Clear Toolkit 3.0 Beta

DTO2Fx

Code generator of Data Transfer Objects

User Guide

Revision: August,17 2008

Table of Contents

1.0 OVERVIEW 3

2.0 INSTALLING DTO2FX AND CONFIGURING JAVA PROJECTS IN ECLIPSE IDE 3

3.0 HOW TO GENERATE ACTIONSCRIPT CLASSES 6

3.1 Ignoring unsupported data types 11

4.0 APPENDIX. CONTACT INFORMATION 16

1.0 OVERVIEW

DTO2Fx is Eclipse plugin that automatically generates ActionScript classes to be used in Flex remoting as data transfer objects (DTO a.k.a. Value Objects) from their Java peers.

It also supports generation of the ActionScript interfaces from the interfaces defined in Java.

DTO2Fx is available free of charge under the Mozilla Public *License* (MPL) as Farata's contribution to Flex community.

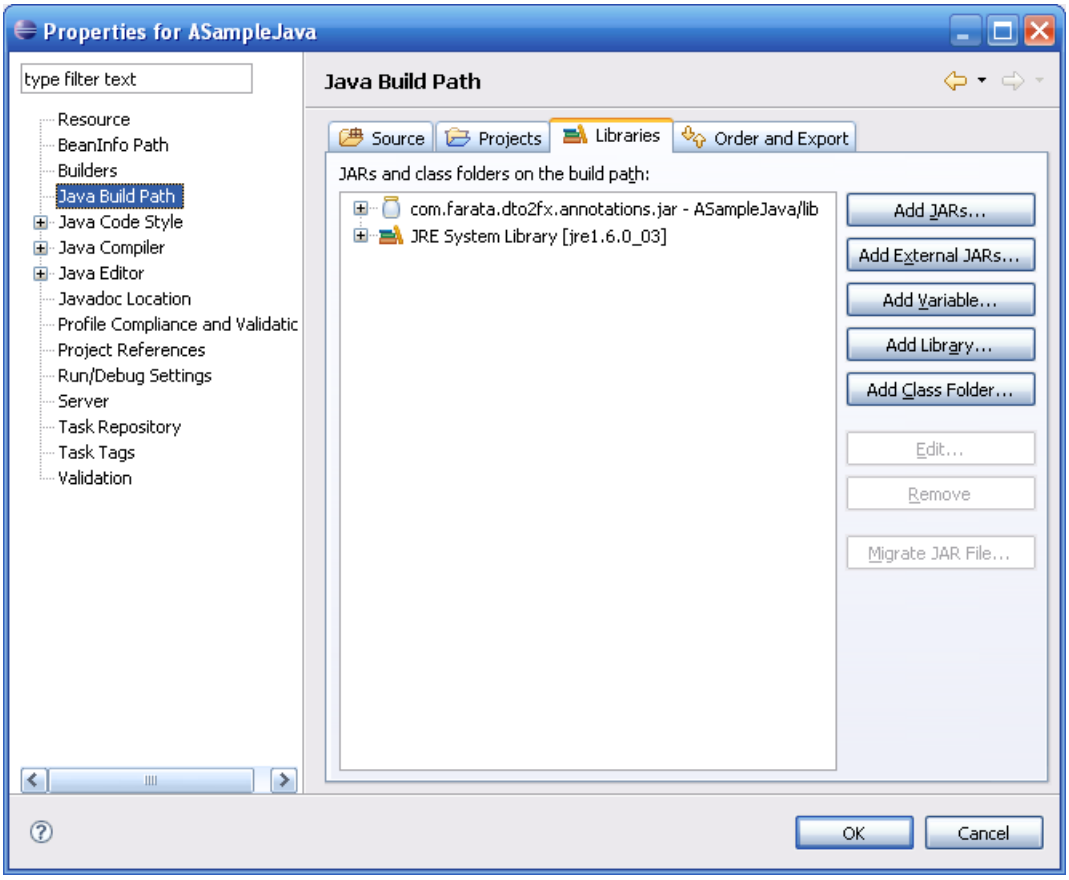
2.0 INSTALLING DTO2FX AND CONFIGURING JAVA PROJECTS IN ECLIPSE IDE

1. Download DTO2Fx plugin http://www.myflex.org/beta/dto2fx/com.farata.dto2fx.asap_1.0.0.jar and save it in your <ECLIPSE_HOME>/plugins directory.

2. Create Java Project in Eclipse or open an existent one containing Java DTO to be used for generation of ActionScript classes.

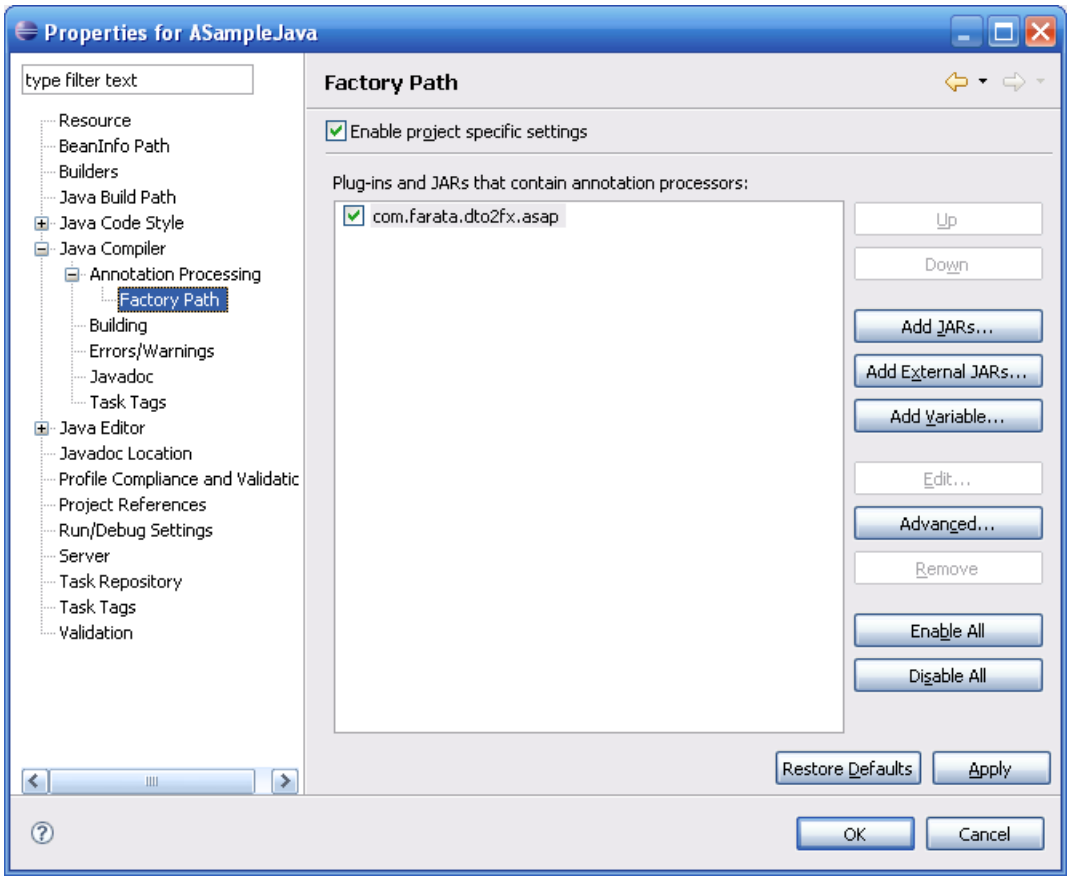
   Important : DTO2Fx works with Java projects that use JDK 5.0 or later .

3. Download DTO2Fx annotation library http://www.myflex.org/beta/dto2fx/com.farata.dto2fx.annotations.jar and add it as an external JAR to the project libraries in the Java Build Path window.



4. Enable annotation processing for your Java project.

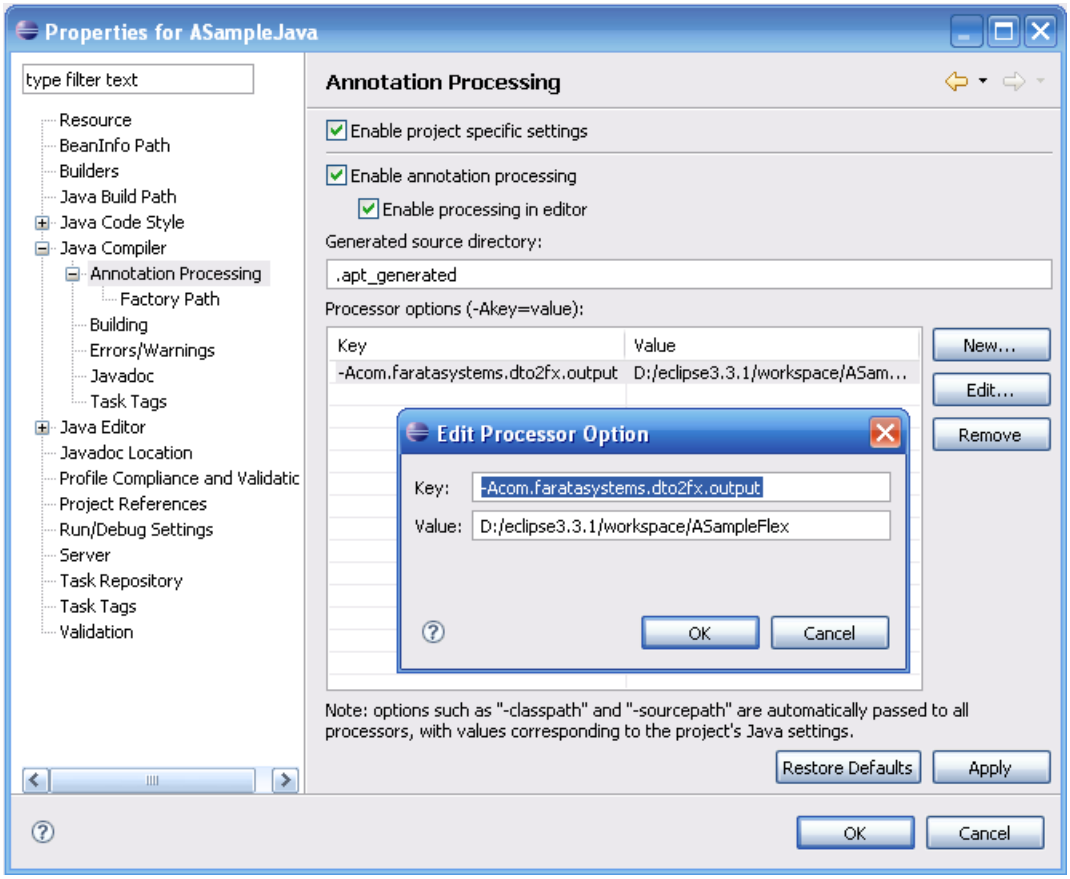
Open *Project -> Properties*, navigate to *Java Compiler -> Annotation Processing -> Factory Path* and select DTO2Fx plugin.



5. Configure the output parameter of DTO2FX annotations processing tool so it knows where to put generated files.

   Add a new option for Annotation processing named ***-Acom.faratasystems.dto2fx.output***. Typically you'll be specifying your Flex Builder project where you want the generated ActionScript classes to be created, but it can be any directory on your disk.

6. .:



Configuration of your Java project is complete.

3.0 HOW TO GENERATE ACTIONSCRIPT CLASSES

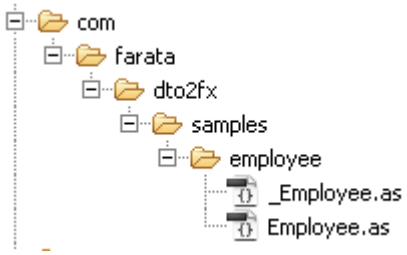
In this section we'll explain how to apply DTO2Fx annotations to a Java class or an interface. We'll use the following class as an example:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;

@FXClass
public class Employee {
      private String   firstName;
      private String   lastName;
      private Double   salary;

      public String getFirstName() {
            return firstName;
      }
      public void setFirstName(String firstName) {
```

```
            this.firstName = firstName;
      }

      public String getLastName() {
            return lastName;
      }
      public void setLastName(String lastName) {
            this.lastName = lastName;
      }

      public Double getSalary() {
            return salary;
      }
      public void setSalary(Double salary) {
            this.salary = salary;
      }
}
```

Please note the Java annotation `@FXClass` that is implemented in DTO2Fx and invokes the Java annotation processing tool that generates the ActionScript class.

As soon as you save the Employee.java class, the corresponding ActionScript class will be generated in directory defined by *Acom.faratasystems.dto2fx.output* parameter of DTO2FX annotation processing tool. If you don't see it right away, refresh the Flex Builder's project.



Two classes _Employee.as and Employee.as will be generated. This is an implementation of the design pattern *generation gap* (see http://www.research.ibm.com/designpatterns/pubs/gg.html).

The _Employee.as is for use by the code generator only. And it'll be overwritten with every recompile of the Java class. Please note that the name of this class starts with the underscore symbol.

```
package com.farata.dto2fx.samples.employee {


  import mx.events.PropertyChangeEvent;

  import flash.events.EventDispatcher;
  import mx.core.IUID;
  import mx.utils.UIDUtil;
```

```
  [ExcludeClass]
  public class _Employee extends flash.events.EventDispatcher implements mx.core.IUID
{

    /* Constructor */
    public function _Employee():void {
      super();
    }

    // implementors of IUID must have a uid property
    private var _uid:String;

    [Transient]
    [Bindable(event="propertyChange")]
    public function get uid():String {
      // If the uid hasn't been assigned a value, just create a new one.
      if (_uid == null) {
        _uid = mx.utils.UIDUtil.createUID();
      }
      return _uid;
    }

    public function set uid(value:String):void {
      const previous:String = _uid;
      if (previous != value) {
        _uid = value;
        dispatchEvent(
          mx.events.PropertyChangeEvent.createUpdateEvent(
            this, "uid", previous, value
          )
        );
      }
    }


    /* Property "firstName" */
    private var _firstName:String;

    [Bindable(event="propertyChange")]
      public function get firstName():String {
      return _firstName;
    }
    public function set firstName(value:String):void {
      const previous:String = this._firstName;
      if (previous != value) {
        _firstName = value;
```

```
        const ev:mx.events.PropertyChangeEvent =
mx.events.PropertyChangeEvent.createUpdateEvent(
          this, "firstName", previous, _firstName
        );
        dispatchEvent(ev);
      }
    }

    /* Property "lastName" */
    private var _lastName:String;

    [Bindable(event="propertyChange")]
      public function get lastName():String {
      return _lastName;
    }
    public function set lastName(value:String):void {
      const previous:String = this._lastName;
      if (previous != value) {
        _lastName = value;
        const ev:mx.events.PropertyChangeEvent =
mx.events.PropertyChangeEvent.createUpdateEvent(
          this, "lastName", previous, _lastName
        );
        dispatchEvent(ev);
      }
    }

    /* Property "salary" */
    private var _salary:Number;

    [Bindable(event="propertyChange")]
      public function get salary():Number {
      return _salary;
    }
    public function set salary(value:Number):void {
      const previous:Number = this._salary;
      if (previous != value) {
        _salary = value;
        const ev:mx.events.PropertyChangeEvent =
mx.events.PropertyChangeEvent.createUpdateEvent(
          this, "salary", previous, _salary
        );
        dispatchEvent(ev);
      }
    }

  }
```

```
}
```

The second class Employee.as extends the first one and can be used by developers may add their own variables or functions, which won't be overriten by DTO2Fx code generator. For example, you may want to add a function to return the full name of employee.

```
package com.farata.dto2fx.samples.employee {

  [RemoteClass(alias="com.farata.dto2fx.samples.employee.Employee")]

  public class Employee extends com.farata.dto2fx.samples.employee._Employee {

    /* Constructor */
    public function Employee():void {
      super();
    }

    public function get fullName():String{
      return firstName + " " + lastName;
    }
  }

}
```

If you'd like to provide a provide a different name for the generated ActionScript file. To do this just use either of the @FXClass annotation forms:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;

@FXClass("com.farata.dto2fx.samples.vo.EmployeeVO")
public class Employee {
      ...
}
```

...or

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;

@FXClass(value="com.farata.dto2fx.samples.vo.EmployeeVO")
public class Employee {
      ...
}
```

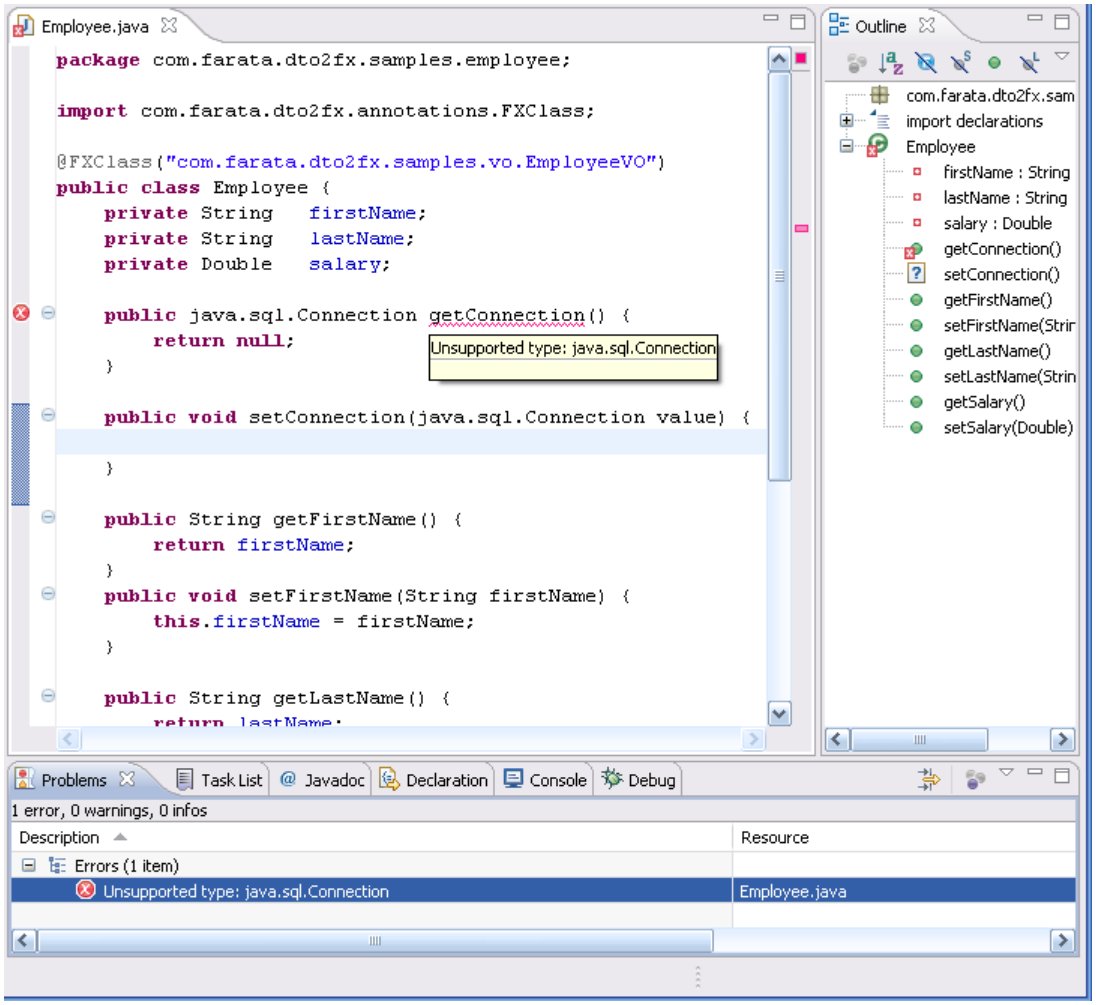
From what you see above, generated ActionScript class contains properties defined at Java side as pairs of getter/setter JavaBean-like property methods. It is also possible to use *non-transient non-static public fields* at Java side to define properties:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;

@FXClass(value="com.farata.dto2fx.samples.vo.EmployeeVO")
public class Employee {
      public String firstName;
      public String lastName;
      public Double salary;
}
```

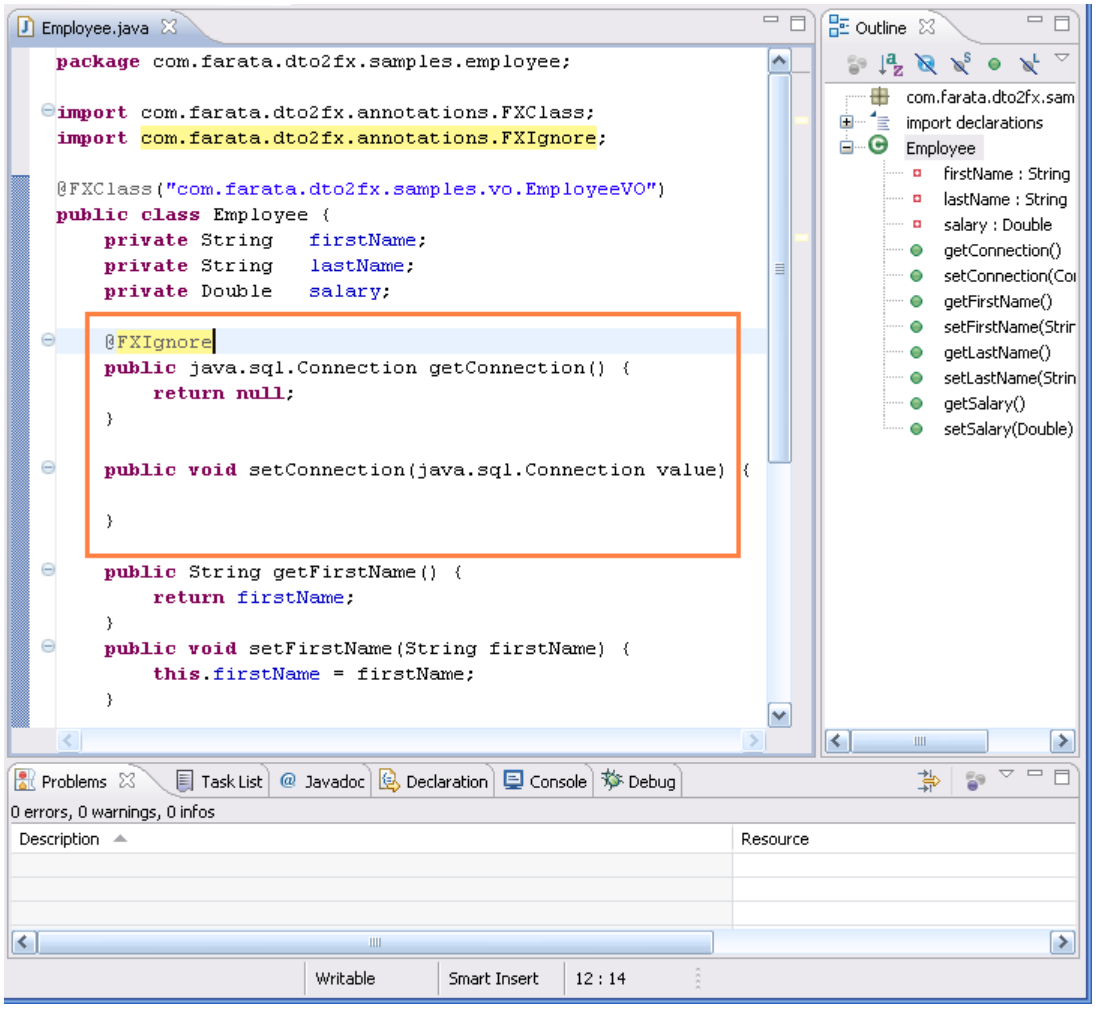
3.1 Ignoring unsupported data types

During code generation, DTO2Fx supports all data types supported by Flex serialization. It also supports all other custom Java objects annotated with @FXClass. This means that your Java classes can include member variables of custom data types.

If you use unsupported data type, the DTO2Fx annotation processing tool will stop generation of corresponding class and report an error. For example it would not know how to generate ActionScript variables that would correspond to Java's Connection data type:



But what if you need to have property of this type on the Java side? To stop it from been picked by DTO2Fx, annotate it with `@FXIgnore` annotation.



Note that *@FXIgnore* is applied to the getter but will exclude both the getter and the setter from annotation processing. If it is applied to setter method (setConnection in this case) only the setter will be excluded from the generation process, and you will end up with a read-only ActionScript sproperty. This feature might be useful when you are applying DTO2Fx tool to interfaces:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;
import com.farata.dto2fx.annotations.FXIgnore;

@FXClass(value="com.farata.dto2fx.samples.vo.IEmployee")
public interface IEmployee {

	public String getFirstName( );
	@FXIgnore
	public void setFirstName(String firstName);

	public String getLastName();
	@FXIgnore
	public void setLastName(String lastName);
```

```
        public Double getSalary();
        @FXIgnore
        public void setSalary(Double salary);
}
```

The generated interface will define all properties as read-only:

```
package com.farata.dto2fx.samples.vo {

  import flash.events.IEventDispatcher;
  import mx.core.IUID;

  [ExcludeClass]
  public interface _IEmployee extends flash.events.IEventDispatcher, mx.core.IUID {

    /* Property firstName */
    function get firstName():String;

    /* Property lastName */
    function get lastName():String;

    /* Property salary */
    function get salary():Number;

  }

}
```

Since we can annotate interfaces, we may implement them in other @FXClass-es as well as extend @FXClass'es from base classes:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;

@FXClass(value="com.farata.dto2fx.samples.vo.EmployeeVO")
public class Employee implements IEmployee {
      ...
}
```

All superclasses (besides java.lang.Object) and all interfaces in the inheritance chain of the class annotated with @FXClass must be also be annotated.If you need to have some superclass/interface with functionality that makes sense only at Java side it may be discarded as well from annotation processing:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;
```

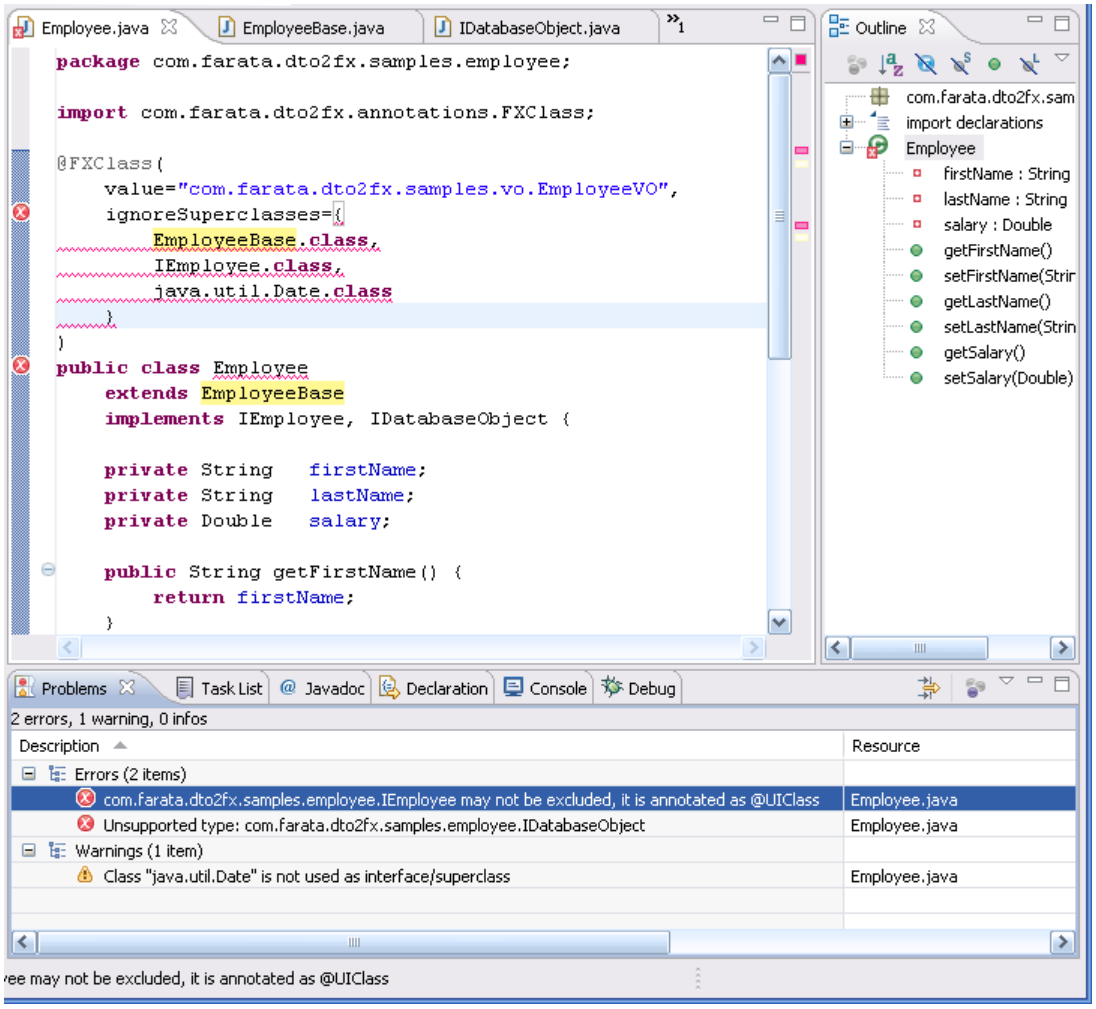
```
@FXClass(
  value="com.farata.dto2fx.samples.vo.EmployeeVO",
  ignoreSuperclasses={EmployeeBase.class, IDataBaseObject.class}
)
public class Employee extends EmployeeBase implements IEmployee, IDataBaseObject {
      ...
}
```

i. e. you just enumerate all unnecessary superclasses/interfaces via @FXClass.ignoreSuperclasses attribute. There is also a shorter form; however, it should be used with care: it's possible to automatically exclude all superclasses/interfaces these are not annotated as @FXClass with the following statement:

```
package com.farata.dto2fx.samples.employee;

import com.farata.dto2fx.annotations.FXClass;
import com.farata.dto2fx.annotations.FXIgnore;

@FXClass(
  value="com.farata.dto2fx.samples.vo.EmployeeVO",
  ignoreSuperclasses={FXIgnore.any.class}
)
public class Employee extends EmployeeBase implements IEmployee, IDataBaseObject {
      ...
}
```

Don't use @FXIgnore.any.class with any other class in list, or to use it twice. Also, it is wrong to list here classes these are annotated with @FXClass: this limitation is caused by requirement to preserve validity of generated code without complex reprocessing of new type system created by excluding arbitrary @FXClass-es.

Here is an example of some correctly applied and some incorrect applications of @FXClass.ignoreSuperclass:



The class EmployeeBase is not annotated and is correctly excluded from processing; IEmployee is annotated as @FXClass and it is an error to list it; IDatabaseObject is not annotated as @FXClass and should be excluded; java.util.Date is no a base superclass or interface, hence warning is shown.

4.0 APPENDIX. CONTACT INFORMATION

Since DTO2Fx is in Beta at this time, please report bugsif any by sending an email at support@faratasystems.com.

You can also leave a comment under the Contact Us menu at Farata Systems Web site: http://www.faratasystems.com.