

# 1 Import Stack – `ImportStack.swift`

## 1.1 Introduction

The *import stack* consists of the functions that read Gedcom files and create the Arrays of `GedcomNode` trees, one for each Gedcom record found in the file. Each record consists of a tree of `GedcomNodes` 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 2 and the Database in Section ??.

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

## 1.2 `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`.

### 1.2.1 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.

## 1.3 `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.

## 1.4 `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, `getRecordsFromDataNodes` 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.

## 1.5 `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.

## 1.6 `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 `GedcomNodes`.

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

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

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

## 1.7 `Looseends`

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

## 2 Validation