How to Modify the Context Menu in OfficeBean

Maîtriser le menu contextuel dans un OfficeBean
Version 1.0
First edition: November 2003
First English edition: February 2004

Contents

Overview

About this guide

This guide explains how to use Java Beans to modify the context menu of an application that uses OpenOffice.org Writer. To understand this guide, you need to be proficient in object-oriented programming and you need to have a basic familiarity with the OpenOffice.org SDK. You will also need to refer to the SDK Developer's Guide, which is available at http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html

Conventions used in this guide

This guide uses the following conventions:

- Code examples are presented on a shaded background, like this:

```
// Obtain the global MultiServiceFactory and store it in mServiceFactory
```

- **Bold** within code examples indicates statements that you should pay particular attention to.

Copyright and trademark information

Feedback

Please direct any comments or suggestions about this document to dev@documentation.openoffice.org

Acknowledgments

This guide was translated from French. The original guide was written by Aurélie Schröder of the University of Geneva (Switzerland), who graciously read over the initial English translation and also provided code examples.

Modifications and updates

Version	Date	Description of Change
1	02/19/04	Translation of French version 1.0 issued
	03/01/04	In the section “About this guide,” referred readers to the SDK Developer's Guide.

Introduction

This guide explains how to modify the context menu of an application that uses OpenOffice.org's word processor, Writer. We use Java in our code examples, but the code could conceivably be translated into another programming language, such as C++, Delphi, or Visual Basic.

For the purpose of this guide, we'll use two examples from the OpenOffice.org SDK, SimpleBean and ContextMenuInterceptor. SimpleBean lets us learn how to use OfficeBean beans, and ContextMenuInterceptor lets us modify the OpenOffice.org context menu. This guide explains how and why to modify each of these two examples.

The research for this guide was done as part of a study that examined the feasibility of controlling Writer's interface. The purpose of the study was to provide users with the simplest interface possible so that they would be guided as they worked.

Documentation on SimpleBean can be found in Chapter 16 of the SDK Developer's Guide. Documentation on ContextMenuInterceptor is unfortunately not available, because the relevant chapter of the Developer's Guide was lost and is in the process of being re-built.

This guide doesn't discuss the OpenOffice.org API. For details about the functions used here, see the SDK.

All of the code examples in this guide are taken from the SDK Developer's Guide. Code examples for OfficeBean are taken from Chapter 16, “Office Bean,” and code examples for ContextMenuInterceptor are taken from Chapter 6, “Office Development.”

Using SimpleBean

SimpleBean connects to OpenOffice.org through its parent class, the BasicOfficeBean bean. (A bean is a reusable component.) SimpleBean can also be used to toggle the menu bar, although we don't exploit that functionality here.

Connecting to OpenOffice using BasicOfficeBean

It's essential to understand how a bean connects to OpenOffice.org.

We'll use OpenOffice.org as a server. (To learn more about using OpenOffice.org in server mode, see Chapter 2, "First Steps," of the SDK Developer's Guide.)

First, we create a service factory to establish communication between the bean and OpenOffice.org:

```
// Obtain the global MultiServiceFactory and store it in mServiceFactory
 XmultiComponentFactory compfactory;
 compfactory = mConnection.getComponentContext().getServiceManager();
 mServiceFactory = (XmultiServiceFactory)UnoRuntime.queryInterface(XMultiServiceFactory.class, compfactory);
```

The code shown above is taken from Chapter 16, "Office Bean," of the OpenOffice.org SDK Developer's Guide. This code is a partial example; it's only given here to highlight the important statements involved in establishing the connection.

The method UnoRuntime.queryInterface()[1] facilitates the creation of the object, in this case the service factory.

Next, we initialize a frame in which to load the OpenOffice.org object. This frame is created using the service factory that we initialized above.

```
XWindow window = (XWindow) UnoRuntime.queryInterface( XWindow.class, mWindow.getUNOWindowPeer());
object  = mServiceFactory.createInstance( "com.sun.star.frame.Task");
if ( object == null )
  object  = mServiceFactory.createInstance( "com.sun.star.frame.Frame");
mFrame = (XFrame)UnoRuntime.queryInterface( XFrame.class, object );
mFrame.initialize(window);
```

Note: *Keep in mind that all frames have a controller. This will be useful later.*

After connecting, we load the OpenOffice.org object that we want to use (Writer, Draw, Impress, or Calc) by means of its URL. For Writer, the URL is private:factory/swriter. We can also specify the URL of an existing document.

1 See Section 2.5, "Working with Objects," in Chapter 2, "First Steps," of the SDK Developer's Guide.

The following example shows how to load an OpenOffice.org object:

```
// Load document

XComponent xComponent = xLoader.loadComponentFromURL( url, mFrame.getName(), FrameSearchFlag.ALL, aArgs
);
```

url specifies the URL of the object that we want to load.

mFrame specifies the frame in which the object will be loaded.

aArgs gives the object's property values. We'll keep the default values given by BasicOfficeBean.

For more details about the use of loadComponentFromURL(), see the SDK Developer's Guide or the API documentation on com.sun.star.Frame.XComponentLoader.

If SimpleBean is a child class of BasicOfficeBean and simpleBean is an instance of SimpleBean, we can instantiate Writer as follows:

```
simpleBean.load("private:factory/swriter");
```

The code shown above will open a new document based on the default template.

Connecting to ContextMenuInterceptor

The ContextMenuInterceptor class was created in the SDK to add functionality to the base context menu. ContextMenuInterceptor lets us intercept and modify the context menu before it's displayed to the user. We won't go into detail here on how ContextMenuInterceptor works, only how it connects to BasicOfficeBean.

(Note that OpenOffice.org must be launched with an option that keeps a port open to allow it to be driven by Java. For more details, see Chapter 2, “First Steps,” of the SDK Developer's Guide.)

Let's look at how the connection is made in this example:

```
private OfficeConnect(String sHost, String sPort){

 try

 {

  String sConnectString = "uno:socket,host=" + sHost + ",port=" + sPort +  ";urp;StarOffice.ServiceManager";

  com.sun.star.lang.XMultiServiceFactory xLocalServiceManager =
                 com.sun.star.comp.helper.Bootstrap.createSimpleServiceManager();

  com.sun.star.bridge.XUnoUrlResolver xURLResolver = (com.sun.star.bridge.XUnoUrlResolver)
UnoRuntime.queryInterface(com.sun.star.bridge.XUnoUrlResolver.class,
xLocalServiceManager.createInstance
("com.sun.star.bridge.UnoUrlResolver"));

  mxServiceManager = (com.sun.star.lang.XMultiServiceFactory) UnoRuntime.queryInterface(

             com.sun.star.lang.XMultiServiceFactory.class,

             xURLResolver.resolve(sConnectString));

 }
```

```
  catch (com.sun.star.uno.RuntimeException exUNO){
    System.out.println("connection failed" + exUNO);
  }
  catch (com.sun.star.uno.Exception exRun){
    System.out.println("connection failed" + exRun);
  }
  catch (java.lang.Exception exJava) {
    System.out.println("connection failed" + exJava);
  }
}
```

The connection process for ContextMenuInterceptor is similar to the connection process for OfficeBean. We create a service factory, and then we use the service factory to establish a connection with ContextMenuInterceptor's frame.

```
OfficeConnect aConnect = OfficeConnect.createConnection("localhost", "8100");

com.sun.star.frame.XDesktop xDesktop = (com.sun.star.frame.XDesktop)
                aConnect.createRemoteInstance(com.sun.star.frame.XDesktop.class,
                                "com.sun.star.frame.Desktop");

com.sun.star.frame.XFrame xFrame = xDesktop.getCurrentFrame();
```

createConnection() calls OfficeConnect, which was defined earlier.

createRemoteInstance uses the service factory to establish the connection:

```
public Object createRemoteInstance(Class aType, String sServiceSpecifier)
{
  Object aResult = null;
  try
  {
    aResult = UnoRuntime.queryInterface(aType,mxServiceManager.createInstance(sServiceSpecifier));
  }
  catch (com.sun.star.uno.Exception ex) {
    System.out.println("Couldn't create Service of type " + sServiceSpecifier + ": " + ex);
    System.exit(0);
  }
  return aResult;
}
```

The frames created in BasicOfficeBean and ContextMenuInterceptor are therefore of the same type.

Intercepting the context menu

To intercept the context menu so that we can modify it, all we have to do is allocate BasicOfficeBean's frame to ContextMenuInterceptor.

First, we modify BasicOfficeBean's load() method as follows:

```
/**
 * Loads a document referenced by a URL.
 *
 * @param url The document's URL string.
 * @exception java.io.IOException if the document loading process has
 *                      failed.
 */
public synchronized void load( String url ) throws java.io.IOException
{
 [...] //no change to the initial code

// Connect to ContextMenuInterceptor to control the right click
  ContextMenuInterceptor mContext = new ContextMenuInterceptor(mFrame.getController() ) ;


}
```

Next, we modify ContextMenuInterceptor's constructor in such a way that it can receive BasicOfficeBean's frame controller as an argument. We use BasicOfficeBean's frame controller because it is the last part of the frame that ContextMenuInterceptor uses to connect. This way, we avoid having to build a new desktop, frame, and so on.

For our modified constructor to work, we first have to add an empty constructor. Then we can add our modified constructor. See the following example:

```
public ContextMenuInterceptor()
{}

public ContextMenuInterceptor(XController aXController )
{
  try {
    com.sun.star.frame.XController xController = aXController ;
    if ( xController != null ) {
      com.sun.star.ui.XContextMenuInterception xContextMenuInterception =
              ( com.sun.star.ui.XContextMenuInterception ) UnoRuntime.
              queryInterface(com.sun.star.ui.XContextMenuInterception.class,
                    xController ) ;
      if ( xContextMenuInterception != null ) {
        ContextMenuInterceptor aContextMenuInterceptor = new ContextMenuInterceptor() ;
        com.sun.star.ui.XContextMenuInterceptor xContextMenuInterceptor =
                        ( com.sun.star.ui.XContextMenuInterceptor )
                        UnoRuntime.queryInterface(
                        com.sun.star.ui.XContextMenuInterceptor.class,
                        aContextMenuInterceptor ) ;
        xContextMenuInterception.registerContextMenuInterceptor(xContextMenuInterceptor ) ;
      }
    }
  }
  catch ( java.lang.Throwable ex ) {
    // catch java exceptions ? do something useful
    System.out.println( " Sample caught exception! " + ex ) ;
    System.exit( 1 ) ;
  }
}
```

Modifying the context menu

We can change the appearance of the context menu by, for example, adding or deleting menu options. To do this, we modify the notifyContextMenuExecute interface. The following example modifies the notifyContextMenuExecute interface to add a Help option to the menu. The code is explained in the comments.

```
/**
  *
  * @param aEvent
  * @return
  * @throws java.lang.RuntimeException
  */
 public ContextMenuInterceptorAction notifyContextMenuExecute(
        com.sun.star.ui.ContextMenuExecuteEvent aEvent ) throws RuntimeException {

    try {

        // Obtain the context menu's container and make a request to the service factory
        // to create sub menus, menu entries, and separator lines
        com.sun.star.container.XIndexContainer xContextMenu = aEvent.ActionTriggerContainer;
        com.sun.star.lang.XMultiServiceFactory xMenuElementFactory =
            (com.sun.star.lang.XMultiServiceFactory)UnoRuntime.queryInterface(
            com.sun.star.lang.XMultiServiceFactory.class, xContextMenu );
        if ( xMenuElementFactory != null ) {
            // Create a root menu entry and a sub menu
            com.sun.star.beans.XPropertySet xRootMenuEntry =
                (XPropertySet)UnoRuntime.queryInterface(
                    com.sun.star.beans.XPropertySet.class,
                    xMenuElementFactory.createInstance( "com.sun.star.ui.ActionTrigger" ));

            // Create a separator line for our new help sub menu
            com.sun.star.beans.XPropertySet xSeparator =
                (com.sun.star.beans.XPropertySet)UnoRuntime.queryInterface(
                    com.sun.star.beans.XPropertySet.class,
                    xMenuElementFactory.createInstance( "com.sun.star.ui.ActionTriggerSeparator" ));

            Short aSeparatorType = new Short( ActionTriggerSeparatorType.LINE );
```

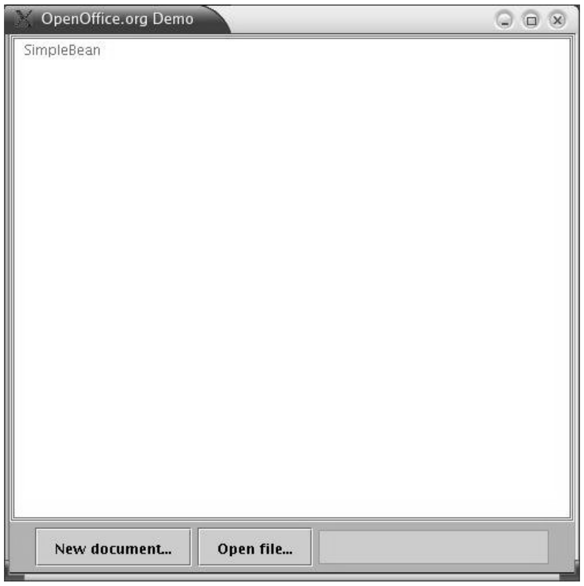
```
xSeparator.setPropertyValue( "SeparatorType", (Object)aSeparatorType );

// Query the sub menu so that the index container can access it
com.sun.star.container.XIndexContainer xSubMenuContainer =
    (com.sun.star.container.XIndexContainer)UnoRuntime.queryInterface(
        com.sun.star.container.XIndexContainer.class,
            xMenuElementFactory.createInstance(
                "com.sun.star.ui.ActionTriggerContainer" ));

// Initialize a Help entry for the root context menu
xRootMenuEntry.setPropertyValue( "Text", new String( "Help" ));
xRootMenuEntry.setPropertyValue( "CommandURL", new String( "slot:5410" ));
xRootMenuEntry.setPropertyValue( "HelpURL", new String( "5410" ));
xRootMenuEntry.setPropertyValue( "SubContainer", (Object)xSubMenuContainer );

// Create menu entries for the new Help sub menu

// Initialize the Content menu entry
XPropertySet xMenuEntry = (XPropertySet)UnoRuntime.queryInterface(
                              XPropertySet.class, xMenuElementFactory.createInstance(
                                  "com.sun.star.ui.ActionTrigger" ));

xMenuEntry.setPropertyValue( "Text", new String( "Content" ));
xMenuEntry.setPropertyValue( "CommandURL", new String( "slot:5401" ));
xMenuEntry.setPropertyValue( "HelpURL", new String( "5401" ));

// Insert the Content menu entry into the Help submenu
xSubMenuContainer.insertByIndex( 0, (Object)xMenuEntry );

// Initialize the Help Agent menu entry
xMenuEntry = (com.sun.star.beans.XPropertySet)UnoRuntime.queryInterface(
                 com.sun.star.beans.XPropertySet.class,
                     xMenuElementFactory.createInstance(
                         "com.sun.star.ui.ActionTrigger" ));
xMenuEntry.setPropertyValue( "Text", new String( "Help Agent" ));
xMenuEntry.setPropertyValue( "CommandURL", new String( "slot:5962" ));
```

```
            xMenuEntry.setPropertyValue( "HelpURL", new String( "5962" ));

            // Insert the Help Agent menu entry into the Help sub menu
            xSubMenuContainer.insertByIndex( 1, (Object)xMenuEntry );

            // Initialize the Tips menu entry
            xMenuEntry = (com.sun.star.beans.XPropertySet)UnoRuntime.queryInterface(
                            com.sun.star.beans.XPropertySet.class,
                               xMenuElementFactory.createInstance(
                                  "com.sun.star.ui.ActionTrigger" ));
            xMenuEntry.setPropertyValue( "Text", new String( "Tips" ));
            xMenuEntry.setPropertyValue( "CommandURL", new String( "slot:5404" ));
            xMenuEntry.setPropertyValue( "HelpURL", new String( "5404" ));

            // Insert the Tips menu entry into the Help sub menu
            xSubMenuContainer.insertByIndex( 2, (Object)xMenuEntry );

            // Add a separator line to the context menu
            xContextMenu.insertByIndex( 0, (Object)xSeparator );

            // Add the new Help sub menu to the context menu
            xContextMenu.insertByIndex( 0, (Object)xRootMenuEntry );

            // The controller should execute the modified context menu and stop notifying other interceptors
            return com.sun.star.ui.ContextMenuInterceptorAction.EXECUTE_MODIFIED;
        }
    }
    catch ( com.sun.star.beans.UnknownPropertyException ex ) {
        // do something useful
        // we used an unknown property
    }
    catch ( com.sun.star.lang.IndexOutOfBoundsException ex ) {
        // do something useful
        // we used an invalid index to access a container
    }
    catch ( com.sun.star.uno.Exception ex ) {
        // Something unexpected happened !
    }
    catch ( java.lang.Throwable ex ) {
        // catch Throwable exceptions, do something useful
    }

    return com.sun.star.ui.ContextMenuInterceptorAction.IGNORED;
}
```

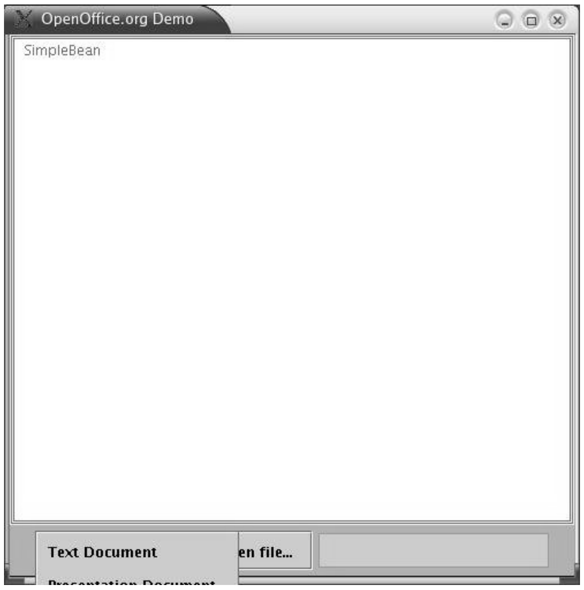
Implementing the context menu

Because our goal is to provide a simplified interface, we'll implement the modified context menu using the example applet SimpleViewer.

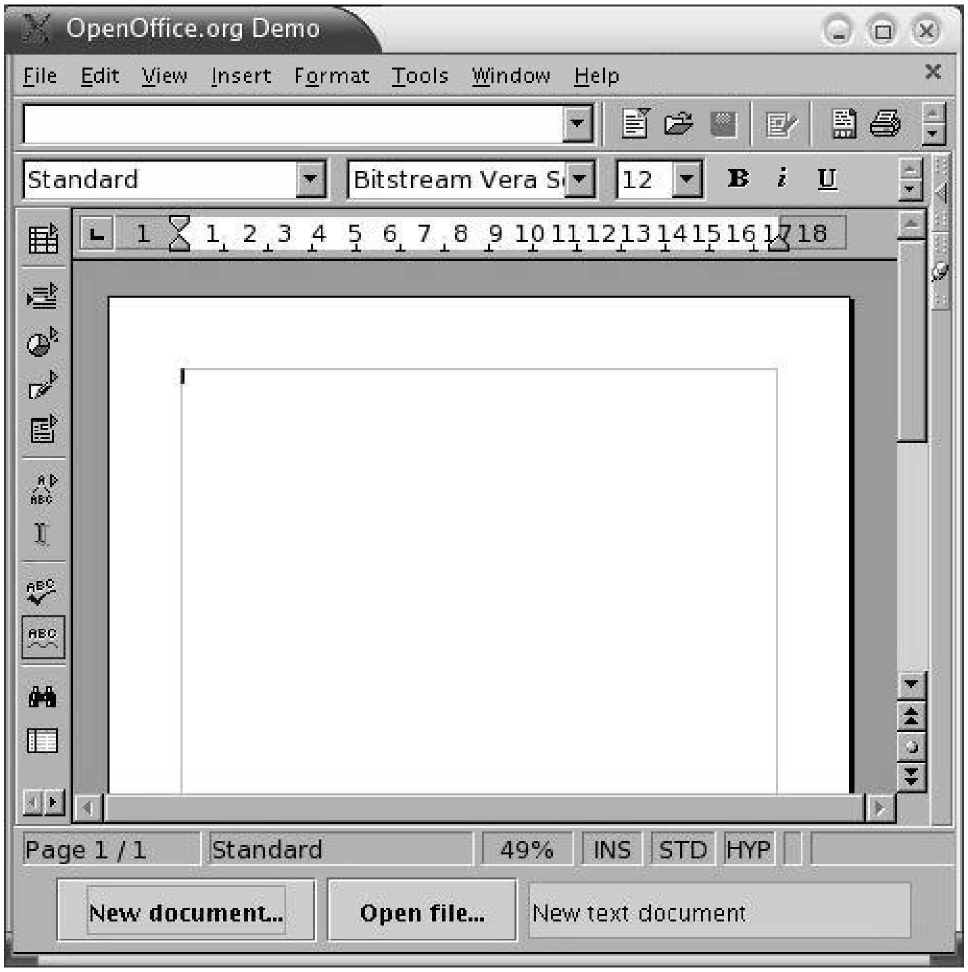
At execution time, SimpleViewer displays a dialog box that allows us to choose whether to open a new document or an existing one:



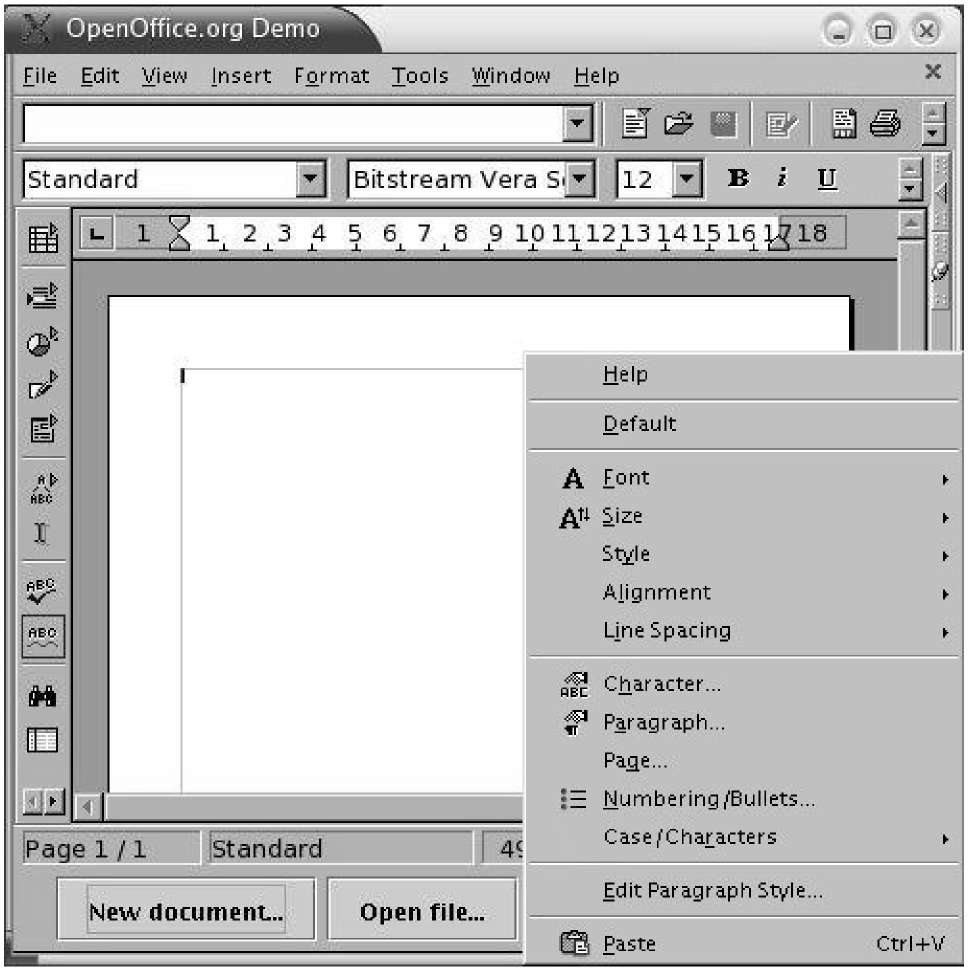
For the purpose of this example, we'll choose a new document. SimpleViewer prompts us to choose the type of document that we want to open:



We'll choose a text document. SimpleViewer opens a new text document:



When we right click in the document, the modified context menu appears. In the picture below, notice that the Help option has been added to the base context menu.



Conclusion

In this guide, we learned how to use a service factory and port to connect a client to a server, and we learned how to use frames to modify OpenOffice's graphical interface.

Service factories allow us to connect a client, in our case a bean, to a server. Once we've launched OpenOffice.org in server mode, we just have to specify which port is open for the connection.

We also learned how to modify OpenOffice's interface. First, we define a service factory, which we then use to define a frame into which we load the desired OpenOffice object. This frame enables us to access object properties that can be manipulated by other modules. For example, we can modify menus by adding, deleting, and modifying menu options.

Once the connection is made, OpenOffice becomes very malleable and easy to configure. We can connect other components by using the properties of the initial frame. This essentially allows us to control the users' environment, either to help them by giving them additional shortcuts or to guide them by giving them a tool that's very simple to use.

Note the one limitation. If, within Writer, we open a new page the normal way, the properties added within SimpleViewer (the class that instantiates the OfficeBean bean) will be lost.