Creating and Editing ActionScript and Flex Application Elements

When working on ActionScript and Flex applications, you create and edit application elements such as packages, ActionScript classes and interfaces, and MXML components.

For ActionScript classes, interfaces and MXML components, IntelliJ IDEA provides a number of predefined file templates.

- Creating a package
- Creating an ActionScript class or interface
- Creating an MXML component
- Initiating ActionScript class creation in the editor
- Template-based ActionScript classes, interfaces and MXML components
- Predefined file template variables for ActionScript and Flex
- An example of creating a custom file template for an MXML component
- Using the SWF metadata tag to control HTML wrapper properties
- Editing ActionScript and Flex sources

Creating a package

1. In the **Project** tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new package. Choose **File | New | Package**, or press `Alt+Insert` and select **Package**.

   Alternatively, right-click the corresponding folder or package and select **New | Package** from the context menu.

2. In the **New Package** dialog that opens, specify the package name and click **OK**.

   Note that you can create more than one package at once if you use dots (`.`) to separate the package names. For example, if you type `myPackage.mySubpackage` and none of these packages currently exists, both these packages (`myPackage` and `mySubpackage`) will be created.

   Also note that you can create a new package when creating a new ActionScript class or interface, or an MXML component. See Creating an ActionScript class or interface and Creating an MXML component.

Creating an ActionScript class or interface

1. In the **Project** tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new ActionScript class or interface. Choose **File | New | ActionScript Class**, or press `Alt+Insert` and select **ActionScript Class**.

   Alternatively, right-click the corresponding folder or package and select **New | ActionScript Class** from the context menu.

2. In the New ActionScript Class dialog that opens, specify the name of the class, the package, the file template to be used, and click **Create**. If the Class with Supers template is used, you can also specify a superclass and/or one or more interfaces that the class should implement.

   Note that if you specify a package that doesn't yet exist, the corresponding package will be created.

See also, Initiating ActionScript class creation in the editor.

Creating an MXML component

1. In the **Project** tool window, select your source root folder (e.g. `src`) or the package in which you want to create a new MXML component. Choose **File | New | MXML Component**, or press `Alt+Insert` and select **MXML Component**.

   Alternatively, right-click the corresponding folder or package and select **New | ActionScript Class** from the context menu.

2. In the New MXML Component dialog that opens, specify the name of the component, the package, the file template to be used, the parent component, and click **Create**.

   Note that if you specify a package that doesn't yet exist, the corresponding package will be created.

Initiating ActionScript class creation in the editor

When working with the source code of a class, you can start creating its subclass right in the editor. Similarly, you can initiate creating a class implementing an interface. For these purposes, IntelliJ IDEA provides intention actions called Create Subclass and Implement Interface respectively.

Here is an example of using the Create Subclass intention action. (The Implement Interface action is accessed in a similar way.)

1. In the editor, place the cursor within the line containing the class declaration.
2. Press `Alt+Enter` and select **Create Subclass**.
3. In the New ActionScript Class dialog that opens, specify the settings for your new class and click **Create**.

Template-based ActionScript classes, interfaces and MXML components

ActionScript classes, interfaces and MXML components are created according to file templates. The following predefined templates are available:

- **ActionScript Class.** This template generates the file contents shown below (the destination package and the name of the class are specified when creating a class):

```
            package myPackage {
public class MyClass {
    public function MyClass() {
    }
}
}
```

- **ActionScript Class with Supers.** This template generates the file contents shown below (the destination package, the class name, the superclass and/or the interfaces that the class implements are specified when creating a class):

```
            package myPackage {
public class MyClass1 extends MyClass implements IMyInterface1, IMyInterface2 {
    public function MyClass1() {
        super();
    }
}
}
```

- **ActionScript Interface.** This template generates the file contents shown below (the destination package and the interface name are specified when creating an interface):

```
            package myPackage {
public interface IMyInterface {
}
}
```

- **Flex 3 Component.** This template generates the file contents shown below (the root tag is defined by the parent component specified when creating the component; when generating the following example, `mx.core.Application` was specified as the parent component):

```
<?xml version=
            "1.0"?>
<mx:Application
            xmlns:mx=
            "http://www.adobe.com/2006/mxml">
</mx:Application>
```

- **Flex 4 Component.** This template generates the file contents shown below (the root tag is defined by the parent component specified when creating the component; when generating the following example, `spark.components.Application` was specified as the parent component):

```
<?xml version=
            "1.0"?>
<s:Application
            xmlns:fx=
            "http://ns.adobe.com/mxml/2009"
            xmlns:s=
            "library://ns.adobe.com/flex/spark">
</s:Application>
```

If necessary, you can modify the predefined templates or create your own, custom file templates. See Creating and Editing File Templates and An example of creating a custom file template for an MXML component.

See also, Predefined file template variables for ActionScript and Flex.

Predefined file template variables for ActionScript and Flex

For ActionScript and Flex file templates, the list of predefined template variables is broader. The following predefined variables are available in addition:

- `${Superclass}` - a superclass for an ActionScript class or a parent component for an MXML component.
- `${SuperInterfaces}` - a list of the interfaces that an ActionScript class implements.

An example of creating a custom file template for an MXML component

As already mentioned, the predefined MXML 4 Component file template just generates the root tag for an MXML component. Let's assume that you want the `<fx:Declarations>` and `<fx:Script>` tags to be generated in addition.

If for some reason you want to keep the predefined template unchanged, you can create the corresponding custom file template.

1. Open the Settings dialog (e.g. **Ctrl+Alt+S**).
2. In the left-hand pane, under **IDE Settings**, select **File Templates**.
3. On the File Templates page that opens in the right-hand part of the dialog, select the **Templates** tab.
4. Select **Flex 4 Component**.
5. Now, to create a copy of this template, click  .
6. Change the name of the template, for example, to *Flex 4 Component with Declarations and Script*.
7. In the template body (shown in the area under the **Reformat according to style** check box), after the line

   ```
   <${Superclass} xmlns:fx="http://ns.adobe.com/mxml/2009">
   ```

   add

   ```
   <fx:Declarations>
   </fx:Declarations>
   <fx:Script><![CDATA[
   ]]></fx:Script>
   ```
8. Click **OK** in the **Settings** dialog.

Now, to check the result:

1. Select your source root folder (e.g. `src`) and press **Alt+Insert**. Note that the **Flex 4 Component with Declarations and Script** option (which corresponds to the name of your new template) is now available in the **New** menu.
2. Select **MXML Component**. In the **New MXML Component** dialog that opens, check the contents of the **Template** list. Note that **Flex 4 Component with Declarations and Script** has also been added to this list.

Using the SWF metadata tag to control HTML wrapper properties

You can use the SWF metadata tag in your main application class to set the title, the background color, and the `width` and `height` properties in your HTML wrapper.

For example, if the ActionScript class contains

```
    package myPackage {
  import flash.display.Sprite;
  [SWF(pageTitle="hello", backgroundColor="#ccddee", width="400", height="200")]
    public class MyClass extends Sprite {
      ...
  }
}
```

the corresponding properties will be set in the HTML wrapper.

In a similar way the SWF metadata tag will work for an MXML component if the corresponding file contains:

```
<?xml version=
      "1.0"?>
```

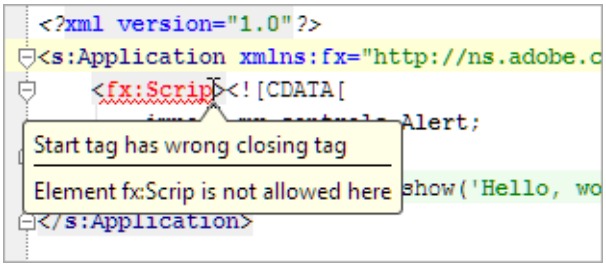
```
<s:Application
      xmlns:fx=
      "http://ns.adobe.com/mxml/2009"
      xmlns:s=
      "library://ns.adobe.com/flex/spark">
    <fx:Metadata>
        [SWF(pageTitle="hello", backgroundColor="#ccddee", width="400", height="200")]
    </fx:Metadata>
    ...
</s:Application>
```

By default, the title is the same as the name of the embedded `.swf` file, the background color is white (`#ffffff`), and the `width` and `height` are both `100%`.

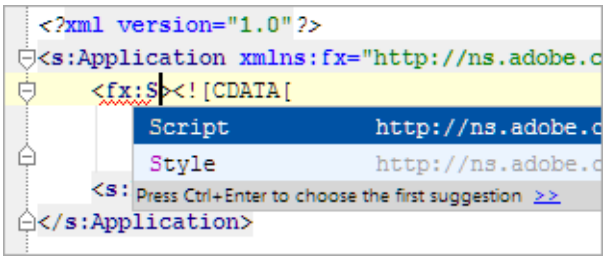
Editing ActionScript and Flex sources

When editing your ActionScript and Flex sources, you can use the following IntelliJ IDEA features:

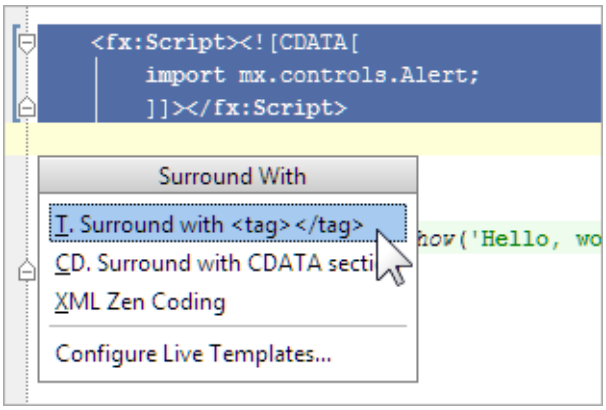
- Syntax and error highlighting. Note that the way your code is highlighted is defined by an active build configuration.



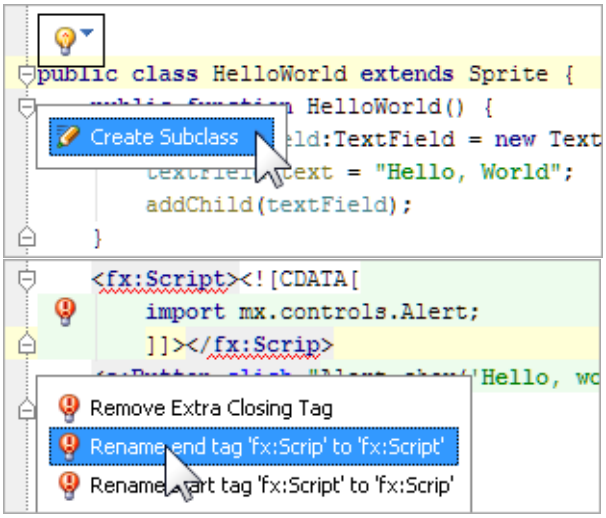
- Code completion.



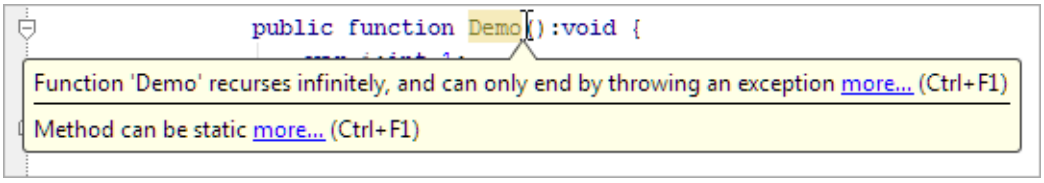
- Surrounding with tags or code constructs (**Ctrl+Alt+T** and **Ctrl+Alt+J**), and removing enclosing tags (**Ctrl+Shift+Delete**).



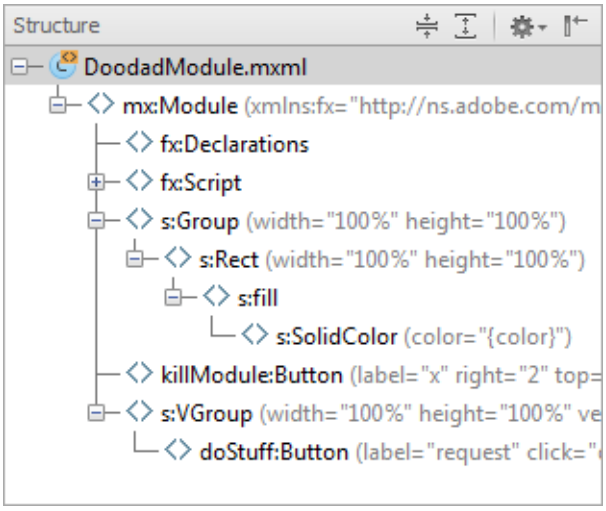
- Intention actions and quick fixes (**Alt+Enter**).



- Code inspections.



- Structure view for MXML components and ActionScript classes.



- Jumping to declarations (**Ctrl+B**, **Ctrl+Button1 Click** or **Button2 Click**).
- Code refactoring.
- Automatic code generation (**Alt+Insert**): generating getters and setters, bindable getters and setters, event handlers, etc.

See Also

Procedures:

- Advanced Editing Procedures

Language and Framework-Specific Guidelines:

- ActionScript and Flex

External Links:

- Developing, running and packaging AIR mobile applications for Android
- The Extract Interface and Extract Superclass refactorings for Flex and ActionScript
- The Change Method Signature refactoring for ActionScript and Flex

Getting Started:

- Familiarize Yourself with IntelliJ IDEA Editor

Web Resources:

- Developer Community