WxSmith tutorial: Adding basic properties into items

From Code::Blocks

Contents

Preface

In previous tutorial we created simple plugin which added one item into wxSmith's palette. After loading this plugin into Code::Blocks we were able to add this item into resource and set some basic properties like size or position. In this tutorial we will add more customization for this item by adding it's specific properties. But before we do this, I need to explain some things.

How wxSmith creates properties

The final set of properties is result of merging three basic sets:

- Some standard properties which are used in almost any widget (size, position, font etc)
- Properties added by parent control like border size inside sizers or notebook page name
- Properties which are specific to one widget only

Standard properties

The first set is added through constructor. In the code from previous tutorial, constructor is as follows:

```
wxsChart::wxsChart(wxsItemResData* Data):
    wxsWidget(
        Data,               // Data passed to constructor
        &Reg.Info,          // Info taken from Registering object previously created
        NULL,               // Structure describing events, we have no events for wxChart
        wxsChartStyles)     // Structure describing styles
{
}
```

so the only thing we see here is set of styles (that's one of standard properties). Because it's rather complex and require extra initialization, this property must be specified separately. If there was `NULL` instead of `wxsChartStyles`, wxChart would have no style property. Other standard properties are given through flags, which are passed through `PropertiesFlags` argument in constructor of wxsWidget:

```
wxsWidget(
    wxsItemResData* Data,
    const wxsItemInfo* Info,
    const wxsEventDesc* EventArray = NULL,
    const wxsStyleSet* StyleSet=NULL,
    long PropertiesFlags = flWidget);
```

We can see here that default value for this argument is flWidget. It means that standard properties which are usually used by any kind of widget will be added into properties set. That value is silently passed to wxsWidget's constructor in case of wxsChart class so we have all those standard properties in property browser without even line of code. Individual flags which can be used are:

- `flVariable` - allow variable and *Is Member* flag
- `flId` - allow identifier
- `flPosition` - allow position
- `flSize` - allow size
- `flEnabled` - allow *Enabled* switch
- `flFocused` - allow *Focused* switch
- `flHidden` - allow *Hidden* switch
- `flColours` - allow foreground and background colours
- `flToolTip` - allow tooltip
- `flFont` - allow font
- `flHelpText` - allow help text
- `flSubclass` - allow changing of class name (the "*subclass*" term comes from XRC terminology)
- `flMinMaxSize` - allow MinSize and MaxSize
- `flExtraCode` - allow property with extra source code added right after item is

generated

And when we want for example only variable and position, we would pass `flVariable|flPosition` instead of `flWidget` into wxsWidget's contructor. This technique is not recommended though, because it may introduce some problems (just imagine widget which have no variable of any kind - there's problem even with creating it).

Some properties are also filtered depending on some circumstances:

- `flMinMaxSize` will be disabled when XRC file is used
- `flVariable` / `flId` are disabled for top-most items (like wxDialog and wxFrame)
- `flHidden` does not affect item when it's shown inside editor (but it's used for exact previews shown after pressing *Preview* button)

Properties added by parent control

Some parent controls require extra information for each of it's children. This data is included in set of properties in property browser and child item have no control of it. Properties added in this way are:

- Settings for item inside sizer (*border size, border flags, proportion* and some flags like *expand')*
- Names of pages and switch for default page in wxNotebook, wxChoicebook and wxListbook

Properties specific to only one item

Most of items have their own proeprties which can not be generalized and provided as common ones. These properties are added in special function which in case of widgets is declared as:

```
void OnEnumWidgetProperties(long Flags);
```

wxSmith has complex property managment system which handle most of work related to properties automatically, including operating on property grid and XRC files. Usually one variable is mapped to one property (in case of most complex properties such as position, special structure is used to keep data in one variable). Binding between variable and property managment system is done in function mentioned above. It require generating one variable for one property and calling `Property` function for it. Usually ot looks like this:

```
void ClassName::OnEnumWidgetProperties(long Flags)
{
    static wxsBoolProperty SomePropertyVariable(
                            _("Name in property grid"), _T("name_in_xrc_file"),
                            wxsOFFSET(ClassName,ClassMemberVariable), false );
    Property(SomePropertyValue);
}
```

Properties may be added in this way or may be defined through macros. Using macro, the code adding bool property will look like this:

```
void ClassName::OnEnumWidgetProperties(long Flags)
{
    WXS_BOOL(ClassName,ClassMemberVariable,_("Name in property grid"), _T("name_in_xrc_file"), false);
}
```

Both methods will give the same result (macros simply generate static variable and call Property on it). So it's up to you to choose one of them. I preffer macros because they give cleaner view, but smoetimes they may hide some extra parameters and may not be preffered by other coders (not everyone likes c macros ;)). So I'll describe both methods.

Before we begin detailed description of available properties, I'll have to make few remarks:

- Variables used in properties must be members of the class which adds them (it's possible to use members of other class, but it's advanced technique and won't be covered in this tutorial). So it's not possible to create property from pointer to variable (for example when we have `bool*` member in class and want to make property for value pointer by it). The technique which allow such things will be described later in this tutorial but won't use wxSmith's property managment system.
- Objects which create binging between property variable and property managment system (like `SomePropertyVariable` in example above) should be declared static or at least should exist as long as all objects using this property are not deleted. Simple `static` modifier eliminates all problems.

List of available properties

Here you have list of properties supported by wxSmith's property managment system. Each one of them have exact type of variable it supports and examples of using this property. In those examples, `ClassName` represents name of class containing properties, `ClassMemberVariable` represents name of member variable inside `ClassName` (note that current there's no type checking so you should check if valid types of variables are used).

Here you have list of properties supported by wxSmith's property managment system. Each one of them have exact type of variable it supports and examples of using this property. In those examples, `ClassName` represents name of class containing properties and `ClassMemberVariable` represents name of member variable inside `ClassName` (note that current there's no type checking so you should check if valid types of variables are used).

bool

Variable type: `bool`

Example using macros:

```
WXS_BOOL(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    false);                       // Default value
```

Example using objects:

```
static wxsBoolProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable
    false );                                  // Default value
Property(SomePropertyValue);
```

wxString

Variable type: `wxString`

Example using macros (for one-line strings):

```
WXS_SHORT_STRING(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T(""),                       // Default value
    false);                       // If set to true, empty strings won't be stored in XRC files (
```

Example using macros (for multiline strings):

```
WXS_STRING(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T(""),                       // Default value
    false);                       // If set to true, empty strings won't be stored in XRC files
```

Example using objects:

```
static wxsStringProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable
    true,                                     // True for multiline string, false for one-line
    _T("") );                                 // Default value
Property(SomePropertyValue);
```

long int

Variable type: `long int`

Example using macros:

```
WXS_LONG(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    10);                          // Default value
```

Example using objects:

```
static wxsLongProperty SomePropertyVariable(
    _("Name in property grid"),              // Name of property shown in property grid
    _T("name_in_xrc_file"),                  // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable
    10);                                     // Default value
Property(SomePropertyValue);
```

enum

Variable type: `long int` which is one of predefined values

Example using macros:

```
static const long    Values[] = { 5, 100 };
static const wxChar* Names[]  = { _("Few"), _("Many"), NULL }; // Must end with NULL entry

WXS_ENUM(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    Values,                       // Array of values
    Names,                        // Array of names
    5);                           // Default value
```

Example using objects:

```
static const long    Values[] = { 5, 100 };
static const wxChar* Names[]  = { _("Few"), _("Many"), NULL }; // Must end with NULL entry

static wxsEnumProperty SomePropertyVariable(
    _("Name in property grid"),              // Name of property shown in property grid
    _T("name_in_xrc_file"),                  // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable
    Values,                                  // Array of values
    Names,                                   // Array of names
    false,                                   // Setting to true notifies that content of arrays may
change (not tested)
    5,                                       // Default value
    true);                                   // If false, XRC node will contain value as number, if
true, it will contain name
Property(SomePropertyValue);
```

flags

Variable type: `long int` which is value produced by or-ing some one-bit flags

Example using macros:

```
static const long    Values[] = { 0x01, 0x02, 0x04, 0x08, 0x10 };
static const wxChar* Names[]  = { _T("C"), _T("C++"), _T("Java"), _T("C#"), _T("D"), NULL };
```

```
WXS_FLAGS(
    ClassName,                  // Name of this class
    ClassMemberVariable,        // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    Values,                     // Array of values
    Names,                      // Array of names
    0x01 | 0x10 );              // Default value
```

Example using objects:

```
static const long    Values[] = { 0x01, 0x02, 0x04, 0x08, 0x10 };
static const wxChar* Names[]  = { _T("C"), _T("C++"), _T("Java"), _T("C#"), _T("D"), NULL };
```

```
static wxsFlagsProperty SomePropertyVariable(
    _("Name in property grid"),              // Name of property shown in property grid
    _T("name_in_xrc_file"),                  // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable
    Values,                                  // Array of values
    Names,                                   // Array of names
    false,                                   // Setting to true notifies that content of arrays may
change (not tested)
    0x01 | 0x10,                             // Default value
    true);                                   // If false, XRC node will contain value as number, if
true, it will contain names separated by '|' character
Property(SomePropertyValue);
```

wxArrayString

Variable type: `wxArrayString`

Example using macros:

```
WXS_ARRAYSTRING(
    ClassName,                  // Name of this class
    ClassMemberVariable,        // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T("xrc_subnode"));         // Name of sub-node generated for each string in xrc file
```

Example using objects:

```
static wxsArrayStringProperty SomePropertyVariable(
    _("Name in property grid"),              // Name of property shown in property grid
    _T("name_in_xrc_file"),                  // Name of XML node used in xrc files
    _T("xrc_subnode"),                       // Name of sub-node generated for each string in xrc file
    wxsOFFSET(ClassName,ClassMemberVariable));// Offset from class beginning to variable
Property(SomePropertyValue);
```

wxArrayString with checked items

Variable type: `wxArrayString` with `wxArrayBool`

Example using macros:

```
WXS_ARRAYSTRINGCHECK(
    ClassName,                    // Name of this class
    ClassMemberVariable1,         // Name of member variable of type wxArrayString
    ClassMemberVariable2,         // Name of member variable of type wxArrayBool
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T("xrc_subnode"));           // Name of sub-node generated for each string in xrc file
```

Example using objects:

```
static wxsArrayStringCheckProperty SomePropertyVariable(
    _("Name in property grid"),                // Name of property shown in property grid
    _T("name_in_xrc_file"),                    // Name of XML node used in xrc files
    _T("xrc_subnode"),                         // Name of sub-node generated for each string in xrc file
    wxsOFFSET(ClassName,ClassMemberVariable1),  // Offset from class beginning to variable
                                               // of type wxArrayString
    wxsOFFSET(ClassName,ClassMemberVariable2)); // Offset from class beginning to variable
                                               //of type wxArrayBool
Property(SomePropertyValue);
```

wxBitmap

Variable type: `wxsBitmapData`

Example using macros:

```
WXS_BITMAP(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable of type wxBitmapData
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T("wxART_OTHER"));           // Name of default art client (see wx docs (http://www.wxwidgets.org/ma
nuals/2.6/wx_wxartprovider.html#wxartprovider))
```

Example using objects:

```
static wxsBitmapProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable,  // Offset from class beginning to variable of type
wxsBitmapData
    _T("wxART_OTHER"));                       // Name of default art client (see wx docs (http://www.wxwi
dgets.org/manuals/2.6/wx_wxartprovider.html#wxartprovider))
Property(SomePropertyValue);
```

wxIcon

Variable type: `wxsIconData`

Example using macros:

```
WXS_ICON(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable of type wxBitmapData
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"),       // Name of XML node used in xrc files
    _T("wxART_OTHER"));           // Name of default art client (see wx docs (http://www.wxwidgets.org/ma
nuals/2.6/wx_wxartprovider.html#wxartprovider))
```

Example using objects:

```
static wxsIconProperty SomePropertyVariable(
    _("Name in property grid"),              // Name of property shown in property grid
    _T("name_in_xrc_file"),                  // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable), // Offset from class beginning to variable of type
wxsIconData
    _T("wxART_OTHER"));                      // Name of default art client (see wx docs (http://www.wxwi
dgets.org/manuals/2.6/wx_wxartprovider.html#wxartprovider))
Property(SomePropertyValue);
```

wxColour

Variable type: `wxsColourData`

Example using macros:

```
WXS_COLOUR(
    ClassName,                   // Name of this class
    ClassMemberVariable,         // Name of member variable
    _("Name in property grid"),  // Name of property shown in property grid
    _T("name_in_xrc_file"));     // Name of XML node used in xrc files
```

Example using objects:

```
static wxsColourProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable)); // Offset from class beginning to variable
Property(SomePropertyValue);
```

wxDimension

Variable type: `wxsDimensionData`

Example using macros:

```
WXS_DIMENSION(
    ClassName,                   // Name of this class
    ClassMemberVariable,         // Name of member variable
    _("Name in property grid"),  // Name of property shown in property grid
    _("Dialog-Units name"),      // Name of dialog units (http://www.wxwidgets.org/manuals/2.6/wx_wxwind
ow.html#wxwindowconvertpixelstodialog) switch entry in property grid
    _T("name_in_xrc_file"),      // Name of XML node used in xrc files
    10,                          // Default value
    true);                       // Default value for dialog units (http://www.wxwidgets.org/manual
s/2.6/wx_wxwindow.html#wxwindowconvertpixelstodialog) switch
```

Example using objects:

```
static wxsDimensionProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _("Name of Dialog-Units"),                // Name of dialog units (http://www.wxwidgets.org/manual
s/2.6/wx_wxwindow.html#wxwindowconvertpixelstodialog) switch in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable),  // Offset from class beginning to variable
    10,                                       // Default value
    true);                                    // Default value for dialog units (http://www.wxwidgets.or
g/manuals/2.6/wx_wxwindow.html#wxwindowconvertpixelstodialog) switch
```

```
Property(SomePropertyValue);
```

wxFont

Variable type: `wxsFontData`

Example using macros:

```
WXS_FONT(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name in property grid"),   // Name of property shown in property grid
    _T("name_in_xrc_file"));      // Name of XML node used in xrc files
```

Example using objects:

```
static wxsFontProperty SomePropertyVariable(
    _("Name in property grid"),               // Name of property shown in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable)); // Offset from class beginning to variable
Property(SomePropertyValue);
```

wxPosition

Variable type: `wxsPositionData`

Example using macros:

```
WXS_POSITION(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name of Default switch"),  // Name of default position switch in property grid
    _("Name of X data"),          // Name of X coordinate in property grid
    _("Name of Y data"),          // Name of Y coordinate in property grid
    _("Name of Dialog-Units"),    // Name of dialog units (http://www.wxwidgets.org/manuals/2.6/wx_wxwind
ow.html#wxwindowconvertpixelstodialog) switch in property grid
    _T("name_in_xrc_file"));      // Name of XML node used in xrc files
```

Example using objects:

```
static wxsPositionProperty SomePropertyVariable(
    _("Name of Default switch"),              // Name of default position switch in property grid
    _("Name of X data"),                      // Name of X coordinate in property grid
    _("Name of Y data"),                      // Name of Y coordinate in property grid
    _("Name of Dialog-Units"),                // Name of dialog units (http://www.wxwidgets.org/manual
s/2.6/wx_wxwindow.html#wxwindowconvertpixelstodialog) switch in property grid
    _T("name_in_xrc_file"),                   // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable)); // Offset from class beginning to variable
Property(SomePropertyValue);
```

wxSize

Variable type: `wxsSizeData`

Example using macros:

```
WXS_SIZE(
    ClassName,                    // Name of this class
    ClassMemberVariable,          // Name of member variable
    _("Name of Default switch"),  // Name of default position switch in property grid
    _("Name of Width data"),      // Name of Width value in property grid
    _("Name of Height data"),     // Name of Height value in property grid
    _("Name of Dialog-Units"),    // Name of dialog units (http://www.wxwidgets.org/manuals/2.6/wx_wxwind
ow.html#wxwindowconvertpixelstodialog) switch in property grid
    _T("name_in_xrc_file"));      // Name of XML node used in xrc files
```

Example using objects:

```
static wxsSizeProperty SomePropertyVariable(
    _("Name of Default switch"),               // Name of default position switch in property grid
    _("Name of Width data"),                   // Name of Width value in property grid
    _("Name of Height data"),                  // Name of Height vlaue in property grid
    _("Name of Dialog-Units"),                 // Name of dialog units (http://www.wxwidgets.org/manual
s/2.6/wx_wxwindow.html#wxwindowconvertpixelstodialog) switch in property grid
    _T("name_in_xrc_file"),                    // Name of XML node used in xrc files
    wxsOFFSET(ClassName,ClassMemberVariable)); // Offset from class beginning to variable
Property(SomePropertyValue);
```

Fully customizable properties

The set of properties available in wxSmith's property managment system may be insufficient for some kind of properties. A good example here may be a situation, when number of available properties dynamically changes. In such cases, we have to take care about properties manually. This technique is shown in next tutorial: Adding advanced properties

Updating wxChart item

Ok, now if we have some theoretical knowledge, it's high time to do some real example. We will update wxChart item and add some more properties into it.

First thing is that extra style we wasn't able to provide last time. We will use flags property for it. Previously we provided empty function body for `wxsChart::OnEnumWidgetProperties` and now it's time to fill it.

To build flags property, we need values for it and it's names. For values array we could use direct values of flags as they are declared in wx/chartctrl.h header file, but this has one disadventage: We won't be able to use DEFAULT_STYLE flag since it contains few other flags inside. So we will have to create out custom flag for that grouping style:

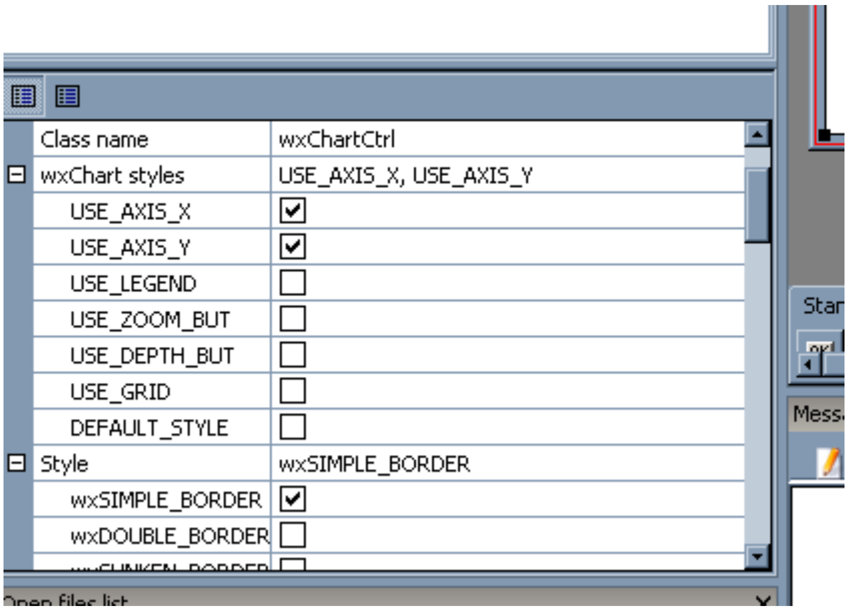
```
void wxsChart::OnEnumWidgetProperties(long Flags)
{
    static const long DEFAULT_STYLE_FIX = 0x1000;  // Fixed flag value for DEFAULT_STYLE
    static const long Values[] = { USE_AXIS_X, USE_AXIS_Y, USE_LEGEND, USE_ZOOM_BUT,
                                   USE_DEPTH_BUT, USE_GRID, DEFAULT_STYLE_FIX };

    static const wxChar* Names[] = { _T("USE_AXIS_X"), _T("USE_AXIS_Y"), _T("USE_LEGEND"),
                                     _T("USE_ZOOM_BUT"), _T("USE_DEPTH_BUT"), _T("USE_GRID"),
                                     _T("DEFAULT_STYLE"), NULL };

    WXS_FLAGS(wxsChart,m_Flags,_("wxChart styles"),_T("wxchart_styles"),Values,Names, DEFAULT_STYLE_FIX )
}
```

So the first thing we do here is to create arrays with values and names for them. We used DEFAULT_STYLE_FIX for last value to allow DEFAULT_STYLE too but in names we used standard name. Also note that names array MUST end with NULL entry to mark end of array. Another thing worth mentioning is that we also have to add `long m_Flags` member into `wxsChar` Class

Ok, let's recompile and test our result:



Class name	wxChartCtrl
⊟ wxChart styles	USE_AXIS_X, USE_AXIS_Y
USE_AXIS_X	✓
USE_AXIS_Y	✓
USE_LEGEND	☐
USE_ZOOM_BUT	☐
USE_DEPTH_BUT	☐
USE_GRID	☐
DEFAULT_STYLE	☐
⊟ Style	wxSIMPLE_BORDER
wxSIMPLE_BORDER	✓
wxDOUBLE_BORDER	☐

So we have our property working. Now it's time to make in affect preview and generated code. But before we do this, let's move those arrays used to create property into global scope (into unnamed namespace) because we will need them in other functions. Also we need to set initial value for m_Flags member, so let's set it to DEFAULT_STYLE_FIX to include DEFAULT_STYLE when creating new charts.

Code after change should look like this:

```
...
    // Defining styles
    WXS_ST_BEGIN(wxsChartStyles,_T("wxSIMPLE_BORDER"))
        WXS_ST_DEFAULTS()
    WXS_ST_END()


    static const long DEFAULT_STYLE_FIX = 0x1000;
    static const long Values[] = { USE_AXIS_X, USE_AXIS_Y, USE_LEGEND, USE_ZOOM_BUT,
                                   USE_DEPTH_BUT, USE_GRID, DEFAULT_STYLE_FIX };
    static const wxChar* Names[] = { _T("USE_AXIS_X"), _T("USE_AXIS_Y"), _T("USE_LEGEND"),
                                     _T("USE_ZOOM_BUT"), _T("USE_DEPTH_BUT"), _T("USE_GRID"),
                                     _T("DEFAULT_STYLE"), NULL };

}

wxsChart::wxsChart(wxsItemResData* Data):
    wxsWidget(
        Data,               // Data passed to constructor
        &Reg.Info,          // Info taken from Registering object previously created
        NULL,               // Structure describing events, we have no events for wxChart
        wxsChartStyles)     // Structure describing styles
{
    m_Flags = DEFAULT_STYLE_FIX;
}
...
```

Ok, now we can change function which builds source code. now it looks like this:

```
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());
    }
}
```

First we iterate through all styles and test if given style is set (note we have to use braces because we declare local variable inside switch statement). If it is, we add style's name ended with '|' operator (to allow more styles). Finally we have to check if string containing styles is empty. If it is, we replace it with zero (because we can not put empty string in place of function's argument). If it's not empty (at least one style was added), we have to remove unnecessary '|' character at the end of string by calling `RemoveLast()`. In Codef function we use %s formating sequence, just like in standard printf function. To prevent compile errors, we convert produced style to STYLE enum.

Now let's affect preview. This will be much easier since we almost have exact values:

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

The only thing which have to be fixed is that if DEFAULT_STYLE_FIX flag is set, we "unfix" that by or-ing value with DEFAULT_STYLE.

Ok, let's find out how it works... everything works fine :). My first impression was that there's something wrong, but wxChart seems to interpret properties in strange way. This is probably caused by fact that there's no chart data to be displayed yet.

Our wxChartCtrl item inside wxSmith is more customizable now, but it still misses the ability to add some data into it. That problem wil be covered in next tutorial.