Overview

The Organizational Chart class creates hierarchical charts from data stored in SAS data sets or SCL lists. A chart can be created from any kind of hierarchical data such as an employee database, a file or library directory, project management data, or class hierarchies.

Nodes can display images as well as text. Nodes of the same type can display the same image or each node can display a different image. With an organizational chart you can

- orient the tree from left-to-right or top-to-bottom and display hierarchical trees symmetrically or asymmetrically
- display text horizontally or vertically
- change the appearance of the chart at run-time, including colors, lines, fonts, and node spacing
- print complete charts or subtrees of a chart
- dynamically add items to the pop-up menu.

Parent:

sashelp.fsp.widget.class

Class:

sashelp.fsp.OrgChart.class
Using the Organizational Chart Class

About the Data

In an organizational chart, each piece of information and its relation to every other piece of information is graphically represented. Every piece of information is a node that has a single parent node (except the root node, which is the top or first node in the tree), and zero or more subordinate nodes or children. A node and all the nodes below it are called a tree.

An organizational chart must be created from a data set or SCL list. Each observation in a data set or sublist in a list represents a node in the tree. The data for the chart tell the organizational chart object where to place each node in the tree. To do this, the data must define a node’s position in the tree by providing either

- the level number for every node. A node’s level indicates the depth of the node in the tree; it is determined by the number of nodes directly between the node in question and the root node, inclusive.
- information about the parent of every node.

Specifying a Level Number

Include the level number in a data set variable and map that variable to the node variable, LEVEL. For example, the data set in Output 84.1 on page 1542 contains the variables NODE and LEVEL. The shaded portion of the output includes notes that explain how the value of LEVEL determines the node’s position in the tree:

Output 84.1 Data Set with Level Number

<table>
<thead>
<tr>
<th>NODE</th>
<th>LEVEL</th>
<th>{shcode Node Position}</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>1</td>
<td>{shcode at the tree root}</td>
</tr>
<tr>
<td>EAST</td>
<td>2</td>
<td>{shcode below USA}</td>
</tr>
<tr>
<td>Pat Watson</td>
<td>3</td>
<td>{shcode below EAST}</td>
</tr>
<tr>
<td>Miguel Aguero</td>
<td>4</td>
<td>{shcode below Pat Watson}</td>
</tr>
<tr>
<td>Fred Wong</td>
<td>3</td>
<td>{shcode below Pat Watson and below EAST}</td>
</tr>
<tr>
<td>WEST</td>
<td>2</td>
<td>{shcode below USA and next to EAST}</td>
</tr>
<tr>
<td>Dana Josephs</td>
<td>3</td>
<td>{shcode below WEST}</td>
</tr>
<tr>
<td>Brenda Kratka</td>
<td>3</td>
<td>{shcode below WEST and next to Dana Josephs}</td>
</tr>
</tbody>
</table>

Notice that a child node such as Pat Watson is listed directly after its parent (EAST). Miguel Aguero is a child node of Pat Watson and is listed directly after its parent and before any siblings of the parent, such as Fred Wong.

The mapping list for this data set is

<table>
<thead>
<tr>
<th>Node Information</th>
<th>Data Set Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>=NODE</td>
</tr>
<tr>
<td>LEVEL</td>
<td>=LEVEL</td>
</tr>
<tr>
<td>CURRENT_NODE</td>
<td>=LEVEL</td>
</tr>
</tbody>
</table>

CURRENT_NODE is set in the Mapping List window field Data set variable that identifies current node. It is not a node variable but a list item that determines hierarchy.
Assigning a level number is the fastest way to build a chart because the chart does not need to search to find a node's parent.

**Specifying the Parent Node**

Include a data set variable that describes the node's parent and map that variable to PARENT_NODE. The data set in Output 84.2 on page 1543 includes the same node data as before (and creates the same chart), but instead of the LEVEL variable the data set contains a variable called FROM that contains the value of the node just above it in the tree (its parent). The exception is the root node, which has no parent so none is specified.

Output 84.2  Data Set Specifying Parent Node

<table>
<thead>
<tr>
<th>TEXT</th>
<th>FROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>USA</td>
</tr>
<tr>
<td>EAST</td>
<td>EAST</td>
</tr>
<tr>
<td>Pat Watson</td>
<td>Pat Watson</td>
</tr>
<tr>
<td>Miguel Aguero</td>
<td></td>
</tr>
<tr>
<td>Fred Wong</td>
<td>EAST</td>
</tr>
<tr>
<td>WEST</td>
<td>USA</td>
</tr>
<tr>
<td>Dana Josephs</td>
<td>WEST</td>
</tr>
<tr>
<td>Brenda Kratka</td>
<td>WEST</td>
</tr>
</tbody>
</table>

The mapping list for this data set is

<table>
<thead>
<tr>
<th>Node Information</th>
<th>Data Set Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>= NODE</td>
</tr>
<tr>
<td>CURRENT_NODE</td>
<td>= NODE</td>
</tr>
<tr>
<td>PARENT_NODE</td>
<td>= LEVEL</td>
</tr>
</tbody>
</table>

PARENT_NODE is set in the Mapping List window field called **Data set variable that identifies parent node.** Like CURRENT_NODE it is not a node variable but a list item that determines hierarchy.

The data set in Output 84.3 on page 1543 also determines the chart hierarchy by defining the parent node, in this case the manager’s employee number. In the data set, each observation represents an employee and includes the employee's number as well as the number of the employee’s manager.

Unlike the previous examples, the variables that are displayed as text (TITLE and NAME) are not the same as the variable specifying the current node (EMPNO). It is not necessary to display either PARENT_NODE or CURRENT_NODE.

Output 84.3  Data Set Specifying Two Text Variables

<table>
<thead>
<tr>
<th>TITLE</th>
<th>NAME</th>
<th>EMPNO</th>
<th>MNGRNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>President</td>
<td>Jean DuBois</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>Eastern VP</td>
<td>Laura Stoyavich</td>
<td>0002</td>
<td>0001</td>
</tr>
<tr>
<td>Western VP</td>
<td>Roy Hodges</td>
<td>0005</td>
<td>0001</td>
</tr>
<tr>
<td>Publisher</td>
<td>Pat Watson</td>
<td>0100</td>
<td>0002</td>
</tr>
<tr>
<td>Sales Manager</td>
<td>Fred Wong</td>
<td>0200</td>
<td>0002</td>
</tr>
<tr>
<td>MIS Manager</td>
<td>Dana Josephs</td>
<td>0300</td>
<td>0005</td>
</tr>
<tr>
<td>Security</td>
<td>Brenda Kratka</td>
<td>0400</td>
<td>0005</td>
</tr>
</tbody>
</table>

The mapping list for this data set is
Assigning the two text items displays the employee's name above the title in each node.

Specifying the parent node is less efficient because the chart searches the entire tree every time it processes an observation and positions a node.

**Populating the Organizational Chart**

When populating from a data set, use either the LEVEL node variable or the Parent node and Current node fields in the Mapping List window to indicate the chart hierarchy. (These fields correspond to the PARENT_NODE and CURRENT_NODE mapping list items).

When populating from an SCL list that specifies node variables, use the CHILDREN sublist to specify all the children of a node. This creates the chart hierarchy. For example, this is the contents of an SCL list that produces the same tree as the previous examples:

```<listid>(TEXT = "USA"
    CHILDREN = ((TEXT = "EAST"
        CHILDREN = ((TEXT = "Pat Watson"
            CHILDREN = ((TEXT = "Miguel Aguero")))
            (TEXT = "Fred Wong")))
        (TEXT = "WEST"
            CHILDREN = ((TEXT = "Dana Josephs"
                (TEXT = "Brenda Kratka")))))
```

For an example of populating an organizational chart from an SCL list, see the _repopulate method in this class.

### About Nodes

Each node in an organizational chart is a widget that can display text or an image, or both. In addition, each node can perform actions or run sections of SCL code when selected.

### Node Variables

Every node in an organizational chart has a predefined set of node variables that stores information about the node data it represents and about the node's appearance. Each node in the tree stores information in the variables described in Table 84.1 on page 1545.
## Table 84.1 Node Variables

<table>
<thead>
<tr>
<th>Node Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKGROUND_COLOR</td>
<td>C</td>
<td>the node’s background color</td>
</tr>
<tr>
<td>BORDER_COLOR</td>
<td>C</td>
<td>the node’s border color</td>
</tr>
<tr>
<td>CHILDREN</td>
<td>N</td>
<td>a list of the children of a node. Used only by the _repopulate method.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The _getChildren method conditionally returns the value of this variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the description of these methods for details.</td>
</tr>
<tr>
<td>CLASS</td>
<td>C</td>
<td>the three- or four-level name of the widget class associated with the node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type assigned to the current observation or list item. Used only by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_repopulate method. No Organizational Chart methods return the value of this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable.</td>
</tr>
<tr>
<td>CVALUE</td>
<td>C</td>
<td>an internal character value assigned to the node. Use the Mapping List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>window to assign a character data set variable to the CVALUE variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You can store any text string here.</td>
</tr>
<tr>
<td>FOREGROUND_COLOR</td>
<td>C</td>
<td>the node’s foreground color</td>
</tr>
<tr>
<td>ID</td>
<td>N</td>
<td>a numeric identification number assigned to the node by the user. Use the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mapping List window to assign a numeric data set variable value to the ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable.</td>
</tr>
<tr>
<td>IMAGE</td>
<td>C</td>
<td>the name of an image:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For type 3 nodes, specify a one-level name of an IMAGE entry in the default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>image icon catalog, or a three- or four-level name of a specific IMAGE entry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For type 2 nodes, specify either a host file name or a three- or four-level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>catalog entry name. (See also the Image class.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A partial image name if the Name field in the Text and Image window</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifies a prefix or mask.</td>
</tr>
<tr>
<td>LEVEL</td>
<td>N</td>
<td>a number indicating the level of the node in the tree</td>
</tr>
<tr>
<td>LINE_COLOR</td>
<td>C</td>
<td>the color of the line connecting the node to its parent</td>
</tr>
<tr>
<td>NODEID</td>
<td>N</td>
<td>a unique identification number assigned to the node by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>organizational chart object. The user cannot set NODEID.</td>
</tr>
<tr>
<td>NVALUE</td>
<td>N</td>
<td>an internal numeric value assigned to the node. Use the Mapping List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>window to assign a numeric data set variable to the NVALUE variable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>You can store any number here.</td>
</tr>
<tr>
<td>OBS</td>
<td>N</td>
<td>the number of the SAS data set observation from which the node was created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The OBS value may not match the true observation number if a WHERE clause</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was active when the organizational chart read in the data set, or if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>node variable OBS is mapped to a different data set variable (not OBS) in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the SAS data set.</td>
</tr>
<tr>
<td>OWNER</td>
<td>N</td>
<td>the numeric identifier of the organizational chart that is automatically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>passed to the node’s _init, _postinit, and _select methods. You can also</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set this value by calling the _getOwner method after calling CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUPER (’SELF’, ’_init’). This value cannot be retrieved by any other</td>
</tr>
<tr>
<td></td>
<td></td>
<td>method.</td>
</tr>
<tr>
<td>TEXT</td>
<td>C</td>
<td>the text of the node; each line of text is a separate TEXT item</td>
</tr>
</tbody>
</table>
### Node Identifier

Organizational Chart methods identify nodes by the numeric node identifier (NODEID) instead of the object identifier (WIDGETID) because during scrolling, the nodes are removed from the frame when they are not visible, rendering the object identifier invalid. However, the organizational chart node identifier is always valid. Therefore, it is best to use only Organizational Chart methods to modify the appearance of a node so that you can scroll a widget off and back onto the display and the widget retains its appearance.

As an alternative, you may subclass a node widget and run other methods in the node's _init or _postinit method. Then if a node is scrolled out of viewing area and removed, when it is scrolled back into view, its _init and _postinit methods run and cause it to reappear.

### Node Types

Nodes are separated into types. Each type has general region and object settings such as region borders, colors, fonts, line widths, and action upon selection. Table 84.2 on page 1546 describes the predefined node types.

#### Table 84.2   Node Types

<table>
<thead>
<tr>
<th>Node Type Index</th>
<th>Description</th>
<th>Widget Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>graphics text only (subclass of the Extended Text Entry class)</td>
<td>SASHELP.FSP.ORGTEXT.CLASS</td>
</tr>
<tr>
<td>2</td>
<td>images only (subclass of the Image class)</td>
<td>SASHELP.FSP.ORGIMAG.CLASS</td>
</tr>
<tr>
<td>3</td>
<td>images with text (subclass of the Image Icon class)</td>
<td>SASHELP.FSP.ORGIMGI.CLASS</td>
</tr>
</tbody>
</table>

The general settings of these types can be changed, but the widget class, node type number, and description cannot be changed. You can create new node types with different colors, fonts, region borders, and so on. from the same widget class. Specify new node types in the Attributes window.

Because general node characteristics are specified in the Attributes window, several different node types can be of the same class, but you can change the default images, fonts, colors, and so forth by the numeric node type without creating more classes. To create a new node widget class, subclass one of the three predefined widget classes in

---

**Table 84.2**

<table>
<thead>
<tr>
<th>Node Type Index</th>
<th>Description</th>
<th>Widget Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>graphics text only (subclass of the Extended Text Entry class)</td>
<td>SASHELP.FSP.ORGTEXT.CLASS</td>
</tr>
<tr>
<td>2</td>
<td>images only (subclass of the Image class)</td>
<td>SASHELP.FSP.ORGIMAG.CLASS</td>
</tr>
<tr>
<td>3</td>
<td>images with text (subclass of the Image Icon class)</td>
<td>SASHELP.FSP.ORGIMGI.CLASS</td>
</tr>
</tbody>
</table>
Table 84.2 on page 1546 or create an entirely new widget class. For more information on subclassing, see “Creating a New Node Type” on page 1550.

The three predefined organizational chart widget classes only override the _init and _binit methods of the base SASHELP.FSP.WIDGET classes they were derived from. In addition, they each define a _setFcolor method the Organizational Chart class calls to set the foreground color of a widget. For graphics text objects (node type 1) and image icons (node type 3), this method sets text color. For image objects (node type 2), _setFcolor has no effect.

The nodes in a chart can be different types, and nodes can be created from any of the following:
- one of the Organizational Chart’s predefined widget classes
- a subclass of one of the predefined widget classes
- a completely new class.

Creating a Sample Organizational Chart

Two sample data sets are provided with the class and can be selected from the Data set field of the Attributes window. To create and test an organizational chart from one of the sample data sets, follow these steps:
1. Open a frame and create an organizational chart object.
2. In the Attributes window, select WORK.ORGSAMP1 from the list displayed by the down arrow in the Data set field. Change both horizontal and vertical node spacing to 32 and turn off Display chart at run time only.
3. Open the Node Appearance window and change Border to Simple.
4. Open the Select Action window and turn on Hide/unhide children upon double click.
5. Return to the frame and run TESTAF.
6. Double click nodes to hide and unhide them. Change the background color of the nodes by selecting Attributes from the pop-up menu. Then open the Node Appearance window and choose a color for Background. Return to the Testaf window in which the nodes display the new color.
7. Close the Testaf window and return to the frame. Note that the nodes retain their original background color.

Instance Variables

Selected instance variables that can be used with an organizational chart widget are described here. Instance variables are typically modified with methods or through the Attributes window. For more information on instance variables, see SAS/AF online help.

You can use the appropriate GETNITEMx() function to retrieve these values from the organizational chart widget identifier. If you set these instance variables with the appropriate SETNITEMx() function, you should make this method call:

```call send (orgid, '_set_option_', 'iv_changed');```

This statement notifies the organizational chart that one or more instance variables were set directly by the user.

For an example of using instance variables to control the run-time menu, see “Controlling the Pop-up Menu with Instance Variables” on page 1551.
DATASET

names the data set that is read during the _repopulate method, if a data set identifier is not given. This value is automatically obtained from the Attributes window.

KILLMENU

controls the organizational chart's run-time menu. If the value is 0 or missing, the menu is available. Otherwise, the menu is unavailable. This variable has no effect on user pop-up options set with the PREPOP and POSTPOP instance variables.

 isSelected

contains the node identifier of the selected node before the current node. This value is cleared if the node is released during a _repopulate method call.

MAPLIST

contains the list of node variables that have been mapped to data set variables. Every item in the list is a character item and should have a valid name. Valid names are listed in Table 84.1 on page 1545. This list is automatically built from values entered in the Mapping List window but can be passed into the _repopulate method.

The character values for the mapping list items are the data set variable names that specify that particular attribute. For example, an item named TEXT with a value of OFFICE means the values in the data set variable OFFICE specify the text to display for each node.

If PARENT_NODE is not in the list, or is blank, you must map the CURRENT_NODE variable to the node level number as well as the LEVEL variable.

You do not have to specify all mapping list variables. The simplest mapping lists would specify TEXT and CURRENT_NODE if level numbers are used, or TEXT, CURRENT_NODE, and PARENT_NODE if level numbers are not used. For examples, see “About the Data” on page 1542.

The PARENT_NODE and the CURRENT_NODE must map to the same variable type. That is, both must be numeric, or both must be character.

_popid

specifies the node identification number of the node the _popup method is running for. It is 0 unless you override _childPopup.

POPMENU_ITEM

returns the item number in the user-defined POPMENU.LIST of the selected item. Valid only in POSTPOP.

POPMENU_GRAYLIST

specifies a list of pop-up menu character values to be grayed out. These values can refer to:

- items in the default organizational chart run-time pop-up menu.
- user defined menu items
- menu items inherited from an object that contains the organizational chart.

For example, specify "Print..." to gray that menu item. Set this list during the PREPOP label/method. An item that does not exist in the menu is ignored. See also the PREPOP instance variable.
POPMENU_LIST

specifies a list of items to add to the end of the run-time pop-up menu. Items marked INACTIVE via the SETLATTR() function are grayed in the pop-up menu.

POSTPOP

defines a labeled section of SCL code or a method to run if the user selects one of the user-defined options set up by the code specified by the PREPOP variable. Values can be a one-level SCL label or the five-level name of a method from another SCL program (for example, WORK.A.A.SCL:MYPOP).

PREPOP

defines a labeled section of SCL code or a method to run before the run-time pop-up menu displays. Values can be a one-level SCL label or the five-level name of a method from another SCL program.

You may add selections onto the end of the menu by defining an SCL list of options and assigning the instance variable POPMENU_LIST to that list.

_rootid

contains the node identification number of the root node, typically the title node. If there is no tree, this value is 0. The value of _rootid changes if the title is hidden or changes, or if the chart is repopulated. Therefore, each time you need this value, you should acquire it by calling the SCL function GETNITEMN(). Do not store this value.

_Selected

contains the node identification number of the most recently selected node in the tree. This value is cleared if the node is released during a _repopulate method call.

---

**Specifying Images for Nodes**

When you use a node type that includes an image, you can choose whether the nodes display a standard image (the same image for every node of that type) or a custom image (the image depends on the content of the node).

If you want every node of a particular type to display the same image, assign the image in the Text and Image window. In this case you explicitly name the image using either the path name of an external file or the name of an IMAGE catalog entry.

If you want every node of a particular type to display a different image whose name is specified in a variable in the data set that populates the chart, you can do it in two ways:

1. Store the name and location of the image associated with each node in a variable in the data set. In the Mapping List window, map the data set variable, for example PICTURES, to the node variable IMAGE. In this case, the data set variable must contain the exact location of the image, either the complete catalog entry or full file path name.

   If you want to assign a default image to display if the image in the data set is missing, specify that image in the Text and Image window and set Node Image Names to Override Name.

2. If the data set does not include the complete location of the image, but has instead only the image name or some part of the name, you can use either a prefix or a mask to build an image reference from that information.
For example, suppose you are populating a chart using both a data set of employee information and a catalog of employee photographs. The data set includes the employee identification number which is also used as the name of the catalog entry containing the employee's photograph.

To use a prefix, in the **Name** field of the Text and Image window, specify the library and catalog containing the photographs and set **Node Image Names** to **Append to Name**. In the Mapping List window map the data set variable, for example **EMPNUM**, to the node variable **IMAGE**.

Using a mask is similar except that instead of specifying a prefix, you specify the complete location using an asterisk (*) where the value of the data set variable should be substituted. For example, if the employee photographs are stored in a UNIX environment in a directory with filenames using the pattern **pxxx.xxx.photo.gif**, where **xxxx** is the employee number, you could specify a pathname such as **/mydir/p*.photo.gif**, set **Node Image Names** to **Override Name**, and map **EMPNUM** to **IMAGE**. When the chart is populated, the employee number is inserted into the mask to build the correct pathname.

---

**Creating a New Node Type**

If you subclass one of the predefined node types to create a new widget class, you need to override the default **_init** and **_binit** methods and initialize the widget yourself from information passed to you from the organizational chart object. You may also override the default **_select** method to have whatever you want happen upon node selection.

In the **_init** method for your own defined class, the instance variable **NODEDATA** is passed to you from the **_SELF_** list. **NODEDATA** is a list containing the information the organizational chart object has about the node it received at population time. This list is only valid during the **_init** and **_binit** methods. The format of the list is the same as the format of the returned list in the **_getChildren** method. The only difference is that the list items **WIDGETID** and **CHILDREN** are omitted.

It is up to you to set the proper instance variables for that widget class and perform a **CALL SUPER** method call if needed. If you subclass an Organizational Chart class and you do not wish the organizational chart to override your initialization settings during the **CALL SUPER**, you can set the numeric instance variable **INITED** to 1 on the **NODEDATA** list before calling **CALL SUPER**.

This is an example of the **_init** method for a user-defined widget class that will be used as a node. This code performs self initialization before **CALL SUPER**:

```c
INIT: METHOD;
    ndinfo = getniteml(_self_,'_nodedata');
    text   = getnitemc(ndinfo,'text');
    value  = getnitemn(ndinfo,'nvalue');
    rc = setnitemc(_self_,text, 'mytext');
    rc = setnitemn(_self_,value,'myvalue');
    rc = setnitemn(ndinfo,1,'inited');
    call super(_self_,_method_);
ENDMETHOD;
```

The instance variable named **_nodeid** can be accessed in the **SELECT** label or **SELECT METHOD** for that widget and used for any appropriate Organizational Chart class methods you wish to call. This is true no matter what widget class you are using.
Controlling the Pop-up Menu with Instance Variables

This example shows how you can customize the run-time pop-up menu by adding items or graying items. It shows how to do this without having to subclass the organizational chart or node classes, and without having to create additional SCL programs to handle pop-up menu methods.

In addition, it shows how to use the _snapshot method (Widget class) to take a "picture" of the organizational chart and save it as an IMAGE catalog entry (see the POSTPOPL label in the example). The saved image can then be displayed with the Image or Image Icon class in any Frame. You can also change the size of the image when it is displayed with these classes.

Other features of the program include:
- testing if the popmenu was selected when the cursor was over a node
- cleanly and safely creating and deleting the pop-up menu lists.

This is how the program works. When the user opens the run-time pop-up menu, the organizational chart object first looks to see if the application specifies a PREPOP label or method. If so, the object runs that label to see if the application wants to gray or add anything to the menu. In this case, it adds two items: Edit and Take snapshot.

Then it tests to see if the user clicked on a node. If so, it grays out the application-defined item Edit by making it INACTIVE. It also grays out the default menu Attribute by putting it on a 'graylist'.

If the user makes a menu selection that is defined by the application, the POSTPOPL section runs. In this program POSTPOPL defines the action to take when the user selects either Edit or Take snapshot. If the selection is not an application defined item, the POSTPOPL section does not run.

```c
/* Get the widget ID for the chart and tell */
/* the object to run PREPOP and POSTPOP sections. */
/* Create the lists. */
*/
INIT:
    call send(_frame_,"_getWidget",
             "ORGCHART",orgid);
    rc = setnitemc(orgid,"PREPOPL",
                   "PREPOP");
    rc = setnitemc(orgid,"POSTPOPL",
                   "POSTPOP");
    graylist = makelist(0);
    popmlist = makelist(0);
    return;

    /* After the program has run, free the lists. */
TERM:
    rc = dellist(graylist,"Y");
    rc = dellist(popmlist,"Y");
    rc = setniteml(orgid,0,
                   "POPMENU_GRAYLIST");
    rc = setniteml(orgid,0,
                   "POPMENU_LIST");
    return;
```
/* When the menu is opened, add two
items to the menu. */

PREPOPL:
rc = clearlist(graylist,"Y");
rc = clearlist(popmlist,"Y");
rc = insertc(popmlist,"Take snapshot",-1);
rc = insertc(popmlist,"Edit...",-1);

/* Test if the cursor was over a
node when */
/* the user clicked to open the
menu. */
popid = getnitemn(orgid,"_popid",1,1,0);

/* If not on a node, gray the
application-defined 'Edit...' */
/* menu item and the default
'Attributes'
menu item. */
if (popid eq 0) then
rc = setlattr(popmlist,"INACTIVE", -1);
rc = insertc(graylist,"Attributes...", -1);
rc = setniteml(orgid,graylist,
"POPMENU_GRAYLIST");
rc = setniteml(orgid,popmlist,
"POPMENU_LIST");
return;

/* If the user selected one of the
items defined */
/* in PREPOPL, this section runs. */
POSTPOPL:
item = getnitemn(orgid,"POPMENU_ITEM");
select (item);
when (1); /* SNAPSHOT */
fname = "WORK.A.SNAPSHOT.IMAGE";
imgdatid = instance(loadclass("SASHELP.
PSP.IMGDAT.CLASS"));
call send(orgid,"_snapshot",imgdatid,
rc,"MAIN");
call send(imgdatid,"_writeCatalog", fname);
call send(imgdatid,"_term");
_msg = "NOTE: Snapshot saved as"
 || fname;
when (2); /* EDIT */
call display("MYEDITOR.FRAME", orgid,popid);
end; /* select */
return;
Methods

Methods specific to the organizational chart class are described here. Inherited methods are described in the Object Class and the Widget class.

Note: For methods that return an SCL list identifier, if you specify 0 or a missing value as the list identifier, the list is automatically created. If you specify the identifier of an existing list, the list is cleared before the method adds items to it.

Dictionary

 Collapse

Hides all levels of children of a node

Syntax

CALL NOTIFY (orgchart-name, '_collapse', node-id);

Argument | Type | Description
--- | --- | ---
node-id | N | specifies the numeric identifier of the node

Details

The frame objects themselves are actually removed from the frame if they existed prior to this method call. This method has no effect if the node passed in is hidden itself.

You can specify this method as the default action for a double click a node with children by choosing Hide/unhide children upon double click in the Select Action window.

Example

This example hides all the children of a node when it is selected:

ORG1:

```plaintext
  call notify('org1','_get_selected_',
    widgetid,nodeid);
  if (nodeid eq 0) then return;
  call notify('org1','_collapse_',nodeid);
  return;
```
_deleteNode

Deletes the current node, and all of its children if they exist, from the tree.

Syntax

CALL NOTIFY (orgchart-name, '_deleteNode', node-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node to be deleted</td>
</tr>
</tbody>
</table>

Details

If you have stored numeric node identifiers returned as a result of the _search or _getSelected methods, or by other means, these identifiers may become invalid if you remove the associated nodes using the _deleteNode or _repopulate method.

Examples

Remove the selected node and its children from the tree.

```plaintext
call notify('ORG', '_getSelected', widgetid,nodeid);
if (nodeid eq 0) then return;
call notify('ORG', '_deleteNode', nodeid);
```

_expand

Displays all the direct children of a node

Syntax

CALL NOTIFY (orgchart-name, '_expand', node-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
</tbody>
</table>

Details

Even if all child nodes are displayed, they may not all appear in the current viewing area. This method has no effect if the node passed in is hidden itself.
Example

This example shows all of the children of the node when it is selected:

```plaintext
ORG1:
    call notify('org1','_get_selected_', widgetid,nodeid);
    if (nodeid eq 0) then return;
    call notify('org1','_expand_',nodeid);
    return;
```

_getChildren

Returns a list containing all of a node’s direct children, or all levels of children, grandchildren, and so forth down the tree.

Syntax

CALL NOTIFY (orgchart-name, '_getChildren', node-id, list-of-names, list-id<, all-levels>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the names of the node variables whose values are to be returned in list-id. Separate multiple node variable names with a space. Specify ‘ALL ‘ to return the value of all variables. You can include these options in the string: /F return the number of direct children. If both /F and all-levels are specified, all-levels is ignored. /R return lists in reverse order.</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>returns the identifier of an SCL list containing a sublist for each child of the specified node. The length of the list is the number of children the specified node has. Each sublist contains the values requested in list-of-names for the child node. If list-of-names is ‘/F ‘, then list-id returns the actual number of child nodes instead of a list identifier.</td>
</tr>
<tr>
<td>all-levels</td>
<td>C</td>
<td>specifies whether to return all levels of node children: only direct children (the default) all levels of children</td>
</tr>
</tbody>
</table>

Details

Although the method automatically creates list-id if its value is missing or 0, you must delete it with the DELLIST function after you have finished with it.
Example

This example retrieves the displayed text, character value, identification number assigned by the user, and observation number of each direct child of the selected node and stores it in OUTLIST.

This code must be in a labeled section of the SCL program associated with the frame. The labeled section can be either the organizational chart's labeled section (in this case the section labeled ORG1), or a labeled section specified in the Select Action window:

```scl
ORG1:
/* Get the identifiers of the organizational chart object */
/* and of the selected node */
call notify ('org1', '_get_selected_',
           widgetid, nodeid);
if (nodeid eq 0) then return;
/* Specify the items whose values you want to retrieve */
buff='text cvalue id obs';
/* Get the values from the child nodes and return them in OUTLIST*/
call notify ('org1', '_get_children_',
           nodeid, buff, outlist);
return;
```

GetCurrent

Returns a list containing information for the specified node

Syntax

CALL NOTIFY (orgchart-name '_getCurrent', node-id, list-of-names, list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the names of the node variables whose values are to be returned in list-id. Separate multiple node variable names with a space. Specify 'ALL' to return the value of all variables.</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>returns the identifier of an SCL list containing the values of items specified in list-of-names</td>
</tr>
</tbody>
</table>

Example

In the following example, MYLIST contains a list of the displayed text and the node's image name. Assume this code is a method specified in the Select Action window:
GET: METHOD;
    call send (_self_, '_get_owner_', orgid);
nodeid=getnitemn (_self_, '_nodeid_');
call send (orgid, '_get_current_', nodeid,
    'text type', mylist);
text=getnitemc (mylist, 'text');
endmethod;

_getLevel

Returns a list containing all of the nodes in the entire tree that are at the given node’s level

Syntax

CALL NOTIFY (orgchart-name, '_getLevel', node-id, list-of-names, list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the names of the node variables whose values are to be returned in list-id. Separate multiple node variable names with a space. Specify ‘ALL’ to return the value of all variables.</td>
</tr>
<tr>
<td>list-id</td>
<td>I</td>
<td>returns the identifier of an SCL list containing a sublist for each node in the tree at the same level as node-id. Each sublist contains the values of items specified in list-of-names.</td>
</tr>
</tbody>
</table>

Example

In this example, MYLIST contains a list of lists of the displayed text of all nodes in the entire tree that are at the same level as the selected node. Assume this code is in a labeled section specified in the Select Action window:

call notify('org1', '_get_selected_',
    widgetid,nodeid);
if (nodeid eq 0) then return;
call notify('org1', '_get_level_',
    nodeid,'text',mylist);

_getParent

Returns a list containing either the direct parent of a node or all of the node’s parents up to the root of the specified node
Syntax

CALL NOTIFY (orgchart-name, '_getParent', node-id, list-of-names, list-id,<, all-levels>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted string containing the names of the node variables whose values are to be returned in list-id. Separate multiple node variable names with a space. Specify 'ALL' to return the value of all variables.</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>returns the identifier of an SCL list containing the values of items specified in list-of-names or a list of lists if all-levels is 'Y'.</td>
</tr>
<tr>
<td>all-levels</td>
<td>C</td>
<td>specifies whether to return all levels of node parents:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' only the direct parent (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' all levels of parents (ancestors)</td>
</tr>
</tbody>
</table>

Example

In this example, MYLIST returns a list of lists containing the displayed text of all of the node's ancestors. Assume this code is in a labeled section specified in the Select Action window:

```scl
call notify('org1','_get_selected_','
   widgetid,nodeid);
if (nodeid eq 0) then return;
call notify('org1','_get_parent_','
   nodeid,
   'text',mylist,'y');
```

_getSelected

Returns the object identifier and optionally, the node identifier, of the previously selected node

Syntax

CALL NOTIFY (orgchart-name, '_getSelected', widget-id,<, node-id>);
### widget-id
N
returns the numeric, object identifier of the widget

### node-id
N
returns the numeric identifier of the node

## Details

Use this method only in the labeled section of the SCL program that runs when a node is selected.

Another way to retrieve `node-id` is from the `_SELF_ list. Use this statement in the `_select` method of a widget or node instead of calling the `_getSelected` method:

```
nodeid = getnitemn(_self_, '_nodeid_');
```

If a click occurred anywhere other than directly on a node, `widget-id` and `node-id` return 0. Be sure to check for 0 before calling other methods with `node-id`.

---

### _gotoNode

Places the specified node in the viewable area

---

## Syntax

CALL NOTIFY (orgchart-name, '_gotoNode', node-id);

### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node to go to; if the node identifier is invalid, no action is taken.</td>
</tr>
</tbody>
</table>

## Example

This example displays the node containing the text string 'Smith':

```java
/* Make a list. */
mylist = makelist(0);

/* Add 'Smith' to the list. */
mylist = insertc(mylist,'Smith',1,'text');

/* Use the _search method and MYLIST to find */
/* the node containing 'Smith'. */
call notify('org1','_search_','nodeid','text /s',mylist);

/* Delete the list. */
rc = dellist(mylist);
```
/* If a valid node has been found, display it. */
if (node-id ne 0) then
    call notify('org1', '_goto_node_', nodeid);
else
    _msg_ = 'Node not found.'

_isExpanded
Reports whether a node’s children are currently visible

Syntax
CALL NOTIFY (orgchart-name, '_isExpanded', node-id, status);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>status</td>
<td>N</td>
<td>returns a value indicating whether the children are visible or hidden:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 hidden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 visible</td>
</tr>
</tbody>
</table>

Example
This example reports whether the children of a node are displayed:
ORG1:
call notify ('org1', '_get_selected_',
   widgetid, nodeid);
if (nodeid = 0) then return;
call notify ('org1', '_is_expanded_',
   nodeid, expanded);
if expanded then _msg_ = "The node’s
children are expanded";
return;

_print

Prints the current chart, or part of the current chart, and optionally opens the Print window before
sending the chart to the printer

Syntax

CALL NOTIFY (orgchart-name, '_print'<, print-list>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print-list</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the items in Table 84.3 on page 1561. A missing value for any numeric variable is interpreted as 0.</td>
</tr>
</tbody>
</table>

Table 84.3 _print List Items

<table>
<thead>
<tr>
<th>List Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'BYPAGE'</td>
<td>N</td>
<td>a value controlling how the nodes are printed. To print all subtrees that are at the same level as the specified node, one per page, specify a nonzero value for BYPAGE and a value for NODEID.</td>
</tr>
<tr>
<td>'DEVICED'</td>
<td>C</td>
<td>the description of the print device. If you do not set this value, successive calls use the value specified in the Print window.</td>
</tr>
<tr>
<td>'DEVICEN'</td>
<td>C</td>
<td>the name of the device to print to. If you do not set this value, successive calls use the value specified in the Print window.</td>
</tr>
<tr>
<td>'LEVELS'</td>
<td>N</td>
<td>the number of tree levels down from the node where printing begins. Specify 0 to print all levels.</td>
</tr>
<tr>
<td>'NODEID'</td>
<td>N</td>
<td>the numeric identifier of the node you wish to begin printing from. If omitted, printing begins at the root of the tree.</td>
</tr>
<tr>
<td>List Item Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><code>'SETUP'</code> N</td>
<td>a value controlling whether the Print window opens:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not open</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0= do not open</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0= open (default)</td>
<td></td>
</tr>
<tr>
<td><code>'SWAPBW'</code> N</td>
<td>a value controlling whether to reverse black and white color interpretation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: do not reverse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0= do not reverse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0= reverse</td>
<td></td>
</tr>
</tbody>
</table>

**Details**

If no list is passed in, the options from the previous call to `_printSetup` are used. If these options have not been set, the options window will come up automatically.

**Example**

This example prints the tree from the last selected node. It also causes the Print window to open. Assigning 0 to SETUP would bypass the Print window.

```plaintext
call notify('org1', '_get_selected_', widgetid, nodeid);
prlist = makelist(0);
prlist = setnitemn(prlist, nodeid, 'nodeid');
/* <= 0 if none clicked on */
prlist = setnitemn(prlist, 1, 'setup');
call notify('org1', '_print_', prlist);
rc = dellist(prlist);
```

**_printSetup**

Defines the printing options and optionally sends the current chart, or part of the current chart, to the printer.

**Syntax**

CALL NOTIFY (orgchart-name, '_printSetup', print-list);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print-list N</td>
<td></td>
<td>specifies the identifier of an SCL list containing the items in Table 84.3 on page 1561. A missing value for any numeric variables is interpreted as 0.</td>
</tr>
</tbody>
</table>

**Details**

The _printSetup method ignores the SETUP list item.
All organizational chart widgets in all frames share generic printer settings. Therefore, setting a device name for one chart sets it for all charts.

_repopulate

Builds or rebuilds all or part of an organizational chart from the specified SAS data set or SCL list

Syntax

CALL NOTIFY (orgchart-name, '_repopulate'<, id-type<, data-id<, maplist<, node-id<, append<, collapse>>>>>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-type</td>
<td>N</td>
<td>specifies a value indicating the type of data in data-id:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 a standard SCL list where each item name and value are shown as a node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 an SCL list whose items are valid node variables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 a SAS data set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If id-type is missing or omitted, the method uses the data type specified in the Attributes window.</td>
</tr>
<tr>
<td>data-id</td>
<td>N</td>
<td>specifies the identifier of the SCL list or SAS data set with which to populate the chart. If data-id is an empty list or a 0, all nodes in the tree are deleted. If data-id is missing or omitted, the method uses the value specified in the Attributes window and stored in the DATASET or DATALIST instance variable.</td>
</tr>
<tr>
<td>maplist</td>
<td>N</td>
<td>specifies the identifier of an SCL list that maps node variables to data set variables. If maplist is missing or omitted, the method uses the mapping list defined in the Mapping List window and stored in the MAPLIST instance variable. Maplist is ignored if the chart is populated with an SLIST entry.</td>
</tr>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node to repopulate. By default, the entire tree is cleared and replaced with the new data. If node-id is not 0 and not missing, the new data replaces the existing children of the node. To add a new tree to the existing children of the node, use the append argument. If node-id is omitted or missing, the entire tree is repopulated.</td>
</tr>
<tr>
<td>append</td>
<td>N</td>
<td>adds the new nodes as children of node-id if the value specified is anything other than 0. If append is omitted, missing, or 0, all existing children of node-id are deleted before the new nodes are added. If append is greater than 0, the nodes are inserted as children beginning at the position ABS(append).</td>
</tr>
</tbody>
</table>
### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collapse</td>
<td>N</td>
<td>specifies whether to display or hide the node's children</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 displays all children (the default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 hides all children</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 hides all grandchildren.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignored if node-id is not used.</td>
</tr>
</tbody>
</table>

### Details

The _repopulate method is automatically invoked by the _postinit method if not invoked directly by the SCL program.

_repopulate can rebuild the entire tree, rebuild individual branches, or append items to existing branches. In addition it can replace or override data specifications given in the Attributes and Mapping List windows. Omitted arguments default to the specification given in the Attributes window or to the value of the associated instance variables: DATALIST, DATASET, or MAPLIST.

If data-id is a list, use the DELLIST function to remove it after _repopulate is called. If data-id is a data set identifier, use the CLOSE function to close it.

All arguments (except id-type) require at least a missing value for all preceding arguments. For example, to specify only the maplist argument, provide missing values for the preceding arguments:

```scl
    call notify('org1', '_repopulate_', ., ., maplist);
```

### Examples

**Example 1: Populating a chart from a data set identifier**

This example populates the chart ORG1 from the data set identifier in DSID. Because append is nonzero, all the data read in from DSID are simply added to node NODEID without deleting any of its prior children. This simulates and _addMode method.

**ORG1:**

```scl
    call notify('org1', '_get_selected_', widgetid, nodeid);
    if (nodeid eq 0) then return;
    infol = 0;
    call notify('org1', '_get_current_', nodeid, "TEXT", infol);
    month = getnitemc(info, "TEXT");
    dsid = open("MYLIB." || month || "SALES","I");
    call notify('org1', '_repopulate_', 2, dsid, ., nodeid, 1);
    rc = close(dsid);
    rc = dellist(infol, "Y");
    return;
```

**Example 2: Creating a leaf node**

This example makes a node a leaf node by deleting all of its children:

**ORG2:**

```scl
    call notify ('org2', '_get_selected_',
```
widgetid, nodeid);
if nodeid = 0 then return;
call notify ('org2', '_repopulate_', 1,
0, ., nodeid);
return;

__saveAs

Creates a data set or SLIST entry from the organizational chart data

Syntax

CALL NOTIFY (orgchart-name, '_saveAs'<, savelist>);

Argument Type Description

savelist N specifies the identifier of an SCL list containing one or
more of the items in Table 84.4 on page 1565. A missing
value is interpreted as 0 for numeric items.

Table 84.4 __saveAs List Items

<table>
<thead>
<tr>
<th>List Item</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'DATASET'</td>
<td>C</td>
<td>names the data set to write to.</td>
</tr>
</tbody>
</table>
| 'LEVELS'  | N    | specifies the number of tree levels down from the node
where saving begins to save. Specify 0 to save all levels. |
| 'NODEID'  | N    | specifies the numeric identifier of the node where the save
begins. If omitted, the save begins at the root of the tree. |
| 'SETUP'   | N    | indicates whether to open the Save Options window:
0 do not open
^=0 open (default) |
| 'SLIST'   | C    | specifies the name of the SLIST entry to write to if you
wish to save the tree in a catalog SLIST entry instead of
a data set. Use the SCL function FILLIST to initialize a
list that can populate another organizational chart. |

Details

This method also provides a way to convert data sets with parenting information to
data sets with level numbers. Datasets with level numbers can be read in faster than
data sets with parenting information because no searching is required for each
observation when building the tree.
Returns the numeric identifier of the first or successive node that contains the specified string or numeric value

**Syntax**

CALL NOTIFY (orgchart-name, '_search', node-id, list-of-names, list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>returns the numeric identifier of the node, or 0 if the value is not found. You may pass in the node identifier of a subtree to search when using the /C option in list-of-names.</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the name of the node variable whose value you are searching for. List-of-names can include one or more of these node variables: ID, CVALUE, NODEID, NVALUE, OBS, or TEXT. Although the list can contain more than one node variable name, you can only search on one node variable at a time. You can control the search by including one of these options anywhere in the text string: /S (substring) if the node variable is character, allow the search value in list-id to be a substring of the actual node data. /N (next) start the search at the node where any previous _search stopped. /C (current) search only the current subtree beginning with the node identified by node-id. /R (reverse) search in reverse order of the default.</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the numeric or character data to search for. Each list item contains a data value and an item name that corresponds to a node variable name. The list can contain multiple items, but only the value of the item whose name is specified in list-of-names is searched for.</td>
</tr>
</tbody>
</table>

**Example**

This example searches various nodes. First, find the first node whose character value is 'MYLIB.MYCAT.MYPROG.SCL' and place that node in the viewable area:

```plaintext
mylist = makelist(0);
mylist = insertc(mylist, 'mylib.mycat.myprog.scl', 1, 'cvalue');
call notify('org1', '_search_', nodeid, 'cvalue', mylist);
```
call notify('org1', '_goto_node_', nodeid);

Next, find the first node whose numeric value is 32 and go to that node:

mylist = insertn(mylist,32,1,'nvalue');
call notify('org1', '_search_', node-id, 'nvalue', mylist);

Then, find the first node whose display text contains 'Smith' and use /N to start where the previous search left off:

mylist = insertc(mylist,'Smith',1,'text');
call notify('org1', '_search_', node-id, 'text /s /n', mylist);
rc = dellist(mylist);

_setClass

Creates a new node type by assigning either a predefined or user-defined class to a node type number

Syntax

CALL NOTIFY (orgchart-name, '_setClass', class, nodetype);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>C</td>
<td>specifies the four-level name of a class. This class can be a subclass of one of the three predefined organizational chart classes or a user-defined widget class.</td>
</tr>
<tr>
<td>nodetype</td>
<td>N</td>
<td>specifies the index of the node type. This number must be greater than 3 because the first three node types are predefined, and it must be equal to or less than 1 plus the number of node types. If it is 1 greater than the number of node types, a new node type is created.</td>
</tr>
</tbody>
</table>

Example

Because there are three predefined node types, any new ones that you wish to create must start at 4. After this example is run, all newly created objects will be node type 4 and class SASUSER.OBS.MYCLASS.CLASS, unless overridden:

call notify('org1','_set_class_','sasuser.obs.myclass.class',4);
call notify('org1','_set_option_','default_type',4);
_setColor

Assigns either region color, region border color, foreground color, or line color to one or more node variables.

Syntax

CALL NOTIFY (orgchart-name, '_setColor', color-var, color,<, nodetype,<, node-id>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-var</td>
<td>C</td>
<td>specifies the node variable to be assigned the color:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BACKGROUND_COLOR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BORDER_COLOR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'FOREGROUND_COLOR'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'LINE_COLOR'</td>
</tr>
<tr>
<td>color</td>
<td>C</td>
<td>specifies a color</td>
</tr>
<tr>
<td>nodetype</td>
<td>N</td>
<td>specifies the index of the node type. If node-id is specified, specify a missing value for nodetype. If omitted, missing, or 0, all existing node types use the specified color.</td>
</tr>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of a single node whose color you want to set. Ignored if nodetype is given.</td>
</tr>
</tbody>
</table>

Details

If both nodetype and node-id are omitted or missing, all node types are set.

Example

This example sets the colors of all nodes of type 1:

call notify('org1', '_set_color_',
   'background_color', 'black', 1);
call notify('org1', '_set_color_',
   'foreground_color', 'white', 1);
call notify('org1', '_set_color_',
   'border_color', 'gray', 1);
call notify('org1', '_set_color_',
   'line_color', 'red', 1);

This example sets the background color for the node whose identifier is stored in the variable NODEID. Because this statement assigns a color to a specific node, a missing value is given for nodetype:

call notify('org1', '_set_color_',
   'background_color', 'blue', , nodeid);
**_setCurrent**

Changes the data for the specified node

**Syntax**

CALL NOTIFY (orgchart-name, '_setCurrent', node-id, list-of-names, list-id);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the names of one or more node variables whose values you want to change. List-of-names can include any of the node variables except CHILDREN, LEVEL, NODEID, and WIDGETID</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list containing the numeric or character data to assign to the node variables specified in list-of-names. Each list entry contains a data value and an item name that corresponds to a node variable name.</td>
</tr>
</tbody>
</table>

**Details**

To change, add, or delete the children of a node, use the _repopulate method.

**Example**

This example replaces the name of the node containing the name Rathers with the name entered by the user in a text entry field called NAME:

```plaintext
NAME:

   slist = makelist (0);
   slist = insertc (slist, 'Rathers', 1, 'text');
   call notify ('org1', '_search_', nodeid, 'text /s', slist);
   if (nodeid ne 0) then do;
      rc = setnitemc(slist, name, 'text');
      call notify ('org1', '_set_current_', nodeid, 'text', slist);
   end;
   rc = dellist (slist, 'y');
return;
```

---

**_setImage**

Assigns an image to a node type
**Syntax**

CALL NOTIFY (orgchart-name, '_setImage', nodetype, image);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodetype</td>
<td>N</td>
<td>specifies the index of the node type that uses the image</td>
</tr>
<tr>
<td>image</td>
<td>C</td>
<td>specifies a text string identifying the location of the default image.</td>
</tr>
</tbody>
</table>

**Example**

This statement sets all nodes of type 2 to use the FILENAME image stored in the default image catalog:

```bash
call notify('org1', '_set_image_', 3, 'filename');
```

This statement sets all nodes of type 3 to use the left and right arrow image from the SASHELP.FSP catalog:

```bash
call notify('org1', '_set_image_', 3, 'sashelp.fsp.lrshift.image');
```

**_setOption**

Sets an option for a particular node class, or all node classes, depending on the option and the argument.

**Syntax**

CALL NOTIFY (orgchart-name, '_setOption', option<, val1<, val2>>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>C</td>
<td>specifies the option to set. Option names and values are described in the following table.</td>
</tr>
<tr>
<td>val1</td>
<td>N</td>
<td>specifies a value for option</td>
</tr>
<tr>
<td>val2</td>
<td>N</td>
<td>specifies a value for option</td>
</tr>
<tr>
<td>Values for option</td>
<td>Description</td>
<td>Values for val1</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>'ALL_LEVELS'</td>
<td>specifies levels of the tree to display</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>'BORDER_TYPE'</td>
<td>sets the border style of one or all node types</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>'BORDER_WIDTH'</td>
<td>sets the border width of one or all node types</td>
<td>0...64</td>
</tr>
<tr>
<td>'DEFAULT_TYPE'</td>
<td>sets the default node type for any new nodes</td>
<td>1...n</td>
</tr>
<tr>
<td>'IV_CHANGED'</td>
<td>reprocess the values of all the instance variables</td>
<td>unused</td>
</tr>
<tr>
<td>'LINE_WIDTH'</td>
<td>sets width of the line that connects the nodes for one or all node types</td>
<td>0...64</td>
</tr>
<tr>
<td>'SMOOTH_SCROLLING'</td>
<td>specifies whether to refresh chart during scrolling or after the mouse button is released</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>'VERTTEXT'</td>
<td>defines how text in nodes is displayed</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

This statement causes the chart to show only the top two layers of the tree (before _repopulate):

```plaintext
call notify('org1', '_set_option_', '
    '_all_levels_', 0);
```

This statement sets the default node type to text and image (type 3):

```plaintext
call notify('org1', '_set_option_',
    '_default_type_', 3);
```

This statement changes all type 1 nodes to a button border:

```plaintext
call notify('org1', '_set_option_',
    'border_type', 4, 1);
```

_setSize

Sets the size at which nodes of a particular type (or all nodes) are created

**Syntax**

```plaintext
CALL NOTIFY (orgchart-name, '_setSize', width, height<, nodetype>);
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>N</td>
<td>specifies the default width</td>
</tr>
<tr>
<td>height</td>
<td>N</td>
<td>specifies the default height</td>
</tr>
<tr>
<td>nodetype</td>
<td>N</td>
<td>specifies the index of the node type. If 0 or omitted, applies to all node types.</td>
</tr>
</tbody>
</table>

**Details**

If Do not resize in the Node Appearance window is off, the _setSize method has no effect, and the nodes are created at the size specified in the window. In this case, the nodes may resize when the INIT section runs.

_setSize has no effect on already-created objects. It must be called in the INIT section of the SCL program or before a _repopulate method is run so the default node size is set before the nodes are created. User-defined classes may ignore these settings.
_setStyle

Controls the appearance of the tree, including display style, orientation, and line style

Syntax
CALL NOTIFY (orgchart-name, '_setStyle', style, orient, lines);

Argument Type Description
style C specifies the type of tree:
   'HIERARCH' hierarchical tree
   'DIR' directory tree
orient C specifies the orientation:
   'HORZ' horizontal
   'VERT' vertical
lines C specifies the style of the lines connecting the nodes:
   'ORTHO' orthogonal lines (lines in 90 degree angles to the nodes)
   'DIRECT' direct lines (lines running straight from parent to child)

Details
To leave a setting unchanged, omit the argument and supply a null value (' ').

Example
This statement creates a horizontal directory style tree with straight lines:
call notify('org1','_set_style_','dir','horz','direct');

This statement creates vertical hierarchical style tree with angled lines:
call notify('org1','_set_style_','hierarch','vert','ortho');

_setTitle

Assigns the text of the title node and, optionally, makes the title node the root of the tree

Syntax
CALL NOTIFY (orgchart-name, '_setTitle', title, is-root, connect, hide-it);
### Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>C</td>
<td>specifies the text of the title</td>
</tr>
<tr>
<td>is-root</td>
<td>C</td>
<td>specifies whether the title node is the root of the tree:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' is the tree root</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' is not the tree root</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If not given, retains the setting specified in the Attributes window.</td>
</tr>
<tr>
<td>connect</td>
<td>C</td>
<td>if the value of is-root is 'Y', specifies whether to draw lines from the title node to each child:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' draws lines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' does not draw lines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If omitted, connect retains its current setting.</td>
</tr>
<tr>
<td>hide-it</td>
<td>C</td>
<td>specifies whether to hide or show the title:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'Y' shows title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'N' hides title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If omitted, retains its current setting.</td>
</tr>
</tbody>
</table>

### Details

To remove the existing title, specify a missing value for title and omit is-root and connect.

### Example

This example creates a title that is the root node of the tree:

```plaintext
call notify('org1','_set_title_','SAS Institute','y');
call notify('org1','_repopulate_');
```

This statement supplies a null string for title and removes an existing title node:

```plaintext
call notify ('org1', '_set_title_', '');
```

### _traverse

Calls the specified SCL label for every node in the tree or specified subtree.

### Syntax

CALL NOTIFY (orgchart-name, '_traverse', node-id, list-of-names, list-id, label);
## Argument Type Description

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node-id</td>
<td>N</td>
<td>specifies the numeric identifier of the node for which the labeled section of SCL code is being run</td>
</tr>
<tr>
<td>list-of-names</td>
<td>C</td>
<td>specifies a single, quoted text string containing the names of the node variables whose values are to be returned in list-id. Separate multiple node variable names with a space. Specify 'ALL' to return the value of all variables. You can control the processing by including one of these options as the last item in the text string:</td>
</tr>
<tr>
<td>list-id</td>
<td>N</td>
<td>specifies the identifier of an SCL list in which the _traverse method stores the variable values for the current node being traversed</td>
</tr>
<tr>
<td>label</td>
<td>C</td>
<td>specifies the label of the SCL program section to run for each node. _traverse calls _getCurrent before running label.</td>
</tr>
</tbody>
</table>

## Details

Each time label runs, the organizational chart object sets node-id to the next node in the sequence. To traverse only a subtree instead of the entire tree, include the /C option in list-of-names and assign node-id on the initial call. The value automatically changes on the first call to label.

_traverse repeats this process for each node:

1. Calls _getCurrent and optionally creates and always fills list-id for the current node.
2. Sets node-id.
3. Calls label.

When done, if _traverse created the list, it deletes it.

## Example

This example uses the _traverse method to change the colors of the nodes. After _repopulate executes, all the nodes with a numeric value less than 0 are set to a red background. Otherwise, the background color is green. The organizational chart object calls the SETCOLOR section for each node in the tree after setting N to the node identifier and filling MYLIST with each node’s data.

```plaintext
REPOP:
call send(orgid,'_repopulate_');
mylist = 0;
call send(orgid,'_traverse_',n,'nvalue',
       mylist,"setcolor");
return;

/* the labeled section called by the _traverse method */

SETCOLOR:
```
value = getnitemn(mylist, 'nvalue');
if (value lt 0) then color = 'red';
else color = 'green';
call send(orgid, '_set_color_',
    'background_color', color, .., n);
return;

_vscroll

Scrolls the organizational chart vertically

Syntax

CALL NOTIFY (orgchart-name, '_vscroll'<=, distance>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distance</td>
<td>N</td>
<td>specifies the number of pixels to scroll vertically. If omitted, the value set by the VSCROLL command is used. If VSCROLL has not been issued, 1/4 of the region height is assumed. If the value is negative, the chart is scrolled up.</td>
</tr>
</tbody>
</table>