1 GedcomNode

1.1 Introduction

A `GedcomNode` encodes a line from a Gedcom file. It is a key class in DeadEnds. Records in a DeadEnds database are GedcomNode trees.

DeadEnds does not have records for persons, families, etc. All DeadEnds records are references to roots of node trees. DeadEnds records are stored in a Database.

Other genealogical systems define specific types for genealogical entities. Of course DeadEnds has routines that deal specifically with persons and families, etc., but all their interfaces use references to `GedcomNodes`.

1.2 Class GedcomNode

```
public class GedcomNode: CustomStringConvertible {

    var key: String? // Key; only on root nodes.
    var tag: String  // Gedcom tag; mandatory.
    var value: String? // Value; optional.
    var nextSibling: GedcomNode? // Next sibling; optional.
    var firstChild: GedcomNode?  // First child; optional.
    weak var parent: GedcomNode? // Parent; not on root nodes.

    public var description: String
    init(key: String? = nil, tag: String, value: String? = nil)
    func printTree(level: Int = 0, indent: String = "")
    lazy var childrenByTag: [String: [GedcomNode]]
    static func == (lhs: GedcomNode, rhs: GedcomNode) -> Bool
```

`key`, `tag` and `value` hold the key (*cross reference identifier*), tag, and value of a Gedcom line. The line's level in not stored because it can be computed. `firstChild`, `nextSibling` and `parent` hold the tree structure.

`description` returns a description of a node.

`init` initializes and returns a new node.

`printTree` is a debug method that prints the tree rooted at a node.

`childrenByTag` gets the dictionary of arrays of child nodes indexed by a tag. It is a lazy computed property.

1.3 Convenience Accessors

```
extension GedcomNode
    func valueForTag(tag: String) -> String?
```

```
    func valuesForTag(tag: String) -> [String] {
    func childWithTag(tag: String) -> GedcomNode?
    func childrenWithTag(tag: String) -> [GedcomNode]
```

These accessor methods give access to the children of a `GedcomNode` with certain properties:

`valueForTag` returns the value of the first child with a given tag.

`valuesForTag` returns the array of non-nil values from a list of children with a given tag.

`childWithTag` finds the first child node a given tag.

`childrenWithTag` finds the array of children with a given tag.

1.4 TagMap

Gedcom files can be large; the same tags may occur thousands of times. The TagMap class provides a way for every GedcomNode with the same tag to share the same string.

```
class TagMap
    private var map: [String:String] = [:]
    func intern(tag: String) -> String
```

`intern` returns the unique copy of a string.

1.5 Notes and ToDo's

How about a computed property for level?

Contrast the DeadEnds way of encoding all information in trees of `GedcomNode`s, rather than as specialized records with specialized types for things like names, events, dates, places, relationships.

2 Import Stack – `ImportStack.swift`

2.1 Introduction

The *import stack* consists of the functions that read Gedcom files and create the `Array`s of `GedcomNode` trees, one for each Gedom record found in the file. Each record consists of a tree of `GedcomNode`s where the root corresponds a level 0 (e.g., `0 INDI`, `0 FAM`) line. After the records are created validation takes place and a Database may be created. Validation is covered in Section 3 and the Database in Section **??**.

The functions making up the import stack are each described in a section below.

2.2 `extractFields`

```
enum ReadResult {
    case success(level: Int, key: String?, tag: String, value: String?)
    case failure(errmsg: String)

func extractFields(from line: String) -> ReadResult
```

`extractFields` extracts the level, key, tag and value from a String that holds a single Gedcom line. The field values are returned via a `ReadResult` value, an enumeration with one value for successful returns and another for errors.

`extractFields` is called only by `getDataNodesFromPath`.

2.2.1 Note on Errors

As far as is possible errors that occur when running the input stack do not stop processing. Errors accumulate so the user can be apprised of as many errors as possible.

2.3 `getDataNodesFromPath`

```
struct DataNodes<Type> { var nodes: [(GedcomNode, Type)] ... }

func getDataNodesFromPath(path: String, tagmap: inout TagMap, keymap: inout KeyMap,
    errlog: inout ErrorLog) -> DataNodes<Int>?
```

`getDataNodesFromPath` returns all lines from a Gedcom source as elements in a flat `[DataNodes<Int>]` array. Elements of the array are `(GedcomNode, Int)` tuples, where the `GedcomNode` holds the fields returned by `extractFields`, and the integer holds the the node's level – a `GedcomNode` does not have a level field, but the next step needs the levels in order to build the trees.

Input parameters:

`path` – path to a Gedcom file.

`tagmap` – `inout TagMap` of unique tag strings. See Section ????.

keyMap – inout [String:Int] dictionary that maps a record's key to its starting line number in the Gedcom file. It is used when generating error messages.

errlog – inout ErrorLog where errors found when processing the Gedcom file are recorded. The source is fully processed regardless of errors. If there are errors nil is returned.

Summary: getDataNodesFromPath reads a Gedcom file and breaks it into an array of lines. It calls extractFields on each line to get its GedcomNode. It then adds a (GedcomNode, level) tuple to the return array.

getDataNodesFromPath is called by getRecordsFromPath as the first step in reading the Gedcom records from a file.

2.4 getRecordsFromDataNodes

```
typealias RootList = [GedcomNode]

func getRecordsFromDataNodes(datanodes: DataNodes<Int>, keymap: KeyMap,
    errlog: inout ErrorLog) -> RootList {
```

getRecordsFromDataNodes converts an array of GedcomNodes, in the form of (GedcomNode, level) pairs in a DataNodes<Int> array, from a Gedcom source, into a RootList, the array of root GedcomNodes of all records from the Gedcom file.

Because the input is a sequential list of all the GedcomNodes from the file, gedRecordsFromDataNodes also needs the levels of the GedcomNodes to be able to construct the trees. This is why its input is a DataNodes<Int> object rather than a simple [GedcomNode] array. It needs the levels to guide the tree building. It converts a flat array of all GedcomNodes, into an array of root nodes with their trees attached. It builds the trees using a simple state machine.

getRecordsFromDataNodes is called by getRecordsFromPath, immediately after it calls getDataNodesFromPath. This completes the two step process where the first gets the full list of GedcomNodes from the file, and the second builds the GedcomNode trees and returns the array of the root GedcomNodes. RootList is an alias for [GedcomNode] to be used when the array of nodes contains only roots.

2.5 getRecordsFromPath

```
func getRecordsFromPath(path: String, tagmap: inout TagMap, keymap: inout KeyMap,
    errorlog: inout ErrorLog) -> RootList?
```

getRecordsFromPath returns the Gedcom records from a source. It uses getDataNodesFromPath and getRecordsFromDataNodes to create a RootList of records.

See the previous two function for details on the process and the meanings of the parameters.

getRecordsFromPath and the functions it calls make up the input stack. The function that follows is located in InputStack.swift, but could have been included in the validation software.

2.6 getValidRecordsFromPath

```
func getValidRecordsFromPath(path: String, tagmap: inout TagMap,
    keymap: inout KeyMap, errlog: inout ErrorLog)
    -> (index: RecordIndex, persons: RootList, families: RootList)?
```

`getValidRecordsFromPath` uses the input stack by calling `getRecordsFromPath` to get the Gedcom records from a file, and it then validates those records by calling:

```
checkKeysAndReferences
validatePersons
validateFamilies
```

`getValidRecordsFromPath` returns a triple consisting of:

`index` – `[String:GedcomNode]` dictionary mapping record keys to their root `GedcomNode`s.

persons – RootList, an Array listing all GedcomNode roots of persons from the file.

families: RootList list all GedcomNode roots of famlies from the file.

See the Validation Section for the documentation on validation and these validation functions.

2.7 Looseends

ValidationContext is defined in ImportStack.swift, though not used there.

3 Validation