



ComfyJ Tutorial

Version 2.7
September 17, 2009

Table of Contents

Introduction	i
1. Setting Up	1
1.1. Environment Configuration	1
1.2. Creating Application Class	1
2. Generating COM Integration Code	3
3. Writing Planner Class	4
3.1. Outlining Planner Functionality	4
3.2. COM Support Initialization and Shutdown.....	4
3.3. Performing Login	5
3.4. Accessing Tasks and Calendar Folders.....	5
3.5. Creating a New Task.....	6
3.6. Creating a New Appointment.....	6
4. Conclusion	8

Introduction

Welcome to the ComfyJ Tutorial. This document will guide you through the process of creating a small application that interacts with COM objects. We will go through the things you need to know to develop Java applications that communicate with COM objects.

The first task in this tutorial is project environment configuration. After that we will generate wrapper classes for selected COM objects using the CodeGenerator application. Then, we will come to the actual programming: using the generated API, we will create a Task and an Appointment in Microsoft Outlook.

The tutorial assumes that you are familiar with JNIWrapper and COM integration features described in the ComfyJ *Programmer's Guide*. You can find it in your ComfyJ installation and on the product site. If you have some difficulties in doing the tutorial tasks, please refer to the ComfyJ *Programmer's Guide* and JNIWrapper *Programmer's Guide* for help.

The complete sample code is available in the full distribution package of ComfyJ and can be obtained on the product site.

Note: This tutorial assumes that ComfyJ is installed on C:\ComfyJ.

Chapter 1. Setting Up

1.1. Environment Configuration

First off, we need to set up the working environment. We will call the program Planner to illustrate its main tasks. Create a folder with this name for our project. In this tutorial, we assume that this folder is created under `C:\ComfyJ\samples`.

Under the project folder, we will need the following subfolders:

- `bin` - This folder is for native code files.
- `lib` - This folder will contain JAR files.
- `src` - This folder will store our Java code.

Now we need to copy necessary files from your ComfyJ installation to the above folders.

Copy the native code DLL (`jniwrap.dll`) from the installation `bin` folder and the run-time license files (`jniwrap.lic` and `comfyj.lic`) from run-time `key.zip` subdirectory to `Planner\bin`. Also copy `comfyj.jar`, `jniwrap.jar` and `winpack.jar` to the `lib` folder.

At this point you should have the following structure:

```
Planner\  
  bin\  
    jniwrap.dll  
    jniwrap.lic  
    comfyj.lic  
  lib\  
    comfyj.jar  
    jniwrap.jar  
    winpack.jar  
  src\  

```

The above folder layout is not mandatory, and you can arrange your project as you wish. If you do, please update the references to the paths in your environment accordingly.

Note that the `comfyj.jar`, `jniwrap.jar` and `winpack.jar` libraries must be present in `CLASSPATH` of your development environment. Also, make sure that `jniwrap.dll` can be accessible to your JVM. If you are not sure how to do that, please refer to the *JNIWrapper Programmer's Guide*.

1.2. Creating Application Class

Now let's create a class for the application and run it before moving to the actual coding. As we already decided, we will call it `Planner`.

Put the following code into `Planner.java`:

```
package com.jniwrapper.samples.planner;  
  
public class Planner  
{  
    public static void main(String[] args)  
    {  
        System.out.println("The Planner is running");  
        System.exit(0);  
    }  
}
```

Compile and run the class to see if your configuration is correct. If you can see the output string in the console, jump to the next tutorial step.

Chapter 2. Generating COM Integration Code

To communicate with Microsoft Outlook, we will generate Java code for Microsoft Office Outlook Type Library MSOUTL.OLB. We can do this using CodeGenerator supplied with ComfyJ. This application requires GUID, version, folder for generated code, and destination package. If you're not familiar with this application, please refer to the ComfyJ *Programmer's Guide*.

The GUID of the Microsoft Office Outlook Type Library is:

```
{00062FFF-0000-0000-C000-000000000046}
```

The version number we need is 9.2.

The command line for code generation should look like this:

```
C:\ComfyJ\bin>CodegenForComfyJ.bat guid {00062FFF-0000-0000-C000-000000000046} 9.2  
C:/ComfyJ/samples/Planner/src com.jniwrapper.win32.samples.planner
```

Note that this command should be started from the `bin` folder of ComfyJ installation.

The value of the destination folder parameter in the example above means that generated sources will be placed in the `C:\ComfyJ\samples\Planner\src` folder. Replace this path with the one you want to generate stubs to.

After code generation is complete, you should get two subpackages called `office` and `outlook`. Their names are automatically taken by CodeGenerator from the COM library.

Chapter 3. Writing Planner Class

3.1. Outlining Planner Functionality

Let's define the structure of our application. First, as the ComfyJ *Programmer's Guide* suggests, we need to initialize COM support for our application. Then, we should log in to MAPI profile to be able to work with Outlook folders. After that, we get the Tasks folder and create a sample task in this folder. Next, we do the same for the Calendar folder to create an appointment.

The general structure of our application is shown below.

```
package com.jniwrapper.samples.planner;

import com.jniwrapper.UInt32;
import com.jniwrapper.win32.ole.OleFunctions;
import com.jniwrapper.win32.com.types.*;

public class Planner
{
    public static void main(String args[])
    {
        Planner p = new Planner();
        int result = p.execute();
        System.exit(result);
    }

    private int execute()
    {
        init();
        try
        {
            login();

            MAPIFolder tasksFolder = getTasksFolder();
            createTask(tasksFolder);

            MAPIFolder calendarFolder = getCalendarFolder();
            createAppointment(calendarFolder);
            return 0;
        }
        catch(ComException e)
        {
            System.out.println("Unable to access Outlook profile.");
            e.printStackTrace();
            return -1;
        }
        finally
        {
            shutdown();
        }
    }
}
```

At this stage, all the methods that our `main()` function calls do nothing and return nulls.

You may also notice that we don't handle errors for login and each object is created separately. We do this again for brevity.

Now that we have the application structure defined, let's put necessary code for initialization and clean-up.

3.2. COM Support Initialization and Shutdown

As the ComfyJ *Programmer's Guide* instructs, each Java thread that works with COM must initialize a COM library at the beginning and uninitialize at the end of its lifecycle.

Since in our sample application we use only one thread, we will use the `OleFunctions.oleInitialize()` and `OleFunctions.oleUninitialize()` methods for initialization and uninitialization, respectively. Since we are not performing other initialization and clean-up tasks for our application, the `init()` and `shutdown()` methods will look like this:

```
private void init()
{
    OleFunctions.oleInitialize();
}

private void shutdown()
{
    OleFunctions.oleUninitialize();
}
```

3.3. Performing Login

Login to Outlook is performed by invoking the `logon()` method of the `_NameSpace` class. The method takes the following parameters:

- *profile* - MAPI profile to log in.
- *password* - a string representing the password.
- *showDialog* - a boolean value that indicates whether the dialog should be shown.
- *newSession* - a boolean value that indicates whether a new session should be started.

The namespace we need is "MAPI" and we will keep the reference to it as a static field of our class that will be used in other methods:

```
public class Planner
{
    private _NameSpace _mapiNS = null;
    ...
}
```

The implementation of the `login()` method is shown below:

```
private void login()
{
    _Application application = Application.create(ClsCtx.LOCAL_SERVER);
    _mapiNS = application.getNamespace(new BStr("MAPI"));
    _mapiNS.logon(new Variant("Outlook"),
        new Variant(""),
        new Variant(false),
        new Variant(false));
}
```

We use "Outlook" profile name with an empty password. Change these parameters to the values specific to your environment.

3.4. Accessing Tasks and Calendar Folders

Any Outlook folder can be accessed with the `MAPIFolder` class. The necessary folder should be obtained from the `_NameSpace` class instance.

All available folders in Outlook are represented by the `OIDefaultFolders` class as appropriate constants. Methods for accessing required folders are listed below:

```
private MAPIFolder getTasksFolder() throws ComException
{
    MAPIFolder result = _mapiNS.getDefaultFolder(
        new OIDefaultFolders(OIDefaultFolders.olFolderTasks));
    return result;
}
private MAPIFolder getCalendarFolder() throws ComException
{
    MAPIFolder result = _mapiNS.getDefaultFolder(
        new OIDefaultFolders(OIDefaultFolders.olFolderCalendar));
    return result;
}
```

3.5. Creating a New Task

Now that we have access to the necessary Outlook folder, we can add new items to it.

For the Tasks folder, this item is a task. Tasks for Outlook are represented by the `_TaskItem` class. It should be instantiated like shown in the sample below. Note that we are using the generated implementation class `_TaskItemImpl` to create an instance of the `_TaskItem` interface.

```
public void createTask(MAPIFolder tasksFolder)
{
    final Variant itemType = new Variant(OIItemType.olTaskItem);
    final IDispatch task = tasksFolder.getItems().add(itemType);
    _TaskItem newTask = TaskItem.queryInterface(task);

    //set up some properties for the new task
    newTask.setSubject(new BStr("This is a test"));
    newTask.setBody(new BStr("How are you doing today?"));

    //assign the current time for it
    Date current = new Date(new java.util.Date());
    newTask.setStartDate(current);
    newTask.setDueDate(current);

    //and finally save it
    newTask.save();
}
```

3.6. Creating a New Appointment

Adding an appointment is similar to adding a task to the Tasks folder:

```
public void createAppointment(MAPIFolder calendarFolder)
{
    final Variant itemType = new Variant(OIItemType.olAppointmentItem);
    final IDispatch appointment = calendarFolder.getItems().add(itemType);

    _AppointmentItem newAppt = AppointmentItem.queryInterface(appointment);

    //sep up some properties
```

```
newAppt.setSubject(new BStr("Another Test"));
newAppt.setBody(new BStr("Hello! Again!"));

Calendar calendar = Calendar.getInstance();
Date current = new Date(calendar.getTime());
calendar.add(Calendar.MINUTE, 30);
Date tmEnd = new Date(calendar.getTime());
newAppt.setStart(current);
newAppt.setEnd(tmEnd);

newAppt.setAllDayEvent(new VariantBool(false));
newAppt.setLocation(new BStr("LPB"));

//and finally save it
newAppt.save();
}
```

After inserting the above code into the Planner class, compile and run it. You should have two new items in your Outlook.

Chapter 4. Conclusion

This is the end of the tutorial. Here you learned how to create a simple application that creates a task and an appointment in Microsoft Outlook profile. Although the application is rather simple, it shows the basics for building COM integration applications based on ComfyJ and can serve as a template for such applications.