

next

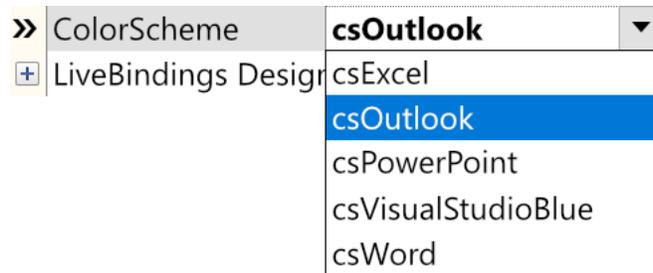
NextStandard 6

Shared Components

This part of suite includes mostly non-visual components (TComponent descendants) that are shared between all components in Next Suite 6.

NxColorScheme6

NxColorScheme6 component requires only to be placed on form where other visual Next Suite 6 components are placed. This component set color palette for all Next Suite components of owner Form. Color "palette" is specified by CoLorScheme property.

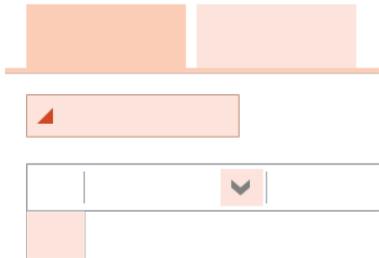


Switching color schemes is only effective when *modern* style is selected for component. Many components in NextSuite 6 have Style published property which when set to stModern tell component to paint itself in modern (Microsoft Office™ 365) style.

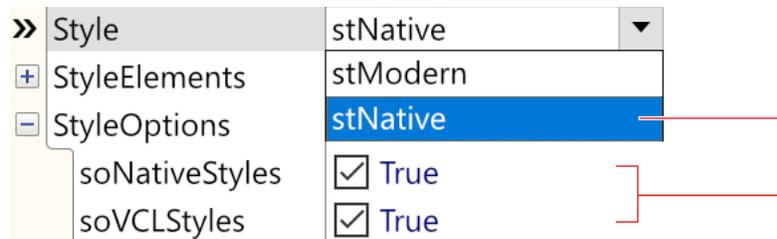
This property can be found in NextGrid6, NextInspector6, NxPageControl6, but also in some smaller components such as NxFlipPanel6.



Component will use colors from specified ColorScheme. For example, csExcel scheme will use green as dominant, csPowerPoint red, csWord blue etc. Example for csPowerPoint:



If Style is set to stNative, Windows (or Delphi VCL Style) will paint component elements such as headers, buttons etc.



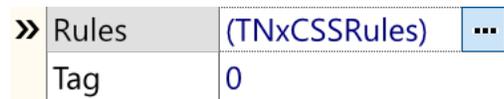
If Delphi version support VCL Styles, these styles will have priority over Windows native styles (themes). Users with older versions of Delphi can only compile project with support for native styles.

If none of flags is set, component will be painted in classic style.

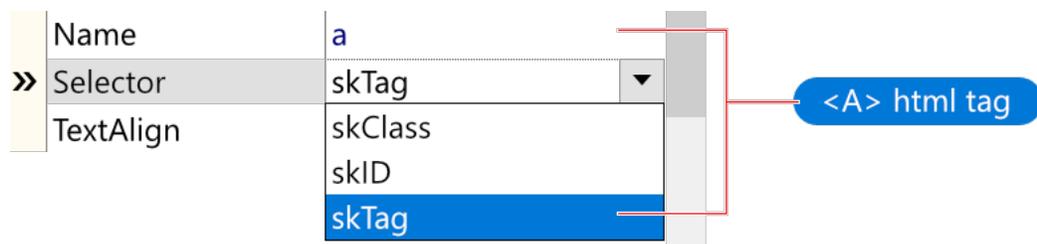
NxStyleSheet6

Several components from Next Suite have capability for rendering simple HTML formatted content (for example NxHTMLLabel6 component from Next Collection 6 package). NextGrid6 even have column type capable of rendering HTML content inside cell.

This HTML content can be additionally formatted via NxStyleSheet6 non-visual component. Same as in standard CSS there are rules which can be applied to tags. These rules can be set inside Rules property:

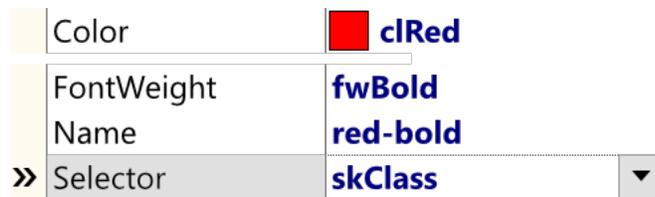


For Example, appearance of A HTML tag can be adjusted (font can be changed, text can be underlined etc.).



If Selector property is set to skClass, class attribute of HTML element can be set.

```
sample <span class="red-bold">text</span>
```



If component support html formatting, there is always a StyleSheet property where NxStyleSheet6 component can be set.



Properties for single Rule are very similar to ones in CSS.

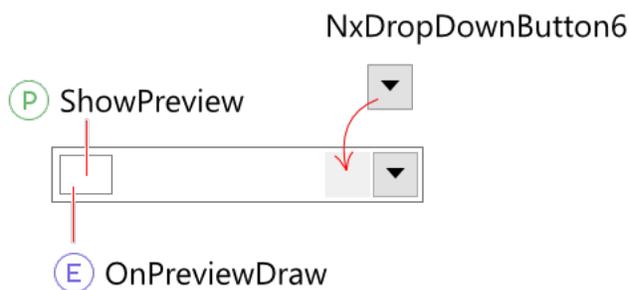
NextEditors 6

Set of stand-alone editor components that are also ready to be used as In-place edit components for bigger components such as NextGrid6, NextDBGGrid6 or NextInspector6.

Next Editors components are placed inside "Next Editors 6" Tool/Component Palette Category.

Common for all editors are that they implement `INxInplaceEdit` interface (one that is property for Column in NextGrid, or for Node inside NextInspector).

Most of them have `TNxCustomEdit6` as ancestor so they share bunch of properties and events. Also, they can accept other controls (such as `NxDropDownButton6`, also in Next Editors 6 palette).



TIP

Place any control inside any `NxEdit6` (`NxButtonEdit6`, `NxSpinEdit6`...) descendants and set `Align` property of child to either `aLeft` or `aRight`.

NxGridPicker6

Version v6.0.13 brings new NxGridPicker6 component, a mini-grid picker control.

Columns can be added and configured via standard TCollection property Columns.

Rows can be easily added via AddRow method:

DELPHI

```
NxGridPicker61.AddRow(['Audi', 'Germany']);  
NxGridPicker61.AddRow(['Toyota', 'Japan']);
```

After user select value, OnSetValue event occurs:

DELPHI

```
procedure TForm1.NxGridPicker61SetValue(Sender: TObject;  
    SelectedIndex: Integer; var Value: WideString);  
begin  
    // by default 1st (index = 0) cell will be used  
    Value := NxGridPicker61.Row[SelectedIndex].Cells[2];  
end;
```

NextGrid 6

NextGrid version 6 is VCL component, completely written from scratch.

Component is continuing with name and most of the features from previous version 5, but code is almost completely rewritten, that said it's cleaner, faster, easier to expand for new features, and it is more reliable.

Design-time

NextGrid component is very similar to standard VCL ListView component where columns are usually placed and configured in design-time via built-in Editor, and data is later managed in run-time.

In standard ListView, data presentation style is set by `ViewStyle` property (which can be `vcIcon`, `vsList`, `vsReport`...). Since in NextGrid6 not all properties are shared between views (for example Report view doesn't have `SlidesHeight` property), each view is separate object which first need to be added and then (optionally) configured. This is done very easy via [Views Editor](#).

Also, since there are dozens of column-types (where each one includes own properties and paint data in different way), design-time editors for Columns are vastly improved over Delphi standard design-time editors.

Views Editor

Views Editor can be opened in couple of ways:

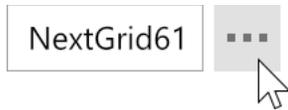
- a) By right clicking on NextGrid6 and choosing "Views Editor..." Option:



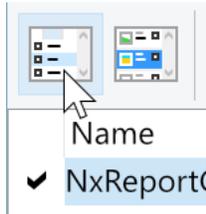
- b) By clicking on ellipsis button beside Views property in Object Inspector:



- c) Or by clicking on ellipsis button inside NextGrid



In Editor, new view (Report or Slides) can be added by clicking on one of first two buttons:



Then, it can be configured in Object Inspector who will list properties of selected View.

If name of view is set, IDE will create variable for each view:

DELPHI

type

```
TForm1 = class(TForm)
  NextGrid61: TNextGrid6;
  NxReportGridView61: TNxReportGridView6;
  NxSlidesGridView61: TNxSlidesGridView6;
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

After all views are added and set, active view (one that will be shown in grid) can be set by ActiveView property:



Faster way to do it, is by clicking on checkmark button beside view inside editor.

	Name
✓	NxReportGridView61
	NxSlidesGridView61

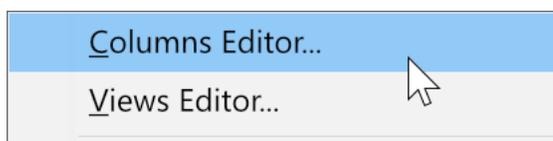
If at least one view is active at the time, Component Editor (right-click popup) will show additional options:



Columns Editor

Columns Editor can be open is several ways:

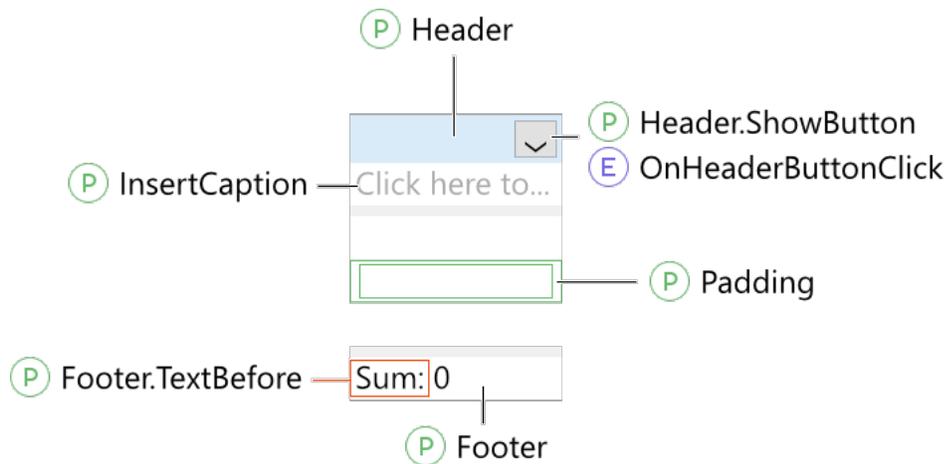
- a. By simply double-clicking on the grid control while it's placed on form.
- b. Similar to Views Editor, Columns Editor can be open by right-clicking on grid and choosing "Columns Editor":



Inside editor, after column is selected they can be configured inside IDE Object Inspector, or after clicking on icon inside 1st column, inside mini quick-editor.

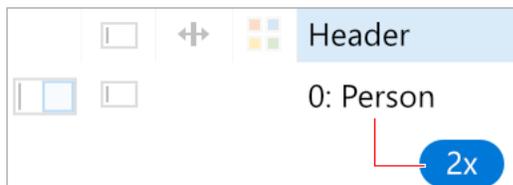
Parts of Column

All columns share some parts (and properties) such as Header and Footer.



Inside Header property (object) there can be found properties such as Caption (text inside header rectangle), Glyph, Font etc.

Caption property can be easily (and quickly) set by double clicking on cell inside Header Column inside Columns Editor:



Run-time

Working with Views and Columns is also easy in run-time via Views and Columns properties.

Views in run-time

Easiest way to a new view is to use Add method:

```
DELPHI
uses NxGridView6;
...
// Report GridView
NextGrid61.Views.Add(TNxReportGridView6);
// or SlidesView
NextGrid61.Views.Add(TNxSlidesGridView6);
```

Add method create view of specified class type, and return newly created view so view can be configured immediately:

```
DELPHI
with NextGrid61.Views.Add(TNxReportGridView6) do
begin
    GridLines := True;
    ShowFooter := True;
    SelectionStyle := stAlphaBlended;
    // etc.
end;
```

Views also provides access to already existing views via array property:

```
DELPHI
NextGrid61.Views[0].GridLines := True;
```

For setting view-type specific properties typecasting can be used:

DELPHI

```
TNxSlidesGridView6(NextGrid61.Views[0]).SlideHeight := 120;
```

Switching views is same as in design-time done by `ActiveView` property of grid:

DELPHI

```
NextGrid61.ActiveView := NextGrid61.Views[2];
```

There is also `ActiveViewIndex` property which can be set to the index of view.

Columns in run-time

Column can be added by calling `Add` method of `Columns` property:

DELPHI

```
NextGrid61.Columns.Add(TNxNumberColumn6, 'Quantity', 'QTY');  
NextGrid61.Columns.Add(TNxTextColumn6, 'Person');
```

First parameter is required and specifies column class (type) to be added. Other two parameters are optional and they specify `Header.Caption` and `Header.ShortCaption` property.

Working with Rows

Adding new rows

New row (or multiple rows at once) can be added in several ways.

Method used to add one, or specified number of rows at the end of grid is `AddRow`:

DELPHI

```
NextGrid61.AddRow;  
NextGrid61.AddRow(100);
```

HINT

Calling AddRow method once with Count parameter set is faster than calling AddRow multiple times (inside loop for example).

New row(s) can be also inserted at any position inside grid by InsertRow method:

DELPHI

```
NextGrid61.InsertRow(NextGrid61.SelectedRow);  
NextGrid61.InsertRow(0, 2); // Insert 2 rows at index 0
```

HINT

Methods for adding new rows are updating handy LastAddedRow method.

Managing rows

Existing rows can be deleted or moved by using one of following methods.

For single row deletion, DeleteRow method is used:

DELPHI

```
NextGrid61.DeleteRow(NextGrid61.SelectedRow);
```

There is couple of shorthand deletion methods such as DeleteSelected, DeleteLastRow which names are self-explanatory.

ClearRows method deletes all rows in grid.

Every row is object with own properties that can be accessed via Rows array property.

Example:

DELPHI

```
NextGrid61.Row[2].Height := 24;
```

In row object there are other important properties such as `Visible`, `Selected`, `Data` etc. Full list can be found inside reference.

Sometimes it is useful to check if row with specified index actually exists. For this purpose, there is `RowExists` method.

DELPHI

```
var IndexToDelete: Integer;  
// code  
if NextGrid61.RowExists(IndexToDelete) then  
NextGrid61.DeleteRow(IndexToDelete);
```

Working with Cells

Accessing Cells

Each cell in `NextGrid6` is single object with own properties and methods.

Cell can be accessed via `Cell`, a two-dimensional array property:

```
NextGrid61.Cell[IndexOfColumn, IndexOfRow].Property :=
```

There are also similar properties such as `Cells` which return value of `AsString` property of `Cell`.

DELPHI

```
MyStringVar := NextGrid61.Cells[2, 1];  
// above is equal to:  
MyStringVar := NextGrid61.Cell[2, 2].AsString;
```

There is also a property `CellBy` that take `Variant` as parameter while calling. Parameter can be one of the strings such as `first`, `last` or `selected`. For column can be a column name. And as for `Cell` property it can be index.

Examples:

DELPHI

```
NextGrid61.CellBy['MyCheckBoxColumn', 'selected'].AsBoolean :=  
CheckBox1.Checked;  
NextGrid61.CellBy[2, 'selected'].AsString := Edit1.Text;  
if NextGrid61.CellBy['first', 'selected'].AsBoolean then //
```

Optimization

Managing large number of cells can be optimized by using BeginUpdate and EndUpdate. First method lock scrollbar, postpone unnecessary calculations and redrawing(s), all until paired EndUpdate is called.

DELPHI

```
NextGrid61.BeginUpdate;  
NextGrid61.AddRow(100000);  
for i := 0 to NextGrid61.RowCount - 1 do  
begin  
    // set, change cells  
end;  
NextGrid61.EndUpdate;
```

Selecting cells

Selected cell is indicated by SelectedCol and SelectedRow properties. Initially both properties are set to -1.

User changes these properties by selecting cells with mouse or keyboard. These two properties are also often changed by developer directly, or by calling SelectCell method directly.

Before selection is changed, OnSelectQuery event is called. Inside this event Accept parameter can be set in order to prevent selection to change. Example:

DELPHI

```
procedure TForm1.NextGrid61SelectQuery(Sender: TObject; ACol,  
    ARow: Integer; var Accept: Boolean);  
begin  
    if (ACol = 2) and (ARow = 2) then Accept := False;  
end;
```

If SelectFullRow is True, all cells in row will be highlighted. Even if this property is True, SelectedCol property may be changed if user select cell in other column.

InvertSelection property will indicate selected cell event if SelectFullRow is turned on.

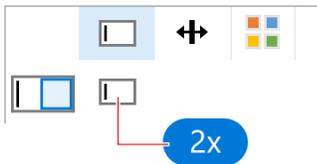
If both SelectFullRow and MultiSelect properties are True, user will be able to select multiple rows by using Shift and Control keys. In so called multi-select mode developer can read Selected Boolean property of each Row and do custom actions based on value.

DELPHI

```
var  
    i, SelectedCount: Integer;  
begin  
    SelectedCount := 0;  
    for i := 0 to NextGrid61.RowCount - 1 do  
        begin  
            if NextGrid61.Row[i].Selected  
                then SelectedCount := SelectedCount + 1  
        end;  
    Edit1.Text := IntToStr(SelectedCount);  
end;
```

Editing

Cells in column can be edited if `Editing` property of column is set to `True`. Double-clicking on icon inside Columns Editor can quickly toggle this property (and also indicate current state):



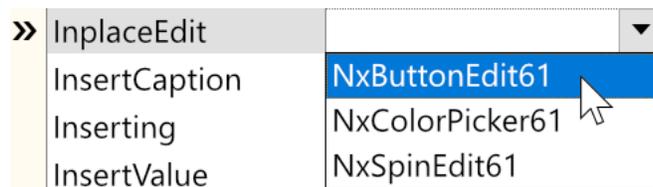
Editing can be also enabled/disabled on cell level via `Enabled` property of Cell:

DELPHI

```
NextGrid61.Cell[0, 0].Enabled := False;
```

User can start editing cell by either clicking on it, or by start typing while cell is selected. If column support editing, it will automatically create own edit control and place it inside cell.

With using column `InplaceEdit` property developer can override default choice for control.



Custom edit control can be also set inside `OnGetInplaceEdit` event which is very handy:

DELPHI

```
uses NxEdit6;
```

```
procedure TForm1.NextGrid61GetInplaceEdit(Sender: TObject; ACol,
  ARow: Integer; var InplaceEdit: INxInplaceEdit);
begin
  if ACol = NxTextColumn61.Index then
    InplaceEdit := TNxSpinEdit6.Create(Self);
end;
```

When editing finally starts OnEditEnter event is called and InplaceEdit property of Grid can be read. This property reference edit control used in editing.

DELPHI

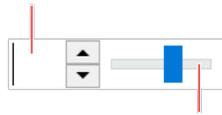
```
uses NxEdit;
...
procedure TForm1.NextGrid61EditEnter(Sender: TObject; ACol, ARow:
  Integer);
begin
  if ACol = NxNumberColumn61.Index then
    begin
      { We know that this column uses TNxSpinEdit6
        as edit control }
      TNxSpinEdit6(NextGrid61.InplaceEdit).Max := 20;
    end;
end;
```

Interactive property

Some columns that doesn't have Edit component for keyboard editing (CheckBox column for example) can disable user interactions by setting Interactive property to False.

Some column types like NxTrackBarColumn include both editing area and interactive part of cell. In that case both properties can be set independently.

 Editing



 Interactive

Importing and Exporting data

NextGrid 6 include several methods for importing (loading) and exporting (saving) columns or cells. They are all grouped inside `Serialize` property.

This property include couple of sub-properties such as:

- `SeparatorChar` (default `","`)
- `Encoding` (default `ekUnicode`)
- etc.

Grid columns (layout/structure) can be saved via several methods (`SaveToINI`, `SaveToRegistry`, `SaveToXML`):

DELPHI

```
NextGrid61.Serialize.SaveToINI('ini_file.ini');  
// and Loading:  
NextGrid61.Serialize.LoadFromINI(OpenDialog1.FileName);
```

Cells (content) can be saved to Comma-Separated-Values (CSV) file:

DELPHI

```
NextGrid61.Serialize.SaveToCSV('text_file.txt');
```

There is also `LoadFromCSV` counterpart method.

Saving to and loading from XML

Version 6.0.16 of NextGrid6 includes `SaveToXML` and `LoadFromXML` methods.

Both methods include optional `TNxDataKind` parameter `Kind` (`dtLayout` and `dtContent`). This parameter specifies whether layout (Columns) or content (cells) will be saved/loaded.

TIP

For Demo project, open `Demos\Next Grid\Delphi XE6\Save and Load from XML`

NextDBGrid 6

NextDBGrid 6 component inherits NextGrid 6, with almost all properties and methods, but extended with DB support.

As many other data-aware controls, NextDBGrid include DataSource property which connects grid with TDataSet (Table, Query...).

Design-time

Same Columns and Views design-time editors are available for NextDBGrid.

Columns

If grid have no columns, and `aoAutoAddFields` is set inside `ActivationOptions` property of Grid, new column will be created for each field automatically after linked DataSet is open (activated). (v6.0.6)

WARNING

In design-time each column requires property Name set. This can be done easily by choosing "Set Missing Names" from popup menu inside Columns Editor.

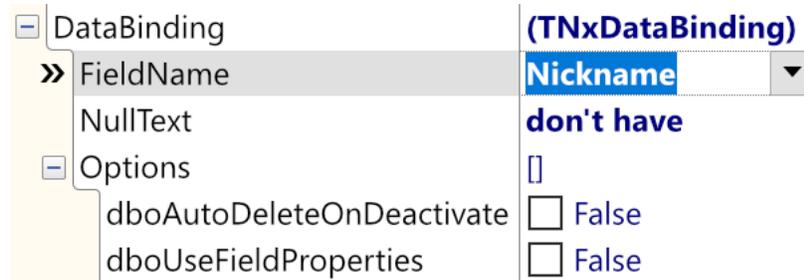
In design-time, there is a handy option inside component popup menu:



With this option column type can be set for every field. In run-time `DoAddFieldQuery` event can be used for same purpose.

DataBinding

NexDBGrid use same columns from NextGrid and extend them with `DataBinding` property:



If `DataSet` is open, field associated with column can be chosen by `DataBinding.FieldName` property after column is selected in Columns Editor.

If `dboUseFieldProperties` value of `Options` property is set to `True`, column will assign some properties from `TField` (e.g. `Alignment`, `Visible`, `ReadOnly` etc.). If Column is `TNxFloatColumn6` descendant, and field is also of numeric type, additional properties are copied too, for example: `DisplayFormat`, `MaxValue`, `Precision`...

Some columns like checkbox column (`TNxDBCheckBoxColumn6`) add some data-related properties that doesn't exists in non-db version.

In run-time `DataBinding` property of Grid can be used to access each Column's Field:

DELPHI

```
MyFieldName := NextDBGrid61.DataBinding[IndexOfColumn].FieldName;
```

Cells

Even if `NextDBGrid6` is data-aware grid component, it still includes `Cell` property, same as `NextGrid6`.

NextInspector 6

NextInspector 6 is Object Inspector (aka. properties control) component with more advanced and modern ways of managing (inspecting) properties.

Smallest element of NextInspector6 is TNxInspectorNode6 which as in RAD Studio Object Inspector can have Caption and Value. Each node also can have children nodes (again, same as in Object Inspector).

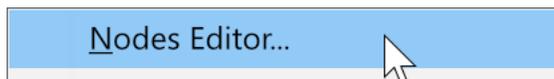
Nodes can be easily managed in both design and run-time.

Design-time

Same as NextGrid6, NextInspector6 also own dedicated design-time editor for managing own data.

Design-time editor can be open in couple of ways:

- a) By double-clicking on NextInspector6 component on form.
- b) By right clicking on NextInspector6 and choosing "Nodes Editor..."



- c) By clicking on ellipsis button beside Nodes property of NextInspector6.

Run-time

Most important property for working with NextInspector6 in run-time is Nodes property.

DELPHI

```
uses NxInspectorNodeClasses6;
```

```
...
```

```
NextInspector61.Nodes.Add(NextInspector61.Nodes[0],  
TNxSpinInspectorNode6);
```

Add method returns reference to newly added node which can be set immediately:

DELPHI

```
with NextInspector61.Nodes.Add(NextInspector61.Nodes[0],  
TNxSpinInspectorNode6) do  
begin  
    Caption := 'Size';  
    ImageIndex := 2;  
    AsInteger := TrackBar1.Position;  
end;
```

In following example Spin Node is added, which introduce several new properties to base node class. By using typecasting these properties can be also set:

```
with NextInspector61.Nodes.Add(NextInspector61.Nodes[0],  
TNxSpinInspectorNode6) as TNxSpinInspectorNode6 do  
begin  
    Caption := 'Size';  
    ImageIndex := 2;  
    AsInteger := TrackBar1.Position;  
    // TNxSpinInspectorNode properties  
    Max := 50;  
    Precision := 2;  
    FormatMask := '0';  
end;
```

Reading properties

Each Node can be linked to published property of other object. For this purpose, there is a BindProperty property for each node.

DELPHI

```
NextInspector61.ReadProperties(ParentNode);
```

Next Collection 6

Next Collection 6 package consists of dozen smaller, but handy components.

Table of Contents

NextStandard 6	2
<i>Shared Components</i>	2
NxColorScheme6	2
NxStyleSheet6	3
NextEditors 6	5
NxGridPicker6	6
NextGrid 6	7
<i>Design-time</i>	8
Views Editor	8
Columns Editor	10
Parts of Column	11
<i>Run-time</i>	12
Views in run-time	12
Columns in run-time	13
<i>Working with Rows</i>	13
Adding new rows	13
Managing rows	14
<i>Working with Cells</i>	15
Accessing Cells	15
Optimization	16
Selecting cells	16
Editing	18
Interactive property	19
Importing and Exporting data	20
Saving to and loading from XML	21
NextDBGrid 6	22
<i>Design-time</i>	22
<i>Columns</i>	22
<i>DataBinding</i>	23
<i>Cells</i>	23
NextInspector 6	25
<i>Design-time</i>	25
	29

BergSoft NextSuite 6 User Guide

<i>Run-time</i>	25
<i>Reading properties</i>	26
Next Collection 6	28