Path Variables

Paths and Options variables



Confidential, unauthorized access forbidden.

CCN Studios internal proprietary software documentation.

□□3.0

Sun 25 May 2025 03:39:23 GMT+1

Synopsis

Path variables are based upon the file paths “/../../..” and “../../..” string formats.

In Free Form Programming Language Path variable is of two kinds:

1. “mesh/value”: **local** to the current program, no starting “/”
2. “/student/dara/script1/mesh/uvcoors” : **external** program thus the path starts with “/”

Specification

1. **Atomic path variable**: “/group/user/script/var”
2. **Composite Path variable**: “/group/user/script/var/value”
3. **Keys**:“/group/user/script/var/value/key”

Key is the identifier or the name of something that consumes computing resources to exist. The latter something is called **Value** as in value for the key. “mesh” is a Key and “mesh/normals” is the

Value, note that “mesh” consumes almost 0 computing but to compute the normals of a surface consumes substantial quantities of computing resources e.g. CPU, memory, network bandwidth and of course **recurring consumption** of resources e.g. GPU to render the mesh as a graphical object 30 frames a second on demand!

- Both local and external path variables are indeed URL based and stored in network cloud objects
- The file path address format is familiar to every grammar school student and therefore was selected as the customary way to address variables in complex and distributed structures
- The said complex structures might expand beyond the current program and into many other programs to share their local variables
- Indeed a newbie to Free Form Programming Language can immediately access and use Path variables constructed by other programs before
- Note that by default all variable values are automatically Serialized and if exceed certain water marks then compressed. Compression is one of the best forms of Serialization
- The Path variables are by default available to all third party programming languages, and always in standard simplest JSON form

http: and https: paths

⚠ **The Free Form Grammarian leaves the http paths unprocessed**. They are copied and passed to operators and functions as constant strings with no further interpretations and no further processing**.

Example

```
var = "/ff/free/image1/img";
img = "var/value";
show img;
save as image2;
```

Output:

```
img=![](https://wiki.compclassnotes.com/uploads/images/gallery/2026-05/embedded-image-aonvdpon.png)
```

```
img ⇶ {"var/value"→☁, "(var)"→☁, "(var)"↪var, "var/compressed"→☁, "var/type"→☁, "var/expression"→☁}
```

☁: icon for the persistent cloud object where the value of the key is stored

⇶ : denotes multi-arrow indicting a composite map

↪ : return arrow back to the variable in the program

Options variable

Alternatively the / can be replaced by ‘s for the existing Path variables:

```
var = "/ff/free/image1/img";
img = var's value;
show img;
save as image2;
```

In Free Form Programming Language ‘s is called option as in var’s value as an option of var. Options ‘s made passing a long list of initialized variables to a function much more compact and easier to remember and manage.

⚠ **Options variables with double quote values require more testing.**

Options variables construct path variables from arbitrary option identifiers not in existence! For example, “dara” variable has never had a path variable “hair”:

```
dara's hair is "kind of wavy";
style = "dara/hair";
show style;
save as test3;
```

Output:

```
Style = "kind of wavy"
```

The internal Free Form Grammarian has a function called VarMaker[ ] to construct standard listed typed variables or the arbitrary on-demand never seen before variables and types.

□□ This example clearly indicates the Free Form Spoken nature of the Free Form Programming Language. As the experiments have indicated the youths learn programming from Free Form language much easier and speedier and with motivation!

□□In order to match the English language grammar, for the plural names ending with s the option variables will have the trailing s’:

```
apples' count is 19;
count = "apples/count";
show count;
save as test3;
```

Output:

```
count = 19
```

If a path variable or option variable is not available, as in completely missing, then the original identifier, in path form, is returned as value. This avoids dealing with obscure error messages and other unpleasantries of missing values. Therefore GetSortedVars[ ] always returns a valid value no matter, and the missing value is checked as follows:

```
GetSortedVars \[ name, cred \] == name
```

If GetSortedVars[ ] returns the same value then that variable is missing or missing its value.

Example

The Free Form rocks are constructed by the function rock[ ] which returns a string which is the name of the lhs variable:

```
mesh = rock\["ellipse", 200, 5\*{1, 1, 6}\];
save as rock1;
```

```
rock\[ \] ⇶ {"(mesh)"→☁, "(mesh)"↪mesh }
```

☁ : icon for the persistent cloud object where the value of the key is stored

⇶ : denotes multi-arrow indicting a composite map

↪ : return arrow back to the variable in the program

A third party Free Form program by another Free Former can easily access this rock mesh:

```
var = "/ff/free/rock1/mesh/value";
uv = "var/uvcoors";
show uv;
save as pathvar;
```

Output:

```
uv = {{0.789561, -2.69016}, {-1.67072, 3.75263}, {1.37182, 4.41598}, {0.199764, 4.03556}, {-1.63712, -0.0143999}, {0.230512, -3.25848}, {1.07342, 3.46127}, {1.72891, 1.27304}, {3.60557, 0.308929}, {2.13059, -3.67745}, {2.37085, 2.78642}, {-0.132767, -4.5409}, {-2.19815, -1.07722}, {-0.272674, 4.59587}, {2.96563, -2.22853}, {-0.422547, -4.48889}, {-0.926991, 2.31835}, {0.376589, -0.486291}, {-2.86105, 1.67475}, {1.01256, -3.51811}, {2.76276, 2.33716}, {-4.19068, -0.835185}, {-3.64371, -3.00476}, {0.569355, 2.71332}, {3.30174, 3.03551}, {-2.22201, -0.964461}, {4.5202, -1.39236}, {2.20664, -3.40651}, {4.37338, 1.05228}, {1.03858, -2.69736}, {4.64808, 0.375639}, {3.26256, 3.2405}, {-2.69063, 1.88625}, {-4.5811, 0.34247}, {-3.96512, -0.983707}, {-4.62285, 1.23987}, {-1.38243, -2.3752}, {-0.513229, -4.06979}, {0.602132, 2.14761}, {-2.39837, -3.98407}, {3.70975, -2.5965}, {-2.47709, 3.37107}, {-1.75433, 4.43549}, {1.52735, 1.43025}, {3.46578, -1.03043}, {-0.618612, -0.794119}, {4.68354, -1.12572}, {2.68421, 3.84622}, {-1.4737, 0.439442}, {-4.8006, 0.81873}, {-3.83208, -2.11617}, {3.53052, 1.5748}, {-1.82833, -3.77776}, {3.47874, 0.256054}, {2.02464, 0.0828023}, {-2.66834, -2.55233}}
```

var ⇶ {"var/coors"→☁, "var/uvcoors"→☁, "var/vertices"→☁, "var/lines"→☁, "var/triangles"→☁, "var/normals"→☁, "(var)"→☁,"(var)"↪var, "var/compressed"→☁, "var/type"→☁, "var/expression"→☁}

⚠ The Free Form Grammarian often compresses the cloud variables under certain criteria of size and contents.

Revision #1
Created 2026-05-16 01:30:45 UTC by Dara
Updated 2026-05-16 01:32:50 UTC by Dara