

Lycia Development Suite



Lycia II Theme Designer Guide

Version 2.2 – April 2014

Revision number 2.00



Lycia II Theme Designer

Reference Guide

Part Number: 022-022-200-014

Svitlana Ostnek, Elena Krivtsova

Technical Writers

Last Updated April 2014

Lycia II Theme Designer Reference Guide

Copyright 2006-2014 Querix (UK) Limited. All rights reserved.

Part Number: 022-022-200-014

Published by:

Querix (UK) Limited. 50 The Avenue, Southampton, Hampshire,
SO17 1XQ, UK

Publication history:

September 2011: Beta edition

December 2011: First edition

April 2014: Updated for Lycia 2.2

Last Updated:

April 2014

Documentation written by:

Svitlana Ostnek / Elena Krivtsova / Olga Gusarenko

Notices:

The information contained within this document is subject to change without notice. It is a stable document and may be used as reference material or cited from another document. If you find any errata or omissions in the documentation, please report errors at documentation@querix.com

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose without the express permission of Querix (UK) Ltd.

Other products or company names used within this document are for identification purposes only, and may be trademarks of their respective owners.

Table of Contents

INTRODUCTION.....	4
GENERAL OVERVIEW	5
<i>Theme Designer Invocation.....</i>	<i>5</i>
<i>Theme File</i>	<i>6</i>
<i>Apply_theme() Function Usage.....</i>	<i>6</i>
<i>Script Files Behaviour.....</i>	<i>6</i>
THE LAYOUT	7
<i>Styles View (Diagram).....</i>	<i>7</i>
<i>Properties and Elements View.....</i>	<i>8</i>
ELEMENTS.....	10
APPLICATION.....	10
WINDOW.....	10
MENU BAR.....	10
<i>Menu Command.....</i>	<i>11</i>
<i>Menu Group.....</i>	<i>11</i>
<i>Menu Separator.....</i>	<i>11</i>
<i>Popup Menu.....</i>	<i>11</i>
RINGAREA.....	11
STATUS BAR	11
TOOL BAR	12
<i>Toolbar Button.....</i>	<i>12</i>
<i>Toolbar Separator.....</i>	<i>12</i>
CONTAINERS.....	12
<i>Border Panel</i>	<i>12</i>
<i>Coord Panel.....</i>	<i>12</i>
<i>Grid Panel.....</i>	<i>12</i>
<i>Tab.....</i>	<i>12</i>
<i>Tab Page</i>	<i>13</i>
<i>Table.....</i>	<i>13</i>
OTHER ELEMENTS	13
<i>Blob Viewer.....</i>	<i>13</i>
<i>Browser.....</i>	<i>13</i>
<i>Button</i>	<i>13</i>
<i>Button Group.....</i>	<i>14</i>
<i>Calendar</i>	<i>14</i>
<i>Check Box.....</i>	<i>14</i>
<i>Combo Box.....</i>	<i>14</i>
<i>Function Field.....</i>	<i>14</i>
<i>Image.....</i>	<i>15</i>
<i>Label.....</i>	<i>15</i>
<i>Progress Bar.....</i>	<i>15</i>
<i>Slider.....</i>	<i>15</i>
<i>Spinner</i>	<i>15</i>
<i>Text Area.....</i>	<i>15</i>





<i>Text Field</i>	15
<i>Time Edit Field</i>	15
<i>Separator</i>	16
FILTERS	17
<i>ID Filter</i>	17
<i>Hierarchical Filters</i>	18
<i>Pseudo-Class Filter</i>	18
PROPERTIES	21
APPLICATION PROPERTIES	21
<i>Compatibility Mode</i>	21
<i>MDI Mode</i>	21
<i>Has Window Menu</i>	21
<i>Show Splash</i>	22
<i>Timeout</i>	22
<i>Font</i>	22
<i>No Scale Pixel Coord</i>	23
BACKGROUND COLOUR AND IMAGE PROPERTIES	24
<i>Background</i>	24
BORDER PROPERTIES	27
<i>Element Border</i>	27
<i>Border Brush</i>	28
<i>Thickness</i>	28
<i>Corner Radius</i>	28
COLOUR PROPERTIES	29
<i>Pre-defined 4GL Colours in fgl_profile</i>	29
<i>RGB Colour Definition for 8-Bit Graphics</i>	30
FORM AND FIELD PROPERTIES	31
<i>The Field Filter</i>	31
<i>Allow Newlines</i>	32
<i>Cursor</i>	32
<i>Tool Tip</i>	32
<i>Use Tabs</i>	32
GRID PANEL PROPERTIES.....	33
<i>BufferCount</i>	33
<i>ColumnLength</i>	33
<i>Grid Color</i>	34
<i>Image Collapsed</i>	34
<i>Image Expanded</i>	34
<i>Is Groupable</i>	34
<i>Is Filterable</i>	35
<i>Is Sortable</i>	36
<i>Is Virtual</i>	36
IMAGE PROPERTIES.....	37
<i>Image</i>	37
LAYOUT PROPERTIES	39
<i>Grid Panel Properties</i>	39
<i>Border Panel Item Location</i>	40
<i>Location</i>	40
<i>Padding</i>	40

<i>Cell padding</i>	41
<i>Margin</i>	41
<i>Visible</i>	42
<i>Z Order</i>	42
SIZE PROPERTIES	43
<i>Preferred Size</i>	43
<i>No Resize</i>	43
<i>Min Size</i>	43
<i>Max Size</i>	43
TEXT PROPERTIES	44
<i>Font</i>	44
<i>To Case</i>	45
<i>ForeColor Property</i>	45
TOOLBAR SETTINGS	46
<i>Toolbar visibility</i>	46
<i>Tool Tip</i>	47
<i>Toolbar Icons</i>	47
<i>Hide Labels</i>	47
<i>Toolbar Location</i>	47
WEB COMPONENT PROPERTIES	48
<i>Component Type</i>	48
<i>Component Path</i>	48
WINDOW SETTINGS	49
<i>Window Width and Height</i>	50
<i>Window Style</i>	50
<i>Title</i>	51
<i>Title Bar Options</i>	51
<i>Relative to Parent</i>	52
THEME DESIGNER TUTORIAL	53
THEME SYNTAX	54
<i>Basic Encoding Rules</i>	54
<i>Theme Code Elements</i>	55
<i>Style Action Syntax Peculiarities</i>	58
APPLYING PROPERTIES TO ELEMENTS	63
<i>Applying Global Properties to All The Form Elements</i>	66
<i>Applying Global Properties to Form Elements of the Same Type</i>	67
<i>Applying Global Properties to Form Elements Using Class Filters</i>	69
<i>Applying Global Properties to Form Elements Using Hierarchical Filters</i>	74
<i>Applying Properties to a Specific Form Element</i>	76
<i>Applying Properties to a Form Element Using the 'With ID ...' Filter</i>	77

INTRODUCTION

This document provides the readers with the detailed information regarding Lycia Theme Designer, Querix graphical tool. Program features description as well as their ways of usage are included.

The guide is intended for the software developers who create their applications using Querix thin-clients. Within this reference guide the following conventions are used:

Convention	Description	Example of usage
Boldface	Used for emphasis (1), dialog box and screen elements names (2,3) and names of the objects available in Theme Designer (4).	1. The unfolded variant of the diagram ... 2. Unfold the Set property group from the Diagram view. 3. Click the OK button. 4. The  Descendants ... filter refers to all the descendants ...
<i>Italic type</i>	Used for code sample variables, functions or methods in the description.	You can use the <i>apply_theme()</i> function.
Monospace	Used for code fragments or command line functions names.	<pre>DATABASE cms MAIN DISPLAY "Hello World" END MAIN</pre>
->	Used to show navigation from one menu selection to another.	Select Background -> Fill Color -> New custom color .
	Used in a table to provide additional information, non-trivial but important details, suggestions, prerequisites or indicates any warnings or possible errors.	All the sizes in the Theme Designer except for the font size should be given in pixels.

The document is a work-in-progress one. It cannot reflect all of the latest changes as some of the elements related to the advanced graphical forms are still in the process of updating. They will be described in further editions of this document.

Any comments, questions or ideas regarding this manual should be addressed to the Querix team via e-mail documentation@querix.com



General Overview

Lycia Theme Designer is a new graphical tool which can be helpful in managing your application view and functionality. It has an intuitive interface, that will help you to create and manage graphical styles for your 4GL application and is intended to replace the legacy script files, that have been used before.

Now, it is not necessary to remember all the script commands. You simply can chose the required element or property from the list carefully made up taking into consideration your needs.

Theme Designer Invocation

Theme Designer can be invoked in the ways described below:

1. as a standalone tool

Theme Designer is installed together with Lycia Studio. To open it go to Start -> Querix -> Lycia II Development Suite -> Lycia Theme Designer.

2. as a standalone tool from Lycia Command Line

You can invoke Theme Designer from command line using the '`LyciaThemeDesigner`' command. This command usage, without any attributes specified, results in Theme Designer invocation: in this case you will be able to save the changes made to a newly created file with the `.qxtheme` extension upon completion of working with the designer.

If you want to open an already existing `.qxtheme` file directly from the command line, please, specify the path to it after the command invoking Theme designer. Here is an example, how the theme file named "`myTheme`" can be opened with Theme Designer from the command line:

```
ThemeDesigner.exe myTheme.qxtheme
```

2. as a standalone tool from Lycia Studio

Besides, we provide you with an opportunity of invoking Theme Designer directly from Lycia Studio. To do this, select the `.qxtheme` file you want to open and double click it or use the File --> Open File... option of the main menu.



Theme File

By default, all the theme settings specified within the editor are saved to the file which has the same as the application *.exe* file and has the *.qxtheme* file extension.

The default theme file has the same name as the program and is located in the same directory as the program executable file. If any of these requirements is not fulfilled, a new default theme file is created automatically on the Theme Designer invocation.

Apply_theme() Function Usage

If you have a theme file with a ready set of settings which you would like to apply to your program, you can use the *apply_theme()* function, which takes the theme file as an argument.

```
CALL apply_theme("my_styles")
```

Script Files Behaviour

If you import a 4GL application with scripts to Lycia 2.2, the script files will remain in your project, but will be ignored and a new empty theme file will be created. The last becomes the default theme file used on Theme Designer invocation.



The Layout

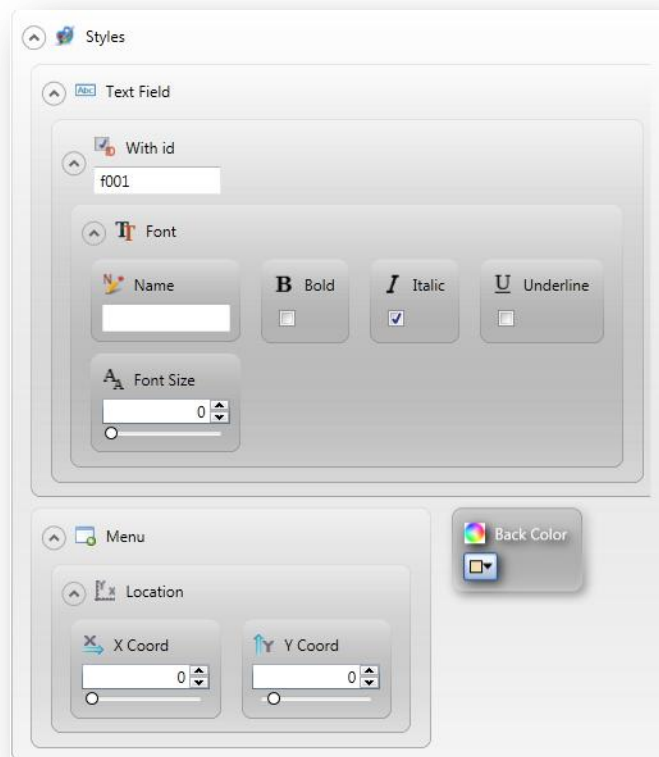
The Theme Designer has a user-friendly graphical interface which allows you to make and trace the changes right away.

Styles View (Diagram)

The Styles view contains all the properties applied to the current program by the opened themes file. The properties are typically grouped by the element objects. The elements, in their turn, allow some nesting.

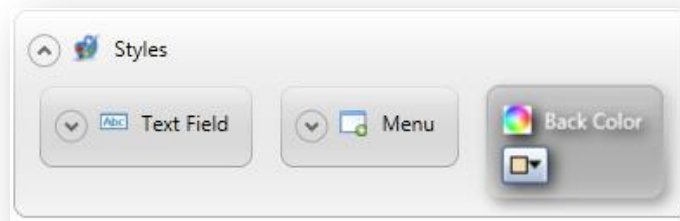
The groups and subgroups can be folded and unfolded in order to keep the work space neat and easy to deal with.

The **unfolded** variant of the Styles view:





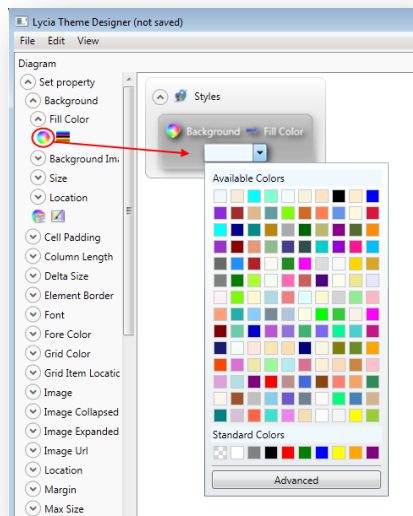
The **folded** variant of the Styles view:



Each group or object can have a number of nested properties and elements. They are added to the view either using the list on the left, the Form Tree view, or the right-click context menu of the Styles view.

Properties and Elements View

The Styles view includes the list of properties and elements on its left. These are all the possible properties, that can be applied to an element.



To select a specific property unfold the **Set Property** group and then expand the desired property group until you get at the icons of this property.







The same way a required element can be selected. To add an element to the Styles view, you need to unfold the **Elements** group and drag it to the diagram within the Styles area. In order to apply a property to this element choose the one you need from the Set Property group.

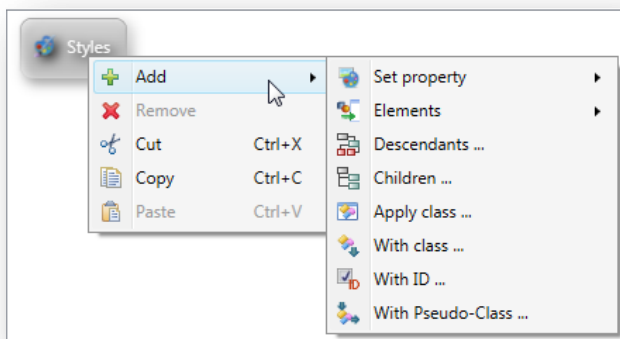


Properties applied to an element, selected in such a way, will affect all the objects of the specified type (e.g. all text fields, all radio buttons, etc.)



There is also a number of objects (filters), which are not categorized and placed under the Elements group list. They are typically used for object filtering and represent some specific features:

Filter Name	Description
 Children ...	used to indicate that the properties will be used for all the children of the object (e.g. if menu is an object, menu buttons are its children).
 Descendants ...	used to indicate that the properties will be used for all the descendants of the object (e.g. if window is an object, all the forms and their widgets will be the descendants)
 With class ...	used to set a number properties in order to apply them at once for a definite element or group of elements
 Apply class ...	used to apply a set of predefined options to a number of objects
 With ID ...	specifies that a property will be applied only to the object with the defined ID
 With Pseudo-class ...	allows to apply some specific features to form elements, depending on their current state



Besides, all the properties, elements and filters are available from the Styles view: use the 'Add' option of the contextual menu (right click on Styles).



Elements

This chapter contains the description of the form elements, that can be modified in Lycia Theme Designer by applying properties to them.

Some of these elements are the graphical ones, and can be dealt with in Form Designer (form tabs, form fields and widgets, etc.) Some of them can be influenced by the 4GL code (menu buttons, message windows, etc.) Besides, there are elements added automatically or constituting parts of other elements (scroll bar, status bar, dropdown list, etc.)

Application



A 4GL application object can have a set of specific properties, that influence the performance of the application as a whole. These properties are described later in this manual.

Window



In 4GL there are several types of windows. There are flat windows and windows that have border. Both these window types are filtered indiscriminately using the Window filter. If a property is applied to the Window filter, it will affect both types of windows. To apply a property exclusively only to flat windows or only to bordered windows, use the [Pseudo-Class](#) filter.


Menu Bar




Menu Bar is an element of a window consisting of a set of options each including one or several commands which are executed when the user chooses the corresponding option. This filter represents the menu area including the background between the menu buttons and the area around them. This element and its corresponding sub elements refer to the Menu created using the form file, it does not apply to the classical 4GL ring menu.



Menu Command

 A visual representation of a menu option which contains the option name and description, and activates the menu option after the user clicks the mouse on it or selects and activates the button from the keyboard.


Menu Group

 A Menu Group is a set of menu commands grouped under one menu item. The menu group is displayed as a menu command which calls a sub-menu.

Menu Separator

 A menu object which is used to visually separate groups of menu buttons from each other.


Popup Menu

 A menu that is called by a right mouse click and contains a list of options available in the current context.

RingArea

The ring area is an element that references the classical 4GL ring menu and its elements. If you want to change the position of a ring menu, you should use the Ring Area filter rather than the Menu Bar filter.


Status Bar

 An area in the screen or window, to where the application displays messages describing the current state of the program execution (for example, error messages).


To configure different Status Bars views the corresponding class should be applied to a Window form widget. For the list of the classes available, please, refer to Appendix B: Classes, Querix 4GL Reference Guide.



Tool Bar

 An element in the screen or window which contains a set of buttons, each of them invoking some actions. Properties applied to this filter will not affect the toolbar buttons themselves, but only the area where the toolbar is located.

Toolbar Button


 Toolbar button is a button on the toolbar, the click on which is associated with a key press or action and is followed by corresponding program actions. Toolbar button, as a graphical object, may contain a label, an icon, or both, and a tool tip. All these elements can be easily modified by means of the Theme Editor.

Toolbar Separator


 A toolbar object which is used to visually separate groups of toolbar buttons from each other.

Containers

Border Panel

 A form container, in which the elements are automatically placed along the borders. The new widget takes all the available space along one of the borders.

Coord Panel

 A form container where the location of the objects is determined by x and y coordinates which specify the location of the object top left corner on the horizontal and vertical axis, respectively.

Grid Panel


 A form container where the objects are placed within a grid - one widget within one grid cell.

Tab

 a page of the form which contains a set of elements, independent of those in the other tabs.



Tab Page

 The area of the form tab, on which the form elements are located.

Table


 A form element, displayed as a grid, each cell of which can be used for input or display. Usually this is either a screen array or a screen grid.

Table Cell


 An element of the table which can be used to display or input one value

Table Column




 A set of table cells, located one under another and having the same properties.

Table Row


 A set of table cells, located one after another in a row, which can be linked to a program record or a program array element.

Other Elements


Blob Viewer

 A form widget which displays the values of the BYTE variables. Typically this is a text field of the BYTE or TEXT data type.

Browser

 A form widget containing a web browser which allows the user to access internet pages from the application.

Button

 A form widget, activated by a mouse click and sending a keypress or an action to the 4GL application.



Button Group



A radio button widget, including each option available in it. Properties applied to a button group filter affect the area where the radio button options are grouped, but not the options themselves. For example, setting the background for the button group will apply the background to the area beneath the radio buttons and their labels whereas setting the font colour will have no effect, because the labels belong to the radio button options.

Radio Button



A single option of a radio button group. Properties applied to a radio button filter will affect the radio buttons themselves and their labels and not the area in between and beneath the labels. For example the foreground colour property will change the colour of the labels and the background colour property will change the shade of the radio buttons but not the colour of the area beneath the labels.

Calendar



A form widget used to display and input dates with a built-in drop-down calendar for facilitating the input.

Check Box



A form widget used to make a choice between two options. The checkbox passes one of the two values to the program - one for checked state and the other for the unchecked state.

Combo Box



A form widget which provides the user with the ability to choose from a list of values and to add a new value to this list, if needed.

Function Field



A form widget which consists of a field used for text input and display and a button attached to it. In Theme Editor this element is subdivided into a button element and a text field element.



Image



A program element used to display an image. You can allow the image size changing by applying the Image scaling property.

Label



There are two types of labels, both of them look like a text string on a form or in a window. The static labels are created using static label form widget or the DISPLAY ... AT statement. The dynamic labels are created with the dynamic label form widget. The Theme designer offers a filter which fits the both types of labels. Any properties applied to this filter will affect both static and dynamic labels. To apply properties exclusively to static or dynamic labels, use the [Pseudo-Class](#) filter.

Progress Bar



The Progress Bar is a widget used to display a numeric value within the specified range. It is typically used to display graphically the level of progression of some process.

Slider

The Slider widget is designed to allow graphical input of numeric data. It represents a space with a slider inside which the user can move thus selecting a value from the previously specified range.

Spinner

The Spinner widget is designed to allow graphical input of numeric data within a range specified in the widget settings. The field includes a spinner with a pre-defined step which allows the user to set the necessary data by clicking up and down keys.

Text Area

A form widget similar to the text field which can contain several lines of text.

Text Field

A form widget that contains one line of text

Time Edit Field

A form widget designed for convenient input of time data.



Separator

The Separator object is used to visually separate groups of toolbar or menu buttons.




Filters

We have briefly described the structure and the interdependency of form elements, the properties of which can be changed with Lycia Theme Designer help.

There are several ways to assign properties depending on the filter you choose.

Name	Description
With ID filter	<p>You can apply a property to any individual object within the application.</p> <p>To do this you need to add an element to the Styles view. Then place the With ID filter inside of it.</p> <p>In the With ID filter field type the ID of the element which you want to use, note that the element should belong to the selected element type.</p> <p>E.g. if you selected the button element filter, the With ID filter inside it can only contain the ID of a button widget, otherwise it will be ignored. After doing this, put the property inside the With ID filter and it will be applied to the selected object.</p> <p>The object IDs can be looked up in the Form Tree view.</p>
Hierarchical filters	<p>It is possible to apply an option to a specific objects or an object type and all its children or descendants.</p> <p>The Children and Descendants filters are used for this purpose.</p>
Class Filters	<p>You can also prepare a set of properties in a kind of a template and then apply them to any object or group of objects.</p> <p>To do this use the Apply Class... and With Class... filters.</p>
Pseudo-Class Filter	<p>Some of the filtering is impossible to do by means of the filter types listed above.</p> <p>This is usually true for the cases when you want the buttons to change colour when the mouse cursor is hovering over them, or when you want to filter all the messages displayed by means of the MESSAGE statement which makes them static labels by nature and means they will be normally filtered together with the rest of the labels.</p> <p>For such specific cases the Pseudo-Class filter is usually used.</p>

ID Filter

The  **'With ID ...'** filter allows you to apply properties to a specific program element. This filter has a field, where the element`s ID must be specified.





The ID of the element required can usually be found in the Form Tree view. In the majority of cases, it is defined by the programmer. The ID of a field is the field name, specified in the form file; the window ID is the name, given to a window when it is opened, or its title, when it is opened by a built-in function, etc.

If this filter is placed into the Styles view as a root element, without being included into any other object (filter), the specified properties will be applied to all of the form elements with the specified ID. If an application contains a window defined as 'f001' and several forms have a form field with the same name, all these elements will be affected. To avoid ambiguity, use unique names for each of the form element.

To apply properties to a single specific element, place the '**With ID ...**' filter within other filters. The 'With ID ...' filter inside the Window element ensures, that the properties will only be applied to the windows with the specified ID. And no other form elements will be influenced by them, even if their names coincide.

Hierarchical Filters

With the help of the  **Children ...** filter, you can apply the properties to all the children of the specified element. Children refer to immediate descendants of the first order. For example, the children of a form element will be the Tool Bar, Status Bar and Content element, but not the form elements, grouped under the Content element and are the descendants of the second-order.

The  **Descendants ...** filter refers to all the descendants of the specified element. For example, the descendants of a form are Tool Bar and Status Bar as well as all the form widgets. The descendants can be viewed from the **Form Tree** view.

Pseudo-Class Filter

A pseudo-class option allows you to apply some specific attributes to the elements of your program only when a specific condition is met. The Pseudo-class filter filters elements based on the selected condition and not on the element type, ID, or element relations. Thus the properties will be applied only to certain elements, if they meet the condition.

These are the conditions available which can be selected for a pseudo-class property:



Name	Description
Focus	filters all the widgets when they have the focus (e.g. the field where the cursor is located during the execution of an INPUT statement)
No Focus	filters all the widgets when they do not have focus (e.g. the fields during the execution of an INPUT statement where cursor is not located)
Hover	filters all the elements when the mouse cursor is moved over them
Inactive	all the widgets when they are not active (e.g. fields that are not currently in the input state)
Active	filters all the widgets when they are active (e.g. fields available for input during the execution of an INPUT statement)
Query	filters all widgets when they are used in the CONSTRUCT statement
Display	filters all widgets when they are used in the DISPLAY ARRAY statement
Input	filters all widgets when they are used in the INPUT or INPUT ARRAY statement
Prompt	prompt line (the prompt message and prompt field) during the execution of the PROMPT statement
Even	filters all the widgets in an even row if an array
Odd	filters all the widget in an odd row if an array
Message	applies only to text displayed with the MESSAGE statement
Error	applies only to text displayed with the ERROR statement
Dynamic Label	filters all dynamic label widgets on all forms
Static Label	filters all static labels created as static label widgets and using the DISPLAY



	... AT statement
Is Protected	filters all widgets that are protected from overwriting. All form widgets are usually protected, except for the labels
Border	filters all the windows that have border attribute
No Border	filters al the flat windows that do not have the border attribute
Form	filters a form with the specified root container
PrintableWidgets	filters the widgets to which input and display can be performed (TextField, TimeEditField, TextArea, Spinner, Calendar, ComboBox and ListBox)
GreyableWidgets	filters the widgets that can be greyed when inactive (TextField, TimeEditField, TextArea, Spinner, Calendar, ComboBox, ListBox, Button)



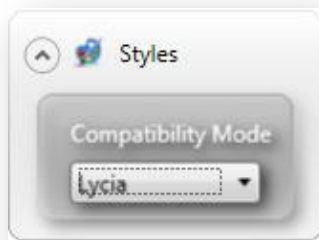
Properties

This chapter contains all the properties available in the Theme Designer grouped by elements they can be applied to or by the effects they produce.

Application properties

Here is the list of properties that can be applied to the Application elements thereby influencing the whole application appearance.

Compatibility Mode



The Compatibility Mode property is used for the compatibility purposes.

Theme Designer allows setting one of the three possible values for this property:

- Lycia
- Informix 4GL
- GBDS

By default, an application is run in Lycia compatibility mode.

MDI Mode



The client will render all applications within an MDI container, as opposed to the default SDI mode. To enable the MDI mode place this property within the editing area and check it.

By default, the application container is maximized when run in the MDI mode. The size of the container can be modified by means of the frame size properties.

Has Window Menu

On this property setting an additional top menu item named Window, showing all the child applications running at the moment, appears on a form. The property takes effect only upon the condition that the mdi mode is set for an application.



Show Splash



The Show Splash property is used to determine, whether the splash screen is used while an application is loading.

This property is highly recommended to be added amongst the first ones, and used after the relevant image has been sent to the client.

Timeout



The Timeout property is used to define the time after which the application will terminate, if the program is not responding.

For example, the program can stop responding, if the network connection suddenly goes down or if the database query takes too much time. If this option is not included into the theme file, there is no timeout and the application will run until it is stopped explicitly.

The Timeout value should be supplied in milliseconds.



The SLEEP statement can also trigger the activation of the Timeout parameter, because when the program is suspended by this statement is it not responding. Be careful, if you use this option for a program which contains SLEEP statements.

Font



The Font property when applied influences the scaling of an application: sizes of all the form widgets are adjusted to the font size assigned.

Besides, this property can be used to set the font for a message box, invoked by `fgl_winmessage()` within the 4GL source, for example. To do this, the font property should be assigned to an application element, used in its turn as a selector. Below is an XMLtheme code extract illustrating how the font (Verdana, 8) can be set for the message box:



```
<ElementFilter ElementName="Application">
  <StyleSheet>
    <DoStyleAction>
      <SetProperty>
        <PropertyPath>
          <PropertyName>Font</PropertyName>
          <PropertyName>Family</PropertyName>
        </PropertyPath>
        <PropertyValue>
          <Name>Verdana</Name>
        </PropertyValue>
      </SetProperty>
    </DoStyleAction>
    <DoStyleAction>
      <SetProperty>
        <PropertyPath>
          <PropertyName>Font</PropertyName>
          <PropertyName>FontSize</PropertyName>
        </PropertyPath>
        <PropertyValue>8</PropertyValue>
      </SetProperty>
    </DoStyleAction>
  </StyleSheet>
</ElementFilter>
```

No Scale Pixel Coord



The objects in the graphical form are located within an abstract grid with cells sized to fit one default font character (Courier New, 13 pt).

This grid is resized automatically, if the font face/size is changed.

The unchecked No Scale Pixel Coord property adjusts the form abstract grid to the customized font size, thus resizes respectively the form itself and the form objects.



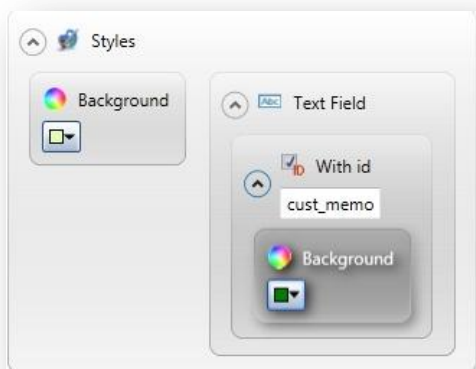
Background Colour and Image Properties

These properties are used to change the appearance of the background of a window. A background can be of a specific colour and it can also include a background image. Colour and image can be used separately or together.

Background

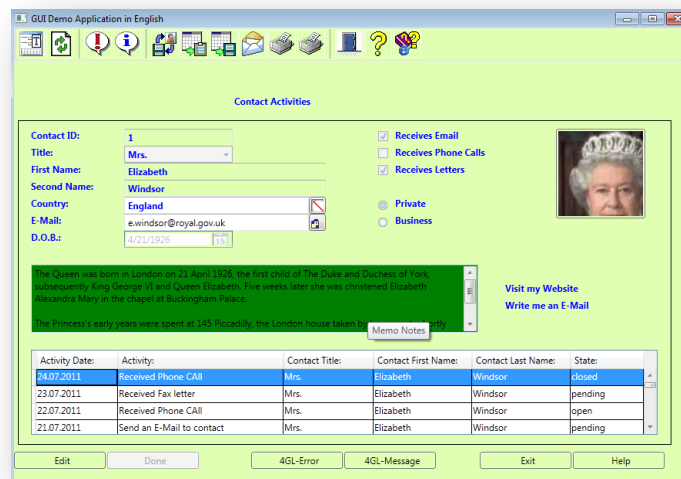


The Background colour property specifies the colour of the background of one or several objects. Usually, it is applied to forms and windows.



As it can be seen from the screenshot, a background is specified as light green for the whole application and as darker green for the text field *cust_memo*.

As a result, the GUI demo window looks as follows:





Background Image



To apply this option, the image ID which refers to the image file you want to use as the background must be specified first.

Background Image Uri

This property is the one specifying an image location. It is a string consisting of a file name with an extension, preceded by one of the following depending on the place where the file is located:

- ***qx://embedded/*** specifies, that an image is an embedded one (is installed together with one of the GUI clients and is located inside them;
- ***qx://application/*** specifies the path relative to an application on an application server;
- ***qx://clienttemp/*** specifies the temporary directory on the client's side.

For example, it may look as follows:

```
qx://application/image/lycia2_logo.png
```

where *image* stands for the folder on the application server. If the image is located in the application server root folder, this part should be omitted or modified according to the path to the image.

An image specified this way can be used as a background of a whole window or of a separate element.

Background Style



Sets the way the border of a background image is displayed. There are several possible options:

- **Normal:** the border image is displayed as it is without adjustments.
- **Stretched:** the background image is stretched along the border length. The image ratio may be distorted in this case.
- **Tiled:** the background image is repeated along the border length.



- **Centered:** the background image is located in the centre of the border. If the image is smaller than the background area, the area around it is filled with the background colour; if the image is larger than the background, its sides are truncated.
- **Uniform:** the image covers the whole background area without the image ratio being distorted. Some margin are added to the image.
- **Uniform to fill:** the image covers the whole background area without the image ratio being distorted. No margin are added.

Size

This property specifies the size of the area occupied by the image. The resizing of the image stretches and shrinks it.

If the property is not set, it is considered that the image occupies the whole background of the element it is assigned to, even if the actual image is smaller and cannot take up the whole area. This happens because the image is treated as the background of the element.

The options available for this property are New Size, Width and Height. The first one allows both the width and height adjustment, to change the width and height separately the corresponding options must be used.

Location




This property specifies the location of the area where the image is situated. Since the Background Style property mainly deals with the image location, this specific property works as an extension which covers the locations impossible to set using the style option.

Image location property should be only used together with the Image Size property. Otherwise, it is considered that the image occupies the whole element background and the whole background will be moved to the new location, if the location coordinates are changed, that may lead to the unwanted visual effects.



Border Properties

The Border property allows to set the border for almost any element available in Theme Designer, e.g. for a toolbar button, a form widget, etc.

Border () is a group property which sets specific properties of the parent element border. To create a border for any object you need to set the following properties as the minimum requirement:

- Border Theme
- Thickness

Other properties are optional and their effect in some cases depends on the border theme you've selected. For example to be able to see a line border you need to add the border brush that will change the colour of the border. On the other hand, a raised etched border does not need colour to be visible.

Element Border




The Element Border property specifies the border appearance. It contains a list of four possible options:

- **New Line border:** the object is surrounded by a line of the given colour and thickness. Needs the Border Brush applied to be visible.
- **New Etched border:** a thin border around the object which is raised or lowered a little above the background. Does not need Border Brush applied to be visible.
- **New Bevel border:** a thick border raising or lowering the object itself a little above the background. Does not need Border Brush applied to be visible.
- **Is Raised:** The property which specifies whether the bevel or etched border is to be raised (checked) or lowered (unchecked).


The most common border theme is the line border when the outline of an object is just represented by a simple line of the given thickness and colour.



Border Brush


 Border brush is an option which specifies the colour of the border. As with any other colour option, you can select from system colours or chose a custom one. If you do not specify the border brush, the border will be invisible, if its border theme is a 'Line Border' one.

Thickness

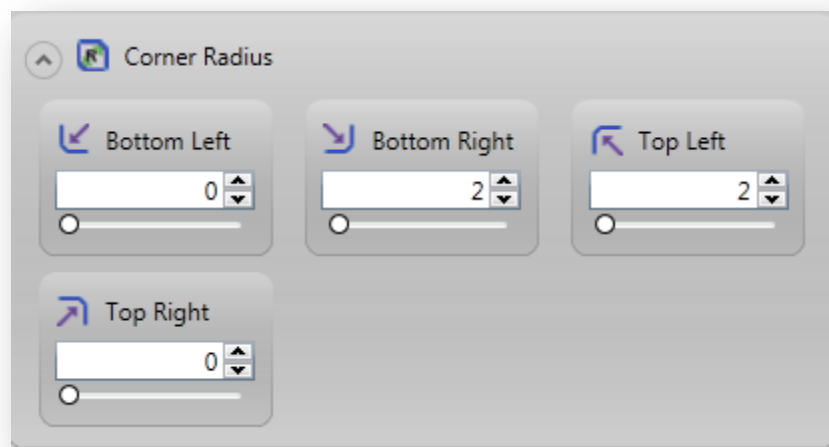
 This property sets the border thickness.

It has four options: Left, Top, Right and Bottom, and you can correspondingly specify the thickness of each border side in pixels either by moving the slider to a predefined value (from -1 to 100) or by entering your own value into the provided text field.

Corner Radius

 The Corner Radius property option specifies the radius of border corners rounding. It is optional and is used only for adjusting the border appearance.



The Corner Radius property contains four options: bottom left, bottom right, top left, top right, each used for specifying the rounding radius (in pixels) for the corresponding corner.

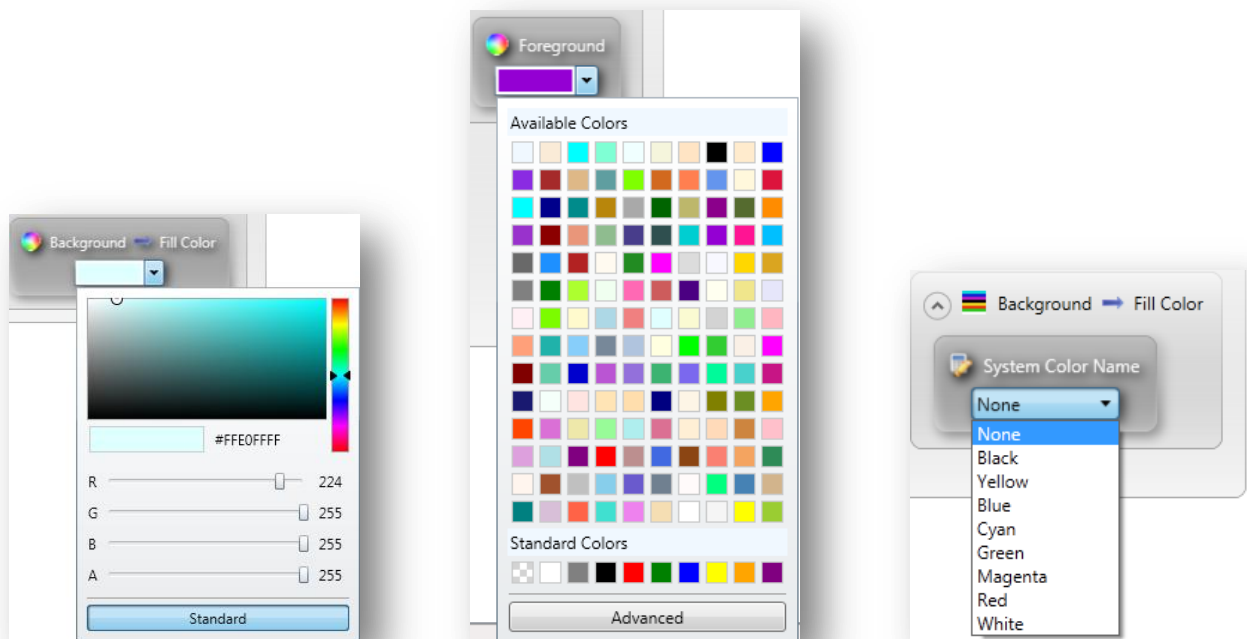




Colour Properties

The colour property can be applied to any graphical element displayed by the GUI client. It sets the colour for the specified element or for the entire application. You can either apply a background colour (the Background Color option) or the foreground colour which serves as the font colour (the Foreground Color option).

You can select either one of the 7 standard 4GL colours () or a customized colour () with a large palette. The last provides the possibility of entering the html colour code.



The same features can be applied to all the dealing with colour properties.

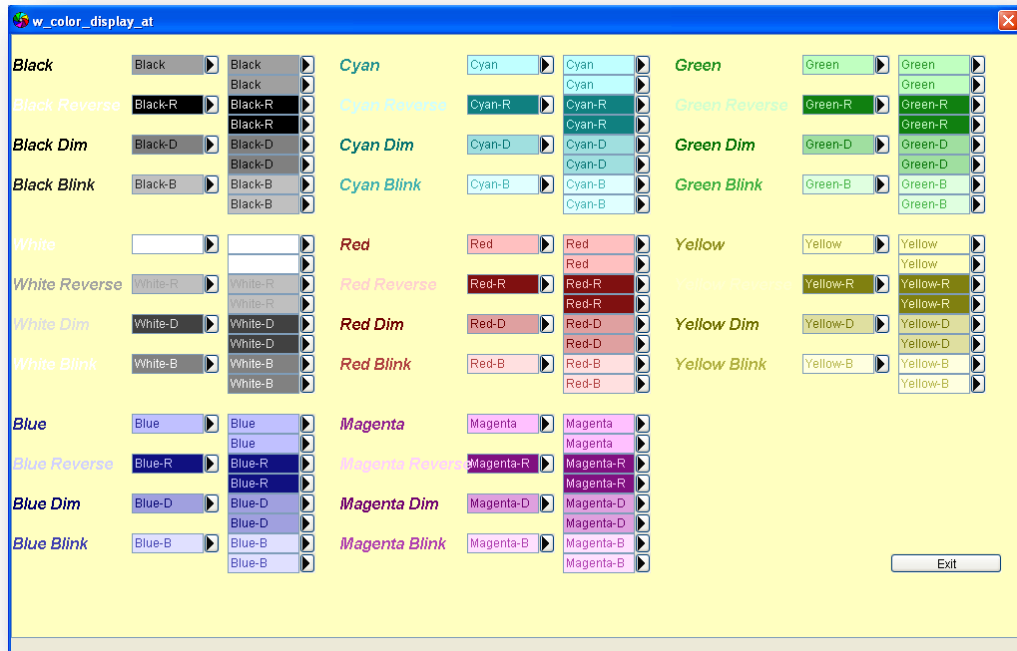
In the Advanced section of the colour picker the colour code needs to be given in 8 symbols, 2 first ones specify the degree of the colour transparency. Thus if the first two symbols are FF, the colour will be non-transparent, of they are 00, the colour will be completely transparent and not visible.

Pre-defined 4GL Colours in fgl_profile

The Querix development suite defines eight colours of the 4GL language in a modern style. Their definition is stored in the fgl_profile file located on the server and can be changed manually when



needed. Besides, there can also be defined additional colours which become available for all the applications connecting to this server.



RGB Colour Definition for 8-Bit Graphics

The table below shows the RGB values of the traditional 8-Bit colours. This can be useful if an application is accessed by graphical client terminals which support only 8-Bit colours.

Black #000000	Gray #808080	Maroon #800000	Green #008000	Navy #000080	Purple #800080	Olive #808000	Teal #008080
White #FFFFFF	Silver #C0C0C0	Red #FF0000	Lime #00FF00	Blue #0000FF	Magenta #FF00FF	Yellow #FFFF00	Cyan #00FFFF



Form and Field Properties

The form object, which is always a child object of the window object, represents a single 4GL form being displayed within a window. It is primarily used as a container for other objects representing 4GL objects and user control objects existing within the form.

The name of a form used as its ID, is based on the name under which the form was opened in the 4GL code. This is the name given to the form in the open form statement, and not the name of the form file:

```
MAIN
OPEN FORM f_myform FROM "f_myform_file"
# Open a form, which will be called "myform" in
# the thin-client
DISPLAY FORM f_myform
CALL fgl_getkey()
END MAIN
```

If the form is opened with an OPEN WINDOW WITH FORM statement, then the name of the form is identical to the file name of the form:

```
MAIN
OPEN WINDOW w_mywindow AT 7, 3
WITH FORM "f_invoice_details"
ATTRIBUTE (BORDER, YELLOW)
# Open a window with a form.
END MAIN
```

The Field Filter

A field object is an object which represents any input field within a form including graphical widgets such as dynamic label, button, bmp_field, combo box and image which are also classified and defined as fields. They are defined in a form file in the Attribute section (in a .per form) or added to the form template (in a .4fm file)



Allow Newlines



The Allow Newlines property, when checked, allows the text within the widget to be split into lines in case the string displayed or inputted to it is too long to fit one line.

Cursor



Sets the type of the cursor when it moves to a selected element such as a text field. The following options are available:

- Arrow
- Cross
- I beam
- Size All
- Size NESW
- Size NS
- Size NWSE
- Size WE
- Up Arrow
- Wait Cursor
- Help
- H Split
- V Split
- Hand

If this option is applied globally, the cursor appearance changes within the whole application.

Tool Tip



The Tool Tip property allows you to set the tooltip used to describe a field or widget.

LyciaDesktop uses tooltips to display the comment information that is provided in the form file. If this information is not appropriate for a GUI environment, it can be replaced with a user defined text, which is to be used as the tooltip for this specific field.

To add or change the form field tooltip, the Tool Tip property must be added to the corresponding form widget or toolbar button with the tooltip text.

To disable a tooltip, the text field must remain empty.

Use Tabs



The UseTabs property is applied to a TextArea widget.




When set to true, it allows the user to insert the tab character to the field value. It means, that if the property is set and the user presses the Tab key when the pointer is within the text area, it does not move to the next field, but a tab space is added to the field.



Grid Panel Properties

Most forms in 4GL programs contain screen arrays and grids for convenient information display and input. Therefore, it is obvious, that the graphic representation of these elements should be as convenient as possible.

Screen grids can be referenced using the following objects:

- Table 
- Table Row 
- Table Column 

BufferCount



The BufferCount allows you to control how many pages of table rows are to be kept at once in the application buffer during the DISPLAY ARRAY and INPUT ARRAY. Here, a page means the number of rows in the screen array to where the display or input is performed. The number passed to the property sets the number of the pages, that to be filled with data stored in the buffer. The data outside this scope are retrieved during the scroll and replace the rows, that are already stored.

For example, if a screen grid has 5 rows to display data, and BufferCont property is set to 3, it means, that only 15 rows of the data retrieved by application is stored in the buffer. The set of this rows can be changed during the scrolling. This allows to minimize the memory used during the application execution, especially when it is intended to work with large arrays of data.



The Buffer Count property has an effect only when the Is Virtual property is checked.

ColumnLength



The Column Length property is used to specify the width of a table column. The property includes two sub-properties: Grid Length Type and Grid Length Value.



Grid Length Type

Grid Length Type specifies the way the grid length value is interpreted. There are three options available:

- **Relative:** uses the relative scale. E.g, if the relative grid length for the columns of a three-column table is set to 3, 5, and 10, it means, that the columns ration will be 3:5:10
- **Absolute:** the length is specified in pixels
- **Automatic:** the program automatically sets the more suitable type

Grid Length Value

Grid length property is used to set the column length value in units, that are processed by the Grid Length Type property.

Grid Color



The Grid Color property is used to specify the colour of a screen grid sheet.

Image Collapsed



Image Collapsed property includes Image URL sub-property which specifies the path to the image, that is to be displayed next to a collapsed tree wrapper item. The property has effect, when a tree view wrapper property is applied to a screen grid.

Image Expanded



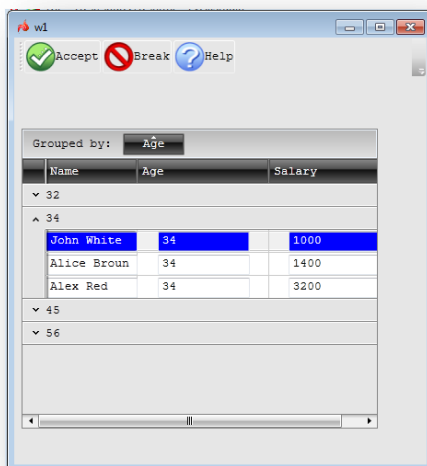
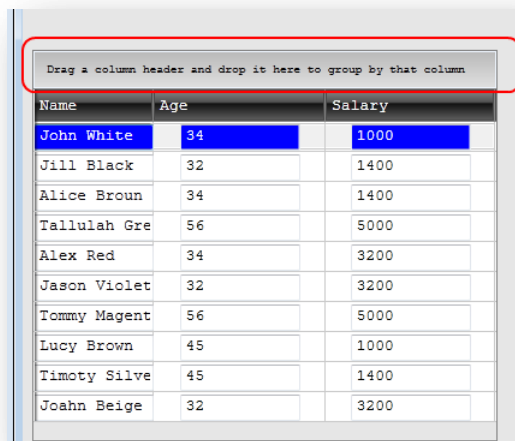
The Image Expanded property includes Image URL sub-property which specifies the path to the image that is to be displayed next to an expanded tree wrapper item. The property has effect when a tree view wrapper property is applied to a screen grid.

Is Groupable



The Is Groupable property allows to group the data in the table according to the repeating values of the column in which the property is set to TRUE.

When the property is set to true at least for one column, a group line is added to the whole table in the runtime.



To group the data in the table, drag and drop the header of the column with the Is Groupable property set to true. This changes the table layout: the table displays the group names only, each name identifying the value, that became the basis for the group creation. Click the group line to unfold it and see the entries containing the grouping value:

To cancel the grouping, drag the group name in the sorting line and drop it to the table.



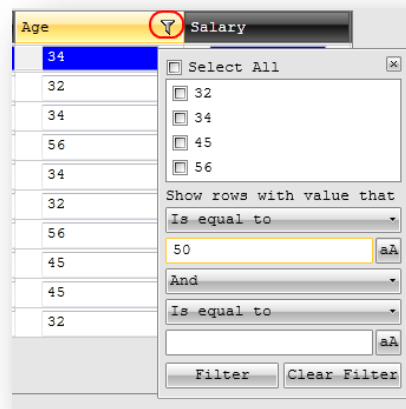
The Is Groupable property is applied only when the column headers are specified.

Is Filterable



The Is Filterable property, when checked, allows to filter the values displayed to the table, according to some criterion. At runtime, there is a Filter icon in the header of the column with Is Filterable property specified to true.

When the user clicks the icon, the Filter dialog box opens. Here one can input the filter criteria:



When the filter is applied, the table displays only the values in the selected column that correspond to the filter criteria.



The Is Filterable property is applied only when the column headers are specified.

Is Sortable



When the Is Sortable property is applied, the user can click the column header to sort the values in the table. The first click sorts the values in the ascending order, and the second click in the descending one. The third click resets the sorting and makes the lines be displayed in the default order.

The sorting changes only the visual display of the table and does not influence the order of records in the program array.

Is Virtual



The Is Virtual property, when checked, makes a grid virtualized, which enables the Buffer Count property to be applied. By default the property is unchecked.



Image Properties

The Image properties are used for several purposes. E.g., to replace an icon with a toolbar button or a normal button with a custom image, to add an image to a widget, etc.

This property must not be confused with the Background Image property: the Image property can not apply the image to the background of objects, and not all the objects have the foreground where it can be applied (e.g. text field). Therefore, in some cases, the Image property may have no effect when applied.

Image



This property can be applied to any element either already having an image or without any images. If the element already has an image or icon (e.g. a toolbar button with an icon), this image is replaced, if you add the Image ID property. Image Size property can change either the size of the default image assigned to the object or the size of the image assigned by the Image ID property.

Images can be applied to the following objects:

- the ring menu buttons;
- program context menu (invoked by right-clicking on the application area);
- toolbar, but not the individual toolbar buttons;
- the following form widgets:
 - button;
 - image;
 - Function Field button image.

Image URL



Defines an icon or an image to be used with an element. It has only the only option - URI, indicating an image location. The requirements to its specification are the same as for the [Background image property](#).

The default image size:

- The widget (i.e. a button widget) cannot be resized to fit the icon size by default. The image added in such a way is not scaled and is presented in its original size, so, if the image is larger than the widget it is applied to, it will be cropped. The image size can be corrected using the Size property.



- Images applied to a toolbar button resize all the buttons of the toolbar to the image size.
- Images applied to ring menu buttons are resized (usually shrunk) to fit the button.

Size



As it was mentioned above, an icon added to a widget remains of its original size in most cases, unless resized. The Icon Size property allows you to set the height and width of the image in pixels. This, for example, may be useful for bringing all the toolbar buttons to the same height.



To change the size of the toolbar icons, the Size property should be applied only to the whole toolbar using the toolbar object.

Individual toolbar buttons cannot have different size, so if you try to change the size of a single button, all buttons will be resized.



Layout Properties

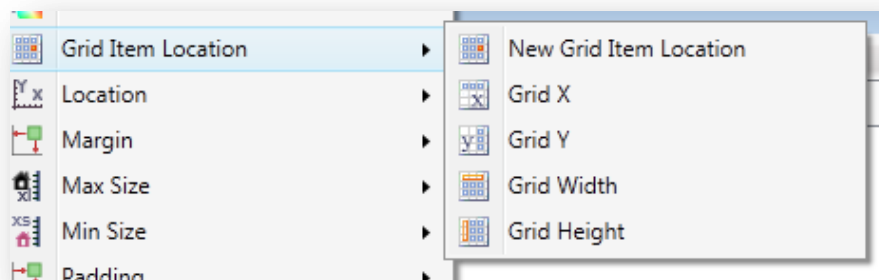
Besides modifying the separate form elements appearance, Theme Designer offers you the possibility of influencing their layout. It supports a number of properties which can help you to manage an object position, the distance between elements, etc.



All the sizes in Theme Designer except for the font size should be given in pixels. The scale of values varies from '1' to '100'.

Grid Panel Properties

The Grid Panel properties can be applied to the objects placed within the Grid Panel container.



Grid Item Location

GridItemLocation property indicates the place and the size of an object, located within a Grid Panel container. It provides five options described below.

Grid Height

Grid Height property sets up the number of the grid rows that the object should occupy.

Grid Width

Grid Width property specifies the number of grid columns that the object should occupy.



Grid X

Grid X - The index of the grid column in which the top left corner of the object is to be placed.

Grid Y

Grid Y - The index of the grid row in which the top left corner of the object is to be placed.

Border Panel Item Location



The Border Panel Item Location property specifies the location of a border panel item relatively to the border. There are five possible options:

- **Top** places an item along the top border;
- **Bottom** places an item along the bottom border;
- **Right** places an item along the right border;
- **Left** places an item along the left border;
- **Center** places an item in the centre of the grid panel.

Location



The Location property sets the position of the selected element within its parent element along the coordinate axes. It provides two options:

- **X coord** defines an element position horizontally in relation to the left edge of the parent element;
- **Y coord** defines an element position vertically in relation to the top edge of the parent element.

Padding



The Padding property sets the distance between the borders of a parent and child elements.

Four options are available:

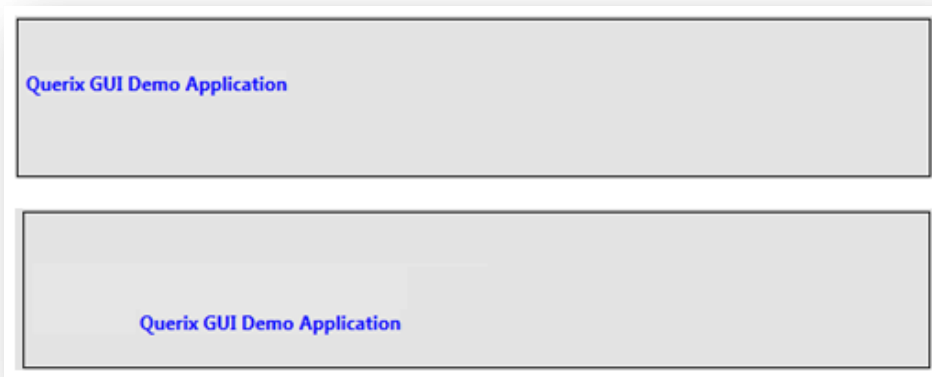
- **Left** defines the distance between the left borders of a parent and a child elements;
- **Right** defines the distance between the right borders of a parent and a child elements;
- **Top** defines the distance between the top borders of a parent and a child elements;



- **Bottom** defines the distance between the bottom borders of a parent and a child elements.

A value to any of these options can be assigned either by moving the slider to a predefined value or by entering the required value into the text field within the property object.

Below are two screenshots illustrating an application of the Padding property: in the first one the padding for the text in the box is not specified, while the second one shows how the left and top padding with the values '70' and '30' respectively effects the form appearance.



Cell padding



Defines the distance between the contents of an element to its border.

For example, from the contents of a text field to the field border. The property has four options:

- **Left** defines the distance between the leftmost symbol within an element and its left border;
- **Top** defines the distance between the top of the element contents and the its top border;
- **Right** defines the distance between the rightmost symbol within an element and its right border;
- **Bottom** defines the distance between the bottom of the element contents and its bottom border.

Margin



Defines the distance between the borders of a child and a parent elements.

For example, the distance between the form border to the border of the 4GL window, to which the form is displayed. The property has four options:




- **Left** defines the distance between the left edges of a child and a parents elements;
- **Top** defines the distance between the top edges of a child and a parent elements;
- **Right** defines the distance between the right edges of a child and a parent element;
- **Bottom** defines the distance between the bottom edges of a child and a parent element.

Visible

 Used to make an element visible or invisible if checked or unchecked respectively.

When the Visible property is applied to a form element which has child objects, they also become invisible, even if their visibility property checkbox is ticked.

Z Order

 The ZOrder property determines which element is atop if two or more objects overlap.

For example, a static label created by the DISPLAY statement can overlap form widgets or another static label. In this case the property should be assigned to each element.

An element with a larger Z order index is atop: if a form widget and a static label have indices '5' and '1' respectively, the form widget is brought forward and displayed above the label.





Size properties

Here is a list of properties which can be used to manage the size of different program objects.

Preferred Size



This property Sets the desired size of an element, that the program should maintain regardless of the extent to which the parent element dimensions might increase or decrease. It has two options:

-  **Width** stands for the preferred width of an element (in pixels).
-  **Height** stands for the preferred height of an element (in pixels).

You can specify either one or both of them. The preferred size has a higher priority compared to the resizing and stretching options, thus if the element has 'stretch' alignment in the form and it is assigned a preferred size in the theme, it will no longer be able to resize together with the container.

No Resize



This property specifies whether the window it is applied to is allowed to be resized at runtime.

Min Size



This property sets the minimum size to which the element can be shrunk, if its container is resized. If the element has both minimum size and preferred size, the minimum size is ignored, since the preferred size prevents the element from being resized. It has two options: Width and Height with the same meaning as the Preferred Size property does.

Max Size



This property sets the maximum size to which the element can be stretched, if its container is resized. If the element has both maximum size and preferred size, the maximum size is ignored, since the preferred size prevents the element from being resized. It has two options: Width and Height with the same meaning as the Preferred Size property does.



Text properties

Lycia Theme Designer supports a set of properties which allow to modify the settings of the text displayed to the screen. These change the look and size of the font as well as its colour.

Font



Sets font characteristics for the selected element.

To set the font changing all the options available, select New Font property option. The changes to the font are visible in an application without exiting Theme Designer.

Font property has the following options:

Family



Defines the name of the font to be used within a selected element (e.g. Tahoma, Arial, etc.).

Bold



Determines whether the text within an element has the bold attribute.

Italic



Determines whether the text within an element is to be displayed in italics.

Underline



Determines whether the text within an element is to be displayed underlined.

Font Size



Sets the font size manually or with the help of the slider.

Lycia Desktop Display Rendering

This property is applied only in the applications run with Lycia Desktop. When it is set to True (checked), the text display is adjusted according to the WPF Ideal text formatting mode, and this is the default setting.



When this property option is set to False (unchecked), the text size is applied strictly according to the text size property specified in pixels.



The parameters specified to more general objects cannot overlap the parameters set for separate objects. This means, that if an object, for example, a label, has a font specification, this font won't be changed by a Font property applied to the whole window.

To Case



The To Case property is used to specify the case for the values inputted by the INPUT, INPUT ARRAY or CONSTRUCT statements.

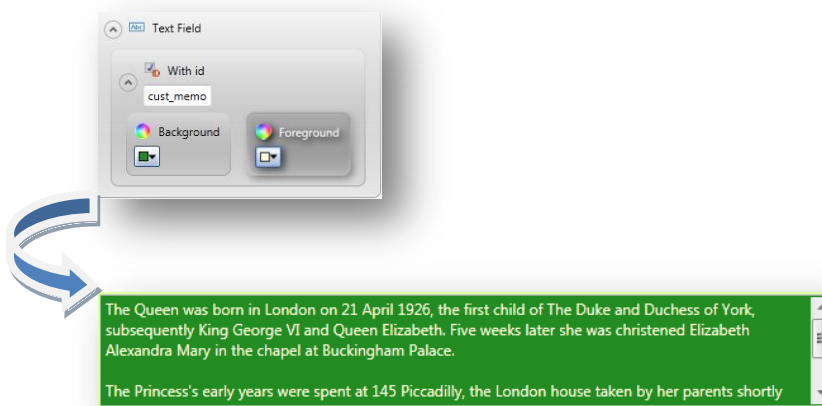
There are three available property values:

- The **None** value is the default one and means that the case of the characters correspond to that selected by the user during the input.
- The **Up** value means that all the characters inputted by the user are converted to the upper case.
- The **Down** value means that all the characters inputted by the user are converted to the lower case.

ForeClolor Property

To set the colour of the font either globally or for a specific element, the Foreground Colour property must be used. This property works as described in the [Colour Properties](#) section of this document.

The screenshot below illustrates the usage of the foreground colour which is dim yellow:



Toolbar Settings

You can use Lycia Theme Editor to modify the settings of the toolbars within your application. You can edit the toolbar settings for all the toolbars within the application or for a specified toolbar.

Toolbar visibility

By default, the window level toolbar is always visible. It can be either the default toolbar or the toolbar created using the theme file. To hide a toolbar of a window, add the toolbar from the Tree view to the Styles view, then add the Visible property to this toolbar and leave the checkbox unchecked:





To hide all the toolbars in within an application, the Tool Bar element without the With ID filter should be used.

Tool Tip



A tool tip is a help message that appears near the tool button, when the pointer is positioned on it. The tool tips can be specified during the toolbar creation. But they can be modified and made appear and disappear depending on the context.

The Tool Tip property can change the text contained in the tooltip.

The tooltip of any object can be modified using this property, moreover, it gives an opportunity to add a tooltip to an object which originally has none.

Toolbar Icons

Lycia supports changes to the scaling of dynamic Toolbar icons.

It is possible to change the default icon height and width using the Icon Height and Icon Width properties as well as the image. For more information, please, refer to the [Image Properties](#) section of this document.

Hide Labels



The Hide Label option allows the developer the choice of hiding dynamic Toolbar labels.

The default value for this property is TRUE, meaning Toolbar labels are invisible when this option is absent from the theme file, if the toolbar button has both the image and the label. If this property is added to the file, but the checkbox remains unchecked, the labels are displayed, if the check box is checked, they become invisible.

Toolbar Location



This property defines the location of toolbar: one of the two options ("top" or "right") can be set.



Web Component Properties

The Web Component element can be modified with the Theme Designer properties described below.

Component Type



The Component Type property is used to set the type of the Web component, that should be used. The front end interface scripts references the specified name.

Component Path



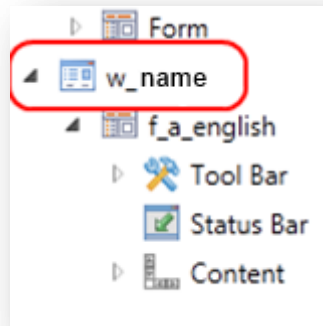
The Component Path property is used to specify the path to the web component to be loaded.



Window settings

A window element is a representation of a 4GL form element. Regardless of how it is created (with the 4GL OPEN WINDOW statement or by the framework as a default window, using built-in functions or the WINDOW data type) each window within an application have a separate window element.

A window object is a child of an application object. The Window element can be viewed from the Form Tree view:



The name of a window (its ID) is predefined when it is opened in 4GL. As 4GL is case insensitive, a window name can be lower, upper or mixed case.

The 4GL Source File:

```
MAIN
  OPEN WINDOW w_name
  AT 3,5
  WITH 19 ROWS, 72 COLUMNS
  DISPLAY "My text" AT 1,1
END MAIN
```

The exception to the window name is the default window. It is created when your program first does screen output, without opening a window prior, and is called `Screen`.

The 4GL Source File:

```
MAIN
  DISPLAY "My First Text Line" At 1,1
  OPEN FORM f_my_form FROM "f_form_file"
  DISPLAY FORM f_my_form
  DISPLAY "My Second Text Line" AT 5,5
END MAIN
```



Window Width and Height

The window width and height can be modified by means of Size properties ([Preferred size](#)).

The size parameters applied to a window specify the absolute value of its width and height to be set. This overrides the value requested by the 4gl in the OPEN statement.

The width and the height options of the Preferred Size property control the default size of the window. Besides, they control whether any scrollbars appear and, if so, the region over which scrolling can occur. It is often more preferably to set the value of the Delta Size property than the Size one.

If height and width properties are set explicitly, any of the delta properties is ignored.

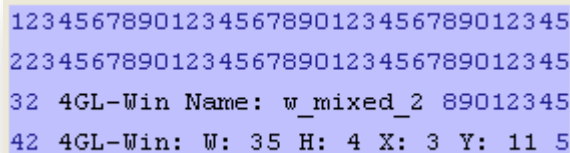
Window Style

This property specifies the style of rendering applied to windows within the application. It can be applied to all the windows or to a specific one. Since the 4GL concept of window varies from the GUI concept, there are a number of equally valid ways in which a 4GL window may be rendered, dependant on the context.

The Window Style property is intended to give the user maximum flexibility over the rendering of the application.

Flat: the windows are drawn without external frame and minimal decoration. Windows without a BORDER attribute merge seamlessly into the parent window, and those with a BORDER attribute are rendered with a single column/row width border.

For example, in the figure below, we can see a window which has been opened without a BORDER attribute (rendered in blue for clarity), merged into the application parent frame of window 'Screen'.



```
12345678901234567890123456789012345
22345678901234567890123456789012345
32 4GL-Win Name: w_mixed_2 89012345
42 4GL-Win: W: 35 H: 4 X: 3 Y: 11 5
```



If you apply the Flat option to a window without the BORDER attribute, the window appearance will not change. When applied to a window with the BORDER attribute, the window appearance changes and it looks like a window without the BORDER attribute.

The window made flat in such a way loses its title bar, status bar and toolbar and merge with the parent window, so that the user is not able to move it around.

Bordered: a bordered window does not merge into the parent window and is displayed as a separate application window with the title bar, toolbar, status bar and the border.

If you apply the Bordered option to a window having the BORDER attribute, the window appearance remains the same.

If applied to a window that does not have the BORDER attribute (a flat window), the window becomes bordered and acquires the tile bar, status bar, toolbar and the border.

Below is a screenshot showing six different bordered windows:



Title

The Title property is used to change the text displayed as the window title.

By default, a title displayed to a window with a border reflects the title defined in the 4gl when opened.

Title Bar Options

The Title bar options are those that hide or show the title bar and its elements.



The title bar is the header of a non-flat window that contains the window title and the title bar buttons: close, minimize and maximize. Flat windows do not have a title bar, so any title bar properties applied to them will have no effect.

New Title Bar Options



Automatically adds all the possible title bar options to be set by the user.

Hide Title Bar



Hides the Title Bar when added and checked: the whole title bar together with the window title and the title bar buttons becomes invisible. However, this does not turn the window into a flat window, because the border and the status bar remain intact.

Disable Title Bar Close Button



Disables the Close button when added and checked: the user is not able to exit an application by using the close button in the upper-right corner of the title bar.

Disable Title Bar Maximize/Title Bar Minimize Buttons



Similar to the option above: when checked, the specified option is inaccessible to the user.

By default, both Minimize and Maximize functionalities of the application window are available. If both these properties are set to true, the maximize and the minimize buttons become invisible. If only one of these options is set to true, the corresponding button becomes disabled.

Relative to Parent



When a new window is opened in a 4GL application being run by a GUI client, by default, the window is positioned relative to the 4GL screen window. (This is a change in behaviour from traditional 4GL, where windows were opened relative to their parent window.) To enable the traditional 4GL method of window positioning, add and check the Relative To Parent option.



Theme Designer Tutorial

Using **Theme Designer** makes the appearance customizing of your application as well as of its separate parts easy and intuitive. One can apply properties to any form element or a group of elements with little effort.

This chapter will tell you how to apply properties to the form elements of any scope beginning with all the elements and finishing with a specific one, and how the class and hierarchical filters can be used for the grouping purposes.

Besides, this chapter explains the principles of the XML theme code generation in order you can learn how to create and/or modify your *.qxtheme* theme file manually without resorting to the help of Theme Designer.

To make your reading easy-to-understand and obvious for the practical use, we have structured each section of this chapter the following way (this does not refer to the 'Theme Syntax. General Overview' section):

- Steps to reproduce in Theme Designer
- Screenshots, illustrating how an application is influenced by the changes at runtime
- Generated XML theme code extract

So, do not waste your time, let us put your knowledge to use!



Theme Syntax

Initially, we would like to explain the general rules of XML theme code creation, so, that you can easily trace the changes in it and learn how a theme can be created or modified manually within the theme file.

Basic Encoding Rules

Generally, the *.qxtheme* theme file code creation follows standard XML encoding rules, that state the following mandatory conditions:

a) case sensitivity:

```
<StyleSheet />  !=  <stylesheet />
```

b) properly nested tags:

Right:

```
<StyleSheet>
  <DoStyleAction>
    ...
  </DoStyleAction>
</StyleSheet>
```

Wrong:

```
<StyleSheet>
  <DoStyleAction>
    ...
  </StyleSheet>
</DoStyleAction>
```

c) attribute values quotation:

```
<PropertyValue type="CustomizedColor" .../>
```

d) obligatory closing tag:

```
<SetProperty></SetProperty>
or
<SetProperty />
```

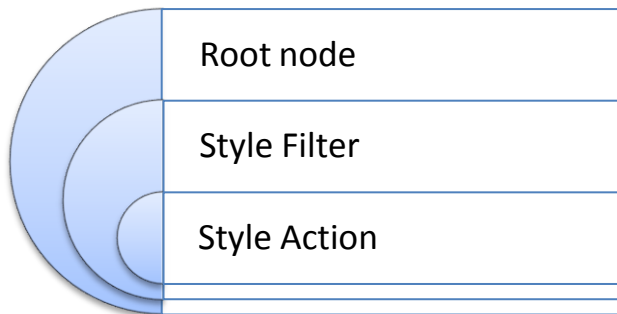


e) root node presence:

```
<StyleSheet xmlns="...">  
  ...  
</StyleSheet>
```

f) clear arrangement of a tree nodes:

Here is the schematic representation of the parent-children relations within a theme file tree elements:



it is the simplest example of such relations illustrating the basic principles of the tree elements nesting.

Theme Code Elements

All the XML theme elements can be grouped according to the purpose they are defined in a theme file for: each of them is used either to add a filter (Style Filter) or to set a property (Style Action) to a theme style.

Below is the table listing the theme code elements (tag and attribute names) constantly used in a theme file, explaining their meanings, indicating the place in a tree structure and parent element, with the usage examples.



Tree Element		Purpose and Parent Element	Example
StyleSheet		a root element	<pre><StyleSheet xmlns="http://querix.com"> [Theme Style content] </StyleSheet></pre>
		a parent element for the content of any Style Filter element: may wrap both Style Filter and/or Style Action	<pre><StyleSheet> <ElementFilter ElementName="TextField"> <StyleSheet> [Style Filter content] </StyleSheet> </ElementFilter> </StyleSheet></pre>
Style Filter	ElementFilter	adds an element object to a theme style	<pre><ElementFilter ElementName="MenuCommand"> <StyleSheet> <ElementIdFilter Identifier="Edit"> <StyleSheet> [Style Action el.] </StyleSheet> </ElementIdFilter> </StyleSheet> </ElementFilter></pre>
		may include another ElementFilter elements wrapped in a ...Filter element, or be included into it (ChildrenFilter, for example)	
		Parent element: StyleSheet	
	ElementName (attr.)	an ElementFilter attribute	
		names an object to be added to a theme file	
		its value must be quoted	
	ElementIdFilter	adds a 'With ID...' filter object to a theme style	<pre><ElementIdFilter Identifier="View"> <StyleSheet> [Style Action el.] </StyleSheet> </ElementIdFilter></pre>
		Parent element: StyleSheet	
	Identifier (attr.)	an ElementIdFilter element attribute	
		specifies an ID of an element object to be filtered by a 'With ID...' filter	
		its value must be quoted	
	WithClassFilter	adds a 'With Class...' filter object to a theme style	<pre><ElementFilter ElementName="TextField"> <StyleSheet> <WithPseudoClassFilter> <StyleSheet> [Style Action el.] </StyleSheet> </WithPseudoClassFilter></pre>
		Parent element: StyleSheet	
	WithPseudoClass Filter	adds a 'With Pseudo-Class...' filter object to a theme style	<pre><WithPseudoClassFilter> <StyleSheet> [Style Action el.] </StyleSheet> </WithPseudoClassFilter></pre>



Style Action		Parent element: StyleSheet	<code></StyleSheet></code> <code></ElementFilter></code>
	DescendantFilter	adds a ' Descendants ' filter object to a theme style	<code><DescendantFilter></code> <code><StyleSheet></code> <code><ElementFilter</code> <code>ElementName="ComboBox"></code> <code><StyleSheet /></code> <code></ElementFilter></code> <code></StyleSheet></code> <code></DescendantFilter></code>
		Parent element: StyleSheet	
	ChildrenFilter	adds a ' Children ' filter object to a theme style	
		Parent element: StyleSheet	
	DoStyleAction	wraps the <code><SetProperty /></code> tags indicating the style action adding	<code><DoStyleAction></code> <code><SetProperty></code> <code><PropertyPath></code> <code><PropertyName>Background</Pro</code> <code>pertyName></code> <code><PropertyName>FillColor</Prop</code> <code>ertyName></code> <code></PropertyPath></code> <code><PropertyValue</code> <code>type="CustomizedColor"</code> <code>RedColor="175"</code> <code>GreenColor="238"</code> <code>BlueColor="238" Alpha="255"</code> <code>/></code> <code></SetProperty></code> <code></DoStyleAction></code>
		a parent node for all the Style Action group elements	
		Parent element: StyleSheet	
	SetProperty	adds an action object to a theme style	
		Parent element: DoStyleAction	
	PropertyPath	specifies the property path	
		wraps a single or several <code>PropertyName</code> elements	
		Parent element: SetProperty	
	PropertyName	identifies the property name	
		Parent element: PropertyPath	
	PropertyValue	sets the property value	
		specifies a number of attributes depending on the <code>PropertyPath</code> stated	
		Parent element: SetProperty	
	ApplyClass	adds an Apply class action to a theme style	<code><ElementFilter</code> <code>ElementName="Button"></code> <code><StyleSheet></code> <code><DoStyleAction></code> <code><ApplyClass</code> <code>Name="class_1" /></code> <code></DoStyleAction></code> <code></StyleSheet></code> <code></ElementFilter></code>
		can be specified only within <code>DoStyleAction</code> element	
	Name (attr.)	an <code>ApplyClass</code> element attribute	
		specifies the class name to be applied	



its value must be quoted

Style Action Syntax Peculiarities

It is significant to note, that the syntax of different properties setting vary depending on the property type. The basic differences are observed in the PropertyValue declaration. It may demand enclosure of different number and type of attributes depending on the property to be set, each of them following certain encoding rules.

Here is the description of the available property types with the simplest examples illustrating the encoding rules for each of them:

- **Atomic property** is always set by a single value: string, number, Boolean or enum. It does not require any attributes, thus, its values must be specified immediately within the PropertyValue start and end tags and must not be enclosed in quotes.

Syntax:

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>AtomPropName</PropertyName>
    </PropertyPath>
    <PropertyValue>AtomPropVal</PropertyValue>
  </SetProperty>
</DoStyleAction>
```

where:

AtomPropName a string standing for the atomic property name,

AtomPropVal a string, number, boolean or enum standing for the atomic property value.

Atomic property value = String (setting the Title property as an illustration):

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath><PropertyName>Title</PropertyName>
  </PropertyPath>
  <PropertyValue>NewTitle</PropertyValue>
</SetProperty></DoStyleAction>
```

Atomic property value = number (setting the GridWidth property as an illustration):



```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>GridWidth</PropertyName>
    </PropertyPath>
    <PropertyValue>5</PropertyValue>
  </SetProperty>
</DoStyleAction>
```

Atomic property value = *BOOL* (setting the HideLabels property as an illustration):

```
TRUE
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>HideLabels
    </PropertyName>
    </PropertyPath>
    <PropertyValue />
  </SetProperty>
</DoStyleAction>
```

```
FALSE
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>HideLabels
    </PropertyName>
    </PropertyPath>
  </SetProperty>
</DoStyleAction>
```

Atomic property value = *enum* (setting the CompatibilityMode property as an illustration):

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>CompatibilityMode</PropertyName>
    </PropertyPath>
    <PropertyValue>Informix4GL</PropertyValue>
  </SetProperty>
</DoStyleAction>
```

- **Record** encloses several sub properties (single records). The latter are defined by a number of attributes, the names and value types of which differ depending on the PropertyName specified.

Syntax:

```
<DoStyleAction>
<SetProperty>
  <PropertyPath>
    <PropertyName>RecPropName</PropertyName>
  </PropertyPath>
  <PropertyValue type="RecType" AtomPropName="AtomPropVal" [...]>
```



```
<RecName RecAttrName="RecAttrVal" [...] />
[...]
</PropertyValue>
</SetProperty>
</DoStyleAction>
```

where:

<i>RecPropName</i>	a string standing for a record type property name
<i>RecType</i>	a string standing for a type attribute value specifying the record type
<i>AtomPropName</i>	a string standing for an atomic property name, if it appears within a record
<i>AtomPropVal</i>	a string, number, boolean or enum specifying an atomic property value
<i>RecName</i>	a string standing for a single record name
<i>RecAttrName</i>	an attribute name specifying a single record property
<i>RecAttrVal</i>	a single record property value
<i>[...]</i>	means, that the previous element may appear in the syntax more than once

Below you can find an illustration of how the property of the record type is set. Here, the Background property, comprising such single records as the BackgroundStyle (an atomic property), FillColor, Size and Location properties, is applied by the user.

Be careful to place single record attributes within opening and closing tags, named according to a single record name, and to quote their values according to general syntax rules regardless of their type (number, string, enum, etc.):

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>Background</PropertyName>
    </PropertyPath>
    <PropertyValue type="Background" BackgroundStyle="Tiled">
      <FillColor type="SystemColor" SystemColorName="Gray" />
      <Size Width="5" Height="3" />
      <Location XCoord="8" YCoord="5" />
    </PropertyValue>
  </SetProperty>
</DoStyleAction>
```

A property of a record type, in its turn, may be of a **polymorphic** type. It means that such a property in various situations may enclose a different number of attributes. In a theme file, the type of the property to be applied, is specified by the type attribute: its value serves as a kind of a filter for a program to know which record of attributes to specify within the PropertyValue node.



Syntax:

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>PropName</PropertyName>
      <PropertyName>SubPropName</PropertyName>
      [...]
    </PropertyPath>
    <PropertyValue type="PropType" PropAttr="AttrValue" [...] />
  </SetProperty>
</DoStyleAction>
```

where:

<i>PropName</i>	a string standing for a property name
<i>SubPropName</i>	a string standing for a sub property name. May appear in the PropertyPath node more than once. The lowermost one is the property to be specified in the PropertyValue node
<i>PropType</i>	a value of the type attribute
<i>AttrValue</i>	a value of any polymorphic property attribute
<i>[...]</i>	means, that the previous element may appear in the syntax more than once

As an example of a polymorphic property setting, a FillColor property can be applied: depending on the type attribute value, it can be set as a customized or system colour. In the first case the RedColor, GreenColor, BlueColor and Alpha attributes will be set to an object. Whereas, the only SystemColorName attribute is required for the System Colour property setting:

Customized Colour

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>Background</PropertyName>
      <PropertyName>FillColor</PropertyName>
    </PropertyPath>
    <PropertyValue type="CustomizedColor" RedColor="0"
GreenColor="255" BlueColor="255" Alpha="255" />
  </SetProperty>
</DoStyleAction>
```



System Colour

```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>Background</PropertyName>
      <PropertyName>FillColor</PropertyName>
    </PropertyPath>
    <PropertyValue type="SystemColor" SystemColorName="Blue" />
  </SetProperty>
</DoStyleAction>
```

For more detailed information concerning the theme syntax, please, refer to the "Querix 4GL UI Types" document included into the package.



Applying Properties to Elements

Theme designer provides you with a wide range of ways in which properties can be applied to form elements. Further as the text goes, we will give the detailed explanation of the most common and consistent of them.

For the demonstration purpose we will be using the following program:

The 4GI Source File:

```
MAIN
DEFINE a,b,c,d STRING
    OPEN WINDOW W1 AT 4,4 WITH FORM "themes_form" ATTRIBUTE (BORDER)

    INPUT a,b,c,d FROM F001, F002, f003, f004
        ON KEY (F12)
        EXIT INPUT
    END INPUT
CALL fgl_getkey()
CLOSE WINDOW W1
END MAIN
```

The XML Source Form File:

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://namespaces.querix.com/2011/fglForms">
<rootcontainer type="coordpanel" identifier="" visible="true"
enable="true" zOrder="0" fieldTable="">
    <classnames/>
    <location xCoord="0" yCoord="0"/>
    <minsize width="244" height="143"/>
    <locale/>
    <displaymodes/>
    <items>
        <textfield identifier="f001" visible="true" enable="true"
zOrder="0" fieldTable="formonly">
            <classnames/>
            <location xCoord="8" yCoord="0"/>
            <preferredsize width="96" height="22"/>
            <locale/>
            <displaymodes/>
            <datatype typeName="Char"/>
            <includes/>
        </textfield>
        <textfield identifier="f002" visible="true" enable="true"
zOrder="0" fieldTable="formonly">
```

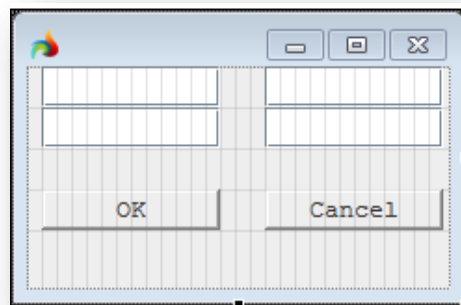


```
<classnames/>
<location xCoord="128" yCoord="0"/>
<preferredsize width="96" height="22"/>
<locale/>
<displaymodes/>
<datatype typeName="Char"/>
<includes/>
</textfield>
<textfield identifier="f003" visible="true" enable="true"
zOrder="0" fieldTable="formonly">
  <classnames/>
  <location xCoord="8" yCoord="22"/>
  <preferredsize width="96" height="22"/>
  <locale/>
  <displaymodes/>
  <datatype typeName="Char"/>
  <includes/>
</textfield>
<textfield identifier="f004" visible="true" enable="true"
zOrder="0" fieldTable="formonly">
  <classnames/>
  <location xCoord="128" yCoord="22"/>
  <preferredsize width="96" height="22"/>
  <locale/>
  <displaymodes/>
  <datatype typeName="Char"/>
  <includes/>
</textfield>
<button identifier="f005" visible="true" enable="true"
zOrder="0" fieldTable="" text="OK">
  <classnames/>
  <location xCoord="8" yCoord="66"/>
  <preferredsize width="96" height="22"/>
  <locale/>
  <textalignment horizontalTextAlignment="Center"/>
  <displaymodes/>
  <oninvoke type="keyeventhandler">
    <keyname keyValue="ACCEPT"/>
  </oninvoke>
</button>
<button identifier="f006" visible="true" enable="true"
zOrder="0" fieldTable="" text="Cancel">
  <classnames/>
  <location xCoord="128" yCoord="66"/>
  <preferredsize width="96" height="22"/>
  <locale/>
  <textalignment horizontalTextAlignment="Center"/>
  <displaymodes/>
  <oninvoke type="keyeventhandler">
    <keyname keyValue="F12"/>
  </oninvoke>
</button>
```



```
        </oninvoke>
      </button>
    </items>
    <preferredsize width="244" height="143"/>
  </rootcontainer>
  <screenrecords>
    <screenrecord identifier="formonly">
      <fields>
        <field>lab</field>
        <field>f001</field>
        <field>f002</field>
        <field>f003</field>
        <field>f004</field>
        <field>f005</field>
        <field>f006</field>
      </fields>
    </screenrecord>
  </screenrecords>
  <phantomfields/>
  <imageid uri=""/>
</form>
```

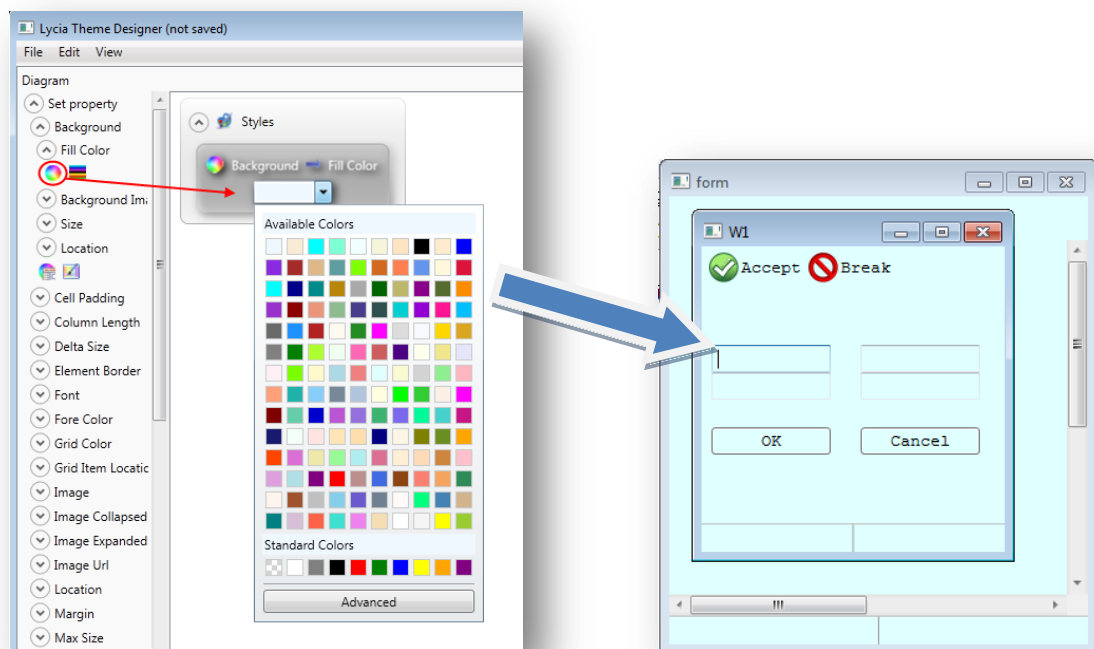
The Form Design:





Applying Global Properties to All The Form Elements

1. Unfold the **Set property** group from the **Diagram** view.
2. Select **Background** -> **Fill Color** -> **New custom color** and add it to the **Styles** view.
3. Select the required colour from the drop-down list. It will be applied to all the object in the application, because this property is not linked to any object:




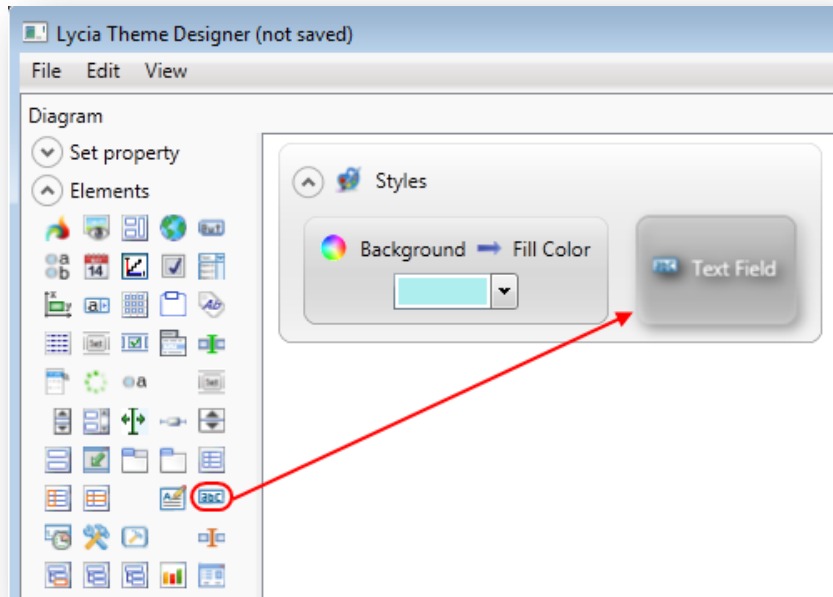
Generated XMLtheme code:


```
<DoStyleAction>
  <SetProperty>
    <PropertyPath>
      <PropertyName>Background</PropertyName>
      <PropertyName>FillColor</PropertyName>
    </PropertyPath>
    <PropertyValue type="CustomizedColor" RedColor="175"
GreenColor="238" BlueColor="238" Alpha="255" />
  </SetProperty>
</DoStyleAction>
```

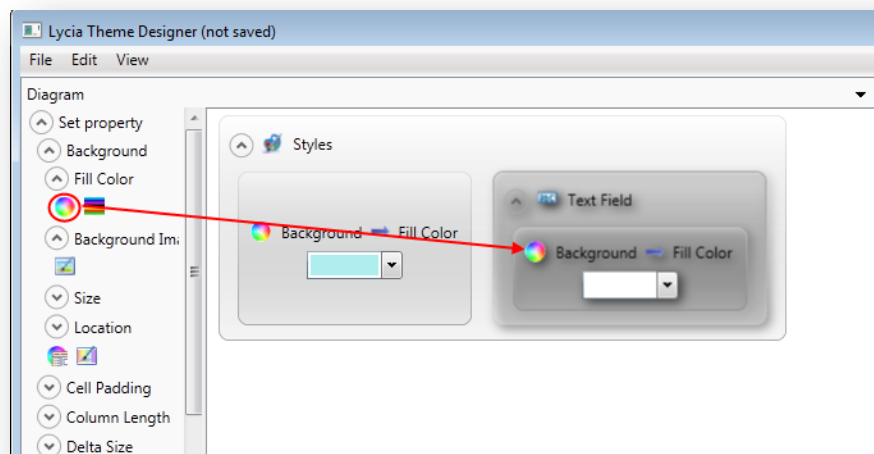


Applying Global Properties to Form Elements of the Same Type

1. Unfold the **Elements** group from the **Diagram** view.
2. Select the Text Field icon () from the list and drag this element to the **Styles** view:



3. Unfold the **Set property** group from the **Diagram** view.
4. Select **Background** -> **Fill Color** ->  'New custom color' and drag it to the **Styles** view placing the property object inside the **Text Field** element:





5. Select the required colour from the drop-down list. It will be applied to all the text fields in the application, because this property was linked to all the elements of this type:



Generated XML-code:

```
<ElementFilter ElementName="TextField">
  <StyleSheet>
    <DoStyleAction>
      <SetProperty>
        <PropertyPath>
          <PropertyName>Background</PropertyName>
          <PropertyName>FillColor</PropertyName>
        </PropertyPath>
        <PropertyValue type="CustomizedColor" RedColor="255"
GreenColor="255" BlueColor="0" Alpha="255" />
      </SetProperty>
    </DoStyleAction>
  </StyleSheet>
</ElementFilter>
```

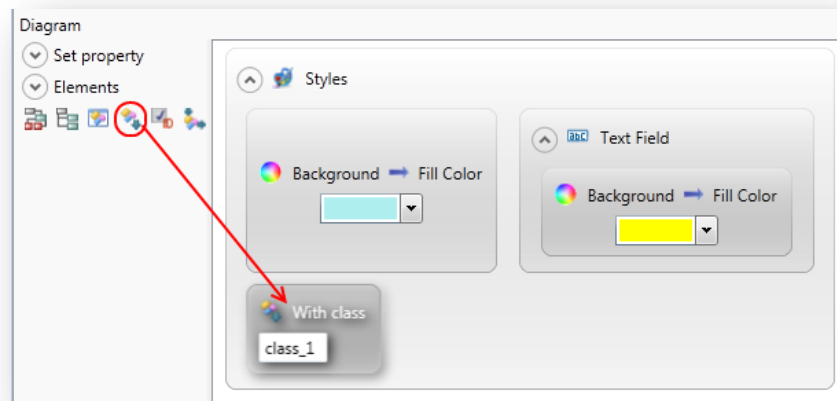



Applying Global Properties to Form Elements Using Class Filters

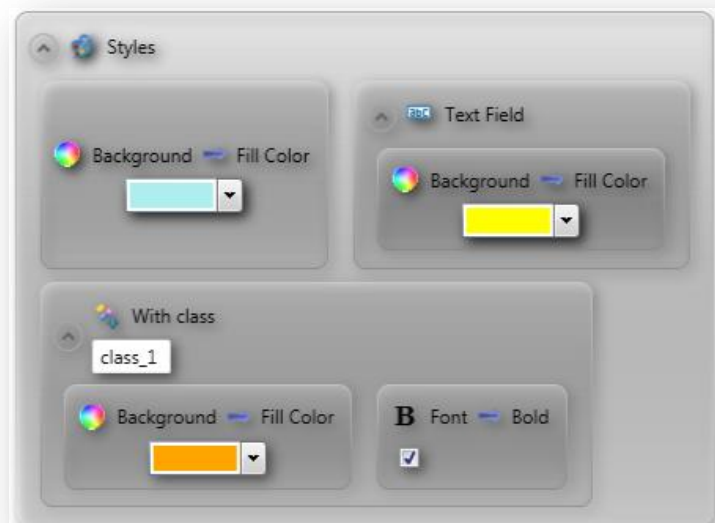
It is possible to apply a predefined set of properties to a number of objects. For this purpose classes are used. For our test program classes can be applied as follows:

'With class ...'

1. Add the  **'With class ...'** object to the **Styles** view outside of any object. In the empty field input the class name:



2. Drag and drop the required properties from the **Property** group inside the **'With class ...'** object. Let select the following:
 - Set property -> Background -> Fill Color -> **New Custom Color**
 - Set property -> Font -> **Bold**

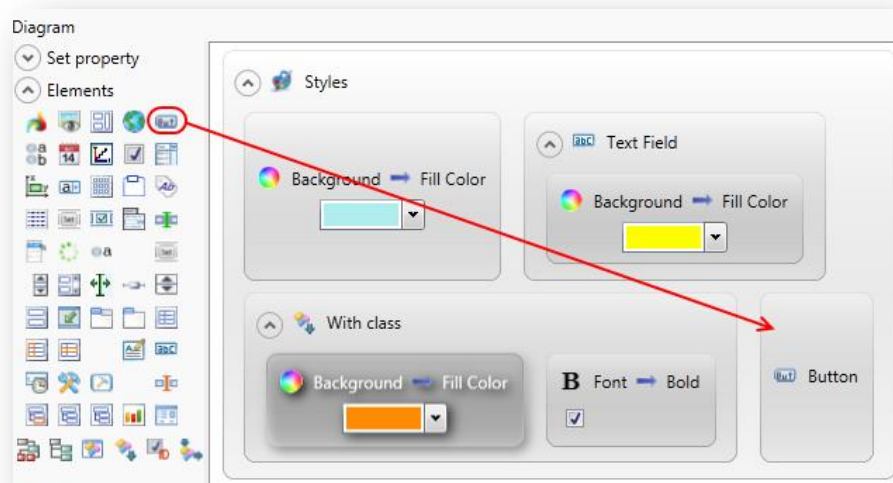



Generated XML-code:

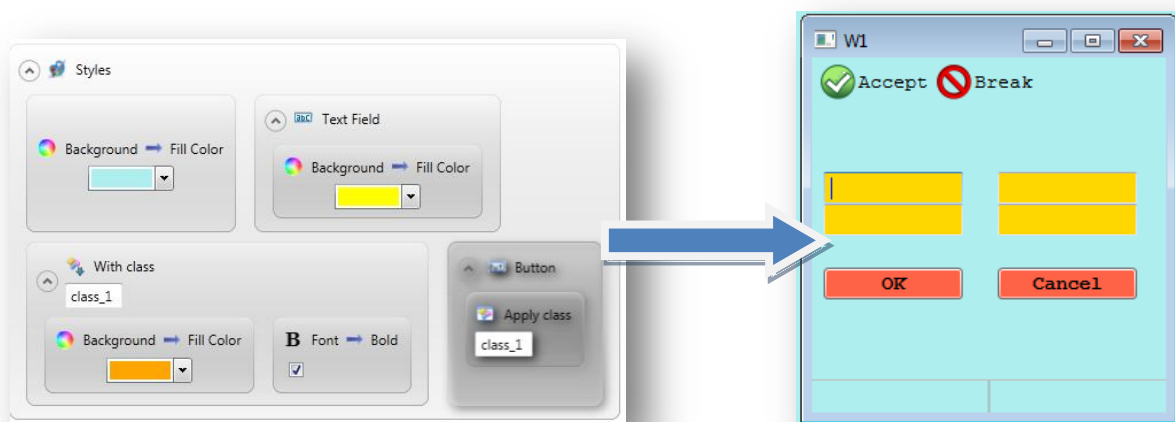
```
<WithClassFilter ClassName="class_1">
  <StyleSheet>
    <DoStyleAction>
      <SetProperty>
        <PropertyPath>
          <PropertyName>Background</PropertyName>
          <PropertyName>FillColor</PropertyName>
        </PropertyPath>
        <PropertyValue type="CustomizedColor" RedColor="255"
GreenColor="165" BlueColor="0" Alpha="255" />
      </SetProperty>
    </DoStyleAction>
    <DoStyleAction>
      <SetProperty>
        <PropertyPath>
          <PropertyName>Font</PropertyName>
          <PropertyName>Bold</PropertyName>
        </PropertyPath>
        <PropertyValue />
      </SetProperty>
    </DoStyleAction>
  </StyleSheet>
</WithClassFilter>
```

'Apply class ...'

1. Add the **Button** element to the **Styles** view from the **Elements** group:



2. Drag and drop the  **'Apply class ...'** filter to the **Styles** view within the **Button** object.
3. Specify the name of the class in the empty field. In our case it will be the 'class_1' (the same class name as we have specified for the 'With class ...' filter above):



Generated XML-code:


```
<ElementFilter ElementName="Button">
  <StyleSheet>
    <DoStyleAction>
      <ApplyClass Name="class_1" />
    </DoStyleAction>
  </StyleSheet>
</ElementFilter>
```

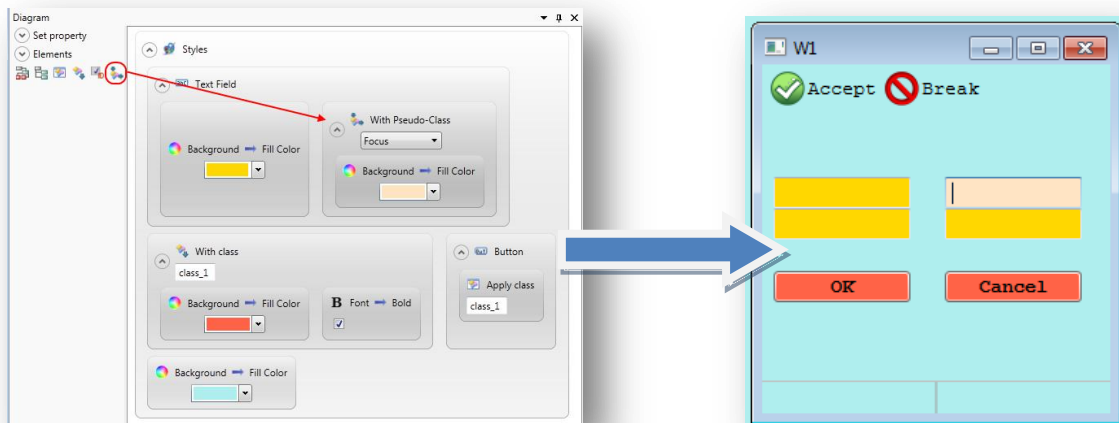


'With Pseudo-Class ...'

A pseudo-class filter allows you to apply some specific property to the elements of your program only when they are in a specific state. Among states from the list of states available for a pseudo-class filter are focus/no focus, active/inactive, input, prompt, display and some others. The detailed description of them you may find in the "Lycia II Theme Designer" guide.

A pseudo-class filter is applied in the following way:

1. Drag and drop the  **'With Pseudo-class...'** object to the **Styles** view inside the Text Field element.
2. Select the **Focus** state from the drop-down list.
3. Place the background colour property within the pseudo class property.
4. The text field will get the specified colour, when the cursor is located in it otherwise the background colour set for all the text field will be applied:



Generated XML-code:

```
<ElementFilter ElementName="TextField">
  <StyleSheet>
    <WithPseudoClassFilter>
      <StyleSheet>
        <DoStyleAction>
          <SetProperty>
            <PropertyPath>
```



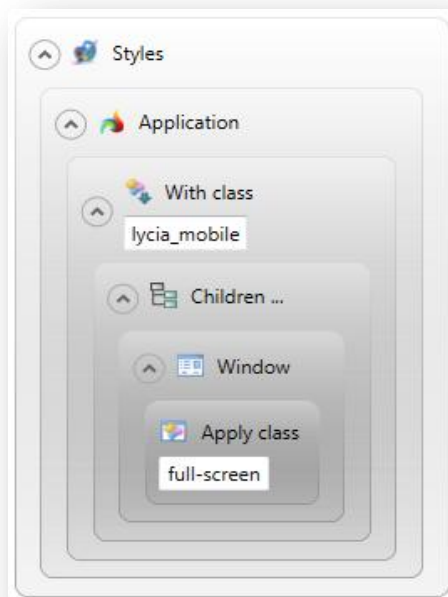
```
<PropertyName>Background</PropertyName>
<PropertyName>FillColor</PropertyName>
</PropertyPath>
<PropertyValue type="CustomizedColor" RedColor="255"
GreenColor="228" BlueColor="181" Alpha="255" />
</SetProperty>
</DoStyleAction>
</StyleSheet>
</WithPseudoClassFilter>
</StyleSheet>
</ElementFilter>
```

Filters Usage for Full-Screen Mode Applying

As the adaptation of the user interface for the device utilized by the user is one of the highest priority tasks of a developer, we provide you with an example of how the full-screen mode can be set in Theme Designer.

Below is an illustration of how a *full-screen* class can be applied to all the windows of an application filtered with the *lycia_mobile* class filter. Note, that such a modification, based on applying the *lycia_mobile* class filter to the theme, effects an application, only if it is run on a mobile device:

Theme Styles diagram:




Generated XML-code:




```
<ElementFilter ElementName="Application">
  <StyleSheet>
    <WithClassFilter ClassName="lycia_mobile">
      <StyleSheet>
        <ChildFilter>
          <StyleSheet>
            <ElementFilter ElementName="Window">
              <StyleSheet>
                <DoStyleAction>
                  <ApplyClass Name="full-screen" />
                </DoStyleAction>
              </StyleSheet>
            </ElementFilter>
          </StyleSheet>
        </ChildFilter>
      </StyleSheet>
    </WithClassFilter>
  </StyleSheet>
</ElementFilter>
```

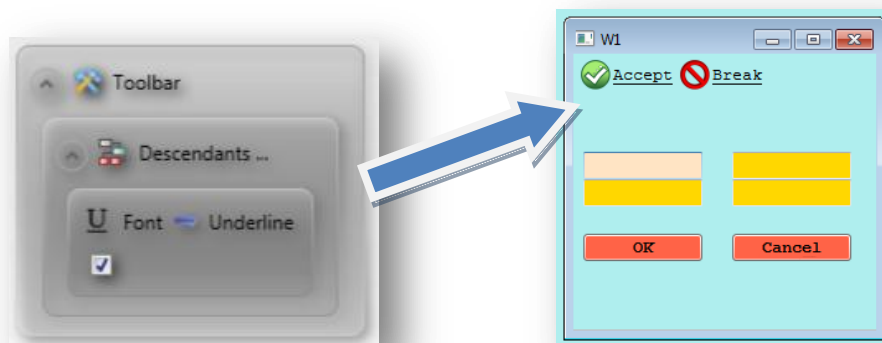
Applying Global Properties to Form Elements Using Hierarchical Filters

'Descendants ...'

The  **Descendants ...** filter refers to all the descendants of the specified element. For example, the descendants of a form are the Tool Bar and Status Bar as well as all the form widgets. The descendants can be viewed from the **Form Tree** view.

To apply a property to all the descendants of an element do the following:

1. Place the **Toolbar** element onto the **Styles** view.
2. Place the  **Descendants ...** filter within this element.
3. Place any property within the filter, let it be **Set property** -> **Font** -> **Underline** This property will be applied to all the elements within the Toolbar element:






Generated XML-code:

```
<ElementFilter ElementName="Toolbar">
  <StyleSheet>
    <DescendantFilter>
      <StyleSheet>
        <DoStyleAction>
          <SetProperty>
            <PropertyPath>
              <PropertyName>Font</PropertyName>
              <PropertyName>Underline</PropertyName>
            </PropertyPath>
            <PropertyValue />
          </SetProperty>
        </DoStyleAction>
      </StyleSheet>
    </DescendantFilter>
  </StyleSheet>
</ElementFilter>
</StyleSheet>
```

'Children ...'

With the help of the  'Children ...' filter, you can apply the properties to all the children of the element specified. Children refer to immediate descendants of the first order. You can use it the same way as the 'Descendants ...' filter.



Applying Properties to a Specific Form Element

You can apply a property to a specific form element following the next steps:

1. Select the required element from the Form Tree view. Let it be **W1** -> **Tool Bar**.
2. Right-click it and select the Add Selector from the context menu. The required element will be added to the Styles view:



3. Now you can apply any property to this particular form element.



Generated XML-code:




```
<ElementFilter ElementName="Window">
  <StyleSheet>
    <ElementIdFilter Identifier="W1">
      <StyleSheet>
        <ChildFilter>
          <StyleSheet>
            <ElementFilter ElementName="Toolbar">
              <StyleSheet>
                <ElementIdFilter Identifier="Tool Bar">
                  <StyleSheet />
                </ElementIdFilter>
              </StyleSheet>
            </ElementFilter>
          </StyleSheet>
        </ChildFilter>
      </StyleSheet>
    </ElementIdFilter>
  </StyleSheet>
</ElementFilter>
```

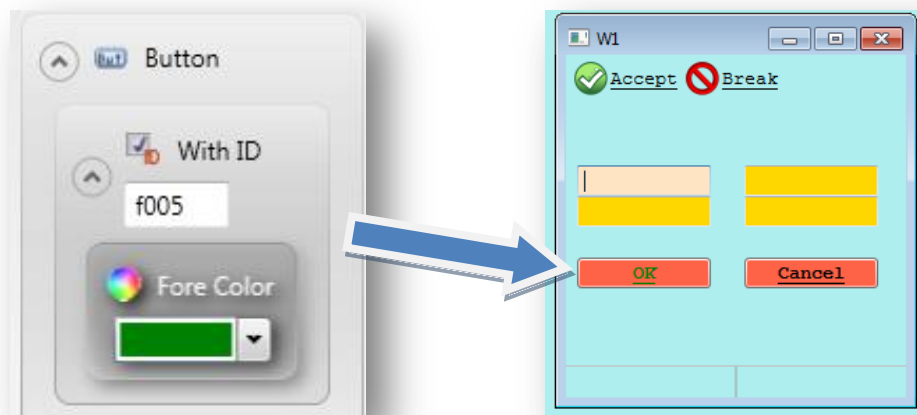
Applying Properties to a Form Element Using the 'With ID ...' Filter

A property to a specific form element can also be applied using the 'With ID ...' filter. The instructions below show how it can be done.

1. Select  **'Button'** object from the **Elements** group, place it onto the **Styles** view.
2. Select the  **'With ID ...'** filter from the **Diagram** section, place it within the **Button** object on the **Styles** view.
3. Type the ID of the button (f005) in the empty field of the **'With ID ...'** object.

Properties placed into the With ID group will be applied only to this specific button. You can drag another 'With ID ...' object into this **Button** group to specify the properties for the second button exclusively. The ID corresponds to the name of the object in the **Form Tree** view and can also be found in the **Properties** view for the object.

4. Select the **Fore Color** ->  **New Custom Color** and drag it into the **'With ID ...'** object. Choose the colour. As a result the specified Fore colour will only be applied to the button with ID 'f005':



Generated XML-code:

```
<ElementFilter ElementName="Button">
  <StyleSheet>
    <ElementIdFilter Identifier="f005">
      <StyleSheet>
        <DoStyleAction>
          <SetProperty>
            <PropertyPath>
              <PropertyName>ForeColor</PropertyName>
            </PropertyPath>
            <PropertyValue type="CustomizedColor" RedColor="0"
GreenColor="128" BlueColor="0" Alpha="255" />
          </SetProperty>
        </DoStyleAction>
      </StyleSheet>
    </ElementIdFilter>
  </StyleSheet>
</ElementFilter>
```