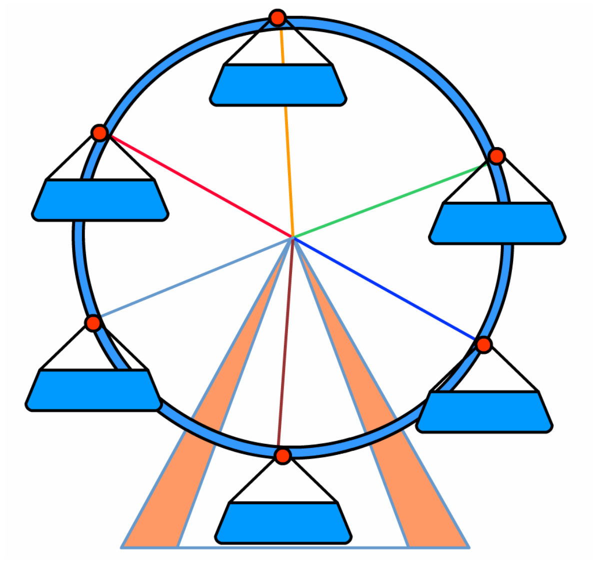

c.def

(pronounced SEE-def)

Macromedia® Flash™ animation language

Whitepaper

Dennis Rakhamimov (dr524@columbia.edu), Group Leader
Eric Poirier (edp29@columbia.edu)
Charles Catanach (cnc26@columbia.edu)
Tecuan Flores (tf180@columbia.edu)

Purpose

In the recent years, Macromedia® Flash™ has gained a prominent position in the web development sector, as the platform is powerful, the learning curve is manageable, and the support is excellent. However, the Flash GUI authoring environment poses significant challenges in creating relatively complex animations containing interdependent objects with differing actions. Short of manually creating an animation frame-by-frame, the GUI enables a Flash artist to utilize “tweening”, a process that automatically generates intermediary frames between a start frame and an end frame. The animations created by tweening are generally limited to simple scaling, translation, or color fade operations. While it is possible for an object to follow a pre-defined path in a translation operation, the complexity of the path is limited by a set of non-overlapping curves. Creating a number of distinct objects that follow the same type of animation could be tedious, forcing the Flash artist to spend considerable time setting up the right amount of layers and frames, copy-and-paste and adjust each object one-by-one, zoom in to precisely position elements, and perform other mundane tasks. As a bottom line, while Flash provides an excellent platform for effective web presentations, it hinders the ability to create complex scenes and animations. Although Flash includes a scripting language called ActionScript that somewhat facilitates a process such as the one described, it is limited in ability, still constraining the Flash artist to using the Flash GUI for creating the graphical components.

The web and graphic developer community can thus greatly benefit from creation of a language that enables the programmatic creation of Flash animations. Such language could then be leveraged by a Flash artist to create complex Flash animations without having to utilize the cumbersome Flash GUI.

Proposed Solution

The solution: **c.def**™, with the name created from a combination of our initials.

This language will enable a developer to algorithmically compose a Flash movie containing animations with scaling, layering, and translation of a complex group of objects. It will logically organize elements in an object-oriented structure, forming a level of abstraction above the Flash layer and timeline components, thus allowing the creation of fully-fledged Flash animation with less effort. Since the code will feature typical language elements such as iteration and arrays, it will also be possible to create a number of objects with similar properties using loops. Other language elements such as conditions and functions would also be present, allowing for sophisticated logic. While the language would permit access to Flash tweening, its main method of animation will automatically create frame-by-frame motion, thus permitting for movement that is generally not supported by tweening. For example, a developer would be able to create an object with a complex shape and set of colors, create a curve in the shape of an oval, and instruct the object to follow that shape over a period of 20 frames.

In syntax, **c.def**™ will resemble a modern language such as Java. A program written in it will be compiled to Java code, which in turn will leverage Flagstone Software’s *Transform SWF* [1] package to create a Flash SWF file. *Transform SWF* is a set of Java libraries that provide programmatic access to elements of the open SWF file format, thus creating an intermediary between low-level implementation details and the actual Flash components.

To judge the success of the basic functionality of our project, we will write programs that create SWF animations of Ferris wheels. The choice of the Ferris wheel, suggested by Professor Edwards, comes from the fact that this animation would feature motion along a guideline, rotation of a complex object, as well as a number of similar objects with slightly different properties. For instance, while each swing on the Ferris wheel must follow the same trajectory, the start positions and colors would probably be different. We created a sample animation in Flash to test out this hypothesis [2], and while the Flash GUI did allow creation of a simple Ferris wheel, the animation was cumbersome to make and required much copy-and-pasting, layer adjustments, and other time-consuming tasks. **c.def**™ will make the task of creating such an animation a breeze.

In summary, our goal is thus to create a well-organized and concise language that streamlines the creation of compound scenes and animations in Flash. We will use the ability to create an SWF file with an animated Ferris wheel to evaluate the success of our project [3].

Goals

Our project aims to accomplish the following goals at a minimum. Additional features may be added depending on time constraints.

- **Object-oriented design:** c.def™ will feature built-in support for these objects: Point, Line, Rectangle, Circle, SEdwards, Polygon, while also allowing user-defined, custom objects through inheritance.
- **Library/package support:** To facilitate the development of complex Flash animation, our language will provide support for reusable packages.
- **Functional capability:** As c.def™ lends itself to complex project development, in order to achieve the desired flexibility, the reuse of sectors of code is pertinent.
- **Iteration, conditionals:** The flow of a c.def™ program can be dependent on conditional statements and/or iterative loops.
- **Shape and motion tweening, motion guides:** c.def™ will enable the developer to programmatically create frame-by-frame animation, instead of relying on the limited Flash tweening capabilities. However, it will also permit the use of tweening for added functionality.
- **Layering:** c.def™ will allow for creation of layers in a Flash movie. In case of tweened motion, it will automatically create the necessary layers.
- **Compile to SWF:**
  - Compile c.def™program to Java code
  - Java code utilizes Flagstone *Transform SWF* library for SWF file output
  - Compile and run the Java code to generate SWF file

Structure

Just as Java provides the base class 'Object', which all non-primitive data-types extend, we will define a base class 'Symbol' as the base class for all graphical objects. This will allow the user to set certain properties of a graphic object in a uniform way. Inheritors of this class will either be graphical shapes, such as circles or rectangles, or consist of object groups. Some necessary properties of the Symbol object include:

- **Frame**: frame containing the object
- **Parent-child relationship:** Link to parent, first child, and next sibling
- **Position**: x and y coordinates

Symbol objects containing graphical shapes would have additional properties, such as these:

- **Color**: Stroke and Fill
- **Stroke Width**
- **Member Functions**: to get/set each of these private data members (i.e. circle.getStrokeWidth())

The language will also contain a series of classes for defining the Flash document, such as creating the scenes and adding frame actions.

Our language will inherit some functionality from Java for access to some language primitives, such as integers. It will thus be possible to also perform various basic operations such as addition and subtraction in addition to leveraging Flash functionality.

Code Sample

Following is a code sample of a simple program written in our language. As demonstrated here, the code follows most syntax conventions of a modern language such as Java.

***Source file**: demo.cdef*

```
/* Create a scene with the width of 500 pixels and the height of 400 pixels.
   A scene in a Flash file is the main container of all objects. A Flash document
   can have multiple scenes */
Scene scene = new Scene(500, 400);
scene.BackgroundColor = Color.White;

/* A scene can have multiple layers. Each layer has a timeline that holds objects
   or animations placed at various frames. For example, the objects can appear
   or move as the movie progresses. This layer will have up to 200
   frames. */
Layer layer1 = new Layer(scene, 200);

/* Create another layer */
Layer layer2 = new Layer(scene, 100);

/* Arrays can hold objects and values as their elements. Here we define an
   array that will hold circles. */
Circle[] circArray = new Circle(6);
```

```
/* Create some more keyframes, using iteration */
for(int i = 1; i <= 6; i++)
{
   circArray[i] = new Circle(30); // Radius = 30
   circArray[i].setType(Symbol.Visual);
   circArray[i].setPosition(i * 10, i * 10); /* Left = 10, 20, ...
                                             Top  = 10, 20, ... */
   circArray[i].setStrokeColor(Color.Black);
   circArray[i].setFillColor(Color.Gray);

   /* Place all of these circles onto frame 0 */
   circArray[i].placeOnFrame(0, layer1);
}

/* Set some distinctive properties for these circles */
circArray[0].setStrokeColor(Color.Red);
circArray[1].setStrokeColor(Color.Blue);
circArray[2].setStrokeColor(Color.Green);

/* Create a new object group, add two circles to it, translate it, and place
   it into the animation */
Group group = new Group();
group.add(circArray[1]);
group.add(circArray[3]);
group.setPosition(50, 30);
group.placeOnFrame(5, layer2);

/* Create a new line, going from (0, 0) to (100, 30)
Line line1 = new Line(0, 0, 100, 30);
line1.setType(Symbol.MotionGuide);

/* Create 30 frames of animation, moving one of our circles over the path
   of the invisible line. The animation will start on frame 1 and end on frame 30 */
circArray[0].animateOverGuide(line1, 1, 30);
```

If the compilation of this code had finished successfully, demo.swf would now be in the current working directory. It could then be viewed in a variety of Flash viewers, such as one installed in Internet Explorer.

References

[1] Flagstone Software Transform SWF Package
http://www.flagstonesoftware.com/transform/overview.html

[2] Simple Ferris wheel animation created in Macromedia® Flash™
http://www.columbia.edu/~dennisr/PLT/ferriswheel.html

[3] Photo of Dennis in front of a real-world Ferris wheel, for inspiration
http://www.columbia.edu/~dennisr/PLT/ferriswheel.jpg