*Reports of the Department of Mathematical Information Technology*
*Series C. Software and Computational Engineering*
***No. C. 2/2006***

# Java-COM integration with JACOB using XML wrappers

Miika Nurminen

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
FI–40014 University of Jyväskylä
FINLAND
fax +358 14 260 2731
http://www.mit.jyu.fi/

ISBN 951-39-2633-8
ISSN 1456-4378

# Java-COM integration with JACOB using XML wrappers

Miika Nurminen*

**Abstract**

Many Windows-based legacy applications can be programmatically accessed using COM interfaces. However, calling COM components from Java is not straightforward. This report compares four open source Java-COM integration packages. A technique for typesafe Java-COM integration is presented. The technique is based on typesafe COM interface wrappers using jcom, java2com and JACOB libraries. Examples with Microsoft Office applications are presented. XML wrapper and code generator can be bundled with future JACOB releases as an alternative to Jacobgen wrapper generator.

## 1 Introduction

There is an increasing need to integrate enterprise systems with legacy applications. Java is probably the most popular contemporary general-purpose programming language, having platform independency as one essential advantage. However, this implies trouble when platform-independent software should interoperate with platform-specific code.

COM (Component Object Model) interfaces provide a standard way to interact with many native Windows applications, most notably the applications in Microsoft Office suite. COM is supposed to be language-independent, to the extent that even non-object-oriented languages like C or old versions of Visual Basic can interact with COM components.

Unfortunately, it is not straightforward to call COM objects from Java, because it requires interacting with native code running outside the JVM (Java Virtual Machine) using JNI (Java Native Interface). It is possible to call COM objects directly using Microsoft's own JVM in Windows [2]. Because of the lawsuit initiated by Sun Microsystems in 1997, Microsoft has effectively dropped its Java support. Microsoft's JVM is based on Java 1.1, lacking significant features and libraries introduced in later Java versions. Therefore alternative techniques for Java-COM integration are needed.

*Department of Mathematical Information Technology, University of Jyväskylä, PO Box 35 (Agora), FI-40014 University of Jyväskylä, Finland, minurmin@jyu.fi

Web services [10] are a contemporary technique for loose integration of heterogeneous systems. However, direct in-process integration is in many ways superior – and in some cases the only viable alternative to Java-COM integration. Some benefits of direct integration are:

- **Performance**. Web services are slow compared to dynamic linking. For some legacy applications web services might not even be feasible at all – or a separate "ws-facade" should be implemented as a native Windows application, adding another layer of overhead to the system.

- **Scripting**. Automation (also known as Dispatch) interfaces allow scripting Windows-based applications at runtime. For example, Microsoft Office suite is used in in many business systems via automation, especially with Visual Basic. Java-COM integration exposes these interfaces to Java applications as well (scripting Office is not a necessarily a good architectural decision but there is nevertheless a need to interoperate with existing systems).

- **Legacy applications**. Some highly specialized Windows-based legacy applications that have not been actively maintained provide only COM interfaces to work with (this may apply even to more recent applications, as demonstrated by Crosby with iTunes [4]). Since contemporary Microsoft Office versions have provided increasing support for purely file-based XML data exchange, the integration problem with Office is not critical compared to other applications. Even *with* XML enabled, it may be easier to create new or modify existing Office documents with COM vs. generating Office XML from scratch.

- **ActiveX controls**. In the user interface side, Java-COM bridge makes it possible to interface with ActiveX (formerly known as OLE) controls and listening to COM events triggered by user actions. This can be indirectly applied also to other kinds of UI components, such as VCL-based (Visual Component Library) controls used in Borland Delphi, since they can be published as ActiveX controls [3].

- **.NET integration**. With the advent of .NET, Java-COM integration provides a compact (at least compared to web services) way to call .NET assemblies, since .NET components can as exposed as COM components [1] [9].

This report focuses on calling Dispatch-based COM components from Java code, but not vice versa (with the exception of events, since some Java-COM bridges allow COM event handlers to be callbacks in Java code). In addition to Dispatch-based COM components there are also Vtable-based components (ie. non-scriptable components that do not implement `IDispatch` interface [9]). It is assumed that the COM components reside on the same machine as the Java code. The more elaborate problem of embedding ActiveX controls physically into Java applications [13] is outside of the scope of this report.

The practical need for integration described in this report arose from a Java-based process development project. One of the project's goals was to read and manipulate a group of existing business process models modelled with Microsoft Excel and Visio and potentially with other process modelling software. One-way transformation based on parsing Office 2003 XML was already implemented, but in order to modify the models it seemed more feasible to manipulate the Office object model directly using COM interfaces.

There are several commercial Java-COM integration tools available, but the report is focused on open source approaches. Four integration packages were compared based on three simple test programs that integrate with Microsoft Office and Visio. The review carried out suggests that contemporary open source Java-COM integration components do not support both wrappers (ie. pregenerated typesafe stubs based on a COM type library) and work with all the test cases without the need to modify the wrappers manually or to write complex code to call the components. An adoption of JACOB [2] library based on the code written by Steven Lewis [7] is presented that alleviates these deficiencies.

This report is organized as follows: section 2 introduces and reviews four integration packages for Java-COM integration. Section 3 describes the technique and required source code modifications to JACOB in detail. The test cases are presented in section 4. Section 5 concludes the report.

# 2 Alternatives to Java-COM integration

The goal of wrapper-based Java-COM integration is to call Dispatch COM components with the same ease and convenience as from Visual Basic. As an example, JACOB allows calling components using `IDispatch` interface, but this requires a lot of browsing through documentation and extensive typecasts whenever a value setter or getter is used. If one manages to get the datatypes and method names right, it works – but is inherently unsafe and tedious for any but the simplest projects. A few lines from JACOB's sample files that fetch some data from Excel illustrate the problem:

```
ActiveXComponent xl = new ActiveXComponent("Excel.Application");
Dispatch.put(xl,"Visible", new Variant(true));
Dispatch workbooks = xl.getProperty("Workbooks").toDispatch();
Dispatch workbook = Dispatch.get(workbooks,"Add").toDispatch();
Dispatch sheet = Dispatch.get(workbook,"ActiveSheet").toDispatch();
Dispatch a1 = Dispatch.invoke(sheet,"Range", Dispatch.Get,
                                    new Object[] {"A1"},
                                    new int[1]).toDispatch();
Dispatch a2 = Dispatch.invoke(sheet,"Range", Dispatch.Get,
                                    new Object[] {"A2"},
                                    new int[1]).toDispatch();
Dispatch.put(a1, "Value","123");
Dispatch.put(a2, "Value","=A1*2");
System.out.println("a2 from excel:"+Dispatch.get(a2,"Value"));
```

A more elegant approach is to use wrapper classes generated from a COM type library. Type libraries (such as `excel.exe`) describe all methods, datatypes, and events related to a COM server. Type libraries can be browsed by many Windows-based development tools such as Microsoft Visual Studio [14] or Borland Delphi [3]. Even some Java-COM tools described in this report provide a type library browser. Using the technique described in in section 3 it is possible to script Excel (and basically any Dispatch-based COM-component) with more simplified notation:

```
Application xl = new Application(); // Inherited from Excel wrapper
System.out.println(xl.getVersion());
xl.setVisible(true);
Workbooks workbooks = xl.getWorkbooks();
Workbook workbook = workbooks.Add();
Worksheet sheet = new Worksheet(workbook.getActiveSheet());
Range a1 = sheet.getRange("A1");
Range a2 = sheet.getRange("A2");
a1.setValue("123");
a2.setFormula("=A1*2");
System.out.println("a2 from excel:"+a2.getValue());
```

Even with generated wrappers all datatypes cannot be inferred. Getter `workbook.getActiveSheet()` returns a Dispatch pointer that is subsequently handed to a Worksheet constructor – effectively a typecast.

Four open source Java-COM integration tools – JACOB, Jawin, jSegue and com4j are compared in this section. There seems to be a clear need to integrate Java code with legacy COM code, since there are a number of projects under development and hundreds of posts in the projects' discussion forums. Unfortunately, none of the integration tools worked ”out of the box” without problems related to wrapper generation, datatype handling or simply the ease of use. The documentation was also somewhat scarce. Wrapper classes generated by different projects were similar in general, but had some variations in detail. Thus, after choosing one integration platform it is not straightforward to move to another. It is crucial to choose the integration tool with respect to requirements relatively early in the implementation.

JACOB 1.10.1, Jawin 2.0 alpha, jSegue 2.0.0.394 and com4j 2006-03-22 were evaluated based on three test cases that integrate with Microsoft Visio (document manipulation), Excel (chart generation) and Word (event handling). The test cases are described in more detail in section 4. Sun's JDK 1.5.0_04-b05 running on Windows XP was used as test environment. NetBeans 5 [11] was used as development environment. The results are summarized in table 1.

Actual support for different datatypes (especially arrays and dates) embedded in COM Variants varies between the projects, but they were not tested systematically.

## 2.1 JACOB – Java-COm Bridge

JACOB [2] is a Java-COM bridge that allows calling COM automation components from Java. It uses JNI to make native calls into the COM and Win32 libraries. JACOB was originally created by Dan Adler and is based on Microsoft's Java SDK. JACOB

| | *JACOB* | *Jawin* | *jSegue* | *com4j* |
|---|---|---|---|---|
| Dispatch interfaces | Yes | Yes | Verbose | Yes |
| Vtable interfaces | No | Yes | Yes | In progress |
| COM events | Partial | No | Yes | In progress |
| Wrapper generator | Yes | Partial | Needs C++ | In progress |
| COM browser | No | Yes | No | Yes |

Table 1: Comparison of Java-COM integration tools.

supports Dispatch-based COM interfaces as well as COM events. Vtable-based interfaces are not supported. JACOB project has been in active development since 1999, latest version 1.10.1 was released in April 2006. Because of its age, JACOB can be considered a stable and mature package.

JACOB is accompanied by a wrapper generator called Jacobgen. It is still on relatively early development stage and did not work adequately with the test cases – for example, Jacobgen did not parse a type library from Excel. Fortunately, there is another wrapper generator component java2com created in 1998 by Steven Lewis [7] that produces XML wrappers and Java stubs. Some minor modifications (see section 3) are required to both contemporary JACOB distribution and old java2com package to get the tests run correctly.

## 2.2 jSegue

jSegue is a toolset for making Java bindings to native code [5]. The toolset includes a wrapper generator tlb2java that generates Java and JNI code to call COM Automation servers. Of the integration packages tested, jSegue is the only tool that works also as *COM-Java* bridge, allowing Java classes to be used as in-process COM servers. The project has been in active development at least since 2004 and earlier as a commercial package from Moebius Solutions. Latest version 2.0.0.394 was released in March 2006.

jSegue is a mature and robust package, supporting both Dispatch and Vtable-based COM interfaces as well as COM events. All the test cases run with jSegue, so it seems to be a feasible integration solution. The only disadvantage with jSegue is that it is somewhat awkward to use. Its wrapper generator produces both Java wrappers *and* C++ stubs that must be separately compiled to make full use of the library. This works if one has Visual Studio or equivalent installed, but is regardless an extra step from purely Java point of view. A more serious inconvenience with jSegue is its convention of calling Dispatch interfaces (even with wrapper-generated code). For example, consider the Excel example presented in section 2 and focus on the line that creates a new Worksheet:

```
Worksheet sheet = new Worksheet(workbook.getActiveSheet());
```

In jSegue, the equivalent code must be represented as follows:

```
IDispatch sh = workbook.getActiveSheet();
_Worksheet[] out_sheet = { null };
_Worksheet sheet;
sh.QueryInterface(out_sheet);
sheet=out_sheet[0];
```

It works, but the required plumbing code makes the approach laborious (in this case, it seems almost simpler to use a purely Dispatch-based approach with typecasts without wrappers). Regarding the goal of making COM as easy to use from Java as from Visual Basic it is a bit too verbose and C++-like. However, when implementing or enhancing a system in immediate production use, jSegue would be recommended because of its maturity.

## 2.3 Jawin

Java/Win32 integration project (Jawin) is a free, open source architecture for interoperation between Java and components exposed through COM or through Win32 Dynamic Link Libraries (DLLs) [9]. The project was initiated by Stuart Halloway and Justin Gehtland in 2003. Currently, the development seems to have slowed down: the latest version 2.0 alpha1 was released in March 2005.

Jawin has many promising features: it supports both Dispatch- and Vtable-based COM components and includes an easy to use COM type library browser as well as XML-based wrapper generator (although incompatible with wrappers generated with java2com). However, COM events are not supported. Jawin works in principle, but in practice the test cases required numerous minor wrapper code changes both in initial XML files and Java code to make it even compile, not to mention making it actually work. Someone with extra time and energy (and deeper knowledge of COM) could most likely fix the bugs associated in wrapper generation with moderate effort. Eventually the wrapper files were generated, but the Variant type handling turned out to be obscure to say the least – at least compared to conventions used in JACOB or even jSegue. Only the Word and Visio-related test cases run with Jawin.

Jawin has potential, especially in the user interface side. It would be ideal tool for a beginner developer, especially because of the COM browser. Unfortunately, at its current state it cannot be recommended, because the generated code must be modified manually and because of the issues with datatype handling.

## 2.4 com4j

Com4j is a new project to develop a Java library that allows Java applications to seamlessly interoperate with COM and to produce a Java tool that imports a COM type library and generates the Java definitions from it [6]. Com4j takes advantages of Java 1.5 features to improve usability (ie. producing significantly cleaner wrapper code compared to all other packages discussed here). Com4j is developed by

Kohsuke Kawaguchi. The project was started in 2004, latest version was signed 22.03.2006.

Com4j is ambitious and promising project. The idea of using Java 1.5 metadata to describe wrappers is elegant and when matured, the package would provide all the essential tools for COM interoperability: interfaces for both Dispatch- and Vtable-based components, support for events as well as a wrapper generator. Unfortunately, the project was in alpha stage while it was tested. The generation of wrappers was not successful, so it is definitely too early to put it in production use. Because of the frequent updates, some of the problems mentioned here may have been fixed already. Com4j might be an optimal solution in the future.

# 3 Enhancing JACOB with java2com

The initial tests indicated that JACOB itself is stable and mature enough to be used for Java-COM integration, but the wrapper generator component should be replaced with a more robust one. An alternative component can be found from Steven Lewis' java2com package [7] that uses wrapper generator based on Yoshinori Watanabe's jcom [12] and a modified JACOB DLL library. Java2com was released in 1998.

When the tests were run on old java2com package, wrapper generation worked fine (although the `enum` keyword was used in generated class names – the package was deployed years before the release of Java 1.5), but there was some problems with some of the test cases (for example, changing font style in Microsoft Visio did not work – perhaps something related to COM enum handling?). However, since the equivalent, purely Dispatch-based code worked in current JACOB 1.10.1 distribution is was concluded that the invoke error was caused by the *old* JACOB DLL, not the java2com component itself.

The logical next step was to integrate the old java2com code with current JACOB distribution. Unfortunately, the current version of JACOB DLL is not directly compatible with Lewis' modifications: there are changes both in C++ and Java code, along with some renamed methods and enhanced functionality. In order to take advantage of the bug fixes of the last 8 years, JACOB package had to be manually modified to make it again compatible with java2com (and Java 1.5). These modifications are described in detail in the following subsections.

## 3.1 Wrapper generation

Java2com includes subprojects Xmlgen and Codegen, for generating XML descriptors from COM type libraries and subsequently generating Java stubs from XML descriptors. In order to generate the descriptors, the type library files must be located. There are numerous tools for this, including Visual Studio or Borland Delphi. Type libraries can be referred with absolute file name (eg. `C:\Program Files\Microsoft Office\OFFICE11\EXCEL.EXE`), a class identifier (128-bit number also known as

GUID) or a human readable program identifier (eg. `Excel.Application`). Type libraries can include other type libraries (for example, Excel is dependent on *Office object library*, *Standard OLE* and *Visual Basic library*), but Xmlgen locates and consolidates all required data without the need to request it explicitly.

Xmlgen was incorporated into the new integration package unaltered. Codegen package was slightly modified to make the generated code compatible with Java 1.5. `enum` literals were renamed and some additional exception handling was added. Overall, the following classes were modified.

```
com.lordjoe.utilities.COMEnumBuilder
com.lordjoe.utilities.COMSurrogateBuilder
com.lordjoe.utilities.IStackTraceable
com.lordjoe.utilities.LException
com.lordjoe.utilities.ReflectiveIterator
com.lordjoe.utilities.StackTrace
com.lordjoe.utilities.TypelibDispatchBuilder
com.lordjoe.utilities.TypelibEnumBuilder
com.lordjoe.utilities.Util
com.lordjoe.utilities.WrapperException
```

Xmlgen and Codegen were combined to the same binary distribution (see `javacom\bin` package), along with XML parser Xerces [15] that is required by Codegen.

## 3.2 Modifications to JACOB

This is the most complicated part of the integration. On the test cases carried out the contemporary JACOB package with java2com-generated wrappers did not work (or even compile). The modifications made by Lewis in 1998 had to be examined and matched with current JACOB source code (both Java and C++), preferably without introducing any new bugs or breaking anything (see also *Modifications to JACOB code for Generated Code* [7] by Steven Lewis that explains his original modifications).

The following classes can be directly replaced by the classes in java2com package. They are either new or there has not been any significant progress in the recent years:

```
com.jacob.activex.COMconstants
com.jacob.activex.JDispatch
com.jacob.activex.Jvariant
com.jacob.activex.ThreadingModelEnum
com.jacob.com.ComThreadingTypes
com.jacob.com.COMUtils
com.jacob.com.DispatchConstants
com.jacob.com.IUnknown
com.jacob.com.StructuredException
com.jacob.com.VTTypeEnum
com.jacob.com.WDispatch
```

The following classes should be kept in JACOB package as is. They are either the same as in java2com or contain fixes compared to code in java2com.

```
com.jacob.com.ComException
com.jacob.com.WrongThreadException
com.jacob.com.ComFailException
com.jacob.com.SafeArray
```

`Dispatch` and `Variant` classes require more extensive changes. A Dispatch constructor with threading model must be copied to JACOB, but the native call `createInstance` should be renamed to `createInstanceNative` (a new stub `createInstanceNative` with threading model argument, as well as `doSetDefaultThreadingModel` stub must also be added). Java2com has a more elaborate constructor `Dispatch(int)` (which requires also adding a `setDispatch` native stub and `getDispatch` method) compared to one in JACOB may be used. The method `obj2variant` differs slightly, but the code in JACOB is newer, so it is kept as is.

| *Method name* | *Notes* |
|---|---|
| `Dispatch(String, ThreadingModelEnum)` | Copy from java2com, rename call `createInstance` → `createInstanceNative` |
| `Dispatch(int)` | Copy and replace from java2com |
| `getDispatch()` | Copy from java2com |
| `createInstanceNative(String, ThreadingModelEnum)` | Add native stub |
| `doSetDefaultThreadingModel(int)` | Add native stub |
| `setDispatch()` | Add native stub |

Table 2: Modifications to `Dispatch` class in JACOB.

`Variant` in java2com has a few methods that have either been renamed or removed and can be ignored. These include `save`, `load`, and `getObjectRef`. However, `Variant(String)` and `Variant(SafeArray, boolean)` constructors, `getIID`, `getObject`, `buildSafeArray` and `buildVariantArray`, as well as `EMPTY_ARRAY` constant should be added from java2com. `ForceObjectVal` should be copied, but modified only to call `toJavaObject` method in JACOB. Finally, constructor `Variant(Object, boolean)` should be augmented with `WDispatch` and `Array` handling. The changes required to `Dispatch` and `Variant` classes are summarized in tables 2 and 3.

The last step needed is altering the Dispatch C++ class in JACOB DLL directly. The following methods have been renamed in newer JACOB:

`putobject` → `putDispatchObject`
`todispatch` → `toDispatchObject`
`createInstance` → `createInstanceNative`

An implementation of `createInstance` with `threadingModel` parameter should be copied and renamed from java2com. Another `createInstanceNative` method must be augmented with `CoInitialize` call in the beginning and handling for

| *Method name* | *Notes* |
|---|---|
| `Variant(String)` | Copy from java2com |
| `Variant(SafeArray, boolean)` | Copy from java2com |
| `Variant(Object, boolean)` | Add `WDispatch` and `Array` handling |
| `getIID()` | Copy from java2com |
| `getObject()` | Copy from java2com |
| `buildSafeArray(Object)` | Copy from java2com |
| `buildVariantArray(Object[])` | Copy from java2com |
| `EMPTY_ARRAY` | Copy from java2com |
| `forceObjectVal()` | Copy from java2com, modify method to call `toJavaObject` |

Table 3: Modifications to `Variant` class in JACOB.

progids starting with '{' in the end of the method. Also, implementations for `setDispatch` and `doSetDefaultThreadingModel` should be added.

The convention of automatically initializing COM threads and specifying threading model explicitly has some effect on existing JACOB sample code. Java2com uses multithreaded model as default, but current JACOB samples assume single-threaded apartments (eg. a single-threaded apartment thread is created in the beginning of the Excel example). The threading model should be explicitly specified in component constructor and matched with the running thread – alternative approach would be to change the default threading model in JACOB. For example, an Excel component using single-threaded apartment would be created as follows:

```
ActiveXComponent xl = new ActiveXComponent("Excel.Application",
  com.jacob.activeX.ThreadingModelEnum.ApartmentThreaded_ENUM);
```

However, if wrapper classes descended from `WDispatch` are to be used, JACOB creates automatically a multithreaded apartment and therefore one cannot make a call like `ComThread.InitSTA()` when wrappers are used (see example `ExcelWrapperTest` and compare it to `ExcelDispatchTest` in `com.jacob.samples.office` package). In both cases the thread should be explicitly released with `ComThread.Release()` to prevent potential memory leaks.

## 3.3 Limitations

Apart the wrapper code, the java2com-enhanced JACOB package has the same general limitations as JACOB. There is no support for Vtable-based COM components. Also, there is no inherent support for graphical ActiveX controls embedded in Java UIs, although the Visio example provided in `com.jacob.samples.visio` demonstrates some basic functionality. JACOB supports both single- and multithreaded apartments, but the thread-related code adapted from java2com is not thoroughly

tested. Liong [8] reviews some intricacies related to COM apartments. It should be also noted that the JACOB default threading model was altered since version 1.7 [2].

Another essential limitation is event handling. JACOB allows events using `DispatchEvents` class, but the type information related to them is lost: all event handlers must have a `Variant[]` type signature. Event handlers are described in type libraries, so Codegen generates wrapper classes for them, but they cannot be directly used. There was no event handling mechanism in java2com and the wrappers are not compatible with JACOB's DispatchEvents. Indirectly, the wrapper classes (eg. `com.microsoft.excel.gen._WorkbookEvents`) could be used to look up the correct method names and excepted types for event handlers, but a custom handler class handled to `DispatchEvents` with `Variant[]` arguments must still be used. A potential workaround would be to change `InvocationProxy` class to call event handlers with correct signatures using reflection, but the initial tests were not successful. As with Jawin, this could probably fixed with moderate effort, since the basic event handling mechanism is already implemented.

# 4 Using modified JACOB distribution and test cases

This section describes the test cases in detail and provides guidelines for building and deploying the modified JACOB distribution. Test cases work also as examples. All the source code is publicly available in WWW at `http://www.mit.jyu.fi/minurmin/javacom/`. The test cases were constructed to evaluate the basic functionality of Java-COM integration tools with following requirements:

- Compilation of the generated wrapper code (this may sound trivial but was actually an issue with almost all integration tools tested).
- Instantiation of COM objects from Java code (eg. starting Excel).
- Calling COM methods from Java (eg. creating a chart with Excel).
- Conversion between Java datatypes and COM variants (these were not tested exhaustively, but include basic datatypes, Dispatch pointers, and COM enumerations).

JACOB 1.10.1 and java2com packages were repackaged to new source and binary distributions. Source distribution is divided to wrapper generators Xmlgen and Codegen, modified JACOB code jacob_1.10.1_mod, sample code com_samples containing standard JACOB samples, the test cases and wrapper classes for *Microsoft Word 2003*, *Excel 2003* and *Visio 2003*. The projects are organized using NetBeans 5 -generated Ant build scripts. Binary distribution contains both JACOB and jcom DLLs, as well as jar packages of all the projects, along with examples. The binary distribution can be used to generate both XML descriptors and wrapper classes and for running the examples.

For rebuilding JACOB or the examples with NetBeans 5, Codegen should be opened first, then JACOB and finally com_samples projects, because of the dependencies. Codegen requires XML parser Xerces contained in the `lib` directory. Xmlgen is separated from other projects and should not need to be recompiled. Heap size may have to be increased, because the wrappers constitute over 1000 classes, requiring extensive memory. If this is the case, NetBeans should be started with the following-like command line parameters that instruct JVM to increase heap size to 400MB, for example:

```
nb.exe -J-Xmx400M
```

Class `jp.jcomgenerator.XMLGenerator` in `XMLGenerator.jar` generates XML descriptors (`jcom.dll` must be in path). `com.lordjoe.lib.xml.TypeLib` in `codegen.jar` generates Java wrappers (`xerces.jar` must be in classpath as well). Using modified JACOB package requires `jacob.jar` and `codegen.jar` in classpath and `jacob.dll` in path.

## 4.1 Visio integration

Visio example starts Visio, creates boxes, changes text style, waits for 5 seconds and exits the application. Along the example, there is some commented code used in the tests with jSegue and Jawin (the Jawin code does not actually work. This was one reason Jawin was abandoned). Java2com-generated COM enums are also demonstrated by changing font size. Changing font style did not work with the original java2com package.

Original JACOB distribution in `com.jacob.samples.visio` package contains a more elaborate example using Visio. This example demonstrates embedding Visio chart in Java `JFrame` as well as event handling, but uses only Dispatch interface without wrappers. Converting it to wrappers would be an interesting and useful exercise.

## 4.2 Creating a chart with Excel

Excel example is based on the original *Talking to COM* example [7]. The application starts Excel, creates a workbook with some cells, creates a chart and exits. The methods to be called have slightly more complex parameters compared to Visio example and it does not run with Jawin either.

There is another, slightly simpler Excel example in `com.jacob.samples.office` package that demonstrates similar functionality with both Dispatch interface + typecasts (`ExcelDispatchTest`) and wrappers (`ExcelWrapperTest`). The `ExcelDispatchTest` is slightly altered to make it work with explicit threading model.

## 4.3 Events with Word

Word example demonstrates event handling. An event handler class is created and registered with Word application object. Then two documents are

created. Java application gets notifications when documents are opened, before closing and on application exit. As discussed in section 3.3, event handlers are restricted to coarse `Variant[]` arguments. However, method names can be checked from `com.microsoft.word._ApplicationEvents2` (simply subclassing `_ApplicationEvents2` with the event handler does not work with JACOB's current event handling, even if one manages to call the handler with correct parameter list. The method calls generated in `_ApplicationEvents2` result in runtime error).

Class `com.jacob.samples.office.WordDocumentProperties` is another example adapted from standard JACOB distribution. It fetches document author metadata from a Word document and exits.

# 5 Conclusion

This report presented a technique and working examples for Java-COM integration using typesafe COM Dispatch interface wrappers. Four open source integration packages were also compared. The approach that was evaluated further was originally presented by Steven Lewis in 1998 and is now adapted to work with current release of JACOB library with detailed explanation of what must be modified in future releases. The source code is publicly available in WWW. XML wrapper and code generator may be bundled with future JACOB releases as an alternative to Jacobgen wrapper generator. XML descriptors could also be used to integrate different Java-COM integration packages, thus making it easier to shift between packages. The type library browser from Jawin could be generalized to be used as a frontend for Xmlgen and Codegen.

The repackaged JACOB worked with examples presented in this report, but further testing is needed, especially with threading. The event handling mechanism in JACOB should be enhanced to be compatible with event classes generated by Codegen. This provides better typesafety and reduces the need to create event handler classes manually. This package hopefully provides a feasible alternative to Java-COM integration using open source software an may provide a robust base for further development.

# References


[1] R. Adhikari, Java & .NET can live together. *Application Development Trends*, December 2001. `http://www.adtmag.com/article.aspx?id=5750`.

[2] D. Adler, The JACOB project: A Java-COM bridge. Techn. rep., 2004. `http://danadler.com/jacob/`.

[3] Borland Delphi 7 for Windows - developer's guide. Techn. rep., Borland, 2002. `http://info.borland.com/techpubs/delphi/Delphi7/DevelopersGuide.pdf`.

[4] N. Crosby, Using the iTunes COM interface with Java and Swing. *workingwith.me.uk*, 2004. http://www.workingwith.me.uk/articles/java/itunes-com-with-java-and-swing.

[5] R. Hastings, jSegue project documentation. Techn. rep., Moebius Solutions, 2004. http://jsegue.sourceforge.net/.

[6] K. Kawaguchi, com4j project documentation. Techn. rep., Sun Microsystems, 2005. https://com4j.dev.java.net/.

[7] S. Lewis, Talking to COM. *SeaJUG*, 1998. http://web.archive.org/web/20050214235807/http://www.lordjoe.com/Java2Com/.

[8] L.B. Liong, Understanding the COM single-threaded apartment - part 1. *The Code Project*, 2005. http://www.codeproject.com/com/CCOMThread.asp.

[9] R.I. Martin and S. Halloway, Jawin project documentation. Techn. rep., 2000. http://jawinproject.sourceforge.net/.

[10] N. Mitra, SOAP version 1.2 part 0: Primer. Tech. Rep. W3C Recommendation 24 June 2003, W3C, 2003. http://www.w3.org/TR/soap12-part0/.

[11] NetBeans web site. Sun Microsystems, 2006. http://www.netbeans.org/.

[12] P. Ombredanne, jcom project web site, nexB, 2003. http://sourceforge.net/projects/jcom/.

[13] D. Srinivas, Embed ActiveX controls inside Java GUI. *The Code Project*, 2000. http://www.codeproject.com/java/javacom.asp.

[14] Visual Studio developer center. Microsoft, 2005. http://msdn.microsoft.com/vstudio/.

[15] Xerces web site. Apache software foundation, 2005. http://xerces.apache.org/.