# Tips for learning ActionScript 3.0

ActionScript 3.0 is a powerful object-oriented language that represents a new programming model for the Flash Player runtime. If you are already familiar with ActionScript 1.0 or 2.0, you should be aware of some language differences as you develop your first application using ActionScript 3.0.

**Note:** For more information, refer also to the list of differences between ActionScript 2.0 and ActionScript 3.0 in the Flex 2.0 Language Reference.

To help ease the transition to ActionScript 3.0, I've compiled the following list of tips and common issues you might encounter during development.

- **Declare types for all variables, parameters, and return values.** Declaring a type for all variables, parameters, and return values is not required, but it is considered best practice. It will help the compiler give you more helpful error messages. It also aids runtime performance because the virtual machine will know the types you're working with ahead of time. In fact, it's so important that we've made it a warning by default.
- **Note that declarations with no access specifier now default to** `package internal`**, not** `public`**.** The default access specifier for declarations is now `internal` instead of `public`, meaning that the definition is visible only to the package containing the definition, not to all code. This is consistent with other languages such as Java. Because ActionScript 2.0 declarations defaulted to `public`, this change will likely be a common pitfall, so always put an access specifier on your declarations to make the intent crystal-clear. To encourage this best practice, the ActionScript 3.0 compiler will output a warning when no access specifier is used.
- **Note that classes are sealed by default, meaning properties cannot be added dynamically at runtime.** Classes can now be either dynamic or sealed. Dynamic classes can add additional dynamic properties at runtime; sealed classes cannot. Sealed classes conserve memory because no internal hash table is needed to store dynamic properties, and the compiler can provide better error feedback. The declaration `class Foo` is sealed. To declare a class dynamic, use the `dynamic` keyword—for example, `dynamic class Foo`.
- **Use `package` declarations to put a class definition into a package.** The `package` keyword is new to ActionScript 3.0.

  **ActionScript 2.0 code:**

  ```
  class mx.controls.Button { ... }
  ```

  **ActionScript 3.0 code:**

  ```
  package mx.controls { class Button { .. } }
  ```

  As in ActionScript 2.0, a `public` class must be in a file with the same name as the class. Multiple classes may be declared in a single file, but only one class may be public, and its name must match the filename.

- **Import classes, even if references to the class are fully qualified.** To use a class `MyPackage.MyClass`, you must import it with:

  ```
  import MyPackage.MyClass;
  ```

  This is true even if all references are fully qualified, that is using the full name

MyPackage.MyClass. In ActionScript 3.0, the import statement indicates that you want to use a class definition from another package, whereas in ActionScript 2.0 it was only used to create shorthand names. In ActionScript 3.0, the full class name is only used for disambiguation, and is no longer a substitute for the import statement.

It is also possible to import all of the definitions in a package using the * wildcard character:

```
import MyPackage.*;
```

It is considered best practice to import definitions individually, because it results in less ambiguity about which definitions your code uses.

- **Always mark method overrides.** The override keyword helps avoid common pitfalls of overriding methods, such as specifying the wrong name or method signature for an overridden method, or when the name of the overridden method changes. It also makes it clear when looking at the code that a method is being overridden. Because it knows whether a method is intended to override another, the compiler can perform more useful validation. The override keyword in ActionScript 3.0 is inspired by the C# override keyword.
- **Declare return types in your functions.** It is considered best practice to declare a return type for a function. If you omit a return type, a warning will be displayed. This is done for type safety, so that you don't accidentally leave a return type off and get the default return type of Object. If a function doesn't return any value, declare its return type as void.
- **Note that delegates are now built into the language, making event dispatching easier.** In ActionScript 2.0, routing an event to a method required use of the mx.utils.Delegate class or other workarounds:

```
import mx.utils.Delegate;
myButton.addEventListener("click", Delegate.create(this, onClick));
```

  In ActionScript 3.0, a reference to a method automatically remembers the object instance from which it was extracted. This is called a *method closure*. In essence, it is an automatic delegate. So, the code can simply be written as:

```
myButton.addEventListener("click", onClick);
```

- **Note that dereferencing a null or undefined reference will now throw an exception.** Dereferencing null or undefined in legacy ActionScript was ignored and evaluated to undefined. Now, a TypeError exception will be thrown. Watch out for code that casually dereferenced null or undefined, and depended on the silent failure behavior. The new exception-throwing behavior is compliant with the ECMAScript specification.
- **Use the -verbose-stacktraces and -debug options.** Compiling with the command-line options -verbose-stacktraces and -debug causes filenames and line numbers to appear in the Flash Player runtime alerts. When a runtime error occurs, a dialog box describes the error and lists the call stack where it occurred. Using the -verbose-stacktraces and -debug options can make it easier to locate the source of an error in your code.
- **Explicitly declare properties to be bindable.** Properties are no longer bindable by default. You must declare them to be bindable by using the [Bindable] metadata tag.
- **Note that the Flash Player API has been reorganized into packages.** Formerly all classes and functions in the Flash Player API were global. Now there are many packages such as flash.display, flash.events, flash.ui, and so on. For example, MovieClip is now flash.display.MovieClip and getTimer and setInterval have been moved to the flash.utils package.
- **Use the new Timer class instead of setInterval/setTimeout.** The new Timer class

provides a cleaner mechanism for timer events than the `setInterval` and `setTimeout` functions. The new `Timer` class has a number of advantages over `setInterval`, such as not having to deal with interval ID numbers, and a more modern, object-oriented interface. We regard using `Timer` instead of `setInterval` and `setTimeout` as a best practice.

- **Be sure to subclass events.** Events are now strongly typed, and must be subclasses of the new `Event` base class. The new `Event` class makes the event system clearer and more efficient. However, this also means that you can no longer use a generic instance of class `Object` when dispatching events, and you cannot use the object literal shorthand—for example, `{type: 'customEvent' }`.

  Instead of creating a generic `Object` class, you now need to use the `Event` class (for example, `dispatchEvent(new Event ('myCustomEventType'))`). You need to subclass `Event` if you want to pass additional properties. The motivation for not using `Object` is to achieve greater type safety and efficiency.

- **Note that visual elements must extend `DisplayObject`, and you can define them like any other class.** Components are now created dynamically with new and added to the display list using `addChild`. As a result, `createChild` has been deprecated. Visual entities, including `TextField`, can be instantiated like any other object and simply added to a display list using `addChild` or `addChildAt`. Note that this means certain APIs are gone, such as `createEmptyMovieClip` and `createTextField`. In order to create a `new TextField` you use `new TextField` instead of `createTextField`.
- **Note that E4X (ECMAScript for XML) is now the recommended means of manipulating XML in Flash.** E4X is far more powerful and better integrated into the language than the legacy Flash XML class, and provides a host of new capabilities. The legacy Flash XML class is still available for use. If you prefer the legacy XML API, renamed to XMLDocument, it is still available in the flash.xml package.
- **Use the `toXMLString` method when using E4X.** The `toString` method does not necessarily return the complete XML markup for the object; to get that, use the `toXMLString` method. The `toString` method returns a convenient string value for the XML object. It does not necessarily serialize the XML object in its entirety. To get the XML markup, call the `toXMLString` method.
- **Note that the `for...in` loop will no longer enumerate properties and methods declared by a class.** It only enumerates dynamic properties of an object. ActionScript 3.0 features a new and more advanced mechanism for object introspection, called `describeType`. Use it to introspect objects in ActionScript 3.0.
- **Note that the root object of a SWF file can now be an instance of a custom class of your choice.** In ActionScript 2.0, the root object of a SWF file was always of class `MovieClip`. In ActionScript 3.0, it may be any subclass of Sprite. You can set a class definition to be the `DocumentRoot` of a SWF file. When it's loaded, the SWF file will instantiate that class to serve as its root object.

Special thanks go out to our developer community for helping suggest entries for this article. The list presented here is by no means exhaustive, but it's a start that will help you hit the ground running with ActionScript 3.0. If you are familiar with other object-oriented languages, you may find these tips little more than a refresher—skills you've learned elsewhere can immediately be put to use in ActionScript 3.0.

If you are new to object-oriented programming and ActionScript 3.0, then these tips will come in handy. For you, this is definitely a list to pin up in plain sight. Happy coding!