GEDCOM Syntax

Characters	`digit = %x30-39 ; 0 through 9`
	`nonzero = %x31-39 ; 1 through 9`
	`ucletter = %x41-5A ; A through Z`
	`underscore = %x5F ; _`
	`atsign = %x40 ; @`

Structure	• Besteht aus einem *structure type*, einem optionalen *payload* und einer Sammlung von *substructures* → structure muss entweder einen nicht-leeren payload oder mindestens eine substructure haben • Payload: String • Eine *structure* ist eintweder ein *record* oder eine *substructure* von genau einer anderen *structure* • Record: ist keine substructure einer anderen structure (also level 0) • *Superstructure*: die parent-structure einer substructure
Pseudo-Structure	• Müssen keinen nicht-leeren payload oder eine substructure besitzen • Header (HEAD): erster Eintrag eines Dokuments → hat level 0 und keinen line value → enthält Metadaten über das Dokument → jeder Header muss GEDC substructure haben, der die Spezifikation festlegt • Trailer (TRLR): letzer Eintrag eines Dokuments → hat level 0 und keinen line value → darf keine substructures enthalten • Line-Continuation: wird verwendet um multi-line payloads zu kodieren → darf keine substructures enthalten
Line	• String-Repräsentation eines Teils einer structure • Besteht aus level, optionalem cross-reference identifier, tag, optionalem line value und dem line terminator • D: delimiter (in dem Fall Leerzeichen) `Line = Level D [Xref D] Tag [D LineVal] EOL`
Level	• Kodiert Zusammenhänge zwischen substructes • Level 0 repräsentiert einen record oder eine record-ähnliche pseudo-structure (z.B. header oder trailer) • Level $x > 0$ repräsentiert eine substructure der nähsten vorhergehenden line `Level = "0" / nonzero *digit`

Cross-reference identifier	• Indiziert, dass es sich um eine structure handelt, auf die ein pointer-type payload zeigt (bzw. zeigen kann)
	→ jeder record auf den andere Structures verweisen, muss einen Xref haben
	→ substructures dürfen keinen Xref haben
	• Müssen eindeutig im Dokument sein
	• Sollten nicht sichtbar für den (End-) Benutzer gemacht werden
	
	`Xref = atsign 1*tagchar atsign ; but not "@VOID@"`
	`tagchar = ucletter / digit / underscore`

Tag	• Kodiert den type der structure • stdTag sind in der Spezifikation festgelegt • extTag werden in Extensions spezifiziert `Tag = stdTag / extTag`

Line value	• Kodiert den payload einer structure
	→ kann entweder ein line string oder ein pointer sein
	• Pointer: ist der Xref der structure auf die gezeigt wird
	→ Xref value muss in gleichem Dokument definiert sein
	→ nullpointer können mit voidPtr dargestellt werden (z.B. wenn zugehörige Xref-Definition nicht mehr vorhanden ist)
	• Line string: non-pointer payload
	→ exakte Kodierung wird durch datatype des payloads festgelegt
	→ leerer payload (empty string) und fehlender payload sind äquivalent
	
	`LineVal = pointer / lineStr`
	
	`pointer = voidPtr / Xref`
	`voidPtr = %s"@VOID@"`
	
	`lineStr = (nonAt / atsign atsign) *nonEOL ; leading @ doubled`
	→ @ als erstes zeichen eines Strings muss escaped werden

Line Terminator (EOL)	• Zeilenumbruch • Line values können keinen internen line terminator haben, aber manche payloads können → payload wird dann in mehrere payloads aufgeteilt → erster payload wird als line value der line kodiert → alle weiteren Teile des payloads werden als line value einer line continuation pseudo-structure dargestellt

Line continuation	• Pseudo-structure
	• Folgt unmittelbar auf die line die fortgeführt werden soll und hat level+1
	• Tag: CONT
	
	`1 NOTE This is a note field that`
	`2 CONT  spans four lines.`
	`2 CONT`
	`2 CONT (the third line was blank)`

Extensions	• Spezifikation kann durch Extensions erweitert werden
	→ wird mit extTag dargestellt
	→ extension tags sollten mit Underscore beginnen
	• Alle extension tags sollten in der schema structure definiert werden (zu einer URI gemapped werden)
	→ schema structure ist substructure des Headers mit tag SCHMA
	
	`0 HEAD`
	`1 SCHMA`
	`2 TAG _SKYPEID http://xmlns.com/foaf/0.1/skypeID`
	`2 TAG _MEMBER http://xmlns.com/foaf/0.1/member`
	
	• Bedeutung von extension tags und alle substructures der extension werden durch die URI festgelegt, nicht durch den tag selbst
	• Standard structures sind über extensions zu preferieren

Removing data	• Pointer zu gelöschten structures sollten durch voidPtr ersetzt werden • Wenn die superstructure durch das Entfernen einer substructure invalid wird, sollte die structre beibehalten und der payload durch einen voidPtr (oder einen dem Datentyp entsprechenden empty value, wenn non-pointer) ersetzt werden • Wenn superstructure durch Entfernen einer substrcuture leer ist (keine substructures und kein payload) sollte superstructure entfernt werden