.

Clipping:
Change whether the text is only drawn within the bounding rectangle.

Editable:
Change whether the text is editable by the end user.

Single Click Edit:
If the text is editable, control whether the user can start editing with a double click or with a single click.

Bold:
Change the boldness of the text.

Italic:
Change whether the text is italic.

Underline:
Change whether the text is underlined.

Strike:
Change whether the text is shown with a strike-through.

Align Left:
Left justify each line.

Center:
Center each line.

Align Right:
Right justify each line.

Text Color:
Brings up a JColorChooser menu from which the user can change the text colour.

Background:
Brings up a JColorChooser menu from which the user can change the background colour.

Transparent:
Set whether or not the text background is filled with the background color before drawing the text (opaque), or if only the text is drawn without hiding what's underneath the text (transparent).

Select Backgro...:
If the text background is transparent and the text is not resizable, control whether gainedSelection() makes the background opaque and whether lostSelection() makes the background transparent again.

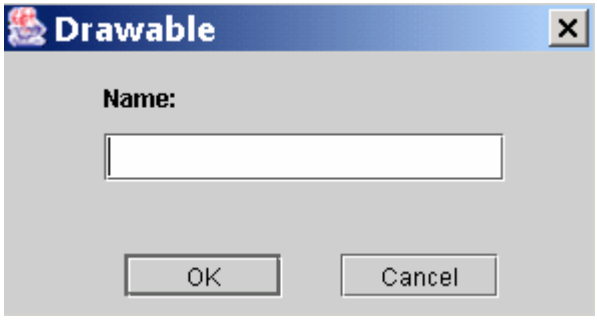
OK:
Apply the changes and close the edit window.

Apply:
Apply the changes.

Cancel:
Cancel any changes and close the edit window.

- **Name**

Brings up an edit menu from which the user can change the name of the free text. The name is used to refer to the text in method calls,



Text Methods:

- setText(string newText)

Changes the text string.

- string getText()

Returns the text string.

- setTextColor(int RGBColor)

Changes the text color.

- int getTextColor()

Returns the current RGB value of the text color.

- setBkColor(int RGBColor)

Changes the background color.

- int getBkColor()

Returns the current RGB value of the background color.

- setFontSize(int size)

Changes the font size.

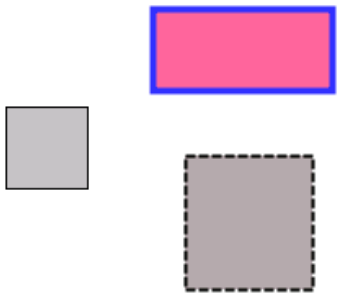
- int getFontSize()

Returns the current font size.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.2 Rectangle

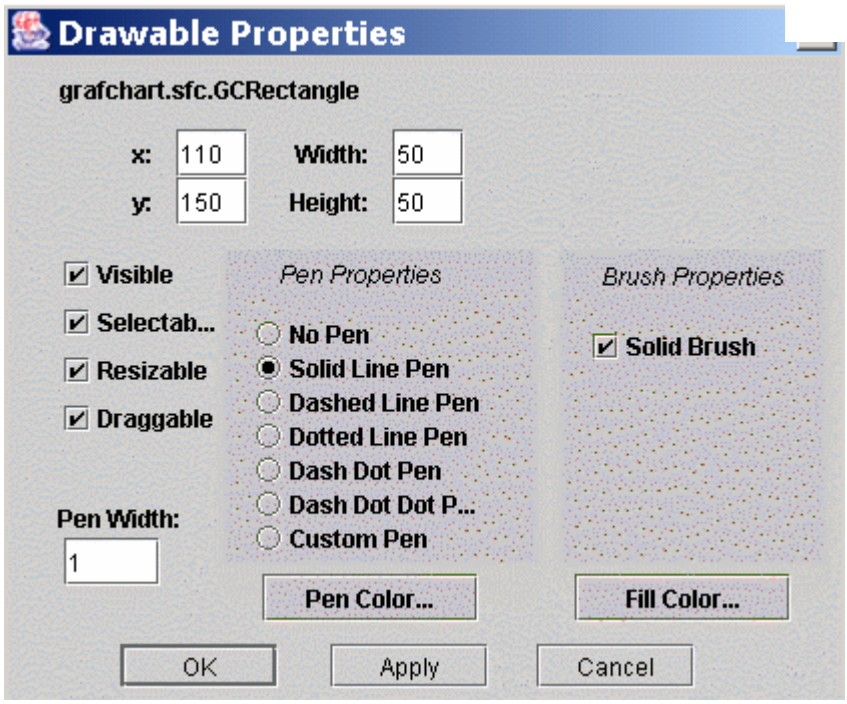
A resizable rectangle object. From the edit menu it is possible to change most of the properties of the rectangle.



Menu Choices:

- **Edit**

  Brings up an editor window from which the text properties can be changed.



x and y:
Changes the position of the rectangle.

Width:
Change the width..

Height:
Changes the height..

Visible:
Decides whether the rectangle should be visible or hidden.

Selectable:
Decides whether it should be possible to select the rectangle with mouse or not.

Resizable:
Decides whether it should be possible to resize the rectangle with the mouse or not.

Draggable:
Decides whether it should be possible to drag the rectangle with the mouse or not.

Pen width:
Change the pen width of the rectangle.

Pen Properties:

- **No Pen**
- **Solid Line Pen** (Default)
- **Dashed Line Pen**
- **Dotted Line Pen**
- **Dash Dot Pen**
- **Dash Dot Dot Pen**
- **Custom Pen.** (Currently not in use)

Brush Properties:

- **Solid Brush**. Change whether the rectangle should be filled or transparent.

Pen Color:
Brings up a JColorChooser menu from which the user can change the pen colour.

Fill Color:
Brings up a JColorChooser menu from which the user can change the fill colour.

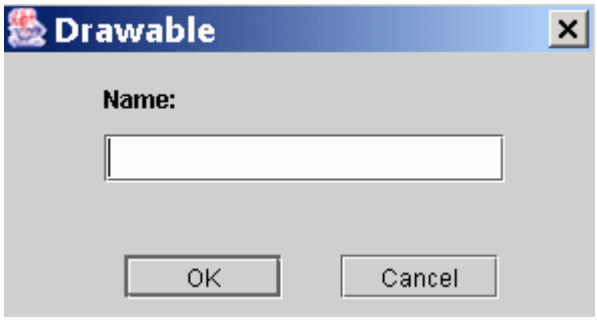
OK:
Apply the changes and close the edit window.

Apply:
Apply the changes.

Cancel:
Cancel any changes and close the edit window.

- **Name**

Brings up an edit window from which the user can change the name of the rectangle. The name is used to refer to the rectangle in method calls,



Text Methods:

- setPenColor(int RGBColor)

Changes the pen color.

- int getPenColor()

Returns the current RGB value of the pen color.

- setFillColor(int RGBColor)

Changes the fill color.

- int getFillColor()

Returns the current RGB value of the fill color.

- setPenWidth(int width)

Changes the pen width.

- int getPenWidth()

Returns the current pen width.

- setPenStyle(int RGBColor)

Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

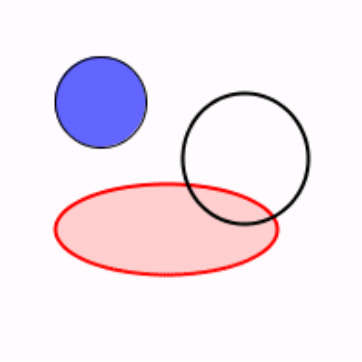
Returns the current pen style.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)
- void remove()

Removes the rectangle.

4.1.3 Ellipse

A resizable ellipse object. From the edit menu it is possible to change most of the properties of the ellipse.



Menu Choices:

- **Edit**

  Brings up an editor window from which the ellipse properties can be changed. The editor window and the options are the same as for the rectangle.

- **Name**

  Brings up an edit menu from which the user can change the name of the ellipse. The name is used to refer to the ellipse in method calls,

Text Methods:

- setPenColor(int RGBColor)

  Changes the pen color.

- int getPenColor()

  Returns the current RGB value of the pen color.

- setFillColor(int RGBColor)

  Changes the fill color.

- int getFillColor()

  Returns the current RGB value of the fill color.

- setPenWidth(int width)

  Changes the pen width.

- int getPenWidth()

  Returns the current pen width.

- setPenStyle(int RGBColor)

  Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

  Returns the current pen style.

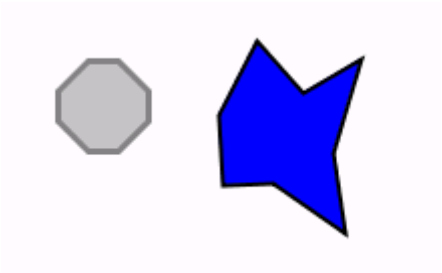
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

- void remove()

Removes the ellipse.

4.1.4 Polygon

A resizable eight-corner polygon object. From the edit menu it is possible to change most of the properties of the polygon. The resize handles are used to move the individual corner points. Polygons with different number of corner points are created using line objects.



Menu Choices:

- **Edit**

  Brings up a editor window from which the polygon properties can be changed. The editor window and the options are the same as for the rectangle with the exception for one additional checkbox.

  Cubic Bezier:
  Change whether the points should be interconnected with straight lines or with cubic Bezier splines.

- **Name**

  Brings up an edit menu from which the user can change the name of the polygon. The name is used to refer to the polygon in method calls,

Text Methods:

- setPoint(int index, int x-coord, int y-coord)

Moves a corner point to a new position.

- int getPointX(int index)

Returns the x coordinate of a point.

- int getPointY(int index)

Returns the y coordinate of a point.

- setPenColor(int RGBColor)

Changes the pen color.

- int getPenColor()

Returns the current RGB value of the pen color.

- setFillColor(int RGBColor)

Changes the fill color.

- int getFillColor()

Returns the current RGB value of the fill color.

- setPenWidth(int width)

Changes the pen width.

- int getPenWidth()

Returns the current pen width.

- setPenStyle(int RGBColor)

Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

Returns the current pen style.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.5 3D Rectangle

A resizable rectangle object with a 3D appearance. From the edit menu it is possible to change most of the properties of the 3D rectangle.

Menu Choices:

- **Edit**

  Brings up a editor window from which the rectangle properties can be changed. The window and the options are the same as for the ordinary rectangle.

- **Name**

  Brings up an edit window from which the user can change the name of the rectangle. The name is used to refer to the rectangle in method calls,

Text methods:

- setPenColor(int RGBColor)

  Changes the pen color.

- int getPenColor()

  Returns the current RGB value of the pen color.

- setFillColor(int RGBColor)

  Changes the fill color.

- int getFillColor()

  Returns the current RGB value of the fill color.

- setPenWidth(int width)

  Changes the pen width.

- int getPenWidth()

  Returns the current pen width.

- setPenStyle(int RGBColor)

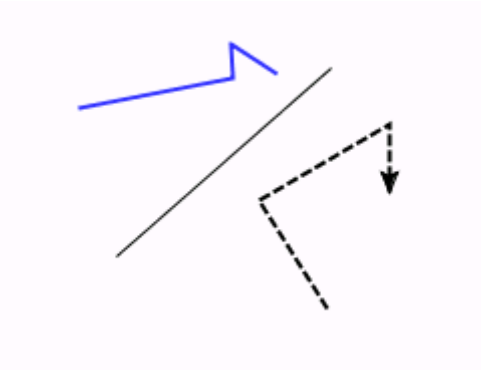
  Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

  Returns the current pen style.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.6 Line

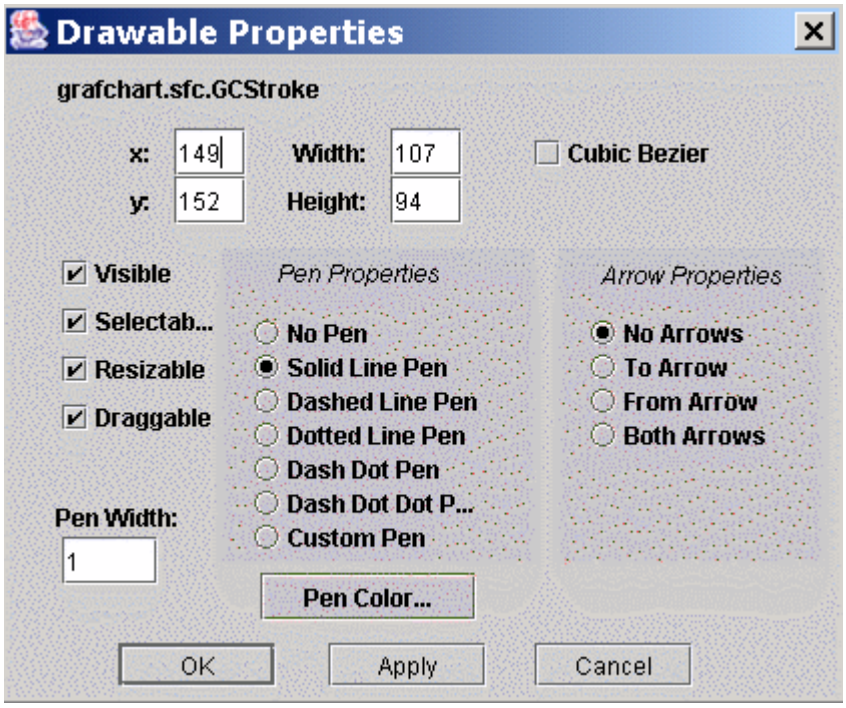


A line is created by clicking on the Line button in the Misc palette. This causes JGrafchart to change into line-drawing mode. A line is created by clicking with the mouse on the desired starting point of the line. A corner point is created for each subsequent mouse click. The line is terminated with a double-click. If the end point is sufficiently close to the start point a closed polygon will be created instead of a line. The points are sufficiently close if the grid point closest to the start point is the same as the grid point closest to the end point. The line corner points can be moved using the resize handles.

Menu Choices:

- **Edit**

  Brings up an editor window from which the line properties can be changed.



x and y:
Changes the position of the rectangle.

Width:
Change the width..

Height:
Changes the height..

Cubic Bezier:
Change whether the points should be interconnected with straight lines or with cubic Bezier splines.

Visible:
Decides whether the rectangle should be visible or hidden.

Selectable:
Decides whether it should be possible to select the rectangle with mouse or not.

Resizable:
Decides whether it should be possible to resize the rectangle with the mouse or not.

Draggable:
Decides whether it should be possible to drag the rectangle with the mouse or not.

Pen width:
Change the pen width of the rectangle.

Pen Properties:

- **No Pen**
- **Solid Line Pen** (Default)
- **Dashed Line Pen**
- **Dotted Line Pen**
- **Dash Dot Pen**
- **Dash Dot Dot Pen**
- **Custom Pen.** (Currently not in use)

Arrow Properties:

- **No Arrow** (Default)
- **To Arrow**. An arrow at the end-point.
- **From Arrow**. An arrow at the start-point.
- **Both Arrows.** Arrows at both the end and the start point.

Pen Color:
Brings up a JColorChooser menu from which the user can change the pen colour.

OK:
Apply the changes and close the edit window.

Apply:
Apply the changes.

Cancel:
Cancel any changes and close the edit window.

- **Name**

Brings up an edit menu from which the user can change the name of the line. The name is used to refer to the line in method calls.

Text Methods:

- setPoint(int index, int x-coord, int y-coord)

Moves a corner point to a new position.

- int getPointX(int index)

Returns the x coordinate of a point.

- int getPointY(int index)

Returns the y coordinate of a point.

- setPenColor(int RGBColor)

Changes the pen color.

- int getPenColor()

Returns the current RGB value of the pen color.

- setPenWidth(int width)

Changes the pen width.

- int getPenWidth()

Returns the current pen width.

- setPenStyle(int RGBColor)

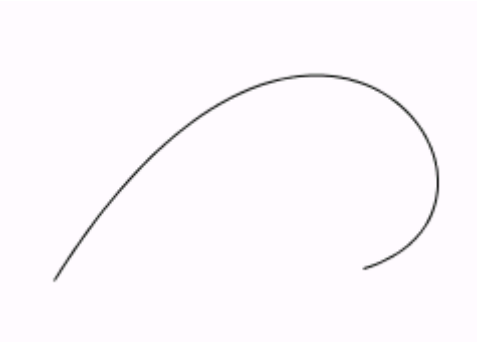
Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

Returns the current pen style.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.7 Spline



A spline is created by clicking on the Spine button in the Misc palette. This causes JGrafchart to change into spline-drawing mode. A spline is created by clicking with the mouse on the desired starting point of the spline. A corner point is created for each subsequent mosue click. The spline is terminated with a double-click. If the end point is sufficiently close to the start point a closed splined polygon will be created instead of a spline. The points are sufficiently close if the grid point closest to the start point is the same as the grid point closest to the end point. The spline corner points can be moved using the resize handles.

Menu Choices:

- **Edit**

  Brings up an editor window from which the spline properties can be changed. The editor window and options are the same as for the line.

- **Name**

  Brings up an edit menu from which the user can change the name of the spline. The name is used to refer to the spline in method calls.

Text Methods:

- setPoint(int index, int x-coord, int y-coord)

  Moves a corner point to a new position.

- int getPointX(int index)

  Returns the x coordinate of a point.

- int getPointY(int index)

  Returns the y coordinate of a point.

- setPenColor(int RGBColor)

  Changes the pen color.

- int getPenColor()

  Returns the current RGB value of the pen color.

- setPenWidth(int width)

  Changes the pen width.

- int getPenWidth()

  Returns the current pen width.

- setPenStyle(int RGBColor)

  Changes the pen style Legal values are 0 - No Pen, .65535 - Solid Pen, 1 - Dashed Pen, 2 - Dotted Pen, 3 - Dash Dot Pen, 4 - Dash Dot Dot Pen.

- int getPenStyle()

  Returns the current pen style.

- setLocation(int x, int y)

- int getXLocation()

- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

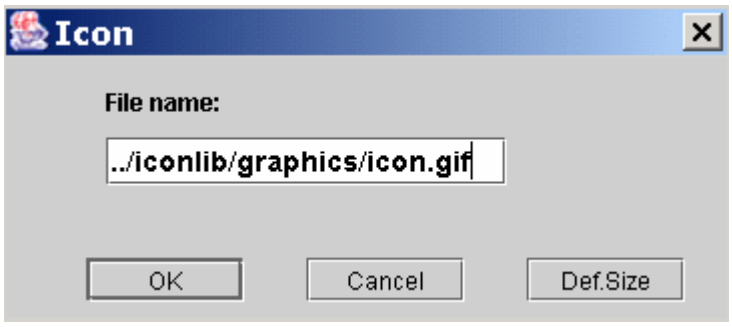
4.1.8 Icon

An icon object. The icon is given by an arbitrary .gif or .jpg file.

Menu Choices:

- **Edit**

  Brings up an editor window from which the icon file name can be changed.



File name:
The icon file name.

Def. Size:
Change the size of the icon to its original size.

- **Name**

  Brings up an edit menu from which the user can change the name of the icon. The name is used to refer to the icon in method calls.

Text Methods:

- loadFile(string fileName)

  Change the icon to a new file name.

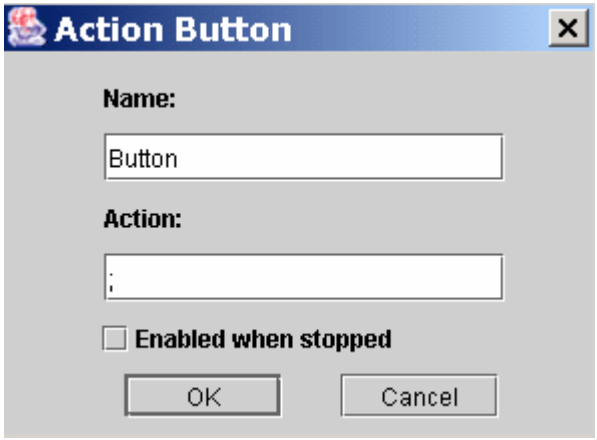
- loadURL(string URL)

  Change the icon to a new URL.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.9 Action Button

A button to which can be associated JGrafchart actions which are executed when the user clicks on the button with the left mouse button. Clicking on the button with the right button brings up the action button menu.



Name:
The name of the button is a text string that is shown in the center of the button. The size of the button is automatically adjusted to fit the length of the string.

Action:
The action to be executed when the user left clicks on the button. This must be one or several stored actions, e.g., an action could be **S Start = 1;**

Enabled when stopped:
An action button must have been compiled to be active. If enabled when stopped is selected the action button will be executed will be executed also when the application is stopped, provided that it has been successfully compiled.
By default an action button is not enabled when the application is stopped.

The action button has no menu choices and no text methods.

4.1.10 Graphical Action Button

A graphical button to which can be associated JGrafchart actions which are executed when the user clicks on the button icon with the left mouse button. Clicking on the button icon with the right button brings up the action button menu. The button icon is given by an arbitrary .gif or .jpg file.

File name:
The icon file name.

Action:
The action to be executed when the user left clicks on the button. This must be one or several stored actions, e.g., an action could be **S Start = 1;**

Enabled when stopped:
A button must have been compiled to be active. If enabled when stopped is selected the button will be executed also when the application is stopped, provided that it has been successfully compiled.
By default a button is not enabled when the application is stopped.

Def. Size:
Change the size of the icon to its original size.

Text Methods:

- loadFile(string fileName)

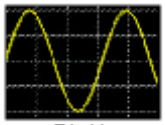
Change the icon to a new file name.

- loadURL(string URL)

Change the icon to a new URL.

- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()
- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.11 Plotter

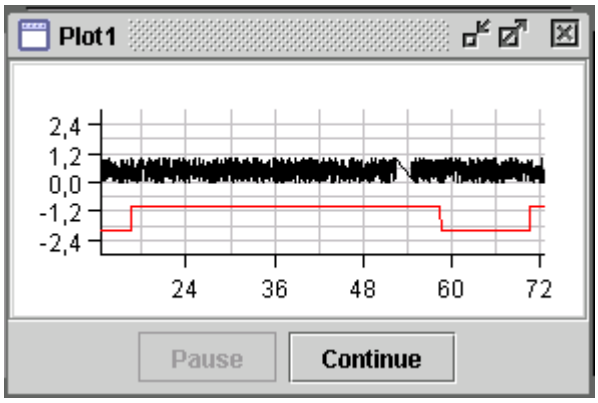


A one-or two-channel plotter object.

Menu Choices:

- **Show/Hide**

  A toggle that shows or hides the plotter panel associated with the plotter. The plotter panel can be resized.

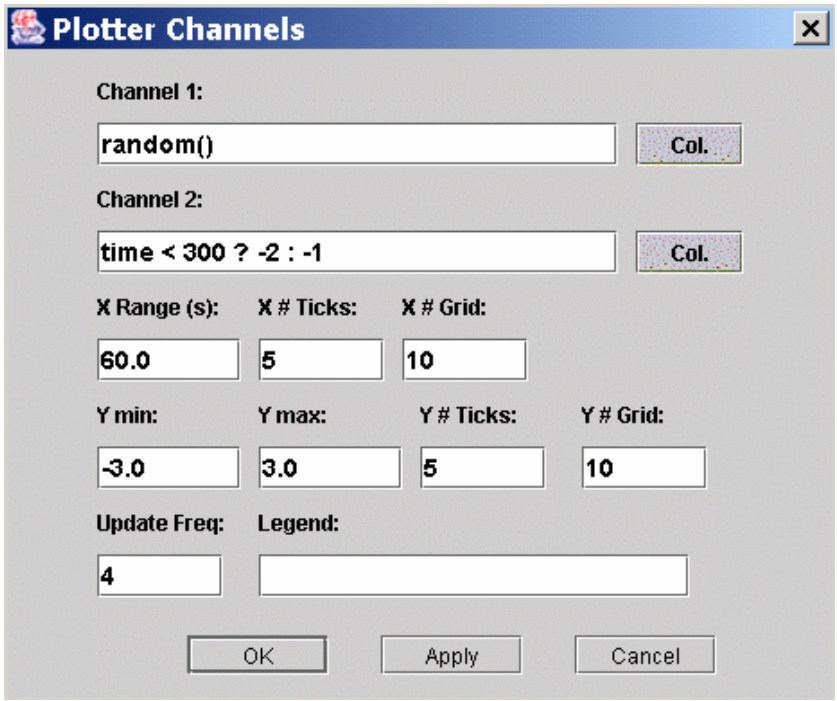


  Pause:
  Pauses the plotting.

  Continue:
  Continues the plotting.

- **Edit**

  Brings up an editor window from which the plotter properties can be changed.



  Channel 1:
  A numerical expression that will be evaluated and plotted on channel one.

Channel 1 Color:
Brings up a JFileChooser from which the pen color of channel one is changed.

Channel 2:
A numerical expression that will be evaluated and plotted on channel two.

Channel 2 Color:
Brings up a JFileChooser from which the pen color of channel two is changed.

X Range (s):
The range of the x-axis in seconds.

X # Ticks:
The number of tick marks on the x-axis.

X # Grid:
The number of grid lines on the x-axis.

Y min:
The minimum value of the y-axis.

Y max:
The maximum value of the y-axis.

Y # Ticks:
The number of tick marks on the y-axis.

Y # Grid:
The number of grid lines on the y-axis.

Update Freq.:
The update frequency in relation to the scan cycle. For example a value of 2 means that the plotter panel will be updated every second scan.

Legend:
A text that will be printed at the top of the plotter panel.

The plotter has no text methods.

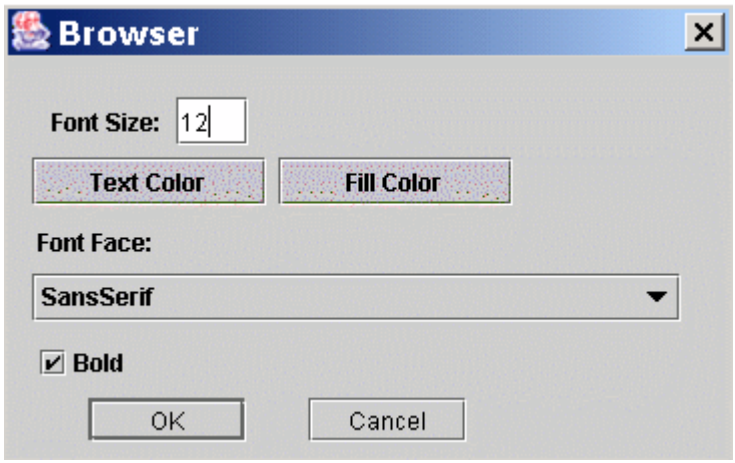
4.1.12 Browser

The browser is a scrollable and resizable message browser that can be used for alarm and event displays. The messages are represented by text strings. The user can select a message by clicking on it. A message is de-selected by clicking on another message, clicking again on the selected message, or clicking on the browser background. The selected message is highlighted (the text is displayed in the background color and the background is displayed in the text color). The browser messages are not persistent in the sense that they are not saved with the application.

Menu Choices:

- **Edit**

Brings up an editor window from which the browser properties can be changed.



Font Size:
The font size at which newly added messages will be displayed. Does not effect the already inserted messages.

Text Color:
The text color of newly added messages. Does not effect the already inserted messages.

Fill Color:
The background color of the browser.

Font Face:
The font face of newly added messages. Does not effect the already inserted messages.

Bold:
When selected newly inserted messages will be displayed in bold face. Does not effect already inserted messages.

- **Name**

Brings up an edit menu from which the user can change the name of the icon. The name is used to refer to the browser in method calls.

Text Methods:

- addItem(string message)

Adds a string at the end of the browser (the bottom of the browser)

- addItem(int index, string message)

Adds the string at the given location. Index 0 corresponds to the top of the browser.

- removeItem(int index)

Removes the item at the given position.

- string getItem(int index)

Returns the string at the given position.

- int getNumItems()

Returns the number of items in the list.

- clear()

Removes all items from the list.

- string getSelectedItem()

Returns the text of the selected item. Returns the empty string if no item is selected.

- int getSelectedIndex()

Returns the index of the currently selected item. Returns -1 if no item is selected.

- int getTextColor()

Returns the current text color of the browser.

- setTextColor(int color)

Sets the text color.

- int getFillColor()

Returns the current fill color of the browser.

- setFillColor(int color)

Sets the fill color.

- int getTextSize()

Returns the text font size..

- setTextSize(int color)

Sets the text font size.

- boolean isBold()

Returns true if the bold property is set.

- setBold(boolean value)

Sets the bold property.

- string getFont()

Returns the current text font face name..

- setFont(string name)

Sets the text font face name.

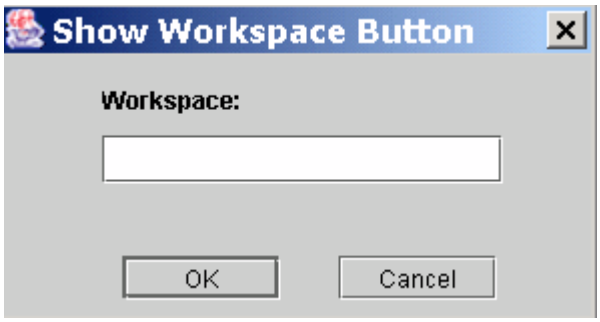
- setLocation(int x, int y)
- int getXLocation()
- int getYLocation()

- int getHeight()
- int getWidth()
- setWidth(int width)
- setHeight(int height)
- setVisible(boolean visible)

4.1.13 Show Workspace Button

A button that is used to show a new workspace slide in a JGrafchart slide show.

Clicking on the button when not executing brings up an edit window in which the global (qualified) name of the next workspace slide to show is entered. Clicking on the button during execution will show this workspace slide.



The show workspace button has no menu choices or text methods.

4.1.14 CHEM Icon

The icon of the CHEM project that has funded the development of JGrafchart.

The CHEM icon has no menu choices and no text methods.

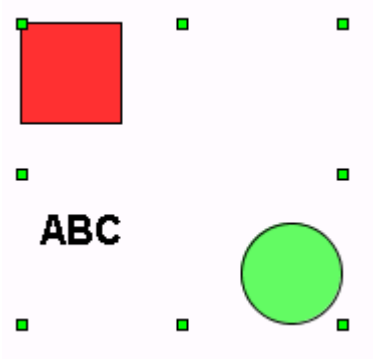
4.1.15 LUCAS Icon

The icon of the LUCAS Center for Applied Software Research at Lund University that also has funded the development of JGrafchart.

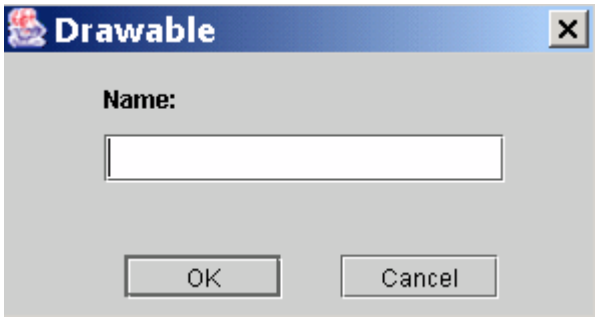
The LUCAS icon has no menu choices and no text methods.

4.1.16 Group

A group of object is created by selecting a number of objects and then applying the menu choice Group.



Clicking with the right mouse button in an "empty" part of the group brings up a name display from which the user can change the name of the group. The name is used to reference the group and its internal parts. If the group does not contain any empty space, then the name display can be reached from the tree view window. The internal parts of a group are referenced using dot-notation.



5 Custom IO

It is possible for the user to provide his own implementations of the digital and analog IO objects. This is done by providing Java implementations of a few simple Java interfaces and to give the class name as a command line argument when JGrafchart is started.

5.1 Interface LocalIO

In order to provide a custom IO the use should provide a class that implements the interface LocalIO. The interface and the implementation class should be placed in the grafchart/source/grafchart/sfc/ directory. The interface defines four methods:

```
package grafchart.sfc;

public interface LocalIO {

   public AnalogInput   createAnalogInput(int channel);
   public AnalogOutput  createAnalogOutput(int channel);
   public DigitalInput  createDigitalInput(int channel);
   public DigitalOutput createDigitalOutput(int channel);

}
```

The interface makes use of four additional interfaces: AnalogInput, AnalogOutput, DigitalInput, and DigitalOutput.

5.2 Interface AnalogInput

This interface defines a generic analog input. It contains the following definition:

```
package grafchart.sfc;

public interface AnalogInput {

   public double get();

}
```

5.3 Interface AnalogOutput

This interfaces defines a generic analog output, It contains the following definition:

```
package grafchart.sfc;


public interface AnalogOutput {

   public void set(double value);

}
```

5.4 Interface DigitalInput

This interfaces defines a generic digital input, It contains the following definition:

```
package grafchart.sfc;
```

```
public interface DigitalInput {

   public boolean get();

}
```

5.5 Interface DigitalOutput

This interfaces defines a generic digital output, It contains the following definition:

```
package grafchart.sfc;

public interface DigitalOutput {

   public void set(boolean value);

}
```

5.6 Example: LundPrintIO

The example implementation LundPrintIO.java provides a simple example that writes all output to the command terminal from which JGrafchart is started and which returns random values as inputs. The example is found in grafchart/source/grafchart/sfc. The directory grafchart/bin contains command files for compiling the example and for starting JGrafchart with this IO.