Simple Component Writer's Guide

Note that most of the following also applies to writing ordinary libraries for Simple.
The preferred language to write Simple components is Java, although it should be possible to write them in any language that can generate Java class files and has support for Java annotations.

The Basics

Type Mapping from Java to Simple

In order to write Simple components in Java it is necessary to map Java types to Simple types and vice versa. The following table shows the equivalent types:

Simple Type	**Java Type**
Boolean	boolean
Byte	byte
Short	short
Integer	int
Long	long
Single	float
Double	double
String	java.lang.String
Date	java.util.Calendar
Object	java.lang.Object
Variant	com.google.devtools.simple.runtime.variants.Variant

Any other Simple object types are mapped to the same name in Java.
Simple function arguments can be either passed by value or by reference. If they are passed by reference than the argument type is mapped as follows:

Simple Type	**Java Type**
Boolean	com.google.devtools.simple.runtime.parameters.BooleanReferenceParameter
Byte	com.google.devtools.simple.runtime.parameters.ByteReferenceParameter
Short	com.google.devtools.simple.runtime.parameters.ShortReferenceParameter
Integer	com.google.devtools.simple.runtime.parameters.IntegerReferenceParameter
Long	com.google.devtools.simple.runtime.parameters.LongReferenceParameter
Single	com.google.devtools.simple.runtime.parameters.SingleReferenceParameter
Double	com.google.devtools.simple.runtime.parameters.DoubleReferenceParameter

String	com.google.devtools.simple.runtime.parameters.StringReferenceParameter
Date	com.google.devtools.simple.runtime.parameters.DateReferenceParameter
Object	com.google.devtools.simple.runtime.parameters.ObjectReferenceParameter
Variant	com.google.devtools.simple.runtime.parameters.VariantReferenceParameter

Any other Simple object reference types are mapped to com.google.devtools.simple.runtime.parameters.ObjectReferenceParameter.

Simple Annotations

The Simple language does not support all the features of Java (e.g. no method overloading). It was designed to be simple (hence the name). On the other hand it does add first-class support for features such as events and properties that are not available as part of the Java language specification. As a consequence of this design, the content of Java class files needs to be annotated, to be recognized by the Simple compiler. The following annotations (from the com.google.devtools.simple.runtime.annotations package) are used to mark Simple recognized content in Java class files.

SimpleObject

Adding this annotation to a Java class or interface declaration makes that class visible from within Simple code.

```
@SimpleObject
public final class Strings {
  // ...
}
```

SimpleFunction

Adding this annotation to a Java method makes that method visible from within Simple code. This requires the class enclosing the marked method to be marked with the SimpleObject annotation.

```
@SimpleObject
public final class Strings {
  /**
   * Converts the given string to all lowercase.
   *
   * @param str  string to convert to lowercase
   */
  @SimpleFunction
  public static void LCase(ObjectReferenceParameter<String> str) {
    str.set(str.get().toLowerCase());
  }

  // ...
}
```

SimpleProperty

Adding this annotation to a non-static Java method defines that method for use as a Simple property. This requires the class enclosing the marked method to be marked with the SimpleObject annotation.
Setter methods must be declared void and have exactly one formal argument, whose type will determine the type of the property. Getters must not have any formal arguments and a return type which will be the type of the property. If both a getter and a setter method are marked then the getter return type and the setter formal argument type must be identical.

This annotation takes two optional parameters. The first, `type`, is a more specialized type than the property's runtime type. It is intended for use by IDEs. By default the type is assumed to be plain text. The second, `initializer`, is the default value for the property given as a string. The annotions parameters should only be used on the property setter method. They will be ignored if found on the property getter method.

type	**Description**
`PROPERTY_TYPE_ASSET`	Resource name as stored in the asset directory (string with quotes, e.g. the default value is \"\")
`PROPERTY_TYPE_BOOLEAN`	True or False
`PROPERTY_TYPE_COLOR`	Color constant values as defined in com.google.devtools.simple.runtime.components.Component
`PROPERTY_TYPE_DOUBLE`	Floating point constant (double precision)
`PROPERTY_TYPE_HORIZONTAL_ALIGNMENT`	Horizontal alignment constant values as defined in com.google.devtools.simple.runtime.components.Component
`PROPERTY_TYPE_INTEGER`	Integer constant (8-bit, 16-bit or 32-bit signed integer)
`PROPERTY_TYPE_TEXTJUSTIFICATION`	Text justification constant values as defined in com.google.devtools.simple.runtime.components.Component
`PROPERTY_TYPE_LAYOUT`	Layout constant values as defined in com.google.devtools.simple.runtime.components.Component
`PROPERTY_TYPE_LONG`	Integer constant (64-bit signed integer)
`PROPERTY_TYPE_SINGLE`	Floating point constant (single precision)
`PROPERTY_TYPE_STRING`	String constant (with quotes, e.g. the default value is \"\")
`PROPERTY_TYPE_TEXT`	Plain text
`PROPERTY_TYPE_TYPEFACE`	Typeface constant values as defined in com.google.devtools.simple.runtime.components.Component
`PROPERTY_TYPE_VERTICAL_ALIGNMENT`	Vertical alignment constant values as defined in com.google.devtools.simple.runtime.components.Component

```
@DesignerComponent
@SimpleObject
public interface Label extends VisibleComponent {
  /**
   * Hint property getter method.
   *
   * @return  hint text
   */
  @SimpleProperty
  String Hint();

  /**
   * Hint property setter method.
   *
   * @param hint  hint text
   */
  @SimpleProperty(type = SimpleProperty.PROPERTY_TYPE_STRING,
                  initializer = "\"\"")
  void Hint(String hint);

  // ...
}
```

SimpleEvent

Adding this annotation to a Java method marks that methods to be a Simple event definition. This requires the class enclosing the marked method to be marked with the SimpleObject annotation.
Events cannot have any return types.

```
@SimpleObject
public interface Label extends VisibleComponent {
  /**
   * Default Initialize event handler.
   */
  @DesignerEvent
  @SimpleEvent
  void Initialize();

  // ...
}
```

SimpleDataElement

Adding this annotation to a Java field declaration makes that field visible from within Simple code. This requires the class enclosing the marked field to be marked with the SimpleObject annotation.
Static final fields of Java primitive and String types are considered to be constants.

```
@SimpleObject
public interface Component {
  /*
   * Text alignment constants.
   */
  @SimpleDataElement
  static final int ALIGNMENT_NORMAL = 0;
  @SimpleDataElement
  static final int ALIGNMENT_CENTER = 1;
  @SimpleDataElement
  static final int ALIGNMENT_OPPOSITE = 2;

  // ...
}
```

SimpleComponent

Declares a class to be a component. This annotation is used by the Simple compiler to identify components. It also be used by an IDE to identify components. The class must also be marked with the SimpleObject annotation.

```
@SimpleComponent
@SimpleObject
public interface Label extends VisibleComponent {
  // ...
}
```

Android Specific Annotations

The UsesPermissions annotation is used to set Android permissions for granting access to certain data and services on Android devices. The annotation takes a single parameter. For information on values for the parameter see http://developer.android.com/reference/android/Manifest.permission.html.
The class must also be marked with the SimpleObject annotation.

```
@SimpleObject
@UsesPermissions(permissionNames = "android.permission.CALL_PHONE")
public final class Phone {
  // ...
}
```

Visible and Non-visible Components

We can distinguish between two different categories of components: visible and non-visible. Visible components have a visual representation on a Form. They are subclasses of com.google.devtools.simple.runtime.components.VisibleComponent. Examples of visible components are com.google.devtools.simple.runtime.components.Button and com.google.devtools.simple.runtime.components.Panel.
Non-visible component don't have a visible representation. They simply need to extend the com.google.devtools.simple.runtime.components.Component interface. Examples of non-visible components are com.google.devtools.simple.runtime.components.Timer and com.google.devtools.simple.runtime.components.OrientationSensor.

Component Implementation

A good recipe for creating a new component is to look for an existing similar component and modify its implementation. The standard components coming with the Simple distribution are located in the com.google.devtools.simple.runtime.components and com.google.devtools.simple.runtime.components.android packages.

A component must define an interface and an implementation of that interface. This separation allows for easy future implementation of components for different operating systems or architectures. As mentioned before, visible components must extends the com.google.devtools.simple.runtime.components.VisibleComponent interface, while must non-visible components must extend the com.google.devtools.simple.runtime.components.Component interface.

Component constructors, for both visible and non-visible components, must take com.google.devtools.simple.runtime.components.ComponentContainer as their only parameter. For non-visible components the value of the container argument is the Form.

Component properties will be initialized implicitly by the compiler and runtime system based upon the initializer parameter of SimpleProperty annotations of property setter methods. An empty string assigned to initializer (or omiting initializer entirely in the annotation) will suppress explicit initialization.

Although Simple doesn't provide any thread support, components should be written to be thread-safe.