Authors:
Yann **STEPHAN** (Yann.Stephan@hp.com)
Sebastien **PELLIZZARI (Sebastien.Pellizzari@hp.com)**

Sip Test Studio
User Manual

SIP Test Studio

Summary

SIP Test Studio

User Manual

Introduction

SIP Test Studio (*STS*) is a Windows application that allows graphical edition of SIPP scenarios – SIP functional and performance test platform – from simple UAC/UAS to complex 3PCC-extended cases involving multiple clients, by representing them as diagrams.

The present document describes the application features from the user point of view. For information regarding STS development and internal mechanisms, please refer to the document entitled “Sip Test Studio: Implementation Reference”.

Warning: this documentation is preliminary and therefore subject to change without notice. The referenced documents may or may not be already available either.

Overview

SIP Test Studio provides a graphical interface allowing the user to edit documents made of multiple SIPP scenarios. Such document is known as *call flow*, and is made of one or several *calls* represented as diagrams. Such calls are sequences of *nodes* representing the various actions needed to model the behavior of a SIP user agent, such as sending or receiving SIP messages, processing them, and playing media streams through RTP. Furthermore, the calls can communicate with each other by sharing data extracted from or inserted into SIP messages, thus making edition of complex scenarios such as 3PCC and call conferencing possible.

Requirements

STS is a Windows only application fully written in C# and relies on the .NET Framework 3.5. Therefore it requires at least – and has only been tested on – Windows XP or Windows Vista with an up-to-date .net environment. Except a 3-buttons mouse, no particular hardware is required. But as the application makes intensive use of graphical assets for both its interface and the diagram editor the hardware resources you may need actually depend on the richness of the documents you will want to manipulate in STS.

Development State

STS is currently in early development stage but is intended to go open source in Q4 of 2008. For more information about STS development, refer to the document entitled “Sip Test Studio: Implementation Reference”.

Document Outline

The present document is made of the following chapters:

- **The User Interface**, which presents the application GUI and its features.
- **Call Flow: the Basics**, which introduces the basics of call flow edition.
- **Call Flow: Extended**, which describes the more complex features of STS regarding data extraction/insertion through variable manipulation and inter-call communication.

– Chapter 1 –

The User Interface

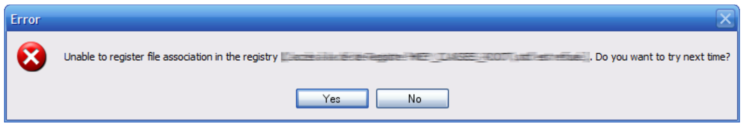
Global Overview

Startup

At startup, the STS application shows a “splash screen” while it is loading in memory. When the initialization completes, that screen disappears and you gain control over the application.

The STS splash screen

As part of the initialization phase, the application checks the registry for valid file association with “.std” documents. When this check is performed, you may encounter the following message:

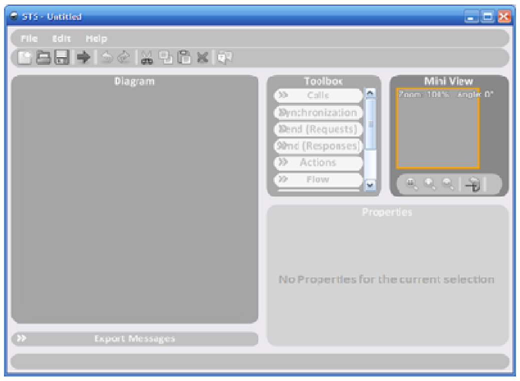


Unable to register file association in the registry [redacted]. Do you want to try next time?

If so, it is likely that you are not logged in with Administrator rights, or didn’t run the application with such rights enabled (e.g. in Vista: right-click on the executable file, then “Run as administrator”), resulting in STS not having write access to the registry. From there you can either click ‘Yes’ and run the application again with the previous condition verified, or ignore this message and click ‘No’ so that this check will be skipped next time.

The Environment

After the initialization completes, the splash screen is closed and the main window is displayed:



The STS workspace

The STS environment consists of one main window that displays and allows edition of one call flow document at a time. To edit multiple documents simultaneously, just run multiple instances of STS.

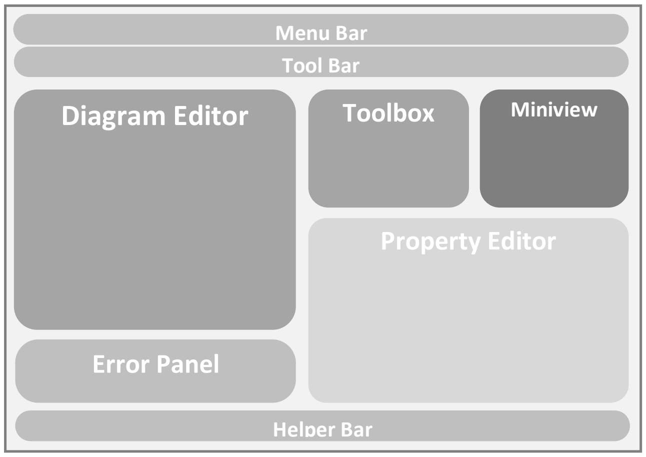
Look and Feel

The STS environment provides a uniform user interface that makes use of a unique color scheme and set of shapes. With gray as its base color, the GUI is made of round-shaped areas of different tones which make them easy to discern and spot. A colored highlight provides instant feedback on sensitive areas like buttons and collapsible panels as the mouse moves over them. The overall look-and-feel of the STS environment aims at providing every feature on-screen while not distracting you from the content you're editing – that is, the diagram.

On the contrary, the diagram editor and the tabs inside the property editor (see section "The Property Editor" for more information) are the only parts of the GUI that make intensive use of colors. The diagram editor uses different shapes and tones on nodes of different categories to help you distinguish them, while the property editor highlights keywords, syntax elements and errors in code and SIP message editors to give prominence to the semantic of their content.

The STS Workspace

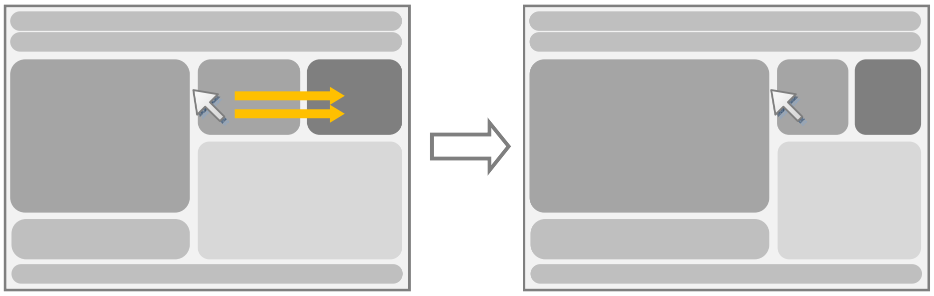
The main window defines the whole STS workspace. Every tool needed for call flow authoring and visualization is available through this interface that exposes the following layout:



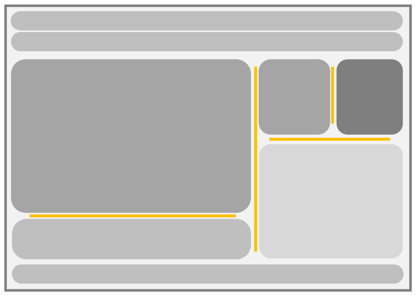
The following sections describe each part of the layout and present their features.

Panel Organization

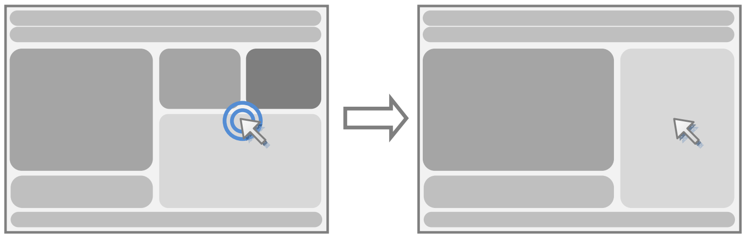
The light-gray space separating the main 5 panels of the workspace is a splitter zone that allows you to resize the panels by using the mouse (click and drag):



The following zones highlighted in orange can be moved with the left mouse button to resize the panels:

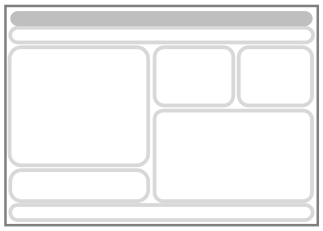


Double-clicking on the headline of a panel will expand it over its neighbor(s):



To revert it back to its former size, double-click on its headline again.

The Menu Bar



The **menu bar** provides access to general actions such as document manipulation (save, open, etc.) and edition commands. It contains 3 top-level menus:

The **File** menu exposes the following items:

- ***New Document***: closes the current document and opens a new blank one. If the current document has unsaved modifications, you will be asked whether to save or discard them.
  Shortcut: Ctrl + N.
- ***Open Document***: closes the current document, prompts for a path to an existing document file and opens it. If the current document has unsaved modifications, you will be asked whether to save or discard them.
  Shortcut: Ctrl + O.
- ***Save Document***: saves the current document. If this is the first time the document is saved since it was created, this will do the same as the “Save Document as” item action (see below).
  Shortcut: Ctrl + S.
- ***Save Document As***: Prompts for a file path and saves the current document at the specified location. The new path will become the active path for this document, and further “Save Document” actions will overwrite the file at this location.
  Shortcut: Ctrl + Shift + S.
- ***Export***: exports the call flow document to SIPP scenarios. The generated scenario files will be output into the same directory as the call flow document.
  Shortcut: Ctrl + E.
- ***Exit***: closes the application. If the current document has unsaved modifications, you will be asked whether to save or discard them.
  Shortcut: Ctrl + Q.

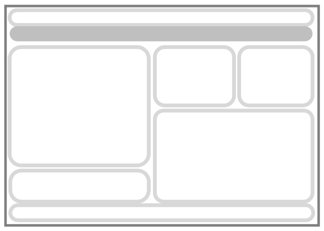
The **Edit** menu exposes the following items:

- ***Undo***: undoes the last action. Only actions that involve node manipulation such as adding, moving, sizing, renaming or removing nodes can be undone. Changes made to the properties of a node are not taken into account by the undo/redo mechanism.
  Shortcut: Ctrl + Z.
- ***Redo***: redoes the last undone action.
  Shortcut: Ctrl + Y.
- ***Select All***: selects all nodes.
  Shortcut: Ctrl + A.
- ***Cut***: cuts the selected nodes and saves them to the clipboard.
  Shortcut: Ctrl + X.
- ***Copy***: copies the selected nodes to the clipboard.
  Shortcut: Ctrl + C.
- ***Paste***: pastes nodes from the clipboard.
  Shortcut: Ctrl + V.
- ***Delete***: removes selected nodes from the diagram.
  Shortcut: Del.

The **Help** menu exposes the following items:

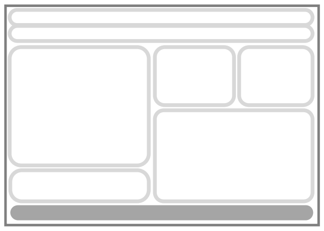
- ***About***: shows the application **about box**.
  Shortcut: F1.

The Tool Bar



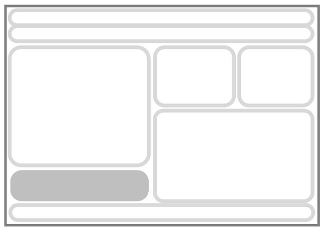
The **tool bar** provides buttons representing the most common actions that are accessible from the menu bar. Their functions are the same, and the tool bar is only here to provide quicker access to them.

The Helper Bar



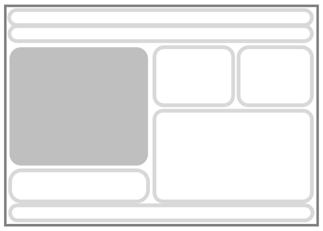
The **helper bar** at the bottom of the workspace provides a quick description of the purpose of the panel at the current cursor location. Its content may change depending on the state of the 3 modifiers keys: *Control*, *Shift* and *Alt*.

The Error Panel



The **error panel** is collapsed by default, and will expand when errors are emitted by the Export action. By double clicking on a message, the editor will select and show the node that is responsible for the corresponding error.

The Diagram Editor

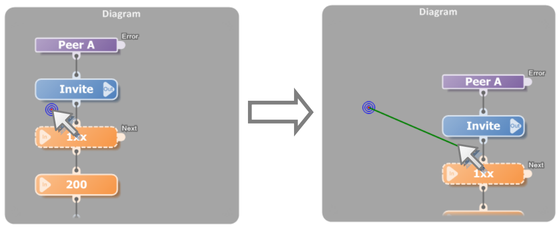


The **diagram editor** displays the diagrams of the current call flow document and allows you to edit them. The nodes are represented by round-shaped rectangles of different inner colors and names, and are surrounded by pins that can have names. Those pins can be connected to each other to form a diagram. The following sections will present you an in-depth overview of the diagram editor features.

View Manipulation

The diagram editor uses vector graphics to render its nodes. As a consequence, you can manipulate the diagram view at will and still be able to edit the diagram. 3 view manipulation modes are provided by the diagram editor:

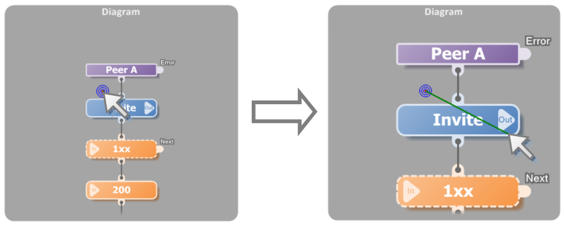
- **Translation**



Translation is performed by pressing the middle button of the mouse and holding it while moving the mouse in the desired direction. A manipulation guide made of blue circles and a green line will show you the manipulation being done. To end the manipulation, release the mouse button.

Note: quick translation can be done by holding the CTRL key and pressing any directional key.

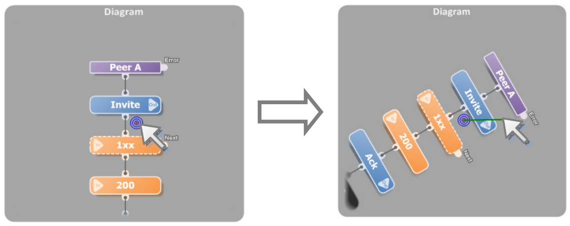
- **Zoom**



Zoom is performed the same way translation is, except you must hold the CTRL key during the process. Only the horizontal mouse displacement modifies the zoom factor.

Note: quick zoom can be done by scrolling the mouse wheel.

- **Rotation**

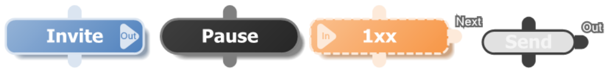


Rotation is performed the same way translation is, except you must hold the SHIFT key during the process. Only the horizontal mouse displacement modifies the rotation angle.

Note: while not as useful as the two other modes, the rotation has been implemented to illustrate the possibilities offered by vector graphics-based editors.

The Diagram Nodes

The nodes in the diagram are round-shaped rectangles that contain text and/or symbols that inform about their meaning. The inner color of a node and the style of its borders (e.g. a solid line or a dashed line) are bound to the semantic of that node.



Examples of nodes of different types

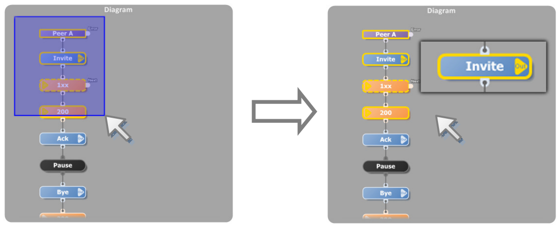
Each node is made of at least the 3 following pieces of information: its location and size, its name, and its pins. Each of them can be changed depending on the action you take and the kind of node considered.

For more information about the different node types and their properties, see the “*Call Flow: the Basics*” and “*Call Flow: Extended*” chapters.

Node Manipulation

The diagram editor allows for various manipulations on nodes. Some of them are only available by using the mouse, while others are accessible through keyboard interaction. You can:

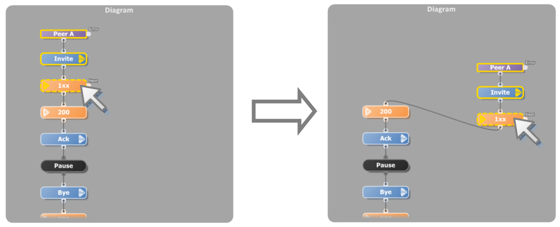
- ### Select nodes



To select nodes on the diagram, you first need to click somewhere on the diagram (neither on a node nor a pin) and hold the left mouse button. As you move the cursor, a blue rectangle is drawn on the diagram, and the nodes it intersects are outlined with a golden bold line, telling you they become selected. To end the selection, release the left mouse button.

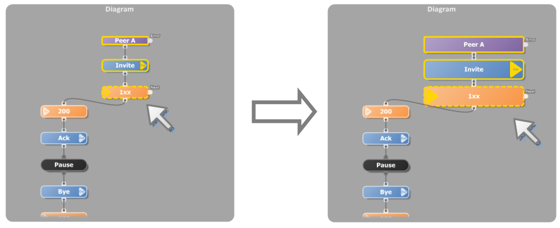
Note: when only one node is selected, you can select nodes around it by pressing any directional key.

- ### Move nodes



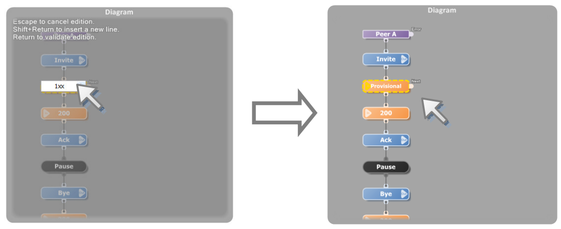
To move selected nodes, click on a selected node and hold the left mouse button. As you move the cursor, the selected nodes move accordingly while still being aligned with an invisible grid that helps you position your nodes. To end moving nodes, release the left mouse button.

- ### Resize nodes



To resize selected nodes, move the cursor around the edge of one of them. When the cursor changes from the regular pointer to a bidirectional arrow, press and hold the left mouse button to start resizing. While moving the cursor around, the selected nodes are resized accordingly, while still being aligned with an invisible grid. To end resizing nodes, release the left mouse button.

- ### Rename nodes

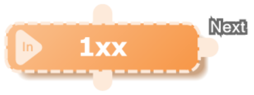

Escape to cancel edition
Shift+Return to insert a new line.
Return to validate edition.

To rename a node, double-click on it with the left mouse button. The editor becomes darker to indicate you that no further modifications can be made to the diagram (like selecting, moving nodes, etc.) while still in this mode, and a text box appears over the node letting you type a new name. You can press ESC to cancel edition, or press ENTER or click anywhere outside the node to validate it.

Note: when only one node is selected, you can rename it by pressing F2.

Node Pins

A node in the diagram can have from zero to many pins around it. A pin can have a name (if so, it is displayed around it) and is either an *input* or *output* pin:



A node with 3 pins: one input pin (top), and two output pins

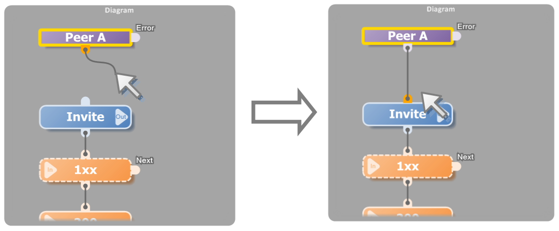
There is no graphical difference between input and output pins. But in their default state, a node will have input pin(s) on the top and left hand side, and output pin(s) on the bottom and right hand side.

Some operations can modify the positions of a pin or the number of pins. They are explained in the paragraph below entitled "*Operations via Contextual Menus*".

Connecting Pins

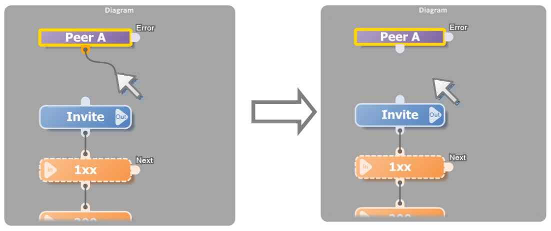
The diagram editor allows you to connect and disconnect the pins between nodes by using the mouse. A pin can only be connected to one other pin, and pins must be of different kinds (input and output).

- **Connect two pins**



To connect two pins, click on one of them with the left mouse button and hold it pressed while moving the cursor to the other pin. During this process, a gray wire will be drawn on the diagram to show you the connection state. When the cursor reaches another pin which connection is possible, the wire will stop moving. To end the connection, release the mouse button over the destination pin.

- **Disconnect two pins**

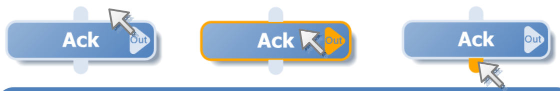


To disconnect two pins, repeat the process above but release the mouse button over the diagram background. This process can be simplified by simply clicking on a connected node and releasing the left mouse button immediately.

Operations via Contextual Menus

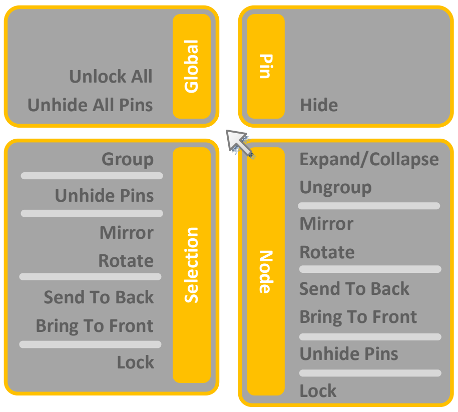
The diagram editor provides a set of contextual menus you can invoke by clicking anywhere on the diagram editor with the right mouse button. The content of those menus varies according to the current state of the selection and the cursor location: depending on what elements are under the cursor, additional menus and actions will be available.

There're two kinds of elements that can affect the contextual menus that way: the nodes and their pins. When the cursor moves over a node or one of its pins, the element will be highlighted in orange to indicate that some additional actions in the contextual menus are made accessible:



Examples of highlighted nodes and pins

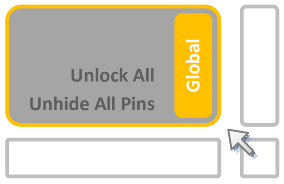
When the menus are invoked, up to 4 contextual windows will pop up around the cursor. Each of them contains actions related to the selection, the node under the cursor, the pin under the cursor, or the whole diagram. The kind of elements they act on is written in an orange margin inside the corresponding pop up window:



The 4 Contextual Menus

A contextual menu will always have the same location relatively to the cursor, and will only be displayed if at least one of its actions is possible in the current editor state. Some of them may not be always available depending on the nature of the highlighted elements and selected nodes.

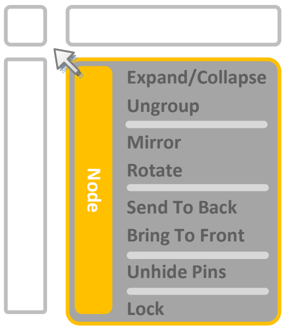
- **The Global menu**



The **Global** menu provides actions that affect the whole diagram. It will always be available, regardless of the highlighted elements and selected nodes.

  - **Unlock All**: unlocks all nodes. To learn more about locking nodes, see the **Lock** command in the **Node** menu.
  - **Unhide All Pins**: unhide all hidden pins. To learn more about hiding pins, see the **Hide Pin** command in the **Pin** menu.

- **The Node menu**

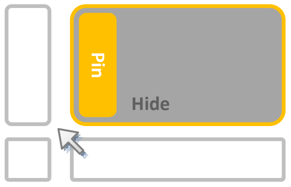


The **Node** menu provides actions that affect the highlighted node if any. It will only be available if the cursor is over an unlocked node when contextual menus are invoked.

Note: if the highlighted node is locked, another menu also named ***Node*** *will be shown instead, providing only the* ***Unlock*** *command.*

  - **Expand/Collapse**: only available if the highlighted node is a *group*; expands or collapses the group so that the elements in it will become hidden or visible. To learn more about groups, see the **Group** command in the **Selection** menu.
  - **Ungroup**: only available if the highlighted node is a *group*; un-groups the nodes contained in the group. To learn more about groups, see the **Group** command in the **Selection** menu.
  - **Mirror**: switches the left and right pins. This command may violate the previously established design rule stating that input pins shall be on the left hand side whereas output pins be placed on the right hand side. Nevertheless, this feature is useful to keep your diagram readable when drawing wires in order to prevent them from crossing other nodes or wires.
  - **Rotate**: this command opens a sub-menu providing either clockwise rotation or counter-clockwise rotation. Those commands will switch the pins on the borders of the node accordingly.
  - **Send to Back**: places the highlighted node under any other node in the diagram. That is, if this node overlaps with another node, it will be drawn under that one.
  - **Bring to Front**: places the highlighted node on top of any other node in the diagram. That is, if this node overlaps with another node, it will be drawn over that one.
  - **Unhide Pins**: shows the previously hidden pins belonging to the highlighted nodes. To learn more about hiding pins, see the **Hide Pin** command in the **Pin** menu.
  - **Lock**: locks the highlighted node. When a node is locked, it cannot be selected nor renamed, moved, etc.

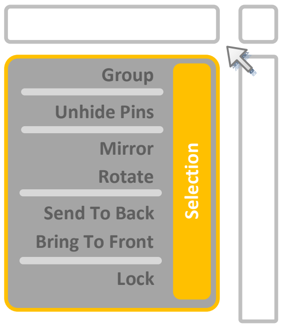
- **The Pin menu**



The **Pin** menu provides actions that affect the highlighted pin if any. It will only be available if the cursor is over a visible pin when contextual menus are invoked.

  - **Hide**: only available if the pin is not connected; hides the pin. When hidden, the pin cannot be highlighted or participate in a connection.

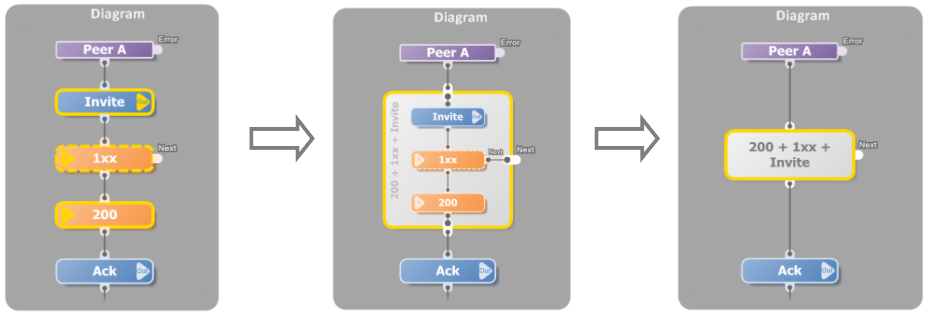
- **The Selection menu**



The **Selection** menu provides actions that affect the selected nodes. It will only be available if there is at least one node that is selected.

Note: take care not to confuse selected nodes and highlighted nodes. Selected nodes have a golden outer glow around them while the highlighted node has a thin orange inner border that disappears when the cursor leaves its surface. The highlighted node may or may not belong to the set of selected nodes.

- **Group**: creates a *group node* that contains the selected nodes and duplicates the pins of the selected nodes that are either not connected at all or connected to nodes that do not belong to the selection:

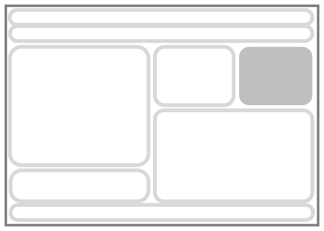


When nodes are grouped, they're considered as one *group* node. The group node will draw inside itself the grouped node whenever it is large enough to do so. If you either collapse it via the **Expand/Collapse** command or resize it to make it smaller, the inside of the node will smoothly fade to a plain text that can be edited like you would edit the name of any other node. This feature allows you to make your diagram compact.

*Note: when in a group, a node cannot be selected, so it becomes impossible to edit its properties. To do so, invoke the contextual menus over the group node and use the **Ungroup** command in the **Node** menu to ungroup the nodes so that you can select the desired node again. To learn more about node properties, see the section entitled "The Property Editor".*

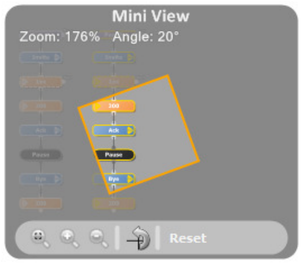
- **Unhide Pins**: shows the previously hidden pins belonging to the selected nodes. To learn more about hiding pins, see the **Hide Pin** command in the **Pin** menu.
- **Mirror**: switches the left and right pins of the selected nodes.
- **Rotate**: opens a sub-menu providing either clockwise rotation or counter-clockwise rotation. Those commands will switch the pins on the borders of the selected nodes accordingly.
- **Send to Back**: places the selected nodes under any other node in the diagram. That is, if these nodes overlap with another node, they will be drawn under that one.
- **Bring to Front**: places the selected nodes on top of any other node in the diagram. That is, if those nodes overlap with another node, they will be drawn over that one.
- **Lock**: locks the selected nodes. When a node is locked, it cannot be selected nor renamed, moved, etc.

The Mini View



The **mini view** displays in the diagram in its entirety and the bounds of the area visible in the diagram editor. It allows you to quickly see which portion of the whole diagram is currently visible and to manipulate the view region.

Quick overview



The mini view in action

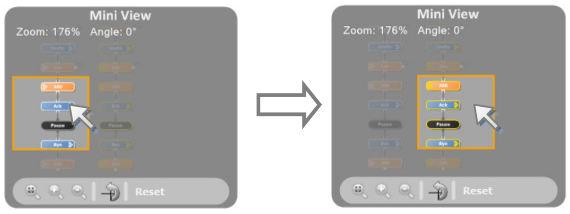
The mini view is made of two main components: a panel that shows you the whole diagram and the current visible area, and a toolbar that provides various command to manipulate the *view region*.

The main panel will always draw the entire diagram, regardless of how large it is, and will darken non visible area while outlining the view region in orange. Whereas the diagram editor displays the diagram after applying to it some transformations (like rotation, zoom, etc.), the mini view will always draw the diagram the same way; instead, the inverse transformation will be applied to the view region, letting you know exactly which part of the diagram you're editing.

View Region Manipulation

The main panel of the mini view allows you to manipulate the view region with the mouse:

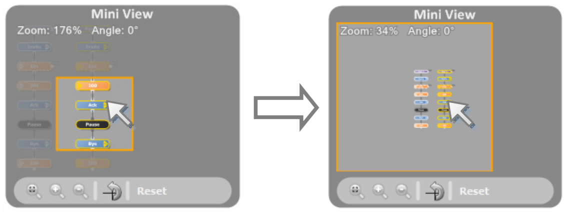
- **Translation**



Translation is performed by pressing the left button of the mouse over the view region and holding it while moving the mouse in the desired direction. The diagram editor will be updated immediately. To end the manipulation, release the mouse button.

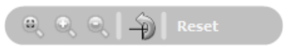
Note: quick translation can be done by clicking on a spot outside of the view region. Doing so will center the view region on that spot.

- **Zoom**



Zoom is performed by scrolling the mouse wheel. The view region will be stretched or shrunk around the cursor location.

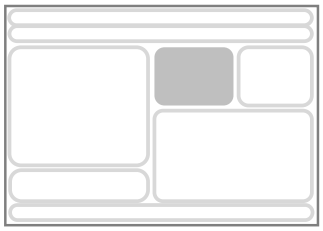
The Mini View Toolbar



The mini view toolbar provides commands that manipulate the view region. The 5 commands are, from left to right:

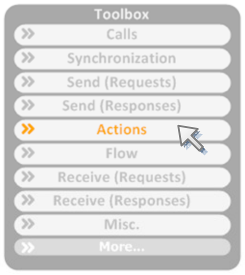
- ***Fit view***: modifies the view region so that the whole diagram becomes visible.
- ***Zoom in***: makes the view region smaller, thus making the diagram look bigger in the editor.
- ***Zoom out***: makes the view region bigger, thus making the diagram look smaller in the editor.
- ***Reset rotation***: resets any rotation transformation performed on the view region.
- ***Reset view***: resets the view to its initial location, size and angle.

The Toolbox



The **toolbox** allows you to add new nodes to your diagrams and store packed groups of nodes for later use – like an intelligent clipboard. The toolbox is made of *libraries* that are named groups of *templates*. A template is a model representing one or several nodes with some connections and properties already set to fulfill a specific task, and can be instantiated any number of times in a diagram.

The Toolbox Libraries

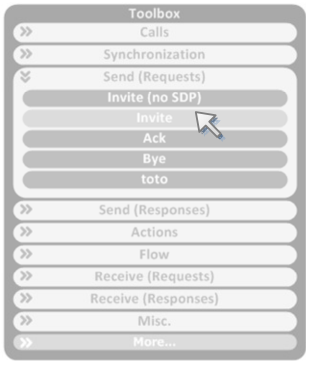


The libraries in the toolbox are represented by gray panels. Those panels can be reorganized with the mouse by dragging them with the left mouse button.

A contextual menu you can invoke by right-clicking on one of them allows you to rename a library or remove it from the toolbox. Deleting a library cannot be undone.

When the application is closed, the toolbox is saved to disk and will be loaded next time. *Note: be careful when you run multiple instances of STS together, as the last one to be closed will be the one to actually save its version of the toolbox.*

The Toolbox Templates



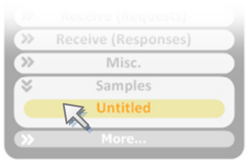
Clicking on the headline of a library panel allows you to expand or collapse it. Each panel contains a list of buttons representing the templates in the library. These buttons can be reorganized with the mouse by dragging them with the left mouse button. They can be moved from a library to another that way.

A contextual menu you can invoke by right-clicking on a template button allows you to rename a template or removing it from its library. Deleting a template cannot be undone.

To instantiate a template, you can either click on the associated button (simple click), or drag the button with the left mouse button and drop it somewhere on the diagram. *Attention: The drag'n'drop mechanism is used for both instantiating and reorganizing.*

When clicking on a template button, if there's only one node selected in the diagram editor, and the template only generates one new node, the system will try to connect and position the new node automatically, allowing for fast diagram composition. When new nodes are created from a template, they become selected in the diagram editor.

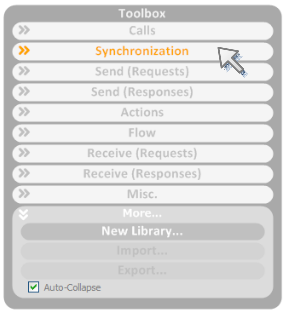
Adding templates



To add templates to the toolbox, select one or more nodes in the diagram editor, hold the SHIFT key, and drag them to the toolbox with the left mouse button. As the cursor drags them over a library panel, this panel will expand and a new button will appear in it. Releasing the mouse button will create a new template button.

Toolbox Options

The special panel at the bottom of the toolbox entitled "More…" is not a library: it is an option panel providing various commands to customize the toolbox content and behavior.



The "New Library" button allows you to add a new empty library to the toolbox. By clicking this button, a new library panel will appear at the end of the toolbox and you will be asked for its name.

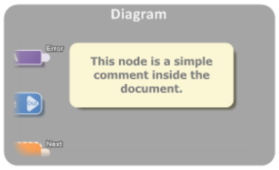
The "Auto-Collapse" option controls how other panels behave when a panel is expanded. If this option is checked, other panels will be automatically collapsed so that only one library panel at a time is expanded.

Note: other commands are unavailable and reserved for future development.

Helper Nodes

The toolbox comes by default with a "Misc." library containing node templates that have no particular purpose regarding call flows. Those nodes are helper nodes which aim at bringing to your diagrams information for the reader/user of the document, and are ignored otherwise. They have no pins and have different forms and colors.

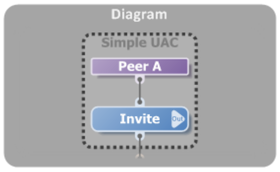
- **Notes**



The *note* allows you to put comments inside your diagram. Their content can be edited like the name of any other node, by double-clicking on it.

Notes are useful to explain things that may be difficult to understand only by seeing the diagram, or that would require the reader to go read the properties of the nodes. It can also be useful to annotate bugs or fixes in shared documents.

- **Delimiters**



A *delimiter* is a semi-transparent node with dash-dot thick borders that are used to mark groups of nodes that are linked together by some semantic that are not represented in the diagram by wires.

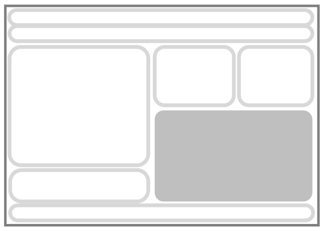
For instance, when two scenarios that are drawn side-by-side are to be communicating with each other via SIP messages, it may become helpful to draw delimiters around the Send/Receive node pairs across the two diagrams, so the reader can easily understand the call flow behavior at runtime.

Note: it can become handy to use the lock/unlock commands available from the diagram editor contextual menus on those kind of nodes so they do not perturb edition.

Note: when you add such nodes to the diagram – especially delimiters – they will be added on top of other nodes. Use the Send to Back/Bring to Front commands from the contextual menus to place them behind other nodes so it becomes easier for you to select them.

To get an idea of the general use of helper nodes, see the 3PCC example next page.

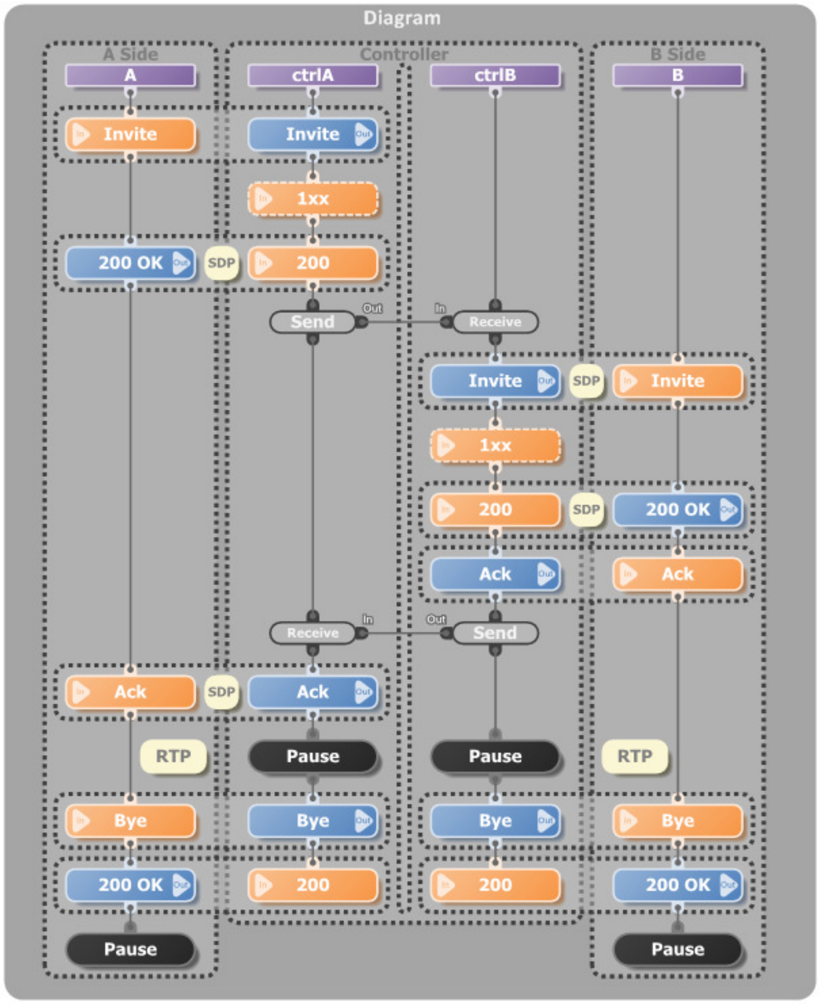
The Property Editor



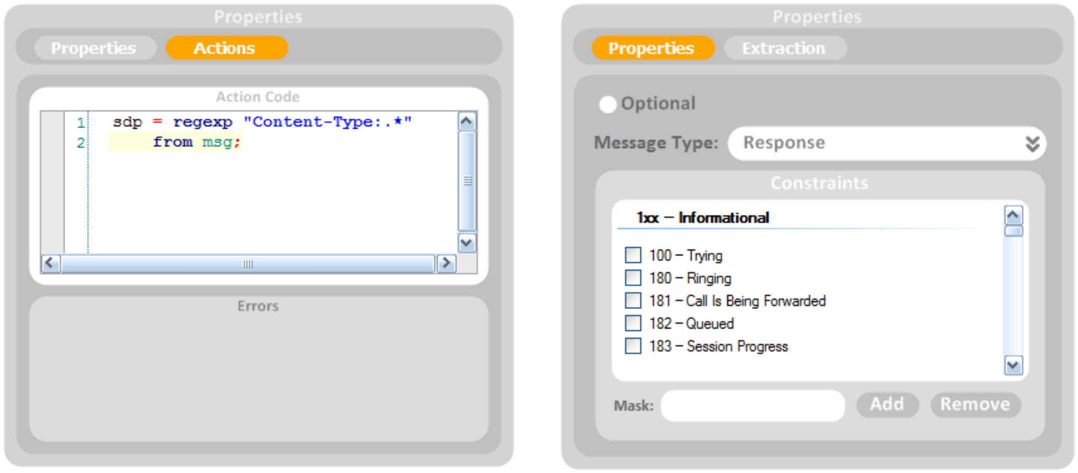
The **property editor** allows you to modify the attributes of a node selected in the diagram editor.

When a node is selected in the diagram editor, the property editor displays one or more tabs providing access to special editors associated with that node. The editors that are displayed are completely dependent on the nature of the node selected and are explained in details in the "Call Flow: the Basics" and "Call Flow: Extended" chapters.

To see examples of property editor tabs, see next page.



Usage of delimiters and notes in the 3PCC example



```
sdp = regexp "Content-Type:.*"
    from msg;
```

Message Type: Response

1xx – Informational

- [ ] 100 – Trying
- [ ] 180 – Ringing
- [ ] 181 – Call Is Being Forwarded
- [ ] 182 – Queued
- [ ] 183 – Session Progress

Examples of property pages

– Chapter 2 –

Call Flow: the Basics

Introduction

Call Flow documents are a set of scenarios made of nodes that are linked together by wires connected to their pins. Each scenario is represented by a **New Call** node which marks the starting point of the scenario, and is made of a chain of nodes, starting with that **new call** node.

The nodes involved in the call flow behavior belong to several categories:

- *Call nodes*
  Those nodes manage call starting and end points. There are two node types inside that category: **New Call** nodes, and **End Call** nodes.
- *SIP message nodes*
  Those nodes manipulate SIP messages, allowing you to either send or receive SIP messages. There are two node types inside that category: **Send** nodes, and **Receive** nodes.
- *Synchronization nodes*
  Those nodes allow synchronization and data exchange between two scenarios. There are two node types inside that category: **Send Signal** nodes, and **Receive Signal** nodes. Synchronization nodes are explained in the next chapter, “Call Flow: Extended”.
- *Action nodes*
  Those nodes allow the application running the scenario (typically SIPP) to perform different tasks aside from sending and receiving SIP messages. There are three node types inside that category: **Pause** nodes, **Log** nodes, and (generic) **Action** nodes. The two latter ones are explained in the next chapter, “Call Flow: Extended”.
- *Flow nodes* (future versions)
  Those nodes will allow you to control the call execution flow by using conditional branching and loops.

Nodes Description

Call Nodes

New Call

New Call
Error

The **New Call** node marks the starting point of a scenario. Any node that is not connected, directly or indirectly, to a **New Call** node, will be ignored on export, and a warning message will be output to the Error panel.

The “Error” output pin represents the execution point that is reached when an unexpected message is received at runtime. For instance, if an ACK is expected at a given point of the scenario, but a BYE is received, the execution point will jump to the node after the “Error” pin, if any.

The **New Call** node has no property pages.

End Call

Success
Failure

The **End Call** node stops the call in its current state. It can mark the call either successful or failed. It has no output pin because no actions are executed in the scenario after this node is reached.

The **End Call** node has one property page allowing you to specify whether the call is successful or has failed.

SIP Message Nodes

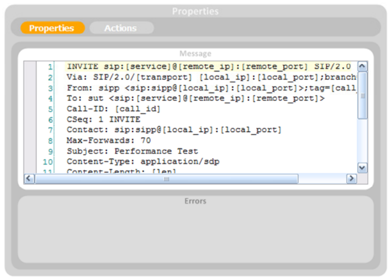
Send

The **Send** node allows you to send a SIP message. This message is specified as a property of the node, and can contain SIPP keywords as well as references to call variables. The **Send** node has two property pages, respectively entitled “Properties” and “Actions”.

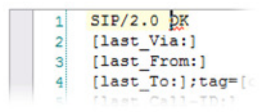
- **The “Properties” page**

This page allows you to type the SIP message to send. It contains two panels: a “Message” panel and an “Errors” panel.

The “Message” panel allows you to edit the SIP message. Its text editor provides several features to make your life easier:

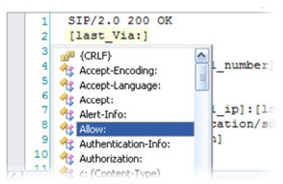


```
INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port];branch
From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call
To: sut <sip:[service]@[remote_ip]:[remote_port]>
Call-ID: [call_id]
CSeq: 1 INVITE
Contact: sip:sipp@[local_ip]:[local_port]
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: [len]
```



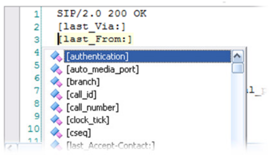
```
SIP/2.0 OK
[last_Via:]
[last_From:]
[last_To:];tag=[
```

- ***SIP syntax checking***: the syntax of your message will be checked as you type it, and errors will be underlined in red and reported in the “Errors” panel.
Note: while errors are reported, it does not forbid you to send invalid SIP messages.



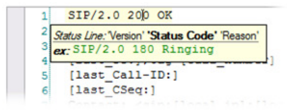
```
SIP/2.0 200 OK
[last_Via:]
```

- ***Code completion***: by pressing the **CTRL + Space** shortcut, a list will be shown under the edition cursor, in which you can navigate by using the **UP** and **DOWN** keys. This list will contain what the editor this is OK to type at that point of the SIP message. If you start typing, the list will update itself to show you whether one of its items begins with what you typed. To close that window, press **ESCAPE**. To insert the entry selected in that list, press **RETURN**.



```
SIP/2.0 200 OK
[last_Via:]
[last_From:]
```

- ***Keyword proposal***: by pressing the **ALT + Space** shortcut, a similar list containing every SIPP keyword will be shown, allowing you to insert one of them.
Note: in order for the editor to analyze the SIP message, keywords are internally replaced by a default value before the message is parsed. For that reason, the editor may actually detect some keywords as errors if it thinks they don’t fit at a specific point.

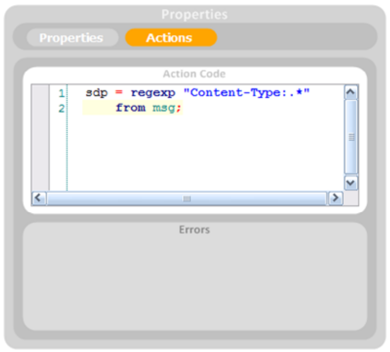


```
SIP/2.0 200 OK
[last_Call-ID:]
[last_CSeq:]
```

- ***Contextual Help***: by pressing the **SHIFT + Space** shortcut, a helper tooltip will be shown under the active line, showing you the expected form of the SIP line you’re editing. If at this point, multiple lines can be typed, the tooltip will show how many possible form there are (e.g. ‘1 of 2’) and will allow you to switch from one to another with the **UP** and **DOWN** keys.

- **The “Actions” page**

This page allows you to type *action code* that will be executed before the message is sent. To learn more about action code, see the next chapter: “Call Flow: Extended”.



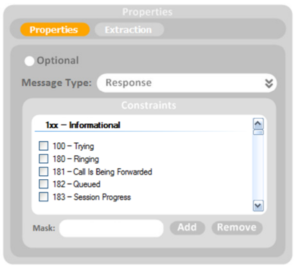
```
sdp = regexp "Content-Type:.*"
    from msg;
```

This page works the same way as the previous one. The “Action Code” panel provides an editor that has the following features: Syntax checking, Code completion, and Contextual help. To learn more about Action code syntax, see the next chapter: “Call Flow: Extended”.

Receive

The **Receive** node allows you to wait for a SIP message. The type of expected message is specified as a property of the node. The **Receive** node has two property pages, respectively entitled "Properties" and "Extraction".

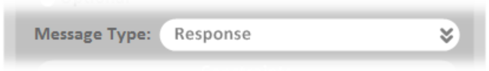
- **The "Properties" page**



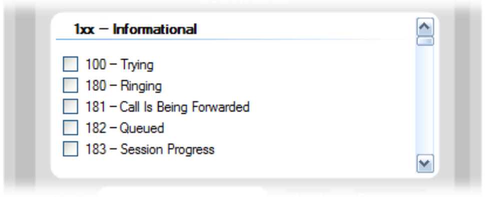
This page allows you to specify the kind of message to receive, check its content and program the behavior of the call flow when such message is received. The property page is split into the following parts:

Optional: when checked, the **Receive** node is optional. That supposes the next node is a non-optional **Receive** node, and means that if one of the specified messages is received while waiting for in next non-optional state, is will not be considered unexpected and will be processed by this node.

When a **Receive** node is marked optional, its borders change from a solid line to a dashed line and a *Next* pin appears. If you connect nodes to that output pin, they will be executed in place of the other nodes connected to the default output pin in case such message is received. If you leave the pin unconnected, the call flow will continue normally.



Message Type: specifies whether the expected message is a *request* or *response* message. When you change this attribute, the content of the *Constraints* panel is modified accordingly.

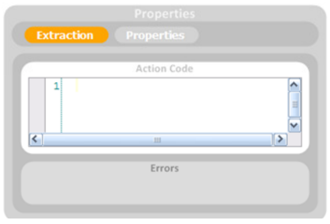


Constraints: specifies the list of expected methods (for a request) or codes (for a response). At least one element must be checked. The semantic of that list changes depending on whether the node is marked optional: if optional, the node will process any message that is of the specified type and contains the specified node or method, and if nodes are connected to the *Next* pin, the call flow will continue in that direction after such message is received; if not optional the node will expect any message that verifies those conditions, and the call flow will only continue when such message is received.

The *Constraints* panel also contains controls that will help you add or remove codes or methods. Their behavior is different whether you expect a *request* or a *response*:

- For a request: the mask must be an upper-case method name, and clicking the **Add** or **Remove** button will check or uncheck that method in the list. If the list contains no such method, it will be automatically added.
- For a response: the mask must contain a 3-character code made of either the X character or a digit (e.g. 200, 21X, 1XX, etc.). If only made of digits, clicking the **Add** button will check the corresponding item in the list, and add it if not present. If not, clicking the **Add** button will check only matching items that are in the list, and will not add any new one. In any case, clicking the **Remove** button will uncheck any matching item in the list.

- **The "Extraction" page**



This page allows you to type *action code* that will be executed after a message is received. To learn more about action code, see the next chapter: "Call Flow: Extended".

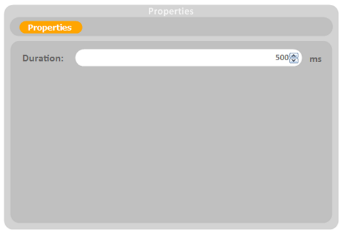
Synchronization Nodes

Those will be presented in the next chapter, “Call Flow: Extended”.

Action Nodes

Pause

The **Pause** node makes the application running scenario (typically SIPP) pause for the specified duration. Its property page allows you to specify that duration in milliseconds.



Duration: 500 ms

Other Nodes

Those will be presented in the next chapter, “Call Flow: Extended”.

Advices & Guidelines

Before starting to edit or create a call flow, keep in mind the following points:

- The toolbox contains by default at least one template for instantiating one of those nodes. If you delete every template that allows you to instantiate a given type of node – STS does not prevent you from doing that – you won’t be able to create it again until you create a new template containing such node.

- The name of a node isn’t linked to its type or content. You are responsible for making the names of your nodes consistent with their role, type and content. For that reason, nothing will warn you if the name of a node becomes inconvenient after you modified one of its attributes (e.g. you instantiated the ‘200 OK’ template from the ‘Receive (Requests)’ template library, changed the expected response codes from the property pages of that node, and forgot to rename it).
  Note: Automatic naming is an optional feature that will be introduced in future versions of STS.

- When you drop nodes to the toolbox, their names and properties are saved in the new template created that way. When you instantiate a template, you are responsible for checking that the properties of the generated nodes are consistent with your actual diagram. The same goes for copy/paste operations.

– Chapter 3 –

Call Flow: Extended

In the next sections the user is supposed to have sufficient knowledge about SIPP scenario semantics such as keywords, command line parameters, SIPP-specific XML nodes and call variables.

Action Code

Action code in STS describes tasks to perform at a given time. For instance, '**Send'** nodes allow you to specify *action code* that is executed before the message is sent, while **'Receive'** nodes use *action code* to extract data from a received message.

Action code also allows you to define variables in a scenario. A variable is a memory storage unit replicated at runtime for each and every call running that scenario – that is, it is present in every of those calls but its value can differ. You will typically use it to store data extracted or computed from received SIP messages and inject it into sent SIP messages.

Language Structure

Action code is a C-like language made of instructions separated by a semicolon. You can type line comments and multi-line comments as in C/C++, and strings are represented by double-quoted strings but without escape sequences.

```
myVar = 1; // This is a line comment.
/* This is a
      multi-line comment */
log "D";
```

Actions are executed in order. They can be split into several categories:

- *Assignment*: actions that assign a new value to a variable.
- *Computation*: actions that assign a value to a variable that is computed from other variables.
- *Commands*: actions that communicate data outside of the application running the scenarios (typically SIPP), like logging, shell command invoking, or PCAP play.

Call Variables

Using a variable is done by typing its name in brackets (e.g. `[SDP]`) if it exists in the same scenario as the one owning the node using it, or typing its name along with the source scenario name otherwise (e.g. `[Controller_A.SDP]`). For example, you can type it in the SIP message of a 'Send' node: before sending the message, this will be replaced by the effective value of the variable.

Note: if the variable belongs to a different scenario, you will need to set a synchronization point between those two that occurs after the variable is set in the source scenario and before it is used in the destination scenario.

Assignment Actions

Basically, an assignment action is of the following form:

```
variable-name = assignment-source ;
```

The `variable-name` part is an identifier representing the name of the variable that will be assigned. It may or may not represent a variable already assigned in this scenario. The variable will belong to the scenario that owns the node defining this *action code* action. The `assignment-source` part represents the source of the value that will be assigned to the variable, and can be any of the following:

Regular Expression Extraction

Regular expression extraction is an assignment source that allows you to extract data from the last received SIP message by using a regular expression. It must be of one of the following form, in which [] represents an optional part:

```
regexp[-i][-m] "a regular expression" [ from source [ at index ] ]
```

- If **`-i`** is specified, the regular expression match will be case insensitive. If **`-m`** is specified, the call will be marked as failed at runtime if the regular expression doesn't match. You can specify both, one, or none, but the order must be preserved.
- The string that follows is the actual regular expression to match.
- If the optional **`from`** block is present, the `source` parameter will specify which part of the SIP message will be searched from. This parameter can be either `msg` to specify the full message (default), `start_line` to specify only the start line, or a string that represents the name of the desired SIP header (e.g. **`"From:"`**).
- If the optional **`from`** block is present and a SIP header name is specified, the optional **`at`** block can help you match the `index` –nth occurrence of that header in the SIP message (1 by default).

Constants

You can specify either an integer or floating point constant value as an assignment source, but also a string that can contain occurrences of other variables or SIPP keywords, such as **`"[field0]"`** or **`"Received SDP: [sdp]"`**. Before assignment, those occurrences will be replaced by the effective value of the variable.

Comparison Result

You can specify a comparison test between a variable and a constant value as an assignment source, like `myCounter > 1`, `myCounter <= 5` or `myCounter == 0.14`, etc. The result of this will be a Boolean value that is the result of the specified test.

Note: tests will be used intensively in future versions of STS when flow nodes will be introduced, such as dynamic branching.

String to Double Conversion

The following assignment source will convert a string variable to a double value:

```
todouble myStringVariable
```

Computation

A computation action is of the following form:

```
variable-name operator right-operand ;
```

This action will compute the operation represented by the `operator` member with `variable-name` and `right-operand` respectively as left and right operands, and store the result in the variable of the current scenario named `variable-name`. `operator` can be any of the following: `+=`, `-=`, `*=`, `/=`. `right-operand` can be either an integer or floating-point constant value, or the name of another variable – which can be either a local variable name or a scenario-qualified name (e.g. `Controller_A.SDP`).

Commands

There are 3 different kinds of command actions available:

Logging Commands

A logging command will output log information using the log mechanism of the application running the scenario (typically SIPP). It must be of the following form, where | represents an alternative spelling:

```
warn|log "your message here" ;
```

You can use either **`warn`** or **`log`** depending on the nature of the message. The following string is the effective message that will be output. That string can also contain occurrences of local variables or keywords that will be replaced.

Execution Commands

The execution command allows you to execute an external shell command from your scenario:

```
exec "shell command" ;
```

PCAP Play Command

The PCAP play command allows you to play PCAP files at runtime:

```
play audio|video "relative path to the file to play" ;
```

To see examples on how to use Action Code in real life scenarios, see the 3PCC and 3PCC-Extended samples that come along STS.

The following section will present you the special nodes that weren't described in the previous chapters.

Extended Nodes

There are 3 types of nodes that are designated as *extended* nodes as they are only used in complex scenarios involving more than a scenario.

Action Code Node

This node has no meaning regarding SIP messages. It only allows you to specify action code that will be executed just before processing the next node.

Synchronization Nodes

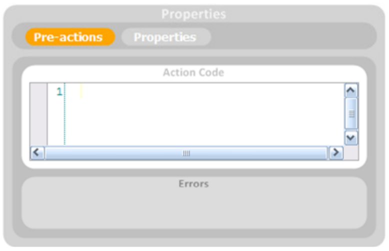
The synchronization nodes work in pair: one **Send Signal** node connected to one **Receive Signal** node. Those nodes encapsulate the behavior of the `SendCmd` and `ReceiveCmd` nodes in SIPP respectively: when a **Send Signal** node is reached in a call, it unblocks the call containing the **Receive Signal** node connected to it and send it a Call-ID parameter along with every variable belonging to that *source* call that will be used by the other *destination* call. And when a **Receive Signal** node is reached in a call, this call will wait until it receives one.

Receive Signal

The **Receive Signal** node has no properties, but has a property tab entitled 'Actions' that allows you to specify action code that will be executed when the expected signal is received.

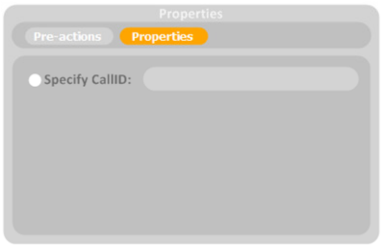
Send Signal

The **Send Signal** node has two property tabs:



- **The "Pre-Actions" page**

  This page allows you to type *action code* that will be executed before the signal is sent.



- **The "Properties" page**

  This page allows you to specify the Call-ID the sent signal will contain. By default, the Call-ID that is sent is the one of the source call. You can have the same behavior by manually specifying `[call_id]` as a Call-ID value.

  If the destination node is the first node of its scenario, a new call will be created with the specified Call-ID.

  Note: if you specify a Call-ID that is different from the one of the source call and want the destination call to answer by a signal at a later time, you will have to follow the next steps:

- Go to the 'Pre-Actions' page of the **Send Signal** node that belongs to the fist scenario `A` and specify:

```
origCID = "[call_id]";
```

- Then in the 'Properties' page of the **Send Signal** node that belongs to the answer scenario `B` and specify the following Call-ID value: `[A.origID]`.