

Using conditional compilation

The `mxmcl` compiler lets you pass the values of constants to the application at compile time. To do this, you use the `compiler.define` compiler option. The constant can be a Boolean, String, or Number, or an expression that can be evaluated at compile time. This constant is then accessible within the application as a global constant. A common use of passing constants is conditional compilation, where a Boolean is used to include or exclude blocks of code such as debugging or instrumentation code.

To use the `define` option, you define a configuration namespace for the constant, a variable name, and a value using the following syntax:

```
-define=NAMESPACE::variable_name,"value"
```

The following example defines the constant `debugging` with a value of `true` in the `CONFIG` namespace:

```
-define=CONFIG::debugging,"true"
```

You use the `+=` operator rather than the `=` operator to append definitions on the command line to the definitions set in the configuration file. Use the `=` operator to replace the definitions in the configuration file with the definitions on the command line.

The following example conditionalizes a block of code by using an inline constant:

```
CONFIG::debugging {
    // Execute debugging code here.
}
```

To set the values of multiple constants on the command-line, use the `define` option more than once; for example:

```
mxmcl -define=CONFIG::debugging,"true" -define=CONFIG::release,"false" MyApp.mxml
```

If you have multiple definitions on one command line, you can use `=` rather than `+=`. If you use a single `+=`, the compiler will also use the definitions in the configuration file.

To set the value of these constants in a configuration file, rather than on the command line, you write this as the following:

```
<compiler>
  <define>
    <name>CONFIG::debugging</name>
    <value>true</value>
  </define>
  <define>
    <name>CONFIG::release</name>
    <value>false</value>
  </define>
</compiler>
```

In a Flex Ant task, you can set constants with a `define` element, as the following example shows:

```
<mxmcl ... >
  <define name="CONFIG::debugging" value="true"/>
  <define name="CONFIG::release" value="false"/>
</mxmcl>
```

Using inline constants

You can use inline constants in `ActionScript`. Constant Booleans are commonly used to conditionalize top-level definitions of functions, classes, and variables, in much the same way you would use an `#IFDEF` preprocessor command in `C` or `C++`. You cannot use it to conditionalize metadata or `import` statements.

The following example conditionalizes which class definition the compiler uses when compiling the application:

```
// compilers/MyButton.as
package {
    import mx.controls.Button;

    CONFIG::debugging
    public class MyButton extends Button {
        public function MyButton() {
            super();
            // Set the label text to blue.
           .setStyle("color", 0x0000FF);
        }
    }

    CONFIG::release
    public class MyButton extends Button {
        public function MyButton() {
            super();
            // Set the label text to red.
           .setStyle("color", 0xFF0000);
        }
    }
}
```

Strings and Numbers can also be passed to the application and used as inline constants, in the same way you might use a `#define` directive in C or C++. For example, if you pass a value named `NAMES::Company`, you replace it with a constant in your application by using ActionScript like the following:

```
private static const companyName:String = NAMES::Company;
```

Passing expressions

You can pass expressions that can be evaluated at compile time as the value of the constant. The following example evaluates to false:

```
-define+=CONFIG::myConst,"1 > 2"
```

The following example evaluates to 3:

```
-define+=CONFIG::myConst,"4 - 1"
```

Expressions can contain constants and other configuration values; for example:

```
-define+=CONFIG::bool2,false -define+=CONFIG::and1,"CONFIG::bool2 && false"
```

In general, you should wrap all constants with double quotes, so that the mxmmlc compiler correctly parses them as a single argument.

Passing Strings

When passing Strings, you must add extra quotes to ensure that the compiler parses them correctly.

To define Strings on the command-line, you must surround them with double-quotes, *and* either escape-quote them (`"\"Adobe Systems\""` or `"\'Adobe Systems\'"`) or single-quote them (`"'Adobe Systems'"`).

The following example shows both methods of including Strings on the command line:

```
-define+=NAMES::Company,"'Adobe Systems'" -define+=NAMES::Ticker,"\"ADBE\""
```

To define Strings in configuration files, you must surround them with single or double quotes; for example:

```
<define>
    <name>NAMES::Company</name>
    <value>'Adobe Systems'</value>
</define>
```

```
<define>
  <name>NAMES::Ticker</name>
  <value>"ADBE"</value>
</define>
```

To pass empty Strings on the command line, use single quotes surrounded by double quotes, as the following example shows:

```
-define+=CONFIG::debugging, ""
```

To pass empty Strings in configuration files, use double quotes (") or single quotes (').