Reference Errors

The reference error is a rarely encountered issue in k. It is also once the most confusing. This article will explain why they happen and how to avoid them.

Notes:
- All code examples must be executed from a fresh k session to get reliable results.
- K Variable Types is required reading for this article.

What is a Reference?

Functions can have references. Global variables appearing in a function will create a reference. A function can reference several global variables. In the following example the anonymous function references the global `.foo`:

```
{.foo}
```

After stating the above function in a k session, you will notice that the .foo variable in the k tree has been initialized to _n. This is a side effect of the reference and you can see that the act of referencing has a direct effect on the state of the k session.

Relative Reference

You can also create a reference to a global variable using its relative name. In the following code `.k.bar` is referenced in an anonymous function:

```
{bar}
```

Again, `bar` will be initialized to _n.

Deep References

Relative and Absolute references and can also have further depth in the k tree using dot notation.

```
rel:{l.m.n}
abs:{.a.b.c}
```

You will see that the two function have initialized dictionaries to the complete depth of the dot assignments with the leaf value as _n.

Reference Error?

Take all functions currently defined in a k tree and list all their references using absolute paths.

```
rel:{l.m.n}  / ref .k.l.m.n
abs:{.a.b.c} / ref .a.b.c
```

If you try to assign any part of these branches of the k tree such that the leaf cannot exist, you will get a reference error.

```
rel:{l.m.n}  / ref .k.l.m.n

.k.l:1     / l is no longer a dictionary to l.m cannot exist or l.m.n
.k.l.m:1   / m is not a dictionary so m.n cannot exit
.k.l.m.n:1 / fine, leaf intact
.k.l.m:.() / m.n no longer exists
```

Examples

An out of order Parse Error

```
f:{a.b}
a:1 / reference error
```

Assigning a to `1` will break `f`'s reference to `a.b`.

```
a:1
f:{a.b} / parse error
```

Assigning `{a.b}` cannot be parsed because a is not a dictionary.

Brackets

```
f:{a[`b]}
/ has a reference to a
a:1 / this assigment leaves the leaf, a, intact
.[f;,_n;:] / calling this function will break, on the 1[`b]
/ (1;"rank")
.k:.k _di `a / this will cause a reference error since we have removed
the leaf
```

One liner

```
a:{a.b}
```

Do Loops!?

It only breaks in raw execution, such that is a global variable. This one is a bit unexplained, as you can see that the expansion of the do loop would not cause a reference error under any conceivable circumstances.

```
do[1;a:.();a.n:1]
```