Fx2Ant Plugin

Version 1.0

User Guide

Table Of Contents

1.0 OVERVIEW

Real-world Flex projects include several Flex modules as well as code written in other languages such as Java. Creation of the build file for a popular Ant tool helps automating the entire build process of your project. While this user manual includes a section describing how to create Ant build files manually, this is a time consuming task, especially for a mid-size or large projects.

Fx2Ant plugin automatically generates an Ant build file for your Flex project. It also creates an HTML wrapper that is required to run swf file in the Web browser. Fx2Ant instantly translates the settings of your existing Flex Projects into Ant scripts, so that you can build your modules, libraries and applications outside of Eclipse and create larger integrated builds. Writing manual Ant scripts is a lengthy process, but with Fx2Ant makes it's just a mouse-click away.

As an extra bonus, Fx2Ant creates build scripts that optimize the size of your Flex project libraries.

2.0 REQUIREMENTS

2.1 Fx2Ant plug-in

Fx2Ant plugin 1.0 requires Java Development Kit (JDK) 1.4.2 or later. The users of Fx2Ant have to have Flex Builder installed as a plugin in Eclipse 3.2 or later.

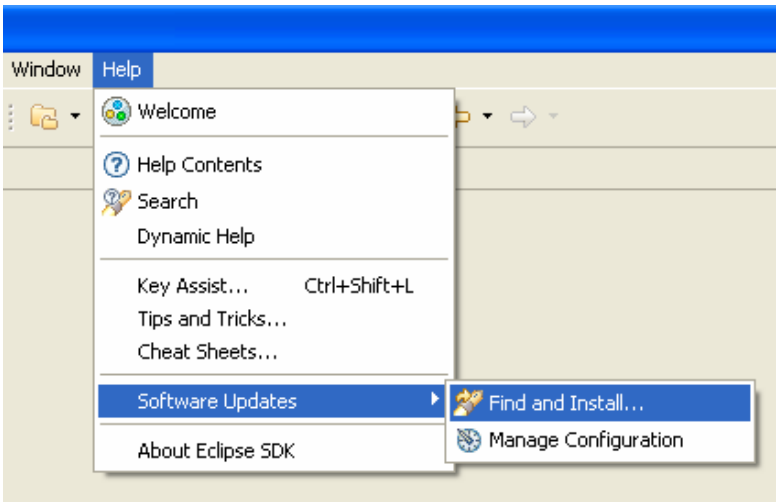
2.2 Apache Ant

Fx2Ant requires Apache Ant 1.6.5 or later, which comes with Eclipse IDE, and it also can be downloaded at http://ant.apache.org/.

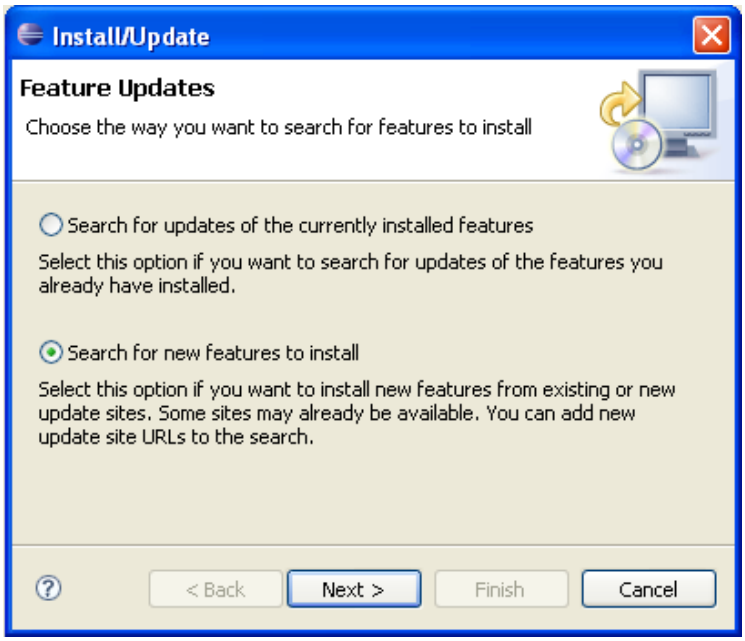
3.0 INSTALLING FX2ANT

3.1 Installing Fx2Ant from a remote update site

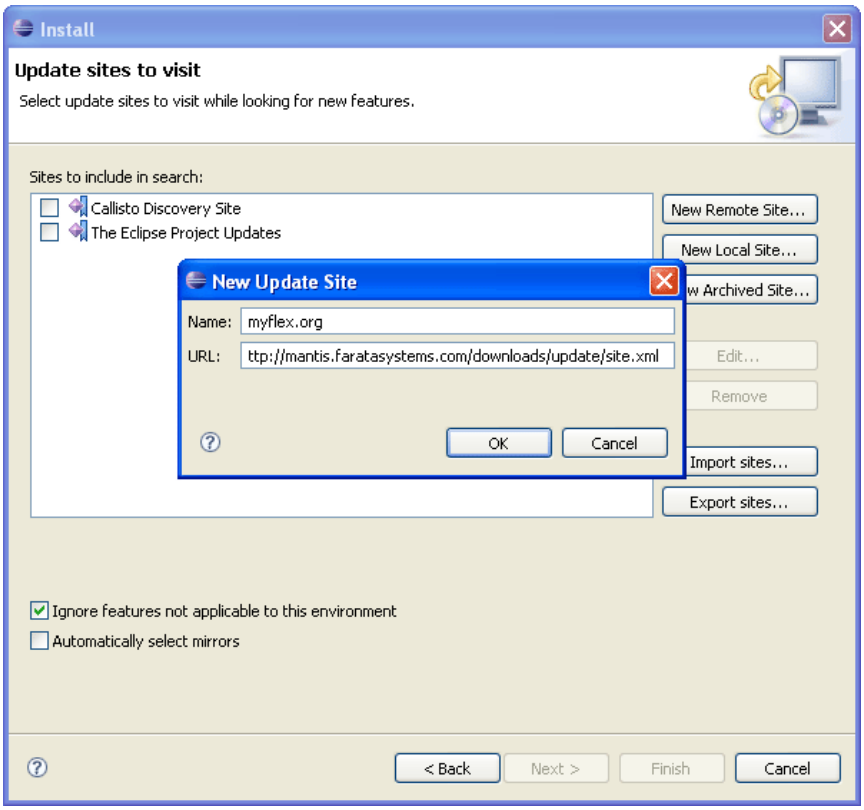
1. Select **Help > Software Updates > Find and Install...** from the main menu.



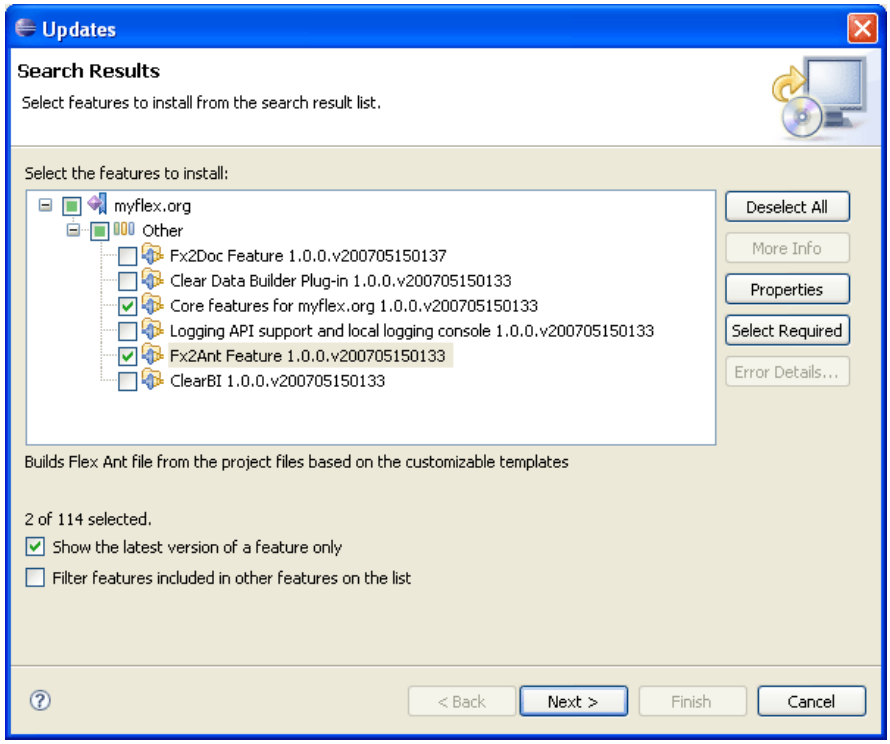
2. Select **Search for new features to install** and press **Next**..



3. You'll see a popup window with a list of available Web sites with Eclipse-related software. Press the button **New Remote Site...** Enter the name of the site (for example, "myflex.org") and the following URL: http://mantis.faratasystems.com/downloads/update/site.xml. Press **OK**, and then **Finish**.

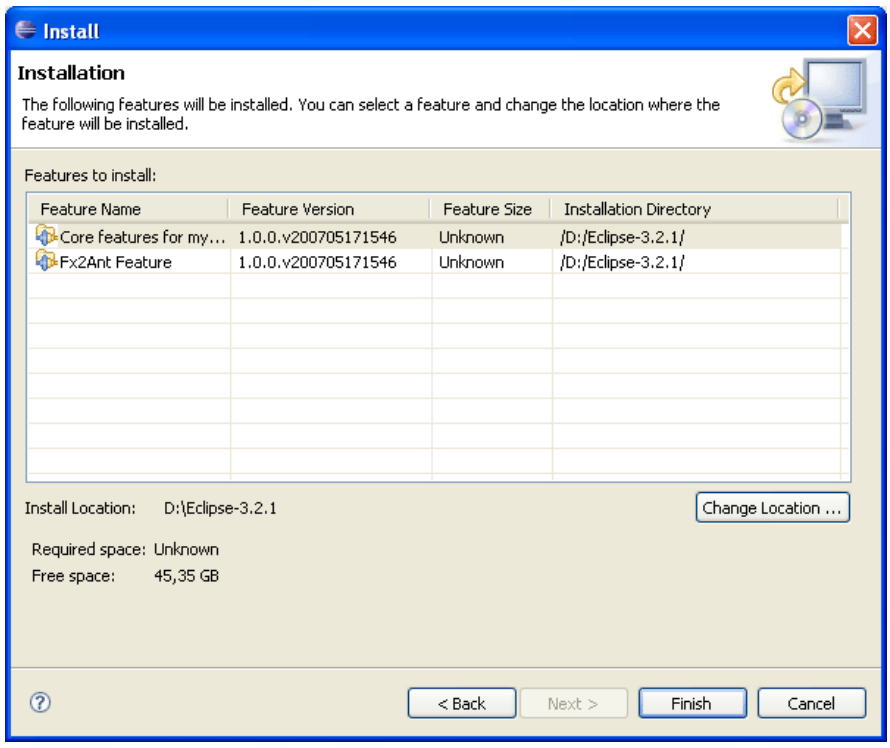


4. Select **Fx2Ant Feature** from the list of features**,** and press the button **Select Required** for adding required plugins. After that press the button **Next**.

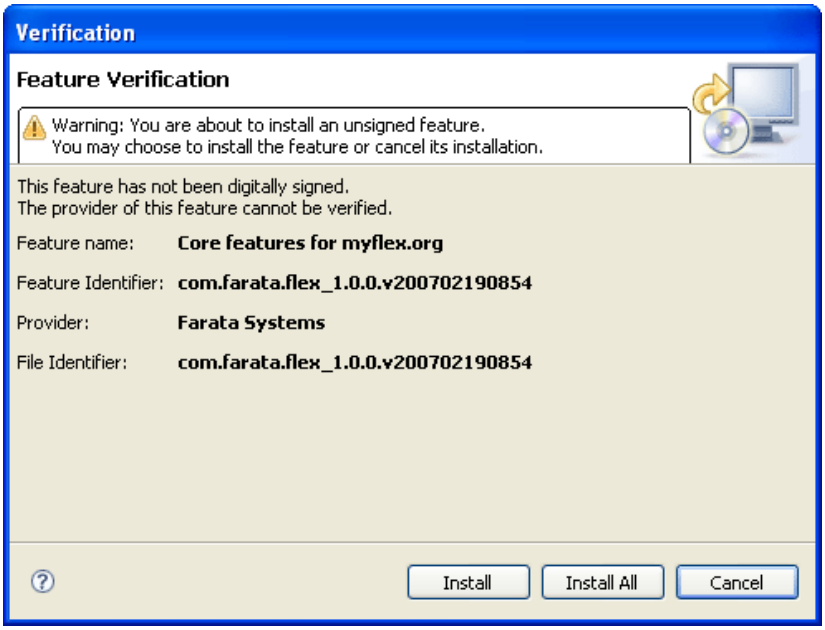


5. Read the License agreement, and if you want to proceed with the installation select **"I accept the terms in the license agreements"** followed by **Next.**

6. Press the **Finish** button on the wizard's confirmation page.



7. Press the button **Install All** on the verification window.



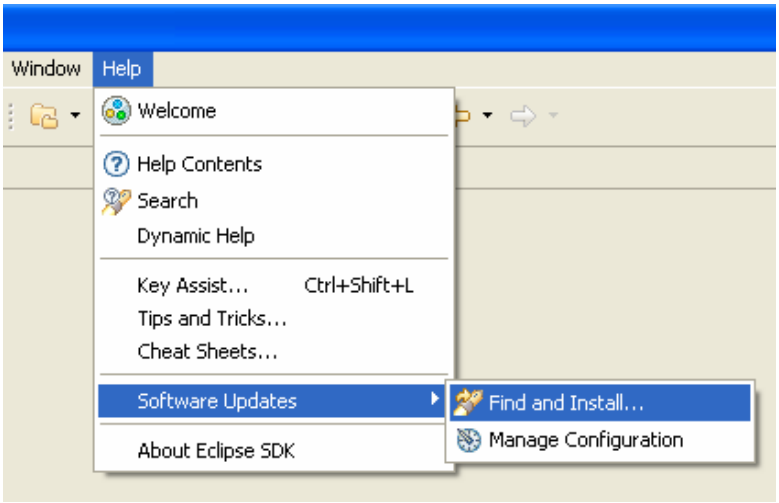
After the installation is complete, reload the workspace - Fx2Ant is ready for use!

3.2 Installing Fx2Ant plugin from the local update site

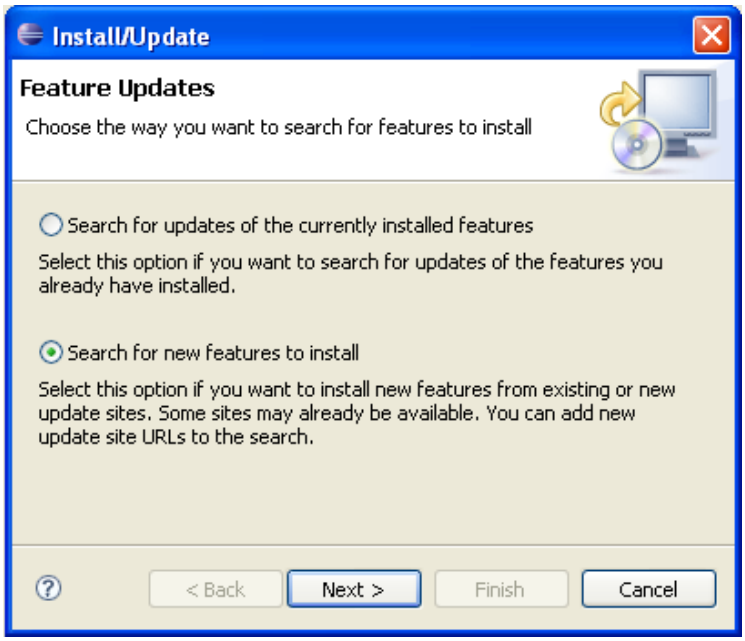
If your computer is protected by a corporate firewall, you might experience difficulties working with remote update site. As a workaround to this issue, create the local update site, and load all required plugins from there.

1. Download the zip-file at http://mantis.faratasystems.com/downloads/local-update/update.zip, which will become your local update site.

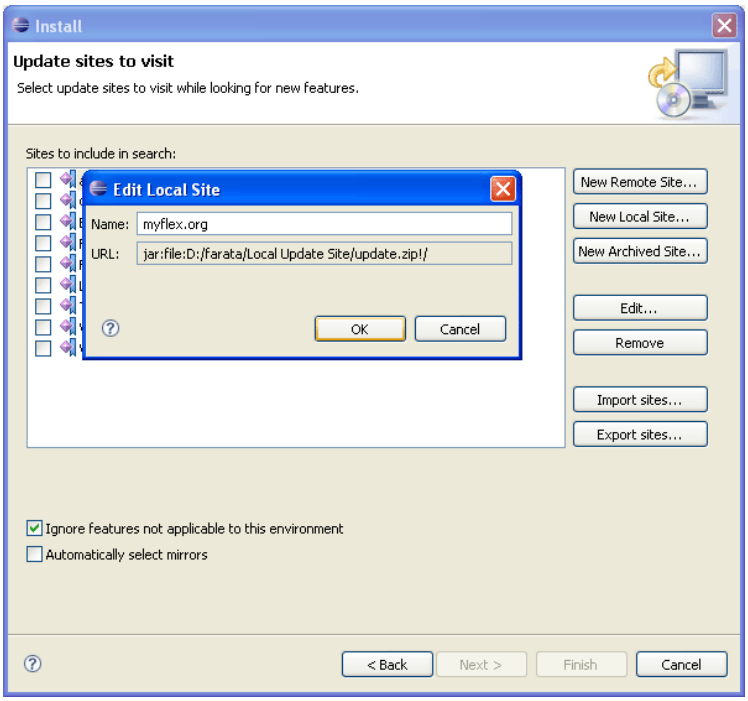
2. Start Eclipse IDE and select the menus **Help > Software Updates > Find and Install**.



3. Select **Search for new features to install** and press **Next**.



4. You'll see a popup window with a list of available Web sites with Eclipse-related software. Press the button **New Archived Site...** select the directory where your update.zip is located. Enter the name of the site (for example, "myflex.org"). Press **OK**, and then **Finish**.



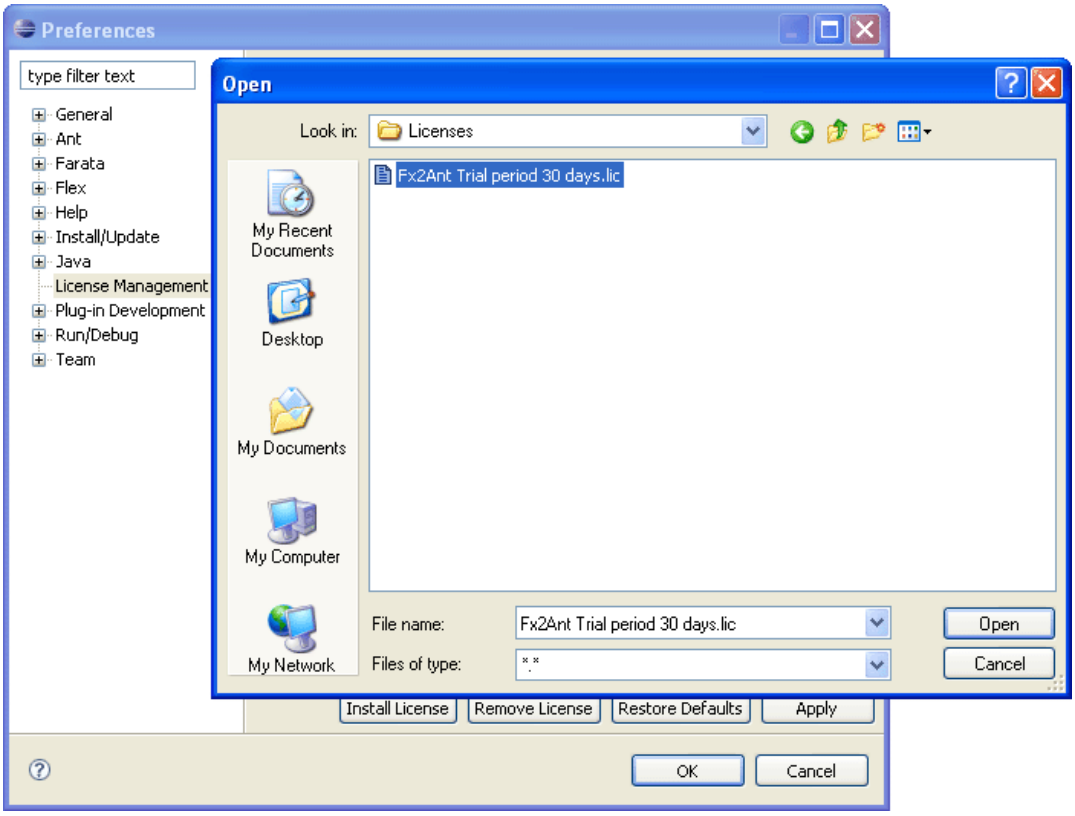
5. Continue installation starting from step 4 in section **3.1**.

3.3 Installing the license for Fx2Ant

1. Download a free trial version or purchase the license at www.myflex.org.

2. Select **Windows > Preferences...** from the main menu.

3. Select **License Management** from the list on the left. On the popup window press **Install License** and select the downloaded license.



4. The selected license becomes your default license. If you did not read the license agreement, please do so by selecting **com.farata.flex2ant** in the **Products under license** section. After that, press the button **OK** and restart Eclipse.

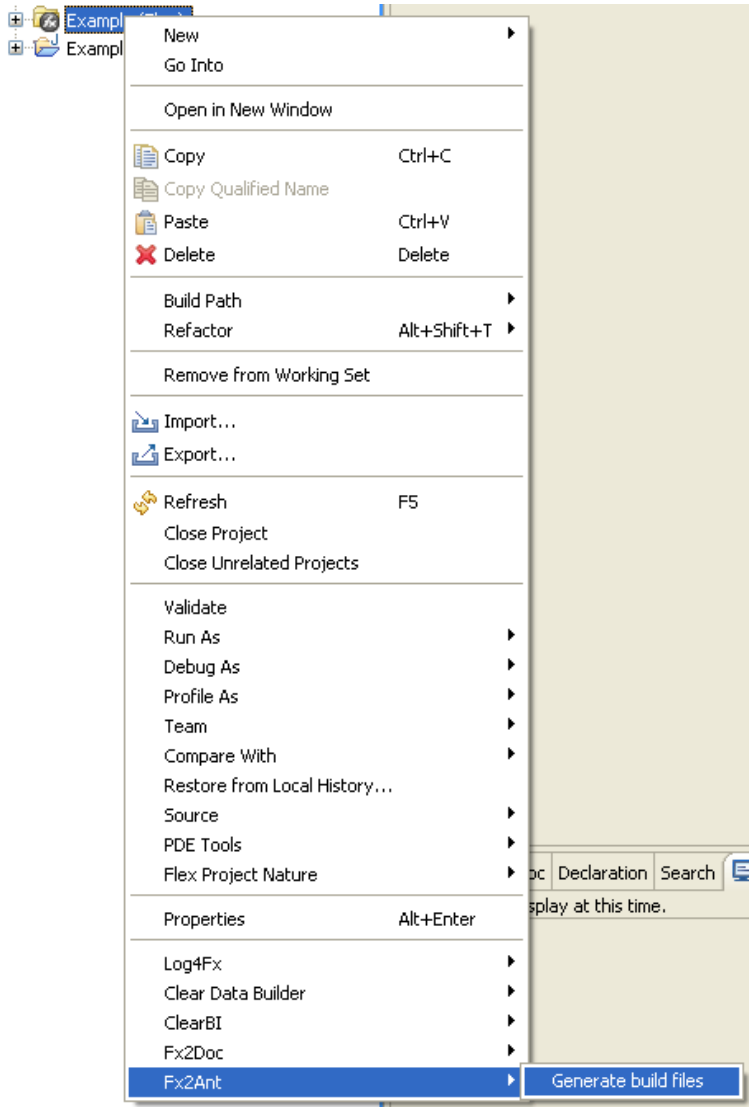
4.0 USING FX2ANT

To build your Flex or Flex Library project you start with auto-generating an Ant build file, and then Ant will execute this build script. The entire process is described below.

4.1 Creating an Ant build file

1. Right-click on your Flex Project or Flex Library Project.

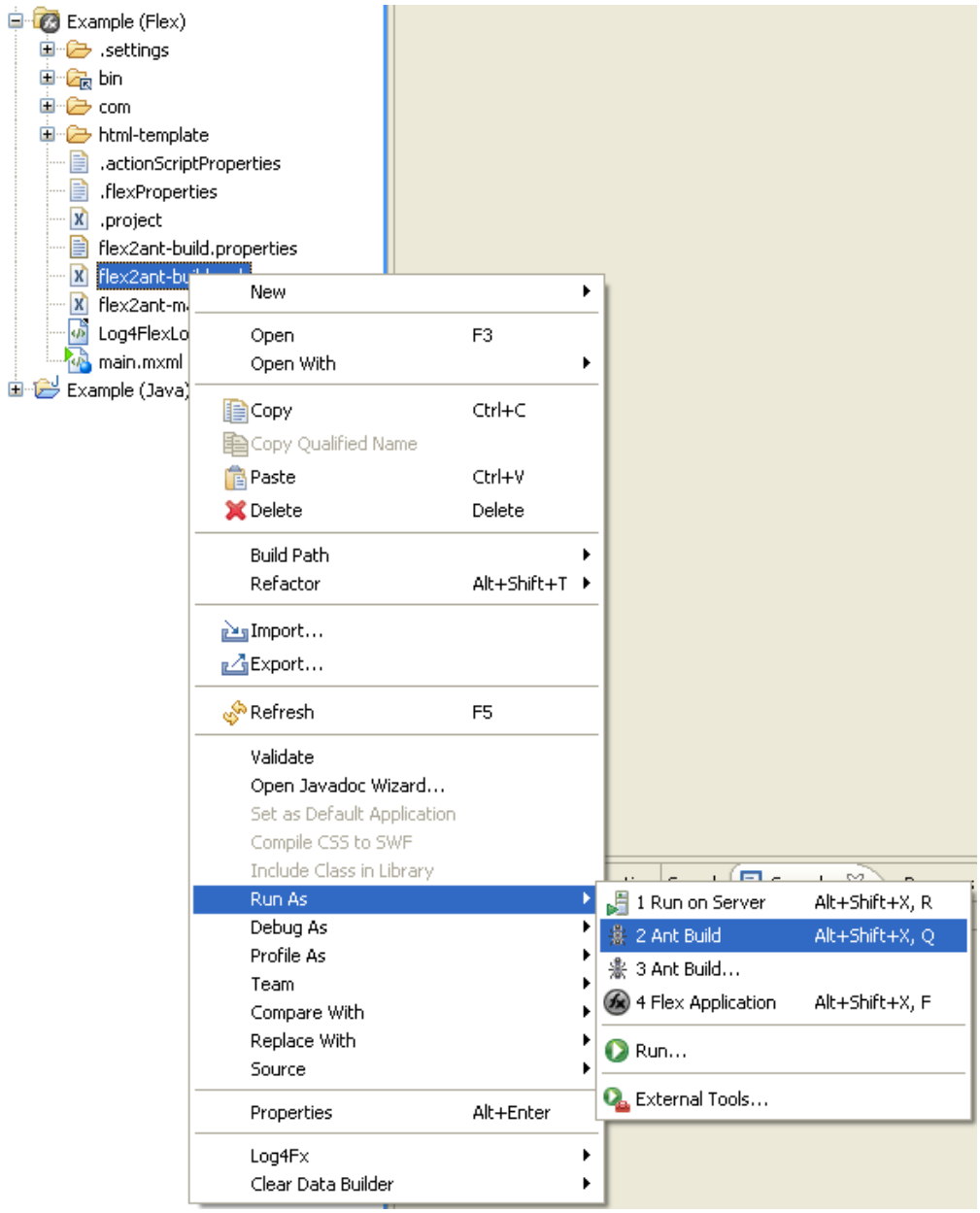
2. Select **Fx2Ant > Generate build files** from the popup menu. This will generate Ant build file according to your project settings.



After the generation is complete, you'll find the file named *flex2ant-build.xml* in your project directory.

4.2 Running the Ant build script

1. Right-click on the generated file name and select **Run As > Ant Build** from the popup-menu.



2. After the execution of the Ant script completes, you'll find the file your_project_name.swf (or your_project_name.swc) in your project directory.

5.0 HELLO WORLD WITH ANT AND FX2ANT

This section contains a two HelloWorld examples. The first one is a mini primer on how use Ant, popular Java-based build tool from Apache.just, and the second one shows how Fx2Ant automates creation of Ant build files for HelloWorld Flex project.

5.1 Building HelloWorld with Ant

Unless your project is as simple as HelloWorld application, compilation and deployment of this Flex application has to be automated. Creating a build script for any decent size application (i.e. hundreds of files in Flex and server-side Java) is a project on its own. Java programmers typically use one of the open source build tools such as Ant or Maven to formally describe and execute all steps required to build and deploy their projects.

Even deployment of such simple application as HelloWorld may consists of several steps, for example

1. Using mxmlc compile HelloWorld.mxml into an swf file.
2. Create an HTML wrapper file(s) from one of the HTML templates.
3. Create or clean the directory in your Web server where HelloWorld files will be deployed.
4. Copy or ftp HelloWorld.swf and supporting files to this directory.

Now imagine that your HelloWorld application is a part of a larger application that includes Java modules that also need to be compiled and deployed under some server location. Deployment process of even a small size project usually involves at least a dozen of dependent tasks that have to be executed in a particular order. Creating and maintaining of a build script is well worth the effort

With Ant, you describe your project in a build file written in xml (it's typically called build.xml). An Ant project consists of targets (i.e. compile), which in turn consist of tasks (i.e. mkdir, exec). Listing below shows a build.xml that will create an output directory bin, and compile our HelloWorld application into this directory. This Ant build file has two targets: *init* and *compile*.

```
<project name="HelloWorld" default="compile">
    <property name="flex.mxmlc" location="C:\Program Files\Adobe\Flex Builder 2
Plug-in\Flex SDK 2\bin\mxmlc.exe" />
    <property name="dest.dir" value="bin" />

    <target name="init">
        <delete dir="${dest.dir}" />
        <mkdir dir="${dest.dir}" />
    </target>

    <target name="compile" depends="init">
        <exec executable="${flex.mxmlc}" failonerror="true">
            <arg line="-output '${dest.dir}/HelloWorld.swf'"/>
            <arg line="HelloWorld.mxml"/>
        </exec>
    </target>
</project>
```

The *init* target performs on the destination directory *bin* two tasks *delete* and *create*.

The *compile* target will compile HelloWorld.mxml producing bin\HelloWorld.swf. If we'd need to manually write a command that compiles HelloWorld.mxml into the bin directory, it'd look like this:

mxmlc -output bin/HelloWorld.swf HelloWorld.mxml

Besides targets and tasks, our build.xml contains two property tags. One of them defines the property called dest.dir that has a value bin, which is the name of the output directory for our compiled HelloWorld.swf:

<property name="dest.dir" value="bin" />

This means that wherever Ant will encounter dest.dir surrounded with ${ and }, it'll substitute it with the value of this property, which is bin in our case.

The above project also defines a property called flex.mxmlc that tells Ant where mxmlc compiler is located. The target compile uses Ant's task exec, which will run mxmlc compiler passing it arguments specified in <arg> tags.

Even though your installation of Eclipse contains Ant, for training purposes download Ant from http://ant.apache.org/. Installation is simple: just unzip the archive and add the path its bin directory (i.e. C:\apache-ant-1.6.5\bin) to the environment variable path of your OS.

Now we can open a command window, get into directory where HelloWorld.mxml and build.xml are located, and enter the command ant that will start the execution of the default target of our project, which is compile. But since compile target is marked as dependent on init, ant will execute the init target first.

Ant has extensive set of predefined tasks like copy, zip, jar, War, FTP, Mail, Condition, and many more described at Ant documentation.

For example, to make our build file work in both Windows and Unix environments, we can specify the location of mxmlc compiler as follows:

```
<condition property="flex.mxmlc" value="${sdkdir}/bin/mxmlc">
    <os family="unix"/>
</condition>

<condition property="flex.mxmlc" value="${sdkdir}/bin/mxmlc.exe">
    <os family="windows"/>
</condition>
```

You can pass the values of the properties from ant command line using –Dpropery=value option. For example, if we use the code snippet above, the property sdkdir can be passed from a command line:

ant –Dsdkdir=/opt/usr/freeflex

You can write more sophisticated ant scripts. For example, you can automate creation of an html wrapper (http://mxdj.sys-con.com/read/310378.htm) for swf files with *replaceregex* task applied to one of the provided html templates. You can write build scripts that will merge Flex generated artifacts with Java Web applications. Automating builds with Ant can get you pretty far.

There are lots of online articles and books on Ant, but do not miss Ant in Anger by Steve Loughran (http://ant.apache.org/ant_in_anger.html), where he discusses some core practices, naming conventions and team development process with Ant.

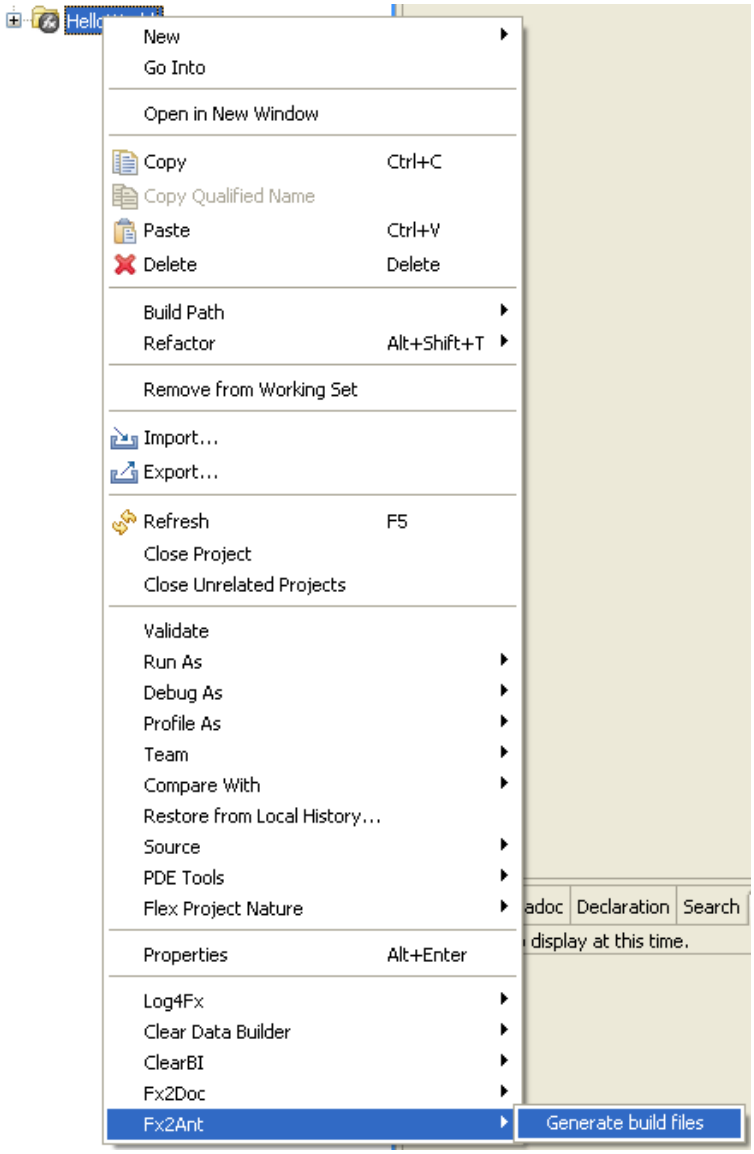
5.2 Building HelloWorld with Fx2Ant

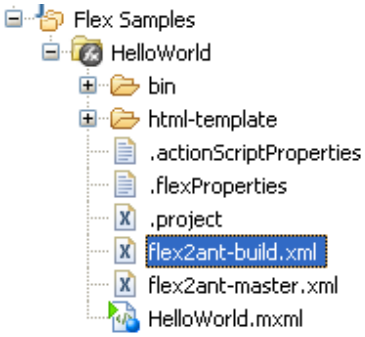
To start, let's create a simple Flex project by selecting the menus **File > New > Project… Flex > Flex Project** and press the button **Next**. On the next window select **Basic** as the data access type and press **Next**. Enter **HelloWorld** as the name of your project and press the button **Finish**.

This simple project will just show "**Hello World!"** in a Label control:

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">

  <mx:Label text="Hello World!"/>

</mx:Application>
```

As you've learned from section 4.1, you can automatically generate the Ant build file *flex2ant-build.xml* using the menus **Fx2Ant > Generate build files**.





Let's consider the generated targets in this file (we'll omit parameters for brevity).

```
<!-- ==================================
```

```
          target: build
         ================================= -->
    <target
     name="build"
     depends="clear,init,compile,copy.wrapper,prepare.wrapper"
     description="-->Compile flex application">
    </target>
```

In the section 5.1.1, we've created just two targets, but now we also find such targets as *clear, copy.wrapper, prepare.wrapper*.

```
<!-- - - - - - - - - - - - - - - - - -
          target: clear
         - - - - - - - - - - - - - - - - -->
 <target name="clear">
       <delete failonerror="false">
            <fileset dir="${build.dir}" includes="**/*.*" />
       </delete>
   </target>


<!-- - - - - - - - - - - - - - - - - -
          target: init
         - - - - - - - - - - - - - - - - -->
    <target name="init">
       <mkdir dir="${build.dir}"/>
    </target>
```

The first two targets *clear* and *init* are replacing the *init* target from 5.1.1. The only difference is that the target *clear* is executed first to delete the directory from the old build (*failonerror="false"* instructs Ant to not generate an error if such directory does not exist). And then the target *init* creates a new empty directory.

The compilation target is also not the same as in 5.1.1. First, we'll start Flex command line compiler not with exec, but with java runtime – we can do this because Flex compiler is nothing else but executable jar file. In the same command line we pass command line parameters for the Java Virtual Machine (i.e. *maxmemory*) and the libraries required for your Flex project – Fx2Ant is smart enough to pick them from your project's configurations.

```
    <!-- - - - - - - - - - - - - - - - - -
          target: compile
         - - - - - - - - - - - - - - - - - -->
    <target name="compile">
     <java
      jar="${mxmlc.jar}"
```

```
      fork="true"
      maxmemory="512m"
      failonerror="true">
      <arg line="${additional.compiler.arguments}" />
      <arg value="+flexlib=${flex.sdk.dir}/frameworks"/>
      <arg value="-context-root=${context.root}"/>
      <arg value="-load-config=${flex.config.xml}"/>
      <arg value="-output=${main.application.out}"/>
      <arg value="-source-path"/>
      <arg value="${src.dir}"/>

   <arg value="-library-path+=${FRAMEWORKS}/libs/utilities.swc"/>
   <arg value="-library-path+=${FRAMEWORKS}/libs/flex.swc"/>
   <arg value="-library-path+=${FRAMEWORKS}/libs/framework.swc"/>
   <arg value="-library-path+=${FRAMEWORKS}/libs/rpc.swc"/>
   <arg value="-library-path+=${FRAMEWORKS}/libs/charts.swc"/>
   <arg value="-library-path+=${FRAMEWORKS}/locale/{locale}"/>
   <arg value="-external-library-path+=${FRAMEWORKS}/libs/playerglobal.swc"/>
      <arg value="${main.application}" />
     </java>
    </target>
```

Finally, the last two targets *copy.wrapper* and *prepare.wrapper* create HTML wrapper for the generated swf file. The first one copies Flex HTML templates, and the second target uses a Ant's standard task replaceregexp to replace the default template values with the ones specific to your swf.

```
 <!-- - - - - - - - - - - - - - - - - -
          target: copy.wrapper
         - - - - - - - - - - - - - - - - - -->
    <target name="copy.wrapper">
     <copy todir="${build.dir}">
      <fileset dir="${wrapper.dir}" />
     </copy>
    </target>

 <!-- - - - - - - - - - - - - - - - - -
          target: prepare.wrapper
         - - - - - - - - - - - - - - - - - -->
    <target name="prepare.wrapper">
     <move file="${unnamed.output.html}" tofile="${output.html}" />
```

```
    <replaceregexp file="${output.html}" flags="gs"  match="\$\{width\}"
replace="${swf.width}"/>

    <replaceregexp file="${output.html}" flags="gs"  match="\$\{height\}"
replace="${swf.height}"/>

    <replaceregexp file="${output.html}" flags="gs"  match="\$\{title\}"
replace="${swf.title}"  encoding="utf-8"/>

    <replaceregexp file="${output.html}" flags="gs"
match="\$\{version_major\}" replace="${swf.version.major}"/>

    <replaceregexp file="${output.html}" flags="gs"
match="\$\{version_minor\}" replace="${swf.version.minor}"/>

    <replaceregexp file="${output.html}" flags="gs"
match="\$\{version_revision\}"  replace="${swf.version.revision}"/>

    <replaceregexp file="${output.html}" flags="gs"  match="\$\{application\}"
replace="${swf.application}"/>

    <replaceregexp file="${output.html}" flags="gs"  match="\$\{bgcolor\}"
replace="${swf.bgcolor}"/>

    <replaceregexp file="${output.html}" flags="gs"  match="\$\{swf\}"
replace="${swf.swf}"/>

   </target>
```

If you reconfigure your Flex project, for example, add new libraries, you won't need to modify the file flex2ant-build.xml manually – just re-run **Fx2Ant > Generate build files**, and the build file will be recreated according to your latest settings.

Nevertheless, if you decide to change one of the default settings, for example use non-standard Flex SDK (i.e. the one that comes with Flex Data Services). Just change one of the default flex2ant-build.xml features as explained in the next section.

6.0 CUSTOMIZING FX2ANT

This section covers internals of generated Ant files. After reading it you'll be able not only modify the process of generation of these files, but also generate your own build files using Fx2Ant.

6.1 The Build File for a Flex Project

Below is a fragment of a generated Ant build file generated for a Flex project.

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="SimpleFlexProject" default="build" basedir=".">

<property name="flex.sdk.dir" location="D:\Adobe\Flex Builder 2 Plug-in\Flex SDK
2"/>

<property name="main.application.path" value="SimpleFlexProject"/>

<property name="src.dir" location=""/>

<property name="build.dir" location="bin"/>

<property name="additional.compiler.arguments" value="-locale en_US"/>

 <property name="FRAMEWORKS" location="D:/Adobe/Flex Builder 2 Plug-in/Flex SDK
2/frameworks"/>

 <property name="DOCUMENTS" location="D:/Eclipse-3.2.1/workspace"/>

    <!-- property name="FRAMEWORKS" location="${flex.sdk.dir}/frameworks"/ -->


 <!-- dirs -->

 <property name="flex.config.xml" location="${flex.sdk.dir}/frameworks/flex-
config.xml"/>


 <!-- files -->

 <property name="mxmlc.jar" location="${flex.sdk.dir}/lib/mxmlc.jar"/>

 <property name="main.application"
location="${src.dir}/${main.application.path}.mxml"/>

 <property name="output.name" value="${main.application.path}"/>

 <property name="main.application.out"
location="${build.dir}/${output.name}.swf"/>


 <!-- wrapper -->

 <property

  name="wrapper.dir"

  location="${flex.sdk.dir}/resources/html-templates/express-installation-with-
history"/>

 <property name="output.html.name" value="${output.name}.html"/>

 <property name="output.html" location="${build.dir}/${output.html.name}"/>

 <property name="unnamed.output.html"
location="${build.dir}/index.template.html"/>
```

```
 <property name="swf.width" value="100%"/>
 <property name="swf.height" value="100%"/>
 <property name="swf.title" value="${output.name}"/>
 <property name="swf.version.major" value="9"/>
 <property name="swf.version.minor" value="0"/>
 <property name="swf.version.revision" value="0"/>
 <property name="swf.application" value="${output.name}"/>
 <property name="swf.swf" value="${output.name}"/>
 <property name="swf.bgcolor" value="#FFFFFF"/>

...
```

The meaning of the properties generated by Fx2Ant is listed in the table below.

Fx2Ant Property name	Description	Default value
General: these parameters can be used both in Flex Projects and in Flex Library Projects		
flex.sdk.dir	Path to Flex SDK	Flex SDK home
main.application.out	Name of generated swf file	${build.dir}/${output.name}.swf
src.dir	Main source folder	Project main source folder
build.dir	Output directory	Project output directory
additional.compiler.arguments	Additional compile arguments	Project additional compile arguments
FRAMEWORKS	Framework directory	Flex SDK frameworks
DOCUMENTS	Eclipse root directory	Your eclipse root directory
flex.config.xml	flex-config.xml file	${flex.sdk.dir}/frameworks/flex-config.xml
Flex Project parameters		
mxmlc.jar	MXML compiler	${flex.sdk.dir}/lib/mxmlc.jar
main.application	Path to main MXML file	${src.dir}/${main.application.path}.mxml
output.name	Name of the output file	${main.application.path}
main.application.path	Main application file name	Project main application file name
wrapper.dir	Wrapper directory	${flex.sdk.dir}/resources/html-templates/express-installation-with-history
output.html.name	Output html file name	${output.name}.html
output.html	Output html file path	${build.dir}/${output.html.name}
unnamed.output.html	Path to html template	${build.dir}/index.template.html
swf.width	SWF width	100%
swf.height	SWF height	100%
swf.title	Title of html page	${output.name}
swf.version.major	Flash Player major version	Project Flash Player major version
swf.version.minor	Flash Player minor version	Project Flash Player minor version
swf.version.revision	Flash Player version revision	Project Flash Player version revision
swf.application	SWF Application name for html wrapper	${output.name}
swf.swf	A generated SWF name for the html wrapper	${output.name}
swf.bgcolor	SWF background color	#FFFFFF

Flex Library Project parameters		
compc.jar	MXML compiler	${flex.sdk.dir}/lib/compc.jar
output.swc.name	Main application path	Project main application path

6.2 The Build File for a Flex Library Project

Below is a fragment of a generated Ant build file generated for a Flex Library project.

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="SimpleFlexLibrary" default="build" basedir=".">

<property name="flex.sdk.dir" location="D:\Adobe\Flex Builder 2 Plug-in\Flex SDK
2"/>

<property name="output.name" value="SimpleFlexLibrary"/>

<property name="src.dir" location="com"/>

<property name="build.dir" location="bin"/>

<property name="additional.compiler.arguments" value=""/>

<property name="flex.config.xml" location="D:\Eclipse-
3.2.1\workspace\SimpleFlexLibrary\lib2ant-flex-config.xml"/>

 <property name="FRAMEWORKS" location="D:/Adobe/Flex Builder 2 Plug-in/Flex SDK
2/frameworks"/>

 <property name="DOCUMENTS" location="D:/Eclipse-3.2.1/workspace"/>

 <!-- property name="FRAMEWORKS" location="${flex.sdk.dir}/frameworks"/ -->


    <property name="main.application.out"
location="${build.dir}/${output.name}.swc"/>

    <property name="compc.jar" location="${flex.sdk.dir}/lib/compc.jar"/>

 ...
```

The previous section of this document describes all parameters that can be used in such a project.

6.2.1 A sample lib2ant-flex-config.xml

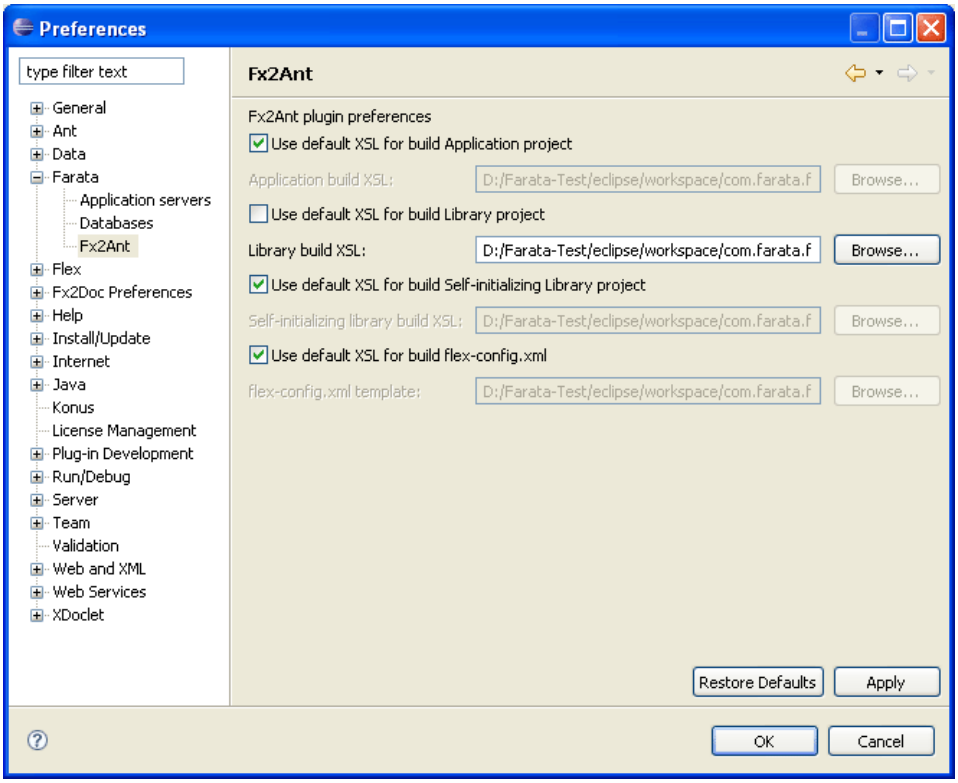
This file includes the required namespaces and classes. Fx2Ant generates this file from your project's metadata.

```
<flex-config xmlns="http://www.adobe.com/2006/flex-config">
   <compiler>
     <namespaces>
     </namespaces>
   </compiler>
   <include-namespaces>
     </include-namespaces>
   <include-classes>


  <class>farata.samples.ArrayFix</class>


   </include-classes>
```

```
</flex-config>
```

6.3 Using Your Own Templates

Fx2Ant generates build files using XSL templates. You can find the list of all used templates and their location in the Fx2Ant preferences window (**Windows > Preferences** and then **Farata > Fx2Ant**).



The table below contains a list of all templates used by Fx2Ant.

Template name	Description	Generated XML name
flex-build.xsl	An Application build XSL (see section **6.1**)	flex2ant-build.xml
flex-lib-build.xsl	A Library build XSL (see section **6.2**)	flex2ant-build.xml
flex-self-init-build.xsl	Self initializing library build XSL (see section **6.4**)	flex2ant-self-init-build.xml
flexLib2flexConfig.xsl	flex-config.xml template (see section **6.2.1**)	lib2ant-flex-config.xml

The table below shows side by side the content of the generated build file (**flex2ant-build.xml)** and the template (**flex-lib-build.xsl)** that was used for its generation. You can use it as a sample should you decide to create your own template to be used by Fx2Ant.

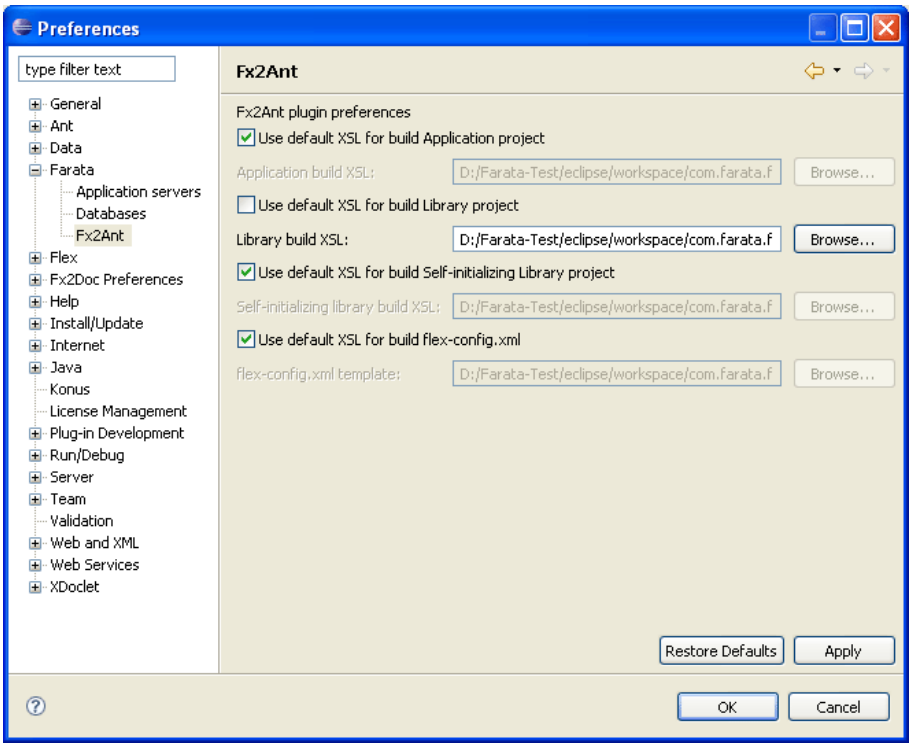
flex-lib-build.xsl	flex2ant-build.xml
<?xml version="1.0"?>	<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">	
<xsl:output indent="yes" method="xml" />	
<xsl:param name="FLEX_SDK" /> <xsl:param name="MAIN_APP" /> <xsl:param name="FLEX_CONFIG" />	
<xsl:template match="/"> <xsl:for-each select="//actionScriptProperties/compiler">	
<xsl:variable name="SRC_DIR" select="@sourceFolderPath" /> <xsl:variable name="BUILD_DIR" select="@outputFolderPath" /> <xsl:variable name="ARGS" select="@additionalCompilerArguments"/>	
<project name="{$MAIN_APP}" default="build" basedir="."> <property name="flex.sdk.dir" location="{$FLEX_SDK}" />	<project name="SimpleFlexLibrary" default="build" basedir="."> <property name="flex.sdk.dir" location="D:\Adobe\Flex Builder 2 Plug-in\Flex SDK 2"/>
<property name="output.name" value="{$MAIN_APP}" /> <property name="src.dir" location="{$SRC_DIR}" />	<property name="output.name" value="SimpleFlexLibrary"/> <property name="src.dir" location="com"/>
<property name="build.dir" location="{$BUILD_DIR}" /> <property name="additional.compiler.arguments" value="{$ARGS}" /> <property name="flex.config.xml" location="{$FLEX_CONFIG}"/>	<property name="build.dir" location="bin"/> <property name="additional.compiler.arguments" value=""/> <property name="flex.config.xml" location="D:\Eclipse-3.2.1\workspace\SimpleFlexLibrary\lib2ant-flex-config.xml"/>
<xsl:for-each select="//compiler/variables"> <xsl:text disable-output-escaping="yes"><![CDATA[<property name="]]></xsl:text><xsl:value-of select="@name"/><xsl:text disable-output-escaping="yes"><![CDATA[" location="]]></xsl:text><xsl:value-of select="@value"/><xsl:text disable-output-escaping="yes"><![CDATA["/>]]></xsl:text>	<property name="FRAMEWORKS" location="D:/Adobe/Flex Builder 2 Plug-in/Flex SDK 2/frameworks"/> <property name="DOCUMENTS" location="D:/Eclipse-3.2.1/workspace"/>

```
</xsl:for-each>

<xsl:text disable-output-
escaping="yes"><![CDATA[
      <!-- property name="FRAMEWORKS"
location="${flex.sdk.dir}/frameworks"/ -->

    <property name="main.application.out"
location="${build.dir}/${output.name}.swc"/>
    <property name="compc.jar"
location="${flex.sdk.dir}/lib/compc.jar"/>

    <!-- ================================
          target: build
         ================================ -->
    <target
     name="build"
     depends="clear,init,compile"
     description="-->Compile flex library">
    </target>
...
```

```
      <!-- property name="FRAMEWORKS"
location="${flex.sdk.dir}/frameworks"/ -->

    <property name="main.application.out"
location="${build.dir}/${output.name}.swc"/>
    <property name="compc.jar"
location="${flex.sdk.dir}/lib/compc.jar"/>

    <!-- ================================
          target: build
         ================================ -->
    <target
     name="build"
     depends="clear,init,compile"
     description="-->Compile flex library">
    </target>
...
```

To use your own templates instead of the standard ones, perform the following steps:

1. Create an XSL file that Fx2Ant should use for generation of the build files.
2. Select **Windows > Preferences...** from Eclipse menu.
3. Select **Farata > Fx2Ant** from the list on the left. You'll see a selection of the XSL templates used for generation of Ant build files. Un-check the Use Defaults checkbox and specify your own XSL template.



6.4 Self Initializing Libraries

Mid-size or large applications can consist of Flex applications (SWF) and a number of Flex library files (SWC). An SWC is an archive file, much like ZIP or JAR, and it contains library.swf and catalog.xml files. The latter describes the hierarchy of dependencies found the former. Depending on how you link library files to your application, the library.swf may be expanded into a SWF file. These are the three ways of linking SWC's to your Flex project:

• **External -** The SWC's *catalog.xml* will be used to resolve references; however the definitions contained in *library.swf* won't be included in the body of your Flex application SWF.The External link type assumes that by the time *Flex application* needs to create instances of the classes cotained in the *library.swf* the definitions for these classes will be accessible from the current application domain.

• **RSL –** The SWC's *catalog.xml* will be used to resolve references; the definitions contained in *library.swf* won't be included in the body of the *FlexApplication.swf*. The difference between the RSL and External is that in the RSL scenario, all definitions originally contained in the *library.swf* will be upfront-loaded into the main applicationDomain during the application startup, which is not the case with external linkage.

• **Merge-in** - The objects and resources contained in the *library.swf* get embedded directly into your application's SWF FlexApplication. This will only happen for those objects that are explicitly referenced by the application code. This is a default option for statically linked applications and guarantees that the definitions of all referenced classes as well as the classes they depend on are loaded into the main application domain.

A Merge-in scenario is often called *static linking*, while External and RSL are the cases of *dynamic linking*. The problem with RSL's though is that if you do not mention in your Flex application a class located in the SWC library, you'll get a runtime error "Can not access a property or method or a null object reference", which means that the Flex application and the library are tightly coupled.

We offer you a brief description of the workaround, that we'll call Self-Initializing Libraries, and you can find detailed description of this techniques in Chapter 10 of the book "Rich Internet Application with Adobe Flex and Java" written by creators of Fx2Ant and published by Sys-Con Publications (see www.theriabook.com). It'll allow your application to use multiple SWC libraries, and Flex application will be able to initialize their classes without the need to know class names in advance. Ant (and Fx2Ant) will help us create such self-initializing library.

Instead of using standard routine of creating SWC with compc and extracting SWF out of it, Fx2Ant will generate the build script that will create this SWF as if it's not a library, but an application that will automatically load all library classes into the *currentDomain* of the hosting application. To minimize the size of the resulting SWF, Fx2Ant will mark Flex Framework's classes as external, assuming that the main application will load them anyway.

This size-optimizing builds prepared by Fx2Ant can help you cut down the download time of your large applications. If you use RSL libraries, Fx2Ant will help you to prevent notorious run-time exceptions caused by improper dynamic class invocations.