

SILVERDEV

COMPONENTS

Notice relative aux droits d'auteurs.

Les informations contenues dans ce document pourront faire l'objet de modifications sans préavis et ne sauraient en aucune manière engager **EXPERIA**. La fourniture du progiciel est régie par un octroi de licence ou un accord de confidentialité. Le progiciel ne peut être utilisé, copié ou reproduit sur quelque support que ce soit que conformément aux termes de cette licence ou de cet accord de confidentialité. L'acheteur ne peut effectuer des copies que dans le but de sauvegarde ou d'archivage.

Aucune partie du manuel et du progiciel ne peut être reproduite ou transmise par quelque moyen que ce soit, électronique ou mécanique, y compris par photocopie, enregistrement ou tout autre procédé de stockage, de traitement et de récupération d'informations, pour d'autres buts que l'usage personnel de l'acheteur sans permission expresse et écrite de la société **EXPERIA**.

IBM, AS/400, iSeries, System i, i5, Power I sont des marques déposées de International Business Machines Corporation. Windows est une marque déposée de Microsoft.

Tous les autres produits sont des marques déposées de leur société respective.

EXPERIA Europe
4, rue L.Beridot
Les jardins d'Epione
38500 VOIRON
FRANCE

Table des matières

Chapitre 1. Common properties	5	Reading an SFL and modified lines	32
Chapitre 2. TWin	7	Cell formatting	33
Chapitre 3. CMainMenu.....	8	Column.style	35
Linking a form and a CMainMenu	8	ColumnField.DataType	38
Editing a CMainMenu	8	Tabulation	38
Separator	9	Sorting	38
Images	10	Printouts.....	39
Shortcuts.....	10	Exports.....	41
Underlined letters.....	10	Multiselection	41
XpMenu	11	Images	41
		Display attributes.....	42
Chapitre 4. CPopupMenu.....	12	Chapitre 14. CDateEdit.....	44
Linking a popup menu to a component	12	Chapitre 15. CCalendar	46
Editing a popup menu	12	Chapitre 16. CMaskEdit.....	47
Notes	12	EditMask	47
		Specific editor.....	49
Chapitre 5. CMemo	13	Chapitre 17. CImage.....	50
Use	13	Design Time	50
WordWrap.....	14	Run Time.....	50
WantReturns	14	Zoomable.....	51
Text.....	15	Chapitre 18. CSplitter	52
		Direction	53
Chapitre 6. CRichEdit	16	Chapitre 19. CTreeView.....	55
Chapitre 7. CCheckBox.....	18	Editing in Design Time	55
Chapitre 8. CRadioButton.....	19	Editing in Run Time	56
Key and KeyChecked	19	Index	58
		Querying the selected items	59
Chapitre 9. CComboBox.....	20	Tick boxes.....	59
Items and Keys	20	Local copy	60
Run Time	22	Drag and Drop example	60
Sorted ComboBox.....	23	Chapitre 20. CTimer	62
Style.....	23	Chapitre 21. CTrayIcon	63
Chapitre 10. CListBox.....	24	Chapitre 22. CNavBar.....	65
Similarities with CComboBox	24	Building in design	65
MultiSelect	24	Add a control in a group	66
		OnLinkClick	66
Chapitre 11. CCheckListBox	26	Building groups at run time.....	66
Chapitre 12. CActionList	27	Chapitre 23. CScheduler	68
Shortcut	27	Adding events	68
Centralisation.....	28	Event properties	68
		Additional event values	69
Chapitre 13. CSFL.....	30	DateNavigator	69
Columns.....	30	Select a period programmatically.	70
Loop example	32		

Different views	70	Link.Font:	78
Resources.....	70	Link.Url:	78
Internal forms and popup menus	71		
Event drag and drop	71	Chapitre 26. CMapPoint	79
Modifying the duration of an event	72	Chapitre 27. CChart.....	80
Deleting an event.....	73	Chapitre 28. TChartSeries.....	81
Double-click an event.....	74	Fonctions.....	83
Double-click the diary.....	74	Propriétés.....	83
Detect a period change	74	Legendes	84
Images in an event.....	75	Etiquettes.....	85
Colour of an event.....	75	Titres.....	86
sddmview program example	76	Axes.....	87
		Pagination	92
		Fond	93
		Murs.....	95
		Zoom	96
		3D 98	
		Couleurs	99
Chapitre 24. CFontDialog, CColorDialog,		Impression avec graphique	100
COpenDialog, CSaveDialog,		Pdf avec graphique.	100
COpenPictureDialog,		Gdi+	100
CSavePictureDialog	77	Ascenseurs	101
Chapitre 25. CLinkLabel	78		
Link.Color:.....	78		

Chapitre 1. Common properties

The following properties are found on most components:

Align:

Use Align to align a control at the top, at the bottom, on the left or on the right of a form or a panel so that it always remains in this position regardless of the size of the form, panel or component. If the parent is re-sized, an aligned control also modifies its dimensions in order to continue to spread towards the top, bottom, left or right side of the parent.

For example: to use a panel component as a tool palette with various controls, attribute the `alLeft` value to the Align property of this panel. This value guarantees the position of the tool palette on the left edge of the form, with a height equal to that of the client height of the form.

The default value of Align is `alNone`, which means that a control remains where it is placed in the form or panel.

Anchors

Use the Anchors property to ensure that a control remains in its current position in relation to the edge of its parent, even if the parent is re-sized. If a parent is re-sized, the control retains its position in relation to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control adjusts when the parent is re-sized. For example: if the Anchors property of a control is set to `[akLeft,akRight]`, the control will be resized when the width of its parent changes.

Anchors are only re-sized if the parent is re-sized. So, for example, if a control is anchored on the opposite sides of a form during design and the form is created in a maximum size, the control will not be stretched since the form is not re-sized after creation of the control.

Caption:

Specifies a text string so that the user can identify the control.

Constraints:

Use the Constraints property to specify minimum and maximum heights and widths for the control. If Constraints contains maximum or minimum values, the control cannot be re-sized beyond these constraints.

Warning: Do not define constraints that may conflict with the value of the Align or Anchors properties. If these properties are conflicting, the control's response to re-size attempts is not well-defined.

Cursor:

Change the Cursor value to let the user know which action is possible when the mouse pointer enters the control. The Cursor value corresponds to the pointer index in the list of

pointers managed by the overall Screen field. In addition to the predefined pointers proposed by TScreen, an application can add its own personalised pointers to the list.

Font:

Determines the written text attributes over or inside the control.

Height:

Indicates the vertical height of the control, expressed in pixels.

Hint:

Contains the text string that appears when the user moves the mouse over the control.

Left:

Determines the horizontal coordinate, expressed in pixels in relation to the form, of the left edge of a component.

PopupMenu:

Identifies the popup menu associated with the control.

TabOrder:

Indicates the position of the control in the tabulation order of its parent.

TabStop:

Determines whether or not the user can tabulate in a control.

Tag:

Stores an integer value in a component.

Top:

Represents the vertical coordinate Y, expressed in pixels in relation to its parent or its container control, of the top left corner of a control.

Visible:

Determines whether or not the component is visible on the screen.

Width:

Determines the horizontal size, expressed in pixels, of the control or form.

Chapitre 2. TWin

TWin represents a standard window of an application.

BorderIcons:

Use the BorderIcons property to know or define the icons that will appear in the form's title bar. BorderIcons can use any combination of the following TBorderIcons values:

biSystemMenu The form has a Control menu (also called System menu).

biMinimize The form has a Minimize button.

biMaximize The form has a Maximise button.

biHelp If BorderStyle has bsDialog as a value or if biMinimize and biMaximize are excluded, a question mark will appear in the form's title bar. If the question mark is clicked, the cursor becomes crHelp; otherwise the question mark is not displayed.

BorderStyle:

Use the BorderStyle property to know or define the appearance and behaviour of the form border.

FormStyle:

Use the FormStyle property to know or define the form style. FormStyle can have any one of the following values:

fsNormal The form is neither an MDI parent nor an MDI child window.

fsMDIChild The form is an MDI child window.

fsMDIForm The form is an MDI parent window.

fsStayOnTop This form remains on top of the desktop and of all the project forms, except those whose FormStyle property contains fsStayOnTop. If a form with fsStayOnTop opens another form, neither of the two forms systematically remains on top.

WindowState:

Define the WindowState property to minimise, maximise or restore the form window. Consult WindowState to determine whether or not the form will be displayed minimised, maximised or normally on screen.

Chapitre 3. CMainMenu

Tab sheet: Standard

This component enables addition of a main menu to a window.

Linking a form and a CMainMenu

The forms have a property called Menu.

To get a form to use a CMainMenu, this Menu property refers to the CMainMenu. When you place a CMainMenu on a form, if the form's Menu property is empty, it is automatically given the value of the CMainMenu.

Editing a CMainMenu

To edit a CMainMenu in Designer, start by identifying the square representing the CMainMenu. This representative will only be visible in Designer. It will not be visible during program execution.

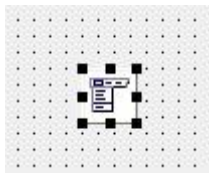


Figure 1

Right click on the representative and choose the Editor menu (Figure 2).

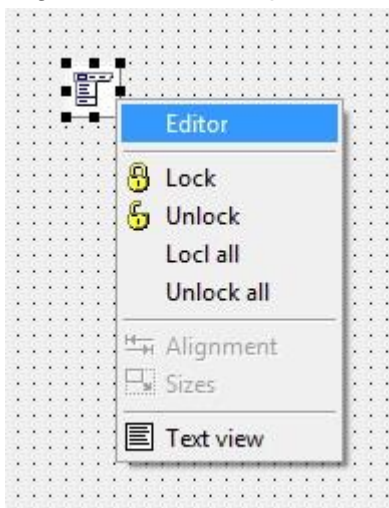


Figure 2

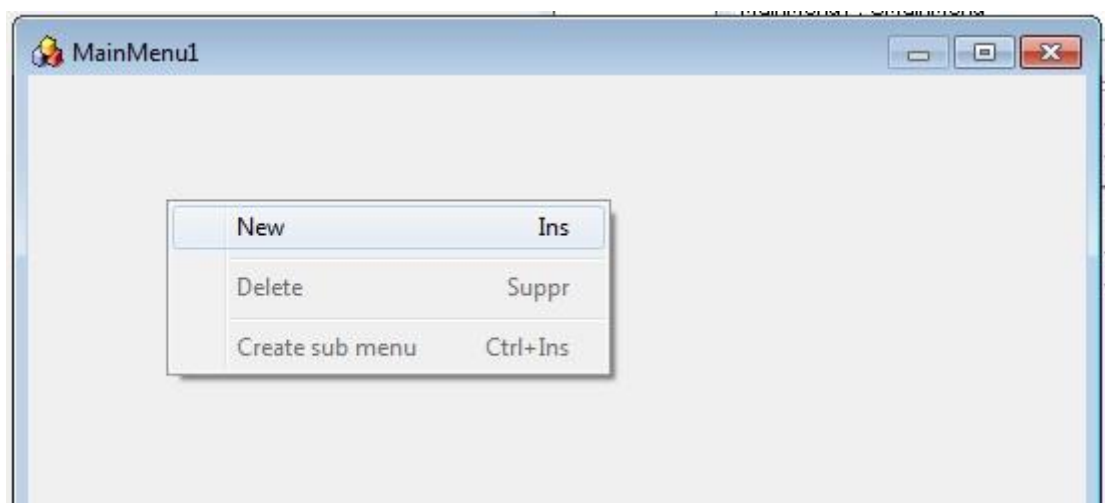
(You can also double-click on the representative or choose the Items property in the property inspector Figure 3)

**Figure 3**

An editor specific to the menus is displayed.

This editor enables the addition of TMenuItem type objects as menu items.

To add a MenuItem, right click on the menu editor then click on New.

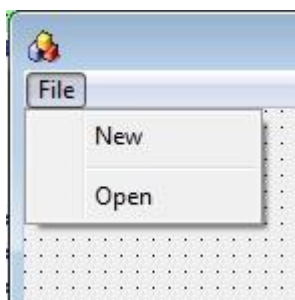
**Figure 4**

The MenuItem appears in both the form menu and in the menu editor.

To change the properties of the MenuItem, just select it in the menu editor and change its properties in the property inspector.

Separator

To add a separator (Figure 5) to a menu, the caption property of the item must be a hyphen.

**Figure 5**

Images

The Images property enables you to refer the CMainMenu to a list of images to be used by the Items.

Then, use the ImageIndex property of the TMenuItem type item.

Shortcuts

To create a keyboard shortcut, use the TmenuItem's ShortCut property.

Underlined letters

To underline a letter in the title, type the & character in front of it. This type of character is called a shortcut character. If Caption includes a shortcut character, the user can select the menu item by pressing Alt while pressing the underlined letter.

Depending on PC configuration, the character will be underlined all the time or only while the user is pressing ALT.

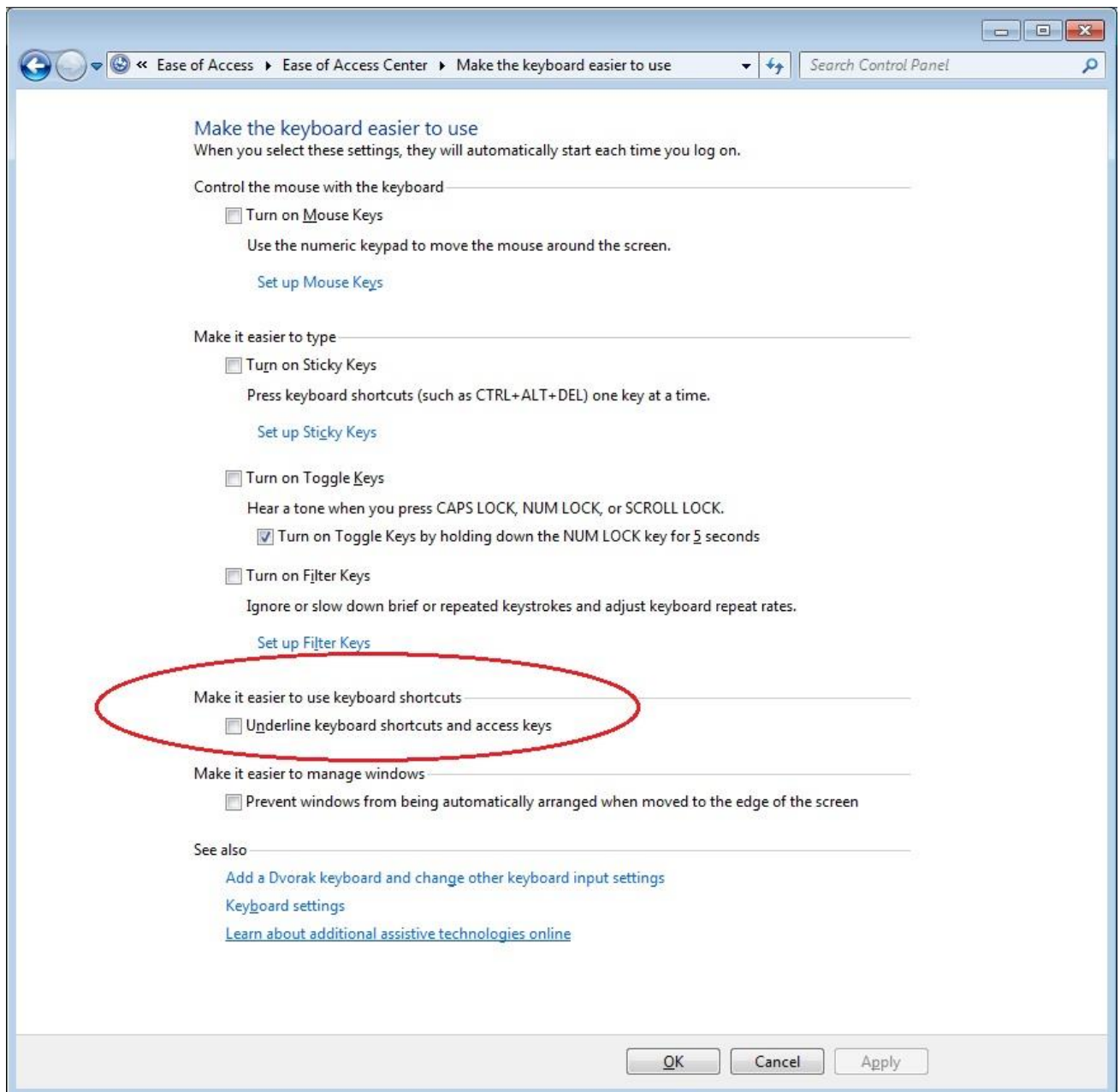


Figure 6

XpMenu

You can make the menu look like an XP menu by adding a Cxpmenu component from the system tab to the form (set the Xpmenu Active property to true).

Chapitre 4. CPopupMenu

Tab sheet: Standard

This component enables addition of a popup menu.

In Designer, the popup menu is not directly visible. It is represented by a square.

This representative is only visible in Designer (Design Time). It is not visible during program execution (Run Time).

Linking a popup menu to a component

All the components derived from TControl have a PopupMenu property.

Refer this property to a PopupMenu of the form using the property inspector (Figure 7).

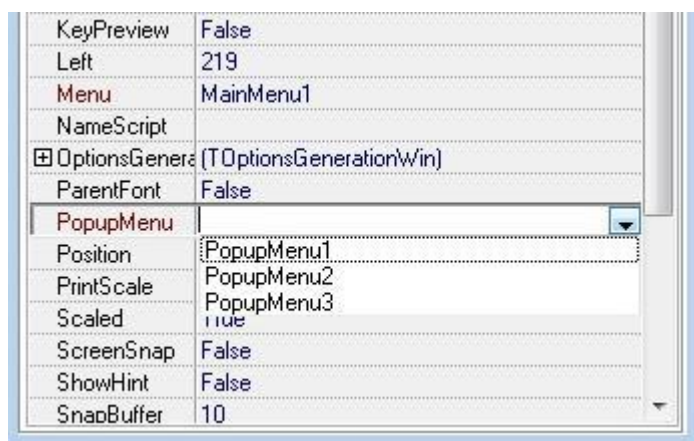


Figure 7

Editing a popup menu

The PopupMenu is edited in Designer in the same way as a CMainMenu.

Notes

See separator, images, shortcuts and XpMenu in CMainMenu

Chapitre 5. CMemo

Tab sheet: Standard

Use

CMemo is a component that enables display of a text on several lines.

The text is a TStrings type object.

This type of object can be found in several components. It is an object containing character strings. There is no limit to the number of character strings contained in the Tstrings (although the CMemo component itself is limited to 64 kb in Windows 9x).

The functions that enable handling of TStrings type objects are:

sdSIAdd	Add a string
sdSIClear	Delete all strings
sdSIGet	Query the value of a string
sdSIInsert	Insert a string

Furthermore, TStrings type objects have a Count property that indicates the number of lines.

Note:

The index of the first item of the TStrings object is 0.

The following example shows how to fill in a component by reading a file and how to fill in a file by reading a component.

File description:

```
R FCOMMENTS
  IDBOOK      P  4,0      1      3      3 Book ID
  IDLINE      P  2,0      4      2      5 Line seq
  LINE        A 100      6     100    105 Line data
K IDBOOK
K IDLINE
```

File to component:

```
P LOADCOMMENTS      B
D                    PI
C                    callp      sdSIClear (F2: 'Comments': 'Lines')
C IDBOOK            setll      SDDMCOM
C IDBOOK            reade (N)   SDDMCOM      77
C *in77             doweq      *off
```

C		callp	sdSlAdd(F2:'Comments': 'Lines':LINE)	
C				
C	IDBOOK	reade(N)	SDDMCOM	77
C		enddo		
P		E		

Component to file:

P	WRITECOMMENTS	B		
D		PI		
D	Nbline	s	5u 0	
D	WidBook	s	like(IDBOOK)	
C		eval	WidBook=IDBOOK	
C	WidBook	setll	SDDMCOM	
C	WidBook	READE	SDDMCOM	76
C	*in76	doweq	*off	
C		delete	SDDMCOM	
C	WidBook	READE	SDDMCOM	76
C		enddo		
C		eval	NbLine =sdGetNum(F2:'Comments':	
C			'Lines.Count')	
C		eval	IDLINE =0	
C		dow	IDLINE < NbLine	
C		eval	LINE=sdSlGet(F2:'Comments.Lines'	
C			:IDLINE)	
C		eval	IDLINE = IDLINE + 1	
C		WRITE	FCOMMENTS	
C		enddo		
P		E		

WordWrap

Assign the value True to the WordWrap property if you want the text entry control to insert a new line character in the text in the right margin so that it remains within the client zone. The new line character is only apparent: the text contains no new line character that is not explicitly entered. Assign the value False to the WordWrap property if you want the text entry control to show only line changes where new line characters are explicitly entered in the text.

Note: There is no need for a horizontal scroll bar if WordWrap is set to True.

WantReturns

Determines whether or not the user can insert new line characters in the text.

Assign the value True to the WantReturns property to enable the user to enter new line characters in the text. Assign the value False to the WantReturns property to enable the form to manage new line characters.

For example, in a form with a button by default (e.g. an OK button) and a memo control, if `WantReturns` is `False`, pressing `Return` will activate the OK button by default. If `WantReturns` is `True`, pressing `Return` will insert a new line character in the text.

Note: *If `WantReturns` is `False`, users can still enter a new line character in the text by pressing `Ctrl+Return`.*

Text

It is possible to retrieve the text in a single action by using the `Text` property.
Two characters are inserted between each character string of the `Lines` object: a carriage return and new line character.

Chapitre 6. CRichEdit

CRichEdit is a component equivalent to the CMemo component, with the extra possibility of formatting text (selFontName, selFontStyle, selFontSize, selFontColor, selBgColor)

```

C      callp      sdSetString(F1:'RichEdit1':'SelText':
C                  'le ' :*OFF)
C      callp      sdSetSet(F1:'RichEdit1':'SelFontStyle':
C                  'fsBold':'fsUnderline')
C      callp      sdSetString(F1:'RichEdit1':'SelFontName':
C                  'Arial')
C      callp      sdSetNum(F1:'RichEdit1':'SelFontSize':
C                  15)
C      callp      sdSet(F1:'RichEdit1':'SelFontColor':
C                  'clRed')
C      callp      sdSet(F1:'RichEdit1':'SelBgColor':
C                  'clYellow')
C      callp      sdSetString(F1:'RichEdit1':'SelText':
C                  'mot')

```

Result:

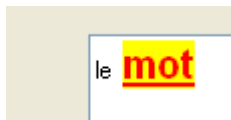


Figure 8

Example 2:

Underline a word in the text:

```

D Word      c      'peux'
D lines     s      *
D line      s      1000   varying
D lentot    s      10u 0  inz(0)
D pos       s      10i 0
D i         s      10u 0
C      eval    lines = sdgetlist(f1:'richedit1.lines')
C      if      lines <> *null
C      for     i=0 to sdgetlistcount(lines)-1
C      eval    line=sdgetlistLabel(lines:i)
C      eval    pos= -1

```



```
C      eval      pos=%scan(word:line)
C      dow      pos <>0
C      callp     sdsetint(f1:'richedit1':'selstart':
C                lentot + Pos-1)
C      callp     sdsetint(f1:'richedit1':'sellength':
C                %len(word))
C      callp     sdSet(F1:'RichEdit1':'SelBgColor':
C                'clYellow')
C      if        pos + %len(word) < %len(line)
C      eval      pos=%scan(mot:line:pos +%len(word))
C      else
C      eval      pos=0
C      endif
C      enddo
C      eval      lentot=lentot+ %len(line) +2
C      endfor
C      callp     sdfreelist(lines)
C      endif
```

Chapitre 7. CCheckBox

Tab sheet: Standard

The CCheckBox component is a tick box. It enables representation of a Boolean.

When the CheckBox is ticked, the Checked property is True.

There are two ways of changing or checking the value of Checked.

Use the sdsetBool and sdGetBool functions on the Checked property.

For example:

C	callp	sdSetBool (F1: 'CHK1': 'CHECKED': *OFF)
---	-------	-----------------------------------------

The component also has the Value, ValueFalse and ValueTrue properties.

These three properties work together.

They are character string type properties and can therefore be modified using the sdGet and sdSetString functions.

When you check the value of the Value property, you retrieve the value of the ValueFalse property if the box is unticked and the ValueTrue property if the box is ticked.

Similarly, if you assign the corresponding string to ValueTrue in Value, this will tick the box (any other value will untick the box).

Chapitre 8. CRadioButton

Tab sheet: Standard

Use CRadioButton to add a radio button to a form. Radio buttons offer users a set of mutually exclusive options, i.e. a single radio button of a group can be selected at once at any given moment. When the user selects a radio button, the previously selected radio button is deselected. Radio buttons are often grouped into a box (CGroupBox). Start by adding the group box to the form, then choose the radio buttons from the component palette and place them in the group box.

By default, all the radio buttons placed within the same container windowed control (like CRadioGroup or CPanel) belong to the same group. For example, two **radio buttons** on a form can only be selected at the same time if they belong to different containers, such as two group boxes.

Key and KeyChecked

Each RadioButton can contain a string type value.

To check the value of the Key property of the CRadioButton component whose Checked property is True among the CRadioButtons from the same container, use the KeyChecked property of one of them.

Chapitre 9. CComboBox

Tab sheet: Standard

The ComboBox component enables selection of an item from a scroll list.

Items and Keys

It is often useful to be able to display a name in a scroll list even if the information of interest is actually the value associated with the name.

For example, you want to display the list of authors in the scroll list, but the important information is the code of the author chosen by the user.

To enable this, the ComboBox component has Item and Key properties.

These two properties are TStrings properties, which means they may be a series of alphanumeric strings (similar to a table of strings).

The Items property contains the list of names to be displayed in the scroll list.

The Keys property contains the list of keys.

The developer must ensure that the number of items in the Keys property is the same as the number of items in the corresponding Items property.

Furthermore, a code and name pair must be in the same position in the Items and Keys objects.

Items and Keys objects can be filled in during design using the property inspector.

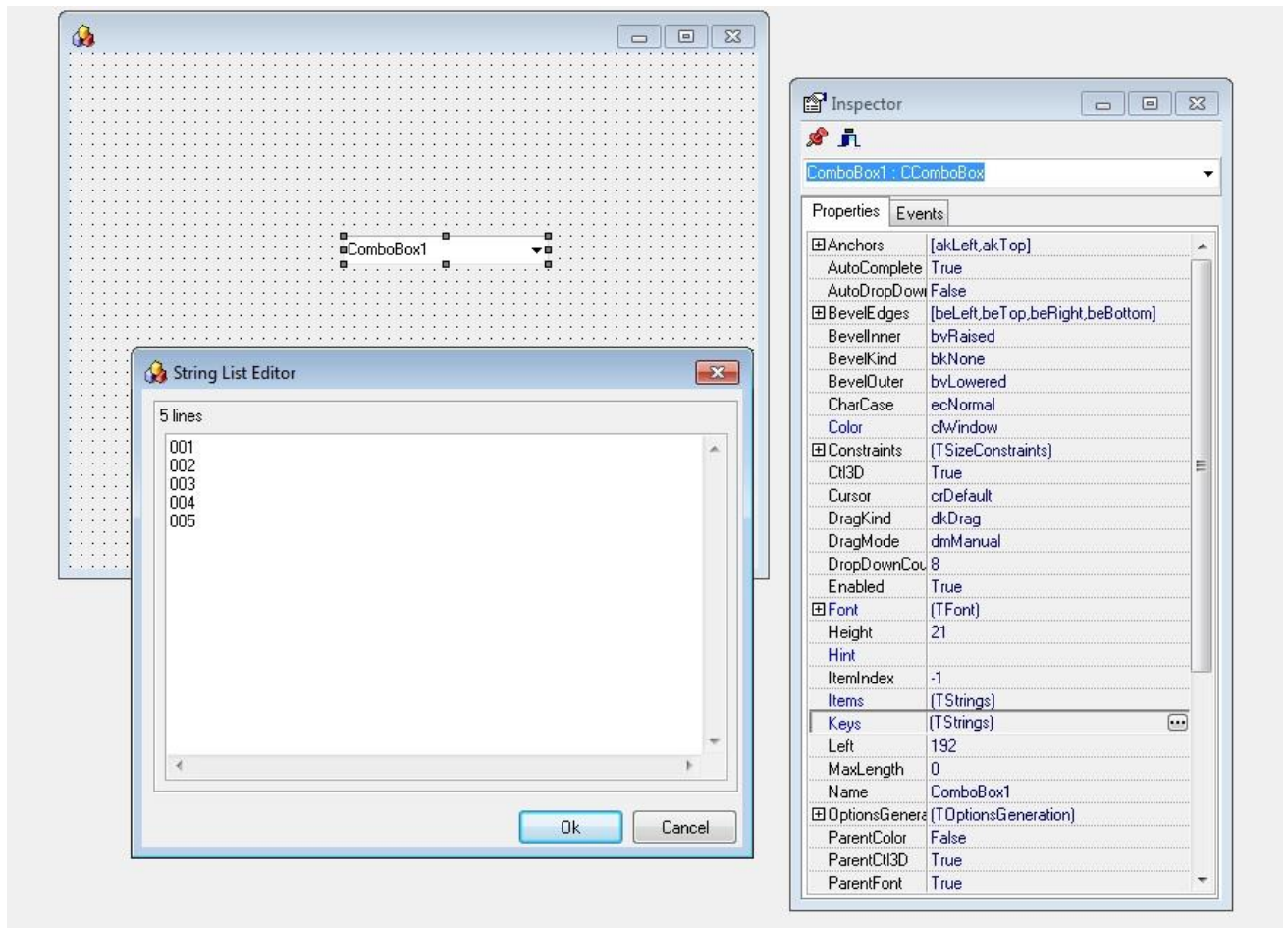


Figure 9

For example:

Strings contained by Items	Strings contained by Keys
Jules Vernes	0001
Romain Gary	0002
Mark Twain	0003
Emile Zola	0004
Marguerite Duras	0005

The user will see:



Figure 10

But the developer needs to know the code of the selected item (in this case, Emile Zola, code 004).

The developer can find out the code of the selected item by querying the KeySelected property of the ComboBox component.

```
C          eval          IDAUTAUT=sdGetInt (F1: 'COMBOAUTHOR' :
C                                     'KeySelected')
```

In this example:

F1 is the identifier of the form on which the ComboBox is placed.

COMBOAUTEUR is the name of the ComboBox component and KeySelected is the name of the property whose value we want to retrieve in the IDAUTEUR field.

Note 1: Here, we use the sdgetInt function because IDAUTEUR is numeric.

If the key is alpha-numeric, the sdget function should be used.

Note 2: The name Keys is used because often the keys to a table are stored, but the strings contained in Keys do not have to be unique.

Run Time

In the above example, the list of names and keys was filled in during design time.

Usually, the data contained in the comboBox come from a table and must therefore be filled in by programming. Tstrings type objects (Items and Keys) are handled using the following functions:

sdAssignTo	Assign a TStrings object to a TStrings object
sdSIAdd	Add a string to a TStrings
sdSIClear	Delete a TStrings
sdSIGet	Retrieve the string at a position in the TStrings
sdSIInsert	Insert a string in a Tstrings

For example:

```
* Clear items and keys
C          callp          sdSIClear (F1: 'COMBOAUTHOR': 'items')
C          callp          sdSIClear (F1: 'COMBOAUTHOR': 'keys')
C      *LOVAL          setll          SDDMAUT
C          READ          SDDMAUT          55
C      *ON              DOWNE          *IN55
* Add item and key to the combobox for each record read
C          callp          sdSIAdd (F1: 'COMBOAUTHOR': 'items':
C                                     %trim(NAMEAUT) )
C          callp          sdSIAdd (F1: 'COMBOAUTHOR': 'Keys':
C                                     %char (IDAUT) )
C          READ          SDDMAUT          55
C          enddo
```

Sorted ComboBox

The ComboBox component has a Boolean type Sorted property that determines whether or not the items in the list are sorted alphabetically. By default, this property is False.

If it is True, the items are sorted according to the name (as seen by the user).

The keys are thus moved automatically in order to maintain the key/name match.

However, when loading, the Sorted property must be set to False before loading and reset to True afterwards. The full code is therefore:

```
c          callp      sdSetBool(F1: 'COMBOAUTHOR':'Sorted':*off)
* Clear items and keys
C          callp      sdSlClear(F1: 'COMBOAUTHOR':'items')
C          callp      sdSlClear(F1: 'COMBOAUTHOR':'keys')
C          *LOVAL      setll      SDDMAUT
C          READ        SDDMAUT          55
C          *ON         DOWNE      *IN55
* Add item and key to the combobox for each record read
C          callp      sdSlAdd(F1: 'COMBOAUTHOR':'items':
C                      %trim(NAMEAUT))
C          callp      sdSlAdd(F1: 'COMBOAUTHOR':'Keys':
C                      %char(IDAUT))
C          READ        SDDMAUT          55
C          enddo
c          callp      sdSetBool(F1: 'COMBOAUTHOR':'Sorted':*on)
```

Style

If you want to prevent the user from entering text in the ComboBox, set the Style property to csDropDownList.

Chapitre 10. CListBox

Tab sheet: Standard

This component enables display of a list of strings.

Its use is similar to that of the ComboBox.

The advantage of this component is that you can select several items (using the SHIFT or CTRL buttons).

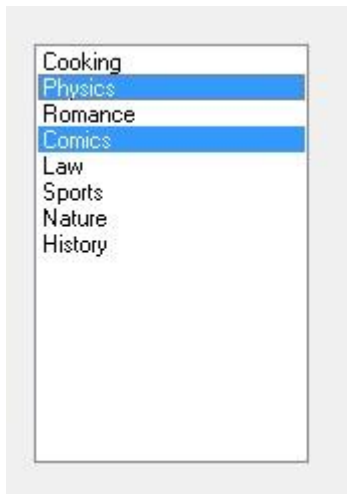


Figure 11

Similarities with CComboBox

Use of the Items and Keys properties (see CcomboBox Run Time)

MultiSelect

The CListBox component has a Boolean type MultiSelect property.

This property is False by default. If it is True, the user can selected several items.

A KeySelected property exists in CListBox. This property is used as for the CComboBox, but is insufficient if the MultiSelect property is True.

In this case, the developer must use the FirstSelected and NextSelected properties.

These two properties are used together to know the selected items.

The FirstSelected property indicates the position of the first item selected (the first item in the list with index 0).

If no item is selected, this property indicates -1.

Then, the developer can find out the associated key using the `sdSlGet` function of the `Keys` property.

Similarly, a query of the `NextSelected` property indicates the position of the next selected item or `-1` if there are no other items selected.

For example:

```
C      eval      Line = sdgetInt(F1:'MultiSelect1':
C      'FirstSelected')
C      dow      line <> -1
C      eval      Key = sdSlGet(F1:
C      'MultiSelect1':'Keys':line)
C      eval      Line=sdgetInt(F1:'Multiselect1':
C      'NextSelected')
C      enddo
```

Note:

The `FirstSelected` and `NextSelected` properties are read only.

Chapitre 11. CCheckListBox

Tab sheet: Standard

This component is similar to the CListBox component in the use of Items and Keys properties (see CComboBox Run Time).

Next to each character string, there is a tick box. The list of ticked boxes can be viewed using the FirstChecked and NextChecked properties.

For example:

```
C          eval      Line = sdgetInt(F1:'CheckSelect1':
C          'FirstChecked')
C          dow       line <> -1
C          eval      Key  = sdSlGet(F1:
C          'CheckSelect1':'Keys':line)
C          eval      Line = sdgetInt(F1:'CheckSelect1':
C          'NextChecked')
C          enddo
```

Chapitre 12. CActionList

Tab sheet: Standard

The CActionList component is a non visual component.

The CActionList object has no purpose in itself.

However, this component has a collection of TAction type objects that can be useful.

To access the TAction objects in Design Time, double click on the icon representing the CActionList object. You will access the collection editor.

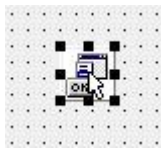


Figure 12

You can now modify the properties of each item of the collection using the property inspector.

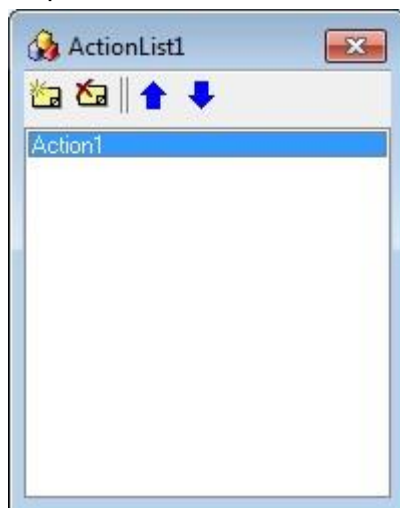
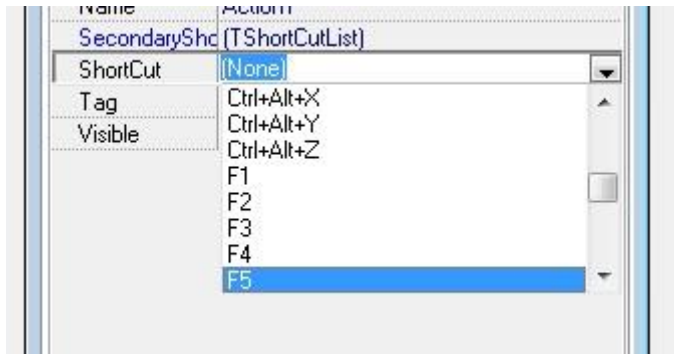


Figure 13

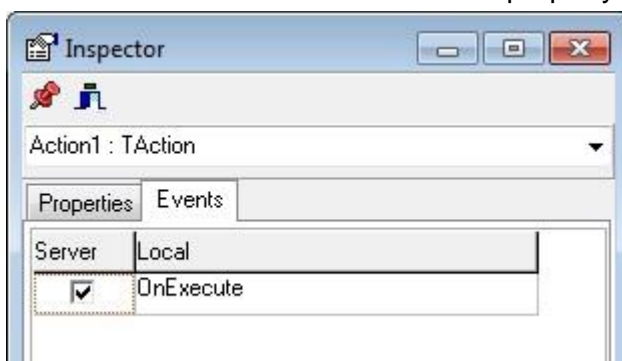
Shortcut

The main advantage of the TAction object is that it has a ShortCut property.

**Figure 14**

If you assign value F5 to the ShortCut property of a TAction, the TAction object will trigger an event.

Ask for this event to be notified in the property inspector using the events tab.

**Figure 15**

It is thus possible to trigger a RPG function when the user presses F5.

Note:

Be careful with conflicting shortcuts.

Centralisation

The second advantage of TAction type objects is that many components can refer to a TAction (Button, MenuItem, etc.).

When a component refers to a TAction, one of its events (often OnClick) triggers the TAction OnExecute event.

For example:

You may want the click of a button and the click on a menu item to trigger the same procedure on the server.

If so, you do not have to ask for the notification of events on the button and the menu item.

Just refer the “action” properties of the button and menu item to a TAction object and associate a function with the TAction OnExecute event.

Similarly, modification (in design time or run time) of a TAction property results in modification of the same property on all components referring to this TAction.

Chapitre 13. CSFL

Tab sheet: Additional

The CSFL component enables display of data in a grid.

The data may or may not be modifiable.

Each column can have specific display properties.

This component is therefore the equivalent of the sub-files on System i, hence its name SFL for Sub File.

Columns

When you place a CSFL component in Designer, it has no column.

To add columns, you have to use the collection editor (columns are a collection of TSFLColumn type objects).

To access the collection editor, edit the Columns property in the property inspector.

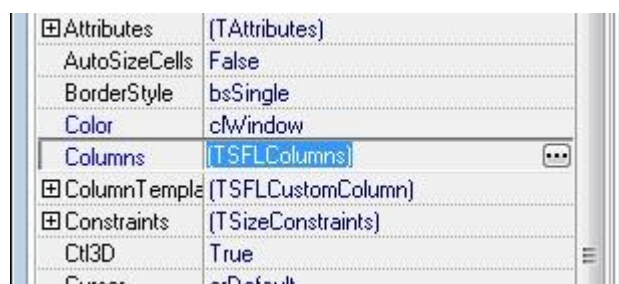
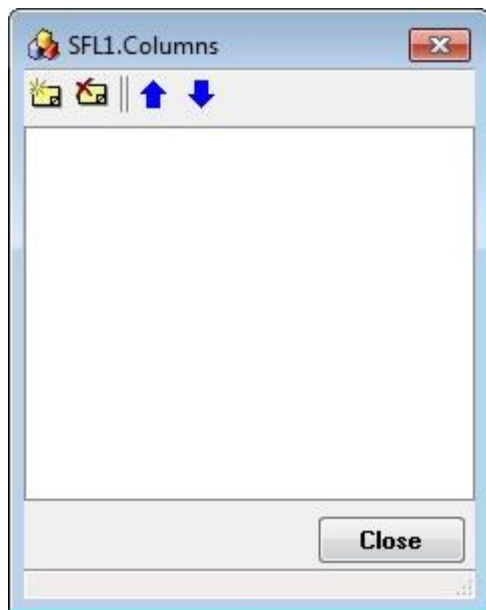


Figure 16

It is also possible to access the collections editor by double clicking on the component. In either case, the following window is displayed.

**Figure 17**

Use the two buttons to add or delete columns.

Each column has its own properties.

The most important ones are Style and ColumnField.FieldName.

The Style property determines the type of in situ editor for the column (date, time, Boolean, scroll list, button triggering an event or simple text entry).

The ColumnField.FieldName property is a string of characters that enables specification of the value of a cell. It will be possible to modify the value of a cell with the traditional functions using the following code:

```
C      callp sdSetString(f1:'sf11':'cells(4,5)':'Demo')
```

This writes 'Demo' in the cell in line 5, column number 4.

However, this method is not recommended as the columns can be moved by the user and you may not know what column 4 corresponds to.

Furthermore, it is impractical as in a loop that fills in all lines, you would have to convert a numeric counter into alphanumeric as follows:

```
C      callp sdSetString(f1:'sf11':'cells(4,'%char(cnt)')':'Demo')
```

To avoid this, special functions exist for the CSFL component. These functions start with sdSetCell.

As parameters, these functions are called columnField.FieldName plus the line number.

For example:

```
C      callp sdSetCell(f1:'sf11':'ColumnName':cnt:'Demo')
```

There are different functions that take a character string, numeric field or date type field as last argument.

These functions are:

sdSetCell, sdSetCellNum, sdSetCellDate, sdSetCellTime, sdSetCellNull

There is an advantage to these functions. The lines do not have to exist in the CSFL before being filled in.

Just call one of these functions on line 10 if there are only 5 lines in the component and the required number of lines will be added automatically.

Note:

In the cells(4,5) expression, 4 corresponds to the fifth column as the columns are numbered from 0 and 5 corresponds to the sixth line. In general, the header line is line 0.

Loop example

Here is an example that shows how to fill in the code with the content of a grid.

The grid has two columns, one containing the identifier of a book and other its title.

Note: the identifier column can be made invisible.

```

C      callp      sdClearSubFile(MainForm: 'SubFile1')
C      eval      Row = 0
C      *LOVAL     setll      sddmbks
C      READ      sddmbks      55
C      *ON       DOWNE     *IN55
C      eval      Row =Row + 1
C      callp      sdSetCellNum(MainForm: 'SubFile1': 'IDBOOK':
C                  Row:L_IDBOOK)
C      callp      sdSetCell(MainForm: 'SubFile1': 'TITLE':
C                  Row:L_TITLE)
C      READ      sddmbks      55
C      ENDDO
C      callp      sdReinitAll(MainForm: 'SubFile1')
```

Reading an SFL and modified lines

If the grid is in data entry, it must be read again to update the database.

It is not necessary to read each line: the CSFL component has FirstModified and NextModified properties that enable the modified lines to be consulted.

The content of the cells is then read using the sdgetCell type functions.

When a line is modified by the user or by programming, it is marked as modified. To remove the modified mark, use the sdReinit function that takes the line number as an argument.

To mark all the lines as unmodified, use the `sdReinitAll` function.
This is why in the previous example, a call to this function follows the filling loop.

The `FirstModified` and `NextModified` functions indicate -1 if there are no more modified lines.

For example:

```
PVALIDATION      B
D                  PI
D form              5u 0 value
DRow              s      10  0
*****
C                  eval      Row = sdGetInt(f1:'SFL1': 'FirstModified')
C                  dow      Row <> -1
C                  eval      test =sdGetCellNum(f1:'SFL1': 'IDBOOK':Row)
C      test        chain      LIVRES      55
C                  if      *IN55 =*OFF
C                  eval      L_TITLE =sdGetCell(f1:' SFL1': 'TITLE':Row)
C                  endif
C                  eval      Row = sdGetInt(f1:'SFL1': 'NextModified')
C                  enddo
C                  callp      sdReinit(f1:'SFL1')
```

Note:

The modified lines can be viewed: include the `goIndicator` and `goStar` values in the `Options` property. The `goIndicator` value indicates that a fixed column is added on the left and the `goStar` value indicates that a star will appear in this column for each modified line.

Note:

There is another way of retrieving the content of a sub-file.
The `sdGetSfl` function enables the data displayed in a grid to be copied locally. See **Erreur ! Source du renvoi introuvable.**

Cell formatting

You can modify the text colour, background colour, font style, alignment and even add a background image to a cell.

The CSFL component uses its `Color` property first to define cell background.

Then, the `Attributes` property is used to determine the formatting of odd or even-numbered cells.

Then, the column's `ColumnAttribute` property is used to determine the format of the cells in this column.

Finally, you can modify the cell background colour by programming, using the `ExAttributes` property.

For example:

The following example shows the priority between the various formats.

	col1	col2

Figure 18

CSFL Color property is clWhite

	col1	col2

Figure 19

CSFL Color property is clWhite

Attributes.Even.BgColor property is clInfoBk

	col1	col2

Figure 20

CSFL Color property is clWhite

Attributes.Even.BgColor is clInfoBk

Column 1's ColumnAttribute.BgColor property is clLime

CSFL Color property is clWhite
 Attributes.Even.BgColor is clInfoBk
 Column 1's ColumnAttribute.BgColor property is clLime
 CSFL ExAttributes property has an item whose BgColor property is clFuschia. This attribute was applied by the instruction:
 callp sdFormatCell(f1:'sf11':col1':3:0)

Note: For fixed columns, colours creating an impression of relief are automatically calculated.

	col0	cc
	01	11

Figure 22

Column.style

The `Column.style` and `ColumnField.DataType` properties are related. Modification of one modifies the other.

The `Style` property of columns can have any of the following values:

csSimple:

The column cells can be edited directly.

csButton:

A button appears in the cell when the user clicks.

King	20
King	10
King	10

Figure 23

An event is triggered when the user clicks on this button: the onEllipsis event.
This event has parameters that enable identification of the cell that was clicked.

```

*/EVENT SFL1_OnEllipsis
d parm          ds          based(pevtinf)
d Win           5u 0
d evt           48a
d ColName       100a varying
d Row           10i 0

```

csPicklist:

A scroll list appears when the user clicks in the cell.

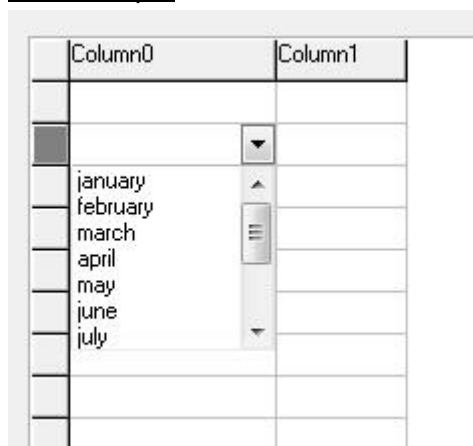
The column has PickItems and PickKeys properties.

These two properties are only considered if the Style property is csPickListe.

In this case, the cell will contain an in situ editor in the form of a scroll list.

The items displayed in the scroll list will be the strings of the PickItems object.

For example:

**Figure 24**

If the data to be displayed in the scroll list come from the database, the PickItems and PickKeys properties must be modified for execution.

To do so, use the sdColAddKey, sdColAddList, sdColClearKeys and sdColClearList functions.

For example:

```

Pload          B
D              PI
DRow           s          10  0
* Load the pick list from themes
C              callp      sdColClearKeys(f1:'sf11':
C              'NAMETHHEME')
C              callp      sdColClearList(f1:' sf11':
C              'NAMETHHEME')
C      *LOVAL      setll    SDDMTHM
C              READ      SDDMTHM          55
C      *ON         DOWNE   *IN55
C              callp      sdColAddKey(f1:' sf11':'NAMETHHEME':
C              sdINTTOSTR(T_IDTHEME))
C              callp      sdColAddList(f1:' sf11':'NAMETHHEME'
C              %trim(T_NAMETHHEME))
C              READ      SDDMTHM          55
C              enddo
P              E

```

csDate

The user can enter a date directly or click on a button to display a calendar.



Figure 25

csTime

The user can enter a time directly or use the buttons.



Figure 26

csBoolean

Tick boxes appear in the cells.

To modify these cells by programming, use the `columnField.PropBoolean.valueTrue` and `columnField.PropBoolean.valueFalse` values.

csImage

An image is displayed in the cell. The image is taken from the `imageList` indicated by the `image` property of the CSFL component. To display image no. `x`, enter the value `x` in the cell.

ColumnField.DataType

Depending on the `DataType` value and style, the `propAlpha`, `propBoolean`, `propNumeric`, `propDate` and `propTime` properties are taken into account.

For example: if the `DataType` property is numeric, then the `propNumeric` property is taken into account.

Tabulation

When the user presses tabulation in a CSFL component, the cursor moves from column to column, then to the next line.

If the `TabStop` property is set to false, the indicator does not go into this column.

Sorting

Clicking on the header of a column enables sorting of the sub-file according to the column. The `TypeSort` property enables indication of whether or not the sort should take type case into account.

A sort can be run by programming using the `sdSort` function.

Prototype:

```
d sdSort          pr
d pform           5u 0 const
d SFL             30    varying value
d Column1         30    varying value
d Order1          10i 0 value
d Column2         30    varying value options(*nopass)
```

```
d Order2          10i 0 value          options(*nopass)
d Column3         30    varying value options(*nopass)
d Order3          10i 0 value          options(*nopass)
```

Parameters:

SFL: Name of the grid to be sorted

Column1: name of the sort column

Order1: Sort direction: 1= increasing, 2 = decreasing

The following parameters are optional and enable a secondary sort.

For example:

```
C          callp      sdSort(F1:'sfl1':'Col1':1)
```

The sortable property of a column defines whether or not a column is sorted automatically when the user clicks the column.

By setting Sortable to false and using the onHeaderClick event and the sdSort function, you can obtain a personalised sort (e.g. with secondary sorting).

Example (local):

```
procedure sfl1_OnHeaderClick (Sender: TObject; ColName: String; Row:
Integer);
begin
  if(colName = 'DATE')or(colName = 'HOUR')then
  begin
    _1.sfl1.sort('DATE',2,'HOUR',2,'',1);
  end;
end;
```

Printouts

For formatted printing, use the report palette components.

However, you can print out the content of a CSFL component with a minimum of code using the sdPrint function.

```
C          callp      sdPrint(F1:'SFL1')
```

The PrintParameters object property enables print configuration.

Each column has a printable property to indicate the columns to be printed.

If the `printParameters.showDialog` property is true, a dialogue box enables the user to modify the print parameters.

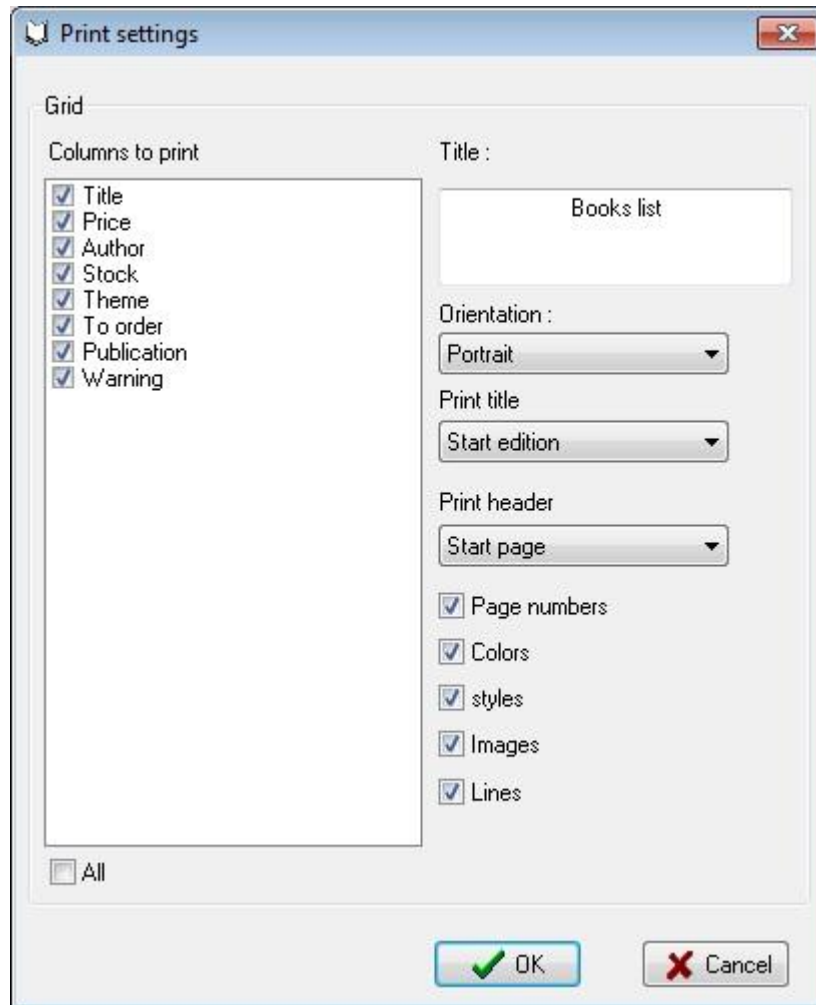
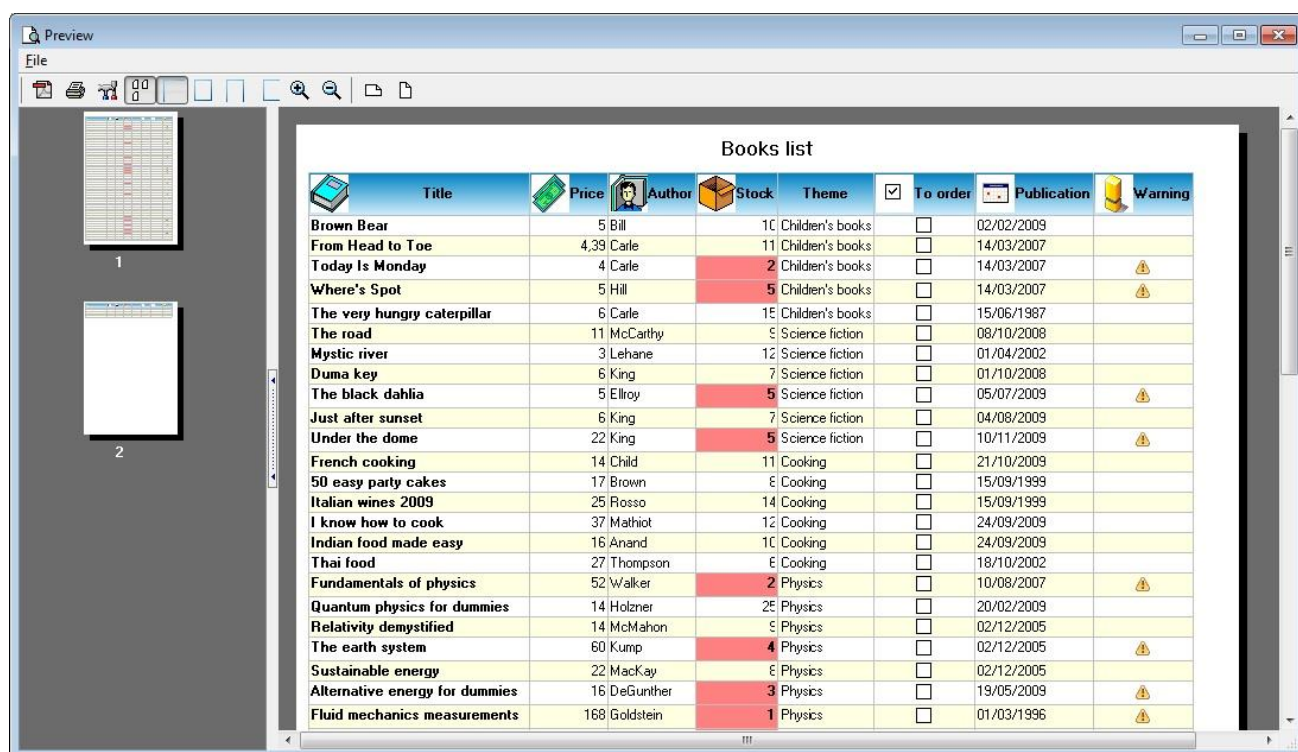


Figure 27

Finally, the `PrintParameters.UserCanChanges` property enables the developer to lock the items of this dialogue box.

A print preview can be obtained using the `sdPreview` functions.



Title	Price	Author	Stock	Theme	To order	Publication	Warning
Brown Bear	5	Bill	10	Children's books	<input type="checkbox"/>	02/02/2009	
From Head to Toe	4.39	Carle	11	Children's books	<input type="checkbox"/>	14/03/2007	
Today Is Monday	4	Carle	2	Children's books	<input type="checkbox"/>	14/03/2007	
Where's Spot	5	Hill	5	Children's books	<input type="checkbox"/>	14/03/2007	
The very hungry caterpillar	6	Carle	15	Children's books	<input type="checkbox"/>	15/06/1987	
The road	11	McCarthy	5	Science fiction	<input type="checkbox"/>	08/10/2008	
Mystic river	3	Lehane	12	Science fiction	<input type="checkbox"/>	01/04/2002	
Duma key	6	King	7	Science fiction	<input type="checkbox"/>	01/10/2008	
The black dahlia	5	Elroy	5	Science fiction	<input type="checkbox"/>	05/07/2009	
Just after sunset	6	King	7	Science fiction	<input type="checkbox"/>	04/08/2009	
Under the dome	22	King	5	Science fiction	<input type="checkbox"/>	10/11/2009	
French cooking	14	Child	11	Cooking	<input type="checkbox"/>	21/10/2009	
50 easy party cakes	17	Brown	8	Cooking	<input type="checkbox"/>	15/09/1999	
Italian wines 2009	25	Rosso	14	Cooking	<input type="checkbox"/>	15/09/1999	
I know how to cook	37	Mathiot	12	Cooking	<input type="checkbox"/>	24/09/2009	
Indian food made easy	16	Anand	10	Cooking	<input type="checkbox"/>	24/09/2009	
Thai food	27	Thompson	6	Cooking	<input type="checkbox"/>	18/10/2002	
Fundamentals of physics	52	Walker	2	Physics	<input type="checkbox"/>	10/08/2007	
Quantum physics for dummies	14	Holzer	25	Physics	<input type="checkbox"/>	20/02/2009	
Relativity demystified	14	McMahon	5	Physics	<input type="checkbox"/>	02/12/2005	
The earth system	60	Kump	4	Physics	<input type="checkbox"/>	02/12/2005	
Sustainable energy	22	MacKay	8	Physics	<input type="checkbox"/>	02/12/2005	
Alternative energy for dummies	16	DeGunther	3	Physics	<input type="checkbox"/>	19/05/2009	
Fluid mechanics measurements	168	Goldstein	1	Physics	<input type="checkbox"/>	01/03/1996	

Figure 28

Exports

To export the content of a CSFL component, use the `sdExport` function.

There are three types of export possible, depending on the `TypeExport` property.

<code>exBin</code>	Data are exported in binary format, compatible with Excel. Excel does not have to be installed on the workstation
<code>exCom</code>	Data are exported in Excel format by controlling excel in com. Excel must be installed on the workstation.
<code>exCsv</code>	Data are exported in CSV format

Multiselection

By adding the `goMultiselect` elt to the options property, the user can select several lines. By holding down the CTRL key, several non-adjacent lines can be selected. By holding down the SHFT key, several adjacent lines can be selected.

The user can deselect lines by clicking again on the line.

Images

To display an image in a sub-file, the image must first be placed in an imagelist and this imagelist must be assigned to the `csfl images` property.

Then, you have two options:

1) Create an image type column (column style property = csImage)

To display image number x in the imagelist, enter the value x in the cell. (Enter -1 to display no images).

2) Place the image in any cell (along with text)

To do so, create an element in the exAttributes collection and assign this element to the cell using the sdFormatCell function.

Display attributes

The display attributes (font, font colour, vertical alignment, horizontal alignment, image, transparent image, stretched image, etc.) can be modified via Tattribute type objects.

TabStop	True
Title	(TSFLColumnTitle)
Caption	
PopupMenu	
TitleAttribute	(TTitleAttribute)
Alignment	saNoModify
BgColor	clBtnFace
FontColor	clNone
FontName	
FontName	
FontSize	0
FontSizeP	0
FontStyle	[]
Gradient	(TSflGradient)
ImageIndex	-1
Images	
ImageStretch	False
Layout	slNoModify
ReadOnly	False
SearchAsU	True
SortedIndex	clNavy
Transparent	clNone
WordWrap	False
Visible	True

Figure 29

Several elements have Tattribute type objects.

The attribute applied to a cell abides by the following priority rules (highest priority to lowest priority):

Attributes due to sdFormatCell (ExAttributes)

Attributes due to the column (ColumnAttribute, Title.TitleAttribute)

Attributes due to the line parity (Attributes.Even, Attributes.Odd)

Others

Example:

The background colour of a cell is that of the CSubFile Color property.

Then, for odd lines, if the odd.BgColor property is different from cINone, the cell takes on the colour of the odd.bgColor property.

Then, if the ColumnAttribute.BgColor property is different from CInNone, the background colour of the cell will be the colour of the ColumnAttribute.BgColor property.

Then, by programming, you can modify the background colour of a cell using the SpecialsAttributes property.

Chapitre 14. CDateEdit

Tab sheet: Additional

This component enables date entry.

To modify or query the date value shown in the edit zone, use the `sdSetDate` and `sdGetDate` functions or use the `DateIso` property.

A button is included in the entry field.

When the user clicks on the field, a window opens with a `CCalendar` component.

To modify the properties of the pop-up calendar, use the properties whose names begin with `Popup` (`PopupAlign`, `PopupAutoClose`, `PopupBackGroundImg`, etc).

The important properties of `CDateEdit` are:

DirectInput: determines whether the user can enter a date directly or if the calendar has to be used.

PopupAutoClose: determines whether or not the calendar closes automatically once the user has selected a date.

CheckOnExit: determines whether the user is authorised to quit the entry field if the date entered is incorrect.

BlankIfNull: determines whether or not the component displays a blank text if the date value is 00/00/0000

ValidDates:

The `ValidDates` collection enables addition of a set of values to be considered valid when the `CheckOnExit` property is true. It is thus possible to authorise '00/00/0000' or ' / / '.

ValidDate: This must not be confused with `ValidDates`. It is a read-only Boolean property to know whether or not the date entered in the component is a valid date.

For example:

```
D mydate          s          D
C                monitor
C                eval      mydate=sdGetDate (F1: 'DateEdit1')
C                on-error
C                eval      Message=Message+'Invalid date'+
C                        + x'0d'
C                endmon
```

Chapitre 15. CCalendar

Tab sheet: Additional

The CCalendar component enables display of a calendar.

This calendar represents a date. This date is ringed in red on the calendar.

To modify or query the data value shown on the calendar, use the `sdSetDate` and `sdgetDate` functions or use the `DateIso` property.

You can modify the calendar's background image using the `BackGroundImage` property.

If you want to restore the default image, delete the image you added.

If you do not want a background image, remove the `ocBitmap` value from the `Options` property.

Chapitre 16. CMaskEdit

Tab sheet: Additional

CMaskEdit is a component that enables a text entry field to be proposed to users.
User entry can be limited to certain characters or a specific format.

User entry is controlled by the EditMask property.

EditMask

The EditMask property is a string type property (character string).
EditMask is a string comprising three fields separated by a semi-colon.

The first part of the mask is the mask itself.

The second part contains the character that determines whether or not the literal characters of the mask are included in the data.

The third part of the mask indicates the character used to materialise the characters to be entered in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in the mask
!	If an ! appears in the mask, the optional characters are shown in the text as spaces at the start. If there is no !, the optional characters are shown in the text as spaces at the end.
>	If this character appears in the mask, all the characters after it are in upper case until the end of the mask or until the < character.
<	If this character appears in the mask, all the characters after it are in lower case until the end of the mask or until the > character.
<>	If these two characters appear together in a mask, no upper case/lower case check is made and the data are formatted according to the upper and lower case characters entered by the user.
\	The character after this character is a literal character. Use this character if you want a special mask character to be a literal character included in the data.
L	This character requires an alphabetical character in this position only. For France, these are A-Z and a-z.
l	This character authorises only alphabetical characters in this position, but there is no obligation.
A	This character requires an alphanumeric character in this position. For France, these are A-Z, a-z, 0-9.
a	This character authorises an alphanumeric character in this position, but there is no obligation.
C	This character requires an arbitrary character in this position.
c	This character requires an arbitrary character in this position but there is no obligation.
0	This character requires a numeric character in this position.
9	This character authorises a numeric character in this position but there is no obligation.
#	This character authorises a numeric character or the plus or minus sign in this position, but there is no obligation.
:	This character enables separation of hours, minutes and seconds. If a different character is defined by the regional parameters of your system's Control Panel for the hour, minutes and seconds separation character, this character will be used instead of :.
/	This character enables separation of months, days and years for dates. If a different character is defined by the regional parameters of your system's Control Panel, this character is used instead of /.
;	This character separates the three fields of the mask.
_	The _ character inserts spaces automatically in the text. If the user enters spaces in the field, the cursors skips the _ characters.

Any character that is not included in this table may appear in the mask as a literal character. Literal characters must have an exact correspondence in the entry of control data. They are inserted automatically and the cursor skips them during entry. Special characters can also appear as literal characters if they are preceded by an anti-slash (\).

The second part of the mask only contains a single character indicating whether or not the literal characters are to be included in the mask in the control's Text property. For example, the mask for a telephone number with a regional code could be the following string:

(000)_000-0000;0;*

A 0 in the second part of the mask indicates that the literal characters should not be included in the Text property; any other character means that they should be included. It is possible to change the character indicating inclusion of the literal characters in the EditMask property editor or by programming by modifying the MaskNoSave type constant.

The third part of the mask indicates the character appearing in the entry control to materialise the blanks (the characters to be entered). By default, the same character as for the literal spaces is used. These two characters therefore appear the same in the entry window. However, if a user enters data in a masked entry control, the cursor successively selects each blank character and skips the space characters.

Specific editor

Designer enables modification of the EditMask property using a specific editor. Click on the ellipsis button in the property inspector.



Figure 30

In the editor, the first zone enables entry of the value of the EditMask property. The second zone enables testing of the mask as the user sees it. The third enables viewing of the result of the text property.

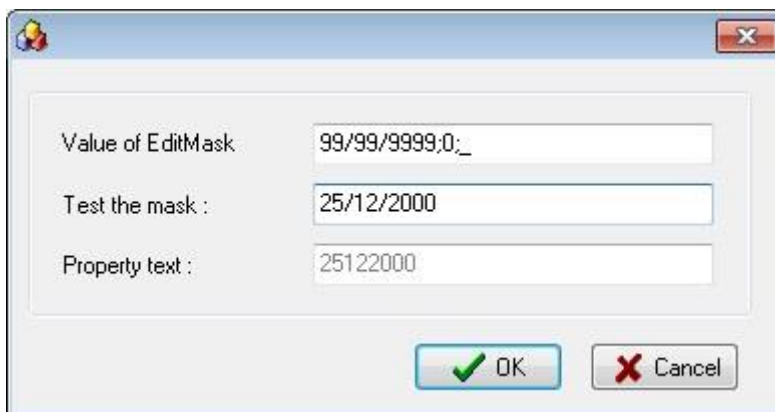


Figure 31

Chapitre 17. CImage

Tab sheet: Additional

The CImage component enables display of an image.

Design Time

To change the image in the CImage, use the Picture property.

An editor enables selection of an image from your disk.

The following formats can be used: bmp, jpg and ico.

Run Time

Ifs file to CImage

To change the image in a CImage component during execution, use the sdSetImg function.

This function has to know the form on which the CImage component can be found, the name of the component, the property receiving the image ('picture'), the type of image and its file path on the ifs.

This function returns a value:

0: All OK

-1: The specified file does not exist or you are not authorised to use it. The image was not downloaded.

-2: The file sent does not appear to be of the type indicated, or the component cannot load this type of image.

(e.g.: you send a jpg image and say that it is a bmp)

CImage to ifs file

To retrieve the content of an image, use the sdGetImg function.

This function has to know:

The form on which the component can be found.

The name of the component.

The property containing the image ('picture')

Where on the ifs you want to save the image.

The sdGetImg function sends back a value:

0: All OK

-1: The image in the component is empty and therefore the file was not created/modified on the ifs.

-2: The ifs file name is incorrect or you do not have the required rights.

Emptying a Cimage

To empty an image, use the following expression:

```
sdSet(F1:'Image1':'Picture.Graphic': 'Null')
```

Loading an image from the user disk

If you want the user to be able to modify the content of CImage by selecting an image from his/her disk, use the sdSelectFile and sdLoadFromFile functions.

For example:

```
PSelectImg      B
D               PI
D PForm         5u 0 value
D path          s      1000    varying
C               if      sdSelectFile('*.jpg |*.jpg':
C                       Path:seOpenPicture)
C               callp    sdLoadFromFile(PForm:
C                       'Image1.picture':
C                       path)
C               endif
P               E
```

You can then retrieve the content of the image using the sdGetImg function.

Zoomable

Specifies whether or not the user can zoom on part of the image.

To enable a zoom, the Stretch property must be true.

The user then selects the part to be zoomed using the mouse.

Note:

Only bitmap, jpeg and gif type images can be zoomed.

Chapitre 18. CSplitter

Tab sheet: Additional

The CSplitter component requires no further code.

On a form, add a separator between two aligned controls (Align property = alLeft or alRight, etc.) to enable users to resize the controls during execution. The separator is between an aligned control on an edge of the form and the controls filling the rest of the client zone. Align the separator in the same manner as the control anchored to the edge of the form. When the user moves the separator, the anchored control is resized. This changes the form's client zone and the controls filling the rest of the client zone are resized accordingly.

Use each control on the form as a separate panel. After positioning each panel, place a separator with the same alignment to authorise resizing of the panel. The last panel placed on the form must be aligned with the client zone to enable automatic resizing to fill the remaining space after resizing of all the other panels.

Day	Expense code	Project	Description	Mileage	Amount euros	Payment	Chargeable
1	Lunch	2188	Cafeteria crescendo	0	8,86	Employee	<input type="checkbox"/>
1	Lunch	2268	Resto midi semaine 47	0	54,87	Employee	<input checked="" type="checkbox"/>
5	Mileage	2199	Aller	160	72,99	Employee	<input type="checkbox"/>
7	Lunch	2567	Flunch resto mission	0	7,08	Employee	<input checked="" type="checkbox"/>
15	Air/Road	2345	Aller train	0	34,93	Employee	<input type="checkbox"/>
18	Hospit.	2191	Le Connetable AJEF	0	284,65	Card	<input type="checkbox"/>
18	Mileage	2163	Kms retour Aix	300	136,87	Employee	<input type="checkbox"/>
20	Lunch	2181	L'Atomixeur	0	11,34	Employee	<input type="checkbox"/>
21	Lunch	2180	Restaurant du Cerf	0	2,83	Employee	<input type="checkbox"/>
21	Hospit.	2178	Resto Chaumiere Voiron	0	68,59	Company	<input type="checkbox"/>
24	Lunch	2182	Frais dejeuner	0	10,41	Employee	<input type="checkbox"/>
24	Toll	2339	AREA péage	0	7,92	Employee	<input checked="" type="checkbox"/>
25	Toll	2191	Visite client cpta SAARI	0	1,64	Card	<input type="checkbox"/>

Summary of the month			
Company pre/post paid total:	68,59	Amount total:	702,98
Credit card total:	286,29	Amount to reimburse:	348,10
Mileage Total:	460		

Figure 32

Direction

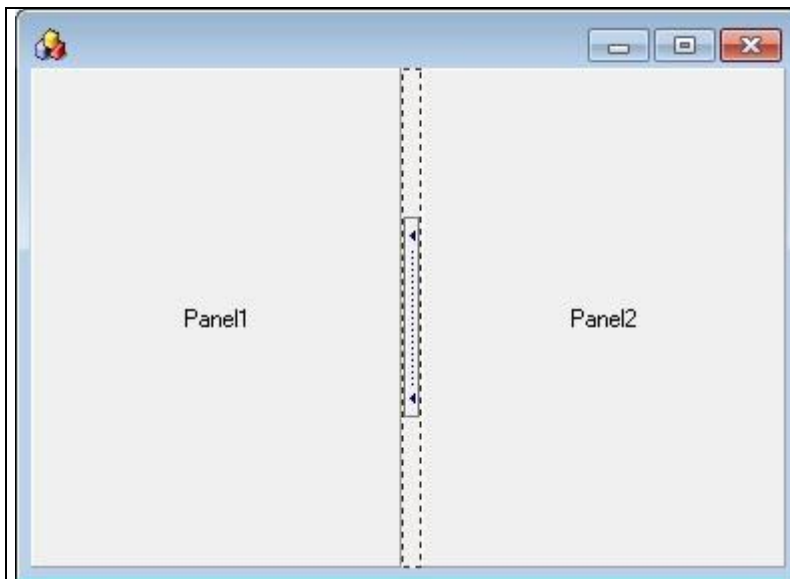


Figure 33

Panel2 in alClient, Panel and splitter in alLeft

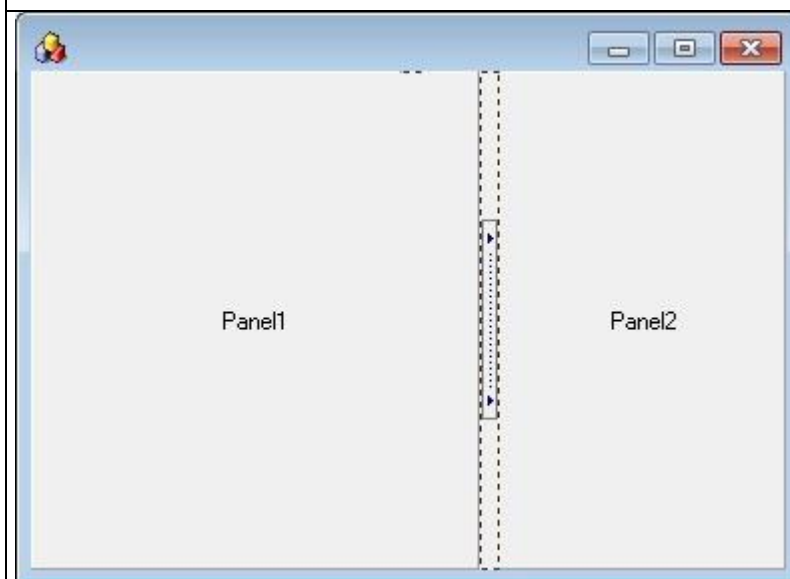


Figure 34

Panel1 in alClient, Panel2 and splitter in alRight

MinSize:

Indicates the minimum size of the panels on either side of the separator, in pixels.

MinSize enables definition of a minimum size that the separator must leave when resizing the adjacent control. For example, if the Align property is alLeft or alRight, the separator may not shrink the zones to the right or left to less than MinSize pixels. If the Align property is alTop or

alBottom, the separator may not shrink the zones above or below the separator to less than MinSize pixels. The default value of MinSize is 30.

Note: Always specify a MinSize value less than half the client width of the parent. If MinSize is equal to half the client width of the separator's parent, the separator cannot be moved since to do so, it would have to reduce one of the panels to less than MinSize pixels.

Chapitre 19. CTreeView

Tab sheet: Win32

The CTreeView component enables display of data in a hierarchical tree. Each node of the Treeview has a Text property and a Key property.

Editing in Design Time

In Designer, use the property inspector, Items property, to edit the nodes. You can also right click on the treeView and click on the item of the “editor” menu.

You can also double click on the CTreeView component.

The following screen is displayed:

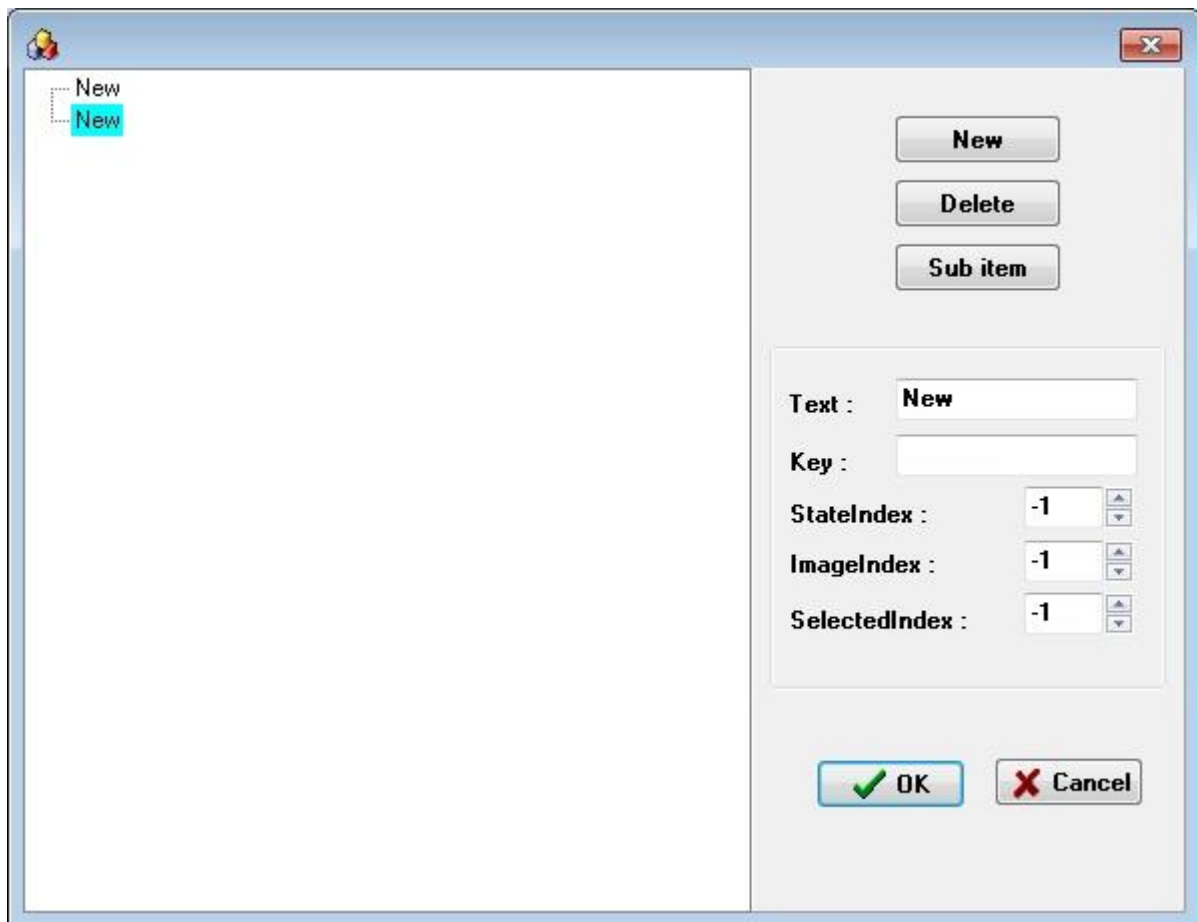


Figure 35

Unlike the other specific editors, this window is displayed in modal.

The modifications are only transferred to the CTreeView when you click OK.

Editing in Run Time

If the data to be displayed in the TreeView come from a database, use the sdCollecClear, sdAddNode, sdAddNode2 and sdEndUpd functions.

sdCollecClear deletes all items.

sdAddNode adds a node.

sdAddNode2 also adds a node. This function is generally quicker and easier to use than sdAddNode1. The sdAddNode2 function was created after the sdAddNode function to improve performance.

The sdAddNode function takes the number of the node to which it will be attached as a third parameter (-1 for a node added to the root).

| Note: the first node is the zero index node.

```
d sdAddNode      pr
d pform          5u 0 value
d Treeview       30   varying value
d position       10i 0 value
d Caption        256   varying value
d Key            256   varying value options(*nopass)
d ImageIndex     10i 0  value options(*nopass)
d StateIndex     10i 0  value options(*nopass)
d SelectedIndex  10i 0  value options(*nopass)
```

```
d sdAddNode2     pr
d pform          5u 0 value
d Treeview       30   varying value
d level          10i 0 value
d Caption        256   varying value
d Key            256   varying value options(*nopass)
d ImageIndex     10i 0  value options(*nopass)
d StateIndex     10i 0  value options(*nopass)
d SelectedIndex  10i 0  value options(*nopass)
```

Example with sdAddNode:

```
PLoadTree      b
p              PI
DiCtry         s          10  0 inz(0)
DiSta          s          10  0 inz(0)
DCpt           s          10  0 inz(0)
C      KeySta  klist
C              kfld          IDCTRY
```



```

C          kfld          IDSTA
C          callp      sdBeginUpd(F1:'treeView1':'items')
C          callp      sdCollecClear(F1:'treeView1':'items')
C      *loval      setll      Ctry
C          read          Ctry          55
C          dow          *in55 = *off
C          callp      sdAddNode(F1:'treeview1':
C          -1:%trim(CTRYNAME) :'' :0)
C          eval          iCtry = cpt
C          eval          cpt = cpt + 1
C      IDCTRY      setll      States
C      IDCTRY      reade      States          56
C          dow          *in56 = *off
C          callp      sdAddNode(F1:'treeview1':
C          iCtry:%trim(STATENAME))
C          eval          iSta = cpt
C          eval          cpt = cpt + 1
C      keySta      setll      CITIES
C      keySta      reade      CITIES          57
C          dow          *in57 = *off
C          callp      sdAddNode(F1:'treeview1':
C          iSta:%trim(CITYNAME))
C          eval          cpt = cpt + 1
C      keySta      reade      CITIES          57
C          enddo
C      idCtry      reade      States          56
C          enddo
C          read          Ctry          55
C          enddo
C          callp      sdEndUpd(F1:'treeView1':'items')
C
P          E

```

Example with sdAddNode 2

```

pLoadTree      b
D          PI
C      KeySta      klist
C          kfld          IDCTRY
C          kfld          IDSTA
C      *facultatif
C          callp      sdBeginUpd(F1:'treeView1':'items')
C      *Reset de tous les noeuds
C          callp      sdCollecClear(F1:'treeView1':'items')
C      *loval      setll      Ctry
C          read          Ctry          55
C          dow          *in55 = *off

```

```

C      callp      sdAddNode2 (F1: 'treeview1':
C      0: %trim(CTRYNAME) )
C      IDCTRY     settl      States
C      IDCTRY     reade      States      56
C      dow        *in56 = *off
C      callp      sdAddNode2 (F1: 'treeview1':
C      1: %trim(STATENAME) )
C      keySta     settl      CITIES
C      keySta     reade      CITIES      57
C      dow        *in57 = *off
C      callp      sdAddNode2 (F1: 'treeview1':
C      2: %trim(CITYNAME) )
C      keySta     reade      CITIES      57
C      enddo
C      idCtry     reade      States      56
C      enddo
C      read       Ctry      55
C      enddo
C      callp      sdEndUpd (F1: 'treeView1': 'items')
pLoadTree      e

```

Index

Regardless of their level in the hierarchy, the nodes have an index as shown in Figure 36.

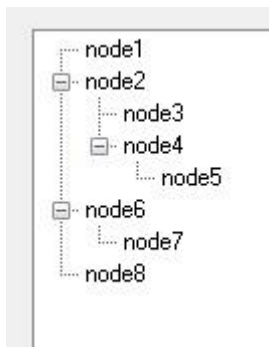


Figure 36

To modify the properties of a node in run time, use the standard `sdSet` and `sdGet` type functions.

To modify the Key property of node number 5, for example, write:

```
sdsetString(f1 : 'treeview1' : 'items(5).key' : 'Key value')
```

Note:

If a property of a node is to be modified and this property is not one of the parameters, you can use the Current property of the Treeview component. It points to the last node created.

```
sdsetString(f1 : 'treeview1' : 'current.Key' : 'Key value')
```

Querying the selected items

The CTreeView component has FirstSelected and NextSelected properties that enable viewing of the list of nodes selected by the user.

FirstSelected and NextSelected return -1 if there are no nodes selected.

It is then possible to query the value of the Key property for these items.

The instruction in the paragraph above is therefore modified because the number of the node is in a field that we will call i.

The new instruction will be:

```
sdsetString(f1 : 'treeview1' : sditem(i) + '.key' : 'Key value')
```

Note 1:

The sdItem() function simply returns a character string.

sditem(0) returns 'items(0)'

Note 2:

(see the MultiSelect and MultiSelectStyle properties for more details).

For example:

```
C      eval      i=sdgetInt(F1: 'TreeView1': 'FirstSelected')
C      dow      i <> -1
C      eval      Code= sdGet(F1: 'TreeView1': sdItem(i) + '.Key')
```

...Processing

```
C      eval      i=sdgetInt(F1: 'TreeView1': 'NextSelected')
C      enddo
```

Tick boxes

Tick boxes can be added for each item of the Treeview.

To do so, modify the component's CheckBoxes property.

The state of a node depends on the value of its StateIndex property.

1 : unchecked

2: Checked

3: gray checked

When the user ticks or unticks a box, the parent and child nodes are automatically modified. For example, if you tick a node, all the child nodes will be ticked and the parent node is ticked if all the brother nodes are ticked and grey if one of the brother nodes is not ticked.

Caution: This check is not active when you load a Treeview by code.
You can use the constants nsUnchecked, nsChecked and nsGrayed

Local copy

The CTreeView component authorises use of sdGetList type functions.
The sdGetList function returns all the information on the CTreeView component locally.
These functions enable optimisation if all nodes have to be read again.

Drag and Drop example

Description:

Authorisation or not of drag and drop in a Treeview component.

Code

```
procedure TreeView1_OnDragOver (Sender: TObject; Source: TObject; X:
Integer; Y: Integer; State: TDragState; var Accept: Boolean);
var
  myNode:Tnode;
begin
  myNode := FChart.treeview1.getnodeat(X,Y);
  Accept := (myNode <> nil) and (myNode.level = 0) and
    (myNode.parent <> MyNode.Parent);
end;
```

```
pTreeView1_OnDragDrop...
p          B
d          PI
d PevtInf          *   const options(*nopass)
d parm          ds          based(pevtinf)
d Win          5u 0
d evt          48a
d name          100a   varying
```

```
d  X                                10i 0
d  Y                                10i 0
DNode1          s                   10i 0
DNode2          s                   10i 0
C               eval                node1=sdgetInt(F1:'TreeView1':
C                                   'FirstSelected')
C               eval                node2=sdNodeAt(F1:'treeView1':X:Y)
C               if                  Node1 = -1  or Node1=Node2
C               return
C               endif
C               if                  sdMsgDlg(boxConfirm:btnYes + btnNo:
C                                   'Move this node?')=btnYes
C               callp                sdMoveNode(F1:'TreeView1':Node1:Node
C                                   'naAddChild')
C               endif
p               E
```

Chapitre 20. CTimer

Tab sheet: System

You cannot decide the moment to execute the code on the server. The code on the server is only executed when required by a client (i.e. upon an event). To produce an application that performs a cyclical task (such as monitoring the arrival of incoming messages), you have to use the CTimer component. This component cannot be seen on screen and it triggers a cyclical event.

Chapitre 21. CTrayIcon

Tab sheet: System

The CTrayIcon component enables an application icon to be hidden in the task bar and make it visible in the notification zone.

The onclick event is triggered when the user clicks the icon in the notification zone. This enables the sdTrayShowMainForm function to be called, which causes the application icon to reappear in the task bar and displays the main window.

To return the application to the notification zone, use the sdTrayHideMainForm function.

If you assign a popup menu to the CTrayIcon component, it will be displayed when the user right clicks on the icon in the notification zone.

To close the application, call sdClose on the main window as for any other SilverDev application.

If you want the application to pass into the notification zone when the user clicks on the cross, you will have to write an event manager for the onclick event. You will also have to add a possibility for the user to close the application via a popup menu for example.

```
pTiQuit_OnClick    B
d                  PI
d PevtInf           *    const options(*nopass)
C                  Eval    CanQuit = *On
C                  callp    sdClose(F1)
p                  E
```

```
pOnClose           B
d                  PI
d PevtInf           *    const options(*nopass)
DParams            ds    based(PevtInf)
D Win              5u 0
D Event            48
D Action           10i 0 overlay(ActionA)
*   Tis parameter specifies the action to take :
*   0 Do nothing
*   1 Hide the window
*   2 Free the windows (Don't forget to set Formhandle to 0!)
*   3 Minimize the window
C                  If      CanQuit
C                  eval    Action=1
C                  else
C                  eval    Action=0
```

```
C          callp      sdTrayHideMainForm(F1: 'TrayIcon1')
C          endif
p          E
```


Chapitre 22. CNavBar

Tab sheet : Standard

The CNavBar component allows to displays groups of links. Groups can ben expanded or collapsed.

Building in design

To build groups and links in the CNavBar component, double click on the component, a specific editor opens.

Create a group by using the + button in the groups part.

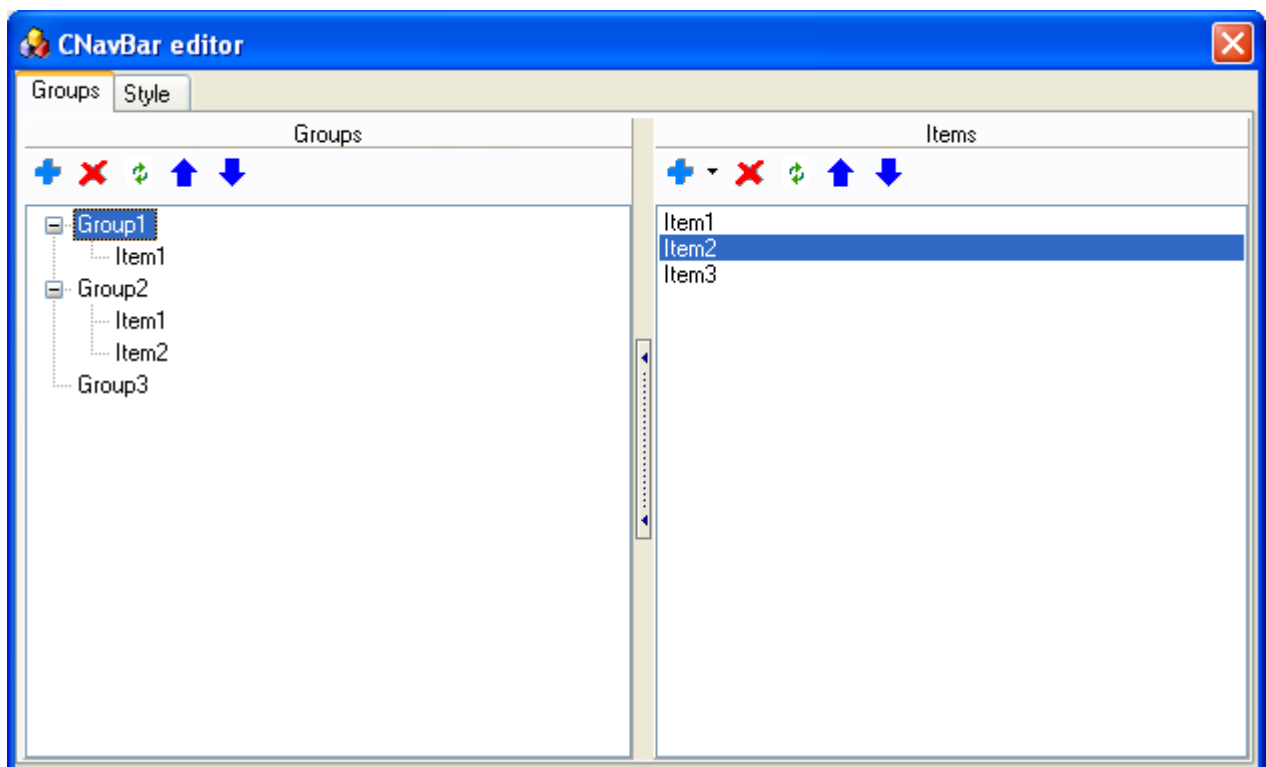


Figure 37

Create an item by using the + button in the items part.

Drag and drop an item from the right part to a group in the left part to associate the item to the group.

An item can be inserted several times in a group and can be inserted in several groups.

To change properties of an item or of a group, select the item or the group in the CNavBar editor, and change the properties in the properties inspector.

Add a control in a group

To insert a control in a group of the CNavBar component, assign the OptionsGroupControl.UseControl group property to true.

A TdxNavBarGroupControl is displayed.

Use the "tools/components tree" menu item to display the components tree.

Use the components tree to move any visual component in the group.

OnLinkClick

Use the CNavBar OnLinkClick event to handle the click on an item.

The event has a parameter that indicates the item name the click is on.

Note : You can also assign an action component to the item.

Building groups at run time

Example of building groups and items at run time :

c	callp	sdNBAddGroup(F1: 'NavBar1': 'group1')
c	callp	sdNBAddItem(F1: 'NavBar1': 'item1')
c	callp	sdNBAddItem(F1: 'NavBar1': 'item2')
c	callp	sdNBAddSeparator(F1: 'NavBar1': 'sep1')
c	callp	sdNBAddLink(F1: 'Group1': 'Item1')
c	callp	sdNBAddLink(F1: 'group1': 'sep1')
c	callp	sdNBAddLink(F1: 'group1': 'item2')

Chapitre 23. CScheduler

Tab sheet: Scheduler

The CScheduler component must be used in conjunction with a CStorage component. The CScheduler component displays the events stored in the CStorage component.

To associate a CScheduler component and a CStorage component, use the Storage property of the CScheduler component.

Adding events

To delete all the events, use the sdClear function.

To add an event in the CStorage component, use the sdNewEvent function.

The sdBeginUpdate and sdEndUpdate functions must be called respectively before and after adding events.

```
c          callp      sdBeginUpd(F1: 'Storage1')
c          callp      sdClear(F1: 'Storage1')
c          callp      sdNewEvent(F1: 'Storage1':A_DateStr:
c                      A_TimeStr:A_DateEnd:A_TimeEnd)
c          callp      sdEndUpd(F1: 'Storage1')
```

Event properties

Once you have added an event, it is possible to refer to this event with the current property of the CStorage component.

```
c          callp      sdSetString(F1: 'Storage1':
c                      'Current.Caption': 'example')
```

The properties of interest for an event are:

DateStart	integer	Event start date
DateEnd	integer	Event end date
TimeStart	integer	Event start time

TimeEnd	integer	Event end time
LabelColor	TColor	Event background colour
Caption	String	Event title
Message	String	Event text
Location	String	Location
Values	Variable	Indexed property. The number depends on the number of fields in CStorage's customFields. The first element has the index 17.
State	Integer	From 0 to 3. Indicates availability. In the day view, the edges of the event change appearance according to the State property.
ResourceId	Variable	Event resource.

State values:

Value	Appearance	Meaning
0	White	Free
1	Blue and white	Temporary
2	Blue	Occupied
3	Purple	Move

Additional event values

As well as fixed properties, you can also add properties for each event. To do so, use the customFields property of the CStorage component. These properties can be accessed via the Field indexed property.

```
C          callp      sdSetString(F1: 'Storage1':
C          'Current.Field(0) ': 'example')
```

The first property in customFields has index number 0, the next has index number 1, and so on.

Use the first customFields property to insert values identifying the event, since the value of this property is transmitted in the scheduler's various events.

DateNavigator

The component displays a navigation calendar so that the user can move around the scheduler pages.

This calendar can show the days with events, according to the `DateNavigator.ShowDatesContainingEventsInBold` property.

Of course, if this property is true, the elements must be added with the `sdNewEvent` function throughout the period of the calendar.

Select a period programmatically.

```
D sdSelectDays      pr
D F1                5u 0 const
d Component         30    varying value
D DateStr           8  0 value
D DateEnd           8  0 value
D ViewDay           N    value
```

To find out the start and end dates of the calendar, query the `NavigatorDateStart` and `NavigatorDateEnd` variables of the `CScheduler` component.

Different views

The `CScheduler` component has different views. They are automatically selected according to the period chosen in the `DateTimeNavigator`.

You can deactivate the views, for example if you do not want the week view to be displayed, set the `viewWeek.CanShow` property to false.

Resources

To display several resources at the same time, add events to the `CStorage.Resources.Items` collection.

Use the event's `ResourceId` property to assign the event to this resource.

For example:

```
C      callp      sdCollecAdd(F1: 'Storage1':
C      'Resources.Items')
C      callp      sdSetString(F1: 'Storage1': 'resources.' +
C      'Items.Items(0).ResourceId: 'Res1')
C      callp      sdSetString(F1: 'Storage1':
C      'Current.ResourceId: 'Res1')
```

Internal forms and popup menus

The CScheduler component has internal forms and popup menus.

In most cases, you will prefer to use your own personalised forms and popup menus, rather than these elements. In this case, remember to modify the following properties:

EventOperations.DialogEditing
 EventPopupMenu.UseBuiltInPopupMenu
 ContextPopupMenu.UseBuiltInPopupMenu
 ViewDay.TimerPopupMenu.items

Event drag and drop

To move an event, change the EventOperations.moving property to true.

Intercept the OnAfterDragEvent event to reflect the modifications in the database.

The move can be cancelled via the Accept parameter.

New information about the event is found in the Datas, StrDate, StrTime, EndDate, EndTime parameters.

Datas contains the values inserted in the first element of customFields.

```
~/EVENT Scheduler1_OnAfterDragEvent
, *-----*
, * Description : *
, *-----*
D Parameters      ds          based(pevtinf)
D Win              5u 0
D Evt              48a
  * Can be changed:
D Accept           N
  * MAY NOT be changed:
D Datas            100      varying
D StrDate          10u 0
D StrTime          10u 0
D EndDate          10u 0
```

```
D EndTime          10u 0
*
```

For example:

```
c          eval      msg = 'Do you want to move this event?'+
c          X'0d25'+ 'New dates available: '+X'0d25'+
c          'De '+DspDate(StrDate)+' ' +
c          DspHour(StrTime) + ' à ' +
c          DspDate(EndDate)+' ' +DspHour(EndTime)
C          eval      Accept = sdMsgDlg(boxConfirm:
c          btnOK + btnCnl:msg) = btnok
c          if        not Accept
c          return
c          endif

c          eval      dsInfos = Datas
C          move      dsNoevt      A_noevt
C      A_noevt      chain      sddmevts
c          if        %found(sddmevts)
c          eval      A_DateStr= StrDate
c          eval      A_DateEnd = EndDate
c          eval      A_TimeStr = StrTime
c          eval      A_TimeEnd = EndTime
c          update    sddmevtsF
c          endif
```

The move can also be prevented in the onBeforeDragEvent event using the Allow parameter.

```
D Parameters      ds          based(pevtinf)
D Win              5u 0
D Evt              48a
,* Can be changed:
D Allow            N
,* MAY NOT be changed:
D Datas            100      varying
,*
```

Modifying the duration of an event

To enable modification of the duration of an event directly in the diary, change the EventOperation.sizing property to true.

Intercept the OnAfterSizingEvent event to reflect the modifications in the database.

The modification can be cancelled via the Accept parameter.

The new event information is found in the Datas, StrDate, StrTime, EndDate and EndTime parameters.

Datas contains the values in the first customFields element.

```
D Parameters      ds              based(pevtinf)
D Win              5u 0
D Evt              48a
  * Can be changed:
D Accept           N
  * MAY NOT be changed:
D Datas            100      varying
D StrDate          10u 0
D StrTime          10u 0
D EndDate          10u 0
D EndTime          10u 0
*
```

The move can also be prevented in the onBeforeSizingEvent event.

```
D Parameters      ds              based(pevtinf)
D Win              5u 0
D Evt              48a
, * Can be changed:
D Allow            N
, * MAY NOT be changed:
D Datas            100      varying
```

Deleting an event

To enable deletion of an event directly in the calendar, change the EventOperation.deleting property to true.

Intercept the OnBeforeDeleting event.

Datas contains the values in the first customFields element.

The Allow parameter enables cancellation of the operation.

```
D Parameters      ds              based(pevtinf)
D Win              5u 0
D Evt              48a
  * Can be changed:
```

```

D Allow                                N
  * MAY NOT be changed:
D Datas                                100    varying
  *
C                                     eval    Allow = sdMsgDlg(boxConfirm:
c                                     btnOK + btnCnl:
c                                     'Delete the event?') = btnok

```

Double-click an event

Use the Datas parameter to find the corresponding record in the database.

```

D Parameters    ds                    based(pevtinf)
D Win                                5u 0
D Evt                                48a
D Datas                                100    varying

```

Double-click the diary

If you want to display an event creation window for the date and time on which the double-click occurs, use the event parameters:

```

D Parameters    ds                    based(pevtinf)
D Win                                5u 0
D Evt                                48a
D Date                                10u 0
D Time                                10u 0
D Index                                10i 0

```

Detect a period change

The onPeriodChanged event enables detection of a change in the period displayed by the calendars. If the DateNavigator.ShowDatesContainingEventsInBold property is true, the database must be consulted again.

The onSelectionPeriodChanged event enables detection of a change in the period selected in the calendar. If the DateNavigator.ShowDatesContainingEventsInBold property is false, the database can be consulted again in this event to refresh the diary.

These two events have the following parameters:

D Parameters	ds	based(pevtinf)
D Win		5u 0
D Evt		48a
D Deb		10u 0
D end		10u 0

Images in an event

To add an image in an event, assign the eventImages property.

Add the information using the customFields property.

Use the onInitEventImages event locally, as illustrated below:

```
var
temp:variant;
begin
temp:=AEvent.Field[0];
if not VarIsNull(temp) and (temp='Y') then
begin
AImages.add(0);
end;
temp:=AEvent.Field[1];
if not VarIsNull(temp) and (temp='Y') then
begin
AImages.add(1);
end;
end;
```

Colour of an event

You can change the colour of an event after creation, as shown below:

C	callp	sdSetInt(F1: 'Storage1':
---	-------	--------------------------

sddmview program example

In silverdemo , the SDDMVIEW program provides an example of how to use the CSCHEDULER component.

Chapitre 24. CFontDialog, CColorDialog, COpenDialog, CSaveDialog, COpenPictureDialog, CSavePictureDialog

These components are used in the same manner.

Use the sdDialog function to display the dialogue box associated with these components.

For example:

```
DName          s          200    varying

C              if          sdDialog(f1:'fontdialog1')
C              eval        name=sdget(f1:'fontdialog1':'font.name')
C              endif
```

```
DCouleur       s          10i 0

C              if          sdDialog(f1:'ColorDialog1')
C              eval        Couleur= sdGetInt(f1:'ColorDialog1':'Coulo
C              endif
```

```
DFromPage      s          10i 0
DToPage        s          10i 0
Dtype          s          1    inz('1')
DprintRange    s          10i 0

C              if          sdDialog(F1:'printDialog1')
C              eval        printRange=sdGetInt(F1:'PrintDialog1':
C              'PrintRange')
C              if          PrintRange=2
C              eval        FromPage = sdGetInt(F1:'PrintDialog1':
C              'FromPage')
C              eval        ToPage = sdGetInt(F1:'PrintDialog1':
C              'ToPage')
C              endif
```

Chapitre 25. CLinkLabel

CLinkLabel is a component that enables inclusion of a link to a web page or sending of an e-mail.

The Link property contains all the characteristics specific to CLinkLabel in relation to CLabel.

Link.Color:

Represents the colour of the background when the mouse moves over it.

Link.Font:

Represents the font characteristics when the mouse moves over it.

Link.Url:

Represents the link to which the user will be directed if he/she clicks.

Note:

*To link to a web page, use an url of the following type: http :
//www.experia.com*

*To link to an e-mail page, use an url of the following type: mailto :
infos@exeria.com*

Comment:

The component opens the document specified in the link property, provided the onclick event is not used. If the onclick event is specified, no action is taken automatically.

Chapitre 26. CMapPoint

The CMapPoint component enables use of the Map Point mapping application. MapPoint must be installed on the PC to be able to use this component.

Chapitre 27. CChart

Tab sheet: Additional

The CChart component enables graphs to be drawn.

To add a chart in a Silverdev form, select chart component in graphics tabsheet of components palette.

Chapitre 28. TChartSeries

This component has a property called SeriesList.

This property is a list of graphs whose generic type is TChartSeries.

To add a graph, use the property inspector or double-click on the CChart component.

The editor is very similar to the collections editor, but when you add a graph, you have to specify its type.

Select the types of graph that you want to add (they appear in yellow). Click OK.

Here, we have chosen two graphs, a TPieSeries and a TLineSeries.

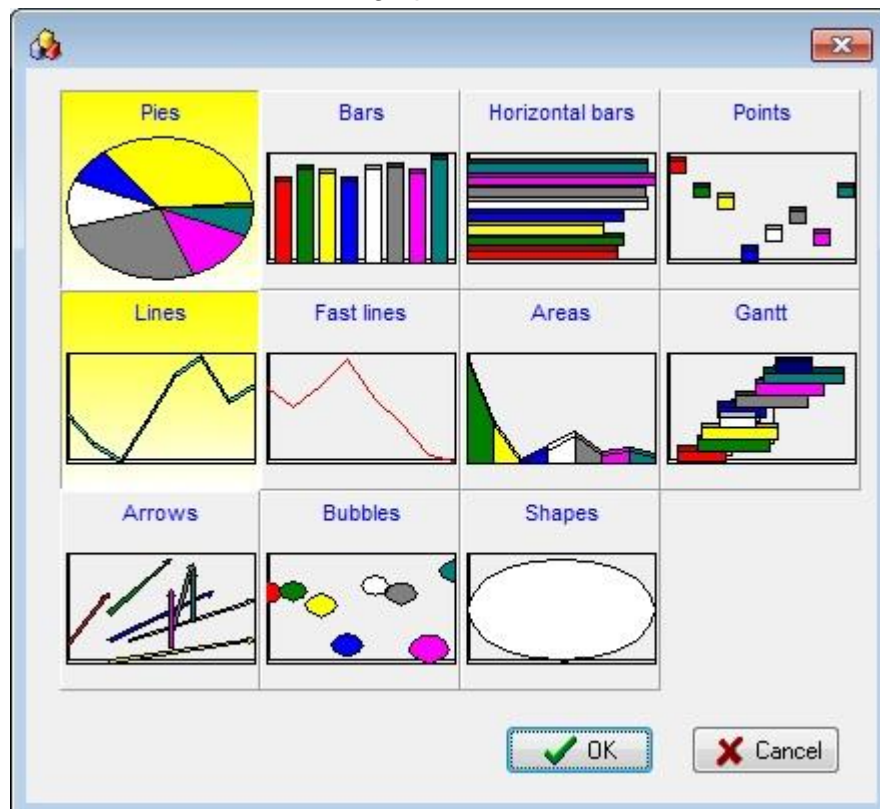


Figure 38

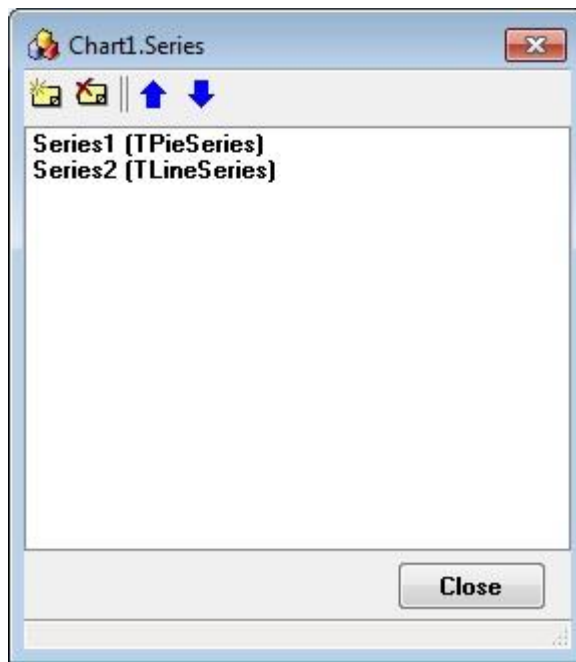


Figure 39

The two graphs can be deactivated at any time using the Active property of the two TLineSeries and TPieSeries objects.

In Designer, the graphs have random data to show the developer what his screen will look like.

The real data are added by programming using functions specific to each type of graph. The sdSeriesClear function deletes all the points of a graph.

In our example, we will use sdAddPie.

(See the help files for the functions specific to each type of graph).

For example:

```

C      callp      sdSeriesClear (F1: 'Series1')
C      *loval      settl      SDDMTHM      80
C      read      SDDMTHM      80
C      *in80      doweq      *off
C      Eval      cpt=0
C      T_IDTHEME      settl      SDDMBKS1      70
C      T_IDTHEME      reade      SDDMBKS1      70
C      *in70      doweq      *off
C      Eval      cpt=cpt+1
C      T_IDTHEME      reade      SDDMBKS1      70
C      enddo
C      callp      sdAddPie (F1: 'Series1':cpt:
C                  T_NOMTHEME)
C      read      SDDMTHM      80
C      enddo

```

Fonctions

Pour supprimer toutes les données dans une série, utilisez la fonction `sdSeriesClear`, quel que soit le type de serie.

Pour ajouter des données dans une série, utilisez la fonction correspondant selon le type de serie :

TPieSeries	sdAddPie
TBarSeries	sdAddBar
TLineSeries TAreaSeries TFastLineSeries TPointSeries	sdAddXY
TArrowSeries	sdAddArrow
TBubbleSeries	sdAddBubble
TGanttSeries	sdAddGantt

Exemple :

```

C      callp      sdSeriesClear(F1: 'Series1')
C      *loval     setll      SDDMTHM              80
C      read      SDDMTHM              80
C      *in80      doweq      *off
C      Eval      cpt=0
C  T_IDTHEME  setll  SDDMLIV1              70
C      T_IDTHEME  reade      SDDMLIV1              70
C      *in70      doweq      *off
C      Eval      cpt=cpt+1
C      T_IDTHEME  reade      SDDMLIV1              70
C      enddo
C      callp      sdAddPie(F1: 'Series1':cpt:
C                  T_NOMTHEME)
C      read      SDDMTHM              80
C      enddo

```

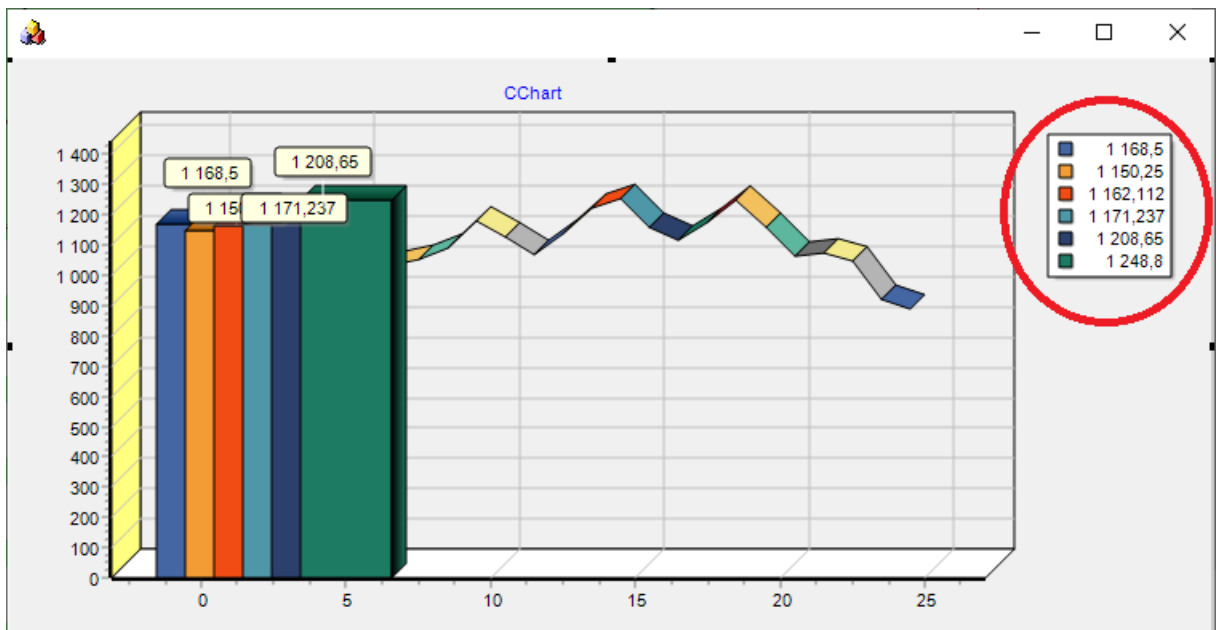
Propriétés

Une des difficultés de l'impression des graphiques est le nombre important de propriétés dans les composants.

Certaines propriétés sont sur le composant `CChart`, et d'autres sur les composants `TChartSeries`.

Legendes

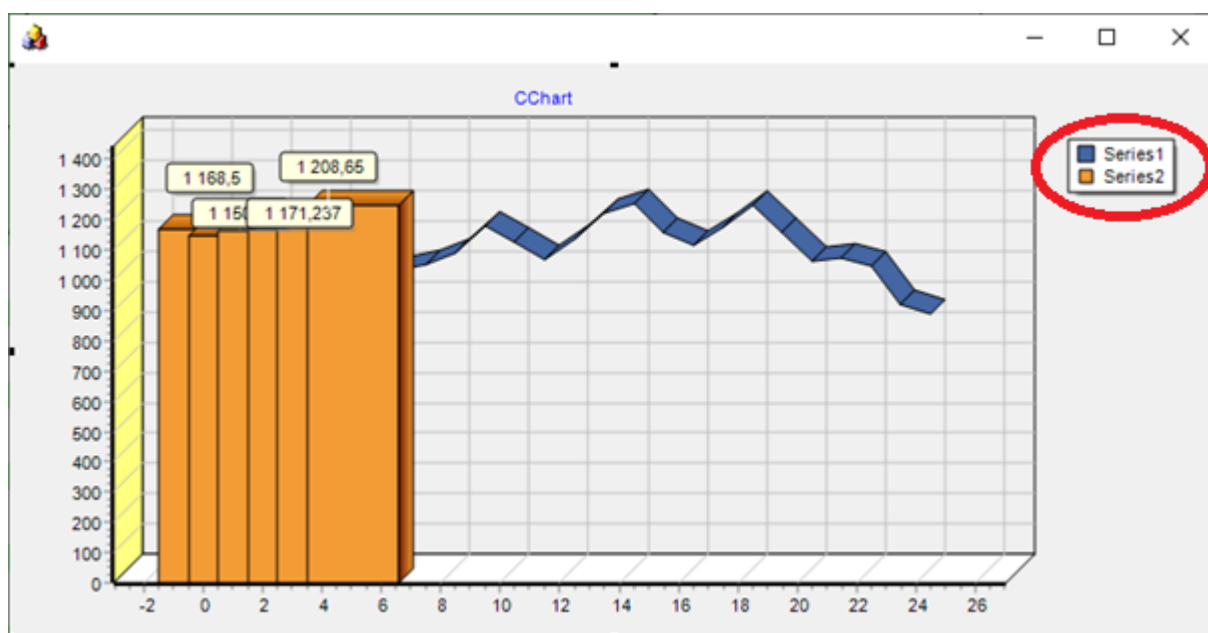
La propriété legend correspond à la partie entourée ci-dessous.



Utilisez la propriété **CChart.Legend** et la propriété **TChartSeries.Legend**

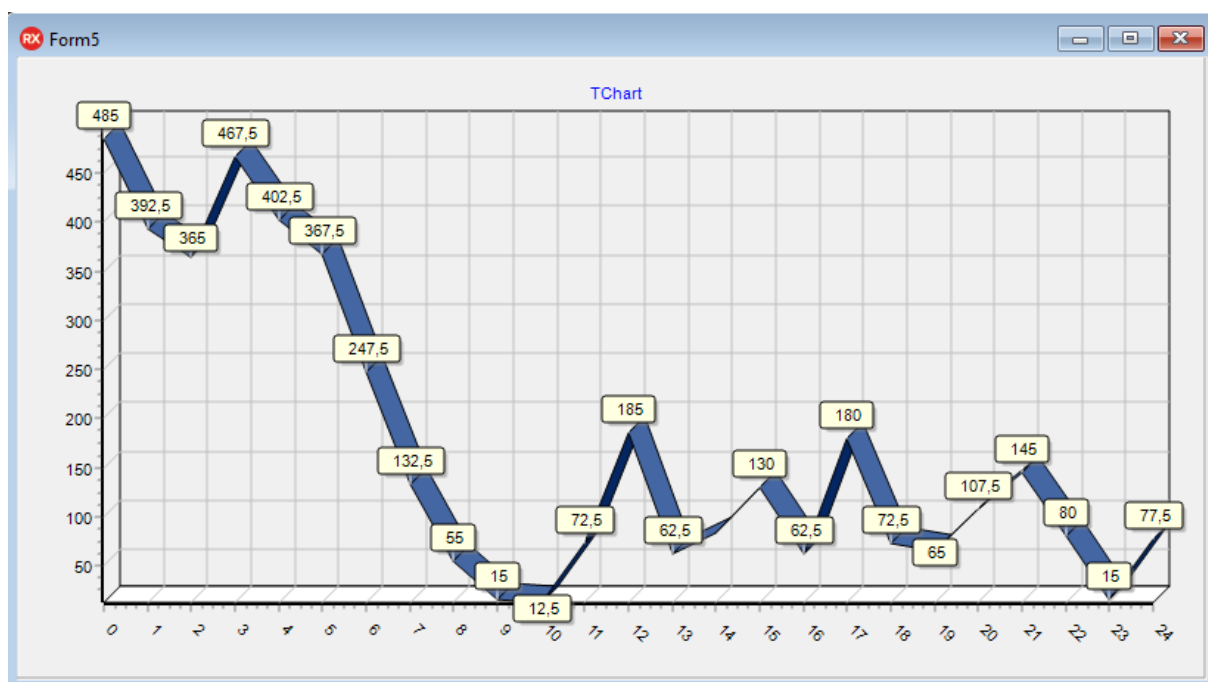
La propriété **CChart.Legend.LegendStyle** permet de choisir d'afficher les valeurs des séries, les noms des series, ou les dernières valeurs de chaque série

Si **CChart.Legend.LegendStyle** est à **IsAuto**, et que plusieurs séries ont la propriété **Legend.visible** à **true**, la légende est ainsi :



Étiquettes

Les étiquettes sont les petits carrés jaunes sur le schéma ci dessous :



Les étiquettes sont paramétrables avec la propriété `TChartSeries.Marks`

`TChartSeriesMarks` est une propriété de type `TSeriesMarks`, qui contient notamment les propriétés suivantes :

Arrow :

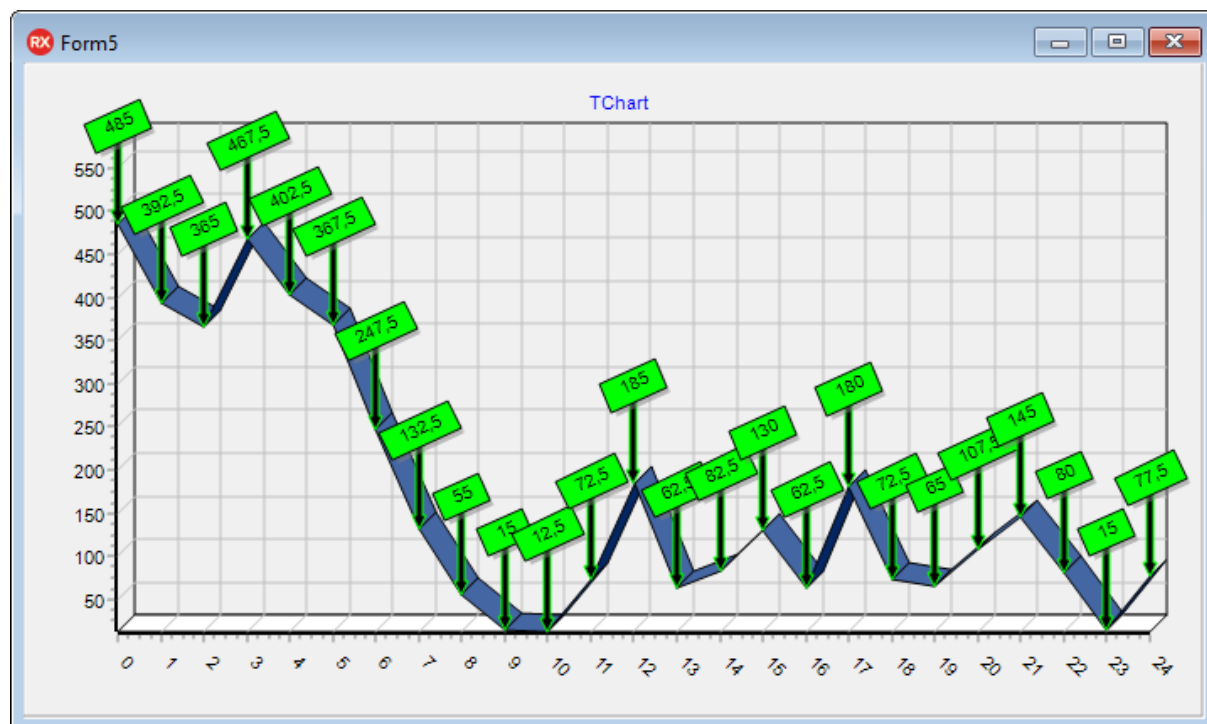
Permet de modifier l'aspect de la fleche qui relie les étiquettes.

ArrowLength :

Longueur de la fleche entre l'étiquette et le point sur le graphique.

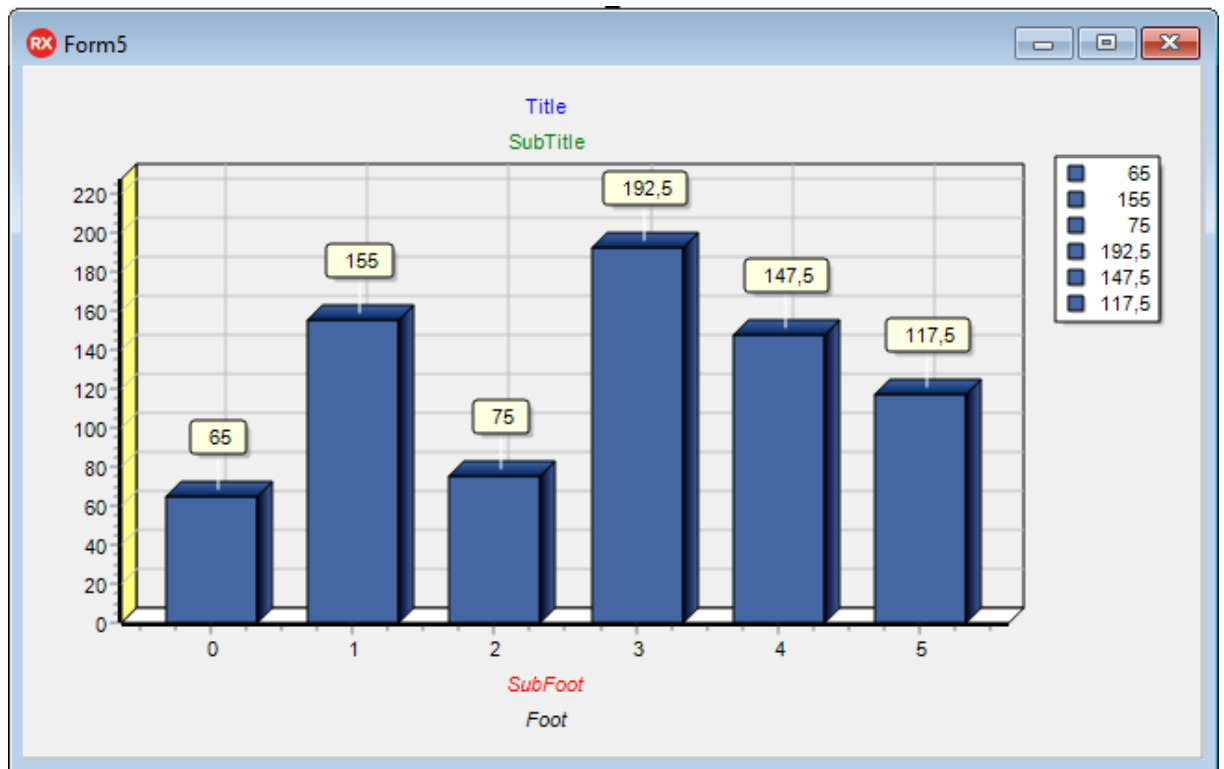
Color :

Couleur du fond de l'étiquette

**Titres**

Les titres se trouvent au niveau du CChart.

Utilisez les propriétés **CChart .Title**, **CChart .SubTitle**, **CChart .Footer** et **CChart .SubFooter**



Chacun de ces titres a les même propriétés.

Pour laisser un espace plus grand entre le titre et le graphe par exemple, ajoutez des lignes vides dans le texte.

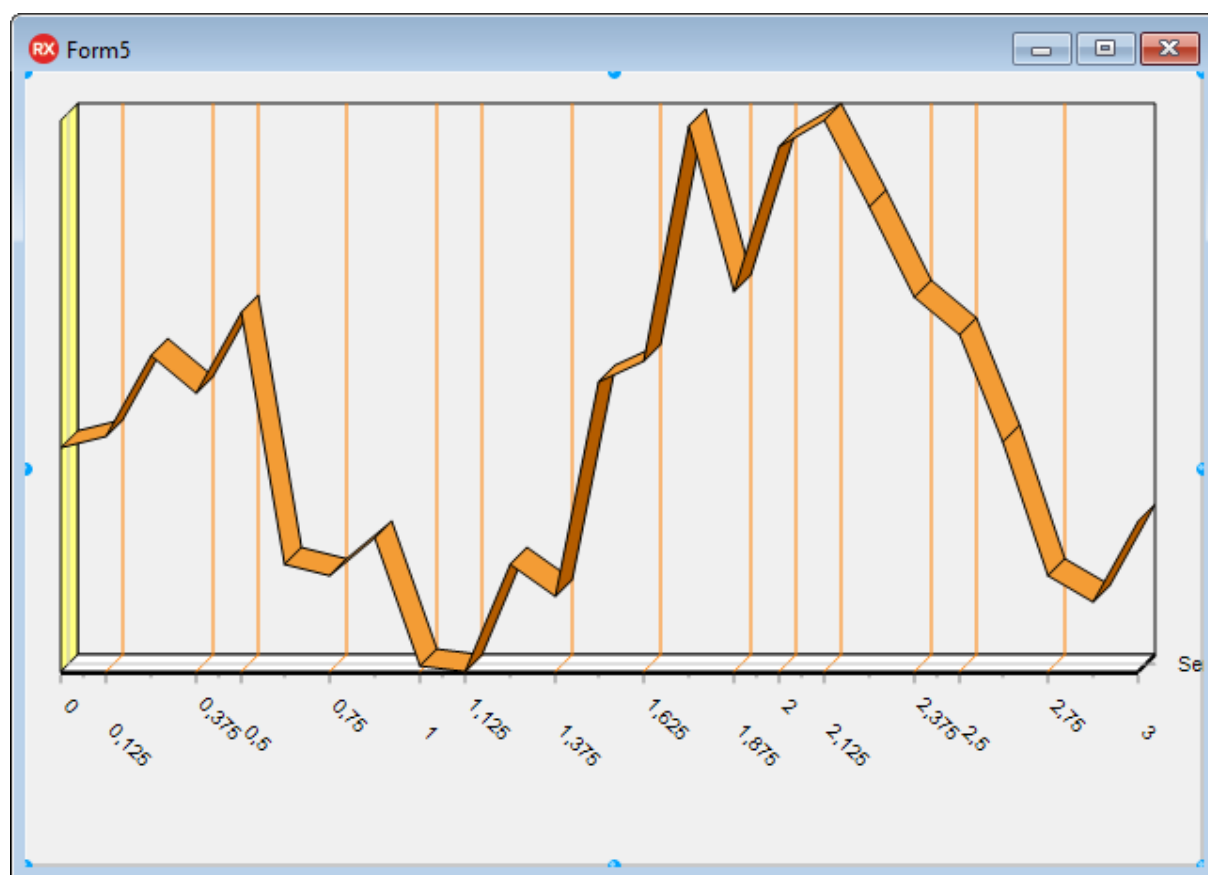
Axes

Pour modifier les axes, utilisez les propriétés **CChart.BottomAxis**, **CChart.LeftAxis**, **CChart.RightAxis** et **CChart.TopAxis**.

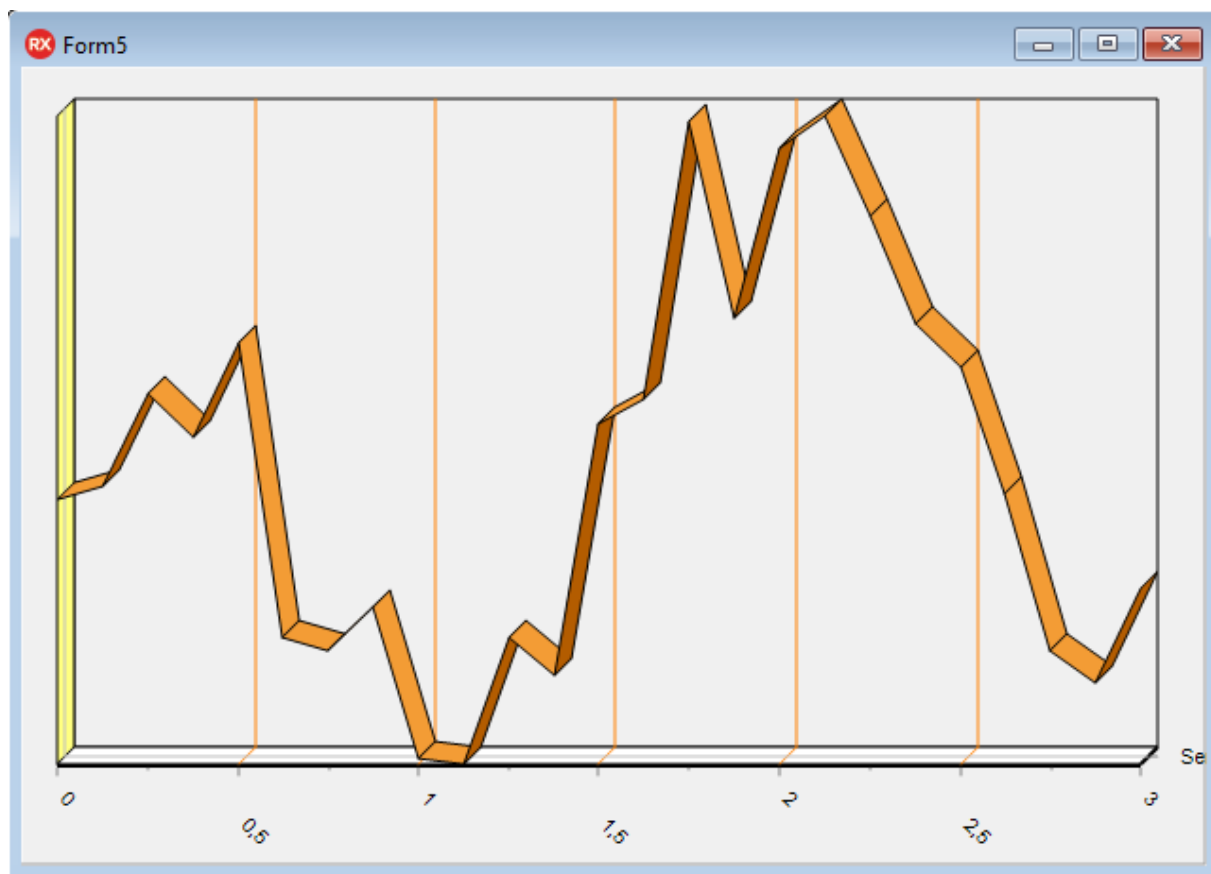
Ces 4 propriétés sont du même type, TChartAxis et ont donc les même sous propriétés.

labelStyle

CChart.xxxxAxis.labelStyle permet de choisir le texte affiché. Par exemple, la valeur `talPointValue` affichera des valeurs uniquement là ou il y a des points :

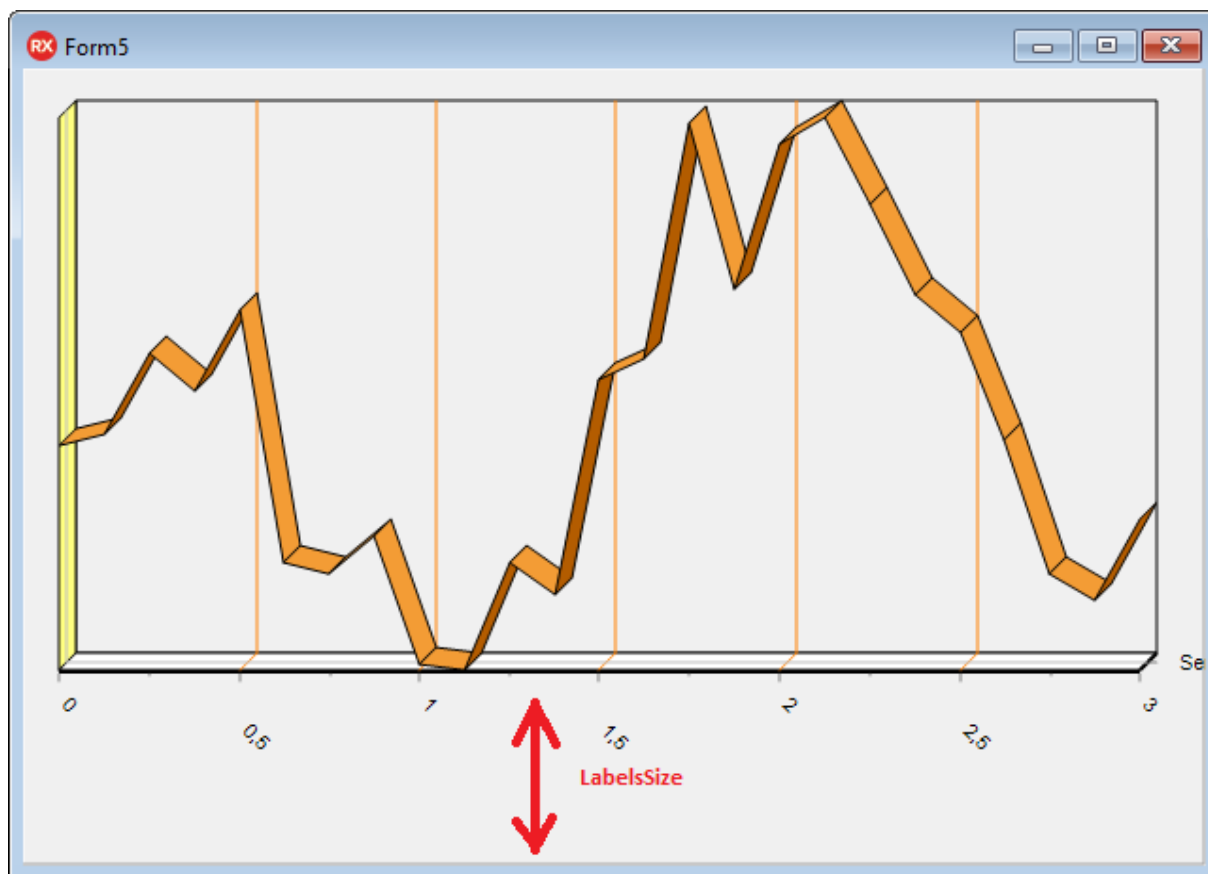


Alors que la valeur `taValue` affichera des valeurs régulièrement :



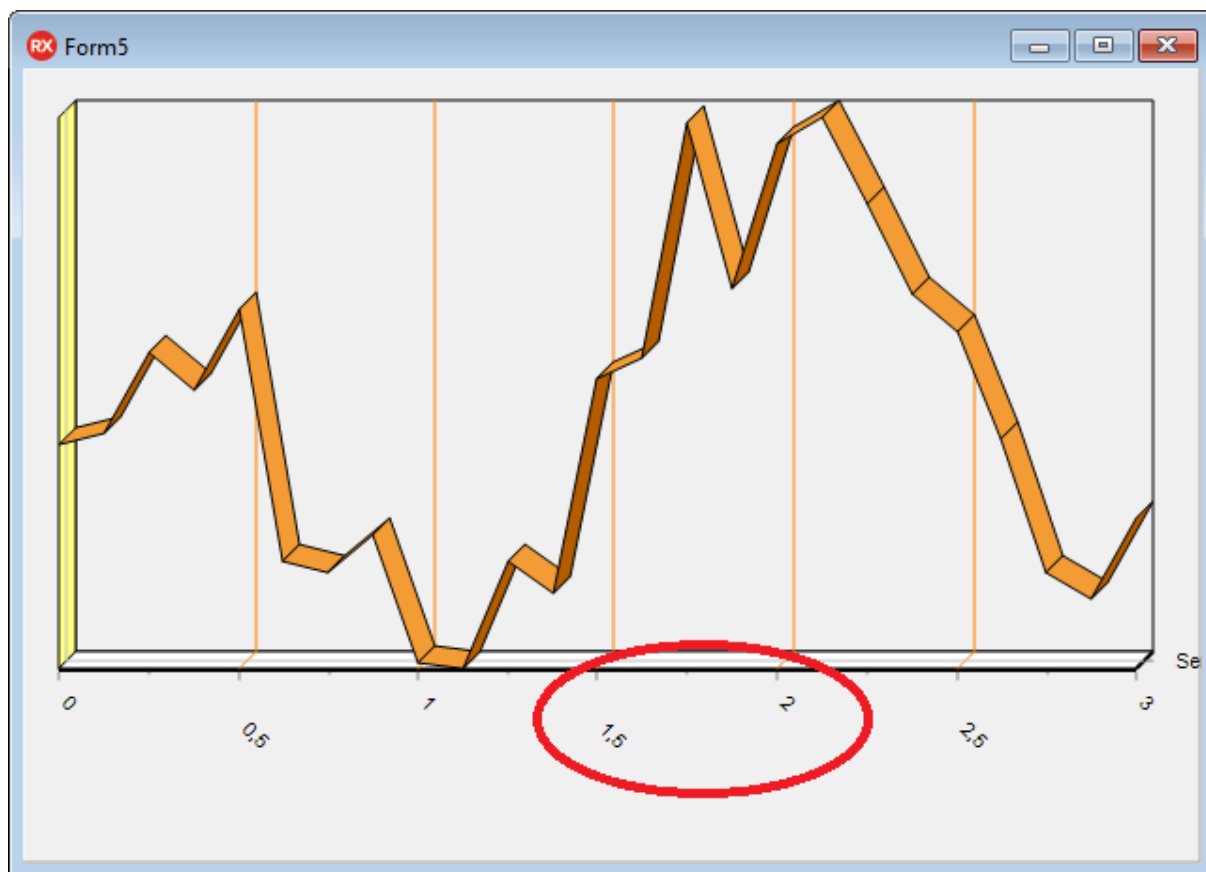
LabelsSize

La propriété LabelsSize vous permet d'agrandir la zone prévue pour les valeurs



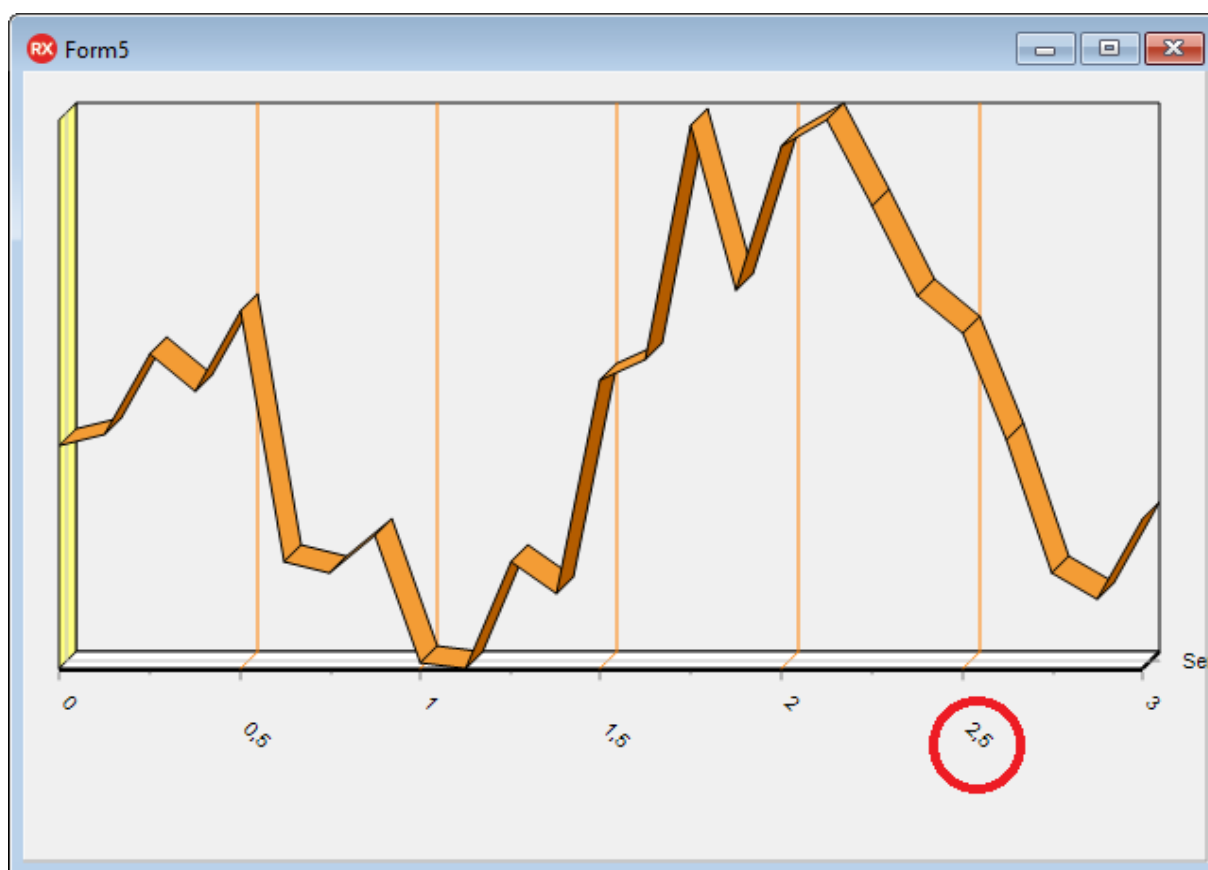
LabelsAlternate

La propriété LabelsAlternate permet une meilleure lecture des axes, une valeur sur deux est décalée



LabelsAngle

CChart.BottomAxis.LabelsAngle permet de modifier l'angle.
Sur l'exemple suivant, l'angle est de 315°



LabelsSeparation

La propriété `LabelsSeparation` spécifie le pourcentage de la distance minimum entre les labels

La valeur 0 supprime le calcul des labels qui se superposent.

Pagination

Utilisez la propriété `CChart.Pages` pour avoir une pagination dans le graphique.

Affectez une valeur à la propriété `CChart.pages.MaxPointsPerPage`.

Interrogez la propriété `CChart.pages.count` pour connaître le nombre de pages au total.

Interrogez/affectez la propriété `CChart.pages.Current` pour paginer.

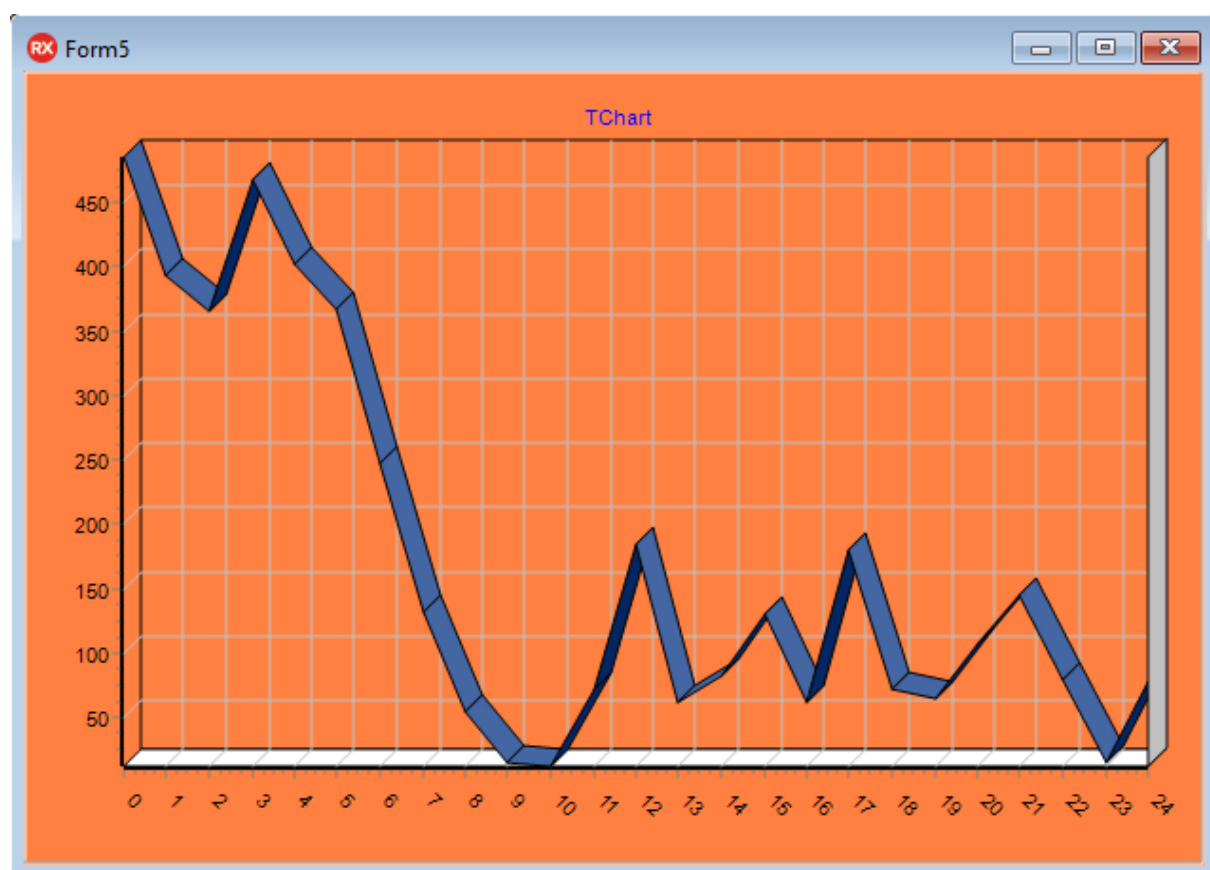
C'est à votre charge d'ajouter des composants qui permettent de naviguer dans la pagination.

Pour réaliser cela, utilisez les propriétés `CChart.pages.count` et `CChart.pages.current`

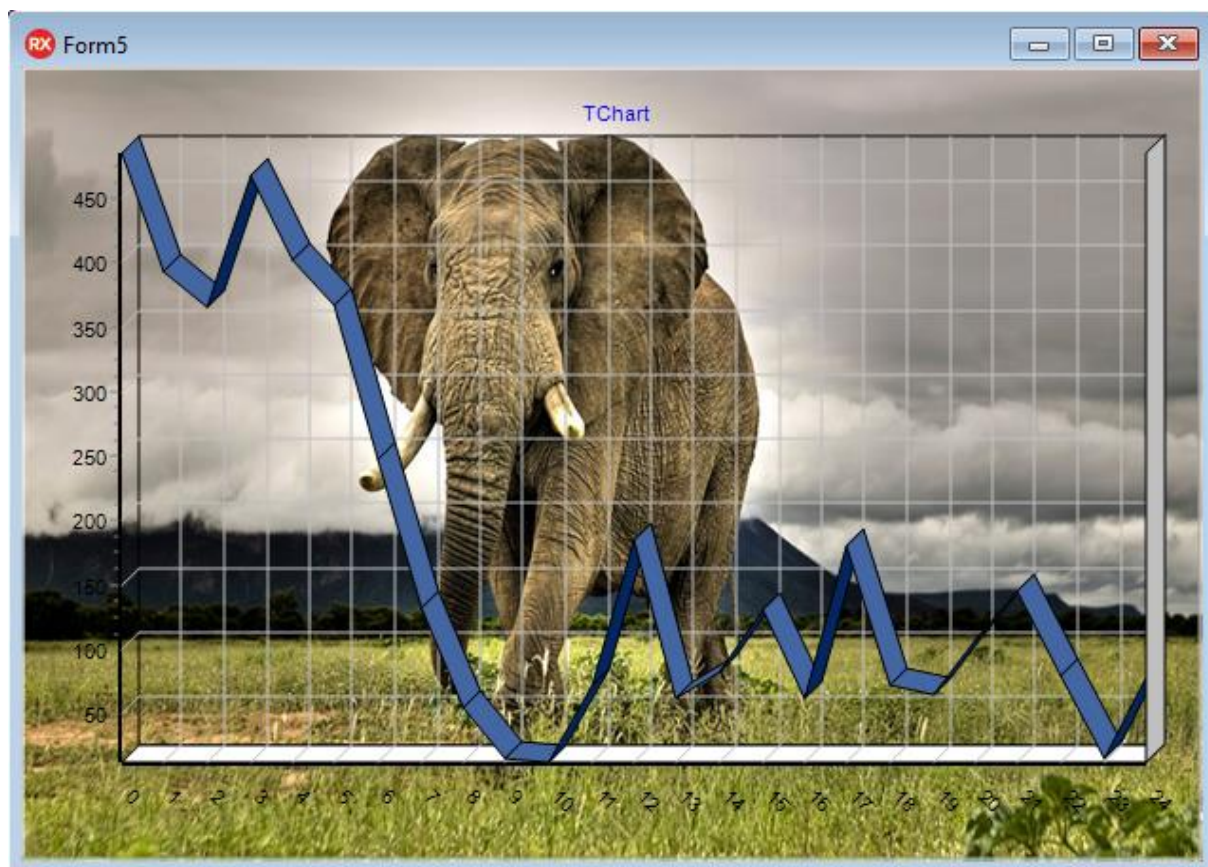
Fond

Le fond du graphique peut être modifié en utilisant

CChart.Color

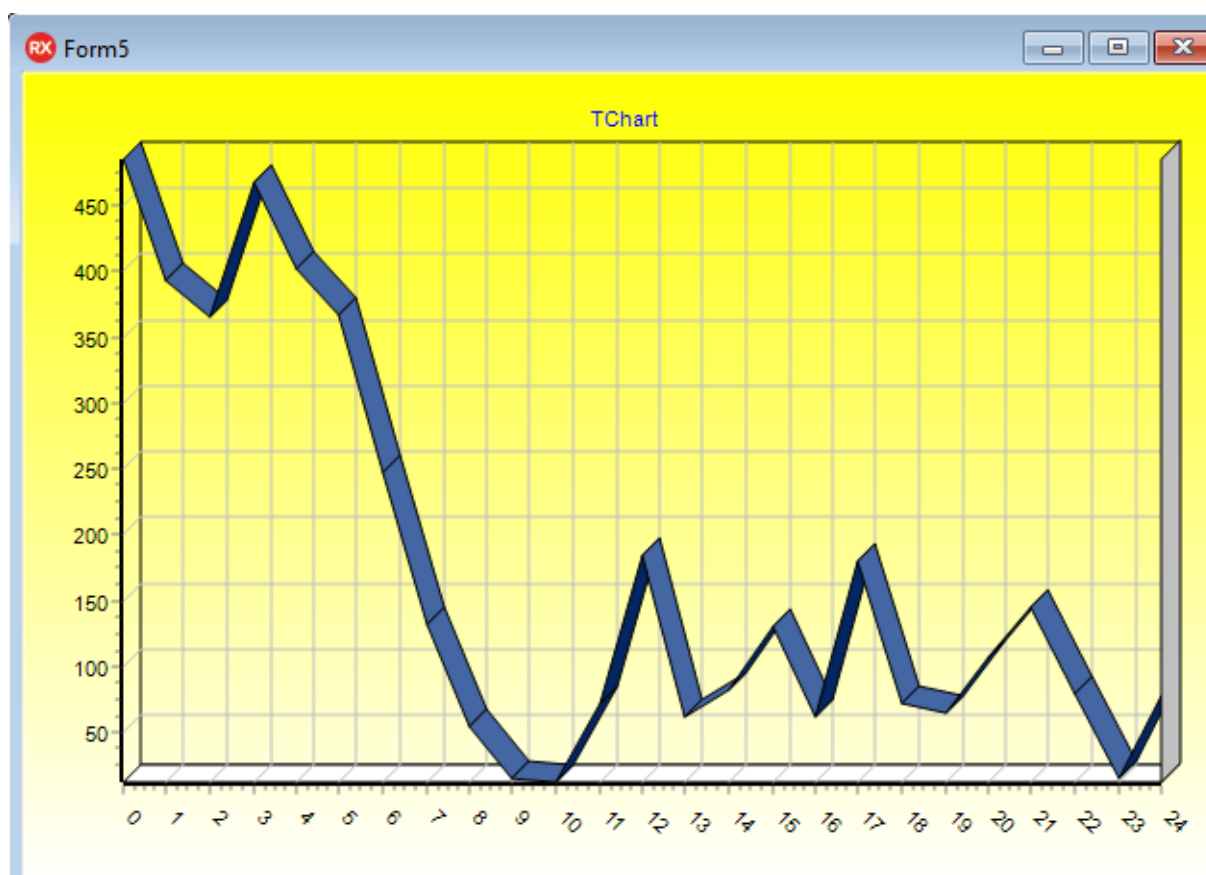


CChart.BackImage



L'aspect de l'image peut être paramétré avec les propriétés **CChart.BackImageInside**, **CChart.BackImageTransp**, **CChart.BackImageMode**

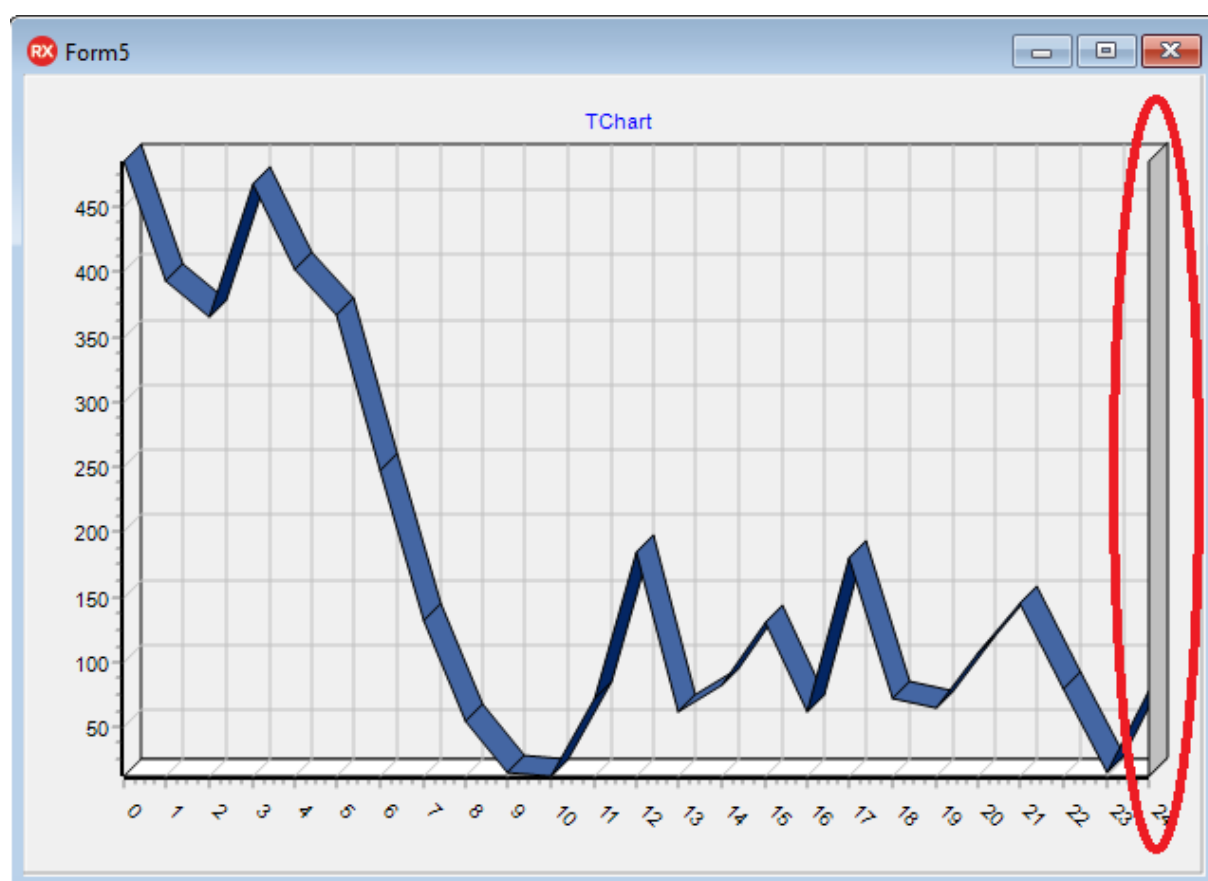
CChart.Gradient



Murs

Il y a quatre walls. LeftWall, RightWall, TopWall et BottomWall.

La figure ci dessous représente le RightWal.



Zoom

Dans le comportement par défaut, l'utilisateur peut zoomer sur une zone du graphique en dessinant un carré de haut en bas et de droite à gauche.

Si l'utilisateur dessine le carré dans un autre sens, le graphique est dézoomé.

L'utilisateur peut ensuite utiliser le click droit pour déplacer la partie zoomée du graphique.

Ce comportement peut être paramétré dans la propriété Zoom du composant CChart.

CChart.Zoom.Allow

Active ou désactive la possibilité de zoomer.

CChart.Zoom.History

Si cette valeur est à true, le mouvement de dezoom reviendra à la valeur de zoom précédente.

Il est alors possible de zoomer deux fois de suite, le dézoom reviendra au premier zoom.

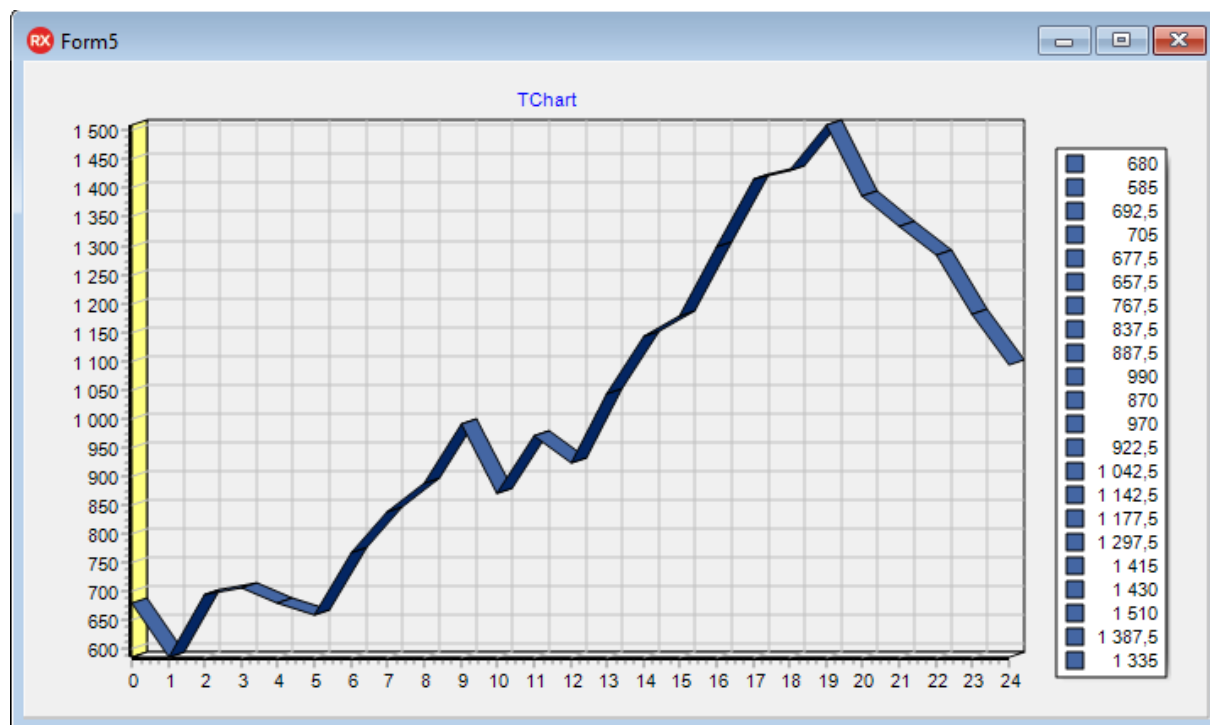
Si cette valeur est à false, le mouvement de dezoom enlèvera complètement le zoom.

3D

Un graphique peut apparaitre en 3D ou non.

Modifiez la propriété **CChart.View3D**

View3D à true



View3D à false :



Les propriétés **CChart.View3DOptions**, **CChart.View3DWalls**, **CChart3DPercent**, **TChartSeries.Dark3D** permettent de paramétrer la vue 3D.

Couleurs

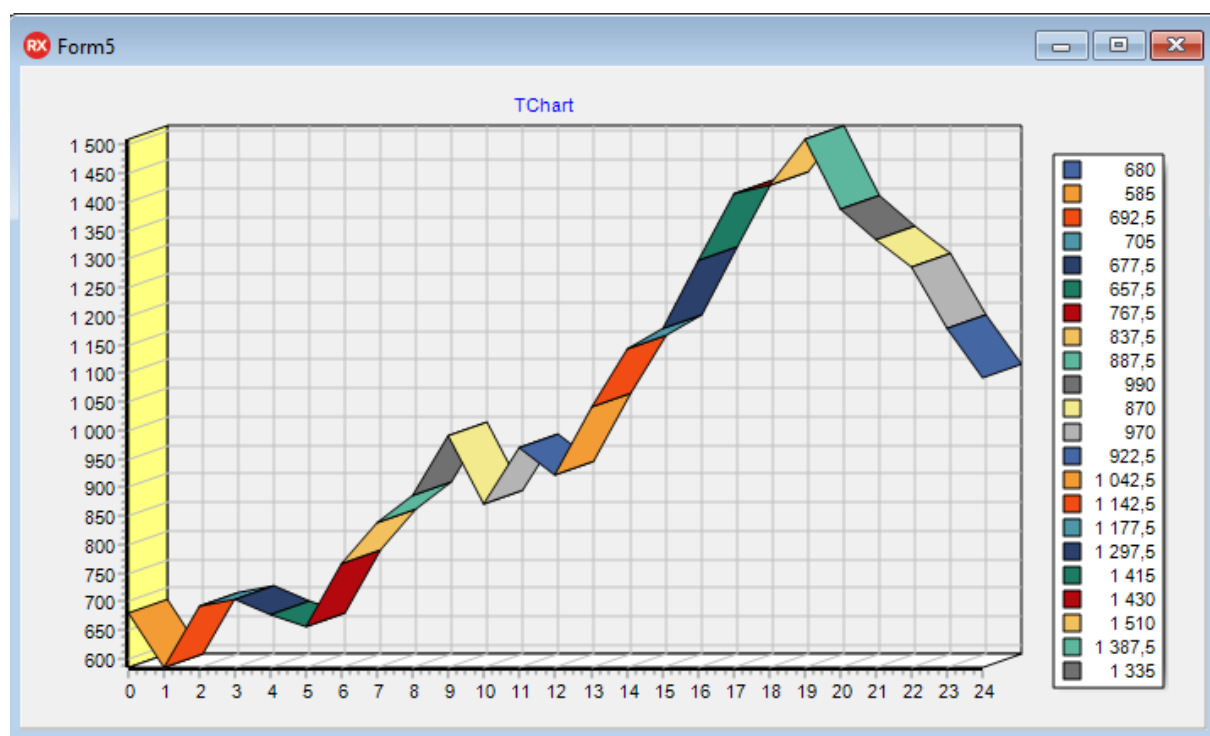
Lors de l'ajout de valeurs dans une série, il est possible de préciser la couleur, à l'aide de paramètres optionnels.

Par exemple :

```
sdAddPie(F1:component:Value:Label:Color)
sdAddBar(F1:component:Value:Label:Color)
sdAddXY(F1:component:ValueX:ValueY:Label:Color)
```

Si aucune couleur n'est précisée, la couleur de la série sera déterminée par la propriété **TChartSeries.SeriesColor**

Il est aussi possible d'affecter true à la propriété **TChartSeries.ColorEachPoint**. Dans ce cas, le composant choisi lui même une couleur pour chaque valeur.



Impression avec graphique

Pour ajouter un graphique dans une impression, utiliser les composants d'impression, CReport et CRepBand.

Posez un composant CRepChart sur un composant CRepBand.

Ajoutez un composant de type CChart.

Modifiez la propriété reference du composant CRepChart en la faisant pointer sur le composant CChart.

Ajoutez les données dans le composant CChart comme vu précédemment, puis lancez l'impression du composant CReport (avec sdPrint ou sdPreview)

Pdf avec graphique.

Pour générer un fichier pdf contenant un graphique, utilisez le composant CReport comme dans l'exemple précédent, puis utilisez la fonction sdSavePdf.

Gdi+

Pour utiliser la bibliothèque GDI+ pour dessiner le graphique, utilisez le composant CChartGDIPlus.

Modifiez la propriété TeePanel du composant CChartGDIPlus, faites la pointer sur le composant CChart.

Modifiez la propriété Active du composant CChartGDIPlus, affectez la valeur true.

Ascenseurs

Le composant ne propose pas d'ascenseurs, mais il est possible de mettre le composant CChart dans un composant CScrollBar et de jouer avec la propriété range de CScrollBar.vertScrollBar et CScrollBar.HorzScrollBar.

