TMS Parameter Controls
DEVELOPERS GUIDE

July 2019
Copyright © 2014-2019 by tmssoftware.com bvba
Web: https://www.tmssoftware.com
Email : info@tmssoftware.com

Table of contents

Availability

TMS Parameter controls are available for Delphi and C++Builder. TMS Parameter controls can be used on Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10. The TMS Parameter controls use Windows XP or higher visual styles when used on Windows XP or higher with Windows XP or higher visual styles enabled.

TMS Parameter controls are available as VCL component set for Win32/Win64 application development.

Introduction

TMS Parameter controls consist currently of 4 components: TParamLabel, TParamListBox, TParamCheckList and TParamTreeview. The parameter controls enable building various types of informative user entry screens such as the Outlook Inbox rules wizard, end-user SQL building tools and much more. The parameter controls layout is managed through HTML formatting. The HTML tag that handles parameter input is the <A> anchor tag. The <A> anchor tag and its options to control the parameter input are explained in detail in this document.

Overview of supported HTML tags

- **B : Bold tag**
  <B> : start bold text
  </B> : end bold text

  Example : This is a <B>test</B>

- **U : Underline tag**
  <U> : start underlined text
  </U> : end underlined text

  Example : This is a <U>test</U>

- **I : Italic tag**
  <I> : start italic text
  </I> : end italic text

  Example : This is a <I>test</I>

- **S : Strikeout tag**
  <S> : start strike-through text
  </S> : end strike-through text

  Example : This is a <S>test</S>

- **A : anchor tag**
  <A href="value"> : text after tag can be a parameter. The 'value' after the href identifier is the parameter identifier which must be a unique value for the component. The anchor value can be found in the OnAnchorClick event

</A> : end of anchor

More information and details on the anchor tag for parameter control is available in the next paragraphs.

- **FONT : font specifier tag**
  <FONT face='facevalue' size='sizevalue' color='colorvalue' bgcolor='colorvalue'> : specifies font of text after tag.
  with
  - face : name of the font
  - size : HTML style size if smaller than 5, otherwise pointsize of the font
  - color : font color with either hexidecimal color specification or Borland style color name, ie clRed,clYellow,clWhite ... etc
  - bgcolor : background color with either hexidecimal color specification or Borland style color name
  </FONT> : ends font setting

  Examples :
  This is a <FONT face="Arial" size="12" color="clRed">test</FONT>
  This is a <FONT face="Arial" size="12" color="#FF0000">test</FONT>

- **P : paragraph**
  <P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If the bgcolorto attribute is used, a paragraph will have a gradient background from color 'bgcolor' to color 'bgcolorto'
  </P> : end of paragraph

  Example : <P align="right">This is a test</P>
  Example : <P align="center">This is a test</P>
  Example : <P align="left" bgcolor="#ff0000">This has a red background</P>
  Example : <P align="right" bgcolor="clYellow">This has a yellow background</P>
  Example : <P align="right" bgcolor="clYellow" bgcolorto="clred">A gradient background</P>

- **HR : horizontal line**
  <HR> : inserts linebreak with horizontal line

- **BR : linebreak**

 : inserts a linebreak

- **BODY : body color / background specifier**
  <BODY bgcolor="colorvalue" background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

  Example :
  <BODY bgcolor="clYellow"> : sets background color to yellow
  <BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp

- **IND : indent tag**
  This is not part of the standard HTML tags but can be used to easily create multicolumn text
  <IND x="indent"> : indents with "indent" pixels

  Example :
  This will be <IND x="75">indented 75 pixels.

- **IMG : image tag**
  <IMG src="specifier:name" [align="specifier"] [width="width"] [height="height"]
  [alt="specifier:name"] > : inserts an image at the location

  specifier can be :
  idx : name is the index of the image in the associated imagelist
  ssys : name is the index of the small image in the system imagelist or a filename for which the corresponding system imagelist is searched
  lsys : same as ssys, but for large system imagelist image
  file : name is the full filename specifier
  res : name of a resource bitmap (not visible at design time)

  Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are : align="top" and align="middle"

  The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

  Examples :
  This is an image <IMG src="idx:1" align="top">
  This is an image <IMG src="ssys:1"> and another one <IMG src="ssys:worfile.doc">
  This is an image <IMG src="file://c:\my documents\test.bmp">
  This is an image <IMG src="res://BITMAP1">
  This is an image <IMG src="res://BITMAP1" width=”32” height=”32”>

- **SUB : subscript tag**

  <SUB> : start subscript text
  </SUB> : end subscript text

  Example : This is ⁹/₁₆ looks like $^{9}/_{16}$

- **SUP : superscript tag**

  <SUP> : start superscript text
  </SUP> : end superscript text

- **UL : list tag**
  <UL> : start unordered list tag
  </UL> : end unordered list

  Example :
  <UL>
  <LI>List item 1
  <LI>List item 2
  <UL>
  <LI> Sub list item A
  <LI> Sub list item B
  </UL>
  <LI>List item 3
  </UL>

- **LI : list item**
  <LI> : new list item

- **SHAD : text with shadow**
  <SHAD> : start text with shadow
  </SHAD> : end text with shadow

- **Z : hidden text**

  <Z> : start hidden text
  </Z> : end hidden text

- **HI : hilight**

  <HI> : start text hilighting
  </HI> : stop text hilighting

- **E : Error marking**

  <E> : start error marker
  </E> : stop error marker

- **Special characters**

  Following standard HTML special characters are supported :

  < : less than : <
  > : greater than : >
  & : &
  " : "
    : non breaking space
  ™ : trademark symbol
  € : euro symbol
  § : section symbol
  © : copyright symbol
  ¶ : paragraph symbol

HTML related properties

The parameter controls share properties that further control the appearance and behaviour of the HTML formatted content. Following properties are available:

EmptyParam: string

This property sets the text to display for parameters that are empty, ie. Zero length strings. The default value is '?'

Hover: boolean;

When true, parameters are highlighted with the colors HoverColor and HoverFontColor when the mouse is over a parameter.

HoverColor: TColor;

Sets the background color for a parameter when the mouse is over it.

HoverFontColor: TColor;

Sets the text color for a parameter when the mouse is over it.

LineSpacing: Integer;

This property sets the default spacing (in pixels) between two lines of HTML formatted text.

ParamHint: Boolean;

When true, a hint is displayed (if ShowHint = true) when the mouse is over a parameter. The hint can be set dynamically through the event OnParamHint or can be set as attribute of the parameter. See parameter attributes.

ParamColor: TColor;

Sets the text color for parameters.

ShadowColor: TColor;

Sets the color of shadows for HTML formatted text that has a shadow through the <SHAD> tag

ShadowOffset: Integer;

Sets the offset of the displayed shadow for text with the <SHAD> tag.

HTML related events

The parameters that can be added inside controls cause following events when the mouse is over or clicked on the parameter. The events are:

OnAnchorClick : triggered when a hyperlink is clicked on a parameter

OnAnchorEnter : triggered when the mouse enters a parameter

OnAnchorExit : triggered when the mouse leaves a parameter

Example:

A hyperlink is added with

```
'This is a <a href="myparam">paramvalue<a>';
```

When the mouse clicked on the parameter, the OnAnchorClick is called with a reference to the parameter identifier 'myparam' and the parameter value 'paramvalue'

The <A> anchor parameter tag

The structure of the <A> anchor parameter tag is:

<A href="paramid" class="paramtype" props="paramprops" hint="paramhint">paramvalue

with:

paramid: a unique parameter identifier. Through this identifier the value of the parameter can be retrieved or set.

paramtype: defines the type of inplace editor that is used to edit the parameter value. The paramtype can be:

LIST : a listbox is used, the PROPS attribute specifies the listbox elements
MENU : a popup menu is used, the PROPS attribute specifies the popup menu items
EDIT : a simple inplace editor is used
SPIN : an inplace spin editor is used
DATE : an inplace date picker is used.
TIME : an inplace time selector is used
MASK : an inplace mask editor is used, the mask value is set in the PROPS attribute
QUERY : an input query dialog can be used
DIR : a directory picker is used
CUSTOM : a custom inplace editor is used
TOGGLE : toggles between two values in the PROPS attribute

If paramtype is empty or not defined, the <A> tag behaves like a normal hyperlink.

paramprops: sets the properties (when used) for the inplace editor, this is in particular a list of options for popup list editor and popup menu editor. The values in the PROPS attribute are separated by the | character. The paramprops attribute is only used for LIST, MENU, TOGGLE and MASK edit types.

paramhint: sets the hint to display when the mouse is over the parameter. This attribute is optional.

Examples:

A normal edit parameter:

<A href="edit" class="EDIT">10</A>

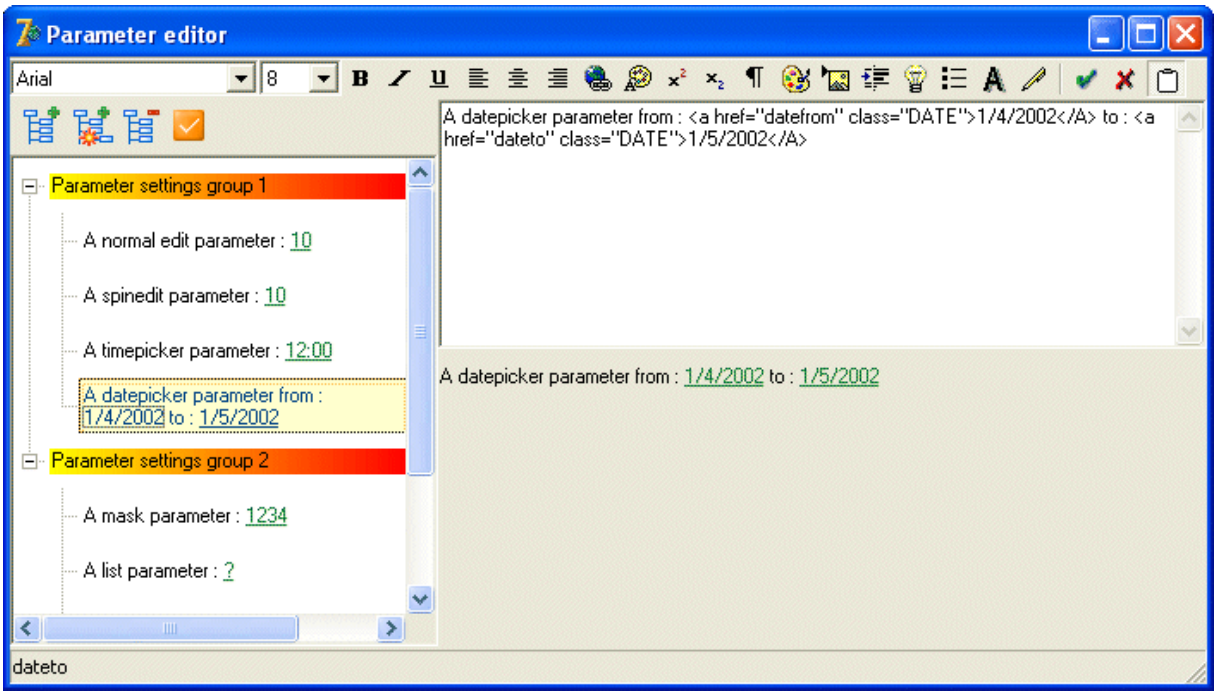
A mask parameter:

<A href="mask" class="MASK" props="9999">1234</A>

A menu parameter:

<A href="menu" class="MENU" props="Menu item 1|Menu item 2|Menu item 3|Menu item 4">?</A>

Note that the HTML formatted control text and parameters can be set at design time with special property editors that facilitate the construction of the required HTML.



Runtime control of parameter editors

With the events OnParamList and OnParamPopup, the list of values to select from either through a popup listbox or a popup menu can be set dynamically at runtime.

OnParamList:
This event returns the href and value of the parameter clicked and allows setting of the values that will appear in the dropdown listbox. If the doPopup parameter is returned true, the dropdown list will appear. The values of the list can be set through the stringlist Values parameter.

Example:

```
doPopup := (href = 'tool');
values.Add('Delphi');
values.Add('C++Builder');
values.Add('JBuilder');
```

OnParamPopup :
This event is similar to the OnParamList except that it controls the popup menu for parameter selection. This event returns the href and value of the parameter clicked and allows setting of the values that will appear in the popup menu. If the doPopup parameter is returned true, the popup menu will appear. The values of the popup menu can be set through the stringlist Values parameter.

Example:

```
doPopup := (href='tool') or (href='os');
if (href = 'tool') then
begin
  values.Add('Delphi');
  values.Add('C++Builder');
  values.Add('JBuilder');
end;

if (href = 'os') then
begin
  values.Add('Windows');
  values.Add('Linux');
  values.Add('Solaris');
end;
```

With the event OnParamClick, either parameters can be edited or for regular hyperlinks, a browser can be opened or another action executed. The event returns the href and value of the parameter clicked. The value can be changed in code.

Example 1: (this toggles the value between Delphi & C++Builder)

If the control contains the text:

Select your tool : <A href="tool">Delphi</A> and OS : <A href="os">Windows</A>

```
if (href = 'tool') and (value = 'Delphi') then
  value := 'C++Builder'
else
  value := 'Delphi';
```

Example 2: (this asks the user for the value)

```
if (href = 'os') then
  InputQuery('Operating system','Name',value);
```

Common parameter control properties and methods

Properties

Control.Parameter[href:string]: string;

Property through which parameter values can be set and retrieved. Note that href is the unique parameter identifier in the control.

Example:

Select your tool : <A href="tool">Delphi</A> and OS : <A href="os">Windows</A>

The parameter values can here be obtained through :

```
var
os,tool: string;

tool := control.Parameter['tool'];
os := control.Parameter['os'];
```

The parameter values can be set through :

```
control.Parameter['tool'] := 'Kylix';
control.Parameter['os'] := 'Linux';
```

Control.ParamRefCount: Integer;

Returns the number of parameters in the control

Control.ParamRefs[idx: Integer]: string;

Returns the parameter identifier for the parameter idx in the control

Control.ParamIndex[href: string]: Integer;

Returns the index of the parameter with href identifier in the control

Control.ParamRect[href: string]: TRect;

Returns the rectangle where the parameter with href identifier is located in the control

Control.FocusParam: Integer;

Gets or sets the focus on the parameter at FocusParam index in the control

Control.ParamItemRefCount[ItemIndex: Integer]: Integer;

Only available in TParamListBox and TParamCheckList. Returns the number of parameters at a given item ItemIndex.

Control.ParamItemRefs[ItemIndex,ParamIndex: Integer]: string;

Only available in TParamListBox and TParamCheckList. Returns the parameter identifier for the parameter ParamIndex in the item ItemIndex in the control

Control.ParamItemIndex[ItemIndex: Integer; href:string]: Integer;

Only available in TParamListBox and TParamCheckList. Returns the index of the parameter with href identifier in the item ItemIndex in the control

Control.ItemParameter[ItemIndex: Integer; href: string]: string;

Only available in TParamListBox and TParamCheckList. Property through which parameter values can be set and retrieved for item ItemIndex in the list

Control.ParamRefNode[href: string]: TTreeNode;

Only available in TParamTreeview. Returns the node where the parameter with identifier href is located.

Control.ParamNodeRefCount[NodeIndex: Integer]: Integer;

Only available in TParamTreeview. Returns the number of parameters at Node with index NodeIndeex.

Control.ParamNodeRefs[NodeIndex, ParamIndex: Integer]: string;

Only available in TParamTreeview. Returns the parameter identifier for the parameter ParamIndex in the Node NodeIndex in the control

Control.ParamNodeIndex[Node: TTreeNode; href: string]: Integer;

Only available in TParamTreeview. Returns the index of the parameter with href identifier in the node Node in the control

Methods

procedure EditParam(href: string);

Starts inplace editing of parameter with identifier href

function GetParamInfo(href: string; var AValue,AClass,AProp,AHint: string): Boolean; (for TParamLabel)

If the parameter identifier href is found, it returns the parameter value and attributes

function GetParamInfo(ItemIndex: integer; href: string; var AValue,AClass,AProp,AHint: string): Boolean; (for TParamLabel)

If the parameter identifier href is found in item ItemIndex, it returns the parameter value and attributes

Using custom inplace editors

Using custom inplace editors is possible with the Parameter controls but require some preparing of the inplace editor to be used correct. The custom inplace editor is invoked with the style 'CUSTOM', ie:

<A href="mycustomcontrol" class="CUSTOM" props="myprops">?</A>

Parameter control events

When the custom inplace editor is about to be edited, the event OnParamCustomEdit is triggered with parameters:

For TParamLabel:

```
TParamCustomEditEvent = procedure(Sender: TObject; href, value, props:
string; EditRect: TRect) of object;
```

For TParamListBox and TParamCheckList:

```
TParamCustomEditEvent = procedure(Sender: TObject; idx: Integer; href,
value, props: string; EditRect: TRect) of object;
```

For TParamTreeView:

```
TParamCustomEditEvent = procedure(Sender: TObject; Node: TTreeNode; href,
value, props: string; EditRect: TRect) of object;
```

The href parameter is the unique ID of the parameter in the control. The value parameter contains the current value of the parameter, the props parameter the value of the PROPS attribute. Last parameter is the rectangle where the inplace editor should be positioned. For the TParamListBox and TParamCheckList, an additional idx parameter refers to the item in the list where the parameter is located, for the TParamTreeView, the Node parameter refers to the node where the parameter is located.

From this event, following code can be used to show the custom inplace edit control:

```
mycustomcontrol.Parent := Sender;
mycustomcontrol.left := EditRect.Left;
mycustomcontrol.Top := EditRect.Top;
mycustomcontrol.Text := value;
mycustomcontrol.Param := href;
mycustomcontrol.OnUpdate := (Sender as TParamlabel).ControlUpdate;
mycustomcontrol.Visible := true;
mycustomcontrol.SetFocus;
```

The ControlUpdate procedure is defined in the parameter component and will set the new parameter value when the editing is done. ControlUpdate is defined as:

```
procedure ControlUpdate(Sender: TObject; Param,Text:string);
```

with Param the unique parameter identifier and Text the parameter value to set.

Preparing the inplace edit control

Create a descendent component of the inplace edit control with following properties and events:

```
TParamUpdateEvent = procedure(Sender: TObject; Param, Text: string) of
object;

TMyCustomControl = class(TMyControl)
private
  FParam: string;
  FOnUpdate: TParamUpdateEvent;
  procedure WMActivate(var Message: TWMActivate); message WM_ACTIVATE;
protected
  procedure DoExit; override;
  procedure CreateParams(var Params: TCreateParams); override;
public
  property Param: string read FParam write FParam;
  property OnUpdate: TParamUpdateEvent read FOnUpdate write FOnUpdate;
end;
```

The implementation for this descendent control is then:

```
{ TMyCustomControl }

procedure TMyCustomControl.CreateParams(var Params: TCreateParams);
begin
  inherited;
  with Params do
    Style := Style AND NOT WS_CHILD OR WS_POPUP;
end;

procedure TMyCustomControl.DoExit;
begin
  inherited;
  if Assigned(OnUpdate) then
    OnUpdate(Self,Param,Text);
end;

procedure TMyCustomControl.WMActivate(var Message: TWMActivate);
begin
  inherited;
  if Message.Active = 0 then
    Hide;
end;
```

Using the custom inplace editor can be done by putting an component on the form where the parameter control is used and set its visible property to False. The inplace editor will then be displayed when the OnParamCustomEdit event is triggered.