

18. Containers

18.1. Prerequisites and Objectives

Before starting this chapter you should have:

1. A knowledge of programming in C++.
2. A knowledge of component interface design.

After completing this chapter you will have:

1. Some knowledge of container components.

18.2. Introduction

A Container is a component that can have other components embedded in it. The embedding can be static or dynamic. In static embedding, the collection of sub-components is fixed when the Container is compiled. In dynamic embedding, new components can be added by the programmer after the component has been compiled. It is possible to embed one container in another to create arbitrarily complex hierarchies.

Although hierarchy is a useful concept in most kinds of programming, its usefulness in component-level programming has yet to be firmly established. Nor is it clear that existing methods for developing component hierarchies are adequate to support true hierarchical programming. In the ActiveX technology, for example, it is possible to create a simple container component, but the components embedded in the container will appear to be embedded in the container's host. Because the container does not establish a new name space, true hierarchical programming is not supported. Other technologies,

most notably JavaBeans, do a better job of supporting hierarchy, but it is still not clear whether this hierarchy is of any benefit to the programmer.

It is likely that hierarchy will become increasingly important as we learn more about component-level programming. For that reason we will demonstrate a dynamic Container that permits programmers to group components in another component window. This will permit components to be moved as a group, but this organization is purely visual in nature. The components embedded in this container must be accessed as if they were embedded in the container's host.

18.3. The Methodology

At the present time, the concept of hierarchy has not established itself well enough in component level programming to permit the exposition of a reasonable methodology for container design. We will give an example to illustrate this type of component. The design of this component is straightforward.

18.4. A Simple Container

Because adding container support to a component is handled by the component design tools in Visual C++, the Simple Container has roughly the same complexity as a "Hello World" component. We will add two properties, *Caption* and *BackColor*, and create a suitable drawing routine, but beyond that little or nothing is required. When creating the project, we will check the "Acts as a simple frame control" check box on the second page of the MFC ActiveX wizard. This will include the necessary features to support component embedding. The entire implementation of the component is given in

Figure 18.2. Figure 18.1 contains the description of the *Caption* and *BackColor* properties.

Property Design Table		
Name	Type	Function
Caption	String Default="Drop Controls Here"	Caption displayed across the top of the container component.
BackColor	Color Default=White	The background color of the component.

Figure 18.1. The Simple Container Property Design Table.

```

void CSimpleContainerCtrl::OnBackColorChanged()
{
    CWnd::Invalidate();
    SetModifiedFlag();
}

BSTR CSimpleContainerCtrl::GetCaption()
{
    return m_caption.AllocSysString();
}

void CSimpleContainerCtrl::SetCaption(LPCTSTR lpszNewValue)
{
    m_caption = lpszNewValue;
    CWnd::Invalidate();
    SetModifiedFlag();
}

void CSimpleContainerCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    CBrush MyBrush;
    COLORREF BackC;
    OleTranslateColor(m_backColor, NULL, &BackC);
    MyBrush.CreateSolidBrush(BackC);
    // paint the background
    pdc->FillRect(rcBounds, &MyBrush);
    if (!m_caption.IsEmpty())
    {
        // paint the text
        pdc->SaveDC();
        pdc->SetBkColor(BackC);
        pdc->TextOut(0,0,m_caption);
        pdc->RestoreDC(0);
    }
    // clean up
    MyBrush.DeleteObject();
}

```

Figure 18.2. The Implementation of the Simple Container.

18.5. Conclusion

At present, containers are useful for physically grouping components in a layout, however it is not yet clear whether hierarchical design will be useful at the component level. Presently containers are not particularly important, but this may change as we learn more about component level programming.