WxSmith tutorial: More on code and preview generation

From Code::Blocks

Contents

Introduction

In previous tutorials we added wxChartCtrl support into wxSmith. It now has some basic and advanced proeprties, it is able to read and store them into xml structures. Now let's continue our wxChart upgrades and affect source code and preview with some advanced properties we've created last time.

Functions which create source code and preview should look quite simillar. That's because souce code generated to item should do the same thing as preview-generating code but it should produce code instead of real object. If you obey this rule, you can be sure that source code generated in wxSmith will give the same effect as shown in preview. The only exception for this would be in case of item which require time-consuming initializaction when we should preffer speed of generating preview over it's accuracy.

Details of Codef function

Main function which should be used to generate source code is Codef function, which is used in current code generating function:

```
void wxsChart::OnBuildCreatingCode()
{
    switch ( GetLanguage() )
    {
        case wxsCPP:
        {
            AddHeader(_T("<wx/chartctrl.h>"),GetInfo().ClassName);

            wxString StyleCode;
            for ( int i=0; Names[i]; i++ )
            {
                if ( m_Flags & Values[i] ) StyleCode << Names[i] << _T("|");
```

```
            }

            if ( StyleCode.IsEmpty() ) StyleCode = _T("0");
            else                       StyleCode.RemoveLast();

            Codef(_T("%C(%W,%I,(STYLE)(%s),%P,%S,%T);\n"),StyleCode.c_str());
            break;
        }

        default:
            wxsCodeMarks::Unknown(_T("wxsChart::OnBuildCreatingCode"),GetLanguage());
    }
}
```

Codef works like Printf function - it takes formating string as first argument and use other arguments to fill the result with dynamic data. In our example we've already used few formating characters but most of them didn't require any extra arguments because wxSmith used some data related to current widget. Codef function uses following scheme: if formating character is capital letter it doesn't need argument (it uses settings of current wxWidget class), if it's lower case, argument is required.

Here's list of all formating characters used in Codef. Note that this function doesn't allow some extra formatting like printf function (for example %08d is not acceptable and simple %d sohuld be used instead).

Formating characters which doesn't require argument

- %A - access prefix

  This formating sequence may be used to access internal functions of current items. It takes care about choosing '->' or '.' operators to access item's internals.
  Example:

  ```
  %AClear();\n
  ```

- %C - create prefix

  This sequence will be replaced by valid prefix generating item. It's expanded to one of following forms:
  - `<VARIABLE NAME> = new <ITEM TYPE>`" in case we operate on item's pointer
  - `<VARIABLE NAME>.Create`" in case we operate on item's instance (note that to use this function, item must provide Create function with arguments set as in constructor, most of wxWidget's internal structures use this scheme)
  - "`Create`" when current item is root item of resource (XRC file format allows any type of item to be root item of resource so wxSmith may add this functionality too)

  Example:

  ```
  %C(%W,%I,(STYLE)(%s),%P,%S,%T);\n
  ```

- %E - get pointer to parent item

  This sequence is replaced with pointer to parent item.
  Example:

  ```
  %ASetParentPointer(%E);\n"
  ```

- %F - get reference to parent item

  This sequence is replaced with parent's object (accessed through ".").
  Example:

```
%ASetParentObject(%F);\n"
```

- %I - get identifier

  This sequence is replaced by item's identifier which can be directly used in source code
  Example:
  ```
  %C(%W,%I,(STYLE)(%s),%P,%S,%T);\n
  ```

- %M - parent's access prefix

  This sequence is replaced by access prefix of parent item. It can be used to easily access parent's members
  Example:
  ```
  %MAddSomeSpeciaChild(%O);\n"
  ```

- %N - get item's name

  This sequence is replaced by item's name (which is now string representation of identifier)
  Example:
  ```
  %C(%W,%I,%P,%S,%T,%N,%V);\n
  ```

- %O - get p**O**inter to current item

  This sequence is replaced by pointer to current item, it may be used as argument for some function when it require pointer to item
  Example:
  ```
  %MAddSomeSpecialChild(%O);\n"
  ```

- %P - get position of current item

  This sequence is replaced by current item's position
  Example:
  ```
  %C(%W,%I,%P,%S,%T,%N,%V);\n
  ```

- %R - get **R**eference to current item

  This sequence is replaced by reference to current item, it may be used as argument for some function when it require reference to item
  Example:
  ```
  %MAddSomeSpecialChildObject(%R);\n"
  ```

- %S - get size of current item

  This sequence is replaced by current item's size
  Example:
  ```
  %C(%W,%I,%P,%S,%T,%N,%V);\n
  ```

- %T - get style of current item

  This sequence is replaced by current item's style
  Example:
  ```
  %C(%W,%I,%P,%S,%T,%N,%V);\n
  ```

- %V - get validator of current item

This sequence is replaced by current item's validator (now it's only wxDefaultValidator but may be expanded in future)
Example:

```
%C(%W,%I,%P,%S,%T,%N,%V);\n
```

- %W - get pointer to parent window

  This sequence is replaced to parent window. The difference between this and %E is that %E may also returo pointers to non-window items such as sizers. %W always returns wxWindow-derived pointer so it may be used for example in constructors
  Example:

```
%C(%W,%I,%P,%S,%T,%N,%V);\n
```

Formating characters that require arguments

- %b - insert boolean value

  This sequence is replaced by representation of boolean in given coding language (for example *true* and *false* in c++)
  Argument type: bool
  Example:

```
%ASetActive(%b);\n
```

- %d - insert decimal value

  This sequence is replaced by representation of decimal integer in given coding language
  Argument type: int
  Example:

```
%ASetShadowWidth(%d);\n
```

- %i - insert image

  This sequence is replaced by code representing image or wxNullBitmap if there's no image.
  Note that two arguments are required for this formating sequence.
  Argument types: wxBitmapData* or wxIconData*; wxChar* (name of art provider used by this image)
  Example:

```
%ASetBitmap(%i);\n
```

- %n - insert non-translated string

  This sequence is replaced with non-translated string representation in given coding language (for example it will be _T("Some\nString") in C++). It takes care about excaping characters not allowed in language
  Argument type: wxChar*
  Example:

```
%ASetArgument(%n);\n
```

- %p - insert custom position

  This sequence works like %P but it adds position from argument, not default

item's position
Argument type: wxPositionData*
Example:

```
%ASetInternalPosition(%p);\n
```

- %s - insert other string

  This sequence works like standard %s in prinf functions
  Argument type: wxChar*
  Example:

```
%ASet%s(0);\n
```

- %t - insert translated string

  This sequence is replaced with translated string representation in given coding language (for example it will be _("Some\nString") in C++). It takes care about excaping characters not allowed in language
  Argument type: wxChar*
  Example:

```
%ASetArgument(%t);\n
```

- %v - insert variable name

  This sequence is replaced by valid variable name sequence in given language. Usually it will simply add string given as argument into code, but some other languages may require extra prefixes (like '$' in php)
  Argument type: wxChar*
  Example:

```
%ASetData(%v);\n
```

- %z - insert custom size

  This sequence works like %S but it adds size from argument, not default item's size
  Argument type: wxSizeData*
  Example:

```
%ASetInternalSize(%z);\n
```

Updating wxChart's code genrating function

Now let's update code generating function. We need to generate wxChartPoints for each set and add all points to this set. At the beginning let's generate code adding sets:

```
    // (1)
    Codef(_T("{\n"));

    for ( size_t i=0; i<m_ChartPointsDesc.Count(); i++ )
    {
        ChartPointsDesc* Desc = m_ChartPointsDesc[i];

        // (2)
        wxString GenStr;
        switch ( Desc->Type )
        {
            case Bar:      GenStr = _T("wxBarChartPoints::CreateWxBarChartPoints"); break;
            case Bar3D:    GenStr = _T("wxBar3DChartPoints::CreateWxBar3DChartPoints"); break;
            case Pie:      GenStr = _T("wxPieChartPoints::CreateWxPieChartPoints"); break;
```

```
            case Pie3D:    GenStr = _T("wxPie3DChartPoints::CreateWxPie3DChartPoints"); break;
            case Points:   GenStr = _T("wxPointsCharPoints::CreateWxPointsChartPoints"); break;
            case Points3D: GenStr = _T("wxPoints3DCharPoints::CreateWxPoints3DChartPoints"); break;
            case Line:     GenStr = _T("wxLineCharPoints::CreateWxLineChartPoints"); break;
            case Line3D:   GenStr = _T("wxLine3DCharPoints::CreateWxLine3DChartPoints"); break;
            case Area:     GenStr = _T("wxAreaCharPoints::CreateWxAreaChartPoints"); break;
            case Area3D:   GenStr = _T("wxArea3DCharPoints::CreateWxArea3DChartPoints"); break;
            default:       GenStr = _T("wxBarChartPoints::CreateWxBarChartPoints"); break;
        }

        // (3)
        wxString VarStr = wxString::Format(_T("PointSet%d"),(int)i);

        // (4)
        Codef(_T("\twxChartPoints* %v = %s(%t);\n"),VarStr.c_str(),GenStr.c_str(),Desc->Name.c_str());

        // (5)
        Codef(_T("\t%AAdd(%v);\n"),VarStr.c_str());
    }

    // (6)
    Codef(_T("}\n"));
```

First thing we add into code is opening brace *(1)*. We do this because we will generate some local variables for this item only and names of these variables may overlap with names used for another wxChartCtrl class used in same resource. Adding { } block will make them local so we would be able to duplicate their names. Next we create code which generate set *(2)*. wxChartCtrl doesn't allow using direct `new` operator on sets so we have to generate it using static function. In point *(3)* we generate local variable name for set and produce code for it *(4)*. Note that when any string data is required in Codef() functon, it must be extracted from wxString using `c_str()` just like in wxString::Format. Also note that we don't put variable name directly into source code. We use %v which will convert variable name to language-friendly format (that would help adding support for another languages). Finally we add new set into wxChartCtrl *(5)* and close block created in *(1)* with '}' *(6)*.

Now let's add points into sets, we will add new code between *(4)* and *(5)*:

```
    for ( size_t j=0; j<Desc->Points.Count(); j++ )
    {
        wxString PointStr = wxString::Format(_T("%lf,%lf"),Desc->Points[j]->X,Desc->Points[j]->Y);
        Codef(_T("\t%v->Add(%t,%s);\n"),VarStr.c_str(),Desc->Points[j]->Name.c_str(),PointStr.c_str());
    }
```

Note that we first generate string for point coodrinates. That's because Codef function doesn't support double values yet.

Some problems

While generating source code I've encountered one problem I haven't noticed so far. It looks like wxChart code doesn't provide all chart types now, charts which are not supported are: Points, Points3D, Lines, Lines3D, Area and Area3D. Because these types may be added in future we will only block them in functions loading xml and generating proeprty grid. Adding them in future will be equal to removing few comments. We change code in OnXmlRead into this:

```
       if ( Type == _T("pie") )      Desc->Type = Pie;      else
       if ( Type == _T("pie3d") )    Desc->Type = Pie3D;    else
       /*
```

```
        if ( Type == _T("points") )   Desc->Type = Points;   else
        if ( Type == _T("points3d") ) Desc->Type = Points3D; else
        if ( Type == _T("line") )     Desc->Type = Line;     else
        if ( Type == _T("line3d") )   Desc->Type = Line3D;   else
        if ( Type == _T("area") )     Desc->Type = Area;     else
        if ( Type == _T("area3d") )   Desc->Type = Area3D;   else
        */
                                      Desc->Type = Bar;
```

and code in AppendPropertyForSet into this:

```
    static const wxChar* Types[] =
    {
        _T("Bar"),    _T("Bar3D"),    _T("Pie"),  _T("Pie3D"),
        NULL, // wxChartCtrl doesn't support all types yet
        _T("Points"), _T("Points3D"), _T("Line"), _T("Line3D"),
        _T("Area"),   _T("Area3D"),   NULL
    };
```

Affecting preview

Since we have support for source code, let's update preview so we would be able to see results in real time during editing. As base code we will use code-generating function to make those two functions simillar:

```
wxObject* wxsChart::OnBuildPreview(wxWindow* Parent,long Flags)
{
    long RealFlags = m_Flags;
    if ( RealFlags & DEFAULT_STYLE_FIX ) RealFlags |= DEFAULT_STYLE;
    wxChartCtrl* Chart = new wxChartCtrl(Parent,GetId(),
(STYLE)RealFlags,Pos(Parent),Size(Parent),Style());

    for ( size_t i=0; i<m_ChartPointsDesc.Count(); i++ )
    {
        ChartPointsDesc* Desc = m_ChartPointsDesc[i];
        wxChartPoints* Points = NULL;

        switch ( Desc->Type )
        {
            case Bar:      Points = wxBarChartPoints::CreateWxBarChartPoints(Desc->Name); break;
            case Bar3D:    Points = wxBar3DChartPoints::CreateWxBar3DChartPoints(Desc->Name); break;
            case Pie:      Points = wxPieChartPoints::CreateWxPieChartPoints(Desc->Name); break;
            case Pie3D:    Points = wxPie3DChartPoints::CreateWxPie3DChartPoints(Desc->Name); break;
            /*
            case Points:   Points = wxPointsCharPoints::CreateWxPointsChartPoints(Desc->Name); break;
            case Points3D: Points = wxPoints3DCharPoints::CreateWxPoints3DChartPoints(Desc->Name); break;
            case Line:     Points = wxLineCharPoints::CreateWxLineChartPoints(Desc->Name); break;
            case Line3D:   Points = wxLine3DCharPoints::CreateWxLine3DChartPoints(Desc->Name); break;
            case Area:     Points = wxAreaCharPoints::CreateWxAreaChartPoints(Desc->Name); break;
            case Area3D:   Points = wxArea3DCharPoints::CreateWxArea3DChartPoints(Desc->Name); break;
            */
            default:       Points = wxBarChartPoints::CreateWxBarChartPoints(Desc->Name); break;
        }

        for ( size_t j=0; j<Desc->Points.Count(); j++ )
        {
            Points->Add(Desc->Points[j]->Name,Desc->Points[j]->X,Desc->Points[j]->Y);
        }

        Chart->Add(Points);
    }
    return Chart;
}
```

I've done simply Copy&Paste, few replaces and it's done :). To compile new source we have only to add extra includes in the beginning of source file:

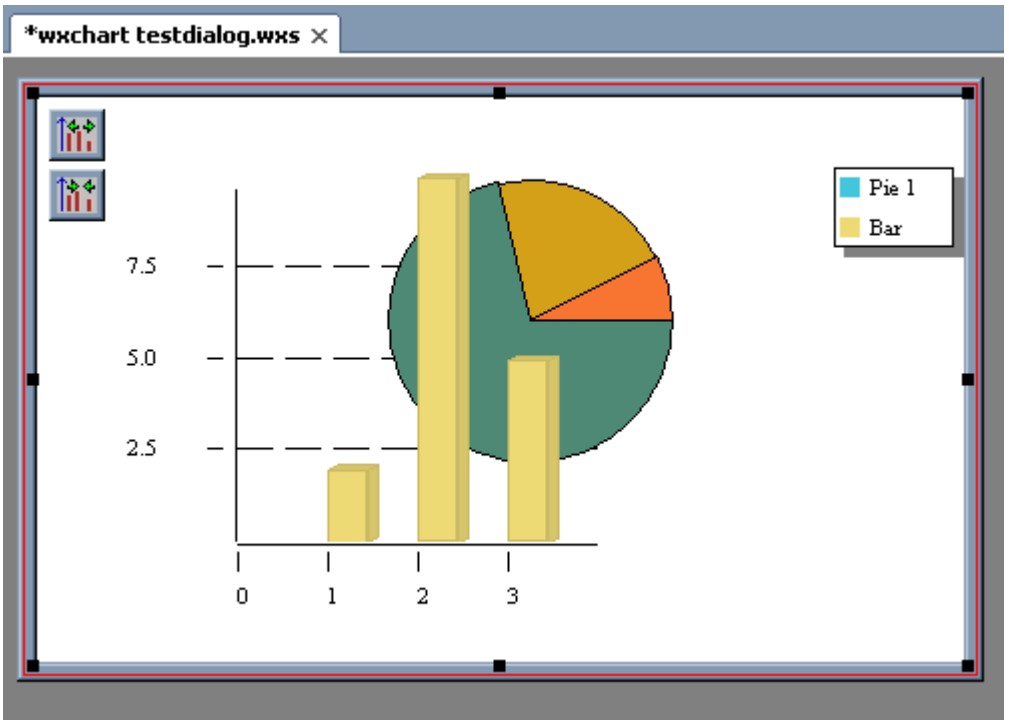
```
#include <wx/barchartpoints.h>
#include <wx/bar3dchartpoints.h>
#include <wx/piechartpoints.h>
#include <wx/pie3dchartpoints.h>
```

These includes should also be listed in list of headers included in generated source code so we enumerate them at the beginning OnBuildCreatingCode:

```
void wxsChart::OnBuildCreatingCode()
{
    switch ( GetLanguage() )
    {
        case wxsCPP:
        {
            AddHeader(_T("<wx/chartctrl.h>"),GetInfo().ClassName);
            AddHeader(_T("<wx/barchartpoints.h>"),_T(""),hfLocal);
            AddHeader(_T("<wx/bar3dchartpoints.h>"),_T(""),hfLocal);
            AddHeader(_T("<wx/piechartpoints.h>"),_T(""),hfLocal);
            AddHeader(_T("<wx/pie3dchartpoints.h>"),_T(""),hfLocal);
            ....
```

Note here, that all extra headers was added with extra hfLocal flag. That's because we need them only when generating item, we don't require them in class header.

And now we can test our enhanced chart. It looks pretty well :



At the end of wxChart trip ;)

In last four tutorials we were working on wxChart extension added into wxSmith. We covered almost all ascepts of this item including dynamic properties, nice preview and full source code generation. This item has been aded into Code::Blocks's source code and is available in following directory:

```
src/plugins/contrib/wxSmithContribItems/wxchart
```

These sources may differ a little bit from the ones used in this tutorial an also may be udpated in future to support other properties not covered yet. But it's always based on source code we created in last tutorials.

If you have adopted some other widget and want to see it inside Code::Blocks, contact me (byo) through forum and I'll officially add it into source code :)