c.def

(pronounced SEE-def)

Macromedia® Flash™ animation language

Final Report

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

Table of Contents

1 Introduction

1.1 Business Level

a. 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.

b. 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 interpreted 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].

1.2 Language Level

a. Goals

The goal of **c.def**™ is to create a well-organized, concise, and easy-to-use language that streamlines the creation of compound scenes and animations in **Macromedia® Flash™**.

A **c.def™** program defines one or more drawing templates, called `Glyph`s, and one or more `Path`s, and then proceeds by rendering the created templates. Each of these elements can be transformed using scaling, rotation, or translation. The language supports control flow logic such as loops and conditions, and is capable of evaluating expressions using basic mathematic operators. The program author can thus conveniently use the program to create several drawing templates, and place them throughout the animation or have them move along a motion path, all with relatively little amount of code. **c.def™** interpreter handles the task of creating Flash symbols, layers, and frames corresponding to the program code.

Since **c.def™** enables creation of Flash animation without using the proprietary Flash GUI, it thus offers an easy and free solution to creating effective graphics. A program written in **c.def™** will be interpreted to Java code, which in turn will leverage Flagstone Software's `Transform SWF` 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. The Java file can then be run to generate the actual SWF file containing the animation.

The resulting SWF file could be viewed in a **Macromedia® Flash™** viewer, or embedded into a webpage using HTML code. In the latter case, users with the Flash plug-in would be able to view the Flash animation in their favorite web browser.

b. Approach

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: Line, Rectangle, Circle, Polygon. Objects can be inserted into groups called Glyphs, which in turn could be nested to create further complex objects.

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. This is achieved through object-oriented design.

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

Interpret **c.def™** program to Java code, utilizing Flagstone `Transform SWF` library for SWF file output.

2 Language Tutorial

2.1 Program Flow

Each **c.def™** program first defines a block called Document. For example,

```
Document myFlash [ &(400, 300), #Blue, 180 ]
{
     statements
}
```

The above statement had defined a Flash movie with a width of 400 pixels, height of 300 pixels, and blue as the background color. The length of the movie, in frames, is indicated by the third argument.

The `Document` block can then contain declarations for `Glyph` objects, `Paths`, and control flow and iteration statements such as `for` loops.

Each `Glyph` object specifies a template consisting of drawing primitives. It can then be rendered into the Flash animation using the `Render` command, or transformed the `Rotate` command.

Following is an example of a `Glyph` declaration that creates a template with a circle superimposed on top of a square.

```
Glyph g [&(center_x, center_y) ]
{
      color[#Red]; // set the stroke color
      rect[&(-10, -10), &(20, 20) ];
      circle[&(0, 0), 50 ];
}
```

The `Glyph` can now be rendered into the Flash movie. It can be placed either on a particular frame, or animated over a motion guide defined by a `Path`. Let's create a simple linear path.

```
Path p [0]
{
      line[&(0, 0), &(100, 100) ];
}
```

This declaration specifies that the origin of the motion is the point (0, 0). The 0 in the `Path` declaration indicates that we would like to start iterating in the beginning (0%) of the path.

Now we can Render the `Glyph`.

```
Render [ g, ->(1, 30), p ];
```

This command will then create 30 frames in the Flash movie to translate the glyph over the path.

Voila! The **c.def™** source file can now be interpreted and thus converted to a SWF animation.

2.2 Nested Glyphs

Since a primary goal of **c.def™** is to manage multiple instances of complex objects, it is possible to nest `Glyph` objects inside each other. Each `Glyph` can be referenced by an `ID`, which makes nesting possible. The following example shows a `Glyph` "spike" that is rendered multiple times inside a ferris wheel:

```
/* A wheel's spike */
Glyph spike[&(0, 0)]
{
   line [&(20, 0), &(120, 0)];
   line [&(120, 0), &(58, 105)];
}

/* Define the wheel */
Glyph wheel[&(150, 150)]
{
   for[ i: ->( 1, 6 ) ]
   {
      Rotate [spike, 60, &(0, 0)];

     /* Place it onto the wheel */
      spike [&(150, 150)];
   }
}
```

2.3 Rotation

Glyph objects may be rotated by a specified number of degrees in space. The following example shows a `Glyph` "spike" that is `Rotate`s, forming multiple radial lines emanating from the center of the ferris wheel:

```
/* A wheel's spike */
Glyph spike[&(0, 0)]
{
   line [&(20, 0), &(120, 0)];
   line [&(120, 0), &(58, 105)];
}

/* Define the wheel */
Glyph wheel[&(150, 150)]
{
   for[ i: ->( 1, 6 ) ]
   {
      Rotate [spike, 60];

     /* Place it onto the wheel */
      spike [&(150, 150)];
   }
}
```

2.4 Translation

c.def™ `Glyph` objects can be translated about a `Path`, which can be linear or circular. The following example shows a `Glyph` "ferrisCar" being rotated around the circular `Path`, which is the ferris wheel:

```
for[ i : ->(1, 6) ]
{
     Path circularPath[i * (100/6)]
     {
          circle[&(200, 200), 120];
     }

     if[ i % 2 == 0 ]
     {
          Render [ferrisCar, ->(1, 180), circularPath];
     }
     else
     {
          Render [ferrisCar2, ->(1, 180), circularPath];
     }

}
```

Note that the arithmetic expressions are computed internally with double precision, only rounding to integers when producing pixel coordinates.

3 Language Reference Manual

3.1 Lexical Conventions

a. Keywords

The **c.def™** language consists of the following keywords. These names are reserved and thus cannot be used for identifiers.

```
Document      For         rect
Glyph         if          circle
Path          else        ellipse
              continue    line
Render        break       polygon
Rotate
Translate     color
              fillcolor
```

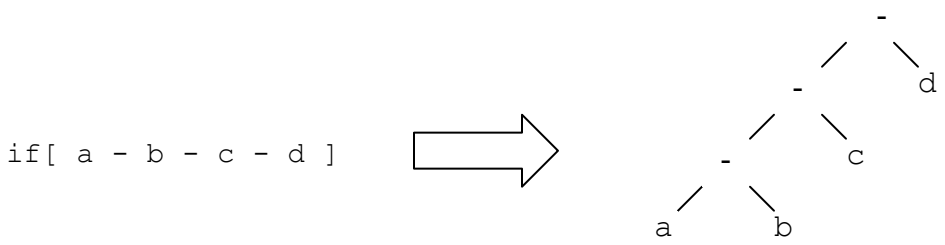
b. Arithmetic and Comparison Operators

c.def™ supports all arithmetic and comparison operators, similar to Java or C. Computations are internally performed with double precision as floating-point numbers, and are converted to integer values when producing pixel values at last stage. Following is the table of operators, listed from highest precedence to lowest.

Operator
()
*, /, %
+, -
==, !=, >, <, >=, <=
!
&&, ||

If two operators have the same precedence are used adjacently, the operator that is parsed first has higher precedence. This means that the all of these operators are *left-associative*.

For example, the following code would internally produce the following syntax tree:



If an operation evaluates a non-zero value, it is considered to be logically true. Otherwise, it evaluates to `0` and is logically false. Thus, an arithmetic expression such as `x` or `n * (z – 5)` can be used as a logical condition.

c. Block Operators

Code in **c.def™** is delimited by block operators. They are as follows:

```
{} /* defines the bounds of a scope */
() /* encloses the parameters for an expression */
[] /* encloses the arguments for a function call
      or an object declaration */
;  /* terminates a statement */
```

d. Color Identifiers

The pound (#) operator is used to identify colors. There are two ways for defining a color.

Examples:

#colorname
#(r, g, b)

```
#Blue
#Green
#(100, 0, 0)
```

e. Coordinate Identifiers

The ampersand (&) operator is used to identify coordinates. Coordinates are defined by an x- and y- value, indicating their Cartesian position on the Document relative to the center of the containing object. The center thus represents the (0, 0) coordinate.

Examples:

```
&(x, y)
&(30, 70)
&(50,90)
```

f. Range Identifiers

The arrow (->) operator is used to identify ranges of integers. The *i* and *j* values indicate the start- and end-position of the range, respectively. The optional *k* value represents the increment that should be used when iterating over the range.

Note that if *i* < *k*, *j* should be a positive value (or can be omitted if desired). If *i* > *k*, *j* should be a negative value.

Examples:

```
->(from, to)
->(from, to, step)
->(1, 10)
->(1, 360, 30)
```

For uses of range identifiers, please see sections on `for` loops and `Render` statements.

g. Case Sensitivity

c.def™ is a case-sensitive language. For example, `Glyph` and `glyph` represent different identifiers.

h. Scoping

The language uses lexical scoping, with each level of scope being defined by curly braces. An identifier defined inside a block will go out of scope and thus be discarded when the block ends. If an identifier inside a block has the same name as an identifier in a parent block, this new identifier will temporarily hide the other one until the block ends.

i. Comments

c.def™ uses Java-style comments. There are two formats for comments:

Block Comments

```
/*
     This section of code is commented out.
     This section of code is commented out.
     This section of code is commented out.
*/
```

Single-line Comments

```
//   This single line of code is commented out.
```

3.2 Data Types and Attributes

a. Fundamental Objects

Each **c.def**™ document is comprised of a single `Document` object, and a collection of `Glyph` and `Path` objects.

Document

The `Document` object serves as the container for `Glyph` and `Path` objects and control flow commands such as `if` and `for`.

Example:

```
Document d[&(width, height), #color, numFrames]
{
     statements
     ...
}
```

Where *width* and *height* specify the size of the Flash movie, and *color* specifies the background color. The *numFrames* indicates the length of the movie, in frames.

Glyph

A `Glyph` object is composed of a collection of primitive objects and can be rendered and moved about a `Path` object as a single entity. Since a `Glyph` can be rendered multiple times along a point or a `Path` on a `Document`, it can be considered a form of a template.

Example:

```
Glyph g[center]
{
   statements
   ...
}
```

The statements specified become part of the template, with the more recent statements specifying graphics that will superimpose the previous graphics. Control flow statements also can be used within Glyph definitions. The *center* parameter of the Glyph indicates the relative center, to be applicable when this Glyph is rendered or rotated.

Path

A Path object is the motion layer on which Glyph objects can traverse. The object will start the path at the given percentage relative to the specified reference point, and continue until the end of path is reached (for non-looping paths) or it has reached the starting point (for looping paths). A single drawing statement is specified inside the Path block to indicate the path to take.

Example:

```
Path p[percentage]
{
   statement
}
```

b. Identifiers

Fundamental objects in the language are referenced by unique identifiers. Identifiers must start with a character and can contain characters, numbers, or underscores.

Examples:

```
Glyph c1 ...
Glyph myCircle ...
Path line ...
Path another_line ...
Document hello123 ...
```

c. Primitive Objects

Primitive objects can be used inside Glyph and Path objects to construct the particular shapes. Their coordinates are defined relative to the containing object, with (0, 0) coordinate being the center of that container. Note that the order of primitive objects matters, as the more recent objects will be rendered above the older ones.

line

A line primitive constructs a line that spans between the points that are specified by the `&(`x_1`, `y_1`)` and `&(`x_2`, `y_2`)` arguments.

Examples:

```
line[&(x1, y1), &(x2, y2)];
line[&(10, 20), &(40, 50)];
```

circle

A circle primitive constructs a circle with radius `rad` and is centered at the point specified by coordinate `&(x, y)`.

Examples:

```
circle[&(x, y), rad];
circle[&(0, 10), 120];
```

rect

The `rect` primitive constructs a rectangle with the lower-left corner at (x_1, y_1) and upper-right corner at (x_2, y_2).

Examples:

```
rect[&(x1, y1), &(x2, y2)];
rect[&(10, 10), &(100, 130)];
```

ellipse

The `ellipse` primitive constructs an ellipse with specified vertical and horizontal radii, and is centered at the point specified by coordinate `&(x, y)`.

Example:

```
ellipse[&(x, y), radiusX, radiusY];
```

polygon

The `polygon` primitive constructs a polygon that is composed of n-points. The first set of points specifies the origin, while each $_i$ translation is specified by coordinates $\&(x_i, y_i)$, where i spans from 1 to n.

Example:

$$\text{polygon}[\&(x_1, y_1), \ldots, \&(x_n, y_n)];$$

d. Colors

Basic Colors

The following table identifies the 16 basic colors that can be specified as *#colorname*. The RGB column identifies the red, green, and blue composition that corresponds to each of the basic colors.

Color	Corresponding RGB Value
Aqua	(0,255,255)
Gray	(128,128,128)
Navy	(0,0,128)
Black	(0,0,0)
Green	(0,128,0)
Teal	(0,128,128)
Olive	(128,128,0)
Blue	(0,0,255)
Lime	(0,255,0)
White	(255,255,255)
Purple	(128,0,128)
Fuchsia	(255,0,255)
Maroon	(128,0,0)
Yellow	(255,255,0)
Silver	(192,192,192)

Red	(255,0,0)

Additionally, we defined a `#None` constant that is interpreted by Flash as an object with no fill.

Examples:

```
color[#Blue];
fillcolor[#Red];
fillcolor[#None];
```

RGB Colors

Advanced colors can be defined by their RGB value. To use a color that is not pre-defined, use the following syntax. Note that *r*, *g*, and *b* are integers between 0 and 255, specifying the RGB color composition.

```
fillcolor[#(r, g, b)];
```

Examples:

```
fillcolor[#(0, 0, 100)];
fillcolor[#(255, 0, 37)];
```

Color Precedence

A Glyph is composed from one or many primitive objects, each of which has a color that is specified in the definition of the Glyph.

3.3 Action Type

a. Render

The Render action only applies to the Glyph foundational object. It draws the Glyph on the Document, on a specified frame. The first construct illustrates this action, which simply draws Glyph g onto Frame frameNum. The Render action also has the flexibility to draw the object on multiple frames, using the second construct below. The second construct draws Glyph g from startFrame to endFrame, placing glyph along Path object p. The range parameter can optionally be specified with a step value if it is desired to render the object with a lesser frequency.

Note that more recent Render statement will render on top of the frames rendered by previous statements. Thus, that the

order in which Glyphs are rendered does matter in the resulting SWF movie.

Examples:

```
Render [g, frameNum, coords];
Render [g, ->(startFrame, endFrame), p, coords];
Render [g, ->(startFrame, endFrame, step), p, coords];
Render [ferrisCar, ->(1, 360), circularPath];
```

b. Rotate

The Rotate action can be used to rotate Glyph objects. The first argument for this action specifies the identifier for the Glyph object to be rotated. The degrees argument specifies the integer number of degrees by which to rotate the foundational object. The Glyph will be rotated around its respective center.

Example:

```
Rotate [g, degrees];
Rotate [p, degrees];
Rotate [mySquare, 45];
```

3.4 Control Flow

a. Conditional

Logical expressions can be given using comparison operators such as == and/or specified as arithmetic expressions. An expression which evaluates to a non-zero value will be considered *true*, and one evaluating to zero will be considered *false*. The operators that are supported in a logical statement are included in **Section 3.2**. Unlike Java or C, curly braces are *required* to denote the actions to be performed.

Example:

```
if[(x + y) % z == 0)] { ... }
```

An optional else clause may be specified to denote actions that should occur if the given condition is false.

Example:

```
if[ condition ]
{
     ...
}
```

```
else
{
      ...
}
```

b. Iterative

An iterative statement loops over a range of integers. The `break` keyword may be used to exit a loop if a specific condition occurs. Additionally, the `continue` keyword may be used to step out of a given iteration of the loop, and begin with the next iteration.

The specified `var` is implicitly declared as an `int` primitive, although all calculations are performed internally as Java's `double` primitive type. The `var` loses scope when the block ends. The second parameter specifies the range of iteration, with an optional step value.
Note that curly braces are required to denote the actions to be performed.

Example:

```
for[ var : -> ( fromValue, toValue) ]
{
      statements
      ...
}
```

3.5 Compilation and Execution

a. Build CDEF Language

The **c.def™** language relies on the two JAR files: `antlr.jar` and `transform.jar`. The `antlr.jar` file is necessary to build the Lexer, Parser and Tree Walker, and the `transform.jar` file is necessary to build the CDEF library. To build the CDEF language, simply type `make` within the directory in which the source code resides.

Example:

```
[MS_DOS]> make
Constructing the Lexer and Parser...
ANTLR Parser Generator   Version 2.7.2   1989-2003
jGuru.com
Constructing the Tree Walker...
ANTLR Parser Generator   Version 2.7.2   1989-2003
jGuru.com
CDEF Language Interpreter
Generating keyframe table... 3243 keyframes created.
```

```
Generating SWF file...

SUCCESS: Created output file 'test.swf'
All done.
```

b. Compile .cdef Program

c.def™ programs are interpreted to Java code. By running the CDEF interpreter on a **c.def™** source file, the corresponding SWF file is produced. The process can be performed using the following steps:

- Write a **c.def™** program.
- Run the **c.def™** interpreter to convert this program to Java code, by executing the following command, and automatically create the SWF movie:

  ```
  java -cp antlr.jar;transform.jar;. filename.cdef
  ```

- The file `filename.swf` will now be in the current working directory. It can be viewed with the Macromedia Flash viewer or embedded in a webpage as desired.

There is a file CDEF.bat that streamlines this procedure on the Windows platform. Simply type "`CDEF [filename.cdef]`", and then "`start [filename.swf]`" at the MS_DOS prompt.

4 Project Plan

4.1 Planning

a. Timeline

Planning and Research	
9/9	Existing FLASH Libraries
9/16	Language goals and Structure
9/20	Requirements
9/23	Whitepaper Due
Specifications	
10/14	Specifications / Core Functionality
10/17	Keywords
10/21	Conventions / Nomenclature
10/28	Language Reference Manual Due
Execution	
11/20	Back End Beta Testing, Front End Incremental Development
12/3	Ferris Wheel Demo Fully Functional
12/6	Code Review
12/10	In-Class Demo
12/17	Final Presentation

b. Role Matrix

	Dennis	Eric	Charles	Tecuan
FLASH Library Research	X	X	X	X
Whitepaper		X		X
Coding Conventions	X		x	X
Structure and Goals	X	X	x	X
LRM		X		X
Front End	X		X	X
Back End	X	X	X	
Regression Testing	X	x		X
Functional Testing	X	X		
Unit Testing	X		X	
In-Class PowerPoint Presentation		X		x
Final Report		X		x

4.2 Specification

a. Nomenclature

CDEF language files: `CDEF[file].java`
Back End files: `CDEF[Grammar/Walker].g`
CDEF source files: `[filename].cdef`
Resulting SWF files: `[filename].swf`

b. Programming Style

- Object-Oriented Programming
- Input/Output Factory
- Interfaces

4.3 Development

a. Environment

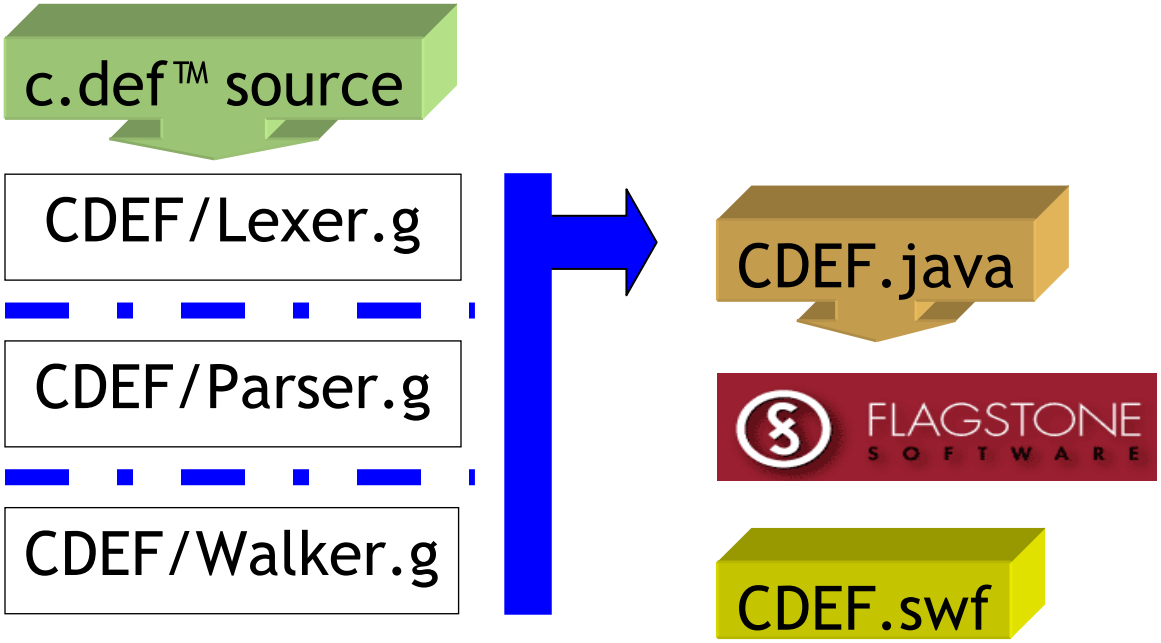
- Windows XP (for Flash Functionality)
- Macromedia Flash MX
- TextPad (Robust Java IDE)

b. Strategy

- Collaboration
- Extreme Programming Variant

5 Architecture

5.1 Diagram



5.2 Description

a. Back End

The back end is composed of the Lexer, Parser, and Tree Walker. It utilizes libraries from the `antlr.jar`. The Lexer code resides in the `CDEFGrammar.g` file which is compiled to `CDEFLexer.java` and `CDEFParser.java` by `ANTLR`. The Tree Walker code resides in `CDEFWalker.g` file which is utilizes `transform.jar` in order to create the key frames that Flagstone translates to valid SWF format. The Tree Walker interprets the **c.def™** source code into Java code.

b. Front End

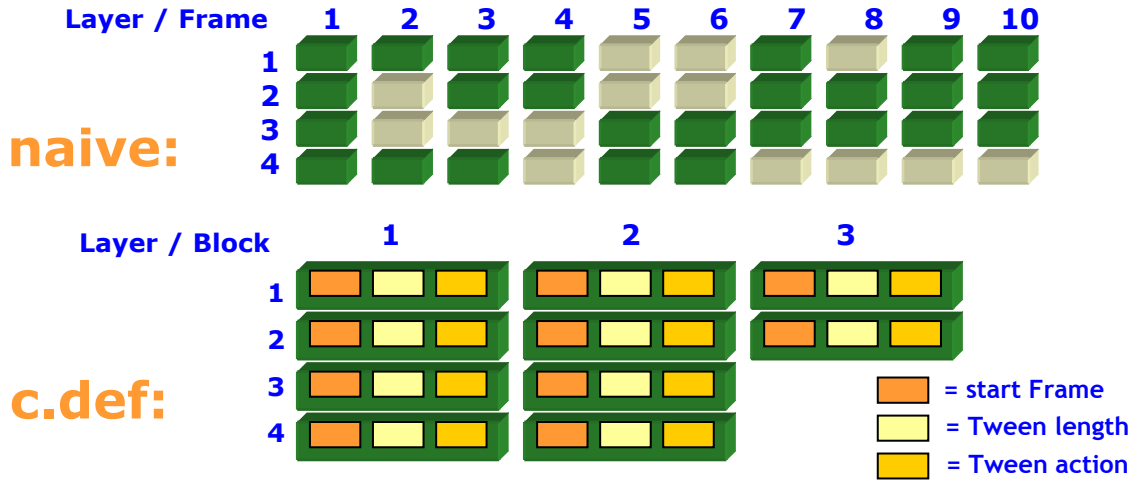
The front end translates the java code produced by the Tree Walker into a valid SWF binary, using Flagstone's `Transform API`.

5.3 Features

a. Key Frames

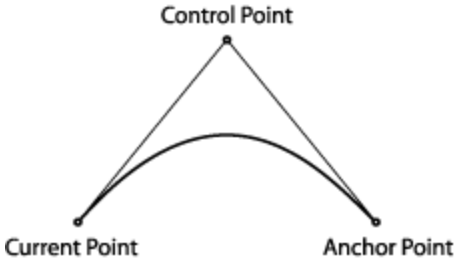
Rather than utilizing a naïve two-dimensional array to store the contents of each frame corresponding to each layer, the **c.def™** language was implemented using the notion of key frames.

Each key frame stores the index of the start frame, the length of frames over which an action will take place, and the action itself. A diagram illustrating this concept is as follows:



b. Bezier Curves

c.def™ implements the circle primitive internally by constructing the shape using a Quadratic Bezier Curve. The curve is specified using three points: the current drawing position, an off-curve control point and an on-curve anchor point which defines the end-point of the curve.



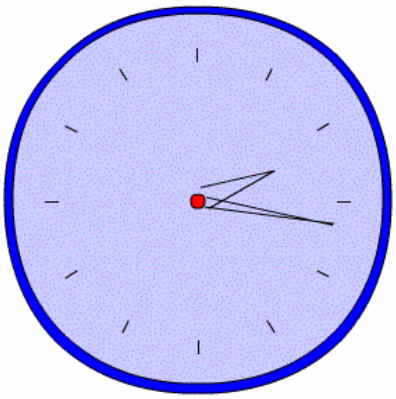
6 Test Plan

6.1 Clock

a. Purpose

The clock tested the functionality of nesting Glyphs within one another, and the order of precedence for layering. The clock face is painted first, then the tick marks, and then the hour and minute hand are rendered over a series of 1080 frames, moving in sync with one another.

b. Output



6.2 Guinness

a. Purpose

The Guinness demo accomplished several goals. First and foremost, it allowed us to pay tribute to a long time comrade. Secondly, it allowed us to test the internal calculation engine, since the foam on the beer grows linearly with the volume of the Guinness. This demo tested multiplication, addition, and subtraction operations.

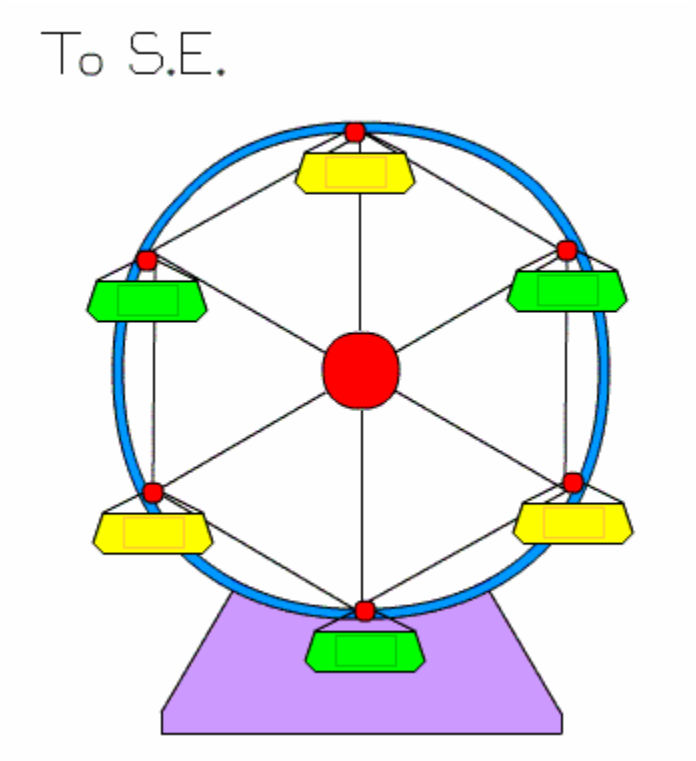
b. Output

6.3 Ferris Wheel

a. Purpose

The Ferris Wheel implements the majority of the functionality of the **c.def™** language. It tests rotation, translation, nested glyphs, conditional statements, arithmetic operations, and color functions.

b. Output

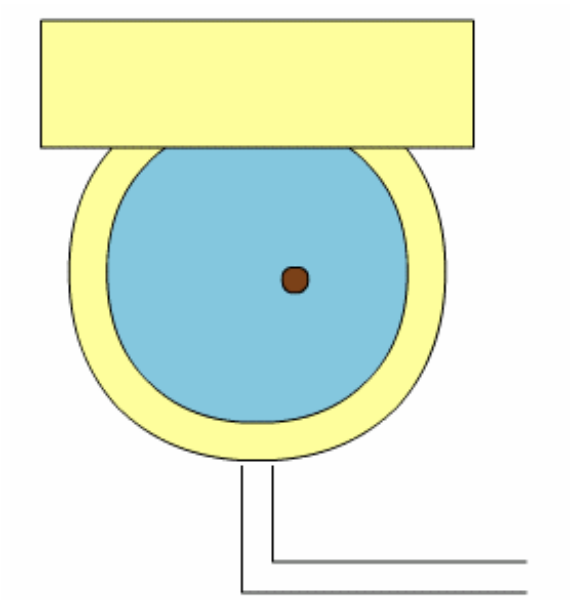


6.4 Toilet

a. Purpose

The purpose of the toilet is to illustrate translation of a Glyph along multiple paths. In this case, the path linearly shrinks with the frame index, so the path is actually a spiral. Once the feces reach the middle of the bowl, it follows a linear path through the plumbing.

b. Output



6.5 Illegal Programs

a. Purpose

We created many sample programs to test the error handling of our back end. These tests initially returned no error output, but created malformed SWF movies. Since the tests were performed incrementally and used to debug the application, they now result in specified error conditions from the back end.

7 Lessons Learned

7.1 Dennis Rakhamimov (Group Leader)

The development of **c.def™** had been a great lesson in both teamwork and the importance of structured programming. While it is probably true that anyone could write code, the best and working code is produced through methodical planning and collaboration. Since my project parts included the lexer, parser, tree walker, and keyframe generation, one immediate challenge was in deciding just how the language should be implemented. Particularly, it was important to decide up front what Flash functionality we would choose to expose to the language user. After spending several days discussing various ideas, I realized that a good idea may be to work backwards and first create a source file that demonstrates language functionality. Upon having created a ferriswheel.cdef source file, the development process became much easier.

Another lesson I learned was in the importance of top-down and object-oriented design, since splitting the functionality into distinct classes greatly facilitated the coding. Last but not least lesson was that of teamwork, as the project would not have been as successful without regular communication with my team members, along with appropriate delegation of tasks and ensuring the seamless integration of components.

In overall, this project had been both a challenging and fun experience -- it's been great to create my first compiler (interpreter) with such a great team!

7.2 Eric Poirier

Since there was a high degree of co-dependency among the language components, it was essential that our team communicate accurately and frequently. Our object-oriented approach made it imperative that we use stub-implementation to design the fundamental structure of our language. From there, we were successful by using a development methodology similar to extreme programming.

While working on the Whitepaper, the Language Reference Manual and this Final Report, I have recognized the importance of paying meticulous attention to detail in the planning, implementation, and presentation of **c.def™.** After all, the language is useful only when others understand the paradigm, advantages and limitations.

7.3 Charles Catanach

The most valuable experience of the project for me was the communication skills I was forced to learn early on; in order to keep track of project status. The problems we faced weren't what I expected, our separate modules were usually integrated seamlessly, we simply had different expectations of what each of the group members was responsible for, and over what timeline. We quickly learned that short one-to-one emails were an effective supplement to the group-wide emails, which greatly assisted project synchronization.

7.4 Tecuan Flores

PLT has demonstrated the importance of strong and effective communication skills. Each member successfully contributed to

overall success of our programming language. Through daily meetings we were able to talk through issues and design decisions. I learned how combining knowledge and skills of all team members makes projects diverse and successful.

It is also important to point out the effectiveness of strict timelines and milestones. Dennis, cdef™ team leader, assisted me in becoming more efficient with time management. The skills and knowledge I have gained from this course will assist me with future endeavors. This project has proven to be an extremely educational and enjoyable experience.

8 Appendix

8.1 Source Code

```
------------------------------

/*
   c.def Language Interpreter
   CDEF.java: Main entry point
   Author: Dennis Rakhamimov
 */

import antlr.*;
import antlr.collections.*;
import antlr.debug.misc.ASTFrame;
import antlr.CommonAST;

import java.io.*;

class CDEF
{
    public static void main(String[] args)
    {
       System.out.println("CDEF Language Interpreter");

       if(args.length != 1)
       {
          System.out.println("Usage: java CDEF <sourcefilef>");
          System.exit(1);
       }

       FileInputStream fileInput = null;

       // Attempt to obtain the input stream for the file
       try
       {
           fileInput = new FileInputStream(args[0]);

       }
       catch(FileNotFoundException e)
       {
           printError("The specified source file '" + args[0] + "' was not
found.");
       }

       try
       {
          // Perform lexical analysis
          CDEFLexer lexer = new CDEFLexer(fileInput);
```

```
            // Parse tokens and build an AST
            CDEFParser parser = new CDEFParser(lexer);
            parser.startRule();

            CDEFWalker walker = new CDEFWalker();
            CommonAST tree = (CommonAST)parser.getAST();

            if(tree == null)
            {
               printError("Aborting due to syntax errors.");
            }

            CDEFDocument doc = new CDEFDocument();

            System.out.print("Generating keyframe table... ");

            // Run the tree walker
            walker.expr(tree, doc);

            System.out.println(doc.getNumKeyframes() + " keyframes created.");

            int extPos = args[0].lastIndexOf(".cdef");
            String filenameSWF;

            if(extPos != -1)
            {
               filenameSWF = args[0].substring(0, extPos) + ".swf";
            }
            else
            {
               filenameSWF = args[0] + ".swf";
            }

            System.out.println("Generating SWF file...");

            // Produce output
            doc.createSWF(filenameSWF);

            System.out.println("\nSUCCESS: Created output file '" + filenameSWF +
"'");
         }
         catch(TokenStreamException e)
         {
            printError(e.getMessage());
         }
         catch(RecognitionException e)
         {
            printError(e.getMessage());
         }
      }

      /* printError()
       * Displays an error and terminates the program
       */
      public static void printError(String errMsg)
      {
         System.out.println("\n" + errMsg);

         System.out.println("\nERROR: Problems encountered, aborting.");
         System.exit(1);
      }
}

//
------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: "CDEFGrammar.g" -> "CDEFLexer.java"$
```

```
public interface CDEFAntlrTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int DIGIT = 4;
	int ALPHA = 5;
	int LBRACKET = 6;
	int RBRACKET = 7;
	int LPAREN = 8;
	int RPAREN = 9;
	int LBRACE = 10;
	int RBRACE = 11;
	int COLON = 12;
	int SEMICOLON = 13;
	int PLUS = 14;
	int MINUS = 15;
	int TIMES = 16;
	int DIV = 17;
	int MOD = 18;
	int COMMA = 19;
	int UNDER = 20;
	int EQ = 21;
	int NE = 22;
	int GT = 23;
	int LT = 24;
	int GE = 25;
	int LE = 26;
	int AND = 27;
	int OR = 28;
	int NOT = 29;
	int AMPER = 30;
	int POUND = 31;
	int RANGE = 32;
	int WHITESPACE = 33;
	int NEWLINE = 34;
	int COMMENT = 35;
	int ID = 36;
	int NUMBER = 37;
	int DECL = 38;
	int COORDS = 39;
	int COLOR_NAME = 40;
	int COLOR_RGB = 41;
	int OBJINFO = 42;
	int CMD = 43;
	int BODY = 44;
	int UMINUS = 45;
	int INSERT = 46;
	int LITERAL_Document = 47;
	int LITERAL_Glyph = 48;
	int LITERAL_Path = 49;
	int LITERAL_line = 50;
	int LITERAL_circle = 51;
	int LITERAL_rect = 52;
	int LITERAL_ellipse = 53;
	int LITERAL_polygon = 54;
	int LITERAL_color = 55;
	int LITERAL_fillcolor = 56;
	int LITERAL_Render = 57;
	int LITERAL_Rotate = 58;
	int LITERAL_Translate = 59;
	int LITERAL_for = 60;
	int LITERAL_break = 61;
	int LITERAL_continue = 62;
	int LITERAL_if = 63;
	int LITERAL_else = 64;
}
//
-------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: CDEFGrammar.g -> CDEFAntlrTokenTypes.txt$
```

```
CDEFAntlr    // output token vocab name
DIGIT=4
ALPHA=5
LBRACKET("'['")=6
RBRACKET("']'")=7
LPAREN("'('")=8
RPAREN("')'")=9
LBRACE("'{'")=10
RBRACE("'}'")=11
COLON("':'")=12
SEMICOLON("';'")=13
PLUS=14
MINUS=15
TIMES=16
DIV=17
MOD=18
COMMA=19
UNDER=20
EQ=21
NE=22
GT=23
LT=24
GE=25
LE=26
AND=27
OR=28
NOT=29
AMPER=30
POUND=31
RANGE=32
WHITESPACE=33
NEWLINE=34
COMMENT=35
ID=36
NUMBER=37
DECL=38
COORDS=39
COLOR_NAME=40
COLOR_RGB=41
OBJINFO=42
CMD=43
BODY=44
UMINUS=45
INSERT=46
LITERAL_Document="Document"=47
LITERAL_Glyph="Glyph"=48
LITERAL_Path="Path"=49
LITERAL_line="line"=50
LITERAL_circle="circle"=51
LITERAL_rect="rect"=52
LITERAL_ellipse="ellipse"=53
LITERAL_polygon="polygon"=54
LITERAL_color="color"=55
LITERAL_fillcolor="fillcolor"=56
LITERAL_Render="Render"=57
LITERAL_Rotate="Rotate"=58
LITERAL_Translate="Translate"=59
LITERAL_for="for"=60
LITERAL_break="break"=61
LITERAL_continue="continue"=62
LITERAL_if="if"=63
LITERAL_else="else"=64

--------------------------------

/*
   c.def Language
   CDEFColor: RGB color container
   Author: Charles Catanach
 */

public class CDEFColor implements CDEFParam
```

```
{
   int r;
   int g;
   int b;

   static final String defColors[] =
      { "None",
        "White",              "Black",
        "Red",                "Green",
        "Blue",               "Yellow",
        "Purple",             "Navy",
        "Gray",               "Aqua",
        "Teal",               "Olive",
        "Lime",               "Maroon",
        "Fuchsia",            "Silver"

      };

   static final int defColorVals[][] =
      { {    -1,  -1,   -1 }
      , { 0xFF, 0xFF, 0xFF }, { 0x00, 0x00, 0x00 }
      , { 0xFF, 0x00, 0x00 }, { 0x00, 0x80, 0x00 }
      , { 0x00, 0x00, 0xFF }, { 0xFF, 0xFF, 0x00 }
      , { 0x80, 0x00, 0x80 }, { 0x00, 0x00, 0x80 }
      , { 0x80, 0x80, 0x80 }, { 0x00, 0xFF, 0xFF }
      , { 0x00, 0x80, 0x80 }, { 0x80, 0x80, 0x00 }
      , { 0x00, 0xFF, 0x00 }, { 0x80, 0x00, 0x00 }
      , { 0xFF, 0x00, 0xFF }, { 0xC0, 0xC0, 0xC0 }
      };

   public CDEFColor(String colorName)
   {
      boolean found = false;

      for(int i = 0; i < defColors.length; i++)
      {
         if(defColors[i].equals(colorName))
         {
            r = defColorVals[i][0];
            g = defColorVals[i][1];
            b = defColorVals[i][2];

            found = true;
            break;
         }
      }

      if(!found)
      {
         CDEF.printError("Unknown color literal '" + colorName + "'");
      }
   }

   public CDEFColor(int rIn, int gIn, int bIn)
   {
      if((rIn < 0 || rIn > 255) || (gIn < 0 || gIn > 255)
            || (bIn < 0 || bIn > 255))
      {
         CDEF.printError("RGB color values must be in the range of 0 to 255.");
      }

      r = rIn;
      g = gIn;
      b = bIn;
   }

   public int getR()
   {
      return r;

   }
   public int getG()
```

```
   {
      return g;

   }
   public int getB()
   {
      return b;
   }
}

------------------------------

/*
   c.def Language
   CDEFDocument: Keyframe container, and Flash file format generation
   Authors: Dennis Rakhamimov, Charles Catanach
 */

import java.util.ArrayList;
import java.util.Vector;
import com.flagstone.transform.*;
import java.io.*;

public class CDEFDocument implements CDEFObject
{
   private CDEFColor bgColor;
   private CDEFXY    size;
   private Vector dispLists;
   private Vector frames;
   private int numFrames;
   private int numKeyframes;

   public CDEFDocument()
   {
      frames = new Vector(10);     // layers

         // initialize Flash format properties
      Transform.COORDINATES_ARE_PIXELS = true;
      Transform.FONT_SIZE_IS_PIXELS = true;
      Transform.LINE_WIDTHS_ARE_PIXELS = true;
      Transform.IMAGE_SIZES_ARE_PIXELS = true;

      frames.add(new Vector(100)); // frames

      dispLists = new Vector(30);

      numKeyframes = 0;
   }

   public void setParams(CDEFXY size, CDEFColor bgColor, int numFrames)
   {
      if(numFrames < 1)
      {
         CDEF.printError("The document must have at least 1 frame.");
      }

      this.bgColor = bgColor;
      this.size = size;
      this.numFrames = numFrames;
   }

   public void render(CDEFGlyph glyph, int from,
         int to, int step, CDEFPath path, CDEFXY origin)
   {
      Vector layer = null;
      int displayListInd = 0;
      int insertPos = 0;

      CDEFXY glyphOrigin = glyph.getCenter();

      displayListInd = dispLists.size();
```

```
FSDefineShape shapeDef = glyph.toDisplayList(displayListInd + 1);

dispLists.add(shapeDef);

// First we determine which layer should hold the new keyframe(s)
boolean foundLayer = false;

for(int i = 0; i < frames.size(); i++) // iterate through layers
{
   layer = (Vector)frames.get(i);

   boolean taken = false;

   for(int j = 0; j < layer.size(); j++)
   {
      ArrayList keyframe = (ArrayList)layer.get(j);

      int fromFrame = ((Integer)keyframe.get(1)).intValue();
      int toFrame = ((Integer)keyframe.get(2)).intValue();

      /* Is the currently examined layer already taken at the needed
         frame range? */
      if( (fromFrame <= from && toFrame >= from)
          ||  (fromFrame <= to && toFrame >= to) )
      {
         taken = true;
         break;
      }
   }

   if(!taken)
   {
      foundLayer = true;
      break;
   }
}

// All layers were taken at that frame position. Create a new layer
if(!foundLayer)
{
   layer = new Vector(100);
   frames.add(layer);
}

/* We need to insert the keyframe before any other that appears in
   that layer */
insertPos = 0;

int xTrans = 0, yTrans = 0;

if(path != null)
{
   path.initItr((to - from + 1) / step);
}
else if(origin != null)
{
   xTrans = (int)(origin.getX() - glyphOrigin.getX());
   yTrans = (int)(origin.getY() - glyphOrigin.getY());
}
else // This should never happen if the tree walker is correct
{
       CDEF.printError("Internal error.");
}


for(int i = from; i <= to; i += step)
{
   ArrayList keyframeInfo = new ArrayList(5);

       // if we're iterating over a path, obtain the next translation
   if(path != null)
```

```
      {
         CDEFXY xyTrans = path.advanceItr();
         xTrans = (int)(xyTrans.getX() - glyphOrigin.getX());
         yTrans = (int)(xyTrans.getY() - glyphOrigin.getY());

      }

      keyframeInfo.add(new Integer(displayListInd)); // display lst index
      keyframeInfo.add(new Integer(i));              // from frame
      keyframeInfo.add(new Integer(i + step - 1));   // to frame
      keyframeInfo.add(new Integer(xTrans));         // x translation
      keyframeInfo.add(new Integer(yTrans));         // y translation

      layer.add(insertPos, keyframeInfo);
      numKeyframes++;
   }
}

public int getNumKeyframes()
{
   return numKeyframes;
}

public void createSWF(String filename)
{
   FSMovie movie = new FSMovie();

   // set Flash movie properties
   movie.setFrameRate(18.0f);
   movie.setFrameSize(new FSBounds(0, 0, size.getIntX(), size.getIntY()));

   movie.add(new FSSetBackgroundColor(new FSColor(bgColor.getR(),
               bgColor.getG(), bgColor.getB())));

   // iterate through frames
   for(int frameInd = 0; frameInd < numFrames; frameInd++)
   {
          // find any display lists that need to be displayed on this frame
      for(int layerInd = 0; layerInd < frames.size(); layerInd++)
      {
         Vector layer = (Vector)frames.get(layerInd);

         for(int keyframeInd = 0;
               keyframeInd < layer.size(); keyframeInd++)
         {
            ArrayList keyframeInfo = (ArrayList)layer.get(keyframeInd);

            int dispListIndex = ((Integer)keyframeInfo.get(0)).intValue();
            int fromFrame     = ((Integer)keyframeInfo.get(1)).intValue();
            int toFrame       = ((Integer)keyframeInfo.get(2)).intValue();
            int xTrans        = ((Integer)keyframeInfo.get(3)).intValue();
            int yTrans        = ((Integer)keyframeInfo.get(4)).intValue();

            if(frameInd == fromFrame)
            {
               FSDefineShape shapeDef
                  = (FSDefineShape)dispLists.get(dispListIndex);

               movie.add(shapeDef);
               movie.add(new FSPlaceObject(dispListIndex + 1,
                                           layerInd + 1,
                                           xTrans,
                                           yTrans));
            }
         }
      }

      // display the frame
      movie.add(new FSShowFrame());

      for(int layerInd = 0; layerInd < frames.size(); layerInd++)
      {
```

```
            Vector layer = (Vector)frames.get(layerInd);

            for(int keyframeInd = 0;
               keyframeInd < layer.size(); keyframeInd++)
            {
               ArrayList keyframeInfo = (ArrayList)layer.get(keyframeInd);

               int dispListIndex = ((Integer)keyframeInfo.get(0)).intValue();
               int toFrame       = ((Integer)keyframeInfo.get(2)).intValue();

               // if the display list expires on this frame, remove it
               if(frameInd == toFrame && numFrames > 1)
               {
                  movie.add(new FSRemoveObject(dispListIndex + 1,
                        layerInd + 1));
                  layer.remove(keyframeInd);
               }
            }
         }
      }

      // complete flash movie
      movie.add(new FSEnd());

      try
      {
          movie.encodeToFile(filename);
      }
      catch (FileNotFoundException e)
      {
          System.err.println("Cannot open file: " + e.getMessage());
      }
      catch (IOException e)
      {
          System.err.println("Cannot write to file: " + e.getMessage());
      }
   }
}

------------------------------

/*
   c.def Language
   CDEFDocument: Drawing element operations
   Author: Dennis Rakhamimov
 */

import java.util.Vector;
import java.util.ArrayList;
import com.flagstone.transform.*;

public class CDEFElement
{
   private Vector points = null;
   private CDEFColor strokeColor = null;
   private CDEFColor fillColor = null;

   public boolean rotSymmetry = false;

   public static final int TYPE_LINE  = 1;
   public static final int TYPE_CIRCULAR  = 2;
   public static final int TYPE_POLY = 3;

   private int type = 0;

   public CDEFElement(int type, Vector points,
            CDEFColor strokeColor, CDEFColor fillColor, boolean rotSymmetry)
   {
      this.strokeColor = strokeColor;
      this.fillColor = fillColor;
      this.points = points;
      this.type = type;
```

```
   this.rotSymmetry = rotSymmetry;
}

/* toDisplayList()
   Adds this element to the Flash display list, allowing it to be re-used
   for display purposes */
public void toDisplayList(ArrayList shape, ArrayList lineStyles,
         ArrayList fillStyles, int offsetX, int offsetY)
{
   int fillInd = 0, strokeInd = 0;

   // set a fill style if we are using a fill color
   if(fillColor != null)
   {
      fillStyles.add(new FSSolidFill(new FSColor(fillColor.getR(),
            fillColor.getG(), fillColor.getB())));
      fillInd = fillStyles.size();
   }
   // set a stroke color if one is specified
   if(strokeColor != null)
   {
      lineStyles.add(new FSSolidLine(1, new FSColor(strokeColor.getR(),
            strokeColor.getG(), strokeColor.getB())));
      strokeInd = lineStyles.size();
   }

   CDEFXY coordStart = null, coordA, coordB = null;

   double width = 0, height = 0;

   // add display list coordinates based on element type
   switch(type)
   {
      case TYPE_LINE:
         coordStart = (CDEFXY)points.get(0);
         coordB = (CDEFXY)points.get(1);

         shape.add(new FSShapeStyle(strokeInd, 0, 0, coordStart.getIntX(),
               coordStart.getIntY()));
         shape.add(new FSLine(coordB.getIntX(), coordB.getIntY()));
         break;

      case TYPE_POLY:
         coordStart = (CDEFXY)points.get(0);
         coordB = (CDEFXY)points.get(1);

         width = coordB.getX();
         height = coordB.getY();

         int totalX = coordStart.getIntX(), totalY = coordStart.getIntY();

         shape.add(new FSShapeStyle(strokeInd, fillInd, 0,
            coordStart.getIntX(), coordStart.getIntY()));

         for(int i = 1; i < points.size(); i++)
         {
            CDEFXY coord = (CDEFXY)points.get(i);

            totalX += coord.getIntX();
            totalY += coord.getIntY();

            shape.add(new FSLine(coord.getIntX(), coord.getIntY()));
         }

         // if this is a filled polygon, automatically close it
         if(fillColor != null)
         {
            shape.add(new FSLine(coordStart.getIntX() - totalX,
                  coordStart.getIntY() - totalY));
         }

         break;
```

```
      case TYPE_CIRCULAR:
         CDEFXY coordControl = null;
         CDEFXY coordAnchor = null;

         coordStart = (CDEFXY)points.get(0);
         CDEFXY coordOffset = (CDEFXY)points.get(1);

         shape.add(new FSShapeStyle(strokeInd, fillInd, 0,
               coordStart.getIntX() + coordOffset.getIntX(),
               coordStart.getIntY() + coordOffset.getIntY()));

         /* iterate through anchor and control points and form the
            Bezier curves */
         for(int i = 2; i < points.size(); i += 2)
         {
            coordControl = (CDEFXY)points.get(i);
            coordAnchor = (CDEFXY)points.get(i + 1);

            shape.add(new FSCurve(coordControl.getIntX(),
               coordControl.getIntY(),
               coordAnchor.getIntX(),
               coordAnchor.getIntY()));
         }

                break;
      default:  // this should never happen.
         CDEF.printError("Internal error.");
         break;

   }
}

/* deepCopy()
   Performs a deep copy of the element data, allowing Glyphs to be
   nested */
public CDEFElement deepCopy(CDEFXY coords)
{
   Vector vect = new Vector(points.size());
   CDEFXY point = null;

   // iterate through each of the points
   for(int i = 0; i < points.size(); i++)
   {
      point = (CDEFXY)points.get(i);

      // adjust the start point according to new origin
      if(i == 0)
         vect.add(new CDEFXY(point.getX() + coords.getX(),
            point.getY() + coords.getY()));
      // other points are simply translations -- copy them
      else
         vect.add(new CDEFXY(point.getX(), point.getY()));
   }

   return new CDEFElement(type, vect, strokeColor, fillColor, rotSymmetry);
}

/* rotate()
   Performs a rotation of this element
 */
public void rotate(CDEFXY center, double cosTheta, double sinTheta)
{
   CDEFXY point = null;
   double pointX, pointY, pointX_, pointY_;

   int rotatePts;//how many points to rotate

   /* if this element has rotational symmetry, we only need to alter
      its origin */
   if(rotSymmetry)
   {
```

```
            rotatePts = 1;
         }
         else
         {
            rotatePts = points.size();
         }

         // Modify each of the points
         for(int i = 0; i < rotatePts; i++)
         {
            point = (CDEFXY)points.get(i);

            pointX = point.getX();
            pointY = point.getY();

            // The first element is the origin, so we translate it to center
            if(i == 0)
            {
               pointX -= center.getX();
               pointY -= center.getY();
            }

            // Calculate the rotation
            pointX_ = cosTheta * pointX - sinTheta * pointY;
            pointY_ = cosTheta * pointY + sinTheta * pointX;

            // Translate the rotated origin back to its position
            if(i == 0)
            {
               pointX_ += (center.getX());
               pointY_ += (center.getY());
            }

            // Update the coordinates
            point.setXY(pointX_, pointY_);
         }
      }
   }

   ------------------------------

   /*
      c.def Language
      CDEFDocument: Drawing element creation
      Author: Charles Catanach
    */

   import java.util.Vector;

   public class CDEFElementFactory
   {
      private CDEFColor strokeColor = null;
      private CDEFColor fillColor = null;

      public CDEFElementFactory()
      {
         strokeColor = new CDEFColor(0, 0, 0);
      }

      public void setStrokeColor(CDEFColor color)
      {
         if(color.getR() != -1)
            strokeColor = color;
         else
            strokeColor = null;
      }

      public void setFillColor(CDEFColor color)
      {
         if(color.getR() != -1)
            fillColor = color;
         else
```

```
      fillColor = null;
}

public void createPoly(CDEFObject cont, Vector coords)
{
   if(cont instanceof CDEFGlyph)
   {
      ((CDEFGlyph)cont).addElement(new CDEFElement(CDEFElement.TYPE_POLY,
                                 coords, strokeColor, fillColor, false));
   }
   else if(cont instanceof CDEFPath)
   {
      ((CDEFPath)cont).setPathInfo(CDEFElement.TYPE_POLY, coords);
   }
   else // This should never occur if the tree walker works correctly
   {
      CDEF.printError("Internal error.");
   }
}

public void createRect(CDEFObject cont, CDEFXY coords1, CDEFXY coords2)
{
   Vector vect = new Vector(2);

   double originX = (coords1.getX() > coords2.getX())
               ? coords2.getX()
               : coords1.getX()
               ;
   double originY = (coords1.getY() > coords2.getY())
               ? coords2.getY()
               : coords1.getY()
               ;

   double width  = (coords1.getX() > coords2.getX())
              ? (coords1.getX() - coords2.getX())
              : (coords2.getX() - coords1.getX())
              ;
   double height = (coords1.getY() > coords2.getY())
              ? (coords1.getY() - coords2.getY())
              : (coords2.getY() - coords1.getY())
              ;

   vect.add(new CDEFXY(originX, originY));
   vect.add(new CDEFXY(width, 0));
   vect.add(new CDEFXY(0, height));
   vect.add(new CDEFXY(-width, 0));
   vect.add(new CDEFXY(0, -height));

   if(cont instanceof CDEFGlyph)
   {
      ((CDEFGlyph)cont).addElement(new CDEFElement(CDEFElement.TYPE_POLY,
                vect, strokeColor, fillColor, false));
   }
   else if(cont instanceof CDEFPath)
   {
      ((CDEFPath)cont).setPathInfo(CDEFElement.TYPE_POLY, vect);
   }
   else // This should never occur if the tree walker works correctly
   {
      CDEF.printError("Internal error.");
   }
}

public void createLine(CDEFObject cont, CDEFXY coords1, CDEFXY coords2)
{
   Vector vect = new Vector(2);

   vect.add(coords1);
   vect.add(new CDEFXY(coords2.getX() - coords1.getX(),
                       coords2.getY() - coords1.getY()));
```

```
   if(cont instanceof CDEFGlyph)
   {
      ((CDEFGlyph)cont).addElement(new CDEFElement(CDEFElement.TYPE_LINE,
                                   vect, strokeColor, fillColor, false));
   }
   else if(cont instanceof CDEFPath)
   {
      ((CDEFPath)cont).setPathInfo(CDEFElement.TYPE_POLY, vect);
   }
   else // This should never occur if the tree walker works correctly
   {
      CDEF.printError("Internal error.");
   }
}

public void  createCircular(CDEFObject cont, CDEFXY coords,
      int radiusX, int radiusY)
{
   boolean isCircle = false;

   if(radiusX == radiusY)
      isCircle = true;

   if(cont instanceof CDEFGlyph)
   {
      Vector vect = new Vector(9);

      double curX = coords.getX();
      double curY = coords.getY();

      vect.add(new CDEFXY(curX, curY)); // top
      vect.add(new CDEFXY(0, -radiusY));

      // Take the faster way if the radius is small
      if(radiusX < 20)
      {
         // Top Right (Quadrant I)
         vect.add(new CDEFXY( radiusX,        0));
         vect.add(new CDEFXY(       0,  radiusY));

         // Bottom Right (Quadrant IV)
         vect.add(new CDEFXY(       0,  radiusY));
         vect.add(new CDEFXY(-radiusX,        0));

         // Bottom Left (Quadrant III)
         vect.add(new CDEFXY(-radiusX,        0));
         vect.add(new CDEFXY(       0, -radiusY));

         // Top Left (Quadrant II)
         vect.add(new CDEFXY(       0, -radiusY));
         vect.add(new CDEFXY( radiusX,        0));
      }
      else
      {
         double dW1 = 0.45 * radiusX;
         double dW2 = 0.28 * radiusX;
         double dW3 = 0.27 * radiusX;
         double dW4 = 0.00 * radiusX;

         double dH1 = 0.45 * radiusY;
         double dH2 = 0.28 * radiusY;
         double dH3 = 0.27 * radiusY;
         double dH4 = 0.00 * radiusY;


         // Top Right (Quadrant I)
         vect.add(new CDEFXY( dW1 ,  dH4 )); // control
         vect.add(new CDEFXY( dW2 ,  dH3 )); // anchor
         vect.add(new CDEFXY( dW3 ,  dH2 )); // control
         vect.add(new CDEFXY( dW4 ,  dH1 )); // anchor

         // Bottom Right (Quadrant IV)
```

```
            vect.add(new CDEFXY(-dW4 ,  dH1 )); // control
            vect.add(new CDEFXY(-dW3 ,  dH2 )); // anchor
            vect.add(new CDEFXY(-dW2 ,  dH3 )); // control
            vect.add(new CDEFXY(-dW1 ,  dH4 )); // anchor

            // Bottom Left (Quadrant III)
            vect.add(new CDEFXY(-dW1 , -dH4 )); // control
            vect.add(new CDEFXY(-dW2 , -dH3 )); // anchor
            vect.add(new CDEFXY(-dW3 , -dH2 )); // control
            vect.add(new CDEFXY(-dW4 , -dH1 )); // anchor

            // Top Left (Quadrant II)
            vect.add(new CDEFXY( dW4 , -dH1 )); // control
            vect.add(new CDEFXY( dW2 , -dH3 )); // anchor
            vect.add(new CDEFXY( dW3 , -dH2 )); // control
            vect.add(new CDEFXY( dW1 , -dH4 )); // anchor
         }


         ((CDEFGlyph)cont).addElement(new
               CDEFElement(CDEFElement.TYPE_CIRCULAR, vect,
                  strokeColor, fillColor, isCircle));

      }
      else if(cont instanceof CDEFPath)
      {
         Vector vect = new Vector(3);
         vect.add(new CDEFXY(coords.getX(), coords.getY()));
         vect.add(new CDEFXY(radiusX, radiusY));


         ((CDEFPath)cont).setPathInfo(CDEFElement.TYPE_CIRCULAR, vect);
      }
      else // This should never occur if the tree walker works correctly
      {
         CDEF.printError("Internal error.");
      }
   }
}

------------------------------

/*
   c.def Language
   CDEFGlyph: Drawing element container and display
   Author: Charles Catanach
 */

import java.util.ArrayList;
import java.util.Vector;
import com.flagstone.transform.*;

public class CDEFGlyph implements CDEFObject
{
   private Vector elements;
   private CDEFXY center;

   public CDEFGlyph(CDEFXY center)
   {
      elements = new Vector();

      this.center = center;
   }

   public void addElement(CDEFElement elem)
   {
      elements.add(elem);
   }

   public CDEFXY getCenter()
   {
      return center;
```

```
   }

   public void appendGlyph(CDEFGlyph glyph, CDEFXY coords)
   {
      glyph.appendTo(elements, coords);
   }


   public void appendTo(Vector parentElements, CDEFXY coords)
   {
      for(int i = 0; i < elements.size(); i++)
      {
         parentElements.add(((CDEFElement)elements.get(i)).deepCopy(coords));
      }
   }

   public void rotate(int degrees)
   {
      double theta = degrees * Math.PI / 180.0;
      double cosTheta = Math.cos(theta);
      double sinTheta = Math.sin(theta);

      for(int i = 0; i < elements.size(); i++)
      {
         ((CDEFElement)elements.get(i)).rotate(center, cosTheta, sinTheta);
      }
   }

   /* toDisplayList()
      Sends this Glyph to a display list by obtaining points and colors of
      all enclosed drawing elements */
   public FSDefineShape toDisplayList(int shapeId)
   {
      FSBounds bounds = new FSBounds(-100, -100, 500, 500);

      ArrayList fillStyles = new ArrayList();
      ArrayList lineStyles = new ArrayList();
      ArrayList shape = new ArrayList();

      // add each element to the display list
      for(int elemInd = 0; elemInd < elements.size(); elemInd++)
      {
         CDEFElement element = (CDEFElement)elements.get(elemInd);

         element.toDisplayList(shape, lineStyles,
               fillStyles, center.getIntX(), center.getIntY());
      }

      shape.add(new FSEndShape());

      FSDefineShape shapeDef = null;

      shapeDef = new FSDefineShape(shapeId, bounds,
            fillStyles, lineStyles, new FSShape(shape));


      return shapeDef;
   }
}

------------------------------

/*
   c.def Language
   CDEFGrammar.g: Lexer and Parser grammar for ANTLR
   Author: Dennis Rakhamimov
 */

class CDEFLexer extends Lexer;

options
{
```

```
   charVocabulary = '\3'..'\377';
   testLiterals = false;
   exportVocab = CDEFAntlr;
   defaultErrorHandler=false;
   k = 2;
}

{
    int total_errors = 0;
    public void reportError( String s )
    {
        super.reportError( s );
        total_errors++;
    }
    public void reportError( RecognitionException e )
    {
        super.reportError( e );
        total_errors++;
    }
}

protected
DIGIT: '0'..'9' ;

protected
ALPHA: 'A'..'Z' | 'a'..'z';

LBRACKET   options { paraphrase = "'['"; } : '[';
RBRACKET   options { paraphrase = "']'"; } : ']';
LPAREN     options { paraphrase = "'('"; } : '(';
RPAREN     options { paraphrase = "')'"; } : ')';
LBRACE     options { paraphrase = "'{'"; } : '{';
RBRACE     options { paraphrase = "'}'"; } : '}';

COLON      options { paraphrase = "':'"; } : ':' ;
SEMICOLON  options { paraphrase = "';'"; } : ';';

PLUS: '+';
MINUS: '-';
TIMES: '*';
DIV: '/';
MOD: '%';

COMMA: ',' ;

UNDER: '_';

EQ:  "==";
NE:  "!=";
GT:  '>';
LT:  '<';
GE:  ">=";
LE:  "<=";
AND: "&&";
OR:  "||";
NOT: '!';

AMPER: '&';
POUND: '#';
RANGE: "->";

// These have been borrowed from sample code
WHITESPACE : (' ' | '\t')+ { $setType(Token.SKIP); } ;
NEWLINE    : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
                { $setType(Token.SKIP); newline(); } ;
COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NEWLINE)
                    | ~( '\n' | '\r' )
                )* "*/"
          | "//" (~( '\n' | '\r' ))* (NEWLINE)
        )                       { $setType(Token.SKIP); }
```

```
        ;


ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT|UNDER)*;

NUMBER: (DIGIT)+;


class CDEFParser extends Parser;

options
{
   k = 2;
   buildAST = true;
   exportVocab = CDEFAntlr;
   defaultErrorHandler=false;
}

tokens
{
    DECL;
    COORDS;
    COLOR_NAME;
    COLOR_RGB;
    OBJINFO;
    CMD;
    BODY;
    UMINUS;
    INSERT;
}

{
    int total_errors = 0;
    public void reportError( String s )
    {
        super.reportError( s );
        total_errors++;
    }
    public void reportError( RecognitionException e )
    {
        super.reportError( e );
        total_errors++;
    }
}

startRule
        : document EOF!
        ;


document :  "Document"^ ID LBRACKET! coords COMMA! color COMMA!
            expr RBRACKET!
            code_body
         ;

code_body
         :
             LBRACE!
                 (declaration | statement)*
             RBRACE!

           { #code_body = #([BODY, "BODY"], code_body);     }
              ;

declaration
        : ("Glyph"^ ID LBRACKET! coords RBRACKET!
           glyph_body

          )
        | ("Path"^ ID LBRACKET! expr RBRACKET!
           path_body
```

```
          )
           ;

glyph_body
        :    LBRACE!
                        (statement)*
             RBRACE!

           { #glyph_body = #([BODY, "BODY"], glyph_body);     }
              ;

path_body
        :
             LBRACE!
                         draw_statement
             RBRACE!
               { #path_body = #([BODY, "BODY"], path_body);     }
        ;

coords
       // Coordinates, such as &(100, 30)
     : AMPER! LPAREN! expr COMMA! expr RPAREN!
      {#coords = #([COORDS,"COORDS"], coords); }
     ;

color
       // Color, such as #Blue or #(0, 0, 255)
     : POUND! ID
          {#color = #([COLOR_NAME,"COLOR_NAME"], color); }

     | POUND! LPAREN! expr COMMA! expr COMMA! expr RPAREN!
          {#color = #([COLOR_RGB,"COLOR_RGB"], color); }
     ;

range  // Range, such as ->(1, 100) or ->(1, 100, 5)
       // ->( from, to ) or ->( from, to, step )
     : RANGE^ LPAREN! expr COMMA! expr (COMMA! expr)? RPAREN!
     ;

statement
     : draw_statement
     | action_statement
     | for_statement
     | if_statement
     | break_statement
     | continue_statement
     ;

draw_statement
           : "line"^ LBRACKET! coords COMMA! coords RBRACKET! SEMICOLON!
           | "circle"^ LBRACKET! coords COMMA! expr RBRACKET! SEMICOLON!
           | "rect"^ LBRACKET! coords COMMA! coords RBRACKET! SEMICOLON!
           | "ellipse"^ LBRACKET! coords COMMA! expr COMMA! expr RBRACKET!
               SEMICOLON!
           | "polygon"^ LBRACKET! coords (COMMA! coords)* RBRACKET!
               SEMICOLON!
           | "color"^ LBRACKET! color RBRACKET! SEMICOLON!
           | "fillcolor"^ LBRACKET! color RBRACKET! SEMICOLON!
           | ID LBRACKET! coords RBRACKET! SEMICOLON!
                {#draw_statement = #([INSERT,"INSERT"], draw_statement); }
           ;

action_statement
           : "Render"^ LBRACKET! ID COMMA! (range | expr) COMMA!
               (coords | ID) RBRACKET! SEMICOLON!
           | "Rotate"^ LBRACKET! ID COMMA! expr RBRACKET! SEMICOLON!
           | "Translate"^ LBRACKET! ID COMMA! coords RBRACKET! SEMICOLON!
           ;

for_statement
           : "for"^ LBRACKET! ID COLON! range RBRACKET!
            code_body
```

```
           ;

break_statement
           : "break"^ SEMICOLON
           ;

continue_statement
           : "continue"^ SEMICOLON
           ;


if_statement
       : "if"^ LBRACKET! expr RBRACKET!
         code_body
         (
           "else"!
           code_body
         )?
       ;

expr
       : expr_not ( ( AND^ | OR^) expr_not)?
       ;

expr_not
       : (NOT^)? expr_compare
       ;

expr_compare
       : expr_add_sub (( EQ^
                       | NE^
                       | GT^
                       | LT^
                       | GE^
                       | LE^
                        ) expr_add_sub)?
       ;

expr_add_sub
       : expr_mul_div ( (PLUS^ expr_mul_div)
                      | (MINUS^ expr_mul_div)
                      )*
       ;

expr_mul_div
       : expr_unary ( (MOD^ expr_unary)
              | (TIMES^ expr_unary)
              | (DIV^ expr_unary)
              )*
       ;

expr_unary
       : (MINUS! NUMBER)
          {#expr_unary = #([UMINUS,"UMINUS"], expr_unary); }
       | atom
       ;

atom
       : (LPAREN! expr RPAREN!)
       | NUMBER
       | ID
       ;
//
--------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: "CDEFGrammar.g" -> "CDEFLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
```

```
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class CDEFLexer extends antlr.CharScanner implements CDEFAntlrTokenTypes,
TokenStream
 {

    int total_errors = 0;
    public void reportError( String s )
    {
        super.reportError( s );
        total_errors++;
    }
    public void reportError( RecognitionException e )
    {
        super.reportError( e );
        total_errors++;
    }
public CDEFLexer(InputStream in) {
      this(new ByteBuffer(in));
}
public CDEFLexer(Reader in) {
      this(new CharBuffer(in));
}
public CDEFLexer(InputBuffer ib) {
      this(new LexerSharedInputState(ib));
}
public CDEFLexer(LexerSharedInputState state) {
      super(state);
      caseSensitiveLiterals = true;
      setCaseSensitive(true);
      literals = new Hashtable();
      literals.put(new ANTLRHashString("ellipse", this), new Integer(53));
      literals.put(new ANTLRHashString("line", this), new Integer(50));
      literals.put(new ANTLRHashString("Document", this), new Integer(47));
      literals.put(new ANTLRHashString("fillcolor", this), new Integer(56));
      literals.put(new ANTLRHashString("rect", this), new Integer(52));
      literals.put(new ANTLRHashString("if", this), new Integer(63));
      literals.put(new ANTLRHashString("for", this), new Integer(60));
      literals.put(new ANTLRHashString("polygon", this), new Integer(54));
      literals.put(new ANTLRHashString("break", this), new Integer(61));
      literals.put(new ANTLRHashString("Rotate", this), new Integer(58));
      literals.put(new ANTLRHashString("Translate", this), new Integer(59));
      literals.put(new ANTLRHashString("Glyph", this), new Integer(48));
      literals.put(new ANTLRHashString("else", this), new Integer(64));
      literals.put(new ANTLRHashString("continue", this), new Integer(62));
      literals.put(new ANTLRHashString("Render", this), new Integer(57));
      literals.put(new ANTLRHashString("circle", this), new Integer(51));
      literals.put(new ANTLRHashString("color", this), new Integer(55));
      literals.put(new ANTLRHashString("Path", this), new Integer(49));
}

public Token nextToken() throws TokenStreamException {
```

```
	Token theRetToken=null;
tryAgain:
	for (;;) {
		Token _token = null;
		int _ttype = Token.INVALID_TYPE;
		resetText();
		try {   // for char stream error handling
			try {   // for lexical error handling
				switch ( LA(1)) {
				case '[':
				{
					mLBRACKET(true);
					theRetToken=_returnToken;
					break;
				}
				case ']':
				{
					mRBRACKET(true);
					theRetToken=_returnToken;
					break;
				}
				case '(':
				{
					mLPAREN(true);
					theRetToken=_returnToken;
					break;
				}
				case ')':
				{
					mRPAREN(true);
					theRetToken=_returnToken;
					break;
				}
				case '{':
				{
					mLBRACE(true);
					theRetToken=_returnToken;
					break;
				}
				case '}':
				{
					mRBRACE(true);
					theRetToken=_returnToken;
					break;
				}
				case ':':
				{
					mCOLON(true);
					theRetToken=_returnToken;
					break;
				}
				case ';':
				{
					mSEMICOLON(true);
					theRetToken=_returnToken;
					break;
				}
				case '+':
				{
					mPLUS(true);
					theRetToken=_returnToken;
					break;
				}
				case '*':
				{
					mTIMES(true);
					theRetToken=_returnToken;
					break;
				}
				case '%':
				{
					mMOD(true);
```

```
        theRetToken=_returnToken;
        break;
}
case ',':
{
        mCOMMA(true);
        theRetToken=_returnToken;
        break;
}
case '_':
{
        mUNDER(true);
        theRetToken=_returnToken;
        break;
}
case '=':
{
        mEQ(true);
        theRetToken=_returnToken;
        break;
}
case '|':
{
        mOR(true);
        theRetToken=_returnToken;
        break;
}
case '#':
{
        mPOUND(true);
        theRetToken=_returnToken;
        break;
}
case '\t':  case ' ':
{
        mWHITESPACE(true);
        theRetToken=_returnToken;
        break;
}
case '\n':  case '\r':
{
        mNEWLINE(true);
        theRetToken=_returnToken;
        break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case 'a':  case 'b':
case 'c':  case 'd':  case 'e':  case 'f':
case 'g':  case 'h':  case 'i':  case 'j':
case 'k':  case 'l':  case 'm':  case 'n':
case 'o':  case 'p':  case 'q':  case 'r':
case 's':  case 't':  case 'u':  case 'v':
case 'w':  case 'x':  case 'y':  case 'z':
{
        mID(true);
        theRetToken=_returnToken;
        break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
        mNUMBER(true);
        theRetToken=_returnToken;
        break;
}
default:
```

```
                                    if ((LA(1)=='!') && (LA(2)=='=')) {
                                          mNE(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='>') && (LA(2)=='=')) {
                                          mGE(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='<') && (LA(2)=='=')) {
                                          mLE(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='&') && (LA(2)=='&')) {
                                          mAND(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='-') && (LA(2)=='>')) {
                                          mRANGE(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='/') &&
(LA(2)=='*'||LA(2)=='/')) {
                                          mCOMMENT(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='-') && (true)) {
                                          mMINUS(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='/') && (true)) {
                                          mDIV(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='>') && (true)) {
                                          mGT(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='<') && (true)) {
                                          mLT(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='!') && (true)) {
                                          mNOT(true);
                                          theRetToken=_returnToken;
                                    }
                                    else if ((LA(1)=='&') && (true)) {
                                          mAMPER(true);
                                          theRetToken=_returnToken;
                                    }
                              else {
                                    if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken
= makeToken(Token.EOF_TYPE);}
                              else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                              }
                              }
                              if ( _returnToken==null ) continue tryAgain; // found
SKIP token
                              _ttype = _returnToken.getType();
                              _returnToken.setType(_ttype);
                              return _returnToken;
                        }
                        catch (RecognitionException e) {
                              throw new TokenStreamRecognitionException(e);
                        }
                  }
                  catch (CharStreamException cse) {
                        if ( cse instanceof CharStreamIOException ) {
                              throw new
TokenStreamIOException(((CharStreamIOException)cse).io);
                        }
```

```
                    else {
                          throw new TokenStreamException(cse.getMessage());
                    }
              }
       }
}

       protected final void mDIGIT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = DIGIT;
              int _saveIndex;

              matchRange('0','9');
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       protected final void mALPHA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = ALPHA;
              int _saveIndex;

              switch ( LA(1)) {
              case 'A':  case 'B':  case 'C':  case 'D':
              case 'E':  case 'F':  case 'G':  case 'H':
              case 'I':  case 'J':  case 'K':  case 'L':
              case 'M':  case 'N':  case 'O':  case 'P':
              case 'Q':  case 'R':  case 'S':  case 'T':
              case 'U':  case 'V':  case 'W':  case 'X':
              case 'Y':  case 'Z':
              {
                    matchRange('A','Z');
                    break;
              }
              case 'a':  case 'b':  case 'c':  case 'd':
              case 'e':  case 'f':  case 'g':  case 'h':
              case 'i':  case 'j':  case 'k':  case 'l':
              case 'm':  case 'n':  case 'o':  case 'p':
              case 'q':  case 'r':  case 's':  case 't':
              case 'u':  case 'v':  case 'w':  case 'x':
              case 'y':  case 'z':
              {
                    matchRange('a','z');
                    break;
              }
              default:
              {
                    throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
              }
              }
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mLBRACKET(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = LBRACKET;
              int _saveIndex;

              match('[');
```

```
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRBRACKET(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RBRACKET;
            int _saveIndex;

            match(']');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LPAREN;
            int _saveIndex;

            match('(');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRPAREN(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RPAREN;
            int _saveIndex;

            match(')');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LBRACE;
            int _saveIndex;

            match('{');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mRBRACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = RBRACE;
```

```
            int _saveIndex;

            match('}');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOLON(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COLON;
            int _saveIndex;

            match(':');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mSEMICOLON(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = SEMICOLON;
            int _saveIndex;

            match(';');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mPLUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = PLUS;
            int _saveIndex;

            match('+');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mMINUS(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = MINUS;
            int _saveIndex;

            match('-');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }
```

```
	public final void mTIMES(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = TIMES;
		int _saveIndex;

		match('*');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mDIV(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = DIV;
		int _saveIndex;

		match('/');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mMOD(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = MOD;
		int _saveIndex;

		match('%');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mCOMMA(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = COMMA;
		int _saveIndex;

		match(',');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mUNDER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = UNDER;
		int _saveIndex;

		match('_');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
```

```
            _returnToken = _token;
      }

      public final void mEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = EQ;
            int _saveIndex;

            match("==");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NE;
            int _saveIndex;

            match("!=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mGT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = GT;
            int _saveIndex;

            match('>');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mLT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = LT;
            int _saveIndex;

            match('<');
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mGE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = GE;
            int _saveIndex;

            match(">=");
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
```

```
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mLE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = LE;
              int _saveIndex;

              match("<=");
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                     _token = makeToken(_ttype);
                     _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mAND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = AND;
              int _saveIndex;

              match("&&");
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                     _token = makeToken(_ttype);
                     _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mOR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = OR;
              int _saveIndex;

              match("||");
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                     _token = makeToken(_ttype);
                     _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = NOT;
              int _saveIndex;

              match('!');
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                     _token = makeToken(_ttype);
                     _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mAMPER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = AMPER;
              int _saveIndex;
```

```
		match('&');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mPOUND(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = POUND;
		int _saveIndex;

		match('#');
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mRANGE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = RANGE;
		int _saveIndex;

		match("->");
		if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
			_token = makeToken(_ttype);
			_token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
		}
		_returnToken = _token;
	}

	public final void mWHITESPACE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
		int _ttype; Token _token=null; int _begin=text.length();
		_ttype = WHITESPACE;
		int _saveIndex;

		{
		int _cnt32=0;
		_loop32:
		do {
			switch ( LA(1)) {
			case ' ':
			{
				match(' ');
				break;
			}
			case '\t':
			{
				match('\t');
				break;
			}
			default:
			{
				if ( _cnt32>=1 ) { break _loop32; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
			}
			}
			_cnt32++;
		} while (true);
		}
		if ( inputState.guessing==0 ) {
			_ttype = Token.SKIP;
```

```
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mNEWLINE(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = NEWLINE;
            int _saveIndex;

            {
            boolean synPredMatched36 = false;
            if (((LA(1)=='\r') && (LA(2)=='\n'))) {
                  int _m36 = mark();
                  synPredMatched36 = true;
                  inputState.guessing++;
                  try {
                        {
                        match('\r');
                        match('\n');
                        }
                  }
                  catch (RecognitionException pe) {
                        synPredMatched36 = false;
                  }
                  rewind(_m36);
                  inputState.guessing--;
            }
            if ( synPredMatched36 ) {
                  match('\r');
                  match('\n');
            }
            else if ((LA(1)=='\n')) {
                  match('\n');
            }
            else if ((LA(1)=='\r') && (true)) {
                  match('\r');
            }
            else {
                  throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
            }

            }
            if ( inputState.guessing==0 ) {
                  _ttype = Token.SKIP; newline();
            }
            if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                  _token = makeToken(_ttype);
                  _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
            }
            _returnToken = _token;
      }

      public final void mCOMMENT(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
            int _ttype; Token _token=null; int _begin=text.length();
            _ttype = COMMENT;
            int _saveIndex;

            {
            if ((LA(1)=='/') && (LA(2)=='*')) {
                  match("/*");
                  {
                  _loop42:
                  do {
```

```
                        // nongreedy exit test
                        if ((LA(1)=='*') && (LA(2)=='/')) break _loop42;
                        if ((_tokenSet_0.member(LA(1))) && ((LA(2) >=
'\u0003' && LA(2) <= '\u00ff'))) {
                                {
                                match(_tokenSet_0);
                                }
                        }
                        else if ((LA(1)=='\n'||LA(1)=='\r')) {
                                {
                                mNEWLINE(false);
                                }
                        }
                        else {
                                break _loop42;
                        }

                } while (true);
                }
                match("*/");
        }
        else if ((LA(1)=='/') && (LA(2)=='/')) {
                match("//");
                {
                _loop45:
                do {
                        if ((_tokenSet_0.member(LA(1)))) {
                                {
                                match(_tokenSet_0);
                                }
                        }
                        else {
                                break _loop45;
                        }

                } while (true);
                }
                {
                mNEWLINE(false);
                }
        }
        else {
                throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
        }

        }
        if ( inputState.guessing==0 ) {
                _ttype = Token.SKIP;
        }
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                _token = makeToken(_ttype);
                _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mID(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = ID;
        int _saveIndex;

        mALPHA(false);
        {
        _loop49:
        do {
                switch ( LA(1)) {
                case 'A':  case 'B':  case 'C':  case 'D':
                case 'E':  case 'F':  case 'G':  case 'H':
                case 'I':  case 'J':  case 'K':  case 'L':
```

```
                    case 'M':  case 'N':  case 'O':  case 'P':
                    case 'Q':  case 'R':  case 'S':  case 'T':
                    case 'U':  case 'V':  case 'W':  case 'X':
                    case 'Y':  case 'Z':  case 'a':  case 'b':
                    case 'c':  case 'd':  case 'e':  case 'f':
                    case 'g':  case 'h':  case 'i':  case 'j':
                    case 'k':  case 'l':  case 'm':  case 'n':
                    case 'o':  case 'p':  case 'q':  case 'r':
                    case 's':  case 't':  case 'u':  case 'v':
                    case 'w':  case 'x':  case 'y':  case 'z':
                    {
                          mALPHA(false);
                          break;
                    }
                    case '0':  case '1':  case '2':  case '3':
                    case '4':  case '5':  case '6':  case '7':
                    case '8':  case '9':
                    {
                          mDIGIT(false);
                          break;
                    }
                    case '_':
                    {
                          mUNDER(false);
                          break;
                    }
                    default:
                    {
                          break _loop49;
                    }
                    }
              } while (true);
              }
              _ttype = testLiteralsTable(_ttype);
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }

       public final void mNUMBER(boolean _createToken) throws
RecognitionException, CharStreamException, TokenStreamException {
              int _ttype; Token _token=null; int _begin=text.length();
              _ttype = NUMBER;
              int _saveIndex;

              {
              int _cnt52=0;
              _loop52:
              do {
                    if (((LA(1) >= '0' && LA(1) <= '9'))) {
                          mDIGIT(false);
                    }
                    else {
                          if ( _cnt52>=1 ) { break _loop52; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
getColumn());}
                    }

                    _cnt52++;
              } while (true);
              }
              if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
                    _token = makeToken(_ttype);
                    _token.setText(new String(text.getBuffer(), _begin,
text.length()-_begin));
              }
              _returnToken = _token;
       }
```

```
       private static final long[] mk_tokenSet_0() {
              long[] data = new long[8];
              data[0]=-9224L;
              for (int i = 1; i<=3; i++) { data[i]=-1L; }
              return data;
       }
       public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());

       }

------------------------------

/*
   c.def Language
   CDEFLookup: Value lookup table
   Author: Charles Catanach
 */

import java.util.Hashtable;

public class CDEFLookup
{
   Hashtable ht = null;
   CDEFLookup parent = null;

   public CDEFLookup(CDEFLookup parent)
   {
      this.parent = parent;

      ht = new Hashtable();

   }

   public Object lookup(String name)
   {
      Object result = ht.get(name);

      // recursively look up the value
      if(result == null && parent != null)
      {
         result = parent.lookup(name);
      }

      return result;
   }

   public void setRec(String name, Object obj)
   {
      Object result = ht.get(name);

      // recursively set the value
      if(result == null && parent != null)
      {
         parent.setRec(name, obj);
      }
      else
      {
         ht.put(name, obj);
      }
   }

   public void set(String name, Object obj, boolean isNew)
   {
      if(isNew)
      {
            // if this is a new identifier, set it in the current scope
         if(ht.get(name) == null)
         {
            ht.put(name, obj);
         }
         else
```

```
         {
            CDEF.printError("Identifier '" + name + "' is already in use.");
         }
      }
      else
      {
         setRec(name, obj);
      }
   }
}
-------------------------------

/*
   c.def Language
   CDEFObject: CDEF object abstraction
   Author: Charles Catanach
 */

public interface CDEFObject { }

-------------------------------

/*
   c.def Language
   CDEFParam: CDEF parameter abstraction
   Author: Charles Catanach
 */

public interface CDEFParam { }
//
-------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: "CDEFGrammar.g" -> "CDEFParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class CDEFParser extends antlr.LLkParser       implements
CDEFAntlrTokenTypes
 {

    int total_errors = 0;
    public void reportError( String s )
    {
        super.reportError( s );
        total_errors++;
    }
    public void reportError( RecognitionException e )
    {
        super.reportError( e );
        total_errors++;
    }
```

```
protected CDEFParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public CDEFParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected CDEFParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public CDEFParser(TokenStream lexer) {
  this(lexer,2);
}

public CDEFParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

       public final void startRule() throws RecognitionException,
TokenStreamException {

              returnAST = null;
              ASTPair currentAST = new ASTPair();
              AST startRule_AST = null;

              document();
              astFactory.addASTChild(currentAST, returnAST);
              match(Token.EOF_TYPE);
              startRule_AST = (AST)currentAST.root;
              returnAST = startRule_AST;
       }

       public final void document() throws RecognitionException,
TokenStreamException {

              returnAST = null;
              ASTPair currentAST = new ASTPair();
              AST document_AST = null;

              AST tmp2_AST = null;
              tmp2_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp2_AST);
              match(LITERAL_Document);
              AST tmp3_AST = null;
              tmp3_AST = astFactory.create(LT(1));
              astFactory.addASTChild(currentAST, tmp3_AST);
              match(ID);
              match(LBRACKET);
              coords();
              astFactory.addASTChild(currentAST, returnAST);
              match(COMMA);
              color();
              astFactory.addASTChild(currentAST, returnAST);
              match(COMMA);
              expr();
              astFactory.addASTChild(currentAST, returnAST);
              match(RBRACKET);
              code_body();
              astFactory.addASTChild(currentAST, returnAST);
              document_AST = (AST)currentAST.root;
              returnAST = document_AST;
```

```
	}

	public final void coords() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST coords_AST = null;

		match(AMPER);
		match(LPAREN);
		expr();
		astFactory.addASTChild(currentAST, returnAST);
		match(COMMA);
		expr();
		astFactory.addASTChild(currentAST, returnAST);
		match(RPAREN);
		coords_AST = (AST)currentAST.root;
		coords_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(COORDS,"COORDS")).add(coords_AST));
		currentAST.root = coords_AST;
		currentAST.child = coords_AST!=null
&&coords_AST.getFirstChild()!=null ?
			coords_AST.getFirstChild() : coords_AST;
		currentAST.advanceChildToEnd();
		coords_AST = (AST)currentAST.root;
		returnAST = coords_AST;
	}

	public final void color() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST color_AST = null;

		if ((LA(1)==POUND) && (LA(2)==ID)) {
			match(POUND);
			AST tmp13_AST = null;
			tmp13_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp13_AST);
			match(ID);
			color_AST = (AST)currentAST.root;
			color_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(COLOR_NAME,"COLOR_NAME")).add(color_AST));
			currentAST.root = color_AST;
			currentAST.child = color_AST!=null
&&color_AST.getFirstChild()!=null ?
				color_AST.getFirstChild() : color_AST;
			currentAST.advanceChildToEnd();
			color_AST = (AST)currentAST.root;
		}
		else if ((LA(1)==POUND) && (LA(2)==LPAREN)) {
			match(POUND);
			match(LPAREN);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(COMMA);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(COMMA);
			expr();
			astFactory.addASTChild(currentAST, returnAST);
			match(RPAREN);
			color_AST = (AST)currentAST.root;
			color_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(COLOR_RGB,"COLOR_RGB")).add(color_AST));
			currentAST.root = color_AST;
			currentAST.child = color_AST!=null
&&color_AST.getFirstChild()!=null ?
				color_AST.getFirstChild() : color_AST;
			currentAST.advanceChildToEnd();
```

```
                color_AST = (AST)currentAST.root;
            }
            else {
                throw new NoViableAltException(LT(1), getFilename());
            }

            returnAST = color_AST;
        }

        public final void expr() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST expr_AST = null;

            expr_not();
            astFactory.addASTChild(currentAST, returnAST);
            {
            switch ( LA(1)) {
            case AND:
            case OR:
            {
                {
                switch ( LA(1)) {
                case AND:
                {
                    AST tmp19_AST = null;
                    tmp19_AST = astFactory.create(LT(1));
                    astFactory.makeASTRoot(currentAST, tmp19_AST);
                    match(AND);
                    break;
                }
                case OR:
                {
                    AST tmp20_AST = null;
                    tmp20_AST = astFactory.create(LT(1));
                    astFactory.makeASTRoot(currentAST, tmp20_AST);
                    match(OR);
                    break;
                }
                default:
                {
                    throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                expr_not();
                astFactory.addASTChild(currentAST, returnAST);
                break;
            }
            case RBRACKET:
            case RPAREN:
            case COMMA:
            {
                break;
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
            }
            }
            expr_AST = (AST)currentAST.root;
            returnAST = expr_AST;
        }

        public final void code_body() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
```

```
		AST code_body_AST = null;

		match(LBRACE);
		{
		_loop57:
		do {
			switch ( LA(1)) {
			case LITERAL_Glyph:
			case LITERAL_Path:
			{
				declaration();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case ID:
			case LITERAL_line:
			case LITERAL_circle:
			case LITERAL_rect:
			case LITERAL_ellipse:
			case LITERAL_polygon:
			case LITERAL_color:
			case LITERAL_fillcolor:
			case LITERAL_Render:
			case LITERAL_Rotate:
			case LITERAL_Translate:
			case LITERAL_for:
			case LITERAL_break:
			case LITERAL_continue:
			case LITERAL_if:
			{
				statement();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			default:
			{
				break _loop57;
			}
			}
		} while (true);
		}
		match(RBRACE);
		code_body_AST = (AST)currentAST.root;
		code_body_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(BODY,"BODY")).add(code_body_AST));
		currentAST.root = code_body_AST;
		currentAST.child = code_body_AST!=null
&&code_body_AST.getFirstChild()!=null ?
			code_body_AST.getFirstChild() : code_body_AST;
		currentAST.advanceChildToEnd();
		code_body_AST = (AST)currentAST.root;
		returnAST = code_body_AST;
	}

	public final void declaration() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST declaration_AST = null;

		switch ( LA(1)) {
		case LITERAL_Glyph:
		{
			{
			AST tmp23_AST = null;
			tmp23_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp23_AST);
			match(LITERAL_Glyph);
			AST tmp24_AST = null;
			tmp24_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp24_AST);
```

```
                        match(ID);
                        match(LBRACKET);
                        coords();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RBRACKET);
                        glyph_body();
                        astFactory.addASTChild(currentAST, returnAST);
                        }
                        declaration_AST = (AST)currentAST.root;
                        break;
                }
                case LITERAL_Path:
                {
                        {
                        AST tmp27_AST = null;
                        tmp27_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp27_AST);
                        match(LITERAL_Path);
                        AST tmp28_AST = null;
                        tmp28_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp28_AST);
                        match(ID);
                        match(LBRACKET);
                        expr();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RBRACKET);
                        path_body();
                        astFactory.addASTChild(currentAST, returnAST);
                        }
                        declaration_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                returnAST = declaration_AST;
        }

        public final void statement() throws RecognitionException,
TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST statement_AST = null;

                switch ( LA(1)) {
                case ID:
                case LITERAL_line:
                case LITERAL_circle:
                case LITERAL_rect:
                case LITERAL_ellipse:
                case LITERAL_polygon:
                case LITERAL_color:
                case LITERAL_fillcolor:
                {
                        draw_statement();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                }
                case LITERAL_Render:
                case LITERAL_Rotate:
                case LITERAL_Translate:
                {
                        action_statement();
                        astFactory.addASTChild(currentAST, returnAST);
                        statement_AST = (AST)currentAST.root;
                        break;
                }
                case LITERAL_for:
```

```
		{
			for_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_if:
		{
			if_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_break:
		{
			break_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		case LITERAL_continue:
		{
			continue_statement();
			astFactory.addASTChild(currentAST, returnAST);
			statement_AST = (AST)currentAST.root;
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		returnAST = statement_AST;
	}

	public final void glyph_body() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST glyph_body_AST = null;

		match(LBRACE);
		{
		_loop63:
		do {
			if ((_tokenSet_0.member(LA(1)))) {
				statement();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop63;
			}

		} while (true);
		}
		match(RBRACE);
		glyph_body_AST = (AST)currentAST.root;
		glyph_body_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(BODY,"BODY")).add(glyph_body_AST));
		currentAST.root = glyph_body_AST;
		currentAST.child = glyph_body_AST!=null
&&glyph_body_AST.getFirstChild()!=null ?
			glyph_body_AST.getFirstChild() : glyph_body_AST;
		currentAST.advanceChildToEnd();
		glyph_body_AST = (AST)currentAST.root;
		returnAST = glyph_body_AST;
	}

	public final void path_body() throws RecognitionException,
TokenStreamException {
```

```
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST path_body_AST = null;

            match(LBRACE);
            draw_statement();
            astFactory.addASTChild(currentAST, returnAST);
            match(RBRACE);
            path_body_AST = (AST)currentAST.root;
            path_body_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(BODY,"BODY")).add(path_body_AST));
            currentAST.root = path_body_AST;
            currentAST.child = path_body_AST!=null
&&path_body_AST.getFirstChild()!=null ?
                  path_body_AST.getFirstChild() : path_body_AST;
            currentAST.advanceChildToEnd();
            path_body_AST = (AST)currentAST.root;
            returnAST = path_body_AST;
      }

      public final void draw_statement() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST draw_statement_AST = null;

            switch ( LA(1)) {
            case LITERAL_line:
            {
                  AST tmp35_AST = null;
                  tmp35_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp35_AST);
                  match(LITERAL_line);
                  match(LBRACKET);
                  coords();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(COMMA);
                  coords();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RBRACKET);
                  match(SEMICOLON);
                  draw_statement_AST = (AST)currentAST.root;
                  break;
            }
            case LITERAL_circle:
            {
                  AST tmp40_AST = null;
                  tmp40_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp40_AST);
                  match(LITERAL_circle);
                  match(LBRACKET);
                  coords();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(COMMA);
                  expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RBRACKET);
                  match(SEMICOLON);
                  draw_statement_AST = (AST)currentAST.root;
                  break;
            }
            case LITERAL_rect:
            {
                  AST tmp45_AST = null;
                  tmp45_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp45_AST);
                  match(LITERAL_rect);
                  match(LBRACKET);
                  coords();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(COMMA);
```

```
		coords();
		astFactory.addASTChild(currentAST, returnAST);
		match(RBRACKET);
		match(SEMICOLON);
		draw_statement_AST = (AST)currentAST.root;
		break;
	}
	case LITERAL_ellipse:
	{
		AST tmp50_AST = null;
		tmp50_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp50_AST);
		match(LITERAL_ellipse);
		match(LBRACKET);
		coords();
		astFactory.addASTChild(currentAST, returnAST);
		match(COMMA);
		expr();
		astFactory.addASTChild(currentAST, returnAST);
		match(COMMA);
		expr();
		astFactory.addASTChild(currentAST, returnAST);
		match(RBRACKET);
		match(SEMICOLON);
		draw_statement_AST = (AST)currentAST.root;
		break;
	}
	case LITERAL_polygon:
	{
		AST tmp56_AST = null;
		tmp56_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp56_AST);
		match(LITERAL_polygon);
		match(LBRACKET);
		coords();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop72:
		do {
			if ((LA(1)==COMMA)) {
				match(COMMA);
				coords();
				astFactory.addASTChild(currentAST, returnAST);
			}
			else {
				break _loop72;
			}

		} while (true);
		}
		match(RBRACKET);
		match(SEMICOLON);
		draw_statement_AST = (AST)currentAST.root;
		break;
	}
	case LITERAL_color:
	{
		AST tmp61_AST = null;
		tmp61_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp61_AST);
		match(LITERAL_color);
		match(LBRACKET);
		color();
		astFactory.addASTChild(currentAST, returnAST);
		match(RBRACKET);
		match(SEMICOLON);
		draw_statement_AST = (AST)currentAST.root;
		break;
	}
	case LITERAL_fillcolor:
	{
		AST tmp65_AST = null;
```

```
                        tmp65_AST = astFactory.create(LT(1));
                        astFactory.makeASTRoot(currentAST, tmp65_AST);
                        match(LITERAL_fillcolor);
                        match(LBRACKET);
                        color();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RBRACKET);
                        match(SEMICOLON);
                        draw_statement_AST = (AST)currentAST.root;
                        break;
                }
                case ID:
                {
                        AST tmp69_AST = null;
                        tmp69_AST = astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp69_AST);
                        match(ID);
                        match(LBRACKET);
                        coords();
                        astFactory.addASTChild(currentAST, returnAST);
                        match(RBRACKET);
                        match(SEMICOLON);
                        draw_statement_AST = (AST)currentAST.root;
                        draw_statement_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(INSERT,"INSERT")).add(draw_statement_AST));
                        currentAST.root = draw_statement_AST;
                        currentAST.child = draw_statement_AST!=null
&&draw_statement_AST.getFirstChild()!=null ?
                                draw_statement_AST.getFirstChild() :
draw_statement_AST;
                        currentAST.advanceChildToEnd();
                        draw_statement_AST = (AST)currentAST.root;
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                returnAST = draw_statement_AST;
        }

        public final void range() throws RecognitionException,
TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST range_AST = null;

                AST tmp73_AST = null;
                tmp73_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp73_AST);
                match(RANGE);
                match(LPAREN);
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(COMMA);
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                {
                switch ( LA(1)) {
                case COMMA:
                {
                        match(COMMA);
                        expr();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                }
                case RPAREN:
                {
                        break;
                }
```

```
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		match(RPAREN);
		range_AST = (AST)currentAST.root;
		returnAST = range_AST;
	}

	public final void action_statement() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST action_statement_AST = null;

		switch ( LA(1)) {
		case LITERAL_Render:
		{
			AST tmp78_AST = null;
			tmp78_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp78_AST);
			match(LITERAL_Render);
			match(LBRACKET);
			AST tmp80_AST = null;
			tmp80_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp80_AST);
			match(ID);
			match(COMMA);
			{
			switch ( LA(1)) {
			case RANGE:
			{
				range();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case LPAREN:
			case MINUS:
			case NOT:
			case ID:
			case NUMBER:
			{
				expr();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			default:
			{
				throw new NoViableAltException(LT(1), getFilename());
			}
			}
			}
			match(COMMA);
			{
			switch ( LA(1)) {
			case AMPER:
			{
				coords();
				astFactory.addASTChild(currentAST, returnAST);
				break;
			}
			case ID:
			{
				AST tmp83_AST = null;
				tmp83_AST = astFactory.create(LT(1));
				astFactory.addASTChild(currentAST, tmp83_AST);
				match(ID);
				break;
			}
```

```
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                match(RBRACKET);
                match(SEMICOLON);
                action_statement_AST = (AST)currentAST.root;
                break;
        }
        case LITERAL_Rotate:
        {
                AST tmp86_AST = null;
                tmp86_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp86_AST);
                match(LITERAL_Rotate);
                match(LBRACKET);
                AST tmp88_AST = null;
                tmp88_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp88_AST);
                match(ID);
                match(COMMA);
                expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(RBRACKET);
                match(SEMICOLON);
                action_statement_AST = (AST)currentAST.root;
                break;
        }
        case LITERAL_Translate:
        {
                AST tmp92_AST = null;
                tmp92_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp92_AST);
                match(LITERAL_Translate);
                match(LBRACKET);
                AST tmp94_AST = null;
                tmp94_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp94_AST);
                match(ID);
                match(COMMA);
                coords();
                astFactory.addASTChild(currentAST, returnAST);
                match(RBRACKET);
                match(SEMICOLON);
                action_statement_AST = (AST)currentAST.root;
                break;
        }
        default:
        {
                throw new NoViableAltException(LT(1), getFilename());
        }
        }
        returnAST = action_statement_AST;
    }

    public final void for_statement() throws RecognitionException,
TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST for_statement_AST = null;

        AST tmp98_AST = null;
        tmp98_AST = astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp98_AST);
        match(LITERAL_for);
        match(LBRACKET);
        AST tmp100_AST = null;
        tmp100_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp100_AST);
```

```
            match(ID);
            match(COLON);
            range();
            astFactory.addASTChild(currentAST, returnAST);
            match(RBRACKET);
            code_body();
            astFactory.addASTChild(currentAST, returnAST);
            for_statement_AST = (AST)currentAST.root;
            returnAST = for_statement_AST;
      }

      public final void if_statement() throws RecognitionException,
TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST if_statement_AST = null;

            AST tmp103_AST = null;
            tmp103_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp103_AST);
            match(LITERAL_if);
            match(LBRACKET);
            expr();
            astFactory.addASTChild(currentAST, returnAST);
            match(RBRACKET);
            code_body();
            astFactory.addASTChild(currentAST, returnAST);
            {
            switch ( LA(1)) {
            case LITERAL_else:
            {
                  match(LITERAL_else);
                  code_body();
                  astFactory.addASTChild(currentAST, returnAST);
                  break;
            }
            case RBRACE:
            case ID:
            case LITERAL_Glyph:
            case LITERAL_Path:
            case LITERAL_line:
            case LITERAL_circle:
            case LITERAL_rect:
            case LITERAL_ellipse:
            case LITERAL_polygon:
            case LITERAL_color:
            case LITERAL_fillcolor:
            case LITERAL_Render:
            case LITERAL_Rotate:
            case LITERAL_Translate:
            case LITERAL_for:
            case LITERAL_break:
            case LITERAL_continue:
            case LITERAL_if:
            {
                  break;
            }
            default:
            {
                  throw new NoViableAltException(LT(1), getFilename());
            }
            }
            }
            if_statement_AST = (AST)currentAST.root;
            returnAST = if_statement_AST;
      }

      public final void break_statement() throws RecognitionException,
TokenStreamException {

            returnAST = null;
```

```
		ASTPair currentAST = new ASTPair();
		AST break_statement_AST = null;

		AST tmp107_AST = null;
		tmp107_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp107_AST);
		match(LITERAL_break);
		AST tmp108_AST = null;
		tmp108_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp108_AST);
		match(SEMICOLON);
		break_statement_AST = (AST)currentAST.root;
		returnAST = break_statement_AST;
	}

	public final void continue_statement() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST continue_statement_AST = null;

		AST tmp109_AST = null;
		tmp109_AST = astFactory.create(LT(1));
		astFactory.makeASTRoot(currentAST, tmp109_AST);
		match(LITERAL_continue);
		AST tmp110_AST = null;
		tmp110_AST = astFactory.create(LT(1));
		astFactory.addASTChild(currentAST, tmp110_AST);
		match(SEMICOLON);
		continue_statement_AST = (AST)currentAST.root;
		returnAST = continue_statement_AST;
	}

	public final void expr_not() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST expr_not_AST = null;

		{
		switch ( LA(1)) {
		case NOT:
		{
			AST tmp111_AST = null;
			tmp111_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp111_AST);
			match(NOT);
			break;
		}
		case LPAREN:
		case MINUS:
		case ID:
		case NUMBER:
		{
			break;
		}
		default:
		{
			throw new NoViableAltException(LT(1), getFilename());
		}
		}
		}
		expr_compare();
		astFactory.addASTChild(currentAST, returnAST);
		expr_not_AST = (AST)currentAST.root;
		returnAST = expr_not_AST;
	}

	public final void expr_compare() throws RecognitionException,
TokenStreamException {
```

```
returnAST = null;
ASTPair currentAST = new ASTPair();
AST expr_compare_AST = null;

expr_add_sub();
astFactory.addASTChild(currentAST, returnAST);
{
switch ( LA(1)) {
case EQ:
case NE:
case GT:
case LT:
case GE:
case LE:
{
       {
       switch ( LA(1)) {
       case EQ:
       {
              AST tmp112_AST = null;
              tmp112_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp112_AST);
              match(EQ);
              break;
       }
       case NE:
       {
              AST tmp113_AST = null;
              tmp113_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp113_AST);
              match(NE);
              break;
       }
       case GT:
       {
              AST tmp114_AST = null;
              tmp114_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp114_AST);
              match(GT);
              break;
       }
       case LT:
       {
              AST tmp115_AST = null;
              tmp115_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp115_AST);
              match(LT);
              break;
       }
       case GE:
       {
              AST tmp116_AST = null;
              tmp116_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp116_AST);
              match(GE);
              break;
       }
       case LE:
       {
              AST tmp117_AST = null;
              tmp117_AST = astFactory.create(LT(1));
              astFactory.makeASTRoot(currentAST, tmp117_AST);
              match(LE);
              break;
       }
       default:
       {
              throw new NoViableAltException(LT(1), getFilename());
       }
       }
       }
```

```
                        expr_add_sub();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
                }
                case RBRACKET:
                case RPAREN:
                case COMMA:
                case AND:
                case OR:
                {
                        break;
                }
                default:
                {
                        throw new NoViableAltException(LT(1), getFilename());
                }
                }
                }
                expr_compare_AST = (AST)currentAST.root;
                returnAST = expr_compare_AST;
        }

        public final void expr_add_sub() throws RecognitionException,
TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST expr_add_sub_AST = null;

                expr_mul_div();
                astFactory.addASTChild(currentAST, returnAST);
                {
                _loop93:
                do {
                        switch ( LA(1)) {
                        case PLUS:
                        {
                                {
                                AST tmp118_AST = null;
                                tmp118_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp118_AST);
                                match(PLUS);
                                expr_mul_div();
                                astFactory.addASTChild(currentAST, returnAST);
                                }
                                break;
                        }
                        case MINUS:
                        {
                                {
                                AST tmp119_AST = null;
                                tmp119_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp119_AST);
                                match(MINUS);
                                expr_mul_div();
                                astFactory.addASTChild(currentAST, returnAST);
                                }
                                break;
                        }
                        default:
                        {
                                break _loop93;
                        }
                        }
                } while (true);
                }
                expr_add_sub_AST = (AST)currentAST.root;
                returnAST = expr_add_sub_AST;
        }

        public final void expr_mul_div() throws RecognitionException,
TokenStreamException {
```

```
		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST expr_mul_div_AST = null;

		expr_unary();
		astFactory.addASTChild(currentAST, returnAST);
		{
		_loop99:
		do {
			switch ( LA(1)) {
			case MOD:
			{
				{
				AST tmp120_AST = null;
				tmp120_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp120_AST);
				match(MOD);
				expr_unary();
				astFactory.addASTChild(currentAST, returnAST);
				}
				break;
			}
			case TIMES:
			{
				{
				AST tmp121_AST = null;
				tmp121_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp121_AST);
				match(TIMES);
				expr_unary();
				astFactory.addASTChild(currentAST, returnAST);
				}
				break;
			}
			case DIV:
			{
				{
				AST tmp122_AST = null;
				tmp122_AST = astFactory.create(LT(1));
				astFactory.makeASTRoot(currentAST, tmp122_AST);
				match(DIV);
				expr_unary();
				astFactory.addASTChild(currentAST, returnAST);
				}
				break;
			}
			default:
			{
				break _loop99;
			}
			}
		} while (true);
		}
		expr_mul_div_AST = (AST)currentAST.root;
		returnAST = expr_mul_div_AST;
	}

	public final void expr_unary() throws RecognitionException,
TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST expr_unary_AST = null;

		switch ( LA(1)) {
		case MINUS:
		{
			{
			match(MINUS);
			AST tmp124_AST = null;
			tmp124_AST = astFactory.create(LT(1));
```

```
                    astFactory.addASTChild(currentAST, tmp124_AST);
                    match(NUMBER);
                    }
                    expr_unary_AST = (AST)currentAST.root;
                    expr_unary_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(UMINUS,"UMINUS")).add(expr_unary_AST));
                    currentAST.root = expr_unary_AST;
                    currentAST.child = expr_unary_AST!=null
&&expr_unary_AST.getFirstChild()!=null ?
                           expr_unary_AST.getFirstChild() : expr_unary_AST;
                    currentAST.advanceChildToEnd();
                    expr_unary_AST = (AST)currentAST.root;
                    break;
             }
             case LPAREN:
             case ID:
             case NUMBER:
             {
                    atom();
                    astFactory.addASTChild(currentAST, returnAST);
                    expr_unary_AST = (AST)currentAST.root;
                    break;
             }
             default:
             {
                    throw new NoViableAltException(LT(1), getFilename());
             }
             }
             returnAST = expr_unary_AST;
      }

      public final void atom() throws RecognitionException,
TokenStreamException {

             returnAST = null;
             ASTPair currentAST = new ASTPair();
             AST atom_AST = null;

             switch ( LA(1)) {
             case LPAREN:
             {
                    {
                    match(LPAREN);
                    expr();
                    astFactory.addASTChild(currentAST, returnAST);
                    match(RPAREN);
                    }
                    atom_AST = (AST)currentAST.root;
                    break;
             }
             case NUMBER:
             {
                    AST tmp127_AST = null;
                    tmp127_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp127_AST);
                    match(NUMBER);
                    atom_AST = (AST)currentAST.root;
                    break;
             }
             case ID:
             {
                    AST tmp128_AST = null;
                    tmp128_AST = astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp128_AST);
                    match(ID);
                    atom_AST = (AST)currentAST.root;
                    break;
             }
             default:
             {
                    throw new NoViableAltException(LT(1), getFilename());
             }
```

```
        }
        returnAST = atom_AST;
    }


    public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "DIGIT",
        "ALPHA",
        "'['",
        "']'",
        "'('",
        "')'",
        "'{'",
        "'}'",
        "':'",
        "';'",
        ",",
        "PLUS",
        "MINUS",
        "TIMES",
        "DIV",
        "MOD",
        "COMMA",
        "UNDER",
        "EQ",
        "NE",
        "GT",
        "LT",
        "GE",
        "LE",
        "AND",
        "OR",
        "NOT",
        "AMPER",
        "POUND",
        "RANGE",
        "WHITESPACE",
        "NEWLINE",
        "COMMENT",
        "ID",
        "NUMBER",
        "DECL",
        "COORDS",
        "COLOR_NAME",
        "COLOR_RGB",
        "OBJINFO",
        "CMD",
        "BODY",
        "UMINUS",
        "INSERT",
        "\"Document\"",
        "\"Glyph\"",
        "\"Path\"",
        "\"line\"",
        "\"circle\"",
        "\"rect\"",
        "\"ellipse\"",
        "\"polygon\"",
        "\"color\"",
        "\"fillcolor\"",
        "\"Render\"",
        "\"Rotate\"",
        "\"Translate\"",
        "\"for\"",
        "\"break\"",
        "\"continue\"",
        "\"if\"",
        "\"else\""
    };
```

```
       protected void buildTokenTypeASTClassMap() {
              tokenTypeToASTClassMap=null;
       };

       private static final long[] mk_tokenSet_0() {
              long[] data = { -1125831187365888L, 0L};
              return data;
       }
       public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());

       }

-------------------------------

/*
   c.def Language
   CDEFPath: Path generation via parametric functions
   Authors: Dennis Rakhamimov, Charles Catanach
 */

import java.util.Vector;

public class CDEFPath implements CDEFObject
{
   private double pathPercentage;
   private int elementType = 0;
   private Vector elementData = null;

   private CDEFXY curPoint = null;
   private double tInc = 0;
   private double t = 0;

   int curSegmentIndex = 1;
   double curSegmentOffset = 0;
   double curSegmentStartLen = 0;
   double curSegmentStartX = 0;
   double curSegmentStartY = 0;


   double curX = 0;
   double curY = 0;
   double curLen = 0;
   double totalLength = 0;
   double stepLen = 0;

   public CDEFPath(int pathPercentage)
   {
         // Perform some error checking
      if(pathPercentage < 0 || pathPercentage > 100)
      {
         CDEF.printError("Invalid path percentage specified: "
               + pathPercentage);

      }

      this.pathPercentage = pathPercentage;
   }

   public void setPathInfo(int elementType, Vector elementData)
   {
      this.elementType = elementType;
      this.elementData = elementData;
   }

   /* initItr()
      Initializes the iterator
    */
   public void initItr(int steps)
   {
      if(elementType == CDEFElement.TYPE_CIRCULAR)
      {
```

```
      tInc = (2.0 * Math.PI) / (double)steps;
      t = (2.0 * Math.PI) * (pathPercentage / 100.0);
   }
   else
   {
      totalLength = 0;
      CDEFXY curVect = null;

      for(int i = 1 ; i < elementData.size(); i++)
      {
         curVect = (CDEFXY)elementData.get(i);

         totalLength += curVect.getLength();
      }

      double startLen = totalLength * (pathPercentage / 100.0);

      stepLen = (double)(totalLength - startLen) / (double)steps;

      CDEFXY origin = (CDEFXY)elementData.get(0);

      curX = 0;
      curY = 0;

      curSegmentStartX = 0;
      curSegmentStartX = 0;

      curLen = 0;

      /* skip to the correct starting segment based on the
         supplied percentage value */
      for(int i = 1; i < elementData.size(); i++)
      {
         curVect = (CDEFXY)elementData.get(i);

         if(curLen + curVect.getLength() >= startLen)
         {
            curSegmentIndex = i;
            curSegmentOffset = startLen - curLen;
            t = (startLen - curLen)/curVect.getLength();

            curLen = startLen;

            curX = curSegmentStartX + t * curVect.getX();
            curY = curSegmentStartY + t * curVect.getY();

            break;
         }

         curLen += curVect.getLength();
         curX += curVect.getX();
         curY += curVect.getY();

         curSegmentStartLen = curLen;
         curSegmentStartX = curX;
         curSegmentStartY = curY;
      }
   }
}

/* advanceItr()
   Advances the path iterator to the next translation point
 */
public CDEFXY advanceItr()
{
   CDEFXY result = null;

   if(elementType == CDEFElement.TYPE_CIRCULAR)
   {
      CDEFXY center = (CDEFXY)elementData.get(0);
      CDEFXY radius = (CDEFXY)elementData.get(1);
```

```
            int radiusX = radius.getIntX();
            int radiusY = radius.getIntY();

               // perform circular parametric calculation using sine and cosine
            result = new CDEFXY((int)(Math.cos(t) * radiusX + center.getX()),
                                (int)(Math.sin(t) * radiusY + center.getY())
                               );

            t += tInc;
         }
         else
         {
            CDEFXY curVect = null;
            CDEFXY center = (CDEFXY)elementData.get(0);

            /* iterate through as many polygonal segments as needed to
               achieve the needed step size */
            for(int i = curSegmentIndex; i < elementData.size(); i++)
            {
               curVect = (CDEFXY)elementData.get(i);

               // the new point fits in the current segment
               if(curLen + stepLen <= curSegmentStartLen + curVect.getLength())
               {
                  curSegmentIndex = i;

                  curLen += stepLen;

                  curSegmentOffset = curLen - curSegmentStartLen;
                  t = (curLen - curSegmentStartLen)/curVect.getLength();

                  curX = curSegmentStartX + t * curVect.getX();
                  curY = curSegmentStartY + t * curVect.getY();

                  break;
               }
               else
               {
                        // a segment is being skipped because it is too short
                  if(curSegmentStartLen == curLen)
                  {
                     curLen += curVect.getLength();
                     curX += curVect.getX();
                     curY += curVect.getY();

                     curSegmentStartLen = curLen;
                     curSegmentStartX = curX;
                     curSegmentStartY = curY;
                  }
                  // we're starting a new segment
                  else
                  {
                     curLen += curLen - curSegmentStartLen;
                     curX = curSegmentStartX + curVect.getX();
                     curY = curSegmentStartY + curVect.getY();

                     curSegmentStartLen = curLen;
                     curSegmentStartX = curX;
                     curSegmentStartY = curY;
                  }
               }
            }

            // produce the calculated translation
            result = new CDEFXY(curX + center.getX(), curY + center.getY());
         }

         return result;
      }
   }

   ------------------------------
```

```
/*
   c.def Language
   CDEFColor: From/to/step range representation
   Author: Charles Catanach
 */

public class CDEFRange implements CDEFParam
{
   int from, to, step;

   public CDEFRange(int from, int to, int step)
   {
      if(step == 0)
      {
         CDEF.printError("Range \"step\" value cannot be zero.");
      }

      if(from < to && step < 0)
      {
         CDEF.printError("Range \"step\" value must be positive, if \"from\" is
greater than \"to\"");
      }
      if(from > to && step > 0)
      {
         CDEF.printError("Range \"step\" value must be negative, if \"from\" is
less than \"to\"");
      }

      this.from = from;
      this.to = to;
      this.step = step;
   }

   public int getFrom()
   {
      return from;
   }

   public int getTo()
   {
      return to;
   }

   public int getStep()
   {
      return step;
   }
}

-------------------------------

/*
   c.def Language
   CDEFUtil: Tree walker utility functions
   Author: Dennis Rakhamimov
 */

import java.util.Stack;

public class CDEFUtil
{
   public Stack lookupTables = null;

   public CDEFDocument doc = null;

   public CDEFUtil()
   {
      lookupTables = new Stack();
   }

   public void setDocument(CDEFDocument doc)
```

```
{
   this.doc = doc;
}

/* enterScope()
   Pushes a new lookup hashtable onto the stack */
public void enterScope()
{
   CDEFLookup top = null;

   if(!lookupTables.empty())
   {
      top = (CDEFLookup)lookupTables.peek();
   }

   lookupTables.push(new CDEFLookup(top));
}

/* exitScope()
   Exits the current scope, dropping the topmost lookup table */
public void exitScope()
{
   lookupTables.pop();
}

/* setValue()
   Sets a value in the lookup table, recursively if needed.
 */
public void setValue(String name, Object obj, boolean isNew)
{
   ((CDEFLookup)lookupTables.peek()).set(name, obj, isNew);
}

public Object getValue(String name)
{
   Object result = ((CDEFLookup)lookupTables.peek()).lookup(name);

   if(result == null)
   {
      CDEF.printError("Identifier not found: " + name);
   }

   return result;
}

/* addToGlyph()
   Inserts a Glyph into another Glyph
 */
public void addToGlyph(CDEFGlyph parent, String glyphName, CDEFXY coords)
{
   CDEFGlyph glyph = null;

   try
   {
      glyph = (CDEFGlyph)getValue(glyphName);
   }
   catch(ClassCastException e)
   {
      CDEF.printError("Cannot insert non-Glyph object '"
               + glyphName + "'");
   }

   parent.appendGlyph(glyph, coords);
}

/* doRender()
   Performs a rendering of the specified glyph using the given
   parameters */
public void doRender(String glyphName, CDEFRange range,
            int frame, String pathName, CDEFXY origin)
{
   CDEFGlyph glyph = null;
```

```
      CDEFPath path = null;

      try
      {
         glyph = (CDEFGlyph)getValue(glyphName);
      }
      catch(ClassCastException e)
      {
         CDEF.printError("Cannot render non-Glyph object '"
                              + glyphName + "'");
      }

      int from, to, step;

      if(pathName != null)
      {
         try
         {
            path = (CDEFPath)getValue(pathName);
         }
         catch(ClassCastException e)
         {
            CDEF.printError("Cannot use non-Path object '"
                  + pathName + "' as a rendering path.");
         }
      }

      // obtain the from/to/step frame values from the parameter
      if(range != null)
      {
         from = range.getFrom() - 1;
         to = range.getTo() - 1;
         step = range.getStep();
      }
      else
      {
         from = frame - 1;
         to = frame - 1;
         step = 1;
      }

      // render the Glyph into the Document.
      doc.render(glyph, from, to, step, path, origin);

   }

   /* doRotate()
      Performs the rotation of a glyph.
    */
   public void doRotate(String glyphName, int angle)
   {
      CDEFGlyph glyph = null;

         try
         {
         glyph = (CDEFGlyph)getValue(glyphName);
      }
      catch(ClassCastException e)
      {
         CDEF.printError("Cannot rotate non-Glyph object '"
                              + glyphName + "'");
      }

      glyph.rotate(angle);
   }
}

------------------------------

/*
   c.def Language
   CDEFWalker.g: Tree Walker for ANTLR
```

```
   Author: Dennis Rakhamimov
 */

{

import java.util.Vector;

}

class CDEFWalker extends TreeParser;

options
{
    importVocab = CDEFAntlr;
    defaultErrorHandler=false;
}

{
   CDEFUtil ipt = new CDEFUtil();
   CDEFElementFactory factory = new CDEFElementFactory();
}

expr
[CDEFObject parent]
{
   CDEFObject r = null;
   CDEFParam p = null;

   CDEFParam a = null, b = null;
   CDEFObject obj = null;
   double c = 0,d = 0;

   String name = null, name2 = null;

   Vector v = null;
}
  : #("Document" name=id a=param b=param c=numericexp doc_body:.)
        {
           if(parent instanceof CDEFDocument)
           {
              CDEFDocument doc = (CDEFDocument)parent;

              doc.setParams((CDEFXY)a, (CDEFColor)b, (int)c);

              ipt.setDocument(doc);

              ipt.enterScope();

              expr( #doc_body, doc );

              ipt.exitScope();
           }
           else // This should never occur if the tree walker has been invoked
           {    // correctly
              CDEF.printError("Internal error.");
           }
        }
  | #("Glyph" name=id a=param glyph_body:. )
        {
           r = new CDEFGlyph((CDEFXY)a);

           ipt.setValue(name, r, true);
           ipt.enterScope();

           expr( #glyph_body, r );

           ipt.exitScope();
        }
  | #("Path" name=id c=numericexp path_body:. )
        {
           r = new CDEFPath((int)c);
```

```
        ipt.setValue(name, r, true);
        ipt.enterScope();

        expr( #path_body, r );

        ipt.exitScope();
      }
| #(INSERT name=id a=param)
       {  ipt.addToGlyph((CDEFGlyph)parent, name, (CDEFXY)a);  }
| #("line" a=param b=param)
       {  factory.createLine(parent, (CDEFXY)a, (CDEFXY)b); }
| #("circle" a=param c=numericexp)
       {  factory.createCircular(parent, (CDEFXY)a, (int)c, (int)c);  }
| #("rect" a=param b=param)
       {  factory.createRect(parent, (CDEFXY)a, (CDEFXY)b); }
| #("ellipse" a=param c=numericexp d=numericexp)
       {  factory.createCircular(parent, (CDEFXY)a, (int)c, (int)d); }
| #("polygon" { v = new Vector(10); } (b=param { v.add(b); } )+ )
       {  factory.createPoly(parent, v);                    }
| #("color" a=param)
       {  factory.setStrokeColor((CDEFColor)a);             }
| #("fillcolor" a=param)
       {  factory.setFillColor((CDEFColor)a);               }

| #("Render" name=id (a=param | c=numericexp) (name2=id | b=param))
      {
        ipt.doRender(name, (CDEFRange)a, (int)c, name2, (CDEFXY)b);
      }

| #("Rotate" name=id c=numericexp)
      {
        ipt.doRotate(name, (int)c);
      }

| #("for" name=id a=param for_body:. )
      {
        ipt.enterScope();

        int curValue = ( (CDEFRange) a).getFrom();
        ipt.setValue(name, new Integer(curValue), true);

        while(true)
        {
           ipt.enterScope();

           expr( #for_body, parent );

           if(curValue >= ( (CDEFRange) a).getTo() )
              break;

           curValue =  ((Integer)ipt.getValue(name)).intValue()
                     + ((CDEFRange)a).getStep();
           ipt.setValue(name, new Integer(curValue), false);


           ipt.exitScope();
        }

        ipt.exitScope();
      }

| #("if" c=numericexp if_body:. ( else_body:.)?  )
      {
         if(c != 0)
         {
            expr( #if_body, parent );
         }
         else
         {
            expr( #else_body, parent );
         }
      }
```

```
  |  #(BODY (expr [parent] )* )
  ;


id returns[String s]
{
    s = null;
}
    :  sval:ID  {  s = sval.getText();   }
    ;


param returns[CDEFParam p]
{
  p = null;
  double a, b, c = 1;
}
    :   #(RANGE a=numericexp b=numericexp (c=numericexp)?)
            { p = new CDEFRange((int)a, (int)b, (int)c); }

    |   #(COLOR_NAME colorName:ID)
            { p = new CDEFColor(colorName.getText()); }

    |   #(COLOR_RGB a=numericexp b=numericexp c=numericexp)
            { p = new CDEFColor((int)a, (int)b, (int)c); }

    |   #(COORDS a=numericexp b=numericexp)
            { p = new CDEFXY(a, b); }

    ;


numericexp returns[double r]
{
   double a, b;
   r = 0;
}
    :   #(PLUS a=numericexp b=numericexp)  {  r =  a +  b;  }
    |   #(MINUS a=numericexp b=numericexp) {  r =  a -  b;  }
    |   #(TIMES a=numericexp b=numericexp) {  r =  a *  b;  }
    |   #(UMINUS a=numericexp)             {  r = -a;       }
    |   #(DIV a=numericexp b=numericexp  ) {  r =  (double)a / (double)b;  }
    |   #(MOD a=numericexp b=numericexp)   {  r =  a %  b;  }
    |   #(EQ a=numericexp b=numericexp)    {  r =  (a == b) ? 1 : 0;  }
    |   #(NE a=numericexp b=numericexp)    {  r =  (a != b) ? 1 : 0;  }
    |   #(GT a=numericexp b=numericexp)    {  r =  (a >  b) ? 1 : 0;  }
    |   #(LT a=numericexp b=numericexp)    {  r =  (a <  b) ? 1 : 0;  }
    |   #(GE a=numericexp b=numericexp)    {  r =  (a >= b) ? 1 : 0;  }
    |   #(LE a=numericexp b=numericexp)    {  r =  (a <= b) ? 1 : 0;  }
    |   #(NOT a=numericexp)                {  r =  a;       }
    |   n:NUMBER   {  r = Integer.parseInt(n.getText());  }
    |   s:ID       {  Object eval = ipt.getValue(s.getText());
                      if(eval != null)
                         r = ((Integer)eval).intValue();
                      else
                      {
                          CDEF.printError("Identifier not found: "
                              + s.getText());
                      }
                   }
    ;

//
------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: "CDEFWalker.g" -> "CDEFWalker.java"$

import antlr.TreeParser;
import antlr.Token;
```

```
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;


import java.util.Vector;



public class CDEFWalker extends antlr.TreeParser       implements
CDEFWalkerTokenTypes
 {

   CDEFUtil ipt = new CDEFUtil();
   CDEFElementFactory factory = new CDEFElementFactory();
public CDEFWalker() {
       tokenNames = _tokenNames;
}

       public final void expr(AST _t,
              CDEFObject parent
       ) throws RecognitionException {

              AST expr_AST_in = (AST)_t;
              AST doc_body = null;
              AST glyph_body = null;
              AST path_body = null;
              AST for_body = null;
              AST if_body = null;
              AST else_body = null;

              CDEFObject r = null;
              CDEFParam p = null;

              CDEFParam a = null, b = null;
              CDEFObject obj = null;
              double c = 0,d = 0;

              String name = null, name2 = null;

              Vector v = null;


              if (_t==null) _t=ASTNULL;
              switch ( _t.getType()) {
              case LITERAL_Document:
              {
                     AST __t2 = _t;
                     AST tmp1_AST_in = (AST)_t;
                     match(_t,LITERAL_Document);
                     _t = _t.getFirstChild();
                     name=id(_t);
                     _t = _retTree;
                     a=param(_t);
                     _t = _retTree;
                     b=param(_t);
                     _t = _retTree;
                     c=numericexp(_t);
                     _t = _retTree;
                     doc_body = (AST)_t;
                     if ( _t==null ) throw new MismatchedTokenException();
                     _t = _t.getNextSibling();
                     _t = __t2;
                     _t = _t.getNextSibling();

                     if(parent instanceof CDEFDocument)
```

```
                        {
                        CDEFDocument doc = (CDEFDocument)parent;

                        doc.setParams((CDEFXY)a, (CDEFColor)b, (int)c);

                        ipt.setDocument(doc);

                        ipt.enterScope();

                        expr( doc_body, doc );

                        ipt.exitScope();
                        }
                        else // This should never occur if the tree walker has been
invoked
                        {    // correctly
                        CDEF.printError("Internal error.");
                        }

                        break;
                }
                case LITERAL_Glyph:
                {
                        AST __t3 = _t;
                        AST tmp2_AST_in = (AST)_t;
                        match(_t,LITERAL_Glyph);
                        _t = _t.getFirstChild();
                        name=id(_t);
                        _t = _retTree;
                        a=param(_t);
                        _t = _retTree;
                        glyph_body = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenException();
                        _t = _t.getNextSibling();
                        _t = __t3;
                        _t = _t.getNextSibling();

                        r = new CDEFGlyph((CDEFXY)a);

                        ipt.setValue(name, r, true);
                        ipt.enterScope();

                        expr( glyph_body, r );

                        ipt.exitScope();

                        break;
                }
                case LITERAL_Path:
                {
                        AST __t4 = _t;
                        AST tmp3_AST_in = (AST)_t;
                        match(_t,LITERAL_Path);
                        _t = _t.getFirstChild();
                        name=id(_t);
                        _t = _retTree;
                        c=numericexp(_t);
                        _t = _retTree;
                        path_body = (AST)_t;
                        if ( _t==null ) throw new MismatchedTokenException();
                        _t = _t.getNextSibling();
                        _t = __t4;
                        _t = _t.getNextSibling();

                        r = new CDEFPath((int)c);

                        ipt.setValue(name, r, true);
                        ipt.enterScope();

                        expr( path_body, r );

                        ipt.exitScope();
```

```
        break;
}
case INSERT:
{
        AST __t5 = _t;
        AST tmp4_AST_in = (AST)_t;
        match(_t,INSERT);
        _t = _t.getFirstChild();
        name=id(_t);
        _t = _retTree;
        a=param(_t);
        _t = _retTree;
        _t = __t5;
        _t = _t.getNextSibling();
        ipt.addToGlyph((CDEFGlyph)parent, name, (CDEFXY)a);
        break;
}
case LITERAL_line:
{
        AST __t6 = _t;
        AST tmp5_AST_in = (AST)_t;
        match(_t,LITERAL_line);
        _t = _t.getFirstChild();
        a=param(_t);
        _t = _retTree;
        b=param(_t);
        _t = _retTree;
        _t = __t6;
        _t = _t.getNextSibling();
        factory.createLine(parent, (CDEFXY)a, (CDEFXY)b);
        break;
}
case LITERAL_circle:
{
        AST __t7 = _t;
        AST tmp6_AST_in = (AST)_t;
        match(_t,LITERAL_circle);
        _t = _t.getFirstChild();
        a=param(_t);
        _t = _retTree;
        c=numericexp(_t);
        _t = _retTree;
        _t = __t7;
        _t = _t.getNextSibling();
        factory.createCircular(parent, (CDEFXY)a, (int)c, (int)c);
        break;
}
case LITERAL_rect:
{
        AST __t8 = _t;
        AST tmp7_AST_in = (AST)_t;
        match(_t,LITERAL_rect);
        _t = _t.getFirstChild();
        a=param(_t);
        _t = _retTree;
        b=param(_t);
        _t = _retTree;
        _t = __t8;
        _t = _t.getNextSibling();
        factory.createRect(parent, (CDEFXY)a, (CDEFXY)b);
        break;
}
case LITERAL_ellipse:
{
        AST __t9 = _t;
        AST tmp8_AST_in = (AST)_t;
        match(_t,LITERAL_ellipse);
        _t = _t.getFirstChild();
        a=param(_t);
        _t = _retTree;
        c=numericexp(_t);
```

```
                        _t = _retTree;
                        d=numericexp(_t);
                        _t = _retTree;
                        _t = __t9;
                        _t = _t.getNextSibling();
                        factory.createCircular(parent, (CDEFXY)a, (int)c, (int)d);
                        break;
                }
                case LITERAL_polygon:
                {
                        AST __t10 = _t;
                        AST tmp9_AST_in = (AST)_t;
                        match(_t,LITERAL_polygon);
                        _t = _t.getFirstChild();
                        v = new Vector(10);
                        {
                        int _cnt12=0;
                        _loop12:
                        do {
                                if (_t==null) _t=ASTNULL;
                                if ((_tokenSet_0.member(_t.getType()))) {
                                        b=param(_t);
                                        _t = _retTree;
                                        v.add(b);
                                }
                                else {
                                        if ( _cnt12>=1 ) { break _loop12; } else
{throw new NoViableAltException(_t);}
                                }

                                _cnt12++;
                        } while (true);
                        }
                        _t = __t10;
                        _t = _t.getNextSibling();
                        factory.createPoly(parent, v);
                        break;
                }
                case LITERAL_color:
                {
                        AST __t13 = _t;
                        AST tmp10_AST_in = (AST)_t;
                        match(_t,LITERAL_color);
                        _t = _t.getFirstChild();
                        a=param(_t);
                        _t = _retTree;
                        _t = __t13;
                        _t = _t.getNextSibling();
                        factory.setStrokeColor((CDEFColor)a);
                        break;
                }
                case LITERAL_fillcolor:
                {
                        AST __t14 = _t;
                        AST tmp11_AST_in = (AST)_t;
                        match(_t,LITERAL_fillcolor);
                        _t = _t.getFirstChild();
                        a=param(_t);
                        _t = _retTree;
                        _t = __t14;
                        _t = _t.getNextSibling();
                        factory.setFillColor((CDEFColor)a);
                        break;
                }
                case LITERAL_Render:
                {
                        AST __t15 = _t;
                        AST tmp12_AST_in = (AST)_t;
                        match(_t,LITERAL_Render);
                        _t = _t.getFirstChild();
                        name=id(_t);
                        _t = _retTree;
```

```
		{
		if (_t==null) _t=ASTNULL;
		switch ( _t.getType()) {
		case RANGE:
		case COORDS:
		case COLOR_NAME:
		case COLOR_RGB:
		{
			a=param(_t);
			_t = _retTree;
			break;
		}
		case PLUS:
		case MINUS:
		case TIMES:
		case DIV:
		case MOD:
		case EQ:
		case NE:
		case GT:
		case LT:
		case GE:
		case LE:
		case NOT:
		case ID:
		case NUMBER:
		case UMINUS:
		{
			c=numericexp(_t);
			_t = _retTree;
			break;
		}
		default:
		{
			throw new NoViableAltException(_t);
		}
		}
		}
		{
		if (_t==null) _t=ASTNULL;
		switch ( _t.getType()) {
		case ID:
		{
			name2=id(_t);
			_t = _retTree;
			break;
		}
		case RANGE:
		case COORDS:
		case COLOR_NAME:
		case COLOR_RGB:
		{
			b=param(_t);
			_t = _retTree;
			break;
		}
		default:
		{
			throw new NoViableAltException(_t);
		}
		}
		}
		_t = __t15;
		_t = _t.getNextSibling();

		ipt.doRender(name, (CDEFRange)a, (int)c, name2, (CDEFXY)b);

		break;
	}
	case LITERAL_Rotate:
	{
		AST __t18 = _t;
```

```
        AST tmp13_AST_in = (AST)_t;
        match(_t,LITERAL_Rotate);
        _t = _t.getFirstChild();
        name=id(_t);
        _t = _retTree;
        c=numericexp(_t);
        _t = _retTree;
        _t = __t18;
        _t = _t.getNextSibling();

        ipt.doRotate(name, (int)c);

        break;
}
case LITERAL_for:
{
        AST __t19 = _t;
        AST tmp14_AST_in = (AST)_t;
        match(_t,LITERAL_for);
        _t = _t.getFirstChild();
        name=id(_t);
        _t = _retTree;
        a=param(_t);
        _t = _retTree;
        for_body = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        _t = __t19;
        _t = _t.getNextSibling();

        ipt.enterScope();

        int curValue = ( (CDEFRange) a).getFrom();
        ipt.setValue(name, new Integer(curValue), true);

        while(true)
        {
        ipt.enterScope();

        expr( for_body, parent );

        if(curValue >= ( (CDEFRange) a).getTo() )
        break;

        curValue =  ((Integer)ipt.getValue(name)).intValue()
        + ((CDEFRange)a).getStep();
        ipt.setValue(name, new Integer(curValue), false);


        ipt.exitScope();
        }

        ipt.exitScope();

        break;
}
case LITERAL_if:
{
        AST __t20 = _t;
        AST tmp15_AST_in = (AST)_t;
        match(_t,LITERAL_if);
        _t = _t.getFirstChild();
        c=numericexp(_t);
        _t = _retTree;
        if_body = (AST)_t;
        if ( _t==null ) throw new MismatchedTokenException();
        _t = _t.getNextSibling();
        {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case DIGIT:
        case ALPHA:
```

```
case LBRACKET:
case RBRACKET:
case LPAREN:
case RPAREN:
case LBRACE:
case RBRACE:
case COLON:
case SEMICOLON:
case PLUS:
case MINUS:
case TIMES:
case DIV:
case MOD:
case COMMA:
case UNDER:
case EQ:
case NE:
case GT:
case LT:
case GE:
case LE:
case AND:
case OR:
case NOT:
case AMPER:
case POUND:
case RANGE:
case WHITESPACE:
case NEWLINE:
case COMMENT:
case ID:
case NUMBER:
case DECL:
case COORDS:
case COLOR_NAME:
case COLOR_RGB:
case OBJINFO:
case CMD:
case BODY:
case UMINUS:
case INSERT:
case LITERAL_Document:
case LITERAL_Glyph:
case LITERAL_Path:
case LITERAL_line:
case LITERAL_circle:
case LITERAL_rect:
case LITERAL_ellipse:
case LITERAL_polygon:
case LITERAL_color:
case LITERAL_fillcolor:
case LITERAL_Render:
case LITERAL_Rotate:
case LITERAL_Translate:
case LITERAL_for:
case LITERAL_break:
case LITERAL_continue:
case LITERAL_if:
case LITERAL_else:
{
       else_body = (AST)_t;
       if ( _t==null ) throw new MismatchedTokenException();
       _t = _t.getNextSibling();
       break;
}
case 3:
{
       break;
}
default:
{
       throw new NoViableAltException(_t);
```

```
                }
                }
                }
                _t = __t20;
                _t = _t.getNextSibling();

                if(c != 0)
                {
                expr( if_body, parent );
                }
                else
                {
                expr( else_body, parent );
                }

                break;
        }
        case BODY:
        {
                AST __t22 = _t;
                AST tmp16_AST_in = (AST)_t;
                match(_t,BODY);
                _t = _t.getFirstChild();
                {
                _loop24:
                do {
                        if (_t==null) _t=ASTNULL;
                        if ((_tokenSet_1.member(_t.getType()))) {
                                expr(_t,parent);
                                _t = _retTree;
                        }
                        else {
                                break _loop24;
                        }

                } while (true);
                }
                _t = __t22;
                _t = _t.getNextSibling();
                break;
        }
        default:
        {
                throw new NoViableAltException(_t);
        }
        }
        _retTree = _t;
}

public final String  id(AST _t) throws RecognitionException {
        String s;

        AST id_AST_in = (AST)_t;
        AST sval = null;

        s = null;


        sval = (AST)_t;
        match(_t,ID);
        _t = _t.getNextSibling();
        s = sval.getText();
        _retTree = _t;
        return s;
}

public final CDEFParam  param(AST _t) throws RecognitionException {
        CDEFParam p;

        AST param_AST_in = (AST)_t;
        AST colorName = null;
```

```
p = null;
double a, b, c = 1;


if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case RANGE:
{
       AST __t27 = _t;
       AST tmp17_AST_in = (AST)_t;
       match(_t,RANGE);
       _t = _t.getFirstChild();
       a=numericexp(_t);
       _t = _retTree;
       b=numericexp(_t);
       _t = _retTree;
       {
       if (_t==null) _t=ASTNULL;
       switch ( _t.getType()) {
       case PLUS:
       case MINUS:
       case TIMES:
       case DIV:
       case MOD:
       case EQ:
       case NE:
       case GT:
       case LT:
       case GE:
       case LE:
       case NOT:
       case ID:
       case NUMBER:
       case UMINUS:
       {
              c=numericexp(_t);
              _t = _retTree;
              break;
       }
       case 3:
       {
              break;
       }
       default:
       {
              throw new NoViableAltException(_t);
       }
       }
       }
       _t = __t27;
       _t = _t.getNextSibling();
       p = new CDEFRange((int)a, (int)b, (int)c);
       break;
}
case COLOR_NAME:
{
       AST __t29 = _t;
       AST tmp18_AST_in = (AST)_t;
       match(_t,COLOR_NAME);
       _t = _t.getFirstChild();
       colorName = (AST)_t;
       match(_t,ID);
       _t = _t.getNextSibling();
       _t = __t29;
       _t = _t.getNextSibling();
       p = new CDEFColor(colorName.getText());
       break;
}
case COLOR_RGB:
{
       AST __t30 = _t;
       AST tmp19_AST_in = (AST)_t;
```

```
                match(_t,COLOR_RGB);
                _t = _t.getFirstChild();
                a=numericexp(_t);
                _t = _retTree;
                b=numericexp(_t);
                _t = _retTree;
                c=numericexp(_t);
                _t = _retTree;
                _t = __t30;
                _t = _t.getNextSibling();
                p = new CDEFColor((int)a, (int)b, (int)c);
                break;
        }
        case COORDS:
        {
                AST __t31 = _t;
                AST tmp20_AST_in = (AST)_t;
                match(_t,COORDS);
                _t = _t.getFirstChild();
                a=numericexp(_t);
                _t = _retTree;
                b=numericexp(_t);
                _t = _retTree;
                _t = __t31;
                _t = _t.getNextSibling();
                p = new CDEFXY(a, b);
                break;
        }
        default:
        {
                throw new NoViableAltException(_t);
        }
        }
        _retTree = _t;
        return p;
}

public final double  numericexp(AST _t) throws RecognitionException {
        double r;

        AST numericexp_AST_in = (AST)_t;
        AST n = null;
        AST s = null;

        double a, b;
        r = 0;


        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case PLUS:
        {
                AST __t33 = _t;
                AST tmp21_AST_in = (AST)_t;
                match(_t,PLUS);
                _t = _t.getFirstChild();
                a=numericexp(_t);
                _t = _retTree;
                b=numericexp(_t);
                _t = _retTree;
                _t = __t33;
                _t = _t.getNextSibling();
                r =  a +  b;
                break;
        }
        case MINUS:
        {
                AST __t34 = _t;
                AST tmp22_AST_in = (AST)_t;
                match(_t,MINUS);
                _t = _t.getFirstChild();
                a=numericexp(_t);
```

```
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t34;
        _t = _t.getNextSibling();
        r =  a -  b;
        break;
}
case TIMES:
{
        AST __t35 = _t;
        AST tmp23_AST_in = (AST)_t;
        match(_t,TIMES);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t35;
        _t = _t.getNextSibling();
        r =  a *  b;
        break;
}
case UMINUS:
{
        AST __t36 = _t;
        AST tmp24_AST_in = (AST)_t;
        match(_t,UMINUS);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        _t = __t36;
        _t = _t.getNextSibling();
        r = -a;
        break;
}
case DIV:
{
        AST __t37 = _t;
        AST tmp25_AST_in = (AST)_t;
        match(_t,DIV);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t37;
        _t = _t.getNextSibling();
        r =  (double)a / (double)b;
        break;
}
case MOD:
{
        AST __t38 = _t;
        AST tmp26_AST_in = (AST)_t;
        match(_t,MOD);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t38;
        _t = _t.getNextSibling();
        r =  a %  b;
        break;
}
case EQ:
{
        AST __t39 = _t;
        AST tmp27_AST_in = (AST)_t;
        match(_t,EQ);
        _t = _t.getFirstChild();
```

```
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t39;
        _t = _t.getNextSibling();
        r =  (a == b) ? 1 : 0;
        break;
}
case NE:
{
        AST __t40 = _t;
        AST tmp28_AST_in = (AST)_t;
        match(_t,NE);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t40;
        _t = _t.getNextSibling();
        r =  (a != b) ? 1 : 0;
        break;
}
case GT:
{
        AST __t41 = _t;
        AST tmp29_AST_in = (AST)_t;
        match(_t,GT);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t41;
        _t = _t.getNextSibling();
        r =  (a >  b) ? 1 : 0;
        break;
}
case LT:
{
        AST __t42 = _t;
        AST tmp30_AST_in = (AST)_t;
        match(_t,LT);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t42;
        _t = _t.getNextSibling();
        r =  (a <  b) ? 1 : 0;
        break;
}
case GE:
{
        AST __t43 = _t;
        AST tmp31_AST_in = (AST)_t;
        match(_t,GE);
        _t = _t.getFirstChild();
        a=numericexp(_t);
        _t = _retTree;
        b=numericexp(_t);
        _t = _retTree;
        _t = __t43;
        _t = _t.getNextSibling();
        r =  (a >= b) ? 1 : 0;
        break;
}
case LE:
{
        AST __t44 = _t;
```

```
                AST tmp32_AST_in = (AST)_t;
                match(_t,LE);
                _t = _t.getFirstChild();
                a=numericexp(_t);
                _t = _retTree;
                b=numericexp(_t);
                _t = _retTree;
                _t = __t44;
                _t = _t.getNextSibling();
                r =  (a <= b) ? 1 : 0;
                break;
        }
        case NOT:
        {
                AST __t45 = _t;
                AST tmp33_AST_in = (AST)_t;
                match(_t,NOT);
                _t = _t.getFirstChild();
                a=numericexp(_t);
                _t = _retTree;
                _t = __t45;
                _t = _t.getNextSibling();
                r =  a;
                break;
        }
        case NUMBER:
        {
                n = (AST)_t;
                match(_t,NUMBER);
                _t = _t.getNextSibling();
                r = Integer.parseInt(n.getText());
                break;
        }
        case ID:
        {
                s = (AST)_t;
                match(_t,ID);
                _t = _t.getNextSibling();
                Object eval = ipt.getValue(s.getText());
                if(eval != null)
                r = ((Integer)eval).intValue();
                else
                {
                CDEF.printError("Identifier not found: "
                + s.getText());
                }

                break;
        }
        default:
        {
                throw new NoViableAltException(_t);
        }
        }
        _retTree = _t;
        return r;
}


public static final String[] _tokenNames = {
        "<0>",
        "EOF",
        "<2>",
        "NULL_TREE_LOOKAHEAD",
        "DIGIT",
        "ALPHA",
        "'['",
        "']'",
        "'('",
        "')'",
        "'{'",
        "'}'",
```

```
		"':'",
		"','",
		"PLUS",
		"MINUS",
		"TIMES",
		"DIV",
		"MOD",
		"COMMA",
		"UNDER",
		"EQ",
		"NE",
		"GT",
		"LT",
		"GE",
		"LE",
		"AND",
		"OR",
		"NOT",
		"AMPER",
		"POUND",
		"RANGE",
		"WHITESPACE",
		"NEWLINE",
		"COMMENT",
		"ID",
		"NUMBER",
		"DECL",
		"COORDS",
		"COLOR_NAME",
		"COLOR_RGB",
		"OBJINFO",
		"CMD",
		"BODY",
		"UMINUS",
		"INSERT",
		"\"Document\"",
		"\"Glyph\"",
		"\"Path\"",
		"\"line\"",
		"\"circle\"",
		"\"rect\"",
		"\"ellipse\"",
		"\"polygon\"",
		"\"color\"",
		"\"fillcolor\"",
		"\"Render\"",
		"\"Rotate\"",
		"\"Translate\"",
		"\"for\"",
		"\"break\"",
		"\"continue\"",
		"\"if\"",
		"\"else\""
	};

	private static final long[] mk_tokenSet_0() {
		long[] data = { 3852585664512L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
	private static final long[] mk_tokenSet_1() {
		long[] data = { -7494042556502638592L, 0L};
		return data;
	}
	public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
	}

//
------------------------------

/*
    Automatically Generated By ANTLR
```

```
*/
$ANTLR 2.7.2: "CDEFWalker.g" -> "CDEFWalker.java"$

public interface CDEFWalkerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int DIGIT = 4;
	int ALPHA = 5;
	int LBRACKET = 6;
	int RBRACKET = 7;
	int LPAREN = 8;
	int RPAREN = 9;
	int LBRACE = 10;
	int RBRACE = 11;
	int COLON = 12;
	int SEMICOLON = 13;
	int PLUS = 14;
	int MINUS = 15;
	int TIMES = 16;
	int DIV = 17;
	int MOD = 18;
	int COMMA = 19;
	int UNDER = 20;
	int EQ = 21;
	int NE = 22;
	int GT = 23;
	int LT = 24;
	int GE = 25;
	int LE = 26;
	int AND = 27;
	int OR = 28;
	int NOT = 29;
	int AMPER = 30;
	int POUND = 31;
	int RANGE = 32;
	int WHITESPACE = 33;
	int NEWLINE = 34;
	int COMMENT = 35;
	int ID = 36;
	int NUMBER = 37;
	int DECL = 38;
	int COORDS = 39;
	int COLOR_NAME = 40;
	int COLOR_RGB = 41;
	int OBJINFO = 42;
	int CMD = 43;
	int BODY = 44;
	int UMINUS = 45;
	int INSERT = 46;
	int LITERAL_Document = 47;
	int LITERAL_Glyph = 48;
	int LITERAL_Path = 49;
	int LITERAL_line = 50;
	int LITERAL_circle = 51;
	int LITERAL_rect = 52;
	int LITERAL_ellipse = 53;
	int LITERAL_polygon = 54;
	int LITERAL_color = 55;
	int LITERAL_fillcolor = 56;
	int LITERAL_Render = 57;
	int LITERAL_Rotate = 58;
	int LITERAL_Translate = 59;
	int LITERAL_for = 60;
	int LITERAL_break = 61;
	int LITERAL_continue = 62;
	int LITERAL_if = 63;
	int LITERAL_else = 64;
}
//
--------------------------------

/*
```

```
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: CDEFWalker.g -> CDEFWalkerTokenTypes.txt$
CDEFWalker    // output token vocab name
DIGIT=4
ALPHA=5
LBRACKET("'['")=6
RBRACKET("']'")=7
LPAREN("'('")=8
RPAREN("')'")=9
LBRACE("'{'")=10
RBRACE("'}'")=11
COLON("':'")=12
SEMICOLON("';'")=13
PLUS=14
MINUS=15
TIMES=16
DIV=17
MOD=18
COMMA=19
UNDER=20
EQ=21
NE=22
GT=23
LT=24
GE=25
LE=26
AND=27
OR=28
NOT=29
AMPER=30
POUND=31
RANGE=32
WHITESPACE=33
NEWLINE=34
COMMENT=35
ID=36
NUMBER=37
DECL=38
COORDS=39
COLOR_NAME=40
COLOR_RGB=41
OBJINFO=42
CMD=43
BODY=44
UMINUS=45
INSERT=46
LITERAL_Document="Document"=47
LITERAL_Glyph="Glyph"=48
LITERAL_Path="Path"=49
LITERAL_line="line"=50
LITERAL_circle="circle"=51
LITERAL_rect="rect"=52
LITERAL_ellipse="ellipse"=53
LITERAL_polygon="polygon"=54
LITERAL_color="color"=55
LITERAL_fillcolor="fillcolor"=56
LITERAL_Render="Render"=57
LITERAL_Rotate="Rotate"=58
LITERAL_Translate="Translate"=59
LITERAL_for="for"=60
LITERAL_break="break"=61
LITERAL_continue="continue"=62
LITERAL_if="if"=63
LITERAL_else="else"=64

-------------------------------

/*
   c.def Language
   CDEFXY: XY coordinate or vector container
   Author: Charles Catanach
```

```
 */

public class CDEFXY implements CDEFParam
{
   double x, y, length = -1;


   public CDEFXY(double x, double y)
   {
      this.x = x;
      this.y = y;

   }

   public double getLength()
   {
      // store the length after first access
      if(length == -1)
      {
         length = Math.sqrt(x * x + y * y);
      }

      return length;
   }

   public String toString()
   {

      return("(XY: " + x + ", " + y + ")");
   }

   public double getX()
   {
      return x;

   }

   public double getY()
   {
      return y;

   }

   public int getIntX()
   {
      return (int)x;
   }

   public int getIntY()
   {
      return (int)y;
   }
   public void setXY(double x, double y)
   {
      this.x = x;
      this.y = y;

   }
}//
-------------------------------

/*
    Automatically Generated By ANTLR
*/
$ANTLR 2.7.2: test.g -> PTokenTypes.txt$
P    // output token vocab name
SourceCode=4
WhiteSpace=5
NewLine=6
Digit=7
RangeSymbol=8
Num=9
```

```
FrameSpecifier=10
Frame=11
```

8.2 Examples

```
/*
   c.def Language Example
   Simple Clock Animation
   Author: Eric Poirier
   Date:   11/30/2003
 */

Document d[&(300, 300), #White, 1080]
{
   // Unit tick
   Glyph tick[&(0, 0)]
   {
      line[&(100, 0), &(110, 0)];
   }

   // Clock face
   Glyph face[&(150, 150)]
   {
      fillcolor[#Blue];
      circle[&(150, 150), 140];
      fillcolor[#(200, 200, 255)];
      circle[&(150, 150), 135];

      for[ i : ->(1, 12) ]
      {
         tick[&(150, 150)];
         Rotate[tick, 30];
      }
   }

   // Hours hand
   Glyph hourHand[&(10, 60)]
   {
      fillcolor[#Red];
      circle[&(10, 60), 5];

      fillcolor[#None];

      line[&( 2, 54), &(10, 0)];
      line[&(18, 54), &(10, 0)];
   }

   // Minutes hand
   Glyph minuteHand[&(10, 100)]
   {
      fillcolor[#Red];
      circle[&(10, 100), 5];

      fillcolor[#None];

      line[&( 6, 94), &(10, 0)];
      line[&(14, 94), &(10, 0)];
   }

   // Display the clock face
   Render[face, ->(1, 1080), &(150, 150)];


   // Display the rotation of the hours hand
   for[ j : ->(1, 1080, 3) ]
   {
      Render[hourHand, ->(j, j+2), &(150, 150)];
      Rotate[hourHand, 1];
```

```
   }

   // Display the rotation of the minutes hand
   for[ j : ->(1, 1080) ]
   {
      Render[minuteHand, j, &(150, 150)];
      Rotate[minuteHand, 4];

   }
}

----------------------------------------

/*
   c.def Language Example
   Test of all built-in colors
   Author: Charles Catanach
 */

Document d[&(70, 190), #White, 1]
{
   Glyph colors[&(0, 0)]
   {
      color[#Black];

      fillcolor[#None];
      rect[&(0, 0), &(20, 10)];

      fillcolor[#White];
      rect[&(0, 20), &(20, 30)];

      fillcolor[#Black];
      rect[&(30, 20), &(50, 30)];

      fillcolor[#Red];
      rect[&(0, 40), &(20, 50)];

      fillcolor[#Green];
      rect[&(30, 40), &(50, 50)];

      fillcolor[#Blue];
      rect[&(0, 60), &(20, 70)];

      fillcolor[#Yellow];
      rect[&(30, 60), &(50, 70)];

      fillcolor[#Purple];
      rect[&(0, 80), &(20, 90)];

      fillcolor[#Navy];
      rect[&(30, 80), &(50, 90)];

      fillcolor[#Gray];
      rect[&(0, 100), &(20, 110)];

      fillcolor[#Aqua];
      rect[&(30, 100), &(50, 110)];

      fillcolor[#Teal];
      rect[&(0, 120), &(20, 130)];

      fillcolor[#Olive];
      rect[&(30, 120), &(50, 130)];

      fillcolor[#Lime];
      rect[&(0, 140), &(20, 150)];

      fillcolor[#Maroon];
      rect[&(30, 140), &(50, 150)];

      fillcolor[#Fuchsia];
      rect[&(0, 160), &(20, 170)];
```

```
      fillcolor[#Silver];
      rect[&(30, 160), &(50, 170)];
   }


   Render[colors, 1, &(10, 10)];
}

---------------------------------------

/*
   c.def Language Example
   Flash animation of a Ferris Wheel
   Author: Dennis Rakhamimov
   Date:   12/16/2003
 */

/* Create a Flash animation with width=400, height=400 and white
   as the background color. The total number of frames is 180. */
Document d[&(400, 400), #White, 180]
{
   /* Ferris wheel base */
   Glyph base[&(100, 0)]
   {
      // the actual base
      fillcolor[#(215, 155, 251)];
      polygon[&(100, 0), &(-100,170), &(0, 10), &(200, 0),
              &(0, -10), &(-100, -170)];
      fillcolor[#None];


      // letter: T
      polygon[&(-60, -170), &(20, 0), &(-10, 0), &(0, 21)];

      // letter: o
      circle[&(-35, -154), 5];

      // letter: S
      polygon[&(-15, -154), &(0,2), &(2,2), &(10, 0), &(3,-3), &(0, -5),
              &(-2, -2), &(-10, 0), &(-3, -3), &(0, -4), &(3, -3),
              &(8, 0), &(2,2)];

      // period
      circle[&(5, -150), 1];

      // letter: E
      polygon[&(25, -170), &(-15, 0), &(0, 10), &(10, 0), &(-10, 0),
              &(0, 10), &(15, 0)];

      // period
      circle[&(30, -150), 1];
   }

   // Display the base
   Render [base, ->( 1, 180 ), &(200, 200)];

   /* A wheel's spike */
   Glyph spike[&(0, 0)]
   {
      line [&(20, 0), &(120, 0)];
      line [&(120, 0), &(58, 105)];
   }

   /* Define the wheel */
   Glyph wheel[&(150, 150)]
   {
      for[ i: ->( 1, 6 ) ]
      {
         Rotate [spike, 60];

         spike [&(150, 150)];
```

```
   }

   // outer circle
   fillcolor[#(0, 128, 255)];
   circle[&(150, 150), 125];

   // inner circle
   fillcolor[#White];
   circle[&(150, 150), 120];

   // small circle in the center
   fillcolor[#Red];
   circle[&(150, 150), 20];

   fillcolor[#None];
}


/* Render the rotation of the wheel over 180 frames */
for[ i : -> ( 1, 180) ]
{
   Render [wheel, i, &(200, 200)]; /* Place wheel on frame i */
   Rotate [wheel, 2]; /* Rotate wheel by 2 degrees */
}

/* One style of Ferris wheel car */
Glyph ferrisCar[&(25, 5)]
{
   fillcolor[#Yellow];

   // The base
   polygon[&(0, 15),
           &(-5, 15),
           &(5, 5),
           &(50, 0),
           &(5, -5),
           &(-5, -15), &(-50, 0)];

   // The door
   fillcolor[#None];
   color[#(232, 186, 47)];
   rect[&(10, 17), &(40, 32)];

   color[#Black];

   // The supporting axes
   line [&(0, 15), &(20, 5)];
   line [&(50, 15), &(30, 5)];
   fillcolor[#Red];
   circle[&(25, 5), 5];
   fillcolor[#None];
}

/* Another style of Ferris wheel car */
Glyph ferrisCar2[&(25, 5)]
{
   fillcolor[#Lime];

   // The base
   polygon[&(0, 15),
           &(-5, 15),
           &(5, 5),
           &(50, 0),
           &(5, -5),
           &(-5, -15), &(-50, 0)];

   fillcolor[#None];
   color[#(0, 186, 0)];

   // The door
   rect[&(10, 17), &(40, 32)];
```

```
      color[#Black];

      // The supporting axes
      line [&(0, 15), &(20, 5)];
      line [&(50, 15), &(30, 5)];
      fillcolor[#Red];
      circle[&(25, 5), 5];
      fillcolor[#None];
   }

   for[ i : ->(1, 6) ]
   {
      // Create a path for the cars
      Path circularPath[i * (100/6)]
      {
         circle[&(200, 200), 120];
      }

      /* Display the Ferris wheel cars along a circle,
         alternating their color styles */
      if[ i % 2 == 0 ]
      {
         Render [ferrisCar, ->(1, 180), circularPath];
      }
      else
      {
         Render [ferrisCar2, ->(1, 180), circularPath];
      }
   }
}

----------------------------------------

/*
   c.def Language Example
   Flash animation of a delicious Guinness
   Author: Tecuan Flores
   Date:   12/13/2003
 */

Document d[&(400, 500), #White, 160]
{

   Glyph glass[&(100, 0)]
   {
      color[#Black];
      polygon[&(0, 0), &(200,0), &(-30, 300), &(-140, 0), &(-30, -300)];
      color[#None];
   }

   Glyph guinness[&(100,0)]
   {
      for[ i : -> (0, 110) ]
      {
         /* Polygon for the foam */
         fillcolor[#(255, 255, 153)];
         polygon[&(30, 300), &(140,0), &(30*(11/10)*i/120, -300*(11/10)*i/120),
         &(-140-60*(11/10)*i/120, 0), &(30*(11/10)*i/120, 300*(11/10)*i/120)];

         /* Polygon for the liquid */
         fillcolor[#Black];
         polygon[&(30, 300), &(140,0), &(30*i/120, -300*i/120),
         &(-140-60*i/120, 0), &(30*i/120, 300*i/120)];

         /* Render a little longer if glass is full */
         if[ i  == 110 ] {
            Render [guinness, ->(i+1,i+50), &(200, 100)];
            Render [glass, ->( i+1, i+50), &(200, 100)];

         } else { /* Render one frame */
           Render [guinness, ->(i+1,i+2), &(200, 100)];
            Render [glass, ->( i+1, i+2), &(200, 100)];
```

```
        }

      }

     fillcolor[#None];
  }

}

---------------------------------------

/*
   c.def Language Example
   Flash animation for toilet
   Author: Tecuan Flores
   Date:   12/13/2003
 */

Document d[&(500, 500), #White, 300]
{

   Glyph toiletbowl[&(100, 0)]
   {
      color[#Black];

      fillcolor[#(255, 255, 153)];
      circle[&(150, 150), 75];
      fillcolor[#(135, 197, 220)];

      circle[&(150, 150), 60];
      color[#None];
      fillcolor[#None];

   }

   Glyph toiletbowlback[&(100, 0)]
   {
      color[#Black];
      fillcolor[#(255, 255, 153)];
      polygon[&(65, 50), &(170, 0), &(0, 50), &(-170, 0),&(0, -50)];
      color[#None];
      fillcolor[#None];
   }

   Render [toiletbowl, ->(1, 300), &(100,100)];
   Render [toiletbowlback, ->(1, 300), &(100,100)];


   Glyph pipe[&(100, 0)]
   {
      color[#Black];
     //fillcolor[#(192, 192, 192)];
      polygon[&(144, 225), &(0, 50), &(112, 0)];
      polygon[&(156, 225), &(0, 38), &(100, 0)];
      color[#None];
      fillcolor[#None];
   }

   Render [pipe, ->(1, 300), &(100,100)];

   Glyph object[&(100, 0)]
   {
      color[#Black];
      fillcolor[#(120, 65, 24)];
      circle[&(150, 150), 5];
      color[#None];
      fillcolor[#None];
   }


   for[ i : ->(1, 8) ]
   {
```

```
      Path circularPath[50]
      {
        circle[&(100, 105), 50/i];
      }
    Render [object, ->((i-1)*20, i*20), circularPath];
  }

  Path verticalPath[0]
  {
      polygon [&(100, 175), &(0, 43), &(80, 0)];
  }
  Render [object, ->(170, 300), verticalPath];




}
```

8.3 Citations

[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