COMS W4115
Programming Languages and Translators

Google Earth Script Language
Language Reference Manual

10/18/2006

Carlos Icaza
cai2101@columbia.edu
Darrell Tang
tt2191@columbia.edu
Wei Chung Hsu
wh2138@columbia.edu
1. Language Conventions

1.1. Comments
This is same as C/C++/Java comment format. We support 2 kinds of comments. First one is multi-lines comments that starts from “/∗” and end at “∗/". Second one is single-line comment that start from “//” until the end of this line.

1.2. Identifiers (Variable, Function, and Object names)
We use the identifiers to name variables, functions and objects. The identifier is a terminal in our grammar. The following is our identifier rule:
letter: _|A|B|...|Z|a|...|z
digit: 0|1|...|9
identifier: letter ( letter | digit )*

1.3. Numbers
A number could be an integer or floating point. It consists of digits and optional decimal point “.” as well as digits. In order to make our language simple, we don’t support exponent number now.
number: (digit)+ (. (digit)+ )?

1.4. Strings
A string is a sequence of characters enclosed by double quotes. Basically, the character here means the 256 types of character in ASCII code.
String: “ (character)* “

1.5. NULL
GECL has NULL value which is same as C/C++.

1.6. Operators

1.7. Keywords

| IS   | string   | While | false | int       |
| Km   | m        | Nm    | mil   | polygon   |
| struct | Coord    | Dist  | dir   | square    |
| If   | Break    | Return| float | else      |
| continue | NULL   | Boolean | for   | true      |
| circle | Hexagon  | pentagon | rectangle | octagon |
| semicircle | perspective | coord | overlay | lookat    |
| folder | LOOKS_AT | Print | line  | feet      |
1.8. Tokens to terminate or separate blocks
   { } ( ) , [ ] ;

2. Types/Objects
   In GECL, we use type followed by object name to allocate an object, and we process this
   object by this object name. There are several type keywords GECL support which is list in
   chapter 1.7. These objects can be allocated like a standard type (i.e.: object
   myObject; ), but require either the IS keyword or assignment operator
   to be defined.

2.1. dist
   Defines a length, that can be given in miles, meters, kilometers feet and nautical miles.
   Instantiated and manipulated like an int.

2.2. dir
   Defines a bearing in degrees. Used to create a new coord object by moving from
   another.

2.3. coord
   The basic geographic object from which all others are built. Defines latitude,
   longitude an elevation of a specific geographical point.

2.6. point
   Creates a basic ‘pop up’ point at a certain coord, with title, comments and
   URL for links or for loading photos.

2.5. Struct
   This is the most important type in GECL used to construct a 3D object which could
   be a building, a car or a polygon.

2.6. Place
   This is the type to allocate a placemark object. The placemark consists of several
   struct objects, overlays and so on, grouped so as to be related to perspective and
   folder objects

2.7 Overlay
   This is the type to allocate an overlay object so that we could specify the area we are
   going to look at with coordinates, photo URL and rotation.

2.8. Perspective
This type specifies where and how the viewer will look at a struct, coord, overlay, line and place object.

2.9. Folder
This type collects place, struct, point, line and overlay objects under a single GoogleEarth ‘folder’ for easy navigation between them. The object names will become the titles for these objects inside the folder, except for point, which specifies its own title.

2.10. Line
This is the type to allocate a line object. The reason to have this object is because we want to have some line mark in the Google Earth instead of using pure 3D objects.

2.11. Array
GECL has array data type which is pretty similar to C/C++/Java array data type. However, in order to make our language easier to implement and use, we only provide bounded array allocation, such as int a[10], which will allocated in the compiling time.

3. Expression
Our GECL is a language that consists of several expressions. The expression could be a list of integers, a boolean number, an identifier or a function call. The following is our expression grammar.

\[
\begin{align*}
\text{expr} & : \text{int-lit} \\
& \mid \text{bool-lit} \\
& \mid \text{identifier} \\
& \mid \text{identifier} = \text{expr} \\
& \mid \text{var} \text{ post-op} \\
& \mid \text{expr} \ \text{bin-op} \ \text{expr} \\
& \mid \text{if} \ \text{expr} \ \text{then} \ \text{expr} \ \text{else} \ \text{expr} \ \text{end} \\
& \mid ( \ \text{expr} \ )
\end{align*}
\]

\[
\begin{align*}
\text{bool-lit} & : \text{true} \ | \ \text{false} \\
\text{int-lit} & : ( \ \text{digit} \ )^+ \\
\text{bin-op} & : \text{arith-op} | \text{rel-op} | \text{bool-op} \\
\text{arith-op} & : + | - | \ast | / | \% \\
\text{rel-op} & : == | < | > \\
\text{bool-op} & : ^ | | \\
\text{post-op} & : ++ | -- | ~
\end{align*}
\]
4. Statements
Semicolon “;” is a statement terminator in GECL. Basically, statements are executed in sequence except specified control-flow statements. Statements can separate into several categories listed below.

statement:
  expression-statement
  compound-statement
  iteration-statement
  conditional-statement

4.1 Expression statement
expression-statement:
  expression_opt ;
An expression becomes a statement when it is followed by a semicolon.

4.1.1 Assign statement
assign-statement:
  primary-expression assignment-operator expression;
Assign-statements are the majority of expression statements. Basically, these statements assign the right side expression’s final value to left side.
For example:
  foo = bar + 10 ;

4.2 Compound statement
compound-statement:
  { declaration-list_opt statement-list_opt };
declaration-list:
  declaration
  declaration-list declaration
statement-list
  statement
  statement-list
A compound statement is composed of declarations and statements by put them into a pair of brackets. The body of a function definition is a compound statement.

4.3 Iteration statement
iteration-statement:
  for (expression_opt ; expression_opt ; expression_opt ) statement
Iteration statements use to loop a statement or a block of statements several times. In this for-loop statement, three expressions separated by semicolon are all optional. The first expression which must have arithmetic type is evaluated once, and thus gives the initial state for the loop. The second expression is evaluated before each iteration, and if it becomes equal to zero (false), the iteration is terminate. The third expression is evaluated after each iteration and thus gives a new state for the loop.

4.3.1 Break statement
4.3.2 Continue statement

```
continue-statement:
    continue;
```

Continue statement only happens inside an iteration statement. It causes control to pass through current loop (the smallest enclosing) and directly go to next loop’s evaluation.

4.4 Conditional statement

```
conditional-statement:
    if (expression) statement
    if (expression) statement else statement
```

The expression part of the conditional statement must have arithmetic type. If its value compares unequal to zero then the first statement is executed. In the second form of conditional statement, the second statement is executed if the expression’s value is zero.

5. Function Definition

```
function-definition:
    function identifier (arg1, arg2, ...) compound-statement
```

`identifier` indicates the name of the function. `args` are the arguments for the function which are optional separated by commas ','. `compound-statement` is the block of code that will be executed when the function been called.

6. User defined functions

The user will be able to declare and define his own functions using a format resembling that of C/C++/Java:

```
func <return type> <function name>(<parameter list>){
    <statement>
        .
        .
        .
}
```

7. Built in functions

These are almost entirely the ones required to define the types, the equivalent of *constructor methods* in Java.
**dist**

\[
\text{dist <object\_name> } = x[m \mid km \mid mi \mid ft \mid nm]; \\
\text{dist}(x[m \mid km \mid mi \mid ft \mid nm]);
\]

Defines a new dist object (first form), or instantiates a new one for a function parameter if needed (second form). In both forms the distance needs to be written like a float or int together with a modifier specifying in which measure the distance is being given: meters, kilometers, miles, feet or nautical miles. By default it is kilometers (km).

The same operations that can be performed on an int can be performed on a dist object. Example

\[
dist \text{ myNewDist } = 10m + 20.5km - 30ft + \text{myOldDist};
\]

**dir**

\[
\text{dir <object\_name> } = \text{int} \mid \text{string } \text{d}; \\
\text{dir}(\text{int} \mid \text{string } \text{d});
\]

Defines a new dir object (first form), or instantiates a new one for a function parameter if needed (second form). \text{d} can be written as a old-style point of the compass (i.e.: N, S, NE, NNE, etc.) or the standard way, an int between 0-359.

**coord**

\[
\text{coord <object\_name> IS float} \mid \text{string } \text{longitude}, \text{ float} \mid \text{string } \text{latitude};
\]

Defines a coord object with longitude and latitude given in either the GE standard floating point format, or in DMS (Degrees Minutes' Seconds") format, using spaces or any other expected symbol as separators. The elevation is given as a dist type.

coord objects can be manipulated using an arithmetic based on navigation (see below).

**point**

\[
\text{point <object\_name> IS coord <cord>, string <title>, string <comment>, string <URL>};
\]

Defines a point object, which appears in Google Earth as a pop-up square with title, coordinates and elevation, comments, and a URL that can connect to a photo or a website itself.
struct
struct <object_name> IS string <shape> .... (arguments vary according to
shape given)

Defines a struct object using several in-built shapes. The arguments required
by each
shape are:

square, coord <cord>, dist <length>, dist <height>, string <elevation_flag> string <tasselated_ON|OFF>;
rectangle coord <cord>, dist <length>, dist <width>, dist
<height>, string <elevation_flag>, string <tasselated_ON|OFF>;
triangle coord <cord>, dist <length>, dist <length2>, dist
<length3>, string <elevation_flag>, string <tasselated_ON|OFF>;
circle coord <cord>, dist <diameter>, string <elevation_flag>
can be either A|a or R|r, for absolute or relative (to the
earth) elevation. The tasselated flag gives the option of extruding a polygon
down to the ground if it is not laying on it.

overlay
overlay <object_name> IS coord <cord1> .... coord <cord4> string
<URL> dir <rotation>;

Defines an overlay object, a graphic file that is laid over the map according to
the 4 coordinates
and rotation passed to the function. The URL points to the graphic file.

line
line <object_name> IS coord | coord[] <cord> .... cord <cord_n>, string

Defines a line object whose points are given either by a list of coord objects, or
by an array of such.

place
place <object_name> IS point | struct | line | overlay <object>....
point | struct | overlay
<object_n> ;

Defines a place, or collection of point, struct, line or overlay objects
that can be associated with a persp object, so as to look at the collection from
the point of view defined by it. It can also be added/associated with a folder
object, so as to group it together in the Google Earth navigation bar. A new instance of these objects can be added to an already existing place by using the 'add and assign' ( += ) operator:

```plaintext
place <object_name> += point | struct | line | overlay <new_object>;
```

**persp**

```plaintext
persp <object_name> IS coord <camera_loc>, dist <range>, dir <tilt>, dir <heading>;
```

Defines a persp object, which can be related to a place, line, overlay, struct or point objects, so that clicking on them on the GoogleEarth navigation bar makes the GE browser look at these objects from the perspective defined by the persp object.

**folder**

```plaintext
folder <object_name> IS point | struct | line | overlay <object>... point | struct | line | overlay <object_n>, string <title>;
```

Groups a place, line, overlay, struct or point object under a single GE browser navigation folder, under title specified by title.

8. A special arithmetic for manipulating coord objects

To ease the finding and fixing of a determined geographical point, and to create a coord object from it, a special arithmetic allows the user to 'move' from one coord to a new point and create a define a new coord object out of it, using the following format:

```plaintext
coord <object_name> = coord <point> + (dist <x>, dir <y>);
```

In this way, <object_name> would be a point defined by moving x meters (all distances are converted to meters once processed by the compiler) in the direction of y degrees by the compass, starting from point.