

# NatStar

---

Version 6.00 Edition 1

# NS-DK

---

Version 6.00 Edition 1

## Controls

*Information in this document is subject to change without notice as a result of changes in the product. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is illegal to reproduce the software on any medium unless specifically authorized within the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Nat System.*

© 2008 Nat System. All rights reserved

The Nat System name and logo are registered trademarks of Nat System.

All other trademarks mentioned in this document are registered trademarks of their respective companies.

Ref. n° NSR500CTRL0003

## Contents

About this Manual.....	iii
Organization of the manual .....	iii
Conventions .....	iii

### Chapter 1 Using controls

Creating and modifying controls.....	1-5
Creating controls .....	1-5
Modifying controls .....	1-6
Presentation text fields: static text and group boxes .....	1-9
Entry Fields .....	1-13
Radio buttons .....	1-19
Simple radio button .....	1-20
Radio buttons control (group of radio buttons) .....	1-21
Lists (list boxes, combo boxes and CBEs) .....	1-27
Lists presentation.....	1-27
Descriptor Syntax .....	1-28
Handling lists.....	1-43
Classifier control .....	1-53
Classifier control components and usage.....	1-54
Creating a Classifier control and defining its settings .....	1-55
Behavior of a Classifier control with the look of Windows XP .....	1-61
Programming Classifier controls .....	1-61
Check boxes .....	1-89
Scroll bars (horizontal and vertical) .....	1-92
Push Button and Picture Button .....	1-94
Icons .....	1-99
Bitmap images.....	1-100
Menus.....	1-103
Multiple line entry fields (MLEs) .....	1-105

## Chapter 2 32-bit Windows controls

Introduction .....	2-4
Definition of 32-bit Windows controls .....	2-4
Displaying 32-bit Windows controls .....	2-6
Methods for using a 32-bit Windows control .....	2-9
Presentation of 32-bit Windows control.....	2-11
Animation .....	2-11
Date and Time Picker .....	2-13
HotKey .....	2-14
List View .....	2-15
Month Calendar .....	2-18
Progress Bar.....	2-19
Track Bar .....	2-20
Tree View .....	2-22
Static settings for 32-bit Windows controls .....	2-24
Creating a control .....	2-24
Manipulating 32-bit Windows controls.....	2-68
Introduction .....	2-68
Installation .....	2-69
Dynamic parameters .....	2-70
Using List Views and Tree Views.....	2-90
The List View control.....	2-90
The Tree View control.....	2-108
Example of use .....	2-129
Reference of the NSMISC library .....	2-130

## About this Manual

This is the "Controls" manual which describes how to use the controls in NatStar and NS-DK.

The SheetBox and 32-bit Windows controls are much more complex, so they have their own chapter.

## Organization of the manual

This manual contains three chapters.

<b>Chapter 1</b>	<b>Using controls</b> This chapter describes the standard controls of NatStar and NS-DK.
<b>Chapter 2</b>	<b>32-bit Windows Control</b> This chapter presents the 32-bit Windows controls implemented in NatStar and NS-DK

## Conventions

### Typographic conventions

<b>Important term</b>	Important terms are printed in <b>bold</b> .
<i>Interface component</i>	The names of windows, dialog boxes, controls, buttons, menus and options are printed in <i>italics</i> .
[F9]	Function key names appear in square brackets.
FILENAME	Filenames are printed in UPPERCASE.
<code>syntax example</code>	Syntax examples are printed in a fixed-width font.

### Notational conventions

- A round bullet is used for lists

- ◆ A diamond is used for alternatives
- 1. Numbers are used to mark the steps in a procedure to be carried out in sequence

## Operating conventions

Choose XXX \ YYY	This means you need to open the XXX menu, then choose the YYY command (option) from this menu. You can perform this action using the mouse or mnemonic characters on the keyboard.
Click the XXX \ YYY button	This means you need to display the tool bar named XXX, then click the YYY button in this tool bar (the name of each button is shown by its help bubble). You can only perform this action with the mouse.
Choose the XXX button	This means you need to choose the XXX button in a dialog box. You can perform this action using the mouse or mnemonic characters on the keyboard.

## Icon codes



**Comment**, note, etc.



**Reference** to another part of the documentation.



**Danger**: precaution to be taken, irreversible action, etc.



**Suggestion**: helpful hints, etc.



**To go a step further**: level of detail or expertise greater than the average level of the document.



Indicates specific information on using the software under DOS-Windows (all versions).



Indicates specific information on using the software under DOS-Windows 32 bits.



Indicates specific information on using the software under DOS-Windows 32 bits.



Indicates specific information on using the software under Unix systems.

## Chapter 1

# Using controls



**Controls** are graphical objects that can be manipulated inside a window in NatStar/NS-DK. This chapter provides a thematically arranged description of how to design controls in a window. You will learn about the static characteristics of all the controls available in NatStar/NS-DK.



Their dynamic characteristics, i.e, those that can be modified by programming, are described in the « NCL Language Reference » manual.

### ***This chapter explains***

- How to create and modify a control
- How to use Nat System's standard controls in your windows



## Contents

Creating and modifying controls .....	1-5
Creating controls 1-5	
To create a control 1-5	
To create a menu 1-6	
Create a menu command 1-6	
Modifying controls 1-6	
Modify control name 1-7	
Modify text associated with a control 1-7	
Modify position and size of a control 1-7	
Modify font used for control text 1-7	
Presentation text fields: static text and group boxes .....	1-9
To associate a text field with an entry field 1-9	
To set relief for a group box 1-11	
Group controls together in a group box 1-12	
Entry Fields .....	1-13
Use keyboard with Entry field 1-13	
Set character display format 1-15	
Validate fields during input 1-16	
Validate fields after input 1-16	
Ensure that the user enters the number of characters specified in the Length field 1-17	
Initialize Entry field when window is opened 1-17	
Save Entry field contents when window closes 1-17	
Define Spin Button field 1-17	
To display the relief under Windows XP 1-18	
Radio buttons .....	1-19
Simple radio button 1-20	
Define reference button 1-21	
Associate other radio buttons with reference button 1-21	
Radio buttons control (group of radio buttons) 1-21	
Set text for a group box associated with a group of radio buttons 1-22	
Define radio button appearance 1-22	
Set text and values for each radio button 1-23	
To fix dynamically the radio button 1-23	
Lists (list boxes, combo boxes and CBEs) .....	1-27
Lists presentation 1-27	
Descriptor Syntax 1-28	
General Rules 1-33	
List characteristics 1-33	
Overriding a format 1-34	
Using types 1-34	
Background (B) and Foreground (F) 1-34	
Qualifiers 1-34	
Borders and grids (G) 1-35	
Justification (J) 1-35	
Inserting icons or bitmaps (M) 1-35	
Resizable column (R) 1-36	
Locking lines at the top of the list (T) 1-37	

<i>Copies attributes of the previous cell or column (C)</i>	1-37	
<i>Inserting a pointer to a text string (P)</i>	1-37	
<i>Extensions for displaying bitmaps with a transparent color</i>	1-38	
Examples	1-39	
<i>Inserting a line with local presentation descriptors</i>	1-39	
<i>Using the COL_*% constants</i>	1-40	
<i>Example of global formatting</i>	1-40	
<i>Applying a new global format</i>	1-41	
<i>Local presentation descriptors in a MOVE instruction</i>	1-41	
<i>Inserting bitmaps</i>	1-41	
<i>Defining a resizable column</i>	1-42	
<i>Locking the first two lines</i>	1-42	
<i>Displaying the selection in a column</i>	1-42	
Handling lists	1-43	
<i>To deselect quickly all lines (except one)</i>	1-43	
<i>Using the keyboard with a list box</i>	1-44	
<i>Load a file into a list</i>	1-44	
<i>Preselect a line</i>	1-46	
<i>Format a file</i>	1-46	
<i>Save the selected line when the window is closed</i>	1-47	
<i>Prevent list height adjustment</i>	1-47	
<i>Set CBE characteristics</i>	1-48	
<i>To display the relief of a CBE under Windows XP</i>	1-49	
<i>Set the characteristics of a combo box</i>	1-50	
<i>To fix the height of a Combo Box</i>	1-51	
Classifier control.....		1-53
Classifier control components and usage	1-54	
<i>Describing pages</i>	1-55	
<i>Using a Classifier control at run-time</i>	1-55	
Creating a Classifier control and defining its settings	1-55	
<i>Create a Classifier control</i>	1-55	
<i>Define a Classifier control's settings</i>	1-56	
<i>Describing the contents of a tabbed window</i>	1-58	
<i>Configuring tabbed windows' appearance</i>	1-59	
Behavior of a Classifier control with the look of Windows XP	1-61	
Programming Classifier controls	1-61	
<i>Retrieve the number of tabs in a Classifier control</i>	1-62	
<i>Select a tab in a Classifier control</i>	1-62	
<i>Retrieve the index of the selected tab</i>	1-62	
<i>Retrieve the window handle associated with a page in a tabbed window</i>	1-62	
<i>To access a control within a window</i>	1-63	
<i>To retrieve the classifier control internal handle used in the TAB_* functions</i>	1-63	
Dynamic parameters	1-66	
Functions and instructions	1-72	
Check boxes.....		1-89
<i>Initialize a check box</i>	1-89	
<i>Define a 3-state check box</i>	1-91	
<i>Adjust the check box's size to the text displayed</i>	1-91	
Scroll bars (horizontal and vertical).....		1-92
<i>Use the keyboard with a scroll bar</i>	1-92	
<i>Define the scrolling interval</i>	1-92	
Push Button and Picture Button .....		1-94
<i>Validate controls in a window by selecting a button</i>	1-95	

## 1-4      *Using controls*

---

*Close a window by selecting a button*      1-96  
*Open a window by selecting a button*      1-96  
*Prevent the button from getting the focus when its keyboard accelerator is selected*      1-97  
*Set the image displayed in a picture button*      1-97  
*Set the transparency color values of the Picture button bitmap*      1-97

### Icons ..... 1-99

*To select an icon*      1-99

### Bitmap images ..... 1-100

*To specify the associated bitmap file*      1-100  
*Create or display the bitmap file* 1-101  
*Load the bitmap file when the application starts up*      1-101  
*Define a bitmap with push button characteristics* 1-101  
*Define a bitmap with check box characteristics* 1-101  
*Adjust the size of a bitmap*      1-102

### Menus ..... 1-103

*Associate a keyboard accelerator with a menu option*      1-103  
*Close a window or open another window by selecting a menu item*      1-103

### Multiple line entry fields (MLEs) ..... 1-105

*Use keyboard with an MLE*      1-105  
*Disable input in an MLE* 1-106  
*Format lines in an MLE* 1-108  
*Give the focus to the next field when the user presses the [Tab] key*      1-108  
*Prevent the MLE's height from being adjusted*      1-108  
*Display a file in an MLE* 1-108

## Creating and modifying controls

Controls are accessible from the *Controls* menu or the tool bar.



A number of additional controls can be accessed from the *Custom* tool bar: these are custom controls supplied with NatStar / NS-DK. You can also enhance the tool bar with buttons that access your own custom controls.

The buttons in these tool bars are disabled if the window displayed doesn't belong to the Dialog class.



### *Creating other kinds of controls*

You can also create reusable sets of controls called **templates** (see Chapter 23 in the « User manual »). Lastly, it is also possible with NatStar/NS-DK to create your own users' controls called **custom controls**.



For more information about creating your own custom controls, see the Graphical Builder part of the "Development Guide."

## Creating controls

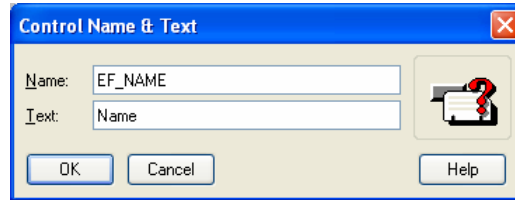
Some controls, like push buttons, are usually associated with text. This text serves as the control's label in the application's user interface.

For controls like menus, push buttons or radio button controls, you can use their text to create a keyboard shortcut by using the '~' character. This character indicates that the subsequent letter will be used as the menu's shortcut key. In the finished application, this letter will be underlined.

If this key isn't available on your keyboard, just press [Alt Gr] + [2] or [Alt] + [1],[2] then [6] (the ASCII code for the ~ character).

### **To create a control**

1. Edit the window or template you want to create a control in.
2. Drag the control's button from the *Controls* or tool bar to the desired position in the window. Depending on the kind of control, either the *Control Name* or the *Control Name & Text* box opens.




The creation box automatically proposes a default name for each new control.

3. Enter its name and, if required, its text.
4. Choose *OK* to confirm the creation.

If the new control has no associated text, the creation dialog box will only contain the *Name* field.

### ***To create a menu***

1. Check the *Menu Bar* option in the properties pane.
2. Drag the *Controls \ Standard \ Menu* button or the  icon to the menu bar. The *Control Name & Text* dialog box opens.
3. Enter the menu's name and text.
4. Choose *OK* to confirm the creation.

### ***Create a menu command***

1. Check the *Action Bar* option in the window's properties pane.
2. Drag the *Controls \ Menu* button to the menu bar. The *Control Name & Text* dialog box opens.
3. Enter the menu's name and text.
4. Choose *OK* to confirm the creation.




Dragging a menu control to an existing menu opens the menu, allowing you to insert a command into it. This action is equivalent to checking the *Pulldown Menu* option in the properties pane, which allows you to create a pulldown menu (for a menu) or a cascading menu (for a menu command).

## **Modifying controls**

As with windows, the settings that define the characteristics of a control are grouped together in the properties pane.

There are several ways to open the properties pane:

- ◆ Double-click the control with the right-hand mouse button.
- ◆ Click on the *Properties* icon  of the toolbox.

### **Modify control name**

1. Modify the *Name* field. This name must be unique. It is only intended for the programmer, and is used to identify the object in NCL.
2. Choose *OK* to confirm.


### **Modify text associated with a control**

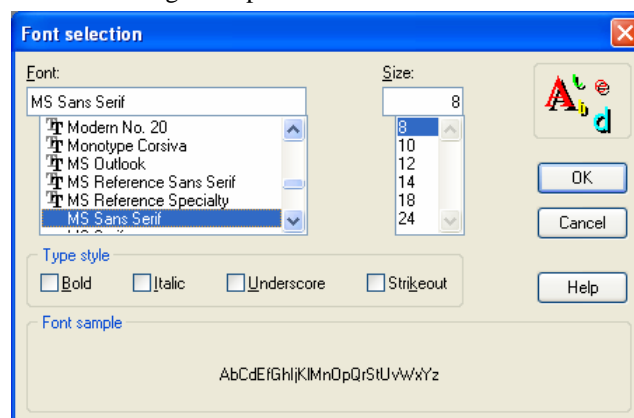
Modify the text in the *Text* field, then choose *OK*.

### **Modify position and size of a control**

1. Specify the control's horizontal position in the *X* field and its vertical position in the *Y* field.  
These positions are expressed in pixels from the lower left-hand corner of the window the control belongs to.
2. Specify the control's width in the *Width* field, and its height in the *Height* field.
3. The modifications are automatic.

### **Modify font used for control text**

1. Select the  button that appears when you select the *Fonts...* field. The *Font selection* dialog box opens.



2. Select the required font, size and style, then choose *OK* to confirm.

### Other actions concerning controls

<b>To disable a control when the window is opened</b>	Check the <i>Disabled</i> option.
<b>To modify the colors used for a control</b>	Select the text color from the <i>Foreground Color</i> field, then the background color from the <i>Background Color</i> field in its properties pane.
<b>To give a control a 3D effect</b>	<p>Select one of the options in the <i>Shadow</i> group:</p> <ul style="list-style-type: none"><li>• <i>None</i>                      standard appearance</li><li>• <i>Light</i>                      convex appearance</li><li>• <i>Dark(sunken)</i>              concave appearance</li></ul> <p>Two other options, <i>Light'opt</i> and <i>Dark'opt</i> are specific to group boxes and radio buttons. For more information, see the sections on these controls.</p>
<b>To justify the text in a control</b>	<p>Select one of the options in the <i>Justification</i> group:</p> <ul style="list-style-type: none"><li>• <i>Left</i>                      left justified</li><li>• <i>Centered</i>                      centered</li><li>• <i>Right</i>                      right justified</li></ul>
<b>To set the control that will get the focus after the current control when the user presses the [Tab] key</b>	Use the <i>Tab to</i> field to enter the name of the next control that will get the focus.

## Presentation text fields: static text and group boxes

Presentation text fields come in two forms: a simple static text field used as a field prompt, or a group box used to group together a set of controls under the same heading. These are known as static controls: no action is associated with them when the application runs. Consequently, no events can be programmed for this type of control.

Static text and group box controls generally include a brief description (often a single word) of the field(s) that the user must fill in for an associated entry field.

The definition of a keyboard shortcut for a static text or group box (one of its letters is underlined) allows the application's user to move rapidly to the associated entry field (by pressing [Alt] + the underlined character).

### ***To associate a text field with an entry field***

1. Open the properties pane for the static text control or group box used as the text field.



TNAME	
- Main	
Name	TNAME
Description	
Text	Na~e :
Tooltip	
Link to	ENAME
Justification	Left
- Layout	
X	17
Y	359
Width	46
Height	16
Auto size	<input checked="" type="checkbox"/>
Anchor	
- Look	
Shadow	None
Half-tone	<input type="checkbox"/>
Erase rect	<input checked="" type="checkbox"/>
Word-wrap	<input type="checkbox"/>
- Font	
Name	
Size	0
Bold	<input type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	Default
Background color	Default

2. Use the *Link to* field to enter or select the name of the entry field you want to associate with it. The list displays the input fields available in the window.
3. The modifications in the properties pane have been taken into account automatically.

### Other actions concerning text fields (using the properties pane)

<b>To gray the text in a text field</b>	Check the <i>Half-Tone</i> option.
<b>To erase the rectangular background in a text field</b>	Check the <i>Erase Rect</i> option. The background color will be the one specified in the <i>Background Color</i> field. If this option isn't checked, the rectangular background will be erased and the background color will be the same as the window's.
<b>To define a multiple-line text field (static text only)</b>	Check the <i>Word Wrap</i> option. If the height of the control allows several lines of the text and the text is too long to fit on one line, it will be formatted so that words are not truncated. The next word will be automatically placed on the following line if it doesn't fit on the current line.
<b>To make all the text visible (for static text only)</b>	Check the <i>Auto Size</i> option. Thus if the text is too long for the static text, the static text's width will be automatically modified to display the entire text.

### To set relief for a group box

GROUP1	
- Main	
Name	GROUP1
Description	
Text	~Search
Tooltip	
Link to	ESEARCH
Justification	Left
- Layout	
X	471
Y	104
Width	108
Height	282
Anchor	
- Look	
Shadow	None
Erase rect	<input checked="" type="checkbox"/>
Half-tone	<input type="checkbox"/>
- Font	
Name	
Size	0
Bold	<input type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	Default
Background color	Default

Select one of the options in the *Shadow* group. Note that the text will no longer be visible if you select *Light'opt* or *Dark'opt*.

- *None* standard appearance
- *Light* convex appearance
- *Light'opt* convex appearance without text
- *Dark(sunken)* concave appearance
- *Dark'opt* concave appearance without text

### **Group controls together in a group box**

It is important to create the group box BEFORE the controls, since placing the group box over existing controls will hide them.

There is a way of creating a group box AFTER the controls, but this involves the following operations:

- Create the group box next to the controls you want to include in it.
- Take each of the controls and move them into the group box.
- Take the group box and move it to the position where the controls were previously located (the controls inside a group box automatically move with it).



To select a group box, click the control's text or border.

## Entry Fields

An entry field is a single line input field that allows the user to enter text. The CBE control behaves in the same way as an entry field, but is combined with a combo box.

When an empty entry field gets the focus, a blinking cursor is displayed in it to indicate where characters will be input. The blink rate is set in the system's Control Panel. The field's contents are initially highlighted. A cursor direction key or mouse must then be used to position the cursor within the field. The keyboard and the mouse are handled automatically for this field together with the auto-repeat function when a key is held down.

### ***Use keyboard with Entry field***

- [Home] moves the cursor to the beginning of the line, before the first character.
  - [End] moves the cursor to the end of the line, after the last character.
  - [Shift]+[Home] selects all characters between the cursor and the beginning of the line.
  - [Shift]+[End] selects all characters between the cursor and the end of the line.
  - [BackSpace] deletes the previous character.
  - [Left] moves the cursor to the previous character.
  - [Right] moves the cursor to the next character.
  - [Ctrl]+[Left] moves the cursor to the beginning of the previous word.
  - [Ctrl]+[Right] moves the cursor to the beginning of the next word.
  - [Shift]+[Left] adds the previous character to the selection.
  - [Shift]+[Right] adds the next character to the selection.
  - [Del] deletes all selected characters.
  - [Shift]+[Del] deletes all selected characters and stores them in the clipboard. This is known as a *Cut* operation.
  - [Ctrl]+[Ins] stores all selected characters in the clipboard. This is known as a *Copy* operation.
  - [Shift]+[Ins] inserts the contents of the clipboard at the cursor position. This is known as a *Paste* operation.
-



When text has been selected, or when the entry field gets the focus (in which case, selected characters are highlighted), all selected characters are deleted as soon as a character is typed! To avoid this, you must first click the required position in the field or use a cursor key.

[Tab] doesn't generate a Tab character - it moves to the next control.

**Set character display format**

Main	
Name	ENAME
Description	
Tooltip	
Input	
Mode	Normal
Type	Default
Max. length	31
Characters	
Format	
Tab to	ELASTNAME
Justification	Left

Events	
INIT	
HELP	
CHANGED	<input checked="" type="checkbox"/>
GETFOCUS	
LOSEFOCUS	
CHECK	

Layout	
X	87
Y	357
Width	127
Anchor	

Behavior	
Upcase	<input checked="" type="checkbox"/>
Skip blanks	<input type="checkbox"/>
No blanks	<input type="checkbox"/>
Auto-tab	<input type="checkbox"/>
Condition	None
Hide text	<input checked="" type="checkbox"/>
Locked text	<input type="checkbox"/>
Disabled	<input type="checkbox"/>
Auto-scroll	<input checked="" type="checkbox"/>

Deprecated	
Output	

Look	
Shadow	None
Margin	<input checked="" type="checkbox"/>
Font	0

Name: ENAME

Check *Upcase* to automatically convert input characters from lowercase to uppercase.

Check *Hide Text* to allow passwords to be entered: characters entered will not be displayed, but instead be replaced with asterisks (\*).

Both these conversions occur as characters are typed.

### ***Validate fields during input***

The characters typed in a field can be checked as the user presses the keys: an invalid key results in a beep.

By default, the maximum number of characters accepted for input is 31. You can use the *Max. length* field to change this value.

By default, all characters are allowed for input. To prohibit spaces, check *No Blanks*. To restrict the range of characters accepted, use the *Characters* field.

For example, to allow only uppercase characters *ABCDEFZ*, type the following in *Characters*:

```
'A' . . 'F' , 'Z'
```

### **Notes on Entry Fields**

- Characters must be placed between apostrophes (').
- To indicate a range of characters (in their ASCII code order), separate the limits with two periods (..).

### ***Validate fields after input***

In addition to the checks made during input, checks can also be made after input, when the user closes the window containing the entry field. When the user attempts to close the window (by choosing **OK**, for example), the focus will be automatically positioned on the entry field if its contents are invalid.

Check *Not empty (required)* to specify that the entry field cannot be left blank.

Select *Integer*, *Number*, *Date* or *Time* to specify that the sequence of characters entered must be an integer, real number, date or time.



Beware of conflicts! Obviously, if you only authorize the characters 'A' . . 'F' , 'Z' in *Characters*, and also select *Integer*, the user will never be able to leave the window! Therefore define your checks carefully to avoid this type of contradiction.

- *Default* all formats accepted.
- *Integer* integer between -2147483647 and +2147483647.
- *Number* real number between +-4.19E-307 and +-1.67E+308.

The thousand and decimal separators recognized are those set in the system's Control Panel. They can, however, be modified using certain NSMISC library functions.

- *Date*      date
- *Time*      time

### ***Ensure that the user enters the number of characters specified in the Length field***

Select the *Force Fill* option in the *Condition* field of the properties pane.

Note that the field can still be left blank if *Not empty (required)* isn't selected in the *Condition* field, but it can never be partially filled. This check is made when the user closes the dialog box that contains the field.

### ***Initialize Entry field when window is opened***

You can have the entry field initialized with a default string by typing one of the following in the *Input* field:

- ♦ A string of characters without quotation marks = a variable name,
- ♦ A string of characters with single quotation marks = an alpha value,
- A numeric value without quotation marks.

### ***Save Entry field contents when window closes***

Enter a variable name without quotation marks in the *Output* field. When the window is closed, this variable will contain the string entered in the field.

### ***Define Spin Button field***

Select the *Spin Button* option in the *Mode* field in the entry field's properties pane.

This option adds two vertical arrows to the right of the entry field that display a series of integers when they're clicked. The up arrow increases the number, and the down arrow decreases it.

Selecting this option forces the entry field format to *Integer*. The default integers displayed range from 0 to 65535.

Contents of the *Input* field in this properties pane:

"init	minimum	maximum	incrément"
-------	---------	---------	------------



The maximum allowed is 65535.  
This option is not available for CBEs.



### Other actions on entry fields (using the properties pane)

<b>To disable input in an entry field</b>	Check the <i>Locked Text</i> option.
<b>To enable auto-scroll for an entry field</b>	Check the <i>Auto-Scroll</i> option. The contents of the entry field will be automatically scrolled when the number of characters entered exceeds the field's size. Any scrolled characters that have become invisible are not lost. Scrolling occurs as characters are input.
<b>To automatically move the focus to the next field</b>	Select the <i>Auto-Tab</i> option. The next control will automatically get the focus when the number of characters typed in the field reaches the value in <i>Max. length</i> .
<b>To automatically eliminate any spaces at the beginning or end of the field</b>	Select the <i>Skip Blanks</i> option. This option doesn't eliminate any embedded spaces. The adjustment is made after input.
<b>To frame an entry field</b>	Check the <i>Margin</i> option.
<b>To display scrolling arrows in an entry field</b>	Select the <i>Scrollable</i> option in the <i>Mode</i> field. This displays two horizontal arrows to the right of the entry field. These arrows are used to scroll the field when the number of characters entered exceeds the size of the field.
<b>To adjust the height of the entry field based on its contents.</b>	<p>Check the <i>Exploding</i> option. The height of the entry field will change if the text entered exceeds the width of the field, ensuring that all the characters input are visible.</p> <p>After input, when the cursor leaves the entry field, the field will return to its initial size.</p>

### To display the relief under Windows XP

The Windows XP system disables, by default, the display of the controls relief. To bypass this problem, select the classic Windows theme or set in the NSLIB.INI file the parameter `UseLookXP` to `False`.



For more information about the NSLIB.INI file, See the "NSLIB.INI file" manual.

## Radio buttons

A radio button is a small round button with associated text displayed to its right. When a radio button has the focus, a dotted box surrounds the associated text.



A radio button has only two states: selected (the circle is filled in black), or deselected (the circle is empty). When it has the focus, it can be selected by clicking it, or by typing its keyboard shortcut.

A radio button cannot be deselected directly. It usually belongs to a group of radio buttons that act like a set of mutually exclusive options. There is always one radio button selected in a group at any time. When a radio button is selected, all the other radio buttons in the group are deselected. The cursor keys are handled automatically and are used to move from one radio button in a group to another.

A radio button never appears on its own. A group of radio buttons represents a set of mutually exclusive options.

In the example below, a person can only be either male or female: selecting both "Male" and "Female" or not selecting either choice must be prohibited.

*There are two alternative ways of creating radio buttons:*

- ◆ Create a group of radio button controls ,
- ◆ Create a radio button control, which automatically sets the number of choices available .

## Simple radio button

RB_CELIB	
- Main	
Name	RB_CELIB
Description	
Text	Célibataire
Tooltip	
Input	1
Tab to	RB_VEUF
Group	=
Value	1
- Events	
INIT	
HELP	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	
- Layout	
X	47
Y	179
Width	86
Auto size	<input type="checkbox"/>
Anchor	
- Behavior	
Disabled	<input type="checkbox"/>
- Deprecated	
Output	
- Look	
Shadow	Light (raised)
- Font	
Name	MS Sans Serif
Size	8
Bold	<input checked="" type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	
Background color	Default

To create a group relationship between radio buttons, one of the buttons must be designated as the group's reference button.

**Define reference button**

1. Open the reference button's properties pane.

The *Group* field defaults to an equal sign ('='), which stands for the value of the control's *Name* field. Since this is the reference button, there's no need to modify *Group*.

2. Each button must have a unique value. Enter this value in the *Value* field.

Select an initial default value for the group of radio buttons. The value specified in the reference button's *Input* field means that the button with this value will be automatically selected when the window containing these buttons is initially displayed.

3. The modifications in the properties pane have been taken into account automatically.

**Associate other radio buttons with reference button**

1. Double-click with the right-hand button of the mouse each of the other radio buttons in the group to display their properties pane.
2. Type the name of the reference button in the *Group* field.
3. Enter a value for the radio button. Each button must have a unique value.
4. The modifications in the properties pane have been taken into account automatically.



The *Input* field is only used for the group's reference button.

**Other actions concerning radio buttons**

**To adjust the length of the radio button to the length of its text**

In the radio button's properties pane, select the *Auto size* option.

**Radio buttons control (group of radio buttons)**

The radio buttons control allows you to create a group of radio buttons automatically by simply setting parameters. This control is displayed as a group box containing the specified radio buttons.

RB_TITLE	
- Main	
Name	RB_TITLE
Description	
Tooltip	
Tab to	
- Events	
INIT	
TERMINATE	
HELP	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	
- Layout	
X	125
Y	25
Width	157
Height	126
Anchor	None
- Behavior	
Disable	<input type="checkbox"/>
Lock	<input type="checkbox"/>
Nb Columns	2
Title	
Columns	
- Look	
Font	MS Sans Serif;8
Foreground	Default
Background	Default
Shadow	Dark (sunken)

### **Set text for a group box associated with a group of radio buttons**

Enter the group's text in the *Title* field.

### **Define radio button appearance**

The *Nb Columns* field allows you to specify the number of columns used to arrange the radio buttons.

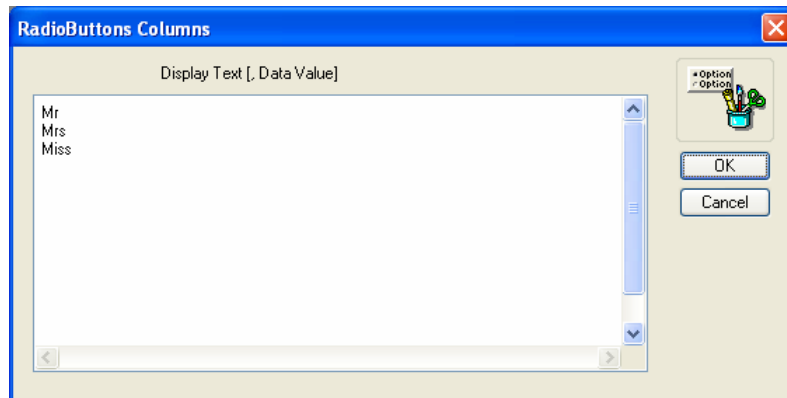
For example, if the specified number of columns is 1, the radio buttons will be left-aligned in a single column. If this field is set to 2, the radio buttons will be arranged in 2 columns.

The number of rows is determined by the number of radio buttons divided by the number of columns.

**Set text and values for each radio button**

Click on the *Press to Edit* button in the *Columns* field

. The *RadioButtons Columns* box appears.



Enter the text for each radio button in the list. Each text field corresponds to a separate radio button, and therefore, the number of radio buttons in the group is equal to the number of lines in this list.

The default value returned when a radio button is selected is the line number of the radio button in the list (starting from the index 0). Thus, the value returned by the first radio button will be 0, the second 1, etc.

To associate a specific value with a radio button, add this value after the radio button's text in the list, separating the two with a comma.

For example:

Masculin,M

means that the radio button "Male" will return the value "M" when it's selected.

**To fix dynamically the radio button**

## **.TITLE dynamic parameter**

To read or change the title of a group of radio buttons control.

**Syntax**                    *NameRadioButtonsGroup*.**TITLE**

**Reading/Writing**

## **.INDEX dynamic parameter**

Allows to fix the radio button index on which dynamic parameter TEXT will be applied. The index of the first radio-button is 0. The default value is -1.

**Syntax**                      *NameRadioButtonsGroup*.**INDEX**

**Reading/Writing**

---



## **.TEXT dynamic parameter**

Returns or modifies the text associated to a radio button of the group indicated by the dynamic parameter INDEX.

If the parameter INDEX is invalid (negative or surplus value): returns a null string in reading and does not do anything in writing.

**Syntax**                      *NameRadioButtonsGroup.TEXT*

**Reading/Writing**

## Lists (list boxes, combo boxes and CBEs)

A list control displays a list of choices from which the user can select a line (or, in some cases, several lines). There are several types of list controls: list boxes, combo boxes and CBEs.

A list box is a list combined with a vertical scroll bar. If the list contains more lines than the maximum number of displayable lines, the scroll bar can be used to scroll the lines up and down so that the hidden lines become visible.

A list box is often used to present the user with a choice. The user can then select a line from the list and, in some cases, several lines.

A line in the list can be automatically selected by default. When the list box has the focus, the current line is framed by a faint dotted box. The selected line is highlighted. A number of keys are then handled automatically.



List Box, Combo Box, CBE controls can not manage more than 32766 lines or an index with more than 32766 lines.



The using of the SheetBox control can replace usefully the List Box control. For more information about the SheetBox control, see the "Services Libraries Reference - Volume 1 - GUI and Printing APIs".

## Lists presentation

The standard List box and Combo box features can be enhanced using presentation descriptors.

These descriptors allow you to:

- color, align and define borders for the contents of lists,
- change the height of a line,
- lock one or more lines at the top of the list,
- insert icons or bitmaps,
- dynamically modify column widths at run-time.

Descriptors are specified with the INSERT or MOVE instruction.

A presentation descriptor can be applied to the intersection of a line and column (cell) in a List box or a Combo Box. In this case, it must be specified before each modified text string when the line is inserted into the list. This type of descriptor is known as a **local descriptor**:

---

```
INSERT AT xxx "<Descriptor><Text><Separator><Descriptor> <Text><Separator>
....." TO LISTBOX
```

A descriptor can also globally modify the characteristics of all the cells in one or more columns in the List box. This type of descriptor is known as a **global descriptor**.

This type of descriptor allows you to define the characteristics applied to any new lines of text inserted into the list. A global modification is specified by inserting a line that begins with the "~" character and consists only of descriptors:

```
INSERT AT xxx "~<Descriptor><Separator><Descriptor> <Separator>....." TO LISTBOX
```

Lines of text can then be inserted in the usual way:

```
INSERT AT xxx "<Text><Separator><Text><Separator>...." TO LISTBOX
```

## Descriptor Syntax

The syntax for a presentation descriptor is as follows:

**{Type[Qualifiers ],Type[Qualifiers ],...}**

Descriptors are enclosed between braces.

They apply to the cell corresponding to their position in the INSERT instruction.

Each component is separated by a comma.

A component is made up of a type identifier followed by the appropriate qualifiers in square brackets.

The type is identified by an upper-case character representing the type of presentation to be modified (color, justification, etc.). The type qualifiers define the selected options. A qualifier can be a letter, keyword or numeric value, depending on the type that it refers to. Some qualifiers can be combined.



Do not enter a space between the separator and the descriptor's brace or between the comma and the type indicator.

Example of a descriptor:

```
{B[BLUE],F[YELLOW],G[LRBT]}
```

which contains the types: B, F and G

and the qualifiers: BLUE, YELLOW, L, R, B and T.

Some qualifiers can only be applied globally.

For each type identifier, the following table describes its meaning, scope (local or global), qualifiers and their applications.

Type	Scope	Qualifier	Explanation/Applications
F	local & global	BLACK BLUE BROWN CYAN DARKGRAY GREEN MAGENTA RED WHITE YELLOW LIGHTBLUE LIGHTCYAN LIGHTGRAY LIGHTGREEN LIGHTMAGENTA LIGHTRED	Foreground color: color used for characters.
			Qualifiers must be entered in upper case.
B	local & global	BLACK BLUE BROWN CYAN DARKGRAY GREEN MAGENTA RED WHITE YELLOW LIGHTBLUE LIGHTCYAN LIGHTGRAY LIGHTGREEN LIGHTMAGENTA LIGHTRED	Background color.
			Qualifiers must be entered in upper case.
G	local &		

global

Grid: draws a border or partial border around a cell.

L

Left: draws a vertical line on the left.

R

Right: draws a vertical line on the right.

B

Bottom: draws a horizontal line below.

T

Top: draws a horizontal line above.

I

Light : draws in relief.

k

Dark : draws in hollows

n

None : cancels i and k.

l

Left: as L, but separated by one pixel.

r

Right: as R, but separated by one pixel.

b

Bottom: as B, but separated by one pixel.

t

Top: as T, but separated by one pixel.

A

Alternate: dotted line.

Using lower case qualifiers prevents column borders from overlapping.

Specifying the G type without qualifiers (G[]) cancels the border for a cell.

J

local  
&  
global

Justification: position of the text on the line in relation to the edges of the column.

C

Center: Centers the text.

L

Left: Left-justifies the text.

R

Right: Right-justifies the text.

Qualifiers must be entered in upper case.

M

local  
&

BitMap: inserts an icon or bitmap.

	global	Y N	Yes: Enables the bitmap. No: Disables the bitmap locally.  Qualifiers must be entered in upper case.
R	global		Resizable: the column can be dynamically resized using the mouse.  This type has no qualifiers (do not use the square brackets []).
T	global	n	Top Lock: locks one or more lines at the top of the list.  Numeric value indicating the number of lines to be locked at the top of the list.  Once locked, the specified lines cannot be scrolled vertically.
H	global	n	Height: height (in pixels) of the lines in the list.  Numeric value indicating the required height.  This type always applies to all lines in the list.
S	Local & global		Allow to change the colours of a selected item. The following table describes its meaning, scope (local or global), qualifiers and their applications.
			Show Selection
		B	BackGround  Only the background colour for the selection is applied. The colour of the characters remains that of a non-selected line.
		F	Foreground

			Only the colour of the characters in the selection is applied. The background colour of the selection remains that of a non-selected line.
		A	<p>All</p> <p>The background and text colour take on the selection colours (normal mode).</p>
		Y	<p>Yes</p> <p>The background and text colour take on the selection colours (normal mode).</p>
		N	<p>No</p> <p>The selected line of the column is not highlighted. The qualifiers must be specified in upper-case.</p>
C	Local & global		Recopies the attributes of the cell (local), or of the previous column (global).
P	Local & global		Inserts a pointer to some text in place of the text itself.
		Y	Yes: Enables pointer
		N	<p>No: Disables pointer</p> <p>Qualifiers should be specified in capital letters.</p> <p>The value of descriptor P is that it allows you to insert the whole length of a string that is longer than 255 characters as a line in a List Box.</p>

## General Rules

### List characteristics

Presentation descriptors concern:

- Multi-column List boxes
  - for which the *Tabulations* and *Separators* fields have been defined in the *Info* box.
  - or for which the dynamic *Tabulations* and *Separators* parameters have been defined.
- Multi-column Combo boxes for which the dynamic *Tabulations* and *Separators* parameters have been defined.

The values specified in the *Tabulations* field define the width (in pixels) of each column. E.g. 0,60,100,100 defines 4 columns whose widths are 0, 60, 100 and 100 respectively.

The first digit defines the width of column 0. This is a special column that cannot contain text.

This column is chiefly used to indent the data displayed in the list by the number of pixels defined as its width. Hence, the first column that contains text is column #1.

You cannot apply a local descriptor to column 0 since local descriptors are always specified in conjunction with formatted text.

Consequently, column 0 can only be modified globally. Valid types for this column are B, F, G and R (F only applies to border colors).



#### Applying global descriptors

```
INSERT AT xxx "~<Descriptor><Separator><Descriptor> <Separator> ....." TO LISTBOX
```

The first descriptor applies to **column 0**.

#### Applying local descriptors

```
INSERT AT xxx "<Descriptor><text><Separator><Descriptor> <text><Separator> ....." TO LISTBOX
```

The first descriptor applies to the first column containing text, i.e. **column 1**.



## Overriding a format

If a local format descriptor has been defined for a given line and column, its options will replace or supplement the options defined globally.

Similarly, the options defined by a global format will replace or supplement any options defined by a previous global format.

Formats are treated like text if the syntax used is incorrect.

## Using types

### *Background (B) and Foreground (F)*

You can specify a COL\_.\*% constant or its corresponding value instead of a color qualifier.

Use the following syntax:

[#"& COL_.*% &"]	using a COL_.*% constant
[#value]	using the corresponding value

If you specify a constant, you must convert it into a character string by concatenation.

Colors set in the Control Panel, such as COL\_BACKGROUND%, can also be used.

The F descriptor (Foreground color: character color) and B descriptor (Background color) have been improved. Henceforth, it is possible to configure character and background colors more accurately. The new syntax {B[%n]} and {F[%n]}, where n is an integer (decimal or hexadecimal), allows an RGB color to be indicated.

**For example:**

```
{B[%$C0D0FF]}
```

gives a background color of sky blue (red=C0, green=D0 and blue=FF).

```
INSERT AT 0 "~|{M[Y],H[30],T[1]}|{B[%$C0D0FF]}" to LB2
```

### *Qualifiers*

The B and F qualifiers descriptors can now be abbreviated. Only the first letters, allowing to distinguish between them, become compulsory



The DARK and LIGHT prefixes can be abbreviated into L and D. The spelling GREY (english) and GRAY (american) are allowed.

BLA = BLACK

BLU = BLUE  
G = GREEN  
C = CYAN  
R = RED  
M = MAGENTA  
BR = BROWN  
LGREY = LIGHTGREY  
LGRA = LIGHTGRAY  
DG = DARKGREY ou DARKGRAY  
LB = LIGHTBLUE  
LGREE = LIGHTGREEN  
LC = LIGHTCYAN  
LR = LIGHTRED  
LM = LIGHTMAGENTA  
Y = YELLOW  
W = WHITE

***Borders and grids (G)***

A cell can be given a full border by specifying all four qualifiers: LRTB or lrtb. The order in which they appear is irrelevant.

If all four grid qualifiers are defined for two adjacent cells, the cells will be separated by a double line. One way of avoiding this is by defining only a partial border (right and bottom border).

By default, borders are drawn with a continuous line. The A qualifier changes the line drawn to a dotted line.

***Justification (J)***

A string in a cell will overlap onto the next cell if the string is wider than the cell AND all the neighboring cells are empty. In other words, a string is not confined to one cell if the cell is the last one on the line containing text.

This also means that the last column in a list is as wide as the longest string in that column.

***Inserting icons or bitmaps (M)***

Bitmaps can be inserted anywhere in a List box, except in column 0.

A bitmap is identified by its handle:

---

- Pointer or system icon handles are expressed in ASCII. They must appear in the form ^## where ## is the value of a SPTR\_ \*% constant (see NCL Manual).
- bitmap file handles are obtained using the NCL CREATEBMP% function. This function associates an INT handle with a bitmap file.
- Example: MOVE CREATEBMP% ("c:\bmp\mybmp.bmp") to hbmp%

A cell cannot contain both text and a bitmap. If a cell is to contain a bitmap, it must be declared as such (by default, cells receive text).

This declaration can be made:

- when the bitmap is inserted, by concatenating its handle with a local descriptor with the M type and Y qualifier,  
or
- when a global format is defined, by specifying the M type with the Y qualifier for each cell that will contain a bitmap.

After this, a simple INSERT instruction with only the bitmap handle as its text is all that is needed to insert the bitmap.

You can still place text in a cell declared as containing a bitmap. To do this, you need to disable the bitmap locally, i.e. precede the text in a local descriptor with the M type and N qualifier when you insert it.

If the line is shorter than the bitmap, the bitmap will be truncated. In all cases, the bitmap will be centered vertically.

A bitmap can be repositioned horizontally using the J justification descriptor.

If the specified handle is invalid, a gray rectangle is displayed in the cell.

### ***Resizable column (R)***

To resize a column while the application is running, simply position the mouse pointer on the vertical line displayed between the column's right border and the next column's left border.

The mouse pointer changes to: "<-||->".

Hold the left mouse button down and drag the mouse to enlarge or shrink the column as required.

We recommend that you set different colors for each column or define them with the G[R] type: in this way, the dividing line between two columns will be clearly visible and the user will know where to position the mouse pointer to enlarge a column.

You cannot use another global presentation format to cancel the resizable attribute set for a column.

### ***Locking lines at the top of the list (T)***

Once locked, the first n lines in the list cannot be scrolled vertically and will be permanently displayed. They cannot be selected.

The main purpose of locking lines is to place column titles or general titles at the top of the list. These titles will remain visible, even if the list is scrolled vertically. However, they can still be scrolled horizontally with the columns.

Note: the indexes of the lines in the list are not modified by locking: locked lines are still counted.

### ***Copies attributes of the previous cell or column (C)***

The new presentation descriptor C copies the attributes of the previous cell (local) or column (global). It allows to avoid the repetition of the same attributes on several consecutive columns.

Moreover, you can add modification to copied attributes. For example, "{C,B[W]}" means that you want the same attributes as the previous cell or column except for the color which will be white.



Modifications are ignored if they precede the presentation descriptor C.

### ***Inserting a pointer to a text string (P)***

The new presentation descriptor P inserts a pointer to a text string at the place of the text itself. As for bitmaps or icons, the pointer must be expressed like an integer (decimal or hexadecimal).



The pointer must always be null (0) or valid (pointing on a CSTRING(n) where n <= 255). The List Box does not manage the de-allocation of the referenced text when the item is removed from the List Box or when it disappears, because it is not allocated by a NEW (it can be a CONST or a GLOBAL variable, or the text can be part of a SEGMENT).



The pointer must remain valid during its existence in the List Box. And, you must not use the address of a local variable or a function's parameter (except for a function CALLing the dialog box containing the List Box).

"{P[N]}" allows to deactivate locally the effect of a "{P[Y]}" global.

**Exemple :**

Declaration in the INIT of the window

```
GLOBAL MYSTRING$
mystring$='By Nat System e-mail pdupont@natsys.fr'
our List Box in this case has '|' for column separator.
Then in any event
Local int i(2)
local pointer h%
NOUPDATE LB3
; assigns the handle of a bitmap control (B1) to h%
move B1.text to h%
; the global format
INSERT AT 0 "~|{M[Y],H[30],T[1]}|{|}{P[Y]}" to LB3
for i = 1 to 10
INSERT AT END h% & "|Test"&"|"&@mystring$ to LB3
endfor
UPDATE LB3
```

**Extensions for displaying bitmaps with a transparent color**

A bitmap has a transparent color when the color of the bitmap is replaced by the background color of the List Box's cell.

Extensions allow the display of bitmaps with a transparent color and/or a horizontal slice of a bitmap (if the bitmap contains several small images, all the same width, next to each other).

**Syntax**

Let us say *hbm*p is the handle of the bitmap

*hbm*p<sup>ct</sup>

*hbm*p<sup>ct</sup><sup>it</sup><sup>It</sup>

*hbm*p<sup>ct</sup><sup>it</sup><sup>It</sup>

*ct* corresponds to the transparent color.

*ct* = TRANSP\_DEFAULT% (or -5) or TRANSP\_NEVER% (or -6) corresponds to an opaque color (default value if it is not specified).

*ct* >= 0 indicates a COL\_\*% color.

Values -4...-1 indicate that one of the corner pixel color, of the bitmap (or the slice), is used as a transparent color.

*ct* = TRANSP\_TOPLEFT% (or -1) corresponds to the upper left corner (or slice).

*ct* = TRANSP\_BOTLEFT% (or -2) corresponds to the lower left corner (or slice).

ct = TRANSP\_TOPRIGHT% (or -3) corresponds to the upper right corner (or slice).

ct = TRANSP\_BOTRIGHT% (or -4) corresponds to the lower right corner (or slice).

*it* corresponds to the index of the bitmap slice, 0 for the left.

*It* is the height (in pixels) of the bitmap slices.

Example

12345678^1 indicates a bitmap with a handle of 12345678 the upper left pixel color of which we want to use as a transparent color.

12345678^^3^16 indicates an opaque bitmap with a handle of 12345678 the 4th 16 pixel wide, slice (from the left) of which we want to display.

Using slices of bitmaps presents some advantages:

- The bitmap's slices allow to reduce the number of hidden bitmap controls containing the displayed bitmaps in one (or more) List Box and then to reduce the GDI resources use.
- With the possibility of a GIF or JPEG bitmap from v3.00 onwards, the slices of bitmaps permit also to reduce the size of binaries (in the case of many bitmaps). Actually, a big bitmap is generally better compressed than several little bitmaps.

## Examples

Let's assume that we have a List box, LB\_TEST, with the following information in its Info box:

Tabulation field: 0,70,70,70,70

Separators field: '-'



The & operator is used to concatenate two strings. Another way of splitting a line of NCL code in the middle of a character string is to avoid indenting the second part of the character string.

### ***Inserting a line with local presentation descriptors***

The following instruction centers aaa (dark gray) in column #1, centers bbb (red) in column #2 and right-justifies ccc (green) in column #3:

```
INSERT AT END " {F[DARKGRAY],J[C]}aaa-{F[RED],J[C]}bbb"&\
               "-{F[GREEN],J[R]}ccc" TO LB_TEST
```

Same instruction without the & operator:

```
INSERT AT END " {F[DARKGRAY],J[C]}aaa-{F[RED],J[C]}bbb\
-{F[GREEN],J[R]}ccc" TO LB_TEST
```

The following instruction inserts a column with a full border, blue characters and a yellow background:

```
INSERT AT END " {F[BLUE],B[YELLOW],G[LRTB]}border"\
TO LB_TEST
```

The following instruction inserts "hello" with blue characters, a white background and a border in column #1 followed by "world" with red underlined characters, and a light green background:

```
INSERT AT END " {F[BLUE],B[WHITE],G[TBLR]}hello"&\
"-{F[RED],B[GREEN],G[B]}world" TO LB_TEST
```

### ***Using the COL\_\*% constants***

Both the following instructions have the same effect, i.e. they write aaa with yellow characters in column #1:

```
INSERT AT END " {F[#"&COL_YELLOW%&"] }aaa" TO LB_TEST
INSERT AT END " {F[#14]}aaa" TO LB_TEST
```

### ***Example of global formatting***

Let's clear what we have inserted into the list and define a global format for it. Column #1 will have a green background and centered yellow characters with a border. Column #2 will have a white background and black characters with a border. Column #3 will have the same background as the list and red characters with a border. A non-overlapping border is defined for the columns by defining only a right border and bottom border for the cells.

This format is defined as follows:

```
INSERT AT 0 "~-{F[YELLOW],B[GREEN],J[C],G[RB]}"&\
"-{F[BLACK],B[WHITE],G[RB]}-{F[RED],G[RB]}"\
TO LB_TEST
```

After this, characters are written to the various columns in the list using the following format:

```
INSERT AT END "aaa-bbb-ccc" TO LB_TEST
INSERT AT END "ddd-eee-fff" TO LB_TEST
```

As a result, the column containing aaa and ddd is green. Characters are displayed in yellow and are centered within the column.



If we omit the '-' separator immediately before the '~', the format {F[YELLOW],B[GREEN],J[C]} will apply to column 0, which has a null width in our example.

### Applying a new global format

Now, let's define a new global format as follows:

```
INSERT AT 0 "~-----{B[YELLOW]} " TO LB_TEST
```

This format will not modify the presentation for columns 0, 1, 2, 3 and 4. The columns will keep the previously defined global format. However, the area in the List box located to the right of Column #4 (last column) will be yellow.

### Local presentation descriptors in a MOVE instruction

Let's suppose that the list now contains 8 lines (indexed from 0 to 7) which are formatted as defined previously. Let's modify the 4th line (index 3) so that column #1 has a blue background and white characters, column #2 has a white background and black characters and column #3 has a red background and white characters. All characters are centered in their respective columns.

Select the line and modify the text:

```
SELECT 3 FROM LB_TEST
MOVE " {F[WHITE],B[BLUE],J[C]}Nat"&\
      "-{F[BLACK],B[WHITE],J[C]}Systems"&\
      "-{F[WHITE],B[RED],G[],J[C]}USA" TO LB_TEST
SELECT 0 FROM LB_TEST
```

The characters in column 2 have a border since a **G** format has not been specified. However, characters in column #3 are no longer underlined since **G[]** has been specified.

Now, let's select the next line and modify its contents:

```
SELECT 4 FROM LB_TEST
MOVE "300-Winston-Drive-#722 07010 NEW JERSEY" TO LB_TEST
SELECT 0 FROM LB_TEST
```

The characters will have the attributes defined by the global format: column #1 will have a green background with centered yellow characters, etc.



The last text string specified with a MOVE or INSERT instruction is not truncated, even if it is wider than the column width defined in the tabulation field. For example, the text "#722 07010 New Jersey" will be displayed in the List box in full. This feature is very useful when inserting a main title in a List box.

### Inserting bitmaps

The following instruction indicates that column #1 will contain bitmaps:

```
INSERT AT 0 "~--{M[Y]} " TO LB_TEST
```

After this, the system bitmap "" can be inserted as follows:



```
INSERT AT END "^20-bbb-ccc" to LB_TEST
```

The following instruction inserts the bitmap identified by the handle hbm%:

```
INSERT AT END hbm% & "-bbb-ccc" to LB_TEST
```



The bitmap handle, which is an INT variable, must be concatenated with the inserted string.

The following instruction writes text in column #1, even though this column has been defined as containing a bitmap:

```
INSERT AT END "{M[N]}aaa-bbb-ccc" to LB_TEST
```

The following instruction inserts the bitmap into column #3, which has not been declared in the global format as containing a bitmap:

```
INSERT AT END "^20-bbb-{M[Y]}^20" to LB_TEST
```

### ***Defining a resizable column***

The following instruction indicates that column #1 is a resizable column:

```
INSERT AT 0 "--{R}" TO LB_TEST
```

### ***Locking the first two lines***

The following instruction locks the first two lines in the List box:

```
INSERT AT 0 "--{T[2]}" TO LB_TEST
```

### ***Displaying the selection in a column***

A window contains two controls ListBox (L1 et L2). The column separator is indicated by the character '@'.

```
DELETE FROM L1
NOUPDATE L1
INSERT AT 0 "~@{R}@{R,G[LRT],J[C],S[N]}" TO L1
; the selected element is not highlighted
insert ASCENDING " @20@1@3" To L1
insert ASCENDING " @30@1@3" To L1
insert ASCENDING " @12@1@3" To L1
UPDATE L1
;Setfocus
;ACTIVATEWINDOW
```

	01	1	3
	01	1	3
	05	1	3
	06	1	3
	09	1	3
	12	1	3
	15	1	3
	16	1	3
	20	1	3
	20	1	3
	22	1	3

```

DELETE FROM L2
NOUPDATE L2
INSERT AT 0 "~@{R}@{R,G[LRT],J[C]}" TO L2
; the selected element is highlighted
insert ASCENDING " @20@1@3" To L2
insert ASCENDING " @30@1@3" To L2
insert ASCENDING " @12@1@3" To L2
UPDATE L2
;Setfocus
;ACTIVATEWINDOW

```

	01	1	3
	01	1	3
	05	1	3
	06	1	3
	09	1	3
	12	1	3
	15	1	3
	16	1	3
	20	1	3
	20	1	3
	22	1	3

## Handling lists

### *To deselect quickly all lines (except one)*

In ListBox of the NatStar / NS-DK interface, you can shift the lines of the list with the bar of vertical run. To select a line, you click once above.

To deselect, you click above still. You can have several lines selected at the same time.

Before, it was necessary to click once on each line selected to deselect it, which was tiresome if there were much of it. For certain ListBox, NatStar/NS-DK enables you from now on to select a line once only and deselect all the others. It is enough to click on the line which you want to select with the right button of the mouse.

Among ListBox profiting from this treatment:

- *File / Publish...* (NatStar)
- *File / Configure...*
- *File / Export*

### ***Using the keyboard with a list box***

- [Up] selects the previous line.
- [Down] selects the next line.
- [Home] selects the first line in the list.
- [End] selects the last line in the list.
- [PageUp] scrolls up the list by as many lines as the list box's height and selects the first line displayed.
- [PageDown] scrolls down the list by as many lines as the list box's height and selects the last line displayed.

The mouse is also handled automatically: a single click selects a line, and the scroll bar can be manipulated as described in the section on Horizontal and Vertical scroll bars in this manual.

### ***Load a file into a list***

It is often useful to fill a list control when the user opens the window containing it. To do this, a text file must already have been created with an editor or Graphical Builder using the *Edit* button in the list control's properties pane.

1. Open the list control's properties pane.

- Main	
Name	LB_CONTRAT
Description	
Tooltip	
Input	
Tab to	
Init. content	CONTRAT
Preload	<input checked="" type="checkbox"/>
Tabulations	
Separators	
- Events	
INIT	
HELP	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	
- Layout	
X	203
Y	38
Width	158
Height	191
No adjust height	<input type="checkbox"/>
Horz scroll-bar	<input type="checkbox"/>
Anchor	
- Behavior	
Disabled	<input type="checkbox"/>
Mult. sel	<input type="checkbox"/>
Real-time scroll	<input type="checkbox"/>
- Deprecated	
Output	
- Look	
Shadow	Dark (sunken)
Font	MS Sans Serif, 8
Name	MS Sans Serif
Size	8
Bold	<input checked="" type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>

2. Specify the name of the list file in the properties pane *Init* field.

3. The changes are immediate.

This file will be automatically displayed when you test the window (but not during the design phase).



List Box, Combo Box, CBE controls can not manage more than 32766 lines or an index with more than 32766 lines.

#### ***Initialization file***

The associated file resource must have the .TXT extension and be defined in the library.

This file is stored in the directory <GLOB>\DICT\TXT.

At run time, this file must be located in the directory specified by the NS- TXT environment parameter. When you install NatStar (or NS-DK), this environment parameter is automatically set to the directory <GLOB>\DICT\TXT.

#### ***Preselect a line***

It can also be useful to provide the user with a default choice by selecting a line from the outset. To do this, specify the line's index number in the *Input* field: **0** to select the first line, **1** for the second line, etc.

#### ***Format a file***

Each new line in the list file (CR+LF) also generates a new line in the list control. The **space** and **tab** characters also affect the list's formatting (list boxes only). A simple way to align line portions is to use the *Tabulations* field to set the position of each line portion in pixels.



The effect of any tab intervals set will vary according to the selected font (*Fonts* field in a list box's properties pane).

The *Tabulations* field sets the spacing between each tabulation in pixels. This provides a simple way of formatting the lines in a list by vertically aligning line portions.

For example, if you specify:

```
10 , 50 , 70
```

each line will start at 10 pixels from the left-hand border. Then, any characters after the first separator will be tabbed to 50 pixels from the first tabulation (i.e. 60 pixels from the left-hand border), and any characters after the second separator will be tabbed to 70 pixels from the second tabulation (i.e. 130 pixels from the left border). In other words, the list box will contain three columns with a width of 10, 50 and 70 pixels.



Spacing is expressed in pixels to ensure independence from the font used. This value must be modified if you change fonts and find that your original spacing isn't wide enough to accommodate words in the new font.

You can also tabulate without using the *Tabulations* field by using the "System Monospaced" font (for list boxes only): this will ensure that all characters have the same width. This way, all you need to do is enter the appropriate number of spaces between each line portion that you want to align in the list file.



If the *Tabulations* field is empty, the separator indicated in the *Separators* field is not kept.

The *Separators* field allows you to modify the tabulation characters used.

For example, if you enter:

```
' - ' , ' . ' , #09
```

the hyphen, the period and Tab will act as separators. When found in the text, they will not actually be displayed, but will be used to tabulate the characters that follow them.

Characters can be entered:

- Between apostrophes: ' - '
- With their ASCII code: #09
- As a control character: ^J



If at least one character is entered in the *Separators* field, the space and tab characters will no longer generate tabs unless they too are specified in the *Separators* field.

### **Save the selected line when the window is closed**

Enter a variable name in the *Output* field of the list's properties pane. The specified variable won't be updated until the user closes the window containing the list.



The selected line in a list box and combo box is identified by its line number. The *Output* variable will contain the number of the selected line. For a CBE, the value of the *Output* variable will be the character string in the CBE's field.

### **Prevent list height adjustment**

Check the *No Adjust Pos* option in the list's properties pane. The last line displayed may be truncated horizontally as a result.

---

If this option is not checked, the size may be reduced so that a whole number of lines are visible in the list. This adjustment is made when the list is initially displayed, based on the selected font.



This option is only used for list boxes.



List Box, Combo Box, CBE controls can not manage more than 32766 lines or an index with more than 32766 lines.

### **Other actions concerning lists (in the )**

**To allow several lines in the list to be selected**

Check the *Mult. sel.* option.

**To scroll the list in real time as the slider is moved**

Check the *Real-time scroll* option. If this option isn't selected, the list won't be scrolled until the slider reaches its destination.

**To display a horizontal scroll bar in a list**

Check the *Horz. scroll-bar* option.

### **Set CBE characteristics**

A CBE is an entry field with a combo box.

Main	
Name	COMBBOXE1
Description	
Tooltip	
Input	
Type	Default
Max. length	31
Characters	
Format	
Tab to	
Justification	Left
Init. content	
Preload	<input type="checkbox"/>

Events	
INIT	
HELP	
BUTTONDOWN	
CHANGED	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	

Layout	
X	273
Y	52
Width	80
Anchor	

Behavior	
Upcase	<input type="checkbox"/>
Skip blanks	<input type="checkbox"/>
No blanks	<input type="checkbox"/>
Auto-tab	<input type="checkbox"/>
Condition	None
Hide text	<input type="checkbox"/>
Locked text	<input type="checkbox"/>
Disabled	<input type="checkbox"/>
Auto-scroll	<input checked="" type="checkbox"/>

Deprecated	
Output	

Look	

It therefore combines the attributes of an entry field with those of a combo box.

### ***To display the relief of a CBE under Windows XP***

The Windows XP system disables, by default, the display of the controls relief. To bypass this problem, select the classic Windows theme or set in the NSLIB.INI file the parameter UseLookXP to False.





For more information about the NSLIB.INI file, See the "NSLIB.INI file" manual.

***Set the characteristics of a combo box***

You configure a combo box the same way you configure a list box.



List Box, Combo Box, CBE controls can not manage more than 32766 lines or an index with more than 32766 lines.

The screenshot shows the Visual Basic Properties Window for a control named COMBB0X1. The window is divided into several sections: Main, Events, Layout, Behavior, Deprecated, and Look. The Main section includes properties like Name, Description, Tooltip, Input, Tab to, Init. content, and Preload. The Events section lists various events such as INIT, HELP, BUTTONDOWN, GETFOCUS, LOSEFOCUS, SELECTED, EXECUTED, and CHECK. The Layout section shows X, Y, Width, Height, and Anchor. The Behavior section has a Disabled checkbox. The Deprecated section has an Output property. The Look section includes Shadow, Font (Name, Size, Bold, Italic, Underscore, Strikeout), Foreground color, and Background color.

- Main	
Name	COMBB0X1
Description	
Tooltip	
Input	
Tab to	
Init. content	PROF
Preload	<input checked="" type="checkbox"/>

- Events	
INIT	
HELP	
BUTTONDOWN	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	

- Layout	
X	89
Y	318
Width	149
Height	21
Anchor	

- Behavior	
Disabled	<input type="checkbox"/>

- Deprecated	
Output	test

- Look	
Shadow	Dark (sunken)
- Font	
Name	MS Sans Serif
Size	8
Bold	<input checked="" type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	
Background color	Default

**To fix the height of a Combo Box**

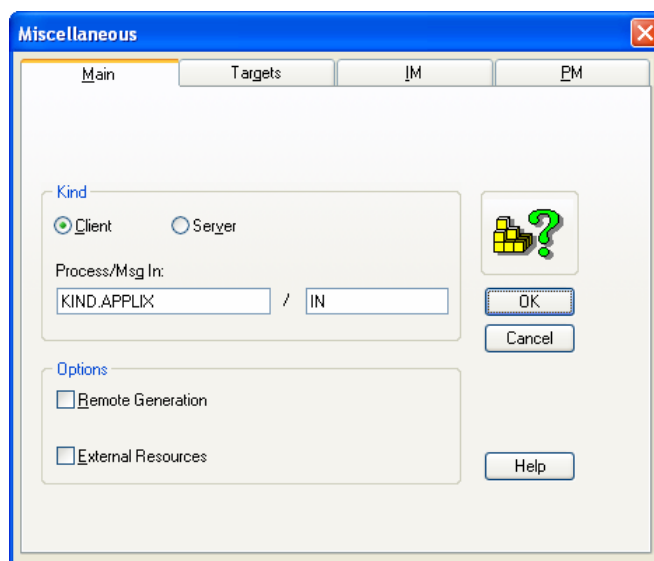
## **.HEIGHT dynamic parameter**

Dynamic parameter HEIGHT is available to specify the height taken by the list displayed by a COMBO BOX or a COMBOX BOX with ENTRY FIELD when you press on the arrow. This height is an integer expressed in pixels. The value which will be really affected (and which will be readable thereafter by consulting the dynamic parameter), will be in fact smaller or equal to the entered number, in order to display whole lines.

## Classifier control

**Classifier controls** allow windows to simulate an organizer with a tab index. Each "page" can display several controls of any type (entry fields, radio buttons, etc.) and is used to group related controls together.

NatStar's Miscellaneous window uses a Classifier control. You open this window from the Select Default Configuration dialog box to set the generation parameters for a particular configuration.



This dialog box displays a single Classifier control consisting of 4 pages with the titles Main, Targets, IM and PM. Each page contains the generation parameters for the category indicated by the page's title.

A Classifier control helps you save space whenever you need to display a large number of controls in the same window. Instead of grouping these controls in group boxes, you can use a Classifier control to categorize them and only display one category of controls at a time. Users can still access any category by selecting the page they want.

In this section, we will:

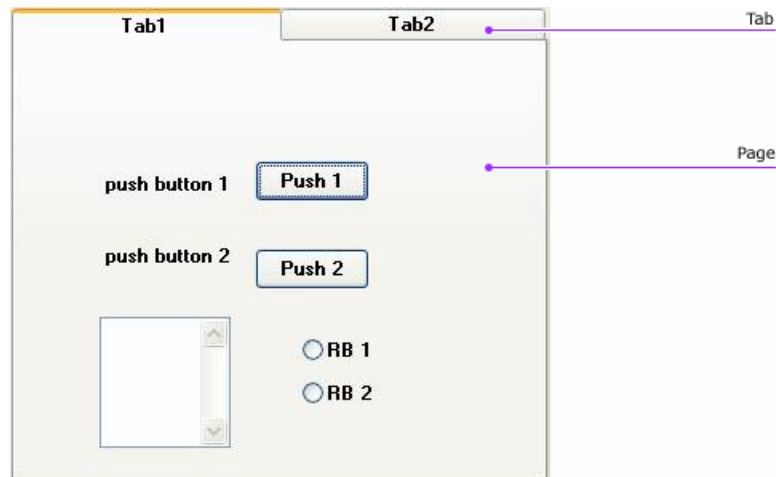
- Describe the components of a Classifier control and how the end-user interacts with it
- Show you how to create a Classifier control and configure it in design mode
- Show you how to program the run-time characteristics of a Classifier control using NCL

## Classifier control components and usage

This section explains the terminology used for the components of a NatStar/NS-DK Classifier control.

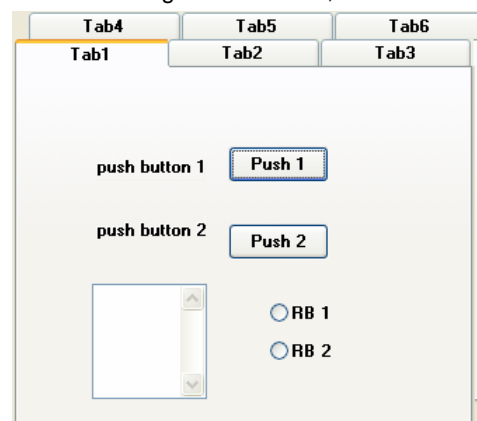
A Classifier control is an area enclosed in a **border**. This area contains a minimum of two tabbed windows and shouldn't contain more than 10. Each tabbed window consists of two parts:

- The **tab**, which contains the tabbed window's **title**
- The **page**, which contains the tabbed window's controls



You can arrange tabs in one or more **rows**.

The following image illustrates a Classifier control with six tabbed windows. The six tabs are arranged in two rows, each containing three tabs.



A classifier control can not contain more than 31 tabs.

### ***Describing pages***

You don't describe the contents of your pages in the Classifier control itself; they are treated as separate windows. When you define the Classifier control, you associate a window with each page.

To describe the components of the pages in your Classifier control, you need to:

- Create a Dialog window for each page
- Edit this window and insert controls into it
- Enter the name of the corresponding window for each page when you define the Classifier control

You can do this in one of the following orders:

- ◆ Create and describe each window first, then associate them with the pages in the Classifier control.
- ◆ Or alternatively, describe the Classifier control, associate a new window with each page, then create and describe all these windows.



To ensure that the Classifier control operates in test mode and at run-time, all the windows associated with your pages must exist.

### ***Using a Classifier control at run-time***

To display a tabbed window, the end-user simply selects its tab in one of the following ways:

- ◆ By clicking the tab for the required page
- ◆ By pressing [Alt] + [mnemonic], if the tab's title contains a mnemonic character (underlined letter)

This displays the selected tabbed window in the foreground, allowing users to see and manipulate the controls on the page.

## **Creating a Classifier control and defining its settings**

### ***Create a Classifier control***

You create a Classifier control in the same way as any other control, by dragging an icon from the tool bar and dropping it into your window. In this case, you need to use

the **Classifier** button  in the tool bar.



If this button does not appear in your toolbox, insert the TABS template in your project.

***Define a Classifier control's settings***

To specify the number of tabbed windows in a Classifier control, set the number of rows, etc.:

1. Open the properties pane for the Classifier control.

CONTROL1

- Main

Name CONTROL1

Description

Tab to

- Events

INIT

TERMINATE

HELP

SELECTED

CHECK

- Layout

X 129

Y 100

Width 359

Height 316

Anchor None

- Behavior

Close Windows ☐

- TABS Mngt

Width mode Automatic fixed

Width 0

Automatic height ☒

Tab height 0

Bold Active ☐

Row Indentation ☒

nb Rows 2

Tabbed Windows

- Look

Foreground color Default

Background color Default

- Font

Name MS Sans Serif

Size 8

Bold ☐

Italic ☐

Underscore ☐

Strikeout ☐

In this pane:

- Enter the number of the tab rows in the *nb Rows* field.
- Enter the list of the tabbed windows in the *Tabbed windows* field by clicking to the *Press to Edit* button, the *Tabbed Windows* box appears allowing to link the tabs to the windows.

Tabbed Windows

Append

Insert

Delete

Home

End

Delete all

Title	Tab
Tab1	TAB1
Tab2	TAB2
Tab3	TAB3
Tab4	TAB4
Tab5	TAB5
Tab6	TAB6

OK

Cancel

Help

- You cannot enter more than 218 characters in that list, otherwise you will get an error message when you try to validate your input by *OK*.
- Choose *OK* to confirm your settings.

In the *Title* column, enter the tab title (in our example, Tab1). In the *Tab* column, enter the name of the window to link (in our example, TAB1).




The *Background* field allows to fix the background color of the Classifier control. It is prevalent towards the background color of the page.



***Describing the contents of a tabbed window***

In design mode, if you drag a control onto one of the tabbed window's pages, the control will be created not on one of the Classifier control's pages, but in an underlying window that contains the Classifier control. You need to describe each page's controls in a separate window.

To describe the contents of a page in a Classifier control:

1. Create a Dialog window whose name matches the one indicated in the *Tabbed windows* group for the corresponding tabbed window.
2. Insert standard controls, templates and custom controls into this window.
3. To save, choose *File \ Save*, or press [Ctrl]+[S], or click the  button .

The section above entitled "Describing pages" shows you the order in which you can describe the Classifier control and the windows associated with its pages.

***Hints***

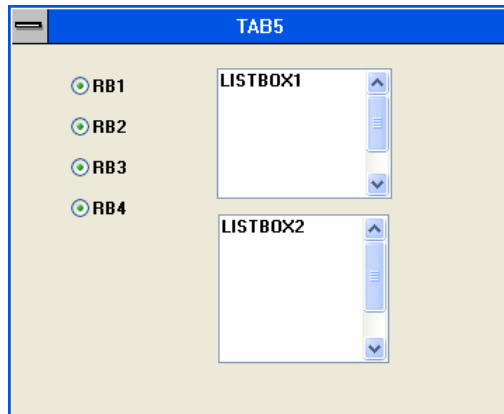
When you associate a window with a page, NatStar/NS-DK doesn't check the window's class, which means that you can associate any type of window. However, only certain window classes function appropriately when you use them in a tabbed window.

We recommend that you follow these guidelines:

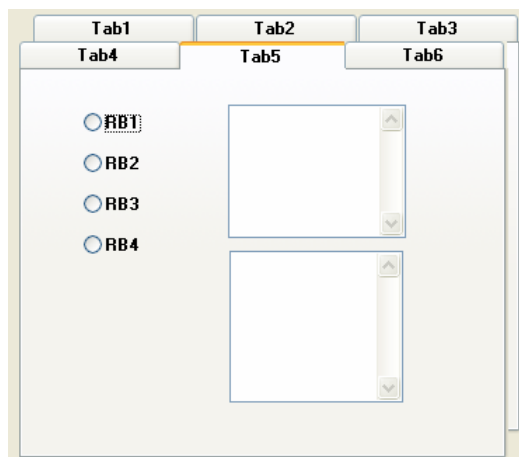
- If the page needs to display various types of controls, use a *Dialog* window. This is by far the most common case.
- If the page only needs to display a multiple line entry field, use an *Edit* window and if it only needs to display a list (which acts like a list box), use a *List* window.
- Don't use any other window classes, as their functionality isn't relevant for Classifier controls.

You don't need to remove the window's title bar since, at run-time, it will be ignored by the page associated with the window.

For example, suppose you define the following window named TAB5:



If you associate this window with the tab entitled "Tab 5" in a Classifier control, you will obtain the following result in test mode or at run-time:



If you define a window that's larger than the page, the latter may only display some of the controls in the window. You should therefore ensure that the window and the page have matching sizes.

### **Configuring tabbed windows' appearance**

Although you can configure the appearance of your tabbed windows manually, we recommend that you accept the default values.

- The *Width* field in the *TABS Mngt* group sets the width of each tab (defaults to zero).

If you accept this value, NatStar / NS-DK will calculate the tab width in proportion to the Classifier control's border width and the maximum number of tabs per row (NatStar / NS-DK calculates this value based on the total number of tabs and rows).

Otherwise, select *User Defined* in the *Width mode* field and enter the required tab width (in pixels).

- The *Tab height* field in the *TABS Mngt* group sets the height of each tab (defaults to zero).

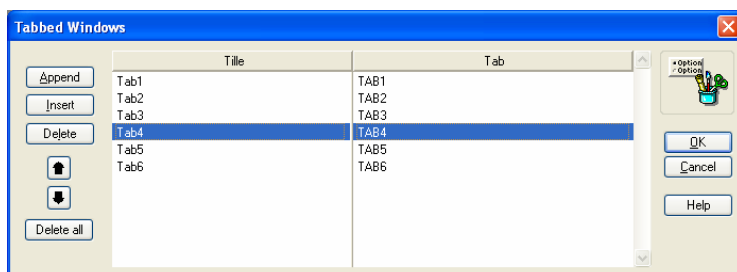
If you accept this value, NatStar / NS-DK will calculate the tab height in proportion to the font height.

Otherwise, untick *Automatic height* and enter the required tab height (in pixels).

### Other actions concerning Classifier controls (using the properties pane)

#### To add a tabbed window

In the *Tabbed Windows* box, add the definition line for the new tabbed window by clicking on the *Append* or *Insert* buttons.




If required, increment the number of rows in the *Nb rows* field by 1.

#### To delete a tabbed window

In the *Tabbed Windows* box, delete the definition line for the tabbed window by clicking on the *Delete* button.

If required, decrease the number of rows in the *Nb rows* field by 1.

#### To modify the tabs position

Select a tab and activate the  button to locate it at the top, and this button  to locate it at the bottom.

#### To ensure that inactive page windows are closed

Check the *Close windows* option. This decreases the number of resources loaded in memory during your application's execution.

If you don't check *Close windows* option, windows of inactive pages that you have opened will stay open.



When *Close windows* option is checked, the management of the inactive pages is not insured. Thus, the modifications of the inactive pages are not saved automatically. The developer must save its modifications by the code.

## Behavior of a Classifier control with the look of Windows XP

The background display color of the Classifier control is disrupted by the Windows XP look:

1. The background color specified in the *Background* field of the properties pane of the Classifier control doesn't affect the tab background color, but only that of the page. With a standard Windows look, the background color is assigned to both the tab and the page.
2. The background color specified for the page is hidden by the background color of the Classifier control.
3. The encoding of a PAINT event for a page without entering a RETURN DEFRET or a PASS at the end of the block of code, blocks application of the background color of the Classifier control for this page.



A simple comment is considered as being code.

## Programming Classifier controls

Each tabbed window in a Classifier control is identified by an index. The index of the first tabbed window is 0, just like the first line in a list. This numbering matches the order defined in the properties pane for the Classifier control.

You manipulate a Classifier control in the same way as a list, which is why some of the verbs applicable to list controls (list boxes, CBEs, etc.) are used in a similar way for Classifier controls.

The SELECTED event for a Classifier control indicates that the user has selected a tab. This event returns the tab number.

The selection of a tab generates a CHECK event with param1% = CHK\_PUSHBUTTON% in the previous tab selected.

### Example

```
INSERT AT END "the Selected Tab is "&&Param12% TO LISTBOX1
```

The following NCL verbs are used to manipulate a Classifier control:

- The LINECOUNT% function
  - The SELECT instruction
  - The SELECTION% function
-

**Retrieve the number of tabs in a Classifier control**

Use the LINECOUNT% function.

```
I% = LINECOUNT%(<Classifier control>)
; I% contains the number of tabs in <Classifier control>
```

**Select a tab in a Classifier control**

Use the SELECT instruction.

```
SELECT <tab nbr> FROM <Classifier control>
; Select the tab whose index is <tab nbr>
; in <Classifier control>
```

**Retrieve the index of the selected tab**

Use the SELECTION% function.

```
I% = SELECTION%(<Classifier control>)
; I% contains the index of the currently selected
; tab in <Classifier control>
```

**Retrieve the window handle associated with a page in a tabbed window**

Use the following syntax: ClassifierControl.[n], where n is the index of the tabbed window. The index begins at 0.

```
H% = <Classifier control>[<tab nbr>]
; H% contains the window handle associated with
; the tabbed window whose index is <tab nbr>
```

**Example**

```
local h%, h1%
local h2%, h3%
;an Internal handle to pass to tab_* functions/instructions it does not
;correspond to a window handle
h% = HECTOR.HANDLE
;Disables the 3d tab
TAB_DISABLE_TAB h%,2,false%

;The Window handle of the dialog box inside the first Tab
h1% = HECTOR[0]
;Modify the text of the pushButton in the first Tab Window
Settext "Submit" to W1(h1%).Button1

;The Window handle of the classifier control
h2% =TAB_GET_CONTAINER%(self%)
;The Window handle of the Window containing the classifier control
h3% =parentwindow%(h2%)
; Set the title of the containig dialog to Classifiers test platform
Settext "Classifiers test platform" to H3%
;Modify the text of the pushButton in the Daialog containing the control
;classifier
Settext "Submit" to VCLASSIF(h3%).Button1
;The Window handle of the Window containing the classifier control here it is
;(self%=h3%)
ENTRY5 = Self%

;N.B.Window Handle means Nat System Window Handle and not Microsoft Window
Handle which can be obtained by getclienthwnd%
```

***To access a control within a window***

Use the following syntax `MyDial(WindowHandle%).ControlName`, where `MyDial` is the name of the window of the type `Dialog`, `WindowHandle%` its window handle and `ControlName` the name of the control within the dialog window.

***To retrieve the classifier control internal handle used in the TAB\_\* functions***

Use the following syntax: `ClassifierControl.Handle`.

```
local h%  
h% = MyClassifier.HANDLE  
TAB_DISABLE_TAB h%, 1, TRUE%  
;N.B. this handle is not a window handle therefore  
;parentwindow%(MyClassifier.HANDLE) would generate a GPF
```

---

## LINECOUNT% function

Returns the number of tabbed windows for a Classifier control.

**Syntax**                      **LINECOUNT%** [(*object*)]

**Comments**

1. For a Classifier control, the *object* parameter is compulsory.

**Example**

```
;Control Classeur CC_TAB contains n tabbed windows, all attached to same window
;DTAB1. This code assigns to the entry field ENTRY of DTAB1 a number
;corresponding to the row of the tabbed window in Classeur control.
Local H% (4)
Local I% (1)

I%=0

FOR I% TO LINECOUNT%(CC_TAB)-1
    H%=CC_TAB[I]
    DTAB1(H%).ENTRY=I%
ENDFOR
```

## SELECT instruction

Selects a tabbed window of a Classifier control.

**Syntax**                **SELECT** *position* **FROM** *objet*

**Comments**

1. The first tabbed window of a Classifier control has an index of 0 (zero).
2. This instruction generates a **SELECTED** event for a tabbed window in a Classifier control.

**Example**

```
SELECT 0 FROM CC_TAB ; Select the first tabbed window of CC_TAB
```



## SELECTION% function

Returns the index of the selected tabbed window in a Classifier control.

**Syntax**                    **SELECTION%** [(*object*)]

**Comment**

1.     The first tabbed window of a Classifier control has an index of 0 (zero).

**Example**

```
Local I%(1)
; Returns the number of the current tabbed window (CC_TAB)
I%=SELECTION%(CC_TAB)
```

### Dynamic parameters

The following parameters allow to modify dynamically the Classifier control.

## .AUTOWIDTH dynamic parameter

### Reading/writing.

Allows to fix dynamically the width of the Classifier control.

This parameter is worth only when the number of tabs rows is strictly superior to 1.

**Syntax**                      **Classifier name.AUTOWIDTH**

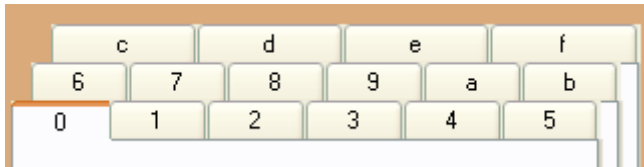
**Possible values**        **TRUE%** : The Classifier control arranges tabs so that every row of tabs has the same width. And thus, every tab can have a different width.

**FALSE%** : The Classifier control arranges all the tabs in the same way. Every row of tabs has not necessarily the same width but every tab has the same width. As a result when the number of tabs is multiple of the number of rows, this parameter has no effect.

### Example

```
TAB1.AUTOWIDTH = TRUE%
```

*Result:*



```
TAB1.AUTOWIDTH = FALSE%
```

*Result:*



## **.BOLD dynamic parameter**

### **Reading/writing.**

Allows to put in bold the tabs titles of the Classifier control.

**Syntax**                      **Classifier name.BOLD**

**Possible values :**     TRUE% : The tabs titles of the Classifier control are written in bold.

                              FALSE% : The tabs titles of the Classifier control are not written in bold.

### **Example**

```
TAB1.BOLD = TRUE%
```

## **.DISABLE\_x\_TAB dynamic parameter**

### **Reading/writing.**

In reading, this parameter allows to know if the number of tab which is indicated by the letter x is activated. There is thus a parameter by tab.

The first tab has for number 1. The parameter of the third tab is called DISABLE\_3\_TAB.

In writing, this parameter deactivates the tab in question.

**Syntax**                      **Classifier name.DISABLE\_x\_TAB**

**Possible values**        TRUE% or FALSE%

### **Example of reading**

```
Est_Onglet_1_desactive% = TAB1.TAB_1_DISABLED
```

### **Example of writing**

```
TAB1.TAB_1_DISABLED = TRUE%
```

## **.HANDLE dynamic parameter**

### **Reading.**

Indicate the handle of the control in the window. The handle is a parameter often used in the functions managing the Classifiers.

**Syntax**                      **Classifier name.HANDLE**

### **Example**

```
i% = TAB_FIND_TAB_TITLE%(TAB1.HANDLE, 'titre 1', TRUE%)
```

## .TAB\_IDENT dynamic parameter

### Reading/writing.

Allows to move or not the rows of tab.

This parameter is worth only when the number of tabs rows is strictly superior to 1.

Permet de décaler ou non les rangées d'onglet.

Ce paramètre n'a de sens que lorsque le nombre de rangées d'onglets est strictement supérieur à 1.

**Syntax**                      **Classifier name.TAB\_IDENT**

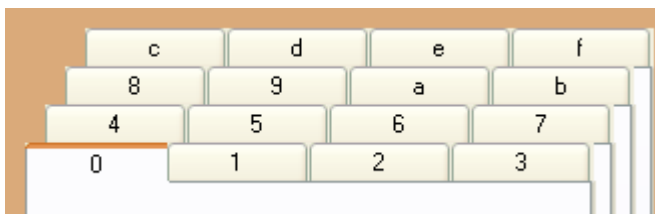
**Possible values**        TRUE% : The Classifier control arranges tabs so that every row of tabs is moved by the precedent of some pixels.

FALSE% : The rows of the control are not moved with regard to the others.

### Example

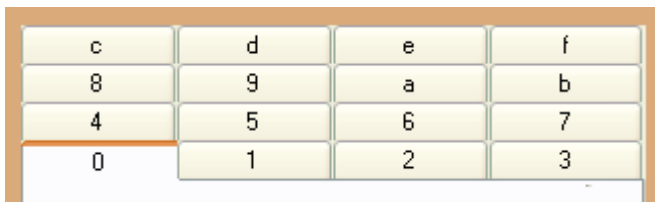
```
TAB1.TAB_IDENT = TRUE%
```

*Result :*



```
TAB1.TAB_IDENT = FALSE%
```

*Result :*



## **Functions and instructions**

The following functions and instructions are taken from the NSCUST library and are declared in the NSCUST.NCL text file.

## TAB\_ADD\_TAB% function

Adds a new tab to the Classifier control

**Syntax**                    **TAB\_ADD\_TAB%** (*hdl*, *title\$*, *dlg\$*, *before\_id%*)

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>title\$</i>	CSTRING(255)	I	Tab title
	<i>dlg\$</i>	CSTRING	I	Name of the window associated with the tab
	<i>before_id%</i>	INT	I	Tab index precedes the new tab that you want to insert.

**Return value**            INT  
0 in a case of failure and the index of the new tab if successful

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.  
*Example*  
`<ControlName>.Handle`
2. The first tab has an index of 1.
3. If *before\_id%* is less than or equal to 0, the new tab is inserted as the last tab.
4. If *before\_id%* is greater than 0, the new tab is placed before the specified tab.
5. If the *dlg\$* (name of the window) parameter is left empty, an empty tab is created. It is therefore necessary to specify a valid index number, otherwise the function returns a 0 and no creation is carried out.
6. A tab never changes its index, but an index may be reassigned.

### Example

*A classifier control Tab*

```
i% = tab_add_tab%(TAB.handle, "Title", "D3", 0)
```



## TAB\_DEL\_TAB% function

Removes the specified tab from the Classifier control

**Syntax**                    **TAB\_DEL\_TAB%**    (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>id%</i>	INT	I	Tab index

**Return value**            INT  
TRUE% if success, FALSE% otherwise.

**Comments**

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. This function deletes the tab, but the index may be reassigned. Next time a tab is added in the Classifier control (by means of the TAB\_ADD\_TAB% function), the deleted tab index will be assigned to it.
3. A tab never changes its index.

**Example**

```
i% = tab_del_tab%(TAB.handle, 1)
```

## TAB\_DISABLE\_TAB instruction

Enables or disables the specified tab of the Classifier control.

**Syntax** **TAB\_DISABLE\_TAB** *hdl, id%, disable%*

<b>Parameters</b>	<i>hdl</i>	POINTERI		Classifier control handle
	<i>id%</i>	INT	I	The index of the tab to be disabled
	<i>disable%</i>	INT	I	Enable/disable tab

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

#### Example

```
<ControlName>.Handle
```

2. The first tab has an index of 1.
3. The tab is enabled if the parameter *disable%* is FALSE%, otherwise it is disabled.
4. The title is grayed when the tabbed window is disabled.
5. When the tab is not active it is grayed out. You can click on it all the same, but the content is frozen.
6. To disable the Classifier control, use the DISABLE instruction. If the Classifier control is disabled, you can no longer change tab.
7. Another way to disable a tab in a classifier window is by means of the computed dynamic parameter DISABLE\_x\_TAB. If *id%* is the index of the tab to be disabled, *x* is (*id%* - 1).

### Example

```
local pointer hclass
hclass = ctrl01.handle
; disable the tab of index 2
TAB_DISABLE_TAB hclass,2,false%
```

## TAB\_EXIST\_TAB% function

Tests for the existence of a tab having the index specified in the parameter.

<b>Syntax</b>	<b>TAB_EXIST_TAB%</b>	<i>(hdl, id%)</i>		
<b>Parameters</b>	<i>hdl</i>	POINTERI	Classifier control handle	
	<i>id%</i>	INT	I	Tab index
<b>Return value</b>	INT(1)			
	TRUE% if the tab is enabled, FALSE% otherwise.			
<b>Comments</b>				

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. The first tab has a rank of 1.
3. Indexes are not always consecutive, when TAB\_ADD\_TAB% and TAB\_DEL\_TAB% have been used.
4. This method of browsing the indexes of a Classifier control is slower than the method set out with the TAB\_NEXT\_TAB function. However, it allows easy changes to be made to existing code that does not take into account possible discontinuity of the indexes.
5. A tab never changes its index, but an index may be reassigned.

**Example**

```
LOCAL POINTER TAB, INTEGER NB%, INTEGER ID%, I%
TAB = TAB1.HANDLE

NB% = TAB_LAST_ID%(TAB)
FOR ID% = 1 TO NB%
    IF TAB_EXIST_TAB%(TAB, ID%)
        ; the ID% tab exists, do sthg with it there
    ENDIF
ENDFOR
```

**See also**            TAB\_NEXT\_TAB, TAB\_ADD\_TAB%, TAB\_DEL\_TAB%

## TAB\_FIND\_TAB\_TITLE% function

Finds the tab having the title which contains the specified text.

**Syntax**                    **TAB\_FIND\_TAB\_TITLE%** (*hdl*, *title\$*, *ignorecase%*)

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>title\$</i>	CSTRING	I	Text researched
	<i>ignorecase%</i>	INT	I	Case specification

**Return value**            INTEGER

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

#### Example

```
<ControlName>.Handle
```

2. If *ignorecase%* is TRUE%, no account is taken of the case of the text. Lowercase and uppercase characters are considered as being identical.
3. The text researched is placed between quotes.
4. A #1 character placed at the beginning and/or the end of the text being sought indicates that other characters may be present before and/or after those indicated.

#### Example :

- "Abc" #1 finds any tab the title of which begins with Abc.
  - #1 "DEF" #1 finds any tab the title of which contains DEF.
  - #1 "GHi " finds any tab the title of which ends with GHi.
5. The function returns a 0 if the search fails, otherwise it returns the identifier of the first tab corresponding to the search criteria.

### Example

```
tab_set_text TAB.handle, 1 , 'Onglet 1'  
i% = tab_find_tab_title%(TAB.handle, "Ong"#1, CK_CASE)
```

## TAB\_GET\_CONTAINER% function

Returns the window handle containing the Classifier control.

**Syntax**                    **TAB\_GET\_CONTAINER%** (*self%*)

**Parameter**                *self%*                    POINTER                    I                    Tab window handle

**Return value**             POINTER. Window handle containing the Classifier control.

**Comments**

1. If you use this function on a window not included in a tab, it returns the window handle itself.
2. This function is to be favored in relation to the ParentWindow% function which will no longer be supported.

**Example**

```
; set the title of the Window containing the classifier control  
settext "Get Container works" to TAB_GET_CONTAINER%(SELF%)
```

## TAB\_GET\_TAB\_BACK\_COLOR% function

Returns the background color of the specified tab.

**Syntax**                    **TAB\_GET\_TAB\_BACK\_COLOR%** (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>id%</i>	INT	I	Tab index

**Return value**            INT(1)

**Comments**

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. If *id%* is less than or equal to 0, the function uses the active tab.

## TAB\_GET\_TAB\_TEXT\_COLOR% function

Retrieves the text color of a tab.

**Syntax**                      **TAB\_GET\_TAB\_TEXT\_COLOR%** (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTERI	Classifier control handle
	<i>id%</i>	INT        I	Tab index

**Return value**            INT(1)

**Comments**

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. If *id%* is less than or equal to 0, the function uses the active tab.
3. If TAB\_SET\_TAB\_COLORS was not called BEFORE, the TAB\_GET\_TAB\_TEXT\_COLOR% function returns an error (even if the initial color is a base color).

**Example**

```
tab_set_tab_colors TAB.handle, 1, col_BLACK%, COL_CYAN%  
  
i%= tab_get_tab_text_color%(TAB.handle, 1)
```

## TAB\_GET\_TEXT\$ function

Retrieves a tab title.

**Syntax**                    **TAB\_GET\_TEXT\$** (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTERI	Classifier control handle
	<i>id%</i>	INT        I	Tab index

**Return value**        CSTRING

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. The first tab has an index of 1.
3. If the tab index is invalid, an uninitialized string is returned.

### Example

```
str$ = tab_get_text$(TAB.handle, 1)
```



## TAB\_ISDISABLED\_TAB% function

Tests if the specified tab is enabled in the Classifier control.

**Syntax**                    **TAB\_ISDISABLED\_TAB%** (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTERI	Classifier control handle
	<i>id%</i>	INT        I	Tab index

**Return value**            INT(1)  
TRUE% if the tab is enable, FALSE% otherwise.

**Comments**

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. The first tab has an index of 1.

**Example**

```
;disable the first tab
tab_disable_tab TAB.handle, 1, False%
i% = tab_isdisabled_tab%(TAB.handle, 1)
```

## TAB\_ISSELECTABLE\_TAB% function

Check if the *id%* tab is mouse selectable.

**Syntax**                    **TAB\_ISSELECTABLE\_TAB%** (*hdl*, *id%*)

<b>Parameters</b>	<i>hdl</i>	POINTERI	Classifier control handle
	<i>id%</i>	INT        I	Tab index

**Returned value**    INT(1).

TRUE% if the Tab is selectable, FALSE% otherwise.

### Comment

1. The disabled tab will still be program selectable.

### Example

```
LOCAL POINTER TAB
TAB = TAB1.HANDLE
INSERT AT END "ISselectable ?"&&TAB_ISSELECTABLE_TAB% (TAB, I) to LB1
```

**See also**                    TAB\_SETSELECTABLE\_TAB

---

## TAB\_LAST\_ID% function

Retrieves the highest index of a Classifier Control where the handle has been passed as a parameter

**Syntax**                    **TAB\_LAST\_ID%**    (*hdl*)

**Parameter**                *hdl*                                    POINTER    I            Classifier control handle

**Return value**            INTEGER

**Comment**

1.    The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

**Example**

```
LOCAL POINTER TAB, INTEGER NB%  
TAB = TAB1.HANDLE  
NB% = TAB_LAST_ID%(TAB)
```

## TAB\_NEXT\_TAB% function

Allows browsing of the indexes of a Classifier control and returns 0 at the end of the browsing process. Browsing is carried out by retrieving the next index via a counter passed as a parameter.

**Syntax**                    **TAB\_NEXT\_TAB%** (*hdl*, *I%*)

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>I%</i>	INTEGER	I/O	Counter to be initialized to 0 and passed to the function

**Return value**           Returns the tab index where a tab exists, otherwise 0 (last tab at the end of the browsing process).

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

#### Example

```
<ControlName>.Handle
```

2. This method allows rapid browsing of the indexes of a Classifier control. The method shown as an example of the TAB\_EXIST-TAB% function is less rapid.
3. Indexes are not necessarily in sequence.



4. Do not confuse the counter (the *I%* parameter) and the index returned by this same counter.

### Example

```
LOCAL POINTER TAB, INTEGER NB%, INTEGER ID%, I%
TAB = TAB1.HANDLE
I% = 0 ; modified by calls to TAB_NEXT_TAB%
ID% = TAB_NEXT_TAB%(TAB, I%)
WHILE ID% > 0
    ; use ID% here
    ID% = TAB_NEXT_TAB%(TAB, I%)
ENDWHILE
```

**See also**                    TAB\_EXIST\_TAB%

## TAB\_SETSELECTABLE\_TAB instruction

Allow or disallow mouse selection of the ID% tab

<b>Syntax</b>	<b>TAB_SETSELECTABLE_TAB</b>	<i>hdl, id%, selectable%</i>
---------------	------------------------------	------------------------------

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>id%</i>	INTEGER	I	Tab index
	<i>selectable%</i>	INT(1)	I	TRUE% (enables the selection)/FALSE%(disables it)

### Example

```
LOCAL POINTER TAB,  INTEGER ID%
TAB = TAB1.HANDLE
TAB_SETSELECTABLE_TAB TAB, I, false%
```

**See also** TAB\_ISSELECTABLE\_TAB%

## TAB\_SET\_TAB\_COLORS instruction

Sets text and background colors of a tab.

**Syntax** `TAB_SET_TAB_COLORS hdl, id%, text_col%, back_col%`

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>id%</i>	INT	I	Tab index
	<i>text_col%</i>	INT	I	Text color
	<i>back_col%</i>	INT	I	Tab background color

### Comments

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

#### Example

```
<ControlName>.Handle
```

2. If *id%* is less than or equal to 0, the function uses the active tab.
3. The values of the parameters *text\_col%* and *back\_col%* may correspond to:
  - a constant COL\_\*
  - -1 to specify the default color of the Classifier control
  - -2 so as not to change the existing color.

### Example

```
tab_set_tab_colors TAB.handle, 1, col_BLACK%, COL_CYAN%  
  
i%= tab_get_tab_text_color%(TAB.handle, 1)
```

## TAB\_SET\_TEXT instruction

Changes the title of the specified tab.

**Syntax**                    **TAB\_SET\_TEXT** *hdl, id%, text\$*

<b>Parameters</b>	<i>hdl</i>	POINTER	I	Classifier control handle
	<i>id%</i>	INT	I	Tab index
	<i>text\$</i>	CSTRING	I	Tab title

**Comments**

1. The *hdl* parameter is retrieved by using the dynamic parameter **.Handle**.

*Example*

```
<ControlName>.Handle
```

2. The first tab has an index of 1.
3. Functionally it is possible to have several tabs of the same name, even if this is obviously not advisable.



NatStar 3.00 and NS-DK 3.00 now permit images in the GIF and JPEG formats to be associated to Picture Buttons.



Justification of the text has been improved for Picture Buttons.

**Example**

```
tab_set_text TAB.handle, 1 , 'Tab 1'
```

## Check boxes

A check box is used to define an option. Unlike a radio button, a check box can appear on its own. It can also be part of a group that represents a set of non-exclusive choices. In the latter case, it may be better to place the check boxes in a group box.

For example, if we wanted to add a "Licenses" group with two check boxes, "Car" and "Motorbike". Both permits are obtained separately, and a person may have neither permit, one of these permits, or both of them.

### ***Initialize a check box***

1. Open the check box's properties pane.
-



CK_BATEAU	
<b>- Main</b>	
Name	CK_BATEAU
Description	
Text	Bateau
Tooltip	
Input	0
Tab to	OK
<b>- Events</b>	
INIT	
HELP	
GETFOCUS	
LOSEFOCUS	
SELECTED	
EXECUTED	
CHECK	
<b>- Layout</b>	
X	47
Y	111
Width	62
Auto size	<input type="checkbox"/>
Anchor	
<b>- Behavior</b>	
3 states	<input type="checkbox"/>
Disabled	<input type="checkbox"/>
<b>- Deprecated</b>	
Output	
<b>- Look</b>	
Shadow	Light (raised)
<b>- Font</b>	
Name	MS Sans Serif
Size	8
Bold	<input checked="" type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	
Background color	Default

2. The check box is automatically checked when it's initially displayed. To set its initial state, in the *Input* field enter:

- ◆ 0 to uncheck the box
- ◆ 1 to check it

***Define a 3-state check box***

Most check boxes have 2 states: checked or unchecked. You can also define a three-state check box (checked, unchecked or grayed) by checking *3 States* in its properties pane box. In this case, the *Input* field will have an additional state: 2 for grayed.

This extra state is used when a simple binary choice is too rigid. Grayed often stands for "unknown", "I don't know", "not applicable" or "not specified."

***Adjust the check box's size to the text displayed***

Open the check box's properties pane box, then check the *Auto size* option.

## Scroll bars (horizontal and vertical)

A scroll bar control is composed of several parts:

- An up arrow (VSCROLL) or left arrow (HSCROLL)
- An empty area between the up or left arrow and the slider
- The slider itself
- Another empty area between the slider and the down or right arrow
- A down arrow (VSCROLL) or right arrow (HSCROLL)

When the scroll bar has the focus, the slider (part 3) is surrounded by a faint dotted box. The slider's position in relation to the two arrows determines the **value** of the scroll bar. Unlike other controls, such as list boxes or MLEs, no other automatic actions are performed by a scroll bar apart from moving its slider. An external action can only be associated with it by programming.

Few standard windows use scroll bar controls separately. They tend to be used automatically and combined with other controls (e.g. vertical scroll bar in list boxes). A number of keys are handled automatically.

### ***Use the keyboard with a scroll bar***

- Pressing [Up] or [Left] is equivalent to clicking inside part 1: the slider moves one unit up or left.
- Pressing [PageUp] is equivalent to clicking inside part 2: the slider moves several units up or left.
- Pressing [PageDown] is equivalent to clicking inside part 4: the slider moves several units down or right.
- Pressing [Down] or [Right] is equivalent to clicking inside part 5: the slider moves one unit down or right.
- The slider itself (part 3) can also be moved by dragging it with the mouse.

### ***Define the scrolling interval***

1. Open the scroll bar's properties pane box.

- Main	
Name	VSCROLLBAR1
Description	
Tooltip	
Input	
Minimum Value	0
Maximum Value	100
Tab to	
- Events	
INIT	
HELP	
GETFOCUS	
LOSEFOCUS	
SELECTED	
CHECK	
- Layout	
X	408
Y	405
Height	51
Anchor	
- Behavior	
Disabled	<input type="checkbox"/>
- Deprecated	
Output	

2. In the *Minimum Value* field, enter control's value when the slider is in the minimum position (left-hand side for horizontal sliders; top for vertical sliders).
3. In the *Maximum Value* field, enter control's value when the slider is in the maximum position (right-hand side for horizontal sliders; bottom for vertical sliders).
4. Lastly, in the *Input* field, enter the slider's initial position in the scroll bar.  
For example, a value equal to the one specified in Minimum Value would place it in the minimum position by default.

### Other actions

**To save the position of the slider**

Enter a variable name in the *Output* field of the control's properties pane box.

## Push Button and Picture Button

Two types of buttons are used: **push buttons** (buttons with associated text) and **picture buttons** (buttons with associated text and a bitmap image). Picture buttons are accessible from the **Custom** tool bar.

For visual design reasons, we advise you to place ellipses ('...') immediately after the text in a button: this will allow the user to identify buttons that display a new choice (and, generally speaking, a new window).

Unlike other controls (static texts, radio buttons, check boxes, etc.), the size of a push button is not automatically adjusted as its text is modified. The text is simply centered in the button. If the text is too long to be fully visible, increase its width by dragging its right or left border with the mouse.



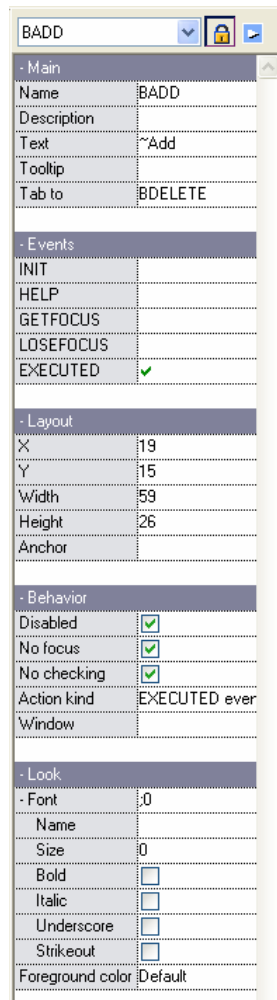
NatStar and NS-DK now permit images in the GIF and JPEG formats to be associated to Picture Buttons.



Justification of the text has been improved for Picture Buttons.



The EXECUTED event of Push Buttons is the default event edited when no code has been entered.

**Validate controls in a window by selecting a button**


Main	
Name	BADD
Description	
Text	~Add
Tooltip	
Tab to	BDELETE
Events	
INIT	
HELP	
GETFOCUS	
LOSEFOCUS	
EXECUTED	<input checked="" type="checkbox"/>
Layout	
X	19
Y	15
Width	59
Height	26
Anchor	
Behavior	
Disabled	<input checked="" type="checkbox"/>
No focus	<input checked="" type="checkbox"/>
No checking	<input checked="" type="checkbox"/>
Action kind	EXECUTED ever
Window	
Look	
Font	0
Name	
Size	0
Bold	<input type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	Default

Pressing a push button triggers checks on the other controls in the same window (e.g. sets the focus on an empty entry field defined as *Condition "Not empty (required)"*), then sends a CHECK event to all the controls.



For more information, see "NCL Language Reference."

To cancel these automatic checks, double-click the push button to display the properties pane, then select *No checking*.



Whatever the status of *No checking*, the button indicated in the *Enter Control* field of the properties pane will always trigger checks. Similarly, whatever the status of *No checking*, the button indicated in the *Escape Control* field of the properties pane will never trigger checks.

### ***Close a window by selecting a button***

If the checks described above were successful, pressing a button allows you to exit the current window unless it's the main window or a modeless child window.

For a main window or modeless window, to ensure that pressing a button exits the window (and the application, in the case of a main window), double-click by the right-hand the push button to display its properties pane, then check *Close on execution*.

For a child modal window, to prevent selection of a button from quitting the window, simply type the comment character ";" in the script associated with the EXECUTED event.

### ***Open a window by selecting a button***

Another way of canceling automatic window closure is to make use of the other function performed by buttons: calling other windows.

To do this, simply enter the name of the window you want to call (window file without the .SCR extension) in the *Window* field of the push button's or picture's button properties pane, and in the *Action kind* field select OPEN to open a modeless window, OPENS a secondary window, CALL a modal window.



As with other options in various controls' properties pane, these options can be emulated by NCL programming. See the CALL and OPEN instructions in the "NCL Language Reference."

- **Modal**

The window called will be modal, which means that the calling window will be locked - no actions will be possible by the user until the called window is closed.

- **Modeless**

The window called will not be modal, which means that the user can continue to interact with the calling window (e.g. select another menu item) while the called window is open. Modeless windows often involve complex processing since the user can perform actions on a number of different levels.

- **Secondary**

A secondary window is a modeless window that is "linked" to the window that called it: each time the parent window is moved, the secondary window moves with it. In addition, when the parent window is clicked, the secondary window does not move into the background but stays visible.

Secondary windows are often used without a title bar and re-sizing borders, and are placed in the parent window's client area. From the end-user's point of view, the secondary window acts just like an area belonging to the parent window and not like a separate window.

### ***Prevent the button from getting the focus when its keyboard accelerator is selected***

Open the menu option's properties pane, then check the *No Focus* option.

### ***Set the image displayed in a picture button***

1. Open the picture button's properties pane.

Look	
Enabled BMP	C:\NSDK\exemp
Disabled BMP	C:\NSDK\exemp
Preload	<input checked="" type="checkbox"/>
button Look	Default
Text position	Above

2. In the *Enabled BMP* field, enter the name of the bitmap resource to be displayed in the picture button when the button is enabled.
3. In the *Disabled BMP* field, enter the name of the bitmap resource to be displayed in the picture button when the button is disabled.
4. Tick *Preload* to load the bitmap at the start of the application.
5. Select in the *Text position* field, the position of the Picture Button text from the bitmap : *Above*, *Below*, *Left*, *Right*.
6. The modifications are automatically take into account.

### ***Set the transparency color values of the Picture button bitmap***

1. Open the properties pane of the Picture Button.

Transparency	
BMP Transparency	Fixed Color
Corner pixel	Default
Transp Color	Default
Font	MS Sans Serif,8
Name	MS Sans Serif
Size	8
Bold	<input type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	Default



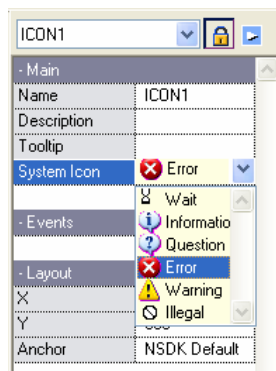
2. Select in the *BMP Transparency* field : *Opaque*, *Corner Pixel* or *Fixed Color*.
3. If you have chose *Corner Pixel*, select in the *Corner Pixel* field, the corner of the pixel bitmap that will be used for the transparency.
  - *Top Left*
  - *Bottom Left*
  - *Top Right*
  - *Bottom Right*
  - *Default* : use the default color defined for the project (almost all the time the top left pixel),
  - *Opaque*
4. The modifications have been taken into account immediately.

## Icons

Icon controls are totally static and cannot be modified by the application's end-user.

### ***To select an icon***

Opens the properties pane of the Picture button, and select the icon in the *System Icon* field.



## Bitmap images

An image is a bitmap file that may have been created with a software tool that produces .BMP files in bitmap format. This type of control is called a **bitmap**.

A bitmap may be static (no actions possible by the end-user) or dynamic (switches between two images when clicked by the user). It can act like an icon (static), check box or push button.



NatStar and NS-DK now permit images in the GIF and JPEG formats to be associated to Picture Buttons. Be sure to precise the extension.



In the BMP resource list of NSDesign tool, you can see only the .BMP images. However, images in GIF or JPEG formats are usable.

### To specify the associated bitmap file

1. Open the bitmap control's properties pane.

2. Enter the name of a bitmap resource in the *Released bitmap* field.

**Create or display the bitmap file**

Create a *Bitmap* resource (NatStar) or *Images* (NS-DK) in the resources browser.

**Load the bitmap file when the application starts up**

Check the *Preload* option in the bitmap control's properties pane.

If this option isn't checked, the bitmap file isn't loaded when the application is started up – it will only be loaded when the user displays the window that contains it.

The size of the bitmap file must not exceed 64 Kb.

**Define a bitmap with push button characteristics**

If there are no actions by the user, the initial *Released bitmap* remains displayed. If the user clicks the bitmap, it changes to *Pressed bitmap* while the mouse button is held down. The bitmap returns to its initial *Released bitmap* state when the mouse button is released. These two bitmaps produce a visual effect equivalent to that of pressing a key on the keyboard.

1. Select *Push Button* in the properties pane.
2. Enter the name of a second bitmap in the field.
3. Uncheck *Disabled* to specify that the second bitmap should only be displayed when the mouse button is pressed. Otherwise, if there are no actions by the user, the first bitmap will be displayed.

To create or display the two bitmap files, simply click the *Edit P* or *Edit R* buttons.

If the *Disabled* option is checked, the bitmap will be disabled. You can then define a third bitmap in the *Disabled* field which will be displayed while the bitmap is disabled.

To create or display the third bitmap file, simply click the **Edit D** button.

**Define a bitmap with check box characteristics**

On the first click, the bitmap changes from *Released bitmap* to *Pressed bitmap*. If the bitmap is clicked again, it returns to its *Released bitamp* state, and so on.

1. Select *Check Box* in the properties pane.
  2. Enter values in the *Released bitmap* and *Pressed bitmap* fields.
  3. Uncheck the *Disabled* option.
-

The displayed bitmap will act like a two-state switch, changing from one state to the other each time the user clicks the mouse.

### ***Adjust the size of a bitmap***

No matter what type has been selected (*icon*, *push button* or *check box*), you must choose between one of two bitmap sizes, depending on the status of the *Adjust size* check box in the properties pane:

- If *Adjust size* is checked (default setting): the displayed bitmap is of the same size as the original bitmap when it was created.

In this case, the control's dimensions (*Width* and *Height*) are ignored. Only the coordinates of the bottom left corner are used (*X* and *Y*).

- If *Adjust size* is unchecked: the displayed bitmap is scaled to fit into the bitmap control.

In this case, the bitmap will be scaled up or down (in both directions) to fit the control's dimensions exactly.

### ***Notes on bitmap files***

If no path has been specified, the bitmap files will be searched for in the directory <GLOB>\DICT\BMP. At run time, the bitmap will be searched for in the directory specified by the NS-BMP environment parameter.

The .BMP extension is added to the file if it was created using the editor and no extension has been specified. The various lists will only display files with the .BMP extension that are part of the bitmap resources. The extension is not shown after the bitmap names.

## Menus

The *Name* field in the properties pane specifies the menu's internal name. It must be unique for each menu and menu option in a given window.

The screenshot shows a properties pane for a menu item named 'MENUITEM1'. The pane is organized into several sections: Main, Events, Layout, Behavior, and Look. The 'Main' section contains fields for Name (MENUITEM1), Description, and Text (Clear). The 'Events' section contains fields for INIT, HELP, SELECTED, and EXECUTED. The 'Layout' section contains a 'Right side' checkbox. The 'Behavior' section contains fields for Disabled, Pulldown menu, Action kind (EXECUTED ever), Window, and Accelerator. The 'Look' section contains a 'Check-mark' checkbox.

MENUITEM1	
- Main	
Name	MENUITEM1
Description	
Text	Clear
- Events	
INIT	
HELP	
SELECTED	
EXECUTED	
- Layout	
Right side	<input type="checkbox"/>
- Behavior	
Disabled	<input type="checkbox"/>
Pulldown menu	<input type="checkbox"/>
Action kind	EXECUTED ever
Window	
Accelerator	
- Look	
Check-mark	<input type="checkbox"/>

Each time a menu or menu option is created, a default internal name is proposed. When you insert a menu item, this name defaults to the name of the selected menu. The name must therefore be modified when you create a new menu item by insertion.

Don't confuse this creation of menu items and the management of menus described in the *User manual*.

### **Associate a keyboard accelerator with a menu option**

Select the key you want from the *Accelerator* field. This key is a shortcut that selects a menu item directly without opening the menu.

No distinction is made between lowercase and uppercase characters.

### **Close a window or open another window by selecting a menu item**

The *Close on Execution* option closes the current window when the menu item is selected. The *Call* field specifies the window to be opened when the menu option is selected.

**Other actions concerning menus (using the properties pane)**

**To display a separator bar after a menu option**    Check the *Separator* option.

**To display a menu item on the right-hand side of the menu bar**    Check the *Right Side* option.

**To check a menu option when the window is initially opened**    Check the *Check Mark* option.

## Multiple line entry fields (MLEs)

An entry field only allows the user to input one line of data, whereas an MLE (Multiple Line Entry Field) allows the user to enter several lines.

The entry field is framed and has scroll bars on the bottom and on the right-hand side which are used to scroll the text. The mouse is handled in the same way as an entry field control or scroll bar, depending on its position.

### ***Use keyboard with an MLE***

- [Home] moves the cursor to the beginning of the line, before the first character.
  - [End] moves the cursor to the end of the line, after the last character.
  - [Shift]+[Home] selects all characters between the cursor and the beginning of the line.
  - [Shift]+[End] selects all characters between the cursor and the end of the line.
  - [BackSpace] deletes the previous character.
  - [Left] moves the cursor to the previous character.
  - [Right] moves the cursor to the next character.
  - [Ctrl]+[Left] moves the cursor to the beginning of the previous word.
  - [Ctrl]+[Right] moves the cursor to the beginning of the next word.
  - [Shift]+[Left] adds the previous character to the selection.
  - [Shift]+[Right] adds the next character to the selection.
  - [Del] deletes all selected characters.
  - [Shift]+[Del] deletes all selected characters and stores them in the clipboard.
  - [Ctrl]+[Ins] stores all selected characters in the clipboard.
  - [Shift]+[Ins] insert the contents of the clipboard at the cursor position.
  - [Up] moves the cursor to the previous line.
  - [Down] moves the cursor to the next line.
  - [Shift]+[Up] adds the previous line to the selection.
  - [Shift]+[Down] adds the next line to the selection.
  - [Ctrl]+[Up] scrolls the text up one line, together with the cursor. The cursor only follows the text if it is positioned on the last visible line.
  - [Ctrl]+[Down] scrolls the text down one line, together with the cursor. The cursor only follows the text if it is positioned on the first visible line.
-



- [PageUp] scrolls the text up by as many lines as the MLE's height. The cursor is repositioned.
- [PageDown] scrolls the text down by as many lines as the MLE's height. The cursor is repositioned.
- [Ctrl]+[Home] moves the cursor to the first line in the text, without changing its horizontal position.
- [Ctrl]+[End] moves the cursor to the last line in the text, without changing its horizontal position.

***Disable input in an MLE***

1. Open its properties pane.

The image shows a configuration window for a Multiple Line Entry Field (MLE1). The window has a title bar with 'MLE1' and standard window controls. The main area is divided into several sections, each with a tab-like header. The 'Main' section contains fields for Name (MLE1), Description, Tooltip, Tab to, and Init. content. The 'Events' section lists INIT, HELP, CHANGED, GETFOCUS, LOSEFOCUS, SELECTED, and CHECK. The 'Layout' section includes X (38), Y (68), Width (156), Height (98), No adjust height (checkbox), and Anchor. The 'Behavior' section includes Uppcase (checkbox), Locked text (checkbox), Disabled (checkbox), Word-wrap (checkbox), Real-time scroll (checkbox), Disable tab stops (checkbox), Buffer Size (16384), Tab stop interval (8), Line (0), and Column (0). The 'Look' section includes Shadow (Dark (sunken)), Font (MS Sans Serif:8), Name (MS Sans Serif), Size (8), Bold (checked), Italic (checkbox), Underscore (checkbox), Strikeout (checkbox), and Foreground color (black). The 'Locked text' checkbox in the Behavior section is selected.

Main	
Name	MLE1
Description	
Tooltip	
Tab to	
Init. content	

Events	
INIT	
HELP	
CHANGED	
GETFOCUS	
LOSEFOCUS	
SELECTED	
CHECK	

Layout	
X	38
Y	68
Width	156
Height	98
No adjust height	<input type="checkbox"/>
Anchor	

Behavior	
Uppcase	<input type="checkbox"/>
Locked text	<input checked="" type="checkbox"/>
Disabled	<input type="checkbox"/>
Word-wrap	<input type="checkbox"/>
Real-time scroll	<input type="checkbox"/>
Disable tab stops	<input type="checkbox"/>
Buffer Size	16384
Tab stop interval	8
Line	0
Column	0

Look	
Shadow	Dark (sunken)
Font	MS Sans Serif:8
Name	MS Sans Serif
Size	8
Bold	<input checked="" type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Foreground color	black

2. Select the *Locked Text* option.
3. The modification is automatic.

### **Format lines in an MLE**

Check the *Word-wrap* option in the MLE's *properties pane*. When the number of characters on a line exceeds *Width*, the text will be formatted so that words aren't truncated. The next word is automatically placed on the next line if it won't fit on the current line.

### **Give the focus to the next field when the user presses the [Tab] key**

Select the *Disable tab stops* option in the control's properties pane.

If this option is not checked (default setting), the [Tab] key moves the cursor inside the MLE to the next tab whose width is specified in *Tab stop interval*.

### **Prevent the MLE's height from being adjusted**

Check the *No Adjust Pos* option in the MLE's properties pane. The last line displayed may be truncated horizontally as a result.

If this option isn't checked, the size may be reduced so that a whole number of lines are visible in the list. This adjustment is made when the MLE is initially displayed, based on the selected font.

### **Display a file in an MLE**

Follow the same steps as for a list.

### **Other actions concerning MLEs (using the properties pane)**

#### **To set the maximum number of characters accepted for input in an MLE**

Enter the number of characters in the *Buffer Size* field. A new line counts as two characters (CR+LF).

The specified size can't exceed 65400.

#### **To define the initial position of the cursor in an MLE**

Enter the position in the *Column* and *Line* fields. *0/0* stands for the first column on the first line.



## Chapter 2

# 32-bit Windows controls



NatStar and NS-DK allow you to import many “Windows Common Controls”.



This document is not exhaustive. It discusses the main aspects of the controls which are implemented in NatStar and NS-DK. For more information about Windows 32-bit controls, see the section on “Platform SDK / User NCL interface Services / Windows Common Controls” in the Microsoft MSDN documentation.

**32  
bits**

Obviously, Windows 32-bit controls are only available for 32-bit Windows platforms (Windows 95/98, NT, ME, 2000, XP).

***This chapter  
shows you how  
to***

- Use the “Windows Common Controls” available with NatStar and NS-DK

## Contents

Introduction.....	2-4
Definition of 32-bit Windows controls	2-4
Displaying 32-bit Windows controls	2-6
Methods for using a 32-bit Windows control	2-9
Dynamic parameters	2-9
Messages	2-9
Message notifications	2-10
Presentation of 32-bit Windows control .....	2-11
Animation	2-11
Creating an Animation control	2-11
Operation in execution	2-12
Date and Time Picker	2-13
Creating a Date and Time Picker control	2-13
HotKey	2-14
Creating a Hotkey control	2-14
List View	2-15
Creating a List View control	2-17
Month Calendar	2-18
Creating a Month Calendar control	2-18
Progress Bar	2-19
Creating a Progress Bar control	2-19
Track Bar	2-20
Creating a Track Bar control	2-20
Tree View	2-22
Creating a Tree View control	2-22
Static settings for 32-bit Windows controls .....	2-24
Creating a control	2-24
Manipulating 32-bit Windows controls .....	2-68
Introduction	2-68
Installation	2-69
Dynamic parameters	2-70
Generic dynamic parameters	2-71
Specific dynamic parameters	2-71
Using List Views and Tree Views .....	2-90
The List View control	2-90
Stages for creating a List View control	2-90
The APIs for creating a List View control	2-90

Example of creating a List View control	2-105
The Tree View control	2-108
Stages for creating a Tree View control	2-108
The APIs for creating a Tree View control	2-109
Example of creating a Tree View	2-123
Example of use	2-129
Reference of the NSMISC library .....	2-130

---

## Introduction

NatStar and NS-DK allow you to import certain “Windows Common Controls” (Animation, Date and Time Picker, Hot Key, List View, Month Calendar, Progress Bar, Track Bar). These controls, which are supplied by Microsoft, are included and used just like other NatStar (or NS-DK) controls.

They allow you to improve the ergonomics of applications by including calendars, animations, progress bars, etc.



This document is not exhaustive. It discusses the main aspects of the controls which are implemented in NatStar and NS-DK. For more information about Windows 32-bit controls, see the section on “Platform SDK / User NCL interface Services / Windows Common Controls” in the Microsoft MSDN documentation.



Obviously, Windows 32-bit controls are only available for 32-bit Windows platforms (Windows 95/98, NT, ME, 2000).

## Definition of 32-bit Windows controls

32-bit Windows controls allow you to create applications with NatStar using controls which are frequently used in Windows applications.

These interfaces have become standards in the world of computing and are therefore well-known to users.

### Control

A **control** is an object which can be placed in a window within an application. Controls are used to display information and to allow the user to enter data into the window and to manipulate it. A Windows 32-bit control is a control which is specific to the Windows application interface.

The 32-bit Windows controls implemented in NatStar and NS-DK allow you to develop applications with a graphic interface and an operation which is similar to that of the most commonly used applications.



The 32-bit Windows controls included in NatStar and NS-DK are as follows:

- Animation,
- Date and Time Picker,
- Hot Key,
- List View,
- Month Calendar,
- Tree View,
- Progress Bar,
- Track Bar.



The LOCK instruction can not be used with 32-bit Windows controls. Indeed, the management of these controls is directly executed by the OS, and Windows doesn't manage this event for these controls.

They are added to the standard NatStar controls:

- Entry-Field,
  - Push-Button,
  - Static-Text,
  - Group-Box,
  - Horizontal Scroll-Bar,
  - Radio-Button,
  - Check-Box,
  - List-Box,
  - Combo-Box,
  - Combo-Box Entry-Field,
  - Icon,
  - Multiple Line Entry-Field,
  - Bitmap.
-

## Displaying 32-bit Windows controls

Microsoft has two versions of the Win32 controls:

- the previous version that is compatible with the different Windows platforms (including XP).
- the new version only compatible with XP (and all future Windows platforms).

By default, for all programs written under Windows XP, the old version of the Win32 controls will be displayed, unless you include a manifest.



A manifest is an XML document which should be in the same directory as your executable file. This file allows Windows XP to decide which version of the Win32 controls and message boxes to display.

In practical terms, in the absence of a manifest, an application under Windows XP will display the Win32 controls and some message boxes (MESSAGE, ASK2%, ASK3%, MESSAGE%) without the Windows XP look, even if the rest of the application is displayed correctly.

When NatStar or NS-DK generate an .EXE file, they automatically add a file .manifest with the same name as the Exe file (eg: MonPrj.EXE will be accompanied by MonPrj.EXE.manifest) in the same directory. If a manifest (not empty) already exists, it is not changed to allow changes to be kept.

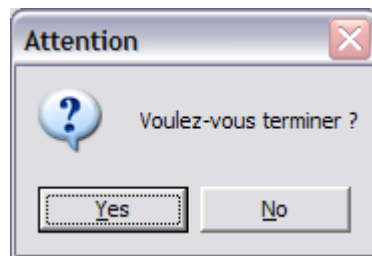


Figure - Message box ASK2% under Windows XP before integration of the manifest.

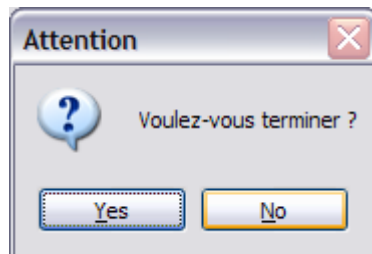


Figure - Message box ASK2% with the integral Windows XP look after integration of the manifest

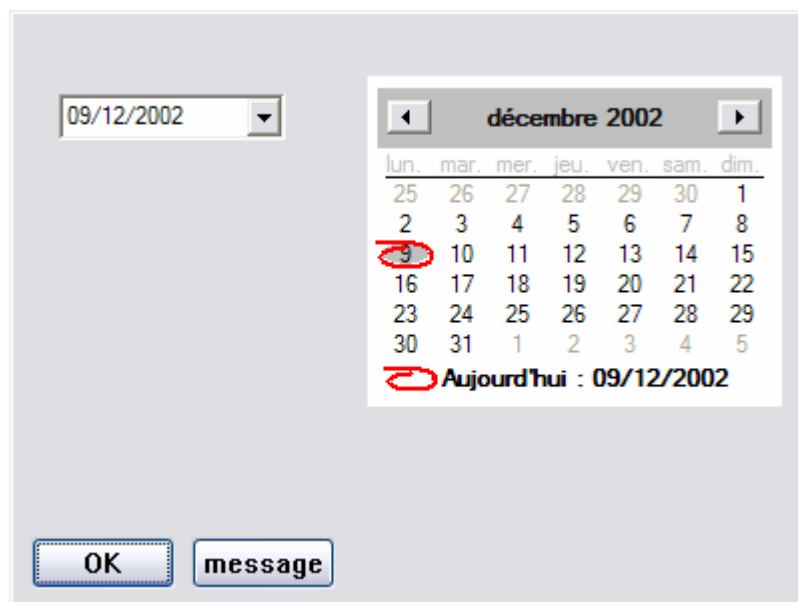


Figure - Windows Win32 Common Controls under Windows XP before integration of the manifest.

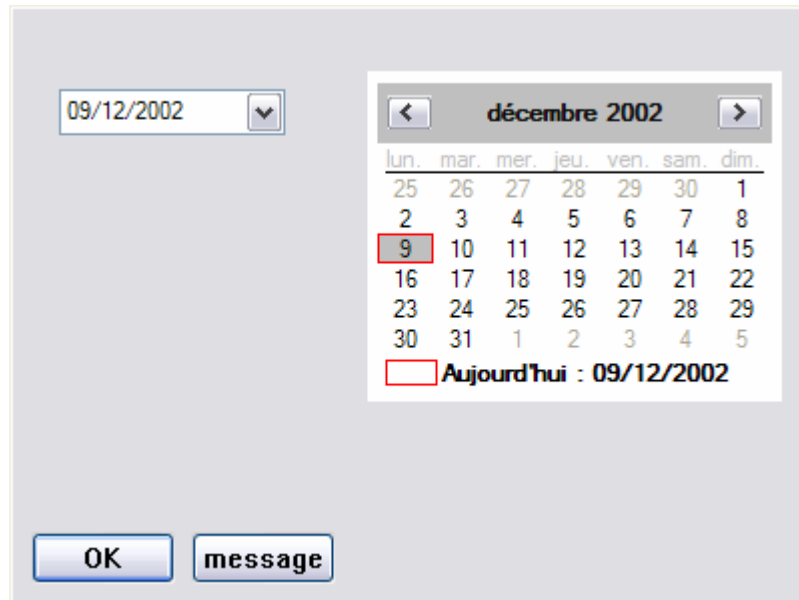


Figure - Windows Win32 Common Controls with the integral Windows XP look after integration of the manifest.

The following environment variables are used to fill in the fields of the manifest:

- NSMANIFESTORG

The name of the organization publishing the application. The default value is YourOrg. The reserved value - (a minus sign on its alone) indicates that a manifest should not be produced.

- NSMANIFESTPROD

Indicates the product name (for example: Office is the product name for Word, Excel etc.). The default value is YourProduct.

- NSMANIFESTDESC

Description of the action. The default value is Your application description here.

#### Example of a manifest

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity
    version="1.0.0.0"
    processorArchitecture="X86"
    name="YourOrg.YourProduct.LOOKXP.EXE"
```

```
    type="win32"
  />
<description>Your application description here.</description>
<dependency>
  <dependentAssembly>
    <assemblyIdentity
      type="win32"
      name="Microsoft.Windows.Common-Controls"
      version="6.0.0.0"
      processorArchitecture="X86"
      publicKeyToken="6595b64144ccf1df"
      language="*"
    />
  </dependentAssembly>
</dependency>
</assembly>
```



Changed or not, the manifest does not influence the look of your application by its content but by its presence.

## Methods for using a 32-bit Windows control

32-bit Windows controls can be manipulated in NatStar according to three different methods: dynamic parameters, messages and message notifications

### Dynamic parameters

Each 32-bit Windows control has a series of dynamic parameters which allow the user to modify the visual appearance and the behaviour of the control.

*For example, the .RANGE parameter allows you to set the minimum and maximum values for a Date and Time Picker control.*



For more information about dynamic parameters, see the section on “Manipulating 32-bit Windows controls / dynamic parameters”.

Certain parameters of the control can also be defined statically in their Windows Common Control properties box.



For more information about the static properties of controls, see the section on “Settings statistic for controls”.

### Messages

A NatStar application can dynamically manipulate 32-bit Windows controls using Windows messages. A specific NCL interface is supplied.

---



For more information about NCL interfaces allowing you to manipulate Windows messages, see the “reference for the NSMISC library” section.

#### Message

A **message** corresponds to an event, used to read or write information. A message may be sent between the system and the NatStar application or between a 32-bit Windows control and the window which contains it.

*For instance, the Animation control's ACM\_OPEN message allows you to open an .AVI file and display it in the control.*

### Message notifications

The NOTIFY event for each NatStar 32-bit Windows control allows you to process notification messages sent by the system when there are changes in status or to give notification of user actions. There are generic notification messages (WM\_NOTIFY) and others specific to each type of control.

All 32-bit Windows controls use the message WM\_NOTIFY to inform the main window of any modifications made following action taken by the user.

Only the Track Bar control uses the messages WM\_HSCROLL and WM\_VSCROLL instead of WM\_NOTIFY.

These messages (WM\_NOTIFY, WM\_HSCROLL, WM\_VSCROLL, etc.) are therefore processed by the NOTIFY event of the 32-bit Windows controls in NatStar.



Neither the events USER0 to USER15, nor any of the other user events can be used with 32-bit Windows controls.

## Presentation of 32-bit Windows control


This section thematically presents the designing of controls.

### Animation

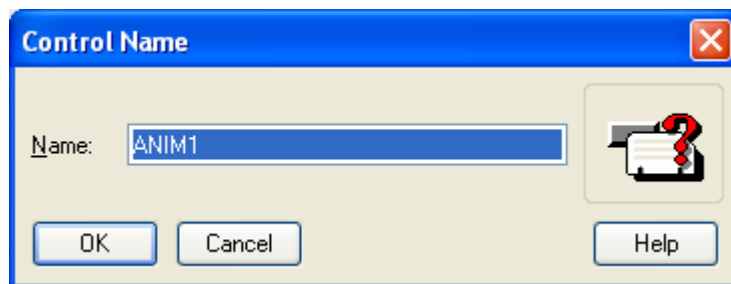
The Animation control allows you to import simple .AVI files (without sound) into your applications. Only .AVI files which do not contain any sounds and which have not been compressed can be imported.

### Creating an Animation control

An *Animation* control is created by:

- ♦ dragging and dropping the *Animation* tool from the toolbox  into the NatStar window.
- ♦ Activating *Controls/Win32/Animation* menu
- ♦ Activating contextual menu by right-click and selecting *Win32/Animation*

A dialog box appears allows you to give a name to the control.



The default text which appears is "ANIM1" if this is the first *Animation* control in the window.



If the default size of the *Animation* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.

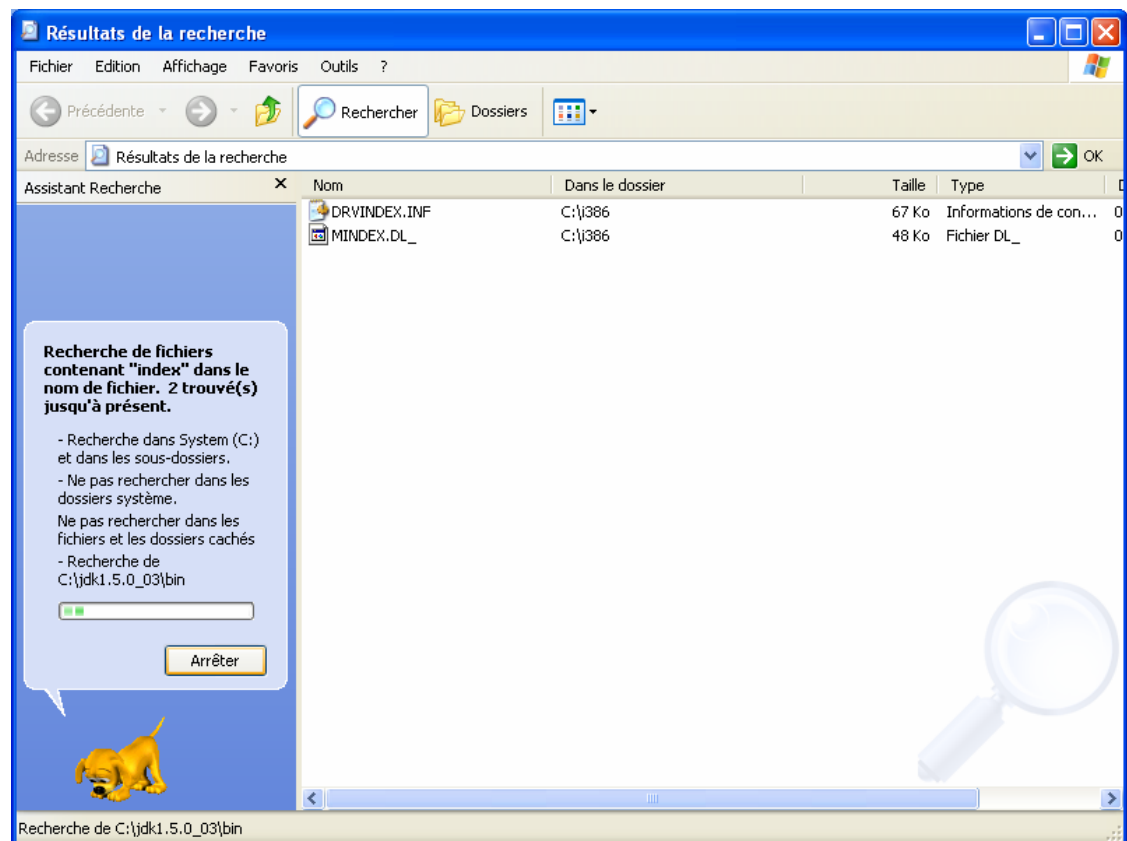


If the *Animation* control is located in a *GroupBox* control, uncheck the Erase Rect box in the *Group Box* properties pane to avoid any display problems.

## Operation in execution

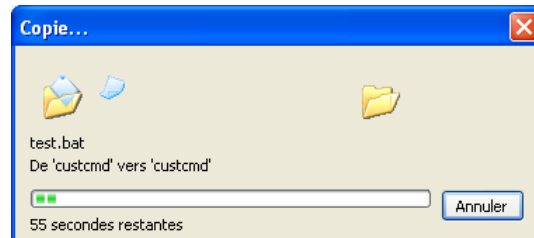
The two following examples show models for the use of an OCX Animation control:

1. The progress bar when searching for files in the Windows Explorer.



2. When copying a file, sheets of paper fly from one folder to the other while the copying is being carried out.






## Date and Time Picker

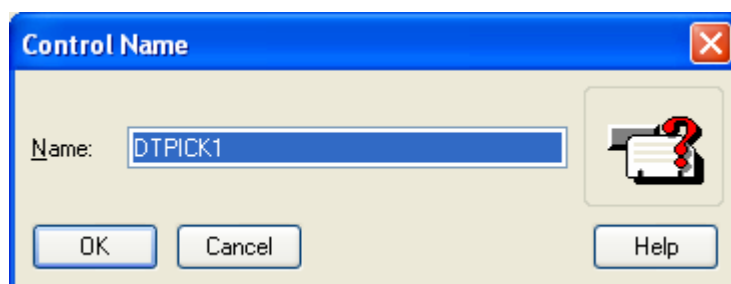
A Date and Time Picker control is a date and time type field allowing dates to be selected. By default, users can select a date from a drop-down calendar which is identical to the Month Calendar control.

### Creating a Date and Time Picker control

A *Date and time Picker* control is created by:

- ♦ dragging and dropping the *Date Time Picker* tool from the toolbox  into the NatStar window.
- ♦ Activating *Controls/Win32/Date Time Picker* menu.
- ♦ Activating contextual menu by right-click and selecting *Win32/Date Time Picker*.

A dialog box appears allows you to give a name to the control.



The default text which appears is "DTPICK1" if this is the first control in the window.



If the default size of the *Date Time Picker* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.




If the *Date Time Picker* control is located in a *GroupBox* control, uncheck the Erase *Rect* box in the *Group Box* properties pane to avoid any display problems.

## HotKey

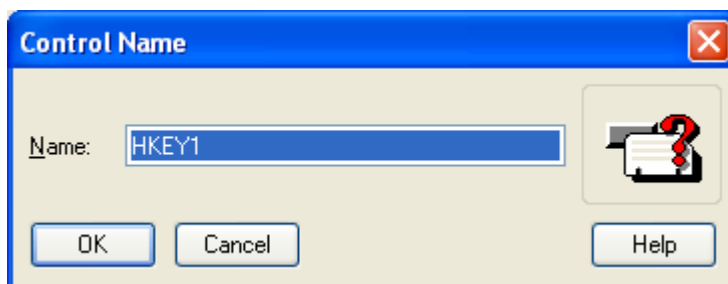
A *HotKey* control allows you to enter a combination of keys which can be used as a keyboard shortcut.

### Creating a Hotkey control

A *HotKey* control is created by:

- ◆ dragging and dropping the *HotKey* tool from the toolbox  into the NatStar window.
- ◆ Activating *Controls/Win32/HotKey* menu.
- ◆ Activating contextual menu by right-click and selecting *Win32/HotKey*.

A dialog box appears allows you to give a name to the control.



The default text which appears is “HKEY1” if this is the first control in the window.



If the default size of the *Hotkey* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.

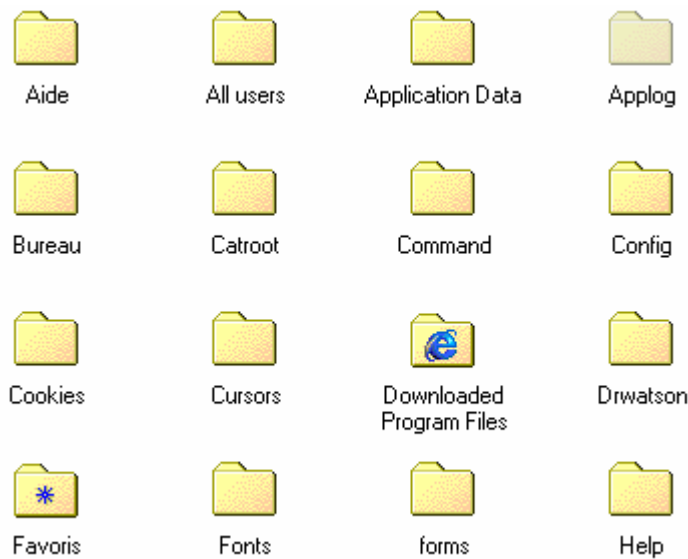


If the *Hotkey* control is located in a *GroupBox* control, uncheck the *Erase Rect* box in the *Group Box* properties pane to avoid any display problems.

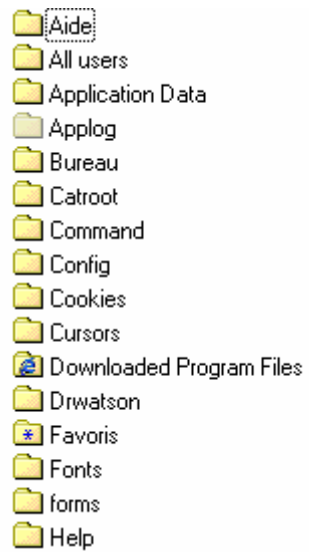
## List View

A List View control is a window which displays a series of elements. Each element has an icon and a label. List View controls allow you to display elements in various ways (small icons, large icons, details, list).

*View with large icons:*



*List view:*




Detailed view:

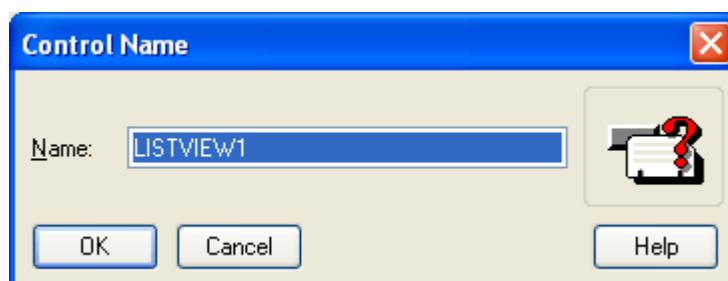
Nom	Type	Modifié
Aide	Dossier	02/11/00 11:49
All users	Dossier	02/11/00 11:30
Application Data	Dossier	02/11/00 11:18
Applog	Dossier	02/11/00 11:22
Bureau	Dossier	02/11/00 11:30
Catroot	Dossier	02/11/00 11:17
Command	Dossier	02/11/00 11:11
Config	Dossier	02/11/00 11:15
Cookies	Dossier	02/11/00 11:34
Cursors	Dossier	02/11/00 11:12
Downloaded Program Files	Dossier ActiveX ...	02/11/00 11:32
Drwatson	Dossier	02/11/00 11:15
* Favoris	Dossier Favori d...	02/11/00 11:32
Fonts	Dossier	02/11/00 11:15
forms	Dossier	02/11/00 11:49
Help	Dossier	02/11/00 11:12

## Creating a List View control

A *List View* control is created by:

- ♦ dragging and dropping the *List View* tool from the toolbox  into the NatStar (NS-DK) window.
- ♦ Activating *Controls/Win32/List View* menu.
- ♦ Activating contextual menu by right-click and selecting *Win32/List View*.

A dialog box appears allows you to give a name to the control.



The default text which appears is “LISTVIEW1” if this is the first control in the window.



If the default size of the *List View* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.




If the *List View* control is located in a *GroupBox* control, uncheck the Erase Rect box in the *Group Box* properties pane to avoid any display problems.

## Month Calendar

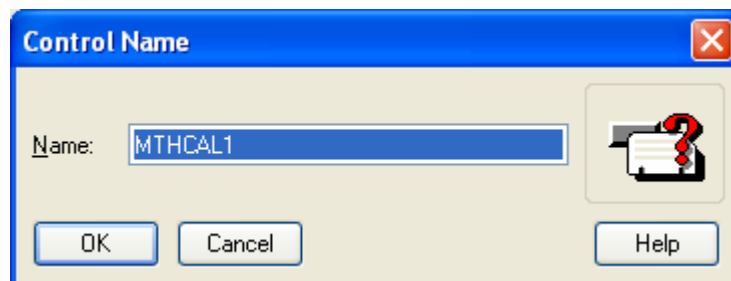
A Month Calendar control implements a calendar type user interface. This interface allows the user to enter or to select a date or a range of dates (from 20/05/2001 to 25/05/2001).

### Creating a Month Calendar control

A *Month Calendar* control is created by:

- ♦ dragging and dropping the *Month Calendar* tool from the toolbox  into the NatStar (or NS-DK) window.
- ♦ Activating *Controls/Win32/Month Calendar* menu.
- ♦ Activating contextual menu by right-click and selecting *Win32/Month Calendar*.

A dialog box appears allows you to give a name to the control.



The default text which appears is “MTHCAL1” if this is the first control in the window.



If the default size of the *Month Calendar* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.



If the *Month Calendar* control is located in a *GroupBox* control, uncheck the *Erase Rect* box in the *Group Box* properties pane to avoid any display problems.


## Progress Bar

The Progress Bar control shows the progress of a long operation.

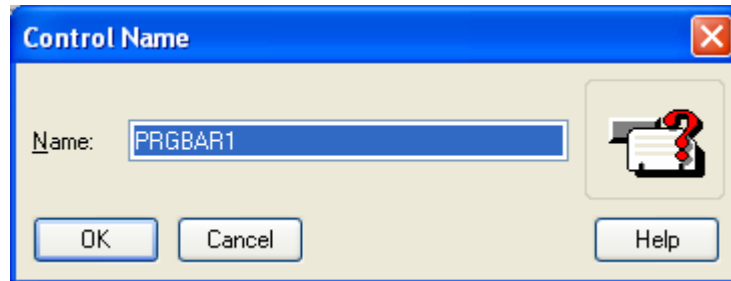


### Creating a Progress Bar control

A *Progress Bar* control is created by:

- ◆ dragging and dropping the *Progress Bar* tool from the toolbox  into the NatStar (or NS-DK) window.
- ◆ Activating *Controls/Win32/Progress Bar* menu.
- ◆ Activating contextual menu by right-click and selecting *Win32/Progress Bar*.

A dialog box appears allows you to give a name to the control.



The default text which appears is “PRGBAR1” if this is the first control in the window.



If the default size of the *Progress Bar* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.



If the *Progress Bar* control is located in a *GroupBox* control, uncheck the *Erase Rect* box in the *Group Box* properties pane to avoid any display problems.


## Track Bar

The Track Bar control allows you to choose a value in an interval using a cursor.



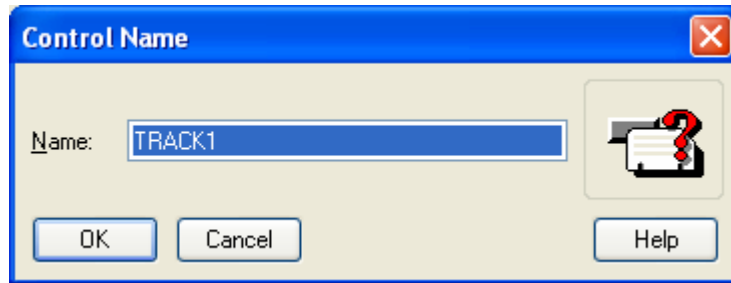
### Creating a Track Bar control

A *Track Bar* control is created by :

- ♦ dragging and dropping the *Track Bar* tool from the toolbox  into the NatStar (NS-DK) window.
- ♦ Activating *Controls/Win32/Track Bar* menu.
- ♦ Activating contextual menu by right-click and selecting *Win32/Track Bar*.

A dialog box appears allows you to give a name to the control.





The default text which appears is “TRACK1” if this is the first control in the window.



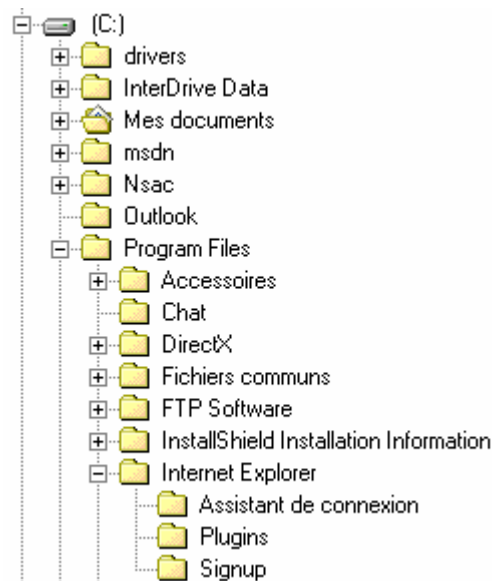
If the default size of the *Track Bar* control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.



If the *Track Bar* control is located in a *GroupBox* control, uncheck the Erase Rect box in the *Group Box* properties pane to avoid any display problems.


## Tree View

A Tree View control displays a tree structure list of elements. Each element is represented by a label and sometimes an associated image. An element may contain sub-elements. By clicking on an element, you can view or not view the sub-elements which it contains.

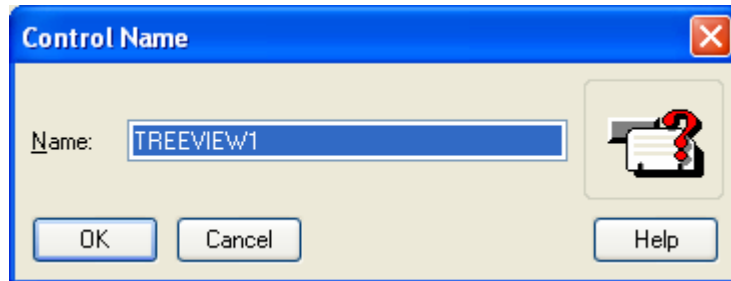


## Creating a Tree View control

A *Tree View* control is created by :

- ◆ dragging and dropping the *Tree View* tool from the toolbox  into the NatStar (NS-DK) window.
- ◆ Activating *Controls/Win32/Tree View* menu.
- ◆ Activating contextual menu by right-click and selecting *Win32/Tree View*.

A dialog box appears allows you to give a name to the control.



The default text which appears is “TREEVIEW1” if this is the first control in the window.



If the default size of the Tree View control is too small to contain the .AVI file, enlarge it by grabbing one of its edges with the mouse.



If the *Tree View* control is located in a *GroupBox* control, uncheck the Erase Rect box in the *Group Box* properties pane to avoid any display problems.

## Static settings for 32-bit Windows controls

### Creating a control

A control is created with the toolbox. Its type depends on the tool selected.




*Fig. The tools*

### Displaying the Windows Common Control Box for a control

The properties pane allows you to modify the standard behaviour of controls.

The properties pane specific to each type of control may be obtained:

- By double-clicking with the right mouse button on the control
- By using the [Return] key when the control is selected
- By using the *Properties* button .

## Animation

### Access in the toolbox



### Display in design phase



## Setting the control's properties

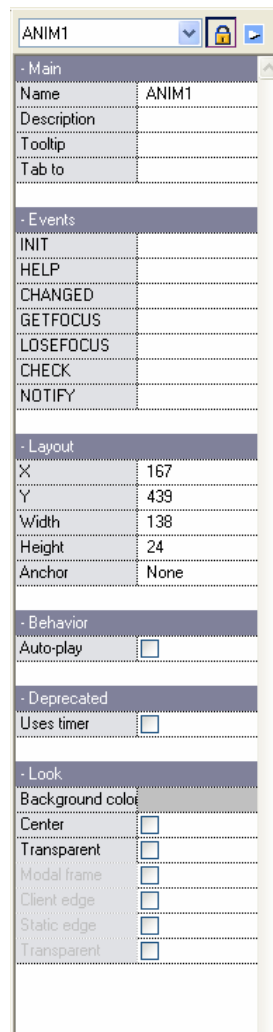


Fig. Properties pane – Animation

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Animation* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *Animation* control, shown in pixels.

**Name**

Name of the *Animation* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Animation* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Anchor**

The anchor allows the *Animation* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Style**

Styles for the control allowing you to modify the control's appearance and behaviour. When a style is selected, it is displayed highlighted and its value passes to TRUE%. Please note that certain styles are exclusive and others are complementary.

**Center**

This style centres the animation in the *Animation* control window. The *Center* style corresponds to the ACS\_CENTER% NatStar constant.

**Transparent**

This style allows you to combine the control's background colour with the main colour of the window. The *Transparent* style corresponds to the ACS\_TRANSPARENT% NatStar constant.

**AutoPlay**

---

The animation is run automatically. The *AutoPlay* style corresponds to the ACS\_AUTOPLAY% NatStar constant.

**Timer**

A timer allowing you to set the number of seconds for which the animation should run. The *Timer* style corresponds to the ACS\_TIMER% NatStar constant.

**ExModalFrame**

A picture of the border of the control in relief.



**ExClientEdge**

A picture of the border of the control in hollow relief.



**ExStaticEdge**

A picture of the border of the control in hollow relief.



**ExTransparent**

Makes the control's background colour transparent.

**Foreground**

Foreground colour of the Animation control.

**Background**

Background colour of the Animation control.

**Events ...**

Displays a code editor to allow you to associate a process with the events.

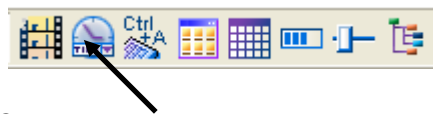


**Font**

Group allowing you to select the name, the size and the format of the police.

## Date and Time Picker

### Access in the toolbox



### Display in design phase



## Setting the control's properties

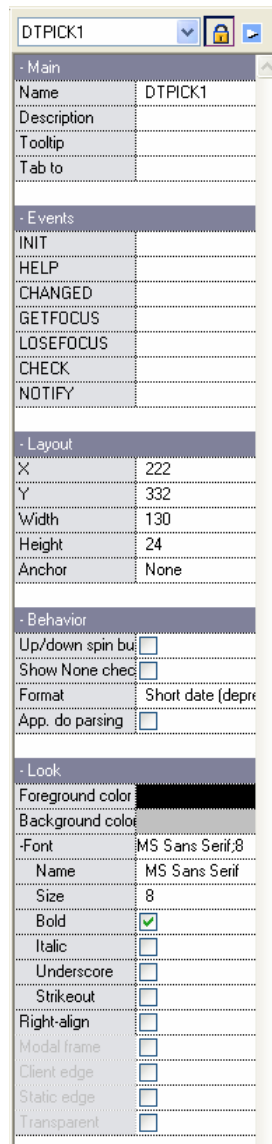


Fig. Properties pane –Date Time Picker

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Date Time Picker* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *Date Time Picker* control, shown in pixels.

**Name**

Name of the *Date Time Picker* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Date Time Picker* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Anchor**

The anchor allows the *Date Time Picker* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tooltip**

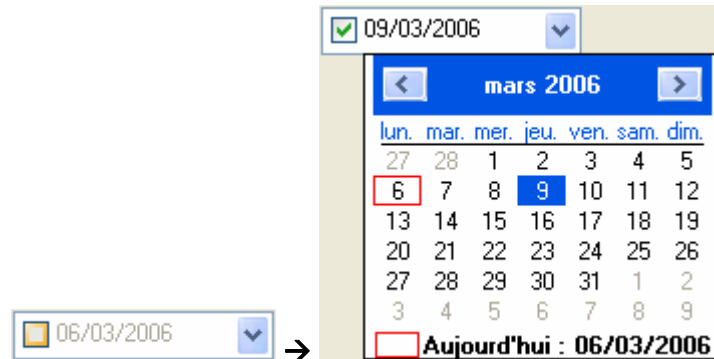
Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Style****Up/Down spin buttons**

Defines an UpDown control represented by a pair of arrows allowing you to remove or add a value to the associated Date control. This style can be used instead of the calendar, which is the default option. The *UpDown* style corresponds to the DTS\_UPDOWN% NatStar constant.

**Show None check-box**



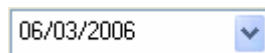
Does not show any date selection in the control. This style displays a checkbox in the control allowing you to confirm a date entered or selected. Unless this box is checked, the application will not be able to find any dates from the control. This style has the DTM\_SETSYSTEMTIME and DTM\_GETSYSTEMTIME messages. It corresponds to the DTS\_SHOWNONE% NatStar constant.



For more information about the DTM\_SETSYSTEMTIME and DTM\_GETSYSTEMTIME messages, see the Microsoft MSDN documentation in the section on “Platform SDK / User Interface Services / Windows Common Controls”.

## Format

### Short Date



Specifies a short date format. (e.g. “26/03/01”) The *Short Date* style corresponds to the DTS\_SHORTDATEFORMAT% NatStar constant.

### Short date with century



Specifies the year in four digits (e.g.: “26/03/2001”)

### Long Date



Specifies a long date format. (e.g. “Tuesday 27 March 2001”)

### Time



Specifies a time format (e.g. “14:53:18”) The *Time* style corresponds to the DTS\_TIMEFORMAT% NatStar constant.

**App. do parsing**

Allows the user to analyse input and to take any necessary action. It authorises users to edit the control’s client area when they press the [F2] key. The *AppCanParse* style corresponds to the DTS\_APPCANPARSE% NatStar constant.

**Look**

**Right-align**

Right alignment (by default, the *Date and Time Picker* control is aligned to the left). The *Right-align* style corresponds to the DTS\_RIGHTALIGN% NatStar constant.

**ExModalFrame**

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control’s background colour transparent.

**Foreground**

Foreground colour of the *Date Time Picker* control.

**Background**

Background colour of the *Date Time Picker* control.

**Events ...**

Displays a code editor to allow you to associate a process with the events.

**Font**

Group allowing you to select the name, the size and the format of the police.

## HotKey

### Access in the toolbox



### Display in design phase



### Operation in execution

The *HotKey* control allows the entry of a key with its name displayed in clear.

## Setting the control's properties

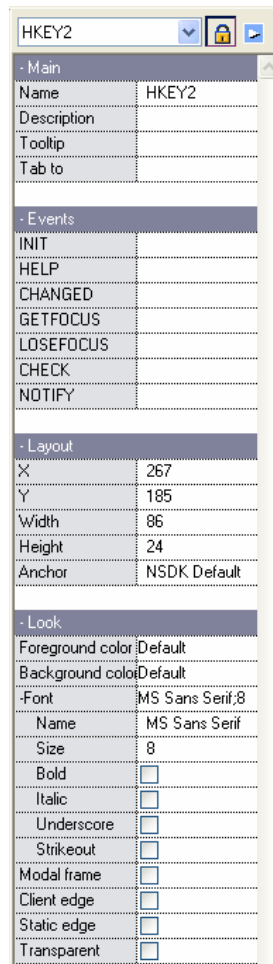


Fig. Properties pane –HotKey

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *HotKey* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

### Width, Height

Width and height of the *HotKey* control, shown in pixels.



**Name**

Name of the *HotKey* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *HotKey* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Anchor**

The anchor allows the *HotKey* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Style****ExModalFrame**

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control's background colour transparent.

**Foreground**

Foreground colour of the *HotKey* control.

---

**Background**

Background colour of the *HotKey* control.

**Events ...**

Displays a code editor to allow you to associate a process with the events.

**Font**

Group allowing you to select the name, the size and the format of the police.

## List View

### Access in the toolbox



### Display in design phase



## Setting the control's properties

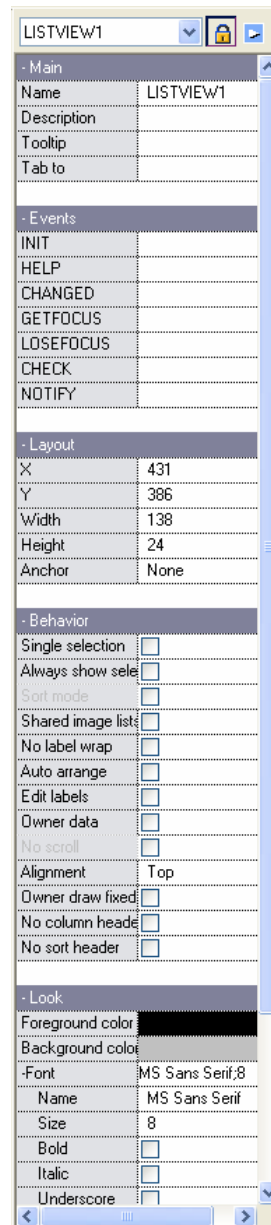


Fig. Properties pane –List View

X, Y

Horizontal and vertical position of the bottom left-hand corner of the *List View* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *List View* control, shown in pixels.

**Name**

Name of the *List View* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *List View* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Anchor**

The anchor allows the *List View* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Look****Single selection**

Restricts the selection to a single element. The *SingleSel* style corresponds to the LVS\_SINGLESEL% NatStar constant.

---

**Always show selection**

The selection is always visible, even if the control no longer has the focus. The *ShowSelAlways* style corresponds to the LVS\_SHOWSELALWAYS% NatStar constant.

**SortAscending**

Sorts the data in ascending order. The *SortAscending* style corresponds to the LVS\_SORTASCENDING% NatStar constant.

**SortDescending**

Sorts the data in descending order. The *SortDescending* style corresponds to the LVS\_SORTDESCENDING% NatStar constant.

**Shared image lists**

The image list is not deleted if the control is deleted. This style allows you to use a single image list for several *List View* controls. This style corresponds to the LVS\_SHAREIMAGELISTS% NatStar constant.

**No label wrap**

This style allows the control to display the text of the element on a single line in icon view. This style corresponds to the LVS\_NOLABELWRAP% NatStar constant.

**Auto arrange**

This style allows the control to keep an icon view. The *AutoArrange* style corresponds to the LVS\_AUTOARRANGE% NatStar constant.

**Edit labels**

This style allows the control to edit the text for the element in place. The *Edit labels* style corresponds to the LVS\_EDITLABELS% NatStar constant.

**Owner data**

This style specifies a virtual List View control. This style allows the control to manipulate millions of elements. The *Owner data* style corresponds to the LVS\_OWNERDATA% NatStar constant.

**No scroll**

This style disables the control's scrollbar. This style is not compatible with the list or detailed views. The *No scroll* style corresponds to the LVS\_NOSCROLL% NatStar constant.

**Alignment****Top**

The elements are aligned to the top of the list in icon view (large or small). The *Top* style corresponds to the LVS\_ALIGNTOP% NatStar constant.

**Left**

The elements are aligned to the left of the list in icon view (large or small). The *Left* style corresponds to the LVS\_ALIGNLEFT% NatStar constant.

**Owner draw fixed**

The owner of the control can draw elements in a detailed view. The *ListView* control sends a WM\_DRAWITEM message to paint each element. It does not send separate messages for each of the sub-elements. The *Owner draw fixed* style corresponds to the LVS\_OWNERDRAWFIXED% NatStar constant.

**No column header**

This style allows you to disable the displaying of column headers in a detailed view. The *No column header* style corresponds to the LVS\_NOCOLUMNHEADER% NatStar constant.

**No sort header**

The column headers are not clickable and do not therefore allow you to sort by column in a detailed view. The *No sort header* style corresponds to the LVS\_NOSORTHEADER% NatStar constant.

**ExModalFrame**

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control's background colour transparent.

**Display mode****Icons**

Displays the data in the form of large icons. The *Icons* style corresponds to the LVS\_ICON% NatStar constant.

**Report**

Displays the data in detailed form. The *Report* style corresponds to the LVS\_REPORT% NatStar constant.

---

**Small Icons**

Displays the data in the form of small icons. The *Small Icons* style corresponds to the LVS\_SMALLICON% NatStar constant.

**List**

Displays the data in the form of a list. The *List* style corresponds to the LVS\_LIST% NatStar constant.

**Foreground**

Foreground colour of the *List View* control.

**Background**

Background colour of the *List View* control.



**Events ...**

Displays a code editor to allow you to associate a process with the events.

**Font**

Opens a *Font Selection* window allowing you to choose the font.

---

## Month Calendar

### Access in the toolbox



### Display in design phase



## Setting the control's properties

The screenshot shows the Properties pane for a control named MTHCAL1. The pane is organized into several sections with expandable/collapsible headers. The 'Main' section is expanded, showing fields for Name, Description, Tooltip, and Tab to. The 'Events' section is also expanded, listing events like INIT, HELP, CHANGED, GETFOCUS, LOSEFOCUS, CHECK, and NOTIFY. The 'Layout' section is expanded, showing coordinates (X, Y) and dimensions (Width, Height) as well as an Anchor. The 'Behavior' section is expanded, showing a 'Multi-select' checkbox. The 'Look' section is expanded, showing color settings (Foreground, Background), font settings (Name, Size, Bold, Italic, Underscore, Strikeout), and various display options (Day state, Show week numt, No Today circle, No Today, Modal frame, Client edge, Static edge, Transparent).

- Main	
Name	MTHCAL1
Description	
Tooltip	
Tab to	
- Events	
INIT	
HELP	
CHANGED	
GETFOCUS	
LOSEFOCUS	
CHECK	
NOTIFY	
- Layout	
X	408
Y	271
Width	180
Height	155
Anchor	None
- Behavior	
Multi-select	<input type="checkbox"/>
- Look	
Foreground color	
Background color	
Font	MS Sans Serif:8
Name	MS Sans Serif
Size	8
Bold	<input type="checkbox"/>
Italic	<input type="checkbox"/>
Underscore	<input type="checkbox"/>
Strikeout	<input type="checkbox"/>
Day state	<input type="checkbox"/>
Show week numt	<input type="checkbox"/>
No Today circle	<input type="checkbox"/>
No Today	<input type="checkbox"/>
Modal frame	<input type="checkbox"/>
Client edge	<input type="checkbox"/>
Static edge	<input type="checkbox"/>
Transparent	<input type="checkbox"/>

Fig. Properties pane –Month Calendar

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Month Calendar* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *Month Calendar* control, shown in pixels.

**Name**

Name of the *Month Calendar* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Month Calendar* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Anchor**

The anchor allows the *Month Calendar* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Behavior****MultiSelect**

This style allows you to select a date interval inside the control. By default, the maximum interval is one week. The *MultiSelect* style corresponds to the MCS\_MULTISELECT% NatStar constant.

**Look****Day state**

This style displays certain dates in bold according to the NOTIFY MCN\_GETDAYSTATE% messages sent by the control. The *Daystate* style corresponds to the MCS\_DAYSTATE% NatStar constant.



### Show week numbers

This style indicates the week number (1-52) to the left of each line. The *WeekNumbers* style corresponds to the MCS\_WEEKNUMBERS% NatStar constant.



### NoTodayCircle

This style deactivates the circle around the day's date. The *NoTodayCircle* style corresponds to the MCS\_NOTODAYCIRCLE% NatStar constant.

### No Today

This style does not display the day's date at the bottom of the control. The *NoToday* style corresponds to the MCS\_NOTODAY% NatStar constant.



### ExModalFrame

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control's background colour transparent.

**Foreground**

Foreground colour of the *Month Calendar* control.

**Background**

Background colour of the *Month Calendar* control.

**Events ...**

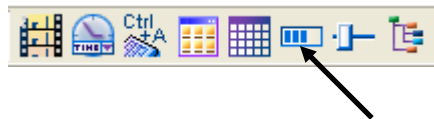
Displays a code editor to allow you to associate a process with the events.

**Font ...**

Group allowing you to select the name, the size and the format of the police.

## Progress Bar

### Access in the toolbox





## Display in design phase



## Setting the control's properties

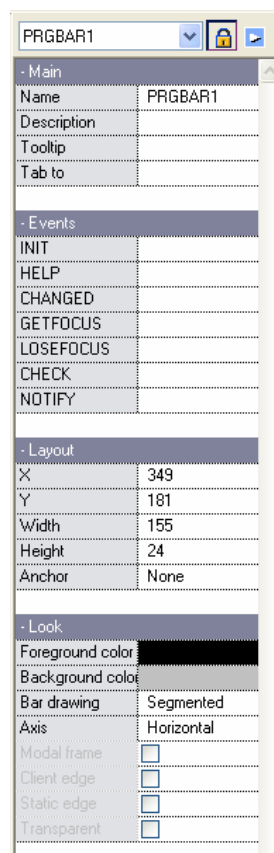


Fig. Properties pane –Progress Bar

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Progress Bar* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *Progress Bar* control, shown in pixels.

**Name**

Name of the *Progress Bar* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Progress Bar* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Anchor**

The anchor allows the List View control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Look**

Styles for the control allowing you to modify the control's appearance and behaviour. When a style is selected, it is displayed highlighted and its value passes to TRUE%. Please note that certain styles are exclusive and others are complementary.

**Axis**

Allows to select the horizontal or vertical axis of the progress bar.

**Vertical**

Displays a vertical progress bar . The *Vertical* style corresponds to the PBS\_VERTICAL% NatStar constant.

**Horizontal**

Displays a horizontal progress bar



### Bar drawing

#### Continuous

Displays a full progress bar



Under Windows XP, the display is not affected by this option.

#### Segmented

Displays a segmented bar



### ExModalFrame

A picture of the border of the control in relief.

### ExClientEdge

A picture of the border of the control in hollow relief.

### ExStaticEdge

A picture of the border of the control in hollow relief.

### ExTransparent

Makes the control's background colour transparent.

### Foreground

Foreground colour of the *Progress Bar* control.

### Background

Background colour of the *Progress Bar* control.

### Events ...

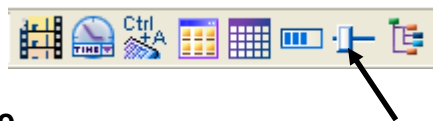
Displays a code editor to allow you to associate a process with the events.

### Font

Group allowing you to select the name, the size and the format of the police.

## Track Bar

### Access in the toolbox



### Display in design phase



### Operation in execution

The *Track Bar* control allows you to choose a value in an interval using a cursor.

## Setting the control's properties

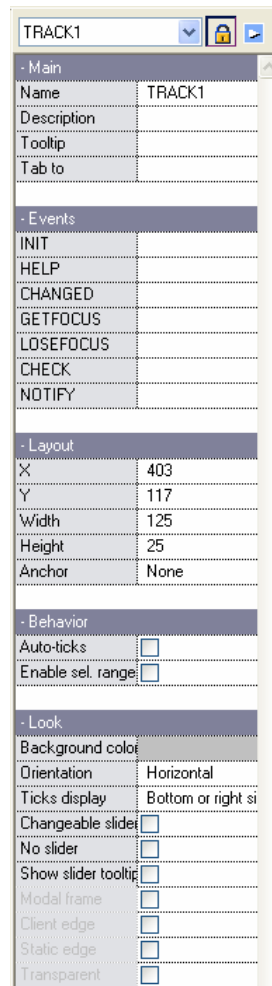


Fig. Properties pane –Track Bar

### X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Track Bar* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

### Width, Height

Width and height of the *Track Bar* control, shown in pixels.

**Anchor**

The anchor allows the *Track Bar* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Name**

Name of the *Track Bar* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Track Bar* control. This comment is only for use by the developer and will not be visible to the user.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Behavior****Auto-ticks**

The control displays a tick for each value. The *AutoTicks* style corresponds to the TBS\_AUTOTICKS% NatStar constant.

**Enable sel.range**

The Track Bar control displays a single selection. The selection is highlighted. The *EnableSelfRange* style corresponds to the TBS\_ENABLESELRANGE% NatStar constant.

**Look****Horizontal**



This style orientates the *Track Bar* control horizontally. This is the default orientation. The *Horizontal* style corresponds to the TBS\_HORZ% NatStar constant.

### Vertical



This style orientates the *Track Bar* control vertically. The *Vertical* style corresponds to the TBS\_VERT% NatStar constant.

### Ticks display

#### Bottom or right side



The ticks are located below (for horizontal style) or to the right (for vertical style) of the cursor.

#### Top or left side



The ticks are located above (for horizontal style) or to the left (for vertical style) of the cursor.

#### Both sides



The ticks are located above and below the cursor.

#### Invisible



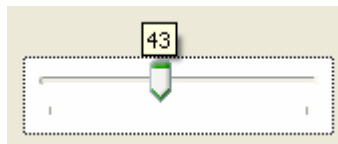
The control does not display any ticks. The *Invisible* style corresponds to the TBS\_NOTICKS% NatStar constant.

**Changeable slider**

The control allows you to change the size of the cursor from the TBM\_SETTHUMBLENGTH message. The *Changeable slider* style corresponds to the TBS\_FIXEDLENGTH% NatStar constant.

**No slider**

The control does not display a cursor. The *No slider* style corresponds to the TBS\_NOTHUMB% NatStar constant.

**Show slider tooltips**

This style automatically creates a tooltip displaying the current position of the cursor. You can change the display position of the tooltips by using the TBM\_SETTIPSIDE message. The *Show slider tooltips* style corresponds to the TBS\_TOOLTIPS% NatStar constant.

**ExModalFrame**

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control's background colour transparent.

**Foreground**

Foreground colour of the *Track Bar* control.

**Background**

Background colour of the *Track Bar* control.



**Events ...**

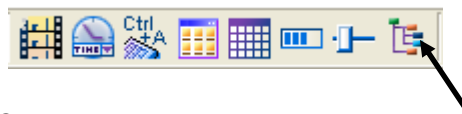
Displays a code editor to allow you to associate a process with the events.

**Font**

Group allowing you to select the name, the size and the format of the police.

## Tree View

### Access in the toolbox



### Display in design phase



### Operation in execution

The *Tree View* control displays data in the form of a tree structure.

## Setting the control's properties

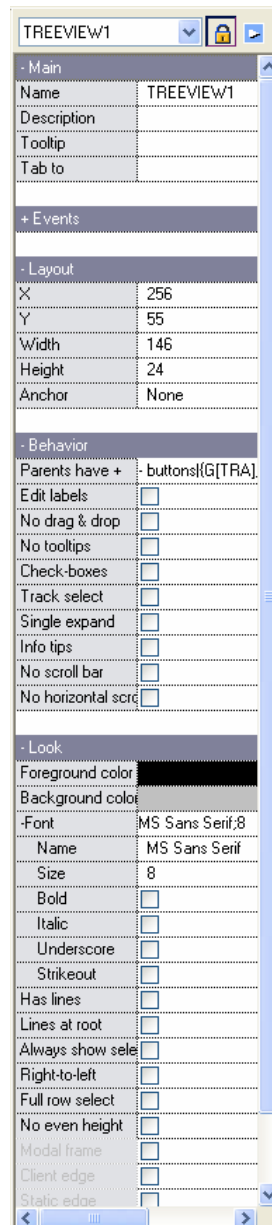


Fig. Properties pane –Tree View

X, Y

Horizontal and vertical position of the bottom left-hand corner of the *Tree View* control.

This position is shown in pixels in relation to the bottom left-hand corner of the window to which the control belongs.

**Width, Height**

Width and height of the *Tree View* control, shown in pixels.

**Name**

Name of the *Tree View* control.

It will not be visible to the user, but will be used internally by the developer to manipulate, modify, query and use the control from the NCL instructions.

**Description**

Comment associated with the *Tree View* control. This comment is only for use by the developer and will not be visible to the user.

**Anchor**

The anchor allows the *Tree View* control, during the resizing of the window, to adapt itself to the new size of the window and take the appropriate position.

**Tab to**

Name of the next control which will take the focus when the user presses the [Tab] key.

**Tooltip**

Contents of the small tooltip which appears when the user leaves the pointing device over the control for a few seconds.

Dynamic configuration and programming using the functions and instructions in the NSTOOLTP library overrides configuration in the properties pane.

**Behavior**

**Parents have +**

Displays (+) and (-) alongside the list's parents elements. They allow you to expand or fold child elements. The *Parents have +* style corresponds to the TVS\_HASBUTTONS% NatStar constant.

**EditLabels**

Editing the properties of the label of the selected element. The *EditLabels* style corresponds to the TVS\_EDITLABELS% NatStar constant.

**No drag & drop**

Disables the drag and drop option. The *No drag & drop* style corresponds to the TVS\_DISABLEDRAHDROP% NatStar constant.

**Track select**

Enables the highlighted displaying of nodes pointed to by the mouse in a *Tree View* control. The *Track select* style corresponds to the TVS\_TRACKSELECT% NatStar constant.

**Single expand**

Leads to the extension of the selected element and folds all the others. The *Single expand* style corresponds to the TVS\_SINGLEEXPAND% NatStar constant.

**Info tips**

Imports the information from the tooltip. The *Info tips* style corresponds to the TVS\_INFOTIP% NatStar constant.

**No scroll bar**

Disables horizontal and vertical scrolling. The control does not display any scrollbars. The *No scroll bar* style corresponds to the TVS\_NOSCROLL% NatStar constant.

**No horizontal scroll bar**

Disables only the horizontal scrolling.

**Look****Has lines**

Uses lines to allow a better view of the hierarchy of the elements. The *Has lines* style corresponds to the TVS\_HASLINES% NatStar constant.

**Lines at root**

Uses vertical lines to connect the elements to the root element of the list. This style is only effective if the *Has lines* style is also specified. The *Lines at root* style corresponds to the TVS\_LINESATROOT% NatStar constant.

**Always show selection**

The selection is always visible, even if the control does not have the focus. The *Always show selection* style corresponds to the TVS\_SHOWSELALWAYS% NatStar constant.

**No tooltips**

---

Tooltips are disabled. The *No tooltips* style corresponds to the TVS\_NOTOOLTIPS% NatStar constant.

**Check-boxes**

Enables checkboxes for *Tree View* control elements. A checkbox is displayed only if an image is associated with the element. The *Check-boxes* style corresponds to the TVS\_CHECKBOXES% NatStar constant.



The *Check-boxes* style cannot be disabled.



For more information about the *Tree View* control *Check-boxes* style, see the Microsoft MSDN documentation in the section on “Platform SDK / User Interface Services / Windows Common Controls”.

**Full row select**

Selects the whole line. Enables the complete selection of the line in a tree structure view. The whole line of the selected element is highlighted whichever element on the line is selected. This style cannot be used with the *Has lines* style. The *Full row select* style corresponds to the TVS\_FULLROWSELECT% NatStar constant.

**No even height**

Sets the height of the elements to an uneven height with the TVM\_SETITEMHEIGHT% message. By default, the height of the elements has an even value. The *No even height* style corresponds to the TVS\_NONEVENHEIGHT% NatStar constant.

**ExModalFrame**

A picture of the border of the control in relief.

**ExClientEdge**

A picture of the border of the control in hollow relief.

**ExStaticEdge**

A picture of the border of the control in hollow relief.

**ExTransparent**

Makes the control's background colour transparent.

**Foreground**

Foreground colour of the *Tree View* control.

**Background**

Background colour of the *Tree View* control.

**Events ...**

Displays a code editor to allow you to associate a process with the events.

**Font**

Group allowing you to select the name, the size and the format of the police.

## Manipulating 32-bit Windows controls

### Introduction

NatStar 32-bit Windows controls allow you to manipulate the main “Windows Common Controls” in an NatStar application according to two methods.

There are two methods available:

- All of the features of NatStar 32-bit Windows controls are accessible thanks to five new functions in the NSMISC library. They allow you to send the messages from the “Windows Common Controls” to the NatStar 32-bit Windows controls.

In addition, the NOTIFY event for each NatStar control allows you to process the notification messages sent by the system when changing status or to give information about user actions. There are generic notification messages and others which are specific to each type of control.

*e.g.*

```
; Code for the notify event for a Win32 control
; MonthCalendar
local a%, m%, j%
if NMHDR(param34%).code = MCN_SELECT%
; the control has received an MCN_SELECT event (selection
; of a new date)
a% = NMSELCHANGE(param34%).stSelStart.wyear
m% = NMSELCHANGE(param34%).stSelStart.wmonth
j% = NMSELCHANGE(param34%).stSelStart.wday
Message "Nouvelle sélection" , " && j% & '/' & m% & '/' & a%
endif
```



For more information about messages or notifications for the “Windows Common Controls”, see the section on “Platform SDK / User Interface Services / Windows Common Controls” in the Microsoft MSDN documentation.

- The main controls (Animation, Date and Time Picker, List View, Month Calendar, Progress Bar, Track Bar and Tree View) can be manipulated by dynamic parameters.



## Installation

1. Declare NSMISC.NCL, NSWCC.NCL and NSCWWAPI.NCL in the libraries on the NatStar *Options / Services* menu.

The NSWCC.NCL file contains among other things the constants and segments corresponding to the messages, notifications and styles for the 32-bit Windows controls.

Windows message constants have their equivalents in NatStar constants. In the same way as Windows structures have their equivalents in NatStar segments.

*e.g.*

The constant for the Windows message: TBM\_GETPOS corresponds to the NatStar constant: TBM\_GETPOS%



For more information about the properties of Windows 32-bit controls, see the section on “Platform SDK / User Interface Services / Windows Common Controls” in the Microsoft MSDN documentation.



For more information about Images Lists, see the section on “Platform SDK / User Interface Services / Windows Common Controls” in the Microsoft MSDN documentation.

The NSMISC.NCL file contains the WCCSEND\* functions allowing you to send Windows messages to 32-bit Windows controls.

2. Check that the NS02MISC.DLL file is in one of the directories of the PATH in Windows (this is done automatically by the NatStar installation program).

## Dynamic parameters

The most frequently used Windows controls are accessible thanks to dynamic parameters.

Dynamic parameters allow you to read or dynamically modify a control's parameters.

The following table summarises the various dynamic parameters used by Windows controls in NatStar.

Dynamic parameter	Animation	Date and Time Picker	List View	Month Calendar	Progress Bar	Track Bar	Tree View
<b>.BACKCOLOR</b>	X	X	X	X	X	X	X
<b>.EXSTYLE</b>	X	X	X	X	X	X	X
<b>.FORECOLOR</b>		X	X	X	X		X
<b>.FORMAT</b>		X					
<b>.NBLINES</b>						X	
<b>.Name of the control [i]</b>		X		X		X	
<b>.RANGE</b>		X		X	X	X	
<b>.STYLE</b>	X	X	X	X	X	X	X
<b>.TEXT</b>		X		X			
<b>.TEXTSEL</b>						X	
<b>.TYPE</b>			X				
<b>.VALUE</b>	X	X		X	X	X	

## Generic dynamic parameters

- For all controls (with the exception of List View and Tree View controls which do not use the .VALUE dynamic parameter), the name of the control is equivalent to the name of the control.value.

*e.g.*

```
wccanim="C:\natstar\f.avi"  
is equal to  
wccanim.value="C:\natstar\f.avi"
```

- The .STYLE and .EXSTYLE dynamic parameters are generic and allow you dynamically to modify the styles and extended styles of all Windows controls included in NatStar. They are manipulated by Boolean operation.

*e.g.*

```
; adds and applies the style  
Control.style BOR (STYLE%)  
; removes and disables the style  
Control.style BAND (BNOT STYLE%)
```

- The .BACKCOLOR and .FORECOLOR dynamic parameters are generic and allow you to set the control's background and foreground colours.

The colour value corresponds either to an integer between 0 and 72 (corresponding to the COL\_BLACK% to COL\_REDEF18% NatStar constants), or to 32-bit integers. These 32-bit integers have a most significant byte at \$80, and the next three bytes numbered from 0 to 255. These last three bytes represent, respectively, the amount of blue, green and red (\$800000FF corresponds to bright red, \$80FF0000 to bright blue, \$80008000 to dark green). The colours are separated from one another by “;”.

## Specific dynamic parameters

The dynamic parameters specific to each control are described in the following sections.

---

## Dynamic parameters for the Animation control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .STYLE parameter allows you to define the control's styles.

The .VALUE parameter allows you to open, stop or run an .AVI file in an *Animation* control.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *Animation* control only supports one background colour.

- **.EXSTYLE**

The control's extended style

- **.STYLE**

The *Animation* control's styles are as follows:

Styles	Read only	Read/Write
ACS_AUTOPLAY%	X	
ACS_CENTER%	X	
ACS_TIMER%	X	
ACS_TRANSPARENT%	X	

They are handled by Boolean operation.

- **.VALUE** (in write mode)

- ◆ Name of an AVI file, allowing you to open an AVI file and to display it from the first frame. This value corresponds to the ACM\_OPEN message.
- ◆ An empty string, allowing you to stop the displaying of an AVI video in the Animation control. This value corresponds to the ACM\_STOP message.
- ◆ A string containing the following optional parameters:

```
: [repetition] [ ; [start_frame] [ ; end_frame] ]
```

This string allows you to identify the sequence's progress repeat number, the launch and end frames for the AVI file.

The default values are repeat = 1; start\_frame= 0 and end\_frame= -1. This value corresponds to the ACM\_PLAY message.

```
wccanim=":5 ;0 ;-1"
```

```
; the wccanim control will run the .AVI file five  
times
```

```
; from the first frame to the last one
```

```
wccanim=" "
```

```
; the wccanim control stops running the animation
```

**Comments:**

- The .VALUE dynamic parameter is in read-only mode.
- The Windows messages are imported into NCL with the % character at the end of the character string.

**e.g.**

```
wccanim="D:\Tests\natstar\f.avi"  
wccanim=":"
```

```
; the wccanim control will run the .AVI file once  
; from the first frame to the last one
```

## Dynamic parameters for the Date and Time Picker control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .FORECOLOR parameter allows you to define the foreground colour.

The .FORMAT parameter allows you to define the date display format.

The .Name of the control[i] parameter allows you to define an range of allowed dates.

The .RANGE parameter allows you to read or write the control's minimum and maximum values.

The .STYLE parameter allows you to define the control's styles.

The .TEXT parameter allows you to read the date in the "date\$, time\$.millisecond" format.

The .VALUE parameter allows you to read or write the current value in the "date\$, time\$.millisecond" format.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *Date and Time Picker* control supports two background colours. The colours are separated from one another by ",". The first colour corresponds to the background colour of the calendar and the second to the selection and heading.

BACKCOLOR = background\_for the\_month;background\_for the\_tray

```
wccdtp.backcolor = col_black% &#59& col_yellow%
```

- **.EXSTYLE**
- **.FORECOLOR**

The *Date and Time Picker* control supports three foreground colours. The colours are separated from one another by “;”.

FORECOLOR = day\_of the\_month\_text;tray\_text;day\_of other\_months\_text

- **.FORMAT (in write mode)**

The date format corresponds to the following syntax. The character strings are optional.

FORMAT = ['character string'] format ['character string']

Value	Description
<b>d</b>	Day shown as one or two digits (e.g. 7, 10)
<b>dd</b>	Day shown as two digits (e.g. 07, 10)
<b>ddd</b>	Abbreviation shown as three characters of the name of the day (e.g. Tue, Wed, Thu, etc.)
<b>dddd</b>	The full name of the day (e.g. Tuesday, Wednesday, etc.)
<b>h</b>	Format in 12-hour mode in one or two digits (e.g. 7, 12)
<b>hh</b>	Format in 12-hour mode in two digits (e.g. 07, 12)
<b>H</b>	Format in 24-hour mode in one or two digits (e.g. 4, 19)
<b>HH</b>	Format in 24-hour mode in two digits (e.g. 17, 22)
<b>m</b>	Minute shown as one or two digits (e.g. 2, 45)
<b>mm</b>	Minute shown as two digits (e.g. 02, 45)
<b>M</b>	Number of the month shown as one or two digits (e.g. 6, 11)
<b>MM</b>	Number of the month shown as two digits (e.g. 06, 11)

<b>MMM</b>	Abbreviation shown as three characters of the name of the month (e.g. Jan, Feb, Mar, etc.)
<b>MMMM</b>	The full name of the month (e.g. January, February, etc.)
<b>s</b>	Second shown as one or two digits (e.g. 2, 45)
<b>ss</b>	Second shown as two digits (e.g. 02, 45)
<b>t</b>	Abbreviation shown as a letter showing AM/PM
<b>tt</b>	AM/PM shown as two letters

**; Format button code**

```
DATETIMEPICKER.format = cb_format
```

**; cb\_format Combo-Box code**

```
insert at end "'Vacances le ' dddd d MMM yyyy' à 'H'h'mm" to \
cb_format
```

```
insert at end "'Formation : ' dddd d MMMM yyyy' à 'H'h'mm" to \
cb_format
```

```
insert at end "'RDV le ' dd'/'MM'/'yyyy' à 'HH'h'mm" to \
cb_format
```

- **.Name of the control[i]** allows you to define a range of dates and times allowed in a *Date and Time Picker* control.

.Name of the control[0]

(read/write): first allowed date and time;last allowed date and time (date%, time%, millisecond)

.Name of the control [1]

(write mode): first allowed date and time;last allowed date and time, formatted by DATE\$ and TIME\$

.Name of the control [2]

(read/write): first allowed date and time (date%, time%.millisecond)

.Name of the control [3]

(read mode): first allowed date and time, formatted by DATE\$ and TIME\$

.Name of the control [4]

(read/write): last allowed date and time (date%, time%.millisecond)



.Name of the control [5]  
(read): last allowed date and time, formatted by DATE\$ and TIME\$

- **.RANGE** (read/write)

Returns or specifies the control's minimum and maximum values. The syntax is as follows: "minimum value; maximum value".

The **.RANGE** parameter corresponds to **.Name of the control[0]**. Nat System recommends the use of the **.RANGE** parameter.

- **.STYLE**

The *Date and Time picker* control's styles are as follows:

Styles	Read only	Read/Write
DTS_APPCANPARSE%	X	
DTS_LONGDATEFORMAT%		X
DTS_RIGHTALIGN%	X	
DTS_SHORTDATEFORMAT%		X
DTS_SHOWNONE%	X	
DTS_TIMEFORMAT%		X
DTS_UPDOWN%	X	

They are handled by Boolean operation.

```
wccdate.style DTS_SHORTDATEFORMAT% BAND DTS_UPDOWN%
```

- **.TEXT** (read only)

Current value in DATE\$ and TIME\$ format (NSDATE library)

- **.VALUE** (read/write)

Current value in DATE% and TIME% format (NSDATE library)

### Example 1: read the current value

```
local buffer$, dat$, heure$, ms$
buffer$ = DATETIMEPICKER
dat$ = copy$ (buffer$, 1, pos%(',' ,buffer$)-1)
heure$ = copy$ (buffer$, pos%(',' ,buffer$)+1, (pos%('.', \
buffer$)) - (pos%(',' ,buffer$)))
ms$ = copy$ (buffer$, pos%('.',buffer$)+1, length (buffer$))
message "valeur courante" , "DATETIMEPICKER = " && \
DATETIMEPICKER && "(" & date$(dat$) & "," & time$(heure$) & "." & ms$
```

**Example 2: read the authorised range**

```
local ret%
local systemtime lsystemtime[2]
; initialise to zero
fill @lsystemtime, sizeof lsystemtime, 0
ret% = WccSendPtr% (int DATETIMEPICKER.Ctrl, DTM_GETRANGE%, \
@lsystemtime)
; Read the min. value
insert at end "Year min : " && lsystemtime[0].wyear
insert at end "Month min : " && lsystemtime[0].wmonth
insert at end "Day min : " && lsystemtime[0].wDay
; Read the max. value
insert at end "Year max : " && lsystemtime[1].wyear
insert at end "Month max : " && lsystemtime[1].wmonth
insert at end "Day max : " && lsystemtime[1].wDay
```

**Example 3: Write the authorised range**

```
local ret%
local systemtime lsystemtime[2]
; initialise to zero
fill @lsystemtime, sizeof lsystemtime, 0
; Set the min. value (1 January 2001 here)
lsystemtime[0].wyear = efa
lsystemtime[0].wmonth = efm
lsystemtime[0].wDay = efj
; Set the max. value (31 December 2001 here)
lsystemtime[1].wyear = efma
lsystemtime[1].wmonth = efmm
lsystemtime[1].wDay = efmj
```

## Dynamic parameters for the List View control

### Description:

The `.BACKCOLOR` parameter allows you to define the background colour.

The `.EXSTYLE` parameter allows you to define the control's extended styles.

The `.FORECOLOR` parameter allows you to define the foreground colour.

The `.STYLE` parameter allows you to define the control's styles.

The `.TYPE` parameter allows you to modify the display as large icons, small icons, list and details.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *List View* control supports two background colours. The colours are separated from one another by “;”.

`BACKCOLOR = background_excluding_text;text_background`

- **.EXSTYLE**

- **.FORECOLOR**

The *List View* control supports only one foreground colour.

- **.STYLE**

The *List View* control's styles are as follows:

Styles	Read only	Read/Write
LVS_ALIGNLEFT%	X	
LVS_ALIGNMASK%	X	
LVS_ALIGNTOP%	X	
LVS_AUTOARRANGE%		X
LVS_EDITLABELS%		X
LVS_ICON%		X
LVS_LIST%		X
LVS_NOCOLUMNHEADER%		X
LVS_NOLABELWRAP%		X
LVS_NOSCROLL%	X	
LVS NOSORTHEADER%	X	
LVS_OWNERDATA%	X	
LVS_OWNERDRAWFIXED%	X	
LVS_REPORT%		X
LVS_SHAREIMAGELISTS%		X
LVS_SHOWSELALWAYS%		X
LVS_SINGLESEL%		X
LVS_SMALLICON%		X
LVS_SORTASCENDING%	X	
LVS_SORTDESCENDING%	X	
LVS_TYPEMASK%	X	
LVS_TYPESTYLEMASK%	X	

They are handled by Boolean operation.

- **.TYPE (read/write)**

Allows you to modify the display as large icons (NatStar LVS\_ICON% constant), small icons (NatStar LVS\_SMALL\_ICON% constant), list (NatStar LVS\_LIST% constant) and details (NatStar LVS\_REPORT% constant).

## Dynamic parameters for the Month Calendar control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .FORECOLOR parameter allows you to define the foreground colour.

The .RANGE parameter allows you to return and specify the upper and lower limits of the control.

The .STYLE parameter allows you to define the control's styles.

The .TEXT parameter allows you to display the date in the DATE\$ format.

The .VALUE parameter allows you to define the current value of the calendar.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *Month Calendar* control supports three background colours. The colours are separated from one another by “;”.

BACKCOLOR = background\_for the \_month;background\_for the \_tray;outside\_background

- **.EXSTYLE**

- **.FORECOLOR**

The *Month Calendar* control supports three foreground colours. The colours are separated from one another by “;”.

FORECOLOR = day\_of the \_month\_text;tray\_text;day\_of other\_months\_text

---

- **Name of the control[i]** allows you to define a range of allowed dates in a *Month Calendar* control.

Name of the control [0]

(read/write): first allowed date;last allowed date (integers in DATE% format)

Name of the control [1]

(read): first allowed date;last allowed date, formatted by DATE\$

Name of the control [2]

(read/write): first allowed date;(integer in DATE% format)

Name of the control [3]

(read): first allowed date, formatted by DATE\$

Name of the control [4]

(read/write): last allowed date;(integer in DATE% format)

Name of the control [5]

(read): last allowed date, formatted by DATE\$

- **.RANGE** (read/write)

Returns or specifies the control's minimum and maximum values. The syntax is as follows: minimum value; maximum value.

The **.RANGE** parameter corresponds to **.Name of the control[0]**. Nat System recommends the use of the **.RANGE** parameter.

- **.STYLE**

The *Month Calendar* control's styles are as follows:

Styles	Read only	Read/Write
MCS_DAYSTATE%	X	
MCS_MULTISELECT%	X	
MCS_NOTODAY%		X
MCS_NOTODAYCIRCLE%		X
MCS_WEEKNUMBERS%		X

They are handled by Boolean operation.

- **.TEXT** (read only)

Current value converted to DATE\$ format (NSDate library)

- **.VALUE** (read/write)  
Current value in DATE\$ format (NSDATE library)

**Examples:**

```
; Use of the .VALUE parameter
local buffer$, debut$, fin$
buffer$ = wccmc
if pos(';', buffer$) <> 0
    debut$ = copy$ (buffer$, 1, pos(';',buffer$)-1)
    fin$ = copy$ (buffer$, pos(';',buffer$)+1, length
(buffer$))
    message ``, "MonthCalendar.value = " && wccmc && "(" & \
date$(debut$) && ";" && date$(fin$) & ")"
else
    message ``, "MonthCalendar.value = " && wccmc && "(" & \
date$(wccmc) & ")"
endif

; Use of the .TEXT parameter
message ``, "MonthCalendar.text = " && wccmc.text

; Use of the Name of the control[0] parameter
message " Affectation Mini/Maxi Autorisés : MonthCalendar[0]=\
date$(Min);date$(Max) (" & date$(efmin) & ";"& date$(efmax) & \
')'

wccmc[0] = date$(efmin) & ";"& date$(efmax)

; Use of the Name of the control[5] parameter
message "Date Maxi Autorisée : MonthCalendar[5]=" && wccmc[5]
```

## Dynamic parameters for the Progress Bar control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .FORECOLOR parameter allows you to define the foreground colour.

The .RANGE parameter allows you to import and specify the upper and lower limits of the control.

The .STYLE parameter allows you to define the control's styles.

The .VALUE parameter allows you to import the current value of the calendar.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *Progress Bar* control only supports one background colour.

- **.EXSTYLE**

- **.FORECOLOR**

The *Progress Bar* control only supports one foreground colour.

- **.RANGE** (read/write)

Returns or specifies the control's minimum and maximum values. The syntax is as follows: "minimum value; maximum value".

- **.STYLE**

The *Progress Bar* control's styles are as follows:



Styles	Read only	Read/Write
PBS_SMOOTH%	X	
PBS_VERTICAL%.	X	

They are handled by Boolean operation.

- **.VALUE** (read/write)  
Current value.

**Examples:**

1. Message « Current value », wccPB
2. for i%=0 to 100  
    wccPB=i%  
    wait 100  
endfor

## Dynamic parameters for the TrackBar control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .NBLINES parameter allows you to import the number of ticks.

The Name of the control[i] parameter allows you to import the position of a tick.

The .STYLE parameter allows you to define the control's styles.

The .TEXTSEL parameter allows you to import or specify the positions of the selected range.

The .VALUE parameter allows you to define the current value.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The colour value corresponds to an integer numbered from 0 to 72 (corresponding to the NatStar COL\_BLACK% to COL\_REDEF18% constants).

- **.EXSTYLE**

- **.NBLINES** (read)

Returns the number of ticks in the control. To be used with the NatStar TBM\_GETNUMTICS% constant corresponding to the TBM\_GETNUMTICS Windows message.

- **.Name of the control[i]** (read)

Allows you to import the position of a tick.

- **.RANGE** (read/write)

Returns or specifies the control's minimum and maximum values. The syntax is as follows: "minimum value; maximum value".

- **.STYLE**

The *Track Bar* control's styles are as follows:

Styles	Read only	Read/Write
TBS_AUTOTICKS%		X
TBS_BOTH%		X
TBS_BOTTOM%		X
TBS_ENABLESELRANGE%		X
TBS_FIXEDLENGTH%		X
TBS_HORZ%		X
TBS_LEFT%		X
TBS_NOTHUMB%		X
TBS_NOTICKS%		X
TBS_RIGHT%		X
TBS_TOOLTIPS%	X	
TBS_TOP%		X
TBS_VERT%		X

They are handled by Boolean operation.

- **.TEXTSEL**

Returns and specifies the start and end position of the current selected range. This parameter needs the *EnableSelfRange* style to be selected to be visible.

- **.VALUE** (read/write)

Current value.

e.g.

```
; WCCTB SELECTED event

; WCCPB = Progress Bar Control

; WCCTB = Track Bar Control

WCCPB=WCCTB

; Assigns the value of the track bar to the progress bar
```

## Dynamic parameters for the Tree View control

### Description:

The .BACKCOLOR parameter allows you to define the background colour.

The .EXSTYLE parameter allows you to define the control's extended styles.

The .FORECOLOR parameter allows you to define the foreground colour.

The .STYLE parameter allows you to define the control's styles.

### Syntax:

**Name of the Dynamic.control parameter**

### Parameters:

- **.BACKCOLOR**

The *Tree View* control only supports one background colour.

- **.EXSTYLE**

- **.FORECOLOR**

The *Tree View* control supports three foreground colours. The colours are separated from one another by “;”.

FORECOLOR=text;insertion\_mark;lines

- **.STYLE**

The *Tree View* control's styles are as follows:

Styles	Read only	Read/Write
TVS_CHECKBOXES%	X	X (only if the checkbox is empty)
TVS_DISABLEDRAHDROP%		X
TVS_EDITLABELS%		X
TVS_FULLROWSELECT%		X
TVS_HASBUTTONS%		X

Styles	Read only	Read/Write
TVS_HASLINES%		X
TVS_INFOTIP%		X
TVS_LINESATROOT%		X
TVS_NONEVENHEIGHT%	X	
TVS_NOSCROLL%	X	
TVS_NOTOOLTIPS%	X	
TVS_SHOWSELALWAYS%		X
TVS_SINGLEEXPAND%		X
TVS_TRACKSELECT%		X

They are handled by Boolean operation.

## Using List Views and Tree Views

This section shows you how to use the *List View* and *Tree View* controls.

### The List View control

The creation of a *List View* control requires various stages which we will summarise in the paragraph on “Stages for creating a List View control”. The paragraph on “Example repository” lists the functions and constants needed for the creation of a *List View*. And finally, the paragraph on “Example showing the creation of a List View control” shows a specific example.

#### Stages for creating a List View control

Creating a *List View* is divided into a number of stages:

1. Creating an Image List,

##### *Image List*

---

An Image List is an image library used to effectively manage a series of bitmaps and icons. All the images contained in an Image List are of the same size. Furthermore, they are included in a single bitmap. An Image List may contain a monochrome bitmap containing masks. These masks are used to draw images transparently (icon mode).

---

2. Associating bitmaps with the Image List,
3. Associating the Image List with the *List View*,
4. Creating columns for detailed presentation,
5. Creating controls (in our example, radio-buttons) allowing you to change view (by list, by icons, etc.)

#### The APIs for creating a List View control

This section lists the main functions and procedures designed to create a *List View* control with NatStar.

## ImageList\_AddMasked function

**Description:**

Adds one or more images to an image list generating a mask for the associated bitmap.

**Syntax:**

**ImageListAddMasked** (*himl*, *hbmImage*, *crMask*)

**Parameters:**

<i>himl</i>	INTEGER	Handle on the image list
<i>hbmImage</i>	INTEGER	Handle on the bitmap containing one or more images. The number of images is deduced from the width of the bitmap.
<i>crMask</i>	INT(4)	Colour used to generate the mask.

**Value returned:**

The index of the first new image, or -1 in case of failure.

## ImageList\_Create function

**Description:**

Creates a new image list

**Syntax:**

**ImageList\_Create**(*cx*, *cy*, *flags*, *cInitial*, *cGrow*)

**Parameters:**

<i>cx</i>	INTEGER	Width, in pixels, of each image
<i>cy</i>	INTEGER	Height, in pixels, of each image
<i>flags</i>	INTEGER	Series of flags specifying the type of image list to be created
<i>cInitial</i>	INTEGER	Number of images initially contained by the image list
<i>cGrow</i>	INTEGER	Number of new integratable images

**Value returned:**

Handle to the image list, or NULL in case of failure.

**Comments:**

The *flags* parameter may contain the following values: ILC\_COLOR24 (24-bit colour), ILC\_COLOR32 (32-bit colour).



## LVCOLUMN segment

### Description:

Contains information about a column from a detailed view. This segment is used both for the creation and for the manipulation of columns in a *List View* control.

### Syntax:

**LVCOLUMN** (*mask*, *fmt*, *cx*, *pszText*, *cchTextMax*, *iSubItem*, *ilImage*, *iOrder*)

### Parameters:

<i>mask</i>	INTEGER	Variable indicating which parameters contain valid information.
<i>fmt</i>	INTEGER	Alignment of the column and text headers and sub-elements.
<i>cx</i>	INTEGER	Width of the column in pixels.
<i>pszText</i> this	POINTER	If the information for the column is fixed, parameter is the address of the character string which is terminated by a NULL value. This character string contains the column header text. If the segment receives information about a column, this parameter specifies the address of the buffer which receives the column header text.
<i>cchTextMax</i>	INTEGER	Size of the buffer pointed to by <i>pszText</i> . If the segment does not receive any information about a column, the <i>cchTextMax</i> parameter is not used.
<i>iSubItem</i>	INTEGER	Index of the sub-element associated with the column
<i>ilImage</i>	INTEGER	Base zero index for an image inside an image list. The specified image appears inside the column.

**Comments:**

1. The *mask* parameter may be equal to zero or to one or more of the following constants: LVCF\_FMT% (the *fmt* parameter is valid), LVCF\_IMAGE% (the *iImage* parameter is valid), LVCF\_SUBITEM% (the *iSubItem* parameter is valid), LVCF\_TEXT% (the *pszText* parameter is valid), LVCF\_WIDTH% (the *cx* parameter is valid).
2. The *fmt* parameter may contain one of the following values: LVCFMT\_LEFT% (the text is aligned to the left) or LVCFMT\_RIGHT% (the text is aligned to the right).

## LVITEM segment

### Description:

Specifies or receives the attributes of an element from the *List View*.

### Syntax:

**LVITEM** (*mask*, *iItem*, *iSubItem*, *state*, *stateMask*, *pszText*, *cchTextMax*, *iImage*, *iParam*, *iIndent*)

### Parameters:

<i>mask</i>	INTEGER	Specifies which members of this segment contain data to be set or are being requested.
<i>iItem</i>	INTEGER	Base zero index for the <i>List View</i> element.
<i>iSubItem</i>	INTEGER	Base one index for sub-element or base zero for a <i>List View</i> element.
<i>state</i>	INTEGER	Indicates the state of the element, the image and the overlaid image.
<i>stateMask</i>	INTEGER	Value specifying which bits of the <i>state</i> parameter can be found or modified.
<i>pszText</i>	INTEGER	If the segment specifies element attributes, <i>pszText</i> is a pointer to a character string terminating with NULL. If the segment receives attributes, <i>pszText</i> is a pointer to a buffer which receives the text from the element.
<i>cchTextMask</i>	INTEGER	Number of characters in the buffer pointed to by <i>pszText</i> . This parameter is only used when the segment receives attribute elements. It is not used when the segment specifies attribute elements.
<i>iImage</i>	INTEGER image	Index of the icon for the element of the list for the control.

<i>iParam</i>	INT(4)	32-bit value specific to the element.
<i>iIndent</i>	INTEGER	Image width number for indenting the image.

## LVM\_INSERTCOLUMNA% constant

**Description:**

Inserts a new column in the *List View* control.

**Syntax:**

**LVM\_INSERTCOLUMNA%** (*iCol*, *pCol*)

**Parameters:**

<i>iCol</i>	INTEGER	Index of the new column
<i>pCol</i>	POINTER	Address of the LVCOLUMN segment containing the attributes of the new column

**Value returned:**

The index of the new column, or -1 in case of failure.

## LVM\_INSERTITEM% constant

### Description:

Inserts a new item in a List View control.

### Syntax:

**LVM\_INSERTITEM%** (*pItem*)

### Parameters:

<i>pItem</i>	INTEGER	Address of an LVITEM segment that specifies the attributes of the List View item.
--------------	---------	---

### Value returned:

The index of the new item if successful, -1 otherwise.

### Comment:

The LVM\_INSERTITEM% is not usable to insert subitems. The *iSubItem* parameter of the LVITEM segment must be zero.

## LVM\_SETIMAGELIST% constant

**Description:**

Associates an image list with a *List View* control.

**Syntax:**

**LVM\_SETIMAGELIST%** (*iImageList*, *hIml*)

**Parameters:**

<i>iImageList</i>	INTEGER	Type of image list
<i>hIml</i>	POINTER	Handle to the image list to be associated

**Value returned:**

POINTER to the associated image list, or NULL in case of failure.

**Comment:**

The *iImageList* parameter can have one of the following constants as a value:  
LVSIL\_NORMAL% (image list with large icons), LVSIL\_SMALL% (image list with small icons) or LVSIL\_STATE% (image list with icons in state).

## LVM\_SETITEM% constant

### Description:

Associates an image list with a List View control.

### Syntax:

**LVM\_SETITEM%** (*pItem*)

### Parameters:

<i>pItem</i>	INTEGER	Address of a LVITEM segment that contains the new item attributes.
--------------	---------	--

### Value returned:

TRUE% if successful, FALSE% otherwise.



## LVS\_ICON% constant

**Description:**

Displays the elements from the *List View* as large icons.

**Syntax:**

**LVS\_ICON%**

## **LVS\_LIST% constant**

**Description:**

Displays the elements from the *List View* in list view.

**Syntax:**

**LVS\_LIST%**

## **LVS\_REPORT% constant**

**Description:**

Displays the elements from the *List View* in detailed view.

**Syntax:**

**LVS\_REPORT%**

## **LVS\_SMALLICON% constant**

**Description:**

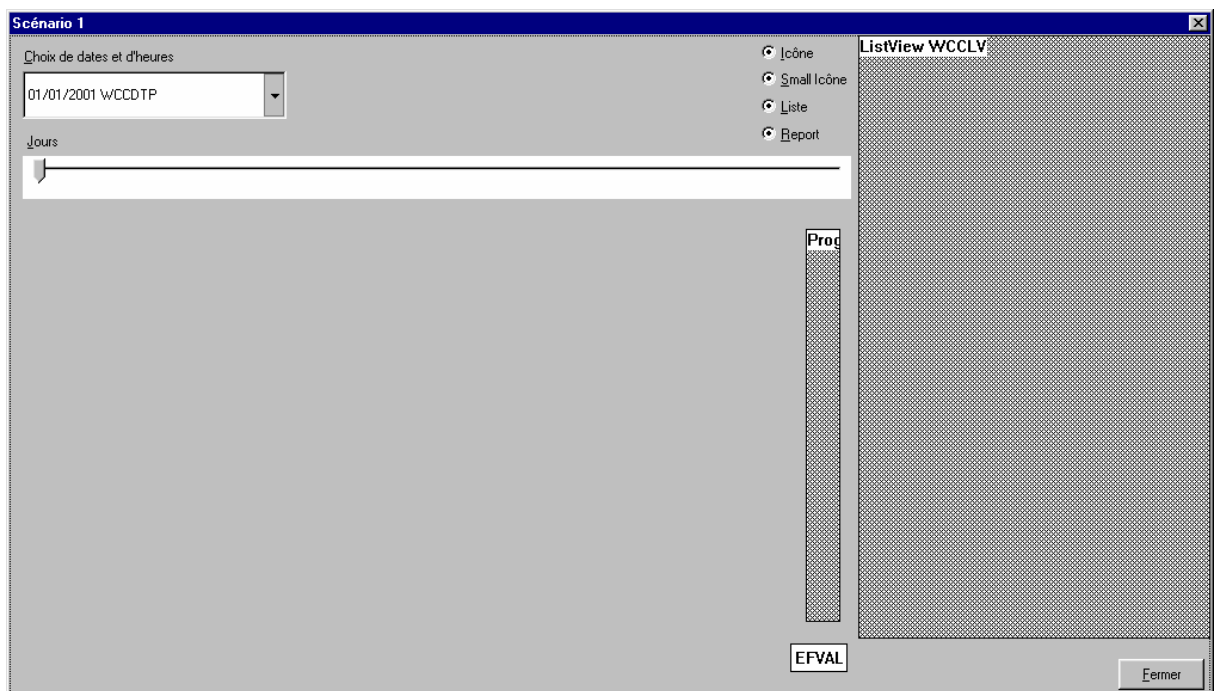
Displays the elements from the *List View* as small icons.

**Syntax:**

**LVS\_SMALLICON%**

## Example of creating a List View control

The following example corresponds to the window below:



This example allows you to select the display mode for the dates selected in the combo-box. By clicking on the associated radio-button, the user can change the display to large icons, small icons, list or detailed mode.

### *INIT event from the Scenario 1 window*

```
global IL_HDL_PI%, IL_HDL_GI%
global LVITEM slvitem
local bmp_hdl%, index%
local Pointer Hdlcol%
local LVCOLUMN lcol
local cstring Text$
local ret%

fill @slvitem, sizeof slvitem, 0
```

**; Creation of the image list**

```
IL_HDL_PI% = ImageList_Create (16, 16, ILC_COLOR24% , 1,
0)
IL_HDL_GI% = ImageList_Create (24, 24, ILC_COLOR24% , 1,
0)
if IL_HDL_PI% = 0
    message 'error', 'IL_HDL_PI% =0'
    exit
endif
if IL_HDL_GI% = 0
    message 'error', 'IL_HDL_GI% =0'
    exit
endif
```

**; Association bitmap <==> Image list**

```
bmp_hdl% = createbmp% ('d:\tests\natstar\16.bmp')
index% = ImageList_AddMasked (il_hdl_pi%, bmp_hdl%,
$FFFFFF)

bmp_hdl% = createbmp% ('d:\tests\natstar\32.bmp')
index% = ImageList_AddMasked (il_hdl_gi%, bmp_hdl%,
$FFFFFF)
```

**; Association ImageList / Listview / Styles**

```
ret% = WCCSEND% (int Wcclv.ctrl, LVM_SETIMAGELIST%, \
LVSIL_NORMAL%, il_hdl_gi%)
ret% = WCCSEND% (int Wcclv.ctrl, LVM_SETIMAGELIST%, \
LVSIL_SMALL%, il_hdl_pi%)
```

**; Creation of the columns for the style report presentation**

```
Text$ = "Date"
lcol.mask = LVCF_TEXT% + LVCF_FMT% + LVCF_WIDTH%
lcol.fmt = LVCFMT_LEFT%
lcol.cx = 100
lcol.pszText = @Text$
lcol.cchTextMax = length Text$

Hdlcol% = WCCSEND% (int wcclv.ctrl, LVM_INSERTCOLUMNA%,
0, \ @LCOL )

Text$ = "date%"
Hdlcol% = WCCSEND% (int wcclv.ctrl, LVM_INSERTCOLUMNA%,
1, \ @LCOL )
```

***INIT event for Date and Time Picker control***

```

local Pointer HdlItem%
local cstring item$

wccdtp = date%('01/01/2000')
wccdtp.range = date%('01/01/2000') & ';' & date%
('31/12/2001')

slvitem.isubItem = 0
slvitem.pszText = @item$
slvitem.mask = LVIF_TEXT% + LVIF_IMAGE%
slvitem.iItem = slvitem.iItem + 1
slvitem.isubItem = 0
item$ = date$(wccdtp)
slvitem.iImage = 0
slvitem.iItem = WCCSENDPTR% (int wcclv.ctrl,
LVM_INSERTITEM%, @sLVITEM )

item$=wccdtp
slvitem.mask = LVIF_TEXT% + LVCF_SUBITEM%
slvitem.isubItem = 1
HdlItem% = WCCSENDPTR% (int wcclv.ctrl, LVM_SETITEM%,
@sLVITEM )

```

***NOTIFY event for Date and Time Picker control***

```

local a%, m%, j%, datec$
local Pointer HdlItem%
local cstring item$

if NMHDR(param34%).code = DTN_CLOSEUP%
    a% = NMDATETIMECHANGE(param34%).ST.wyear
    m% = NMDATETIMECHANGE(param34%).ST.wmonth
    j% = NMDATETIMECHANGE(param34%).ST.wday
    datec$ = j% & '/' & m% & '/' & a%
    wcctbj = date% (datec$)

    item$ = date$(wcctbj)
    slvitem.isubItem = 0
    slvitem.pszText = @item$
    slvitem.mask = LVIF_TEXT% + LVIF_IMAGE%
    slvitem.iItem = slvitem.iItem + 1
    slvitem.isubItem = 0
    slvitem.iImage = 0
    slvitem.iItem = WCCSENDPTR% (int wcclv.ctrl, \
LVM_INSERTITEM%, @sLVITEM )

    item$=wccdtp
    slvitem.mask = LVIF_TEXT% + LVCF_SUBITEM%

```

```
        slvitem.iSubItem = 1
        HdlItem% = WCCSENDPTR% (int wcclv.ctrl, LVM_SETITEM%,
@sLVITEM )

        wccpb = wccpb + 1
        efvalpb = wccpb
    endif
```

***SELECTED event from a Large Icon View radio-button***

```
wcclv.type = LVS_ICON%
```

***SELECTED event from a Small Icon View radio-button***

```
wcclv.type = LVS_SMALLICON%
```

***SELECTED event from a List View radio-button***

```
wcclv.type = LVS_LIST%
```

***SELECTED event from a Detailed View radio-button***

```
wcclv.type = LVS_REPORT%
```

## The Tree View control

The creation of a *Tree View* control requires various stages which we will summarise in the paragraph on “Stages for creating a Tree View control”. The paragraph on “Example showing the creation of a Tree View control” shows a specific example. And finally, the paragraph on “Example repository” lists the functions and constants used in the example to give you a better understanding.

### Stages for creating a Tree View control

Creating a *Tree View* is divided into a number of stages:

1. Creating an Image List,
2. Associating the Image List with the *Tree View*,
3. Inserting the data into the control.



## **The APIs for creating a Tree View control**

This section lists the main functions and procedures designed to create a *Tree View* control with NatStar.

## **ImageList\_AddMasked function**



For more informations about ImageList\_AddMasked function, report to the section “The APIs for creating a ListView control”.

## ImageList\_Create function



For more informations about ImageList\_Create function, report to the section “The APIs for creating a ListView control”.

## ImageList\_GetImageCount function

### Description :

Retrieves the number of images of an image list.

### Syntax :

**ImageList\_GetImageCount** (*himgl*)

### Parameter :

<i>himgl</i>	INTEGER	Handle of the image list.
--------------	---------	---------------------------

## TVINSERTSTRUCT segment

**Description:**

Contains the information used to add a new element in the *Tree View* control. This segment should be used with the TVM\_INSERTITEM% constant.

**Syntax:**

**TVINSERTSTRUCT** (*hParent*, *hinsertAfter*, *itemex*)

**Parameters:**

<i>HParent</i>	INTEGER	Handle to the parent element.
<i>hinsertAfter</i>	POINTER	Handle to the element after which the new element or a constant is inserted.
<i>itemex</i>	TVITEMEX	Segment containing information about the element to be inserted.
<i>item\$</i>	TVITEM	Segment containing information about the element to be inserted.

**Comment:**

1. The *hInsertAfter* parameter can have the following constants as a value: TVI\_FIRST% (includes the element at the start of the list), TVI\_LAST% (includes the element at the end of the list), TVI\_SORT% (includes the element in alphabetical order) or TVI\_ROOT% (includes the element as a root element).

## TVM\_INSERTITEM% constant

### Description:

Inserts a new element in the *Tree View* control.

### Syntax:

**TVM\_INSERTITEM%** (*lpis*)

### Parameters:

<i>lpis</i>	INTEGER	Address of the TVINSERTSTRUCT segment which specifies the attributes of the element to be inserted.
-------------	---------	---

### Value returned:

POINTER to the new element, or NULL in case of failure.

## TVITEM segment

### Description:

Specifies or receives the attributes of the element from the *Tree View* control.

### Syntax:

**TVITEMEX** (*mask*, *hItem*, *state*, *stateMask*, *pszText*, *cchtextMax*, *ilImage*, *iSelectedImage*, *cChildren*, *lParam*)

### Parameters:

<i>mask</i>	INTEGER	Table of flags showing which of these elements is valid.
<i>hItem</i>	POINTER	Pointer to the control element.
<i>state</i>	INTEGER	Series of flags and image list index which shows the element's state. When we set the state of an element, the <i>stateMask</i> parameter indicates the valid bits of this parameter. When we import the state of an element, this parameter returns the current state of the bit indicated by the <i>stateMask</i> parameter.
<i>stateMask</i>	INTEGER	Bits of the <i>state</i> parameter which are valid.
<i>pszText</i>	POINTER	Pointer to a character string terminating with a NULL value. This character string contains the text of the element if the segment specifies the element's attributes.
<i>cchtextMax</i>	INTEGER	Size of the buffer pointed to by the <i>pszText</i> parameter. If the segment is used to set the attributes of elements, this parameter is ignored.
<i>ilImage</i> control	INTEGER	Index in the image list of the <i>Tree View</i> for the icon image icon to be used when the element is not selected.
<i>iSelectedImage</i>	INTEGER	Index in the image list of the <i>Tree View</i>

		control for the icon image to be used when the element is selected.
<i>cChildren</i>	INTEGER	Flag which indicates whether the element has child elements.
<i>lParam</i>	INT(4)	32-bit value to be associated with the element.

**Comment:**

1. The *mask* parameter can have one of the following constants as a value: TVIF\_IMAGE% (the *iImage* element is valid), TVIF\_SELECTEDIMAGE% (the *iSelectedImage* element is valid) or TVIF\_TEXT% (the *pszText* and *cchTextMax* elements are valid).
2. The *cChildren* parameter can have 0 as a value (no child element) or 1 (one or more child elements).



## TVITEMEX segment

### Description:

Specifies or receives the attributes of the element from the *Tree View* control.

### Syntax:

**TVITEMEX** (*mask*, *hItem*, *state*, *stateMask*, *pszText*, *cchtextMax*, *iImage*, *iSelectedImage*, *cChildren*, *lParam*)

### Parameters:

<i>mask</i>	INTEGER	Table of flags showing which of these elements is valid.
<i>hItem</i>	POINTER	Pointer to the control element.
<i>state</i>	INTEGER	Series of flags and image list index which shows the element's state. When we set the state of an element, the <i>stateMask</i> parameter indicates the valid bits of this parameter. When we import the state of an element, this parameter returns the current state of the bit indicated by the <i>stateMask</i> parameter.
<i>stateMask</i>	INTEGER	Bits of the <i>state</i> parameter which are valid.
<i>pszText</i>	POINTER	Pointer to a character string terminating with a NULL value. This character string contains the text of the element if the segment specifies the element's attributes.
<i>cchtextMax</i>	INTEGER	Size of the buffer pointed to by the <i>pszText</i> parameter. If the segment is used to set the attributes of elements, this parameter is ignored.
<i>iImage</i>	INTEGER	Index in the image list of the <i>Tree View</i> control for the icon image icon to be used when the element is not selected.
<i>iSelectedImage</i>	INTEGER	Index in the image list of the <i>Tree View</i>

		control for the icon image to be used when the element is selected.
<i>cChildren</i>	INTEGER	Flag which indicates whether the element has child elements.
<i>lParam</i>	INT(4)	32-bit value to be associated with the element.

**Comment:**

1. The *mask* parameter can have one of the following constants as a value: TVIF\_IMAGE% (the *iImage* element is valid), TVIF\_SELECTEDIMAGE% (the *iSelectedImage* element is valid) or TVIF\_TEXT% (the *pszText* and *cchTextMax* elements are valid).
2. The *cChildren* parameter can have 0 as a value (no child element) or 1 (one or more child elements).

## TVM\_DELETEITEM% constant

**Description :**

Removes an item and all these children from a TreeView control.

**Syntax :**

**TVM\_DELETEITEM%** (*hitem*)

**Parameter :**

<i>hitem</i>	INTEGER	Handle vers l'élément à supprimer.
--------------	---------	------------------------------------

**Value returned:**

Returns TRUE if successful, and FALSE otherwise.

**Notes :**

1. If *hitem* is set to TVI\_ROOT%, thus all items will be deleted.
  2. Once an item is deleted, its handle is invalid and can't be used.
-

## **TVM\_GETCOUNT% constant**

**Description :**

Returns the number of items in a Tree View control.

**Syntax :**

**TVM\_GETCOUNT%**

## TVM\_SETIMAGELIST% constant

**Description:**

Sets the type of image list for a *Tree View* control and redraws the control using the new images.

**Syntax:**

**TVM\_SETIMAGELIST%** (*iImage*, *himl*)

**Parameters:**

<i>iImage</i>	INTEGER	Type of Image List
<i>himl</i>	POINTER	Handle on the Image List

**Value returned:**

POINTER on the previous image list, or NULL in case of failure.

**Comment:**

The *iImage* parameter can correspond to the following constants: TVSIL\_NORMAL% or TVSIL\_STATE%.

---

## **TVSIL\_NORMAL% constant**

**Description:**

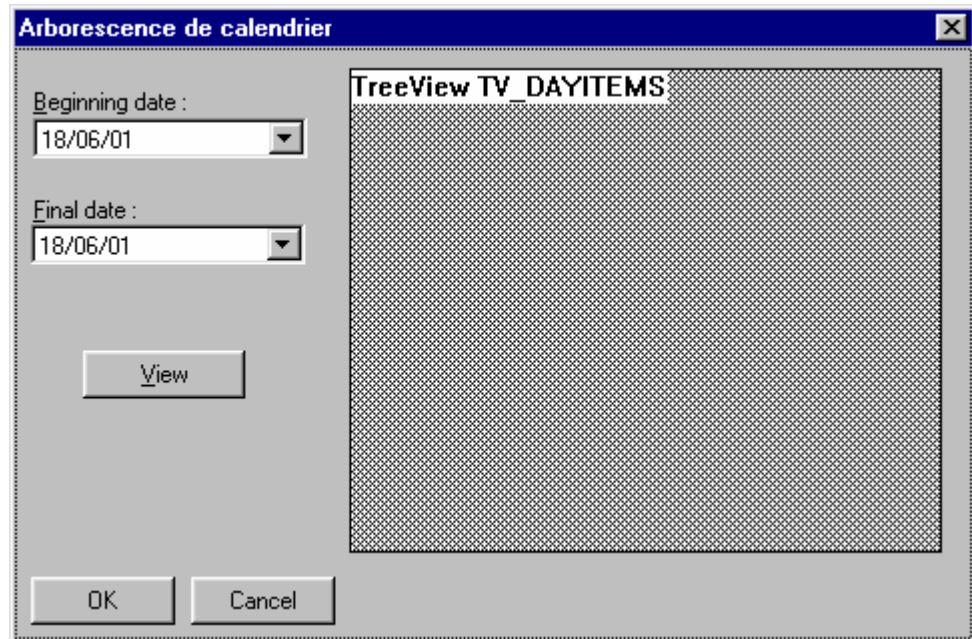
Corresponds to a normal image list, that is to say containing three images (selected, deselected and overlay).

**Syntax:**

**TVSIL\_NORMAL%**

## Example of creating a Tree View

The following example corresponds to the window below:



This example displays several years hierarchically. Select on the corresponding combo-boxes the dates. By clicking on the View button, you get the hierarchical display, in the Tree View control TV\_DAYITEMS. By expanding the branch for a year, you display the months. By expanding the branch for a month, you display the days.

*Functions used in the example to integrate in a NCL file :*

**; Tools functions for the Calendar Tree sample :**

```
segment PChar
    CString          CStr
endsegment
```

**; String functions**

**; Extracts a sub-string of a string (StrP) in order of  
; a string separator.**

```
function CStrExtract(StrP$, Order%, Separator$) return
CString
    Local Pointer PDeb, Pointer PSep
    Local Buffer$
```

```
PDeb = @StrP$
repeat
    PSep = PDeb + Pos$(Separator$,
PChar(PDeb).CStr)\ - 1
    if Int(PSep) < Int(PDeb) ; no separator found
        PSep = PDeb + Length(PChar(PDeb).CStr)
; points to the azt char
    endif
    Buffer$ = Copy$(PChar(PDeb).CStr, 1, PSep -
PDeb)
    PDeb = PSep + Length(Separator$)
    Order% = Order% - 1
until (Int(PDeb) > @StrP$ + Length(StrP$)) or
(Order%
= 0)

if Order% = 0
    Return Buffer$
else
    Return ""
endif
endfunction

;*****
; Converting functions
;*****

; Converts and formats a Date and/or a Time in a API
; SystemTime segment
function DateTimeFmtToSystemTime(aDate%, aTime%,
SYSTEMTIME@ stDateTime) return Int(1)
    Local Buffer$, ShortDay$
    Local Int iDayOfW(1)

    if not IsDate$(Date$(aDate%)) or not
IsTime$(Time$(aTime%)) or (@stDateTime = 0)
        Return FALSE%
    endif
```



```

; Initializes the record structure
Fill @stDateTime, SizeOf SYSTEMTIME, 0

; formats the date argument
Buffer$ = String$(aDate%, "yyyy m d ddd")
stDateTime.wYear = Int(CStrExtract(Buffer$, 1, " "))
stDateTime.wMonth = Int(CStrExtract(Buffer$, 2, " "))
stDateTime.wDay = Int(CStrExtract(Buffer$, 3, " "))
    ShortDay$ = UpCase(CStrExtract(Buffer$, 4, " "))
for iDayOfW = 0 to 6
    if UpCase(GetShortDay$(iDayOfW)) = ShortDay$
        stDateTime.wDayOfWeek = iDayOfW
        Break
    endif
endfor

; formats the time argument (the wMilliseconds member
; is ignored)
Buffer$ = String$(aTime%, "h:m:s")
stDateTime.wHour = Int(CStrExtract(Buffer$, 1, ":"))
stDateTime.wMinute = Int(CStrExtract(Buffer$, 2, ":"))
stDateTime.wSecond = Int(CStrExtract(Buffer$, 3, ":"))

Return TRUE%
endfunction

;*****
; Colors constants
;*****

; The high order byte, set to $80, allow to
; differentiate this RGB values
; with the NSDK palette entries (COL_BLACK% to
; COL_REDEF18% constants)

CONST RGB_BLACK%          $80000000
CONST RGB_GRAY%           $80C0C0C0    ; average tones

```

---

```
CONST RGB_BLUE%           $80FF0000
CONST RGB_GREEN%          $8000FF00
CONST RGB_CYAN%           $80FFFF00
CONST RGB_RED%            $800000FF
CONST RGB_MAGENTA%        $80FF00FF
CONST RGB_YELLOW%         $8000FFFF
CONST RGB_WHITE%          $80FFFFFF
```

***INIT event from the window:***

```
Local hImageList%, hBmImg%
Global g_nYear%, g_nMonth%, g_nDay%, g_nSelect%
; Creates the ImageList object utility
hImageList% = ImageList_Create(16, 16, ILC_MASK%, 4, 0)
if hImageList% = 0
    Message "Erreur", "Echec en création de la liste
d'images"
    Exit
endif
; Adds the image for the year items
hBmImg% = CreateBmp%('(PROJDIR)\IYEAR.BMP')
if hBmImg% <> 0
    g_nYear% = ImageList_AddMasked(hImageList%, hBmImg%,
RGB_GRAY%)
    DeleteBmp(hBmImg%)
endif
; Adds the image for the month items
hBmImg% = CreateBmp%('(PROJDIR)\IMONTH.BMP')
if hBmImg% <> 0
    g_nMonth% = ImageList_AddMasked(hImageList%, hBmImg%,
RGB_GRAY%)
    DeleteBmp(hBmImg%)
endif
; Adds the image for the day items
hBmImg% = CreateBmp%('(PROJDIR)\IDAY.BMP')
if hBmImg% <> 0
    g_nDay% = ImageList_AddMasked(hImageList%, hBmImg%,
RGB_GRAY%)
    DeleteBmp(hBmImg%)
endif
```

```

; Adds the image for the selected item
hBmImg% = CreateBmp('(PROJDIR)\ISELECT.BMP')
if hBmImg% <> 0
    g_nSelect% = ImageList_AddMasked(hImageList%, hBmImg%,
    RGB_GRAY%)
    DeleteBmp(hBmImg%)
endif
; Fails if not all the images were added
if ImageList_GetImageCount(hImageList%) < 4
    Message "Erreur","Anomalie durant l'insertion des
    images"
    Close(SELF%)
    Halt
endif
; Associates the ImageList to the TreeView control
if WCCSEND%(int TV_DAYITEMS.ctrl, TVM_SETIMAGELIST%,
TVSIL_NORMAL%, hImageList%)
endif

```

***EXECUTED event of the View button which displays the Tree View:***

```

local TVINSERTSTRUCT TVIns
local hItemYear%, hItemMonth%
local Label$, SavePtr%
local SYSTEMTIME stDate, SYSTEMTIME stPrevDate
local iDate%, dStart%, dEnd%
; Positions the egg-timer cursor
SavePtr% = GetPtr%
SetPtr GetSpPtr%(SPTR_WAIT%)
; empty the TreeView control if necessary
if WCCSend%(TV_DAYITEMS.Ctrl, TVM_GETCOUNT%, 0, 0)
    if not WCCSend%(TV_DAYITEMS.Ctrl, TVM_DELETEITEM%, 0,
    TVI_ROOT%)
        Message "Error","Failure on deleting all items"
    endif
endif
; Initialize the definition structure of an element
for
; insertion

```

```
Fill @TVIns, SizeOf TVIns, 0
; Initialize the common members for all items
TVIns.hInsertAfter = TVI_LAST%
TVIns.itemex.mask = TVIF_TEXT% BOR TVIF_IMAGE% BOR
TVIF_SELECTEDIMAGE%
TVIns.itemex.iSelectedImage = g_nSelect%
; Initialize the previous date structure
Fill @stPrevDate, SizeOf stPrevDate, 0
; Read the beginning and the final dates
dStart% = Int(CStrExtract(DTP_DATEDEB, 1, ","))
dEnd% = Int(CStrExtract(DTP_DATEFIN, 1, ","))
for iDate% = dStart% to dEnd%
    ; break-down the date
    if DateTimeFmtToSystemTime(iDate%, 0, stDate)
        ; break on year
        if stDate.wYear <> stPrevDate.wYear
            TVIns.hParent = TVI_ROOT%
            Label$ = "Year" && stDate.wYear
            TVIns.itemex.pszText = @Label$
            TVIns.itemex.iImage = g_nYear%
            hItemYear% = WCCSend%(TV_DAYITEMS.Ctrl,
TVM_INSERTITEM%, 0, @TVIns)
            if hItemYear% = 0
                Message "Error","The new item" &&
Label$ && "has not been inserted"
                Break
            endif
        endif
    ; break on month
    if stDate.wMonth <> stPrevDate.wMonth
        TVIns.hParent = hItemYear%
        Label$ = GetLongMonth$(stDate.wMonth - 1)
        TVIns.itemex.pszText = @Label$
        TVIns.itemex.iImage = g_nMonth%
        hItemMonth% = WCCSend%(TV_DAYITEMS.Ctrl,
TVM_INSERTITEM%, 0, @TVIns)
        if hItemMonth% = 0
            Message "Error","The new item" &&
Label$ && stDate.wYear && "has not been inserted"
```

```
                Break
            endif
        endif
        ; add the day item
        TVIns.hParent = hItemMonth%
        Label$ = GetLongDay$(stDate.wDayOfWeek) &&
stDate.wDay
        TVIns.itemex.pszText = @Label$
        TVIns.itemex.iImage = g_nDay%
        if WCCSend%(TV_DAYITEMS.Ctrl, TVM_INSERTITEM%,
0, @TVIns) = 0
            message "Error","The new item" && Label$
&& stDate.wMonth && stDate.wYear && "has not been
inserted"
            Break
        endif
    endif
    ; store the current iteration
    stPrevDate = stDate
endfor
; Restore the initial cursor
SetPtr SavePtr%
```

## Example of use

An example, created with Win32 controls, is provided by NatStar and NS-DK. This example is located in samples folder. To use it, carry out the following instructions :

1. Import the file explorer.exp by activating the *View/Import* menu.
2. Copy the file explorer.cfg into your GLOB/INI directory.

## Reference of the NSMISC library

The file NSMISC.NCL contains a number of functions: Only the WCCSEND\* functions concerning 32-bit Windows controls are referenced in this XML document.



For a more exhaustive introduction to the functions of the NSMISC library, see the NatStar or NS-DK on-line help.

The WCCSEND\* functions described below should be used according to the Windows messages.

## WCCSEND% function

### Description:

Allows you to send Windows messages requiring two parameters (wParam, lParam) to the Windows 32-bit control implemented in NatStar and NS-DK.



This function should therefore be used when the Windows message indicated in the MSDN documentation requires two parameters.

*e.g.*

The DTM\_SETRANGE message for the Date and Time picker control contains two parameters and therefore must be used with WCCSEND%.

MSDN definition:

DTM\_SETRANGE

wParam = (WPARAM) flags;

lParam = (LPARAM) lpSysTimeArray;

### Syntax:

**WCCSEND%** (*Pwcc, Msg, Wparam, lParam*)

### Parameters:

<i>Pwcc</i>	POINTER	pointer on the Windows control.
<i>Msg</i>	INTEGER	Windows message to be sent
<i>Wparam</i>	INTEGER	specifies the value of the message parameter
<i>lParam</i>	INT(4)	specifies the value of the message parameter

### Value returned:

INT(4)

**e.g.**

```
local Systemtime lsystemtime
local ret%

fill @lsystemtime, sizeof Systemtime, 0
lsystemtime.wyear = 2001
lsystemtime.wmonth = 4
lsystemtime.wdayofweek = 4
lsystemtime.wDay = 12
lsystemtime.wHour = 18
lsystemtime.wMinute = 10
lsystemtime.wSecond = 09
lsystemtime.wMilliseconds = 0

; assigns a value to the Date and Time Picker control

ret% = WCCSEND%(int DATETIMEPICKER.ctrl,DTM_SETSYSTEMTIME% \
,GDT_VALID%, @lSystemTime)
```



## WCCSEND0% function

### Description:

Allows you to send Windows messages which do not require any parameters to the Windows 32-bit control implemented in NatStar and NS-DK.



This function should therefore be used when the Windows message indicated in the MSDN documentation does not require any parameters.

*e.g.*

The DTM\_GETMCFONT message for the Date and Time picker control does not contain any parameters and therefore must be used with WCCSEND0%.

MSDN definition :

DTM\_GETMCFONT

wParam = 0;

lParam = 0;

### Syntax:

**WCCSEND0%** (*Pwcc*, *Msg*)

### Parameters:

<i>Pwcc</i>	POINTER	pointer on the Windows control.
<i>Msg</i>	INTEGER	Windows message to be sent

### Value returned:

INT(4)

*e.g.*

```
; read the value of a track bar control  
position%=wccsend0% (int wcctb.ctrl1, TBM_GETPOS%)
```

## WCCSEND1% function

### Description:

Allows you to send Windows messages requiring one parameter (wParam) to the Windows 32-bit control implemented in NatStar and NS-DK.



This function should therefore be used when the Windows message indicated in the MSDN documentation requires one parameter.

*e.g.*

The DTM\_GETMCCOLOR message for the Date and Time picker control contains one parameter and therefore must be used with WCCSEND0%.

MSDN definition:

```
DTM_GETMCCOLOR wParam = (WPARAM)(INT)iColor;

lParam = 0;
```

### Syntax:

**WCCSEND1%** (*Pwcc*, *Msg*, *wParam*)

### Parameters:

<i>Pwcc</i>	POINTER	pointer on the Windows control.
<i>Msg</i>	INTEGER	Windows message to be sent
<i>wParam</i>	INTEGER	specifies the value of the message parameter

### Value returned:

INT(4)

*e.g.*

```
local color%
; imports the background colour for a Date Time Picker control
; in RGB code
color% = WCCSEND1% (int DATETIMEPICKER.ctrl, \
                    DTM_GETMCCOLOR%, \
                    MCSC_BACKGROUND%)
```

## WCCSENDPTR% function

### Description:

Allows you to send Windows messages requiring a pointer type parameter to the Windows 32-bit control implemented in NatStar and NS-DK.

### Syntax:

**WCCSENDPTR%** (*Pwcc*, *Msg*, *lParam*)

### Parameters:

<i>Pwcc</i>	POINTER	pointer on the Windows control.
<i>Msg</i>	INTEGER	Windows message to be sent
<i>lParam</i>	INTEGER	specifies the value of the message parameter

### Value returned:

INT(4)

### e.g.

```
local Systemtime lsystemtime
local ret%

; returns the value of a Date Time Picker control

fill @lsystemtime, sizeof Systemtime, 0
ret% = WCCSENDPTR%(int DATETIMEPICKER.ctrl,DTM_GETSYSTEMTIME%,\
@lSystemTime)
if ret% = GDT_VALID%
    efda = lsystemtime.wyear
    efdm = lsystemtime.wmonth
    efdjs = lsystemtime.wdayofweek
    efdj = lsystemtime.wDay
    efdh = lsystemtime.wHour
    efdmi = lsystemtime.wMinute
    efds = lsystemtime.wSecond
    efdms = lsystemtime.wMilliseconds
endif
```

## WCCSENDSTR% function

**Description:**

Allows you to send Windows messages requiring a character string type parameter to the Windows 32-bit control implemented in NatStar and NS-DK.

**Syntax:**

**WCCSENDSTR%** (*Pwcc*, *Msg*, *lParam*)

**Parameters:**

<i>Pwcc</i>	POINTER	pointer on the Windows control.
<i>Msg</i>	INTEGER	Windows message to be sent
<i>lParam</i>	INTEGER	specifies the value of the message parameter

**Value returned:**

INT(4)

**e.g.**

```
; opens an .AVI file in an Animation control
local ret%

ret% = WCCSENDSTR%(INT WCCANIM.CTRL, ACM_OPEN%, \
"c:\temp\f.avi")
```