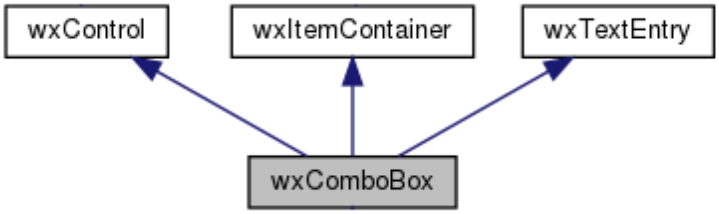
# Chimeric Pointer

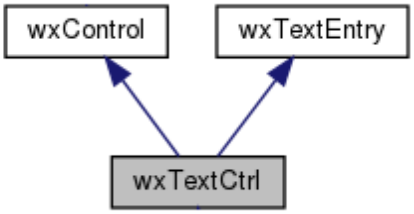
Earlier today I was writing code for a graphical user interface program on a desktop PC using the **wxWidgets** framework. I had designed a dialog box that had several widgets, in particular text boxes and combo boxes.

I wanted to write a function that could manipulate either a combo box or a text box, as follows:

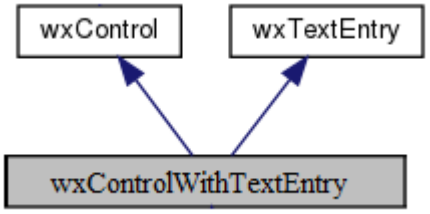
```
void Red(wxControl *const p)
{
    p->SetBackgroundColour( *wxRED );
    p->SetValue("pending");
    p->Refresh();
}
```

This function didn't compile however, because '*SetValue*' is not a member function of the class **wxControl**. So I took a look at the **wxWidgets** documentation and I found the class hierarchy diagrams:




Text boxes and combo boxes both inherit the '*SetValue*' method from **wxTextEntry**. It would have been nice if there were an intermediate class called **wxControlWithTextEntry** as follows:




because then **wxComboBox** and **wxTextCtrl** could inherit from it, and so then my function '*Red*' above could take a pointer to a **wxControlWithTextEntry**.

Not wanting to write a template function, I had to re-write my function as follows:

```
void Red(wxControl *const pC, wxTextEntry *const pT)
{
    pC->SetBackgroundColour( *wxRED );
    pT->SetValue("pending");
    pC->Refresh();
}
```

and then invoke it as follows:

```
wxTextCtrl *const p1 = new wxTextCtrl;
Red(p1,p1);

wxComboBox *const p2 = new wxComboBox;
Red(p2,p2);
```

This is when I came up with the idea of a '*chimeric pointer*'. A chimeric pointer would work as follows:

```
void Red( chimeric_pointer<wxControl,wxTextEntry> p )
{
    p->SetBackgroundColour( *wxRED );
    p->SetValue("pending");
    p->Refresh();
}
```

The way that this chimeric pointer would work is as follows:

**(1)** When defining a chimeric pointer, you specify all of the base classes you require, for example:
`chimeric_pointer<wxControl,wxTextEntry> p;`
**(2)** When assigning to a chimeric pointer, the expression on the right-hand side must be a pointer to a class which can convert implicitly to a pointer to all of the base classes specified in the first point above, otherwise the compiler will terminate compilation and issue a diagnostic message.
**(3)** When you apply the '`->`' operator to a chimeric pointer and then try to access a member object or a member function, the compiler tries to find the member in all the base classes specified in the first point above.

So for example, in the above code snippet where I have:

```
p->SetBackgroundColour( *wxRED );
```

The compiler searches for a member named '*SetBackgroundColour*' in **wxControl**, and it successfully finds such a method and invokes it.

For the next example, let's take the next line:

```
p->SetValue("pending");
```

The compiler searches for a member called '*SetValue*' in **wxControl**, and it fails to find it. So next it searches for a member called '*SetValue*' in **wxTextEntry**, and it finds it and invokes it.

If the compiler cannot find the member inside any of the base classes, then compilation is terminated and the compiler must issue a diagnostic.