UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Simple ID3

Grammatica per il riconoscimento dei TAG degli MP3

Corso di Linguaggi Formali e Compilatori

Greco Salvatore 1053509

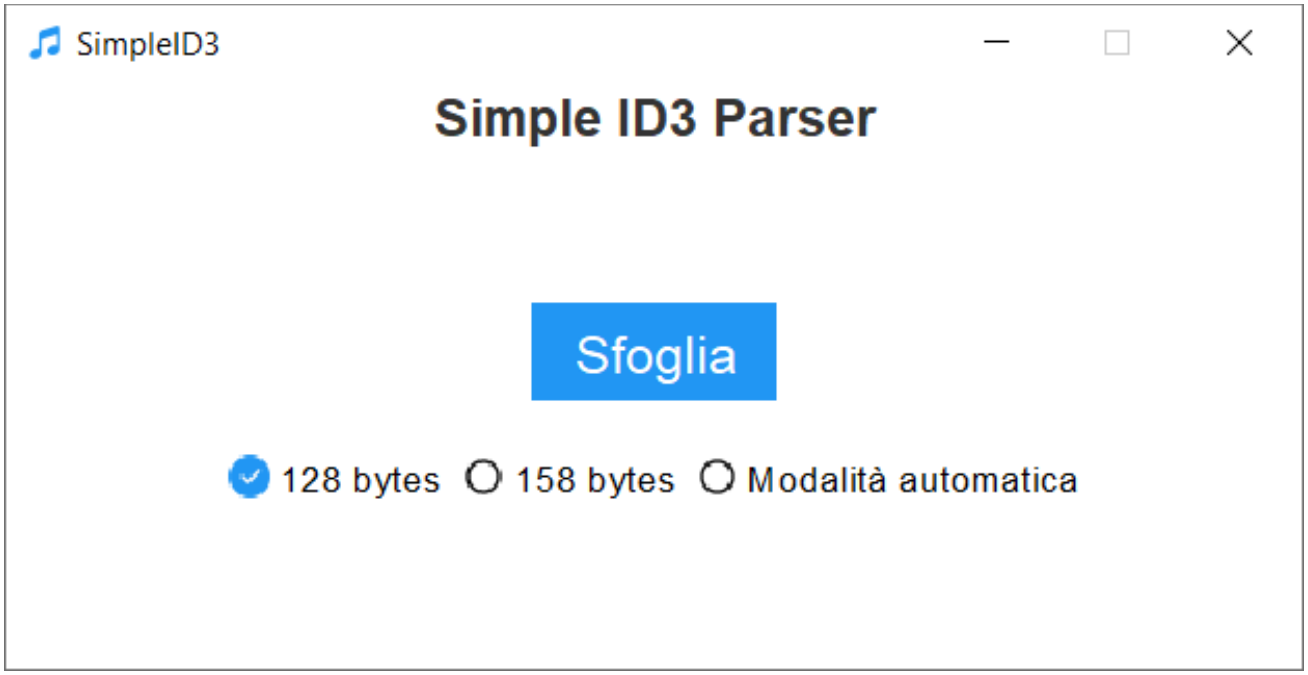
Gamba Fabio 1053157

Indice

Interfaccia e utilizzo

- Interfaccia

SimpleID3 è un JAR eseguibile dopo aver installato JDK o JRE sul proprio computer windows. Il tool permette di estrapolare le informazioni ID3v1 con varie modalità.



- Selezione modalità

Il programma può funzionare in tre diverse modalità:

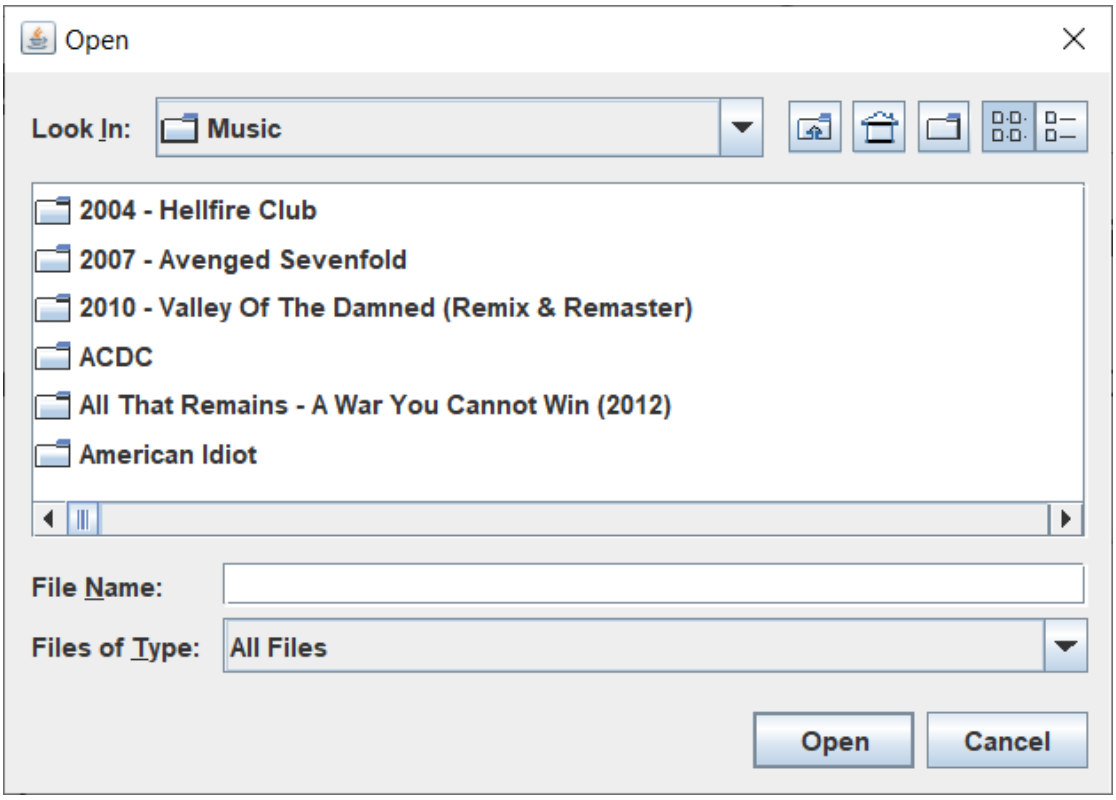
- Modalità 128byte -> il tool estrae gli ultimi 128 byte del file mp3 indifferentemente dal loro contenuto. È la modalità più rapida, visto che le informazioni ID3v1 sono quasi sempre alla fine del file mp3, ma può avere problemi in caso di tag non conformi allo standard (mancanza/eccesso di byte presenti) poiché effettua il riconoscimento semantico in un secondo momento.

- Modalità 158byte -> vengono estratti gli ultimi 158 byte, dovuti all'aggiunta di campi per il riconoscimento delle informazioni.

- Modalità automatica -> viene analizzato tutto il file mp3 alla ricerca dell'inizio delle informazioni ID3v1 segnalate dalla parola «TAG». Può cercare dati in tutto il file e questa modalità non è affetta da problematica riguardanti bit in eccesso o difetto, ma non è altrettanto veloce e può, talvolta, dare qualche errore, poiché parsa tutto l'mp3 come file di testo.

- Selezione del file mp3

Selezionata la modalità, il tool ci permette di scegliere un file mp3 in qualsivoglia posizione all'interno della memoria di massa della nostra macchina. Per i classici mp3, la lettura viene effettuata con modalità 128 byte o automatica.



- Risultato parsing

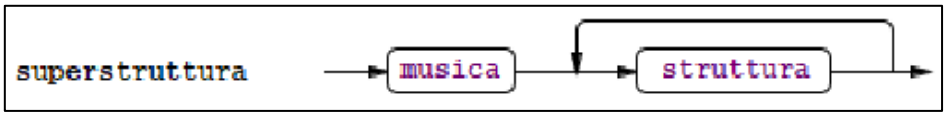
Infine, a meno di informazioni non conformi allo standard, abbiamo le informazioni visualizzate a schermo.

Grammatica

- Struttura generale

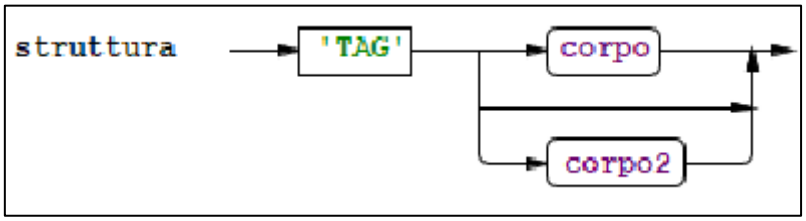
La grammatica utilizzata permette di riconoscere 2 differenti strutture di dati, basate sullo stesso concetto di informazioni ID3v1.

Possiamo anche analizzare un intero file mp3 fino al riconoscimento della struttura principale dell'ID3v1. Come slot «musica» indichiamo un qualsivoglia numero di caratteri.



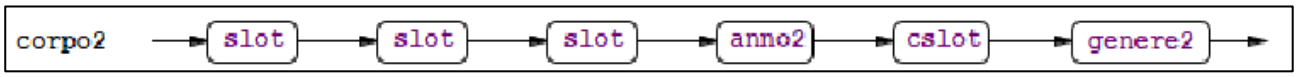
La struttura inizia sempre con la parola chiave «TAG» qualsiasi sia il suo tipo.

```
superstruttura: musica struttura+;
musica  :       .*;
struttura: 'TAG' (corpo||corpo2);
```



- Grammatica senza header (ID3v1)

La grammatica che riconosce l'ID3v1 standard è caratterizzata in “corpo2” come segue:



Ogni slot è una sequenza di 30 caratteri (char)

```
slot returns [List<Token> p]
@init { p = new ArrayList<Token>();}
        :
        c1=CHAR c2=CHAR c3=CHAR c4=CHAR c5=CHAR c6=CHAR c7=CHAR c8=CHAR c9=CHAR c10=CHAR
        c11=CHAR c12=CHAR c13=CHAR c14=CHAR c15=CHAR c16=CHAR c17=CHAR c18=CHAR c19=CHAR c20=CHAR
        c21=CHAR c22=CHAR c23=CHAR c24=CHAR c25=CHAR c26=CHAR c27=CHAR c28=CHAR c29=CHAR c30=CHAR
        {p.add(c1);p.add(c2);p.add(c3);p.add(c4);p.add(c5);p.add(c6);p.add(c7);p.add(c8);p.add(c9);p.add(c10);
        p.add(c11);p.add(c12);p.add(c13);p.add(c14);p.add(c15);p.add(c16);p.add(c17);p.add(c18);p.add(c19);p.add(c20);
        p.add(c21);p.add(c22);p.add(c23);p.add(c24);p.add(c25);p.add(c26);p.add(c27);p.add(c28);p.add(c29);p.add(c30);}
        ;
```

Invece “anno” è formato da una sequenza di 4 caratteri e “cslot” è una variante dello slot mostrato qui sopra, ed è creato per leggere senza errori anche le informazioni ID3v1.1, senza però supportare le informazioni della traccia.

- Grammatica senza header (ID3v1.1)

La grammatica ID3v1.1 aggiunge al posto degli ultimi due bit dello slot commento, un byte di informazione per il numero di traccia. Questo byte non viene letto come char per far sì che le versioni v1.1 possano essere comunque lette, anche se non totalmente. Al posto degli ultimi due char, quindi, abbiamo un qualsiasi carattere.

```
c27=CHAR c28=CHAR c29=. c30=.
```

Il riconoscimento del genere è affidato alla parte Java che elabora l'informazione contenuta nel byte e cerca una eventuale corrispondenza. Il problema viene affrontato successivamente nel dettaglio.

```
genere2 returns [Token t]:p=.  {t=p;};
```

- Grammatica con header

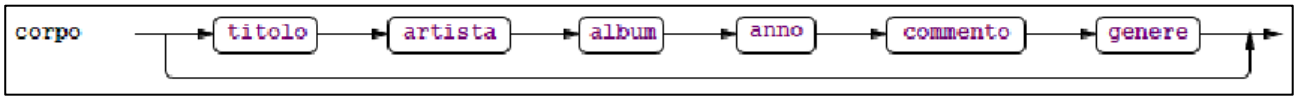
La grammatica con header è pensata per poter individuare eventuali errori all'interno dei dati del file ID3. Si basa sullo stesso standard ma vengono aggiunge delle «parole» all'inizio di ogni slot dati e per questo motivo i byte totali da leggere salgono a 158.

Gli header sono necessari per poter individuare byte/caratteri mancanti senza che essi vengano “scalati” prendendo il successivo.
Infatti, per come è costruito lo standard non c'è differenza tra lo slot “titolo” e lo slot immediatamente successivo “artista”, poiché entrambi sono una sequenza non nulla di 30 char.

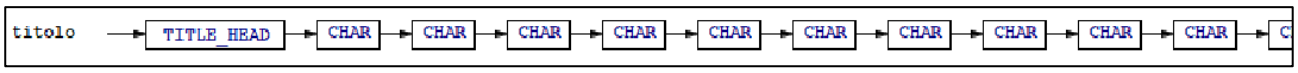
L'aggiunta di queste parole permette sia di riconoscere byte mancanti sia di individuare errori ben più evidenti come la totale mancanza di uno slot.

La struttura in Antlr inizia con il token HEAD che è sempre uguale a “TAG”

Il corpo che contiene le informazioni è così definito

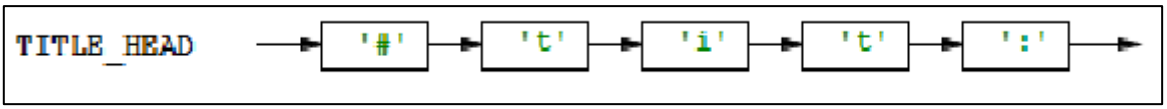


Ogni elemento del corpo è definito in modo simile, tranne che per il numero di byte che caratterizza un certo campo e l'header che li contrassegna.
Per esempio, il campo “titolo" è definito in questo modo:



I “CHAR" sono ripetuti 30 volte.

Il token “TITLE_HEAD" ha questa struttura:



Tutti gli header sono anticipati dal carattere #. Questo preclude il suo utilizzo all'interno degli slot dati (infatti è stato escluso dalla definizione di CHAR) per evitare problemi di riconoscimento dei componenti del TAG.
Gli header utilizzati per gli altri campi sono:

Titolo -> #tit
Artista -> #art
Album -> #alb
Anno -> #ann
Commento -> #com
Genere -> #gen

- Gestione dei dati

Una problematica, derivante da questo tipo di grammatica, è la lettura dei CHAR da parte dell'handler del programma in Java.
Avendo una lunghezza non variabile di 30 caratteri (vuoti, ma presenti anche se non utilizzati) bisogna forzare la quantità di CHAR a 30, e rispettivamente recuperare 30 volte il Token nella posizione corretta.

```
TITLE_HEAD: '#tit:';
titolo returns [List<Token> p]
@init { p = new ArrayList<Token>();}
  :  TITLE_HEAD
  c1=CHAR c2=CHAR c3=CHAR c4=CHAR c5=CHAR c6=CHAR c7=CHAR c8=CHAR c9=CHAR c10=CHAR
  c11=CHAR c12=CHAR c13=CHAR c14=CHAR c15=CHAR c16=CHAR c17=CHAR c18=CHAR c19=CHAR c20=CHAR
  c21=CHAR c22=CHAR c23=CHAR c24=CHAR c25=CHAR c26=CHAR c27=CHAR c28=CHAR c29=CHAR c30=CHAR
  {p.add(c1);p.add(c2);p.add(c3);p.add(c4);p.add(c5);p.add(c6);p.add(c7);p.add(c8);p.add(c9);p.add(c10);
  p.add(c11);p.add(c12);p.add(c13);p.add(c14);p.add(c15);p.add(c16);p.add(c17);p.add(c18);p.add(c19);p.add(c20);
  p.add(c21);p.add(c22);p.add(c23);p.add(c24);p.add(c25);p.add(c26);p.add(c27);p.add(c28);p.add(c29);p.add(c30);}
  ;
```

- Gestione e riconoscimento del Genere

L'informazione contenuta nel byte non è conforme alla codifica ASCII e pertanto non avendo simboli a disposizione nella grammatica, dobbiamo selezionare il byte e occuparci del riconoscimento in Java tramite l'handler.

```
GENERE_HEAD:'#gen:';
genere returns [Token t]: GENERE_HEAD p=.
 {t=p;};
```

Delle 256 combinazioni possibili sono utilizzate solo le prime 145.
Tramite la funzione in Java ci occupiamo del riconoscimento del genere contenuto nel frame.

```
public String riconosciGenere (int i)
{
    switch (i) {
    case 32: return "Blues";
    case 1: return "Classic Rock";
    case 2: return "Country";
    case 3: return "Dance";
    case 4: return "Disco";
        default: return "not recognised";
    }
}
```

Java

- Interfaccia e metodi

L'interfaccia è stata implementata come componente separato dal launcher, utilizzando componenti sia di Java Swing che AWT. Di seguito una breve documentazione dei metodi dell'interfaccia.

- Costruttore che si occupa di impostare e mostrare tutti i componenti dell'interfaccia. Può lanciare un'eccezione I/O per via della lettura di design personalizzati dell'interfaccia (png).

```
public Interfaccia() throws IOException{
```

- Questo metodo si occupa di recuperare le informazioni ottenute dall'handler, impostare e mostrare l'interfaccia per la visualizzazione delle informazioni. In base al parametro intero passato viene selezionata la cella dell'interfaccia, mentre la stringa è l'informazione che vogliamo che mostri

```
public static void setData(int c, String data) {
```

- Metodo che mostra il selettore del file da leggere

```
public String fileChooser() throws NullPointerException
```

- Metodo lanciato dall'ActionListener sul pulsante “Sfoglia”, che a sua volta si ricollega alla classe ParserLauncher.java per lanciare il selettore del file e le operazioni di visualizzazione delle informazioni.

```
private void startParsing(ActionEvent e) {
```

- Metodo per la rilevazione dall'interfaccia della modalità selezionata.

```
public boolean checkMode()
```

Proseguiamo con i metodi del launcher:

- Questo metodo si occupa di ricercare all'interno del file selezionato, l'inizio delle informazioni ID3 tramite la ricerca di TAG. Successivamente in base all'offset scelto vengono prelevati un numero fisso di Byte e restituiti sotto forma di array di char. La presenza dell'oggetto di tipo FileInputStream può lanciare eccezioni di file non trovato e di I/O.

```
public static char[] getTagFromFile(String name, int offset) throws FileNotFoundException, IOException
```

- Metodo che imposta l'oggetto da dare al parser, con un'operazione intermedia di scrittura su file. Recupera la stringa delle informazioni con il metodo prima descritto, selezionando la directory del file interessato tramite il fileChooser dell'interfaccia e passandogli l'offeset selezionato. In questo metodo il file viene “ritagliato" prendendo un numero di bit a partire dalla posizione di “TAG" pari all'offset.

```
public static InputStreamReader setReaderWithStartSearch(Interfaccia a) throws FileNotFoundException, IOException
```

- Funzione che imposta l'oggetto da dare dal parser, restituendo tutto il file mp3 selezionato senza modifiche. Viene utilizzato per il riconoscimento con la modalità automatica.

```
public static InputStreamReader setReaderNoSearch(Interfaccia a) throws FileNotFoundException, IOException
```

- Metodo che restituisce la stringa di informazioni ID3, in modo simile al setReaderWithStartSearch ma senza scrivere su un file intermedio. Viene utilizzato per le due modalità 128/158 byte.

```
public static String setString(Interfaccia a) throws FileNotFoundException, IOException
```

- Metodo richiamato dall'interfaccia che avviail processo di selezione del file e riconoscimento. In base alla modalità selezionata costruisce il parser in maniera diversa (argomento String o BufferedReader) ed effettua quindi il parsing. Infine, in console restituisce la serie di errori trovati, nel caso in cui si siano verificati. Quest'ultimi non sono visibili dall'interfaccia del tool.

```
public static void avvio(Interfaccia a) throws FileNotFoundException, IOException, RecognitionException {
```

- Metodo main che istanzia l'interfaccia.

```
public static void main (String[] args) throws FileNotFoundException, IOException, RecognitionException {
```

Nota sulla lettura dei file:

- I file vengono letti con codifica ANSI ("Cp1252") per permettere la lettura dei simboli del genere, che, altrimenti, creerebbe problemi di ridondanza dei simboli.

Gestione degli errori

- Descrizione

Vengono effettuati controlli sulla presenza e correttezza degli Header e si controlla che ogni campo, contenente le informazioni del brano, sia di lunghezza pari a 30 caratteri.
Sfruttiamo il riconoscimento delle eccezioni di ANTLR per capire che tipo di errore riceviamo durante il parsing

- Test del tool

Il tool è stato testato utilizzando il pacchetto fornito disponibile sul sito dell'organizzazione ID3 -> https://id3.org/Developer%20Information

Dei 270 test + extra, il tool riesce a riconoscere correttamente le informazioni in tutti i 270 casi (extra esclusi).

Sono stati creati alcuni corrispettivi con gli header per testare anche la grammatica alternativa, con buoni risultati durante la prova.

- Casi di test

  - Mancanza o errata scrittura del TAG iniziale

```
Errori rilevati
1 - Syntax Error 1 at [1, 4]: Found TITLE_HEAD ('tit:') - mismatched input 'tit:' expecting 'TAG'
```

- Errata scrittura di «TAG» (es. TaG TA TAGa)

```
Errori rilevati
1 - Syntax Error 1 at [1, 1]: Found CHAR ('T') - mismatched input 'T' expecting 'TAG'
```

```
line 1:2 mismatched character '#' expecting 'G'
Errori rilevati
1 - Syntax Error 1 at [1, 4]: Found TITOLO_HEAD ('tit:') - mismatched input 'tit:' expecting 'TAG'
```

```
Errori rilevati
1 - Syntax Error 1 at [1, 4]: Found CHAR ('a') - no viable alternative at input 'a'
```

- Lunghezza di titolo, artista e album < 30 caratteri

```
Parsing con ANTLR
Handler Inizializzato
Titolo: aaaaaaaaaaaaaaaaaadaaaaaaaaaA<missing CHAR>
Artista: bbbbbbbbbbbbbbbbbdbbbbbbbbbB<missing CHAR>
Album: ccccccccccccccccccccccccccC<missing CHAR>
Anno: 2003
Commento: ddddddddddddddddddddddddddddD
Genere: Blues
Errori rilevati
1 - Syntax Error 1 at [1, 38]: Found CANC - missing CHAR at '#'
2 - Syntax Error 1 at [1, 72]: Found CANC - missing CHAR at '#'
3 - Syntax Error 1 at [1, 106]: Found CANC - missing CHAR at '#'
```

- Lunghezza di titolo, artista e album > 30 caratteri

```
Parsing con ANTLR
Handler Inizializzato
Titolo: aaaaaaaaaaaaaaaaaadaaacmaaaaaa
Artista: bbbbbbmbbbbbbcbbbbbdbbbbbbbbbb
Album: ccccccccncccccccccccccccccccc
Anno: 2009
Commento: ddddddddddddddddddddddddddddD
Genere: Blues
Errori rilevati
1 - Syntax Error 1 at [1, 39]: Found CHAR ('A') - extraneous input 'A' expecting CANC
2 - Syntax Error 1 at [1, 75]: Found CHAR ('B') - extraneous input 'B' expecting CANC
3 - Syntax Error 1 at [1, 111]: Found CHAR ('C') - extraneous input 'C' expecting CANC
```

- Carattere mancante in Anno

```
Anno: 200<missing CHAR>
```

```
Errori rilevati
1 - Syntax Error 1 at [1, 117]: Found CANC - missing CHAR at '#'
```

- Genere mancante o non riconosciuto

```
Genere: not recognised
Parsing completato con successo
```

- Errata scrittura della head di titolo (es. -> tit: #tit #ti: #tito:)

```
Errori rilevati
1 - Syntax Error 1 at [1, 4]: Found TITOLO_HEAD ('tit:') - no viable alternative at input 'tit:'
```

```
line 1:7 mismatched character 'a' expecting ':'
```

```
Errori rilevati
1 - Syntax Error 1 at [1, 9]: Found CHAR - missing TITOLO_HEAD at 'a'
2 - Syntax Error 1 at [1, 38]: Found CANC - missing CHAR at '#'
```

```
line 1:6 mismatched character ':' expecting 't'
```

```
Errori rilevati
1 - Syntax Error 1 at [1, 8]: Found CHAR - missing TITOLO_HEAD at 'a'
```

```
line 1:7 mismatched character 'o' expecting ':'
```

```
Errori rilevati
1 - Lexical Error 0 at [1, 9]: Found SCAN_ERROR (':') - mismatched input ':' expecting TITOLO_HEAD
```

- Alcune considerazioni
  - Durante lo sviluppo della grammatica ci siamo accorti che senza una chiara distinzione dei frame è molto difficile riuscire a fare una rilevazione degli errori accurata.

  - Nello standard ID3 in esame, infatti, non c'è una chiara distinzione tra il frame Titolo ed il frame Artista, se quindi dovesse mancare un byte nel Titolo la cosa si rifletterebbe nell'Artista e così via a causa della sequenzialità dei dati

  - Per ovviare e operare una buona gestione degli errori abbiamo aggiunto allo standard l'utilizzo delle “head” con caratteri unici non utilizzabili all'interno dei frame, in modo da riconoscere il tipo di errore e la sua localizzazione

  - Nonostante ciò, la grammatica è pensata per riconoscere automaticamente se stiamo trattando delle informazioni con le “head” oppure senza, evitando di doverlo specificare. L'unica opzionalità è lasciata al metodo con il quale estrapolare i dati dai file.