API for Tree version 0.4.0

These methods are implemented by `Tree` version `0.4.0`

cd

change value of `currentPath`

syntax:

```
cd [ path ]
```

currentNode

send message to the node specified by `currentPath`

syntax:

```
currentNode [ message ]
```

forceRemoveNode

remove a node and all its children

syntax:

```
forceRemoveNode path
```

isMethod

determine whether this dictionary has the named method.

returns the full path to the executable that implements the method for this Dictionary

syntax: isMethod *methodName*

returns the name of the executable.

resolve in this order:

local method: *`methodName`*

common method: `Dictionary.`*`methodName`*

isNode

determine if a path represents a valid node

syntax:

```
isNode path
```

path is either relative or absolute

if relative, is relative to currentPath

returns:

the absolute path if it exists, null string otherwise.

ls

list all nodes that are children of the specified one

syntax:

```
ls path
```

makeNode

create a node in a Tree

syntax:

```
makeNode path [ cloneOf ]
```

path may be absolute or relative to currentPath. *cloneOf* specifies an optional dictionary to clone.

makeRootNode

create root node of a Tree

syntax:

```
makeRootNode [ cloneOf ]
```

cloneOf specifies an optional dictionary to clone.

node

reference a node in a Tree

syntax:

```
node path [ message ]
```

path is either relative or absolute

if relative, is relative to currentPath

parseMessage

For a dictionary, parses a message and converts it to a canonical form of *method arguments*

specifically (apart from the normal key syntactic sugar),

path message converts to node *path message*

syntax:

```
parseMessage message
```

returns a message in the form *method arguments*

NOTE This uses a deprecated methodology which is no longer consistent with other Offsiders. Should just override `sugar` instead.

removeNode

remove a node only if there are no children

syntax: $ removeNode *path*

rootNode

reference the root node of a Tree

syntax:

```
rootNode message
```

upgradeMe

version

walk

walk through the tree structure, sending a particular message to each node in turn

syntax:

```
walk [ depthFirst ] path message
```

start at node *path*

if `depthFirst` is specified, descend down into each node's children before sending the message to that node.

The default is to send the message to a node before descending down into that node's children.