ProvideX

VERSION 6

NOMADS Enhancements

best software

Publication Release: V6

NOMADS Enhancements

NOMADS, the Non-procedural Object Module Application Development System, simplifies the development of complex GUI-based ProvideX applications. This toolset is bundled with the Base System and is fully integrated with the Graphical Program Editor.

Feature Changes and Additions

Several new features, additions, and changes to the NOMADS toolkit for Version 6 are introduced in this supplement in conjunction with the NOMADS Enhancements presentation. The topics covered below will be incorporated into the new ProvideX NOMADS Reference scheduled for release after Version 6:

User-Defined Colours

NOMADS can now access a palette of up to 255 colours.

Gradient Fill

Folder tabs and shapes (arc, circle, pie, polygon, rectangle) can now be defined using colour gradient fill.

Object Persistence

This feature allows applications to optionally save control coordinates (column, line, width and height) as well as format information for list boxes and grids on the termination of a panel, and restore them the next time it is created.

Note: Included in this supplement are sections on **Panel Persistence** and **Object Resizing**. Although they do not describe new features, these sections are required to complete the description of Object Persistence.

Smart Display Fields

Display-only multi-line controls that are an extension to the Smart List package used to provide an automatic lookup and display of data on the panel.

Load-on-Demand Lists

On-demand loading has been added to standard, formatted, and listview list boxes.

Background List Box/Grid Loading

This allows you to define a program/entry point to be performed for loading the contents of a control – this logic executes when there is no user input.

Tree View Enhancements

Images can now be set up to appear in front of a list box entry to indicate whether an item has been selected or not.

COM Controls

COM (Common Object Module) controls have been added to NOMADS - this control type replaces exisiting VBX controls and supports COM event handling.

User-Defined Colours

NOMADS can now access a palette of 255 colours. By default, the system pre-loads the first 16 colours (0-15) for use by the system. The user-defined colours start at Colour # 16.

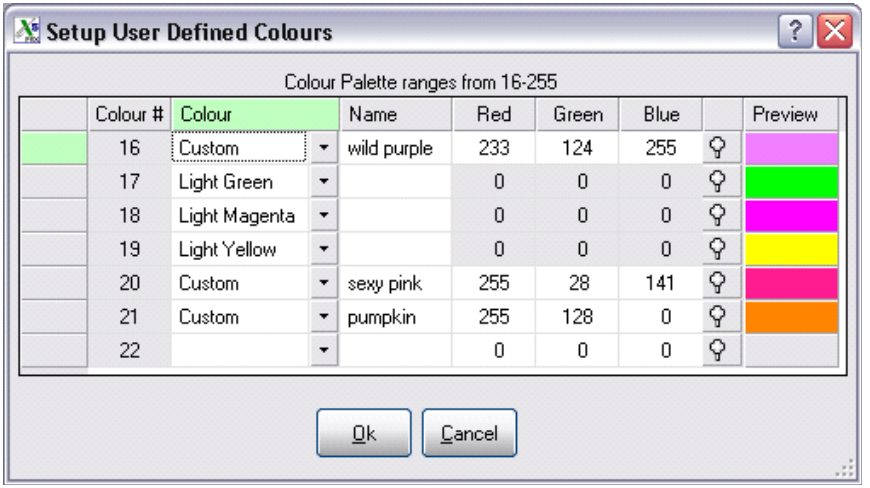
A new NOMADS utility allows the developer to set up colour schemes defined in terms of RGB values. Once these colours are loaded into the palette, they can then be assigned to existing controls, panel defaults and library defaults.

This feature allows the application to be more flexible because any change to an existing user-defined colour is passed on to all controls/panels using that colour number.

During the assignment, the user can select an existing colour from the new colour file `providex.clr` or use an expression or variable to represent the colour. Expressions and variables will be evaluated at runtime.

Defining User-Defined Colours

The following utility has been added to the NOMADS main menu. This new utility can be accessed by selecting the Options > User Defined Colours items.



The colours are stored in the `providex.clr` file. If the file does not exist, the system will inquire whether you would like to create one. The file `providex.clr` contains the colour # index 16-255 (pre-loaded from 16), the colour selected (i.e., *Custom* or one of the existing 16 colours such as *Light Green*), name of the colour (optional), and the RGB values that make up the colour.

If you select *Custom* from the Colour drop box, then you must enter RGB values to define the colour combination, or press the colour *query* button beside the Preview column to browse the windows colour palette. The Preview column shows the selected colour.

Assigning User-Defined Colours

User-defined colours can be assigned to panel defaults, library defaults and all controls.

In the Font/Clr panel of a control definition, select User Defined from either the Foreground or Background drop box. The colour can be a fixed value or an expression. If the value is fixed, enter the colour number or `Colour+nnn` (`Color+nnn`) or press the *lookup* button to retrieve a selection from the colour file. The following is an example expression/variable:

```
Foreground_clr$="22"
Foreground_clr$="Colour22"
Foreground_clr$="Color22"
```

This variable would contain `Colour+nnn` or `Color+nnn` (where *nnn* is the colour index). This variable should be loaded after the control is drawn.

Loading the Colour Palette

NOMADS reads the `providex.clr` file and loads all of the colours into the palette using the '**OPTION**' mnemonic; e.g.,

PRINT 'OPTION'("Colour*nnn***","RGB:***rrr ggg bbb***")**

Where:

nnn	colour index
rrr	number between 0-255
ggg	number between 0-255
bbb	number between 0-255

Gradient Fill

Folder tabs and shapes (arc, circle, pie, polygon, rectangle) can now be defined with gradient fill.

The gradient filling requires two colours: a starting colour and an ending colour. It also requires a direction: top to bottom, left to right, top-left to bottom-right, or top-right to bottom-left. In order to specify the two colours, the **'FILL'** mnemonic now supports an additional colour option representing the ending colour to be used in the two-colour gradient.

How it Works

NOMADS uses the **'FILL'** mnemonic when drawing folder tabs and shapes.

The direction of the gradient fill is derived from the fill pattern using the existing fill patterns: **2** (top to bottom), **3** (left to right), **5** (top-left to bottom-right), and **6** (bottom-left to top-right). Also supported are non-gradient two-colour fill patterns: **4** and **7** (2nd colour is background - 1st colour for lines).

The fill patterns are based on the first colour being blue and the second colour being yellow:

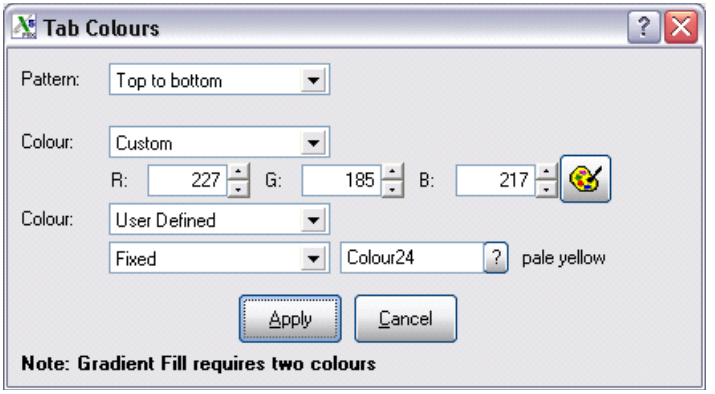
The fill pattern is generally envisaged to be used with the **'RECTANGLE'** mnemonic, however it will be supported by all "fillable" graphic-drawing entities.

Defining Folder Tabs with Gradient Fill

Applying gradient fill to tabs:

1. In the Display panel of the folder definition, select an existing tab from the list box. This will cause the Tab Colours panel to display automatically.
2. Select the pattern for the tab. Select any pattern other than *Solid Fill.*
3. Two colours are required for gradient filling. The first colour is the starting colour, the second colour is the ending colour.
4. The colour can be one of the existing 16 colours (0-15) such as *Light Green*, a custom colour (using RGB values that make up the colour combination), or a user-defined colour (using an existing colour in the `providex.clr` file or an expression/variable holding the colour number). See **User-Defined Colours** for more information on colours and the `providex.clr` file.

Sample Panel:



Defining Shapes using Gradient Fill

The assignment is the same as *folder tabs*. Select a pattern and two fill colours.

Note: Gradient Fill is only applicable to arcs, circles, pies, polygons and rectangles.

Object Persistence

The object persistence option in NOMADS allows the user to optionally save control coordinates (column, line, width and height) and format information for list boxes and grids on the termination of a panel, and restore those controls to the same size and location along with the new format definition, if applicable, the next time it is created.

Object persistence is only supported on input objects (multi-lines, list boxes, drop boxes and grids).

Note: Column names must be assigned to a grid for this feature to work properly when the **'SwapEnabled** property is set.

The same generic sample program (used by **Panel Persistence**) called `*winpnl` is supplied to create, read, and write the object information data file. `*winpnl` creates a file called `panel.inf` in the same directory as the ProvideX executable; e.g., `/pvx/panel.inf`. This is the same file used to store the panel coordinates (location and size). The records in the file are keyed on the first forty characters of the *user ID* + *panel name* (uppercase, padded to 12 with spaces) + *simple panel library path*.

Object Coordinates

The object coordinates are stored in the file as follows:

SIZ:[panel_coordinates***;***panel_name***01***control_name***=c,l,w,h;...]**

This includes the *panel_name* (padded to 12 characters) and *control_name* separated by a hex 01 and followed by the coordinates (column, line, width and height). A semi-colon will follow each object's coordinates.

Refer to **Panel Persistence** for a breakdown on the panel coordinates.

Format Information

Format information for list boxes and grids are also stored in the file. The format information follows the ***SIZ** definition.

FMT:[panel_name***01***control_name***=***format definition***;...]**

This includes the *panel_name* (padded to 12 characters) and *control_name* separated by a Hex 01 and followed by the format definition. A semi-colon will follow each object's format definition.

For grids, the format definition will contain:

[*column title***](***cell type***:***column name***)***alignment character width***

Where:

column title	title that will appear in the top row of the grid.
cell type:column name	default cell type for the column followed by the column name. The default cell type is Normal. Refer to the *Language Reference Manual* for a list of available cell types. The column name can be a string or numeric variable.
alignment character	alignment of cell contents. There are 3 possible alignment characters. **L** (left), **C** (centered), **R** (right). The default alignment character is **L.**
width	default width of the column

Example:

```
"[Client ID](Multi_line:CST_ID$)L10 [Name](Normal:CST_NAME$)L10"
```

For list boxes, the format definition will be different depending on the type of list box. Refer to the *Language Reference Manual* for list box format definitions.

Activating Object Persistence

Once *Panel Persistence* and *Object Resizing* has been turned on (see **Panel Persistence** and **Object Resizing** sections for details) you need to decide if you want Object Persistence set globally or on individual controls.

Global Activation

To activate at the application level, set the global variable `%NOMAD_OBJECT_PERSISTENCE=1.`

Activating Individual Controls

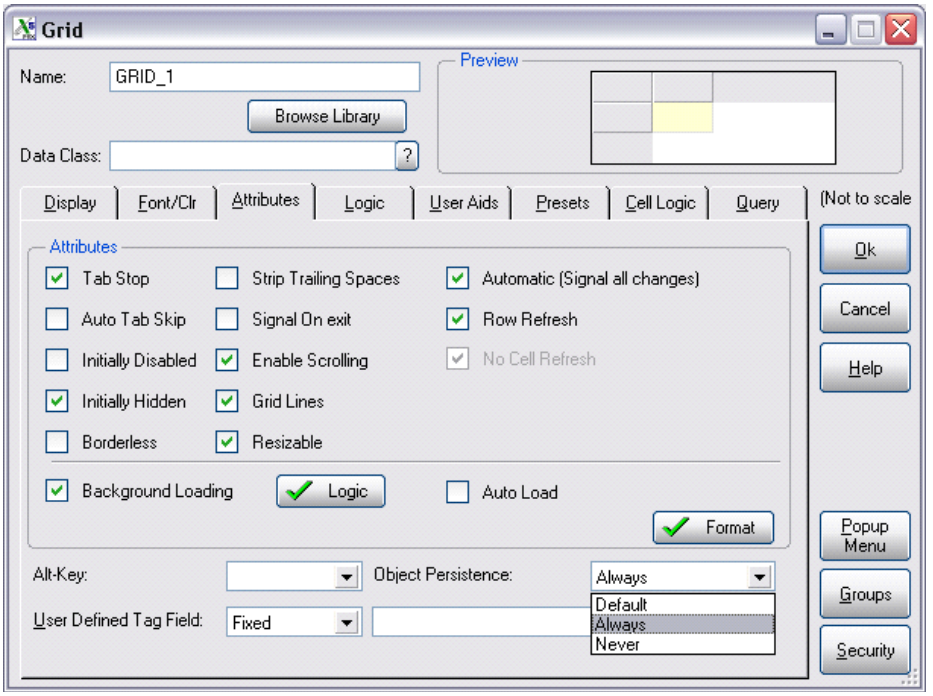
Make a selection from the Object Persistence drop box in the Attributes panel of the Grid, List Box, Drop Box or Multi-line definition. There are 3 possible selections:

Default - determined by the global activation setting

Always - Object Persistence will always be on for the selected control.

Never - Object Persistence will not be on for the selected control.

Note: Object Persistence on individual controls will override global activation.



Folders and Object Persistence

Folder controls will also support Object Persistence. Location and format information for controls on sub-panels will be updated when you move between tabs.

Panel Persistence

The NOMADS ***winproc** engine optionally saves panel size and location information on termination of a panel, and restores that panel to the same size and location the next time it is created. This affects panels designated as 'dialogue', as well as query objects.

A generic sample program called ***winpnl** is supplied to create, read, and write the panel information data file. ***winpnl** creates a file called `panel.inf` in the same directory as the ProvideX executable; e.g., `/pvx/panel.inf`. The records in the file are keyed on the first forty characters of the user ID + panel name (in uppercase, padded to 12 with spaces) + simple panel library path.

The panel information is stored in the file as ***SIZ:[*=***hhh*,*www*;**]**

Where *hhh* refers to the panel height and *www* refers to the panel width. All the object coordinates will follow the panel coordinates. Refer to **Object Persistence** for more information.

You may use this program, or write your own to store and retrieve the panel information.

Activating Panel Persistence

To enable the persistence option using *winpnl, simply **CALL** or **PERFORM "*winpnl"**. If you wish to write your own program, load the **%NOMAD_PANEL_INFO_PROG$** variable with the name of the program that will read and write the panel information to a file. The program will be called and must have two entry points: **GET_PANEL_INFO**, which retrieves the stored location and size of the specified panel; and **SAVE_PANEL_INFO**, which stores the location and size of the specified panel to a disk file. Details of the **CALL**s and the required arguments can be found by listing *winpnl.

Note: This feature is overridden when testing panels and queries from within the NOMADS Designer.

Object Resizing

The object resizing feature in NOMADS allows users to resize various input objects at runtime by simply dragging their edges. List boxes, drop boxes, multi-lines, and grids are all resizable. These controls can be expanded vertically and horizontally; however, multi-lines that are one-line high, and drop boxes, can be expanded horizontally only.

Note: Standard and Formatted list boxes should have their No Height Adjustment attribute turned on to prevent the bottom of the list box from 'fluttering' when the size is adjusted and to give a true indication of the bottom edge.

Activating Object Resizing

To activate object resizing at the panel level, check the Size Adjustment attribute in the Panel Header. To activate at the application level, set the global variable `%NOMAD_OBJECT_RESIZE=1.`

(***Note:*** Setting `%NOMAD_OBJECT_RESIZE=1` takes precedence over the Size Adjustment setting in the Panel Header.)

Note: If a panel is resized, then any object whose size has been adjusted is resized based its original dimensions and not on its adjusted size.

Smart Display Fields

This feature is an extension to the Smart List package. Smart Display Fields are *display-only* multi-line controls used for the automatic lookup and display of data on the panel. Typically, this type of field might be used to lookup and display a salesman name beside the input field where a salesman code was entered.

Each Smart Display Field would be linked to an existing query definition. The query definition consists of a "Smart Key" that contains the control name(s) on the panel and/or literals that will be passed into the query. A plus sign is used to concatenate multiple segments in building the key; e.g.,

```
%COMPANY_CO$+CST_SMN$
```

Smart Display Fields are triggered to load data based on specified trigger variables. When the value in a trigger variable changes, the field is loaded. If the query fails to retrieve data, a predefined text is loaded into the Smart Display Field. If there is no text defined, then spaces are loaded.

Defining Smart Display Fields

The following steps outline how to set up a Smart Display Field:

Step 1. Set Up Attributes

The first step is to turn on the Auto Load option in the Attributes panel of the multi-line definition. Because these are "display only" fields you should also turn off the Tab Stop and turn on the Locked and Borderless attributes.

Step 2. Define the Load-Triggering Criteria

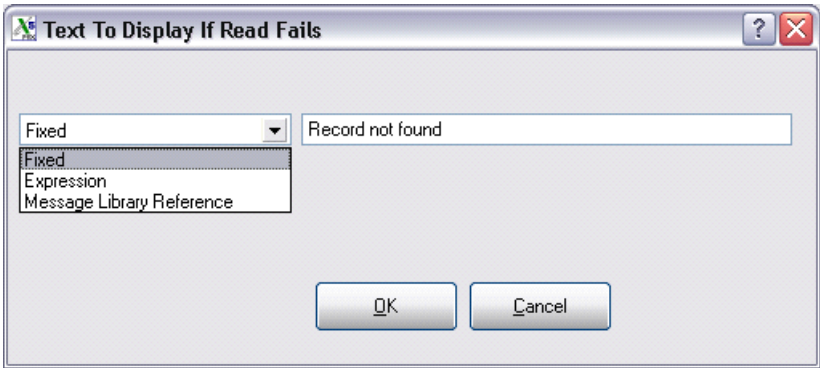
The second step is to define the trigger variables/controls.

Press the Triggers button on the Attribute panel to define the trigger variables/controls. (This button becomes visible when the Auto Load attribute is selected.)

Trigger variables serve two purposes: they initiate the load logic, and their values can be used to define the smart key (*see Step 5*). If you define trigger variables, any change in the value of a trigger variable causes the Smart Display Field to be reloaded after the trigger control's On Select Logic has been executed.

Step 3. Define Text to Display if No Record is Found

Press the No Data Msg button on the Attributes panel to define the text that will display if the query failed to retrieve a record. (This button becomes visible when the Auto Load attribute is selected.)



The text can be a fixed value, an expression or an entry in a message library file. A Browse button will appear if the drop box selection is a Message Library Reference. Press the Browse button to select an entry from a previously defined message file (To setup message libraries, select Options>Message Manager from the NOMADS menu bar.)

Step 4. Associate a Query Object

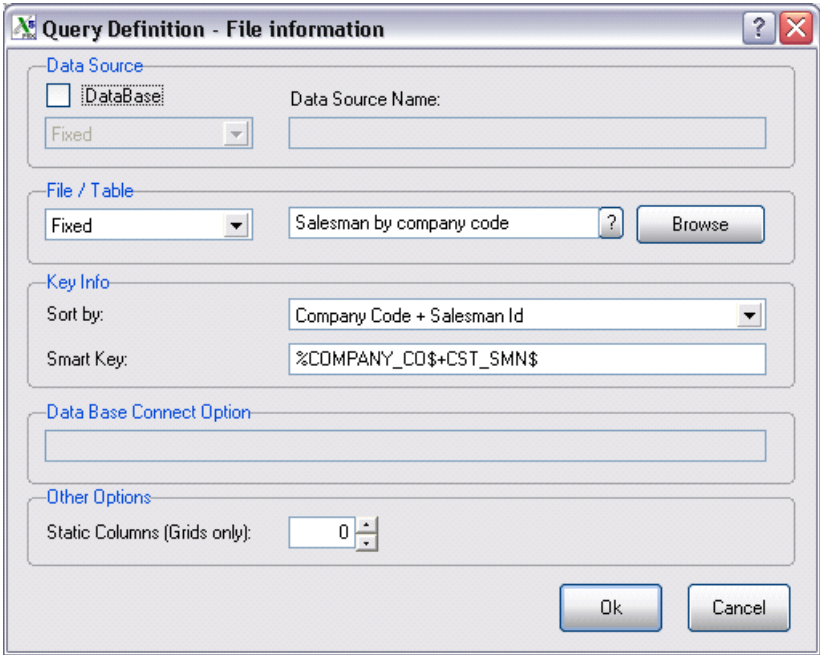
The fourth step is to assign a query object to define what fields will make up the Smart Key and determine what return value will be sent back to the Smart Display Field. This is specified on the Query panel of the multi-line definition. The associated query definition must be a special query definition called a *query list* definition. An existing definition may be selected or a new name entered. The query definition logic can then be accessed by pressing the Define button. If it is a new definition, it will default to a *query list* type of definition.

You may also create or update a query definition by entering the name of the query object in the Objects New>Maint input box of the Library Objects Selection panel, and then pressing the Query Object button.

The *query list* type of definition is a subset of the *standard query* type. Smart Display Fields only work with *query list* definitions. (See references to *Query List Object* in the *NOMADS Reference Manual* for more detailed information.)

Step 5. Defining the Smart Key

The final step is to build the *Smart Key*. The definition consists of a combination of trigger variables from the panel, global variables, and literal values that are passed into the key. If the key consists of multiple components, use a plus sign between each component.



Load-on-Demand Lists

One of the problems facing developers is that of loading large lists into controls. The time and overhead it takes to load the list (especially when using WindX or JavX) can be lengthy. If the number of elements in the list is known, the new on-demand loading feature can be used to reduce overhead.

On-demand loading has been added to standard, formatted and list view list boxes. This allows the developer to pre-declare the number of items that the list box is to have (by setting the **'ItemCount** property) and then, as the list box requires the items, NOMADS generates a CTL/event that triggers the load-on-demand logic.

The application obtains the index number of the items needed for the list box by reading the new **'ItemNeededFrom** and **'ItemNeededTo** properties. It then loads the list box with the contents of the specified items by setting **'Item** and **'ItemText$**. The application will only read and display those items that the user actually scrolls into view. If there are no elements required, **'ItemNeededFrom** and **'ItemNeededTo** will be zero.

NOMADS will reserve a CTL value between 23001-23999 for load-on-demand events.

Defining On-Demand Loading

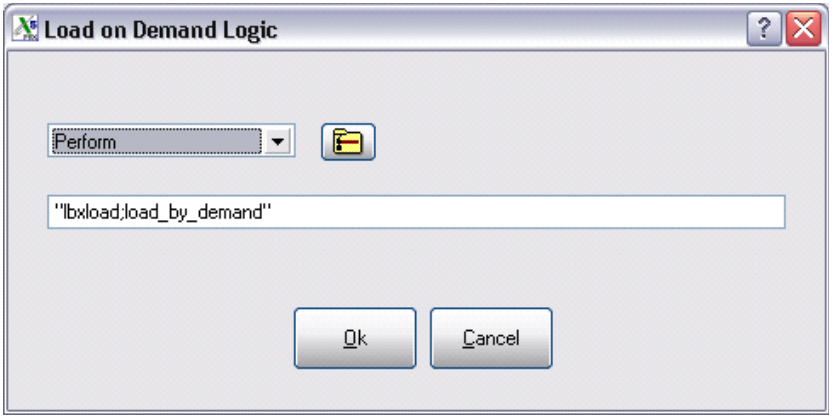
The following steps outline how to set up load-on-demand lists:

Step 1. Set the Load-on-Demand Attribute

The first step is to turn on the Load-on-Demand checkbox in the Attributes panel of the listbox definition. A Logic button will become visible when this checkbox is selected.

Note: The Load-on-Demand checkbox will not appear if the list box type in the Display panel is a *Tree View*.

Step 2. Assign Load-on-Demand logic



Click on the Logic button and enter the program/entry point that will be performed, called or executed when a load-on-demand signal occurs.

Sample program:

```
00010 SET_LISTBOX_COUNT:            ! Initialization logic
00020  LET X=HFN;
       OPEN INPUT (X,ISZ=512,ERR=*NEXT)"CSTFILE";
       READ RECORD (X,ERR=*RETURN)R$;
       CLOSE (X)
00030  LET LB1.CTL'ITEMCOUNT=DEC(R$(15,4)),TOTAL_ITEMS=LB1.CTL'ITEMCOUNT
00040  LET CST_FN=HFN;
       OPEN (CST_FN,IOL=*)"cstfile"
00050  LET I=1
00060  RETURN
00070 LOAD_BY_DEMAND:               ! Load-on-demand logic
00080  LET
      NEEDED_TO=LB1.CTL'ITEMNEEDEDTO,NEEDED_FROM=LB1.CTL'ITEMNEEDEDFROM
00090  IF NEEDED_FROM=0 OR NEEDED_TO=0 \
        THEN RETURN
00100  FOR I=NEEDED_FROM TO NEEDED_TO
00110  READ (CST_FN,RNO=I,DOM=*NEXT)
00120  IF CST_AMT<0 \
        THEN LET CLR$=EVS("'red'") \
        ELSE LET CLR$=""
00130  LET LB1.CTL'ITEM=I;
       LET
      LB1.CTL'ITEMTEXT$=CST_ID$+"/"+CST_NAME$+"/"+CLR$+STR(CST_AMT:"-$###
      ,##0.00")+"/"
00140  NEXT
00150  RETURN
```

Note: **RNO=** usage causes performance to degrade with large files.

Background List Box/Grid Loading

The Background Loading option in NOMADS allows you to define a program/entry point for loading the contents of a control. Multiple controls on the panel can be set up to load in the background. At runtime, NOMADS checks for the presence of user input and (if none is received) performs the load logic.

This logic defines the loading of one or more lines into a list box or grid and should return reasonably quickly so that NOMADS can check for user input in a timely manner. While the loading is executing, the user is still able to interact with the other controls on the panel. NOMADS will continue to perform the load logic until a *load completion* flag is set by the application.

The list box or grid can also be forced into a re-load by simply setting a *reload* flag. This flag may be set by the application when various fields change (for example if the list box or grid contained an invoice number, the flag could be set to reload whenever the customer number changed.) *Reload* is executed automatically when moving between tabs.

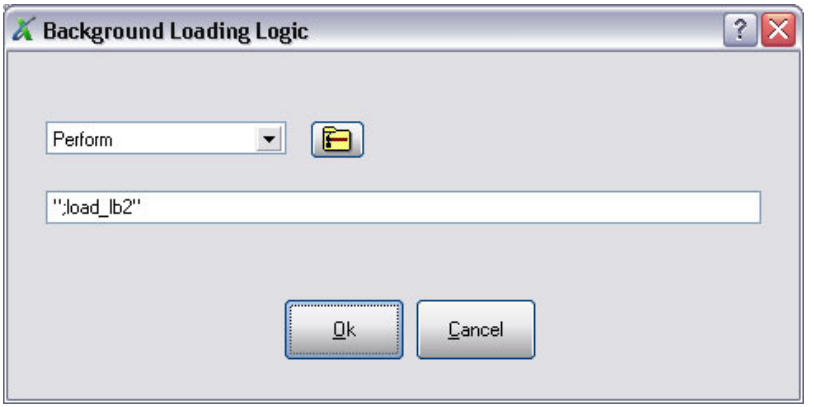
Defining Background Loading

The following steps outline how to set up background loading:

Step 1. Set the Background Loading Attribute

Set the Background Loading attribute. The first step is to turn on the Background Loading checkbox in the Attributes panel of the list box or grid definition. A Logic button will become visible when this checkbox is turned on.

Step 2. Assign Load Logic



Click on the Logic button and enter the program/entry point that will be performed, called or executed to load the control.

Terminating a Load / Forcing a Reload

The status of a background load is stored in a new variable… **control_name.load.** There are 4 possible values:

0	*load not yet completed*	Value is assigned after panel is drawn.
1	*load completed*	*Set by the application.*
-1	*force reload*	
-2	*NOMADS assigns this value when moving between tabs.*	

To end a load, simply set **control_name.load** equal to 1.

Important: NOMADS will continue executing the load logic until this flag is set.

To force a reload of a list box or grid set **control_name.load** equal to -1. When the loading is complete, be sure to set **control_name.load** equal to 1.

The following sample program loads two list boxes using the background load option:

```
0020 predisplay_logic:
0030 LET x=0,z=0,c=0,y=0
0040 EXIT
0050 LOAD_LB2:
0060 LET z=z+100
0070 IF z>10000 THEN LET lb2.load=1; EXIT
0080 SETTRACE PRINT "loading another 100 items starting at
     "+STR(c)+" for lb2"
0090 FOR I=c TO z
0100 LIST_BOX LOAD LB2.CTL,0,STR(I)
0110 NEXT
0120 LET c=z+1
0130 EXIT
0140 LOAD_LB3:
0150 LET y=y+100
0160 IF y>80000 THEN LET lb3.load=1; EXIT
0170 SETTRACE PRINT "loading another 100 items starting at
     "+STR(x)+" for lb3"
0180 FOR I=x TO y
0190 LIST_BOX LOAD LB3.CTL,0,STR(I)
0200 NEXT
0210 LET x=y+1
0220 EXIT
```

Tree View Enhancements

State indicators are basically images that will appear in front of a list box entry that can be used to indicate whether the item has been selected or not. State indicators are currently supported for tree-view list boxes. These images can be set up during the definition of the list box control. (See **Defining State Indicators**, *p.20*.)

A maximum of 15 images can be assigned to the list box; NOMADS stores these images in the **'StateBitmaps** property. All images must be of the same size/format and may specify transparency options. The image can be external or internal. Internal bitmaps contain an exclamation point **!** preceeding the bitmap name.

V6 enhancements also include automatic editing of Tree View items (**'Edit** property), and the detection of expand/collapse requests (**'NotifyExpand** property).

Toggle Between States - 'ItemState

To toggle between different images or states, you need to use the **'ItemState** property in your application. The numeric value in **'ItemState** determines what image will appear next to the row text. This property is 1-based, a value of 0 = No state indicator.

For example, assuming a list box is defined with 3 images. The first image will appear if the item state is one, the second image will appear if the item state is two and the third image will appear if the item state is three.

Auto Toggling Of States - 'AutoState

'AutoState is a numeric property to control auto toggling of states. This property can be setup in the list box definition. (See **Defining State Indicators**, *p.20*.) If this property is set to a non-zero value, state indicators will automatically be toggled without generating a CTL event with `EOM = "S"`. This code is returned in the NOMADS variable **_EOM$** in the When Entry Is Selected From List Box logic.

The number of states that the system will toggle through will be determined by the value set in this property or, if the property is set to one (1), the number of bitmaps assigned to the Tree View (see *Defining State Indicators*). In addition, when the user toggles a state indicator while holding the **SHIFT** key down, all entries between the current entry and the last entry will be toggled to the new state of the current entry -- in effect allowing for group select/deselect.

Cascading States - 'CascadeState

'CascadeState is a numeric property to control cascading of states. This property allows for states to be cascaded from parent to children and vice-versa. This property can be set up in the list box definition (see **Defining State Indicators**, *p.20*)

If the **'CascadeState** property is set to non-zero, the system will automatically cascade parent states to their children and correspondingly make parents states representative of all of their children. Setting a parent's state, either under program control or using the **'AutoState** property in the Tree View definition, will result in all sub-ordinate children to be set to the same state. When a child's state is set, its parent state will be set according to the state of all of the child's siblings - that is if all children are in a consistent state, the parent will be set to the same state. If a parent has children of various states (some on, some off), the parents state will be set to the value set in the **'CascadeState** property.

For example, you could have three state indicators - *Off* (state 1), *On* (state 2), and *Partial* (state 3). You would set **'AutoState** to 2 and **'CascadeState** to 3 to have children that automatically toggle off/on and parents that will be *On* if all children are on, *Off* if all children are off, and *Partial* (state 3) if the children are not in a consistent state.

When cascading, only items with states will be affected. In addition, items without states will not affect their parents states, nor will changing the parent of an item without a state affect the children of that item.

Defining State Indicators

The following steps outline how to set up state indicators for a tree view.

Step 1. Open the State Images Grid

The first step is to click on the States button in the Attributes panel of the list box definition - the States button will only appear if the list box type in the Display panel is a tree view.

Step 2. Assigning State Images

In the State Images grid, enter the name of the bitmap or press the lookup button to the right of the cell to browse for existing internal and external bitmaps. A sample screen shot is provided on the following page. For internal bitmaps, an exclamation point must prefix the bitmap name; for example, **!Stop**. A maximum of 15 images can be assigned to the list box.

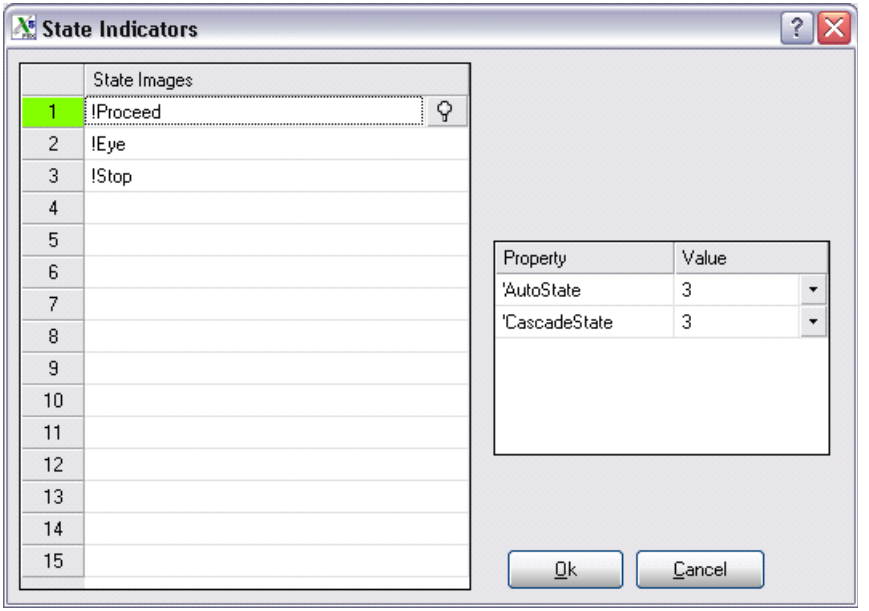
Reshuffling the Images. To change the order of the images, drag a numbered cell in the leftmost column to its new destination.

Step 3. Set 'AutoState *and* 'CascadeState

Set the values for the **'AutoState** and **'CascadeState** properties

Note: These values cannot be greater than the assigned images. For example, if 3 images are assigned, then the **'AutoState** values can be 0,1,2 or 3.

The State Indicators grid appears as follows:



Tree-View Expand Signals - 'NotifyExpand

The **'NotifyExpand** property can be used to detect Tree View expand/collapse requests.

If this property is set to non-zero, a standard CTL event with a EOM code of "+" (expand) or "-" (collapse) will be generated whenever a tree view node is expanded or collapsed. This code is returned in the NOMADS variable **_EOM$** in the When Entry Is Selected From List Box logic. The application can also read the **'Expanded** property to determine what the new state of the node is. If this property is set to zero (default setting), no notification of the tree view expanding or collapsing is sent to the application in keeping with the current implementation. **'Expanded** values include: **1**= expand level; **0**= collapse level; **+** = expand level and all subordinates; **-2**= collapse level and all subordinates.

Notifications are only sent when the expand/collapse is done by the user, not by the application accessing the **'Expanded** property.

Direct Editing - 'Edit

'Edit is a numeric property to allow automatic editing of Tree View items. This property can contain 3 possible values:

0 Item cannot be edited (default).
1 Item can be edited.
-1 Force item into edit mode (must be set in application).

Toggle the Direct Edit check box in the Attributes panel of the Tree View definition to turn direct editing on or off (0/1).

If the Direct Edit check box is turned on, then a double click on an item will put the item into edit mode. When the value changes, a CTL event will be generated with an EOM code of "C" indicating the item has changed. (No event occurs should the user cancel out of the edit mode). This code is returned in the NOMADS variable **_EOM$** in the When Entry Is Selected From List Box logic.

Please note that the editing of items does not support any of the following normal input options:

- format mask processing
- justification (right/center)
- password masking
- input start at end (append text)
- force numeric or uppercase
- reverse input
- input length limitations
- signal on all keystrokes

COM Controls

COM (Common Object Module) controls have been added to NOMADS for OLE, OCX, and ActiveX support. This control type replaces exisiting VBX controls and supports COM event handling.

The Providex COM interface supports method calls, properties, event handling, as well as post-create logic. An object type of lowercase `"o"` is used to represent these controls in NOMADS in the library file containing the COM records.

The existing properties, methods, and events for COM controls are automatically displayed in NOMADS once a selection is made from the COM list. This list is displayed using the **DEF OBJECT** directive; i.e.,

DEF OBJECT *com_id*,"*"

NOMADS reserves a CTL value between 22001-22999 for the event handling.

Defining COM Controls

The following steps outline how to set up a COM control:

Step 1. Open COM Controls Panel

In the designer, select the COM control from the Controls window or through the menu items Controls>COM.

Step 2. Select a COM Control from List

Press the query button to the right of the COM Name to get a list of all the available COM controls on the system.

Step 3. Define Properties

Once a COM control is selected, the Properties/Methods tab will be loaded automatically. A sample screen shot is provided on the following page.

The Properties/Methods tab contains all of the properties and methods available for the COM control. Methods are identified by empty brackets appended to their name; e.g., AboutBox( ). Although Methods are displayed in the NOMADS Properties/Methods grid, they are not accessible during the design phase. If an application needs to access one or more of these methods, then it must do so either in the Post-Create Logic, or from within the application itself.

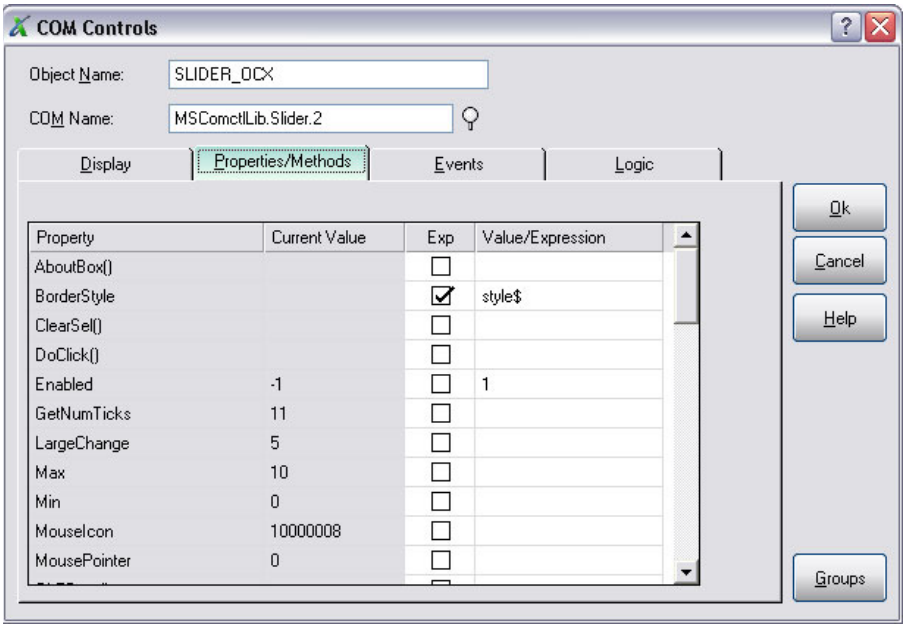
Providex properties are prefixed with Pvx. Both properties and methods are sorted in alphabetical order.

The current value for a property is displayed in the column next to the property name. The text <unreadable> is displayed if the property is inaccessible.

If the assigned value for a property is an expression, place a check mark in the Exp column for that property. The default is no check mark, indicating a literal value.

Use the Value/Expression column to assign a value to the property after the COM control is created. If the property is read-only then this cell will be locked.

The Properties/Methods tab appears as follows:



Step 4. Define Events for COM

The Events tab contains all the events available for the COM control. A sample screen shot is provided on the following page.

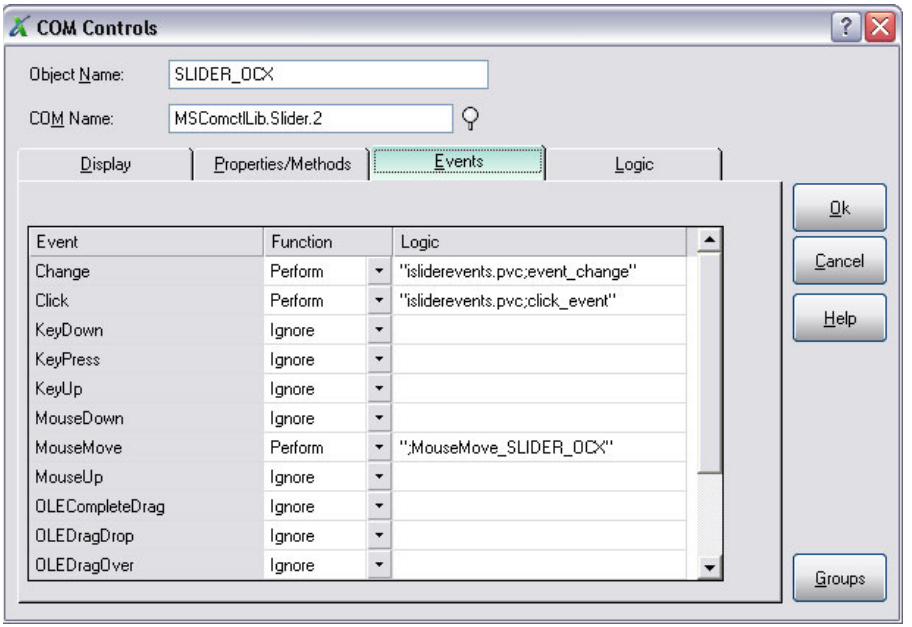
These event names are obtained from the **'PvxEvents$** property. They are loaded into the Event column in alphabetical order.

The Function column describes how the event (if any) will execute. Preset values (Ignore/Link/Perform/Call/Execute) are loaded into the drop list.

Ignore	No event occurs
Link	Invoke a panel Syntax: "*object name*","[*library*]"[, *arg_1$*,*arg_2$*...*arg_20$*]
Perform	Perform a program with optional label entry point. (For proper syntax, refer to the **PERFORM** directive in the *Language Reference Manual*.)
Call	Call a sub-program with optional label entry point. (For proper syntax, refer to the **CALL** directive in the *Language Reference Manual*.)
Execute	Run a series of Providex commands separated by semi-colons.

The Logic column will contain the associated logic that will execute when an event signal is triggered.

The Events tab appears as follows:



A button will appear in a Logic cell if the Function is a Perform or Call. Press this button to launch the graphical editor.

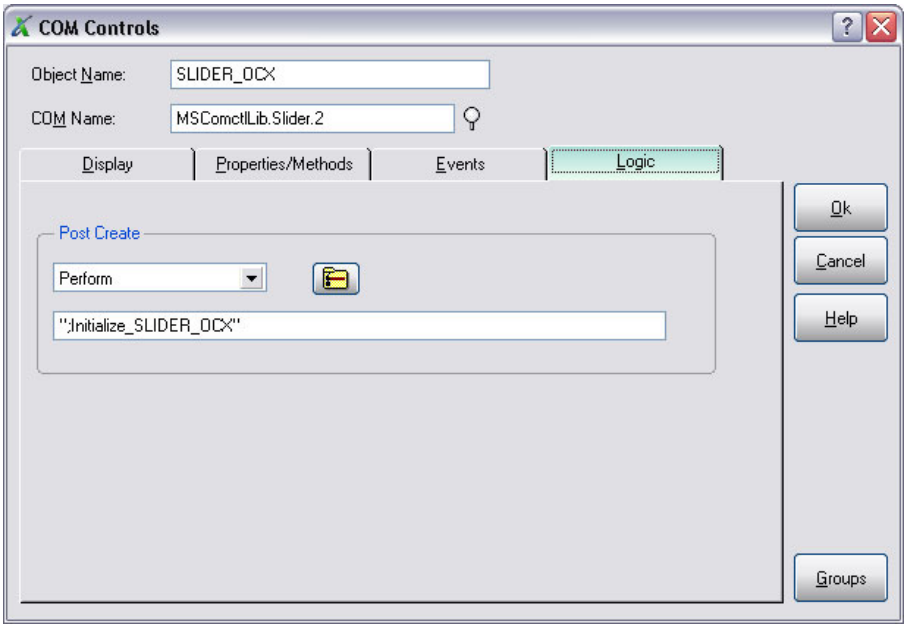
If the program and label *already exists* for the event, then the editor will be positioned to that label automatically.

If the program label on a event *does not exist*, then NOMADS will generate a label automatically using the *Event_Name+"_"+Control_Name* and insert it in the program. The program will be created if it does not exist.

Step 5. Assign Post-Create Logic

Enter the program/entry point to be performed, called or executed after the COM control is created. A sample screen shot is provided on the following page.

The Logic tab appears as follows:



How NOMADS Handles Events

COM controls generate a unique CTL value for each event defined in NOMADS. NOMADS reserves CTL values between 22001-22999. These are loaded into a table at runtime after the COM is created using:

ON EVENT "*event_name*" *FROM COM_id* **PREINPUT** *CTL_value*

Note: COM_id is stored in *control_name*.ctl.

When an event occurs, NOMADS will scan the event CTL table. If a match is found then the logic for that event is executed.

Order of Execution

1. COM control is created using **DEF OBJECT**
2. Post-create logic is applied.
3. Properties are assigned.
4. Events are assigned using:
   ON EVENT "*event_name*" **FROM** *COM_id* **PREINPUT** *CTL_value*