

Documentazione eF API 3.0 - Specifica EFTL© (elixForms© Tag Language) - Versione Cliente

Indice cambiamenti		
Versi one	Modifica apportata	Revie wer
1.0.0	Stesura iniziale con le prime funzionalità EFTL©	RP
1.7.3	Aggiunto info su consultazione log come da commento in EWF-90	RV
1.8.0	Aggiunta funzione SPLIT, throw di StackOverflow in WHILE, iterable type a VAR	RP
2.0.0	Aggiornamento infrastruttura a jdk11	RP
2.1.0	Aggiunta della funzione SIZE_OF per collezioni e potenziamento di VALUE_OF con attributo "index"	RP
2.1.1	Aggiunta attributo threshold a while tag	RP
3.0.0	Adeguamento a piattaforma JDK 17+, Jakarta 10+	RP
3.0.1	Introduzione currentDateTime, currentLocale, defaultLocale, types e formats	RP
3.0.2	Si introduce il TRIM tag	RP
3.0.3	Si introduce il concetto di direttiva e si corrge bug che potrebbe modificare a string il type di variabili number a seguito dell'uso di Value_of; introduzione in VAR di attribute unique	RP
3.1.0	Introduzione di tag CONTAINS	RP

- [INTRODUZIONE](#)
- [RELAZIONI LOGICHE E DIPENDENZE](#)
 - [EFTL Statements declared in EFTL Parser](#)
 - [EFTL whole language fragments relationships \(both Lexer and Parser\)](#)
- [SINTASSI](#)
 - [I tags EFTL©](#)
 - [Regole per la giusta sintassi](#)
 - [Buone pratiche di scrittura del codice](#)
 - [Annidamento](#)
 - [Elementi](#)
 - [Attributi](#)
 - [Struttura di base di una pagina EFTL©](#)
- [Contesto Esecutivo](#)
- [EFTL TAGs](#)
 - [TAG EFTL](#)
 - [TAG HEADER](#)
 - [TAG COMMENTO](#)
 - [TAG LOG](#)
 - [TAG VAR](#)
 - [TAG VALUE_OF](#)
 - [TAG CONTAINS](#)
 - [TAG FORMAT](#)
 - [TAG IS_EMPTY, IS_NOT_EMPTY](#)
 - [TAG SIZE_OF](#)
 - [TAG SPLIT](#)
 - [TAG TRIM](#)
 - [TAGs Elixforms© "TAG" e JSON-TAG](#)
- [STATEMENTS](#)
 - [Costrutti condizionali](#)
 - [TAG IF/THEN/ELSE IF/ELSE \(non case sensitive\)](#)
 - [Costrutti iterativi](#)
 - [TAG WHILE/DO](#)
 - [TAG FOR](#)
 - [BLOCCHI DI CODICE](#)
 - [BLOCCO DI ASSEGNAZIONE](#)
 - [BLOCCO VALUTATIVO](#)
 - [BLOCCO DI ESPOSIZIONE](#)

[Examples](#)

INTRODUZIONE

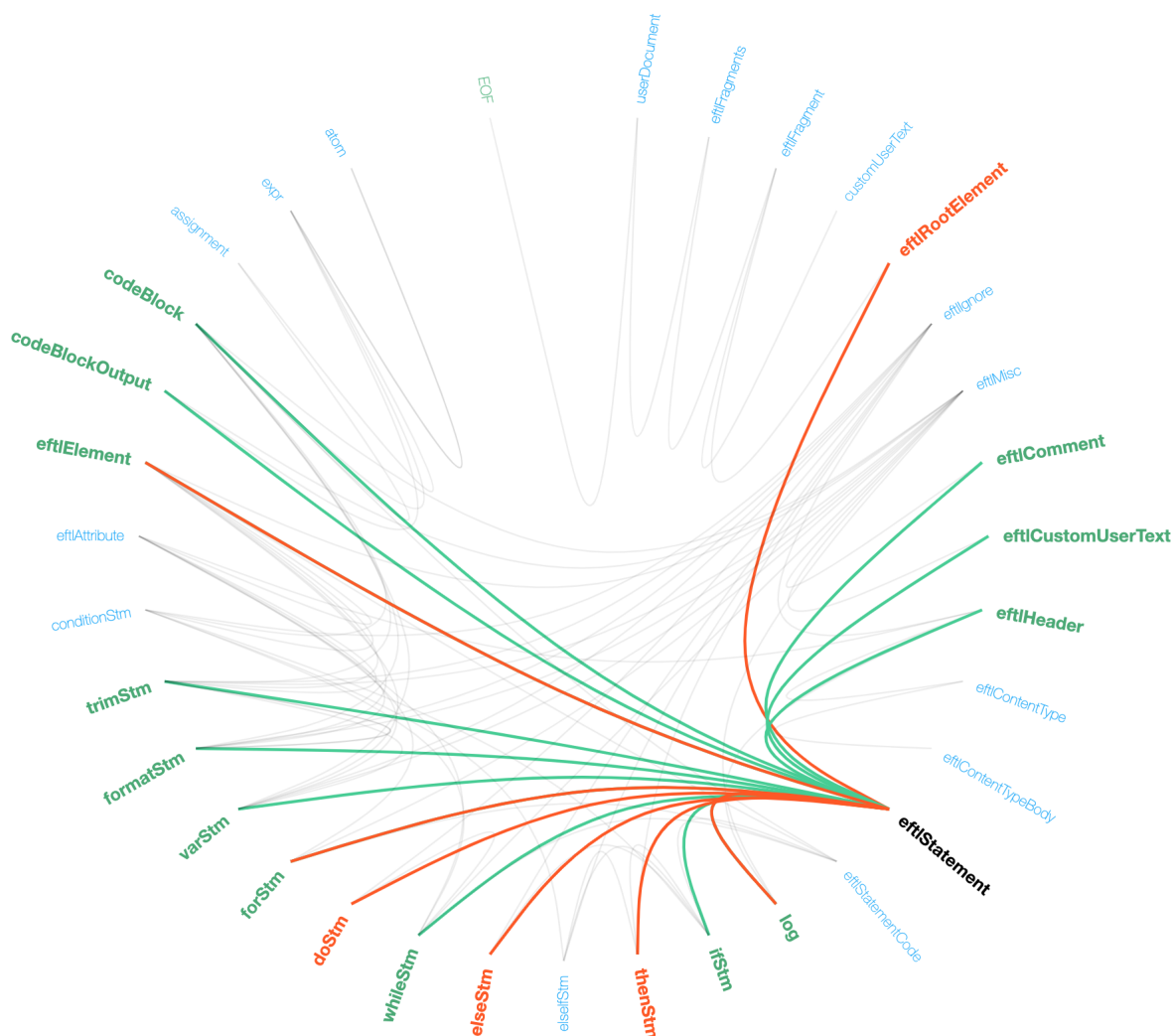
Per soddisfare le recenti richieste di **interazione** tra un contesto di business, specifico per ogni cliente (detto "*user context*") e il sistema di risoluzione di informazioni conosciuto come **TagEngine** di Elixforms®, via costrutti già noti nell'utilizzo dei moduli quali [TAG]...[/TAG], un sistema logico potente e agnostico, denominato **EFTL®** (Elixforms® Tag Language) è stato pensato per tradurre un semplice ed, al tempo stesso, essenziale linguaggio a tag, identificati dai caratteri quadra, "[" e "]" (es. [MIO_TAG attr1="value 1"... attrN="value n"] ... [/MIO_TAG]), in applicazioni logiche decisionali atte a "compilare" a runtime, ossia durante la processazione e produzione di documentazione, il contesto utente completandolo con informazioni proveniente dalla modulistica Elixforms®.

Il linguaggio EFTL® è utilizzabile direttamente dai clienti di tipo amministratore Elixforms® di livello intermedio: è studiato in modo tale da poter essere scritto anche da gestori di modelli o procedimenti senza approfondite conoscenze informatiche.

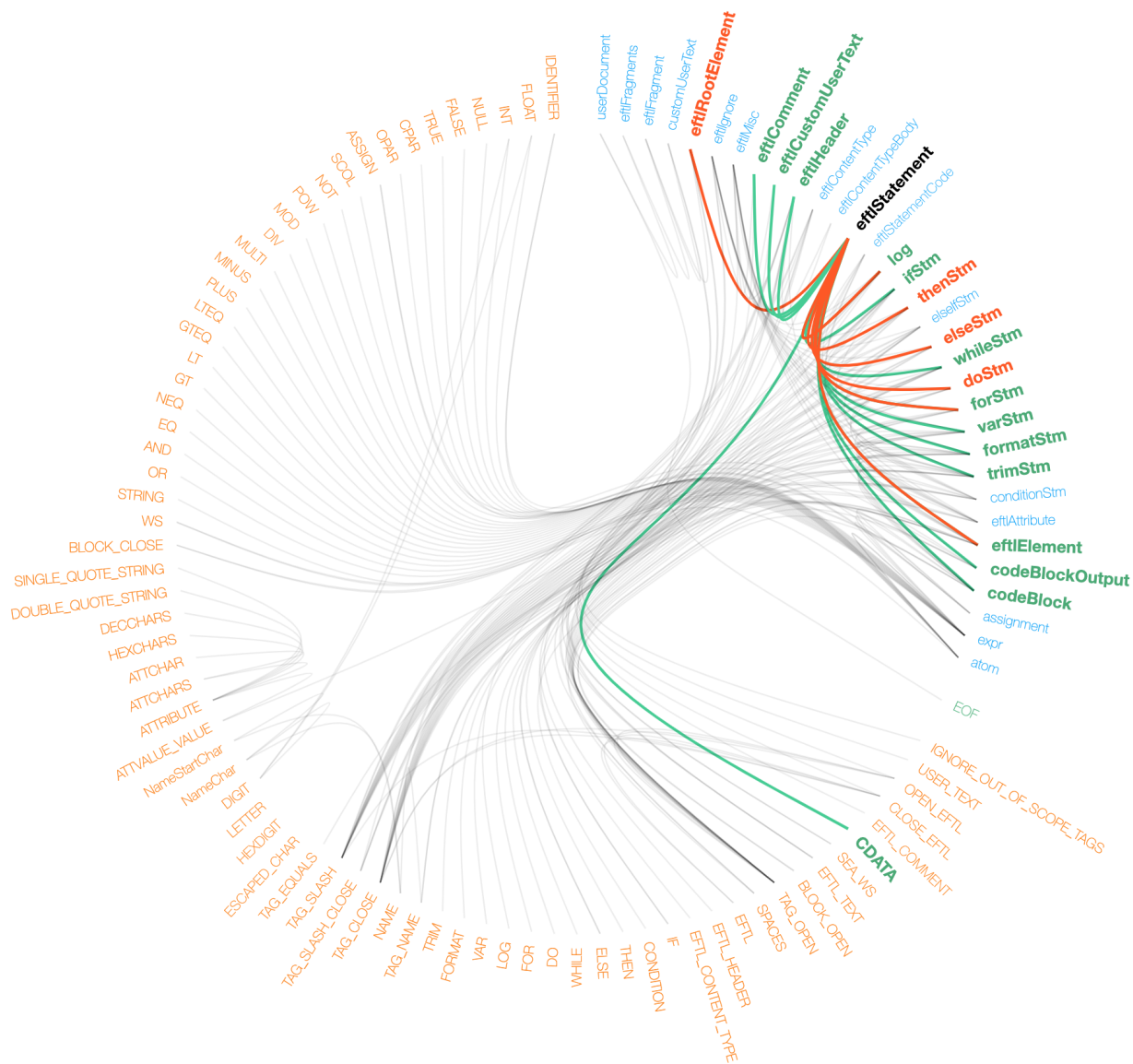
Il principio di base su cui si fonda EFTL®, infatti, è la sintassi dei tags [TAG]...[/TAG], già nota ad un amministratore Elixforms® ed un modello di scrittura concettualmente simile a quello delle formule di fogli di calcolo quali, MS Excel, OSX Pages o LibreOffice, che sono molto conosciuti nel mercato business in generale. Nel presente documento vedremo, infatti, come utilizzare il servizio, mentre rimandiamo all'utilizzo del servizio alla documentazione dedicata: Il linguaggio EFTL è uno dei servizi raggiungibili tramite funzionalità **eF API 3.0 - Servizi per l'interoperabilità applicativa** dove viene descritto l'utilizzo del servizio REST, il suo pathInfo e come invocarlo. Essendo un servizio esposto, significa che può essere invocato da qualsivoglia contesto utente con accesso alle API 2.0 di Elixforms® e avente questa estensione attiva.

RELAZIONI LOGICHE E DIPENDENZE

EFTL Statements declared in EFTL Parser



EFTL whole language fragments relationships (both Lexer and Parser)



Eftl© statements syntax relations



EFTLParser syntax rels.pdf

Versione HTML con SVGs: [eftlParser.zip](#)

Versione standard Dot files (Graphviz standard): [EFTL dot files.zip](#)

SINTASSI

La sintassi è molto simile all'html, nel concetto, sebbene il carattere identificativo di un tag EFTL© siano i caratteri "[" e "]" (e non... <>).

I tags EFTL©

I tag sono i marcatori che consentono al motore EFTL di dare struttura al contenuto del documento processato. I tag sono identificati da nomi diversi a seconda della funzione per cui sono programmati.

I "comandi" (tags) hanno una forma sintattica particolare, es:

- [eftl] *contenuto dell'elemento*[/eftl]
- [tag] *contenuto dell'elemento*[/tag]

Ogni elemento è composto da due tag, uno di apertura e uno di chiusura

Il primo tag <nome dell'elemento> dice al motore EFTL che in quel punto del documento processato inizia la marcatura di qualcosa che deve essere interpretato in qualche modo.

Il secondo tag, quello di chiusura, è caratterizzato dal simbolo "/" e dice al motore EFTL che la rappresentazione dell'elemento termina in quel punto. Con lo stesso principio del secondo tag, quando il contenuto dell'elemento non esiste, o per costruzione semantica non è possibile inserirlo, il primo tag viene chiuso con /: in tal caso si parla di tag senza corpo o *bodyless* o *empty*

Regole per la giusta sintassi

Una giusta sintassi aiuta a creare e rispettare gli standard a vantaggio della costruzione e manutenzione dei documenti, soprattutto quando si gestisce in gruppo.

Le regole per una giusta sintassi sono:

- i documenti devono essere **ben formati**
- **annidare** i tag *in modo corretto e non sovrapposti*
- per i nomi di elemento usare le minuscole (sebbene le maiuscole siano interpretate ugualmente)
- usare sempre i tag di chiusura
- racchiudere gli attributi tra apici ="attributo"
- chiudere gli elementi vuoti con /

Buone pratiche di scrittura del codice

- aderire agli standard
- nomi di file: creare uno standard
- utilizzare il markup semantico
- validare il codice

Annidamento

L'annidamento è la marcatura di un elemento all'interno di un altro elemento.

Gli elementi di blocco possono annidare altri elementi di blocco ed elementi in linea; gli elementi inline, possono annidare solo altri elementi inline, anche se la proprietà `display` in CSS fosse impostata come `display: block`;

L'ordine di annidamento: l'ultimo tag aperto deve essere il primo tag ad essere chiuso; se un elemento "a" deve essere posto all'interno di un elemento "b", allora il tag di chiusura di "a" deve precedere il tag di chiusura di "b".

L'elemento annidato prende il nome di **"figlio" (child)**, l'elemento contenitore quello di **"genitore" (parent)**.

Si dice relazione padre-figlio se l'annidamento è di un livello; relazione antenato discendente, se l'annidamento è di due o più livelli.

Elementi

Conoscere a quale categoria appartiene un elemento è importante per creare pagine EFTL e applicare in modo corretto il linguaggio.

In EFTL© avremo solo elementi di tipo:

- blocco o **block**
- in linea o **inline**

A differenza dell'html non vi saranno, almeno per il momento, elementi rimpiazzati e non.

Attributi

L'attributo di un elemento indica una proprietà di quell'elemento. Spesso gli attributi sono opzionali, in altri casi il comportamento predefinito del motore è quello desiderato, quindi l'attributo non è necessario. Gli attributi perciò è meglio usarli solo quando servono.

Gli attributi si possono indicare in qualsiasi ordine, ma è preferibile crearsi uno standard nello scrivere il codice così da renderlo più facile da scrivere, leggere e mantenere.

È buona norma scrivere gli attributi in minuscolo e i valori posti tra apici.

Struttura di base di una pagina EFTL©

L'esempio che segue è il tipico inizio di un blocco EFTL: lo standard applicato per la tabella codici carattere è il più semplice charset="utf-8" ed al momento non è sovra-scrivibile.

Come si nota nel documento di tipo html seguente, sono stati innestati 2 blocchi EFTL© che verranno processati; possiamo in linea generale considerare che:

- il documento da processare deve avere ALMENO un blocco EFTL per essere preso in carico dal motore: in caso contrario il documento viene riportato as is
- il codice al di fuori dei blocchi di apertura e chiusura EFT **non** verranno interpretati in alcun modo: eventuali scritture quali quelle della riga 14: "... <p style="text-align: center; font-size: 14pt;">[TAG]SCHEMAID,374,COL0004,ID_OBJECT, , [/TAG]</p> ..." saranno riportate come semplice testo
- il documento deve essere sintatticamente corretto
- tutte le scritture tra tag EFTL, ove permesso sintatticamente, saranno riportate come semplice testo

Esempio blocchi EFTL

```
<html>
<!-- codice NON processato da EFTL -->
  <head>
    </head>
    <body style="font-family: Tahoma, Verdana, Segoe, sans-serif; font-size: 12px; font-
style: normal; line-height: 19.6px;">
      <p style="text-align: center; font-size: 13pt; padding-top: 5pt;">
        <a href="#ITALIANO">VERSIONE ITALIANA</a> -
        <a href="#DEUTSCH">DEUTSCHE FASSUNG</a> -
        <a href="#ENGLISH">ENGLISH VERSION</a>
      </p>
      <p style="padding-bottom: 5pt;"> </p>
      <h2 align="center"><strong>TITOLO DEL DOCUMENTO</strong><br /> <strong>TITEL DES
DOKUMENTS</strong><br /> <strong>TITLE OF DOCUMENT</strong></h2>
      <p style="text-align: center; font-size: 14pt; padding-top: 25pt;"><strong>PROCEDURA di
VALUTAZIONE COMPARATIVA</strong></p>
      <p style="text-align: center; font-size: 14pt;">per il conferimento di</p>
      <p style="text-align: center; font-size: 14pt;">[TAG]SCHEMAID,374,COL0004,ID_OBJECT, , [
/TAG]</p>
      <p style="text-align: center; font-size: 14pt;"><strong>ASSEGNO/I a tempo determinato
per la COLLABORAZIONE ad ATTIVITÀ di RICERCA </strong></p>
      <p style="text-align: center; font-size: 14pt; padding-top: 5pt;">Decreto del Rettore<
/p>
      <hr />
      ...

[eftrl] [!-- Inizio blocco EFTL --]
[!-- Blocco di codice riportato in output come scrittura non EFTL --]
<tag>testo tra tag html</tag> [!-- Blocco di codice riportato in output come scrittura non EFTL
--]
```

```

testo senza tags

[VAR name="i" type="number"] [!-- Inizio blocco VAR per la definizione di una variabile --]
  [% i = 0; %] [!-- Blocco di assegnazione valore ad una variabile --]
[/VAR]

<p>un po' di testo che verr&agrave; riportato</p> [!-- Blocco di codice riportato in output
come scrittura non EFTL --]

[IF] [!-- Inizio blocco if/then/eelse --]
  [CONDITION] [!-- Condizione del blocco IF da valutare: risultato aspettato di tipo
booleano --]
    [% lang == "IT" %]
  [/CONDITION]
  [THEN] [!-- Se blocco condizione true --]
    [WHILE] [!-- Blocco while --]
      [CONDITION] [% docs != null && i < docsSize %] [/CONDITION] [!--
Condizione Blocco while --]
        [DO] [!-- Blocco while da eseguire se condizione è "true" --]
          <p> value of var "i" is: [%= i %]</p>

          [FOR varName="doc" iterable="docs"] [!-- Blocco FOR di
iterazione --]
            doc corrente: [%= doc %] [!-- Blocco esposizione del
valore della variabile --]
              <strong> Sto ciclando sul documento con nome e prezzo <
/strong>

            [/FOR]

          [% i = i + 1; %] [!-- Blocco di assegnazione valore ad una
variabile --]
            <p> value of var "i" after increment is: [%= i %]</p>

          [/DO]
        [/WHILE]
      [/THEN]
    [ELSE] [!-- Se blocco condizione false --]
      <strong>fixed-time RESEARCH ASSISTANT CONTRACT/S for
COLLABORATION at RESEARCH ACTIVITY</strong>
    [/ELSE]
  [/IF]
[/EFTL] [!-- Fine blocco EFTL --]

<!-- codice NON processato da EFTL -->
...
<p>testo tra un documento blocco EFTL&copy; e l'altro</p>
...

[EFTL] [!-- Inizio 2° blocco EFTL --]
  <ol>
    <li>
      Decreto del rettore del [TAG]SCHEMAID,202,COL0030,ID_OBJECT, , [/TAG]
      nr. [TAG]SCHEMAID,202,COL0022,ID_OBJECT, , [/TAG]
    </li>
    <li>
      Decreto del rettore nr.
      [TAG]GETVALUEBYTAG,NUM_PROT,REQUEST,IUQOID[FILTER]
ACTION_NAME=pers_a_inviaPerFirmaRemotaBando_facEcon[/FILTER][[/TAG]
      del
      [TAG]GETVALUEBYTAG,PROVV_DATE,REQUEST,IUQOID[FILTER]
ACTION_NAME=pers_a_inviaPerFirmaRemotaBando_facEcon[/FILTER],,,dd/MM/yyyy[/TAG]
    </li>
  </ol>
  <tag>altro tag</tag>

[/EFTL] [!-- Fine 2° blocco EFTL --]

<!-- codice NON processato da EFTL -->

```

```
<p align="center">
    <em>Resto del documento...</em>
</p>
<p> </p>
<p>The Rector</p>
<p>Digitally signed</p>
</body>
</html>
```

Contesto Esecutivo

Il contesto esecutivo, o contesto utente, rappresenta essenzialmente una zona di memoria dove vengono messe a disposizione ai blocchi di valutazione i valori funzionali alla risoluzione del outcome di business (il documento processato, in questo caso) e recuperabili via "chiavi univoche" rappresentate da stringhe: un esempio è quello del tag VAR, spigato poco più avanti nel documento, che permette di interagire in un blocco valutativo EFTL con il contesto utente andando a salvare con il nome dato all'attributo "name", e trattato come chiave univoca, un valore dinamico, risultato della valutazione degli statements processati nel contesto operativo stesso.

Alcuni valori di contesto sono inizializzati staticamente come:

- **currentDateTime** data ora di inizio processazione del documento
- **defaultLocale** "it_IT" ossia la rappresentazione in string da Locale di default, rappresentante della nazione, linguaggio, dialetto e varianti del contesto esecutivo (può essere personalizzato utilizzando currentLocale)
- **currentLocale** impostato dall'utente, su invocazione del servizio, indica un Locale specifico dal utilizzare durante la valutazione dei blocchi: se il valore non dovesse essere un Locale valido o definito, verrà utilizzato il defaultLocale
- **requestId** id della richiesta Elixforms® del contesto corrente

Eventuali altri valori che hanno senso solo nella chiamata di valutazione del documento processato, devono essere inviati con la richiesta al servizio di processazione.

In particolare:

- **unescapeOutcome** (true/false): per effettuare html4 unescaping sul documento processato in fase di rientro
- **defaultContentType** (media type valido): supportato in questo momento solo "text/html"

Tutte le altre coppie di valori passate alla property json **userContext** del servizio **paramName/paramValue** saranno resi disponibili nel contesto applicativo con il nome dato a paramName; per oggetti complessi utilizzare la serializzazione json

EFTL TAGs

TAG EFTL

[EFTL] ... [/EFTL] o [eftl] ... [/eftl]

Il blocco EFTL/eftl rappresenta il tag **radice**, ossia il contenitore, il cui contenuto rappresenta il "codice" da valutare da parte del motore EFTL.

Il testo utente contenuto all'interno del blocco, ma fuori dal contesto di esecuzione di un qualunque tag EFTL, viene riportato come normale testo facente parte del documento vero e proprio, così come scritto dall'utente.

Le sequenze di tags identificate, invece, verranno valutate per la specifica logica del contesto del singolo tag.

Questo tag al momento NON prevede attributi.

TAG HEADER

[HEADER name="..." value="..." type="..."]

Dalla versione 3.0.3, si aggiunge il tag HEADER con tre attributi **name**, **value** e **type** per permettere la dichiarazione di direttive, che verranno rese disponibili al contest applicativo con chiave "@@+varname".

Direttive

"trimDocument" (@@trimDocument)

- al momento, in versione 3.0.3, è l'unica direttiva definita; viene processata in automatico ad ogni esecuzione del tag EFTL© (entry-point del documento utente) ed simula, in post processazione, il tag [TRIM] in maniera implicita. Il trimming è applicato su tutto il contenuto del documento utente se almeno un blocco EFTL©, una volta valutato, ha richiesto la presente direttiva. I valori ammessi, di tipo booleano, sono true o false.
Es.: [HEADER name="trimDocument" value="true" type="boolean" /]

TAG COMMENTO

I commenti in EFTL© si formano con i caratteri [!-- --]

es: [!-- questo è un commento --]

I commenti sono importanti perché costituiscono dei promemoria per l'autore e possono aiutare a ricordare il perché si sono usate certe particolarità di codice, o punti dove terminano marcature di struttura, o ancora informazioni per altri autori che possono accedere al documento.

I commenti in EFTL© possono essere più lunghi di una riga e non vengono interpretati dal motore EFTL©.

TAG LOG

[LOG] ... [/LOG] o [log] ... [/log]

Il tag LOG/log permette di scrivere il contenuto del tag sul log del server di esecuzione; è possibile innestare blocchi di codice condizionale EFTL da valutare contestualmente.

Questo tag al momento NON prevede attributi.

TAG VAR

[VAR name="..." type="[string, boolean, number, date, object, iterable]" unique="[true/false]"] [% expr = expr; %] [/VAR] o [var] ... [/var]

Il blocco VAR/var permette di definire una variabile con nome rappresentato dall'attributo **name** di tipo **type**; è possibile innestare blocchi di codice condizionale EFTL da valutare contestualmente.

La variabile viene iniettata nel contesto utente corrente così da renderla visibile in un qualunque punto del processo EFTL© dal momento della sua definizione.

Dalla versione 3.0.3 della libreria, è stato aggiunto anche l'attributo "**unique**", con i possibili valori booleani true/false (default: false) e attivo solo per le variabili di tipo "**iterable**": se impostato a **true**, i valori vengono presi una volta sola se ripetuti nell'iterable stesso (per chi conosce java, agisce come un **Set**).

es: [VAR name="i" type="number"] [% i = 1; %] [/VAR]

es: [VAR name="i" type="iterable"][SPLIT regex=";"][TAG]...,IUQ0ID, ;[/TAG][SPLIT][VAR] (il tag fa riferimento ad es, ai valori della colonna specificata dei record di un form multiplo)

es: [VAR name="i" type="iterable" unique="true"][SPLIT regex=";"][TAG]...,IUQ0ID, ;[/TAG][SPLIT][VAR] (il tag fa riferimento ad es, ai valori della colonna specificata dei record di un form multiplo presi una volta sola, se ripetuti)

TAG VALUE_OF

[VALUE_OF varname="..." (index="<another varname>") /]

Scriva il valore della variabile del contesto utente, avente nome assegnato all'attributo **varname**, nel documento di destinazione ove non sia possibile utilizzare un **blocco di esposizione**.

Nel caso di valore di tipo **Iterable** (collezione ciclabile) identificato da **varname**, è possibile ottenere il valore alla posizione "n"-esima, posizione rappresentata dal valore della variabile specificata dall'attributo **index**

es: [VALUE_OF varname="miaVar" /] o [VALUE_OF varname="miaColl" index="i" /]

TAG CONTAINS

[CONTAINS varname="iterable, string" value="[string, number, date]" /]

[CONTAINS varname="iterable, string"][eftlStatement][CONTAINS]

[CONTAINS varname="iterable, string"][% ...expr; %][CONTAINS]

Verifica se una string o una variabile di tipo iterable contengono o il valore ottenuto in value attribute o il valore ottenuto dalla valutazione di uno statement EFTL o di un blocco di codice; value e tag body sono mutui esclusivi con priorità all'attributo value. Se non specificato l'attributo varname o il suo contenuto è null, il contains restituirà:

- se il value di comparazione è null
 - true se null anche il valore nella sua rappresentazione stringa
 - false altrimenti
- se il value non è null e varname è di tipo
 - string: true se varname contiene o è uguale alla rappresentazione del valore in stringa, altrimenti false
 - iterable: true se l'iterable contiene il value, false altrimenti

es:

```
[IF]
...
[VAR name=list type="iterable"]...[/VAR]
...
[CONDITION][CONTAINS varname="list" value="aValue" /][CONDITION]
[THEN] contains: true [/THEN]
[ELSE] contains: false [/ELSE]
[/IF]
...
[IF]
...
[VAR name="list" type="iterable"]...[/VAR]
[VAR name="anIterable" type="iterable"]...[/VAR]
[VAR name="atIndex" type="number"]...[/VAR]
...
[CONDITION][CONTAINS varname="list"][VALUE_OF varname="anIterable" index="atIndex" /][CONTAINS][CONDITION]
[THEN] contains: true [/THEN]
[ELSE] contains: false [/ELSE]
[/IF]
```

TAG FORMAT

[FORMAT varname="..." type="[string, number, date]" /]

[FORMAT type="[number, date]" pattern="..."[% ...expr; %] [/FORMAT]

[FORMAT type="[number, date]" pattern="..." ... [/FORMAT]

Tenta di formattare il valore in ingresso a seconda del pattern richiesto.

Il valore può essere ottenuto:

- della variabile del contesto utente, avente nome assegnato all'attributo **varname**: in questo caso il body viene ignorato
- dal valore di ritorno un **blocco di valutazione**
- **valore statico** nel **body**

Al momento sono supportati essenzialmente il tipo:

- **number**: a seconda di come viene trattato il numero il risultato sarà diverso in funzione dei possibili patterns (<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/text/DecimalFormat.html>), dipendenti dal *currentLocale* impostato in contesto utente (o dal *defaultLocale* "it_IT" se non definito)
 - **currency**: imposta automaticamente il formato valuta per il Locale corrente (secondo il default it_IT: 1.200,00 €).
 - **integer**: imposta automaticamente il formato intero per il current Locale corrente (secondo il default it_IT: 1200).
 - **double**: imposta automaticamente il formato intero per il current Locale corrente (secondo il default it_IT: 1.200,0).
 - **percent**: imposta automaticamente il formato intero per il current Locale corrente (secondo il default it_IT: 12%).
 - **generic**: imposta automaticamente il formato intero per il current Locale corrente (secondo il default it_IT: 1200.0).
- **date**:
 - formati data validi (<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/text/SimpleDateFormat.html>)

TAG IS_EMPTY, IS_NOT_EMPTY

[IS_EMPTY varname="..." /], [IS_NOT_EMPTY varname="..." /]

Valuta se il valore della variabile del contesto utente, avente nome assegnato all'attributo **varname**, è vuota o nulla; se la variabile è di tipo:

- stringa: **true** se vuota o nulla (i.e. "blank"), **false** altrimenti
- iterabile (collezione, lista, array, etc): **true** se l'elemento iterabile non contiene elementi (i.e. "empty"), **false** altrimenti. Se la variabile non esiste o è nulla, il caso rientra nel caso generale.

- mappa chiave/valore: **true** se l'elemento mappa non contiene alcuna chiave (i.e. "empty"), **false** altrimenti. Se la variabile non esiste o è nulla, il caso rientra nel caso generale.

In caso di **variabile non esistente** nel contesto utente, il risultato sarà positivo ("**true**")

In caso di variabile non specificata nell'attributo, il risultato sarà negativo ("**false**").

La stessa valutazione, ma al **negativo**, viene effettuata per **IS_NOT_EMPTY**.

es: `[IS_EMPTY varname="miaColl" /] o [IS_NOT_EMPTY varname="miaColl" /]`

TAG SIZE_OF

`[SIZE_OF varname="cognomi" /]`

Se il valore, identificato dal nome della variabile specificata in **varname**, è di tipo **Iterable**, questa funzione calcola la dimensione degli elementi della collezione, 0 altrimenti.

es: `[SIZE_OF varname="miaColl" /]`

TAG SPLIT

`[SPLIT regex="..." emptyIfBlank="true/false" ] eftl statement [/SPLIT]`

Se il valore [unescape] restituito dalla valutazione del corpo del tag (l'eftl statement) non è una stringa, viene restituito il valore valutato stesso.

Se il valore, invece, è una stringa, ad essa viene applicata la regular expression specificata nell'attributo **regex**; se la regex è valida il valore computato [unescape] viene separato in tanti elementi quanti sono i gruppi a cui corrispondono al parsing.

Se l'attributo **emptyIfBlank** (valori possibili: true o false [DEFAULT]) è stato definito a **true**, allora la collezione restituita sarà **vuota**, se il valore nel blocco risulterà essere una string **"blank"**, che significa formata da soli caratteri vuoti, spazi, a capo o "tabs". ATTENZIONE: nei contesti in cui avesse senso il valore "vuoto" (aka blank string), se l'attributo emptyIfBlank è specificato a true, si rischia di perdere il valore i-esimo della collezione eventualmente presente di caratteri vuoti!! E' in carico a chi scrive lo script e utilizza lo split in questa modalità usarlo coscienziosamente.

Se la regex NON fosse valida l'esecuzione viene interrotta con una ParsingException.

TAG TRIM

`[TRIM varname="..." /]`

`[TRIM] eftl statement [/TRIM]`

Restituisce il valore troncato (rimuove da fronte e retro del valore spazi bianchi o a capo)

- o di una variabile in contesto applicativo con dato nome da attributo "varname"
- o del contenuto, risultato della valutazione del corpo del tag,

Se l'attributo "varname" fosse valorizzato tanto quanto al blocco del corpo, la precedenza viene data alla valutazione della variabile e, se non trovata (NB: non trovata NON significa valore "null" presente in contesto applicativo!), allora verrà valutato il corpo del tag.

TAGs Elixforms© "TAG" e JSON-TAG

Si rimanda alla documentazione relativa ai plugin di Elixforms©, in particolare del TAG Engine.

STATEMENTS

Costrutti condizionali

TAG IF/THEN/ELSE IF/ELSE (non case sensitive)

```
[IF]
  [CONDITION] [% ... %] [/CONDITION]
  [THEN] ... [/THEN]
  {[ELSE IF]
    [CONDITION] [% ... %] [/CONDITION]
    [THEN] ... [/THEN]
  [/ELSE IF]}
  [ELSE] ... [/ELSE]}
[/IF]
```

Questo tag al momento NON prevede attributi.

I costrutti condizionali sono delle espressioni che consentono di alterare l'usuale modo di esecuzione di un programma e di **eseguire una certa porzione di codice in base ad una "scelta"**.

I costrutti condizionali sono quindi semplicemente il modo di tradurre sotto forma di codice algoritmi simili al seguente:

 SE condizione1
 codice da eseguire se condizione1 è soddisfatta
 SE INVECE condizione2
 codice da eseguire se condizione2 è soddisfatta
 ...
 ALTRIMENTI
 codice da eseguire se tutte le precedenti condizioni non sono soddisfatte

Le condizioni sono espressioni di tipo boolean

Iniziamo con il dire che tutte le "condizioni" (tag [CONDITION] ... [/CONDITION]), il cui contenuto può essere on altro statement o un blocco di codice o la loro combinazione, devono essere espressioni di tipo boolean (ossia il loro risultato finale deve essere **true** o **false**).

Precedenza negli if multipli

Gli if sono valutati in successione e sarà eseguito solamente il primo per il quale la condizione risulterà vera!

'else' e 'else if' non sono obbligatori

Ci può essere un numero arbitrario di **[else if]** (il nostro "SE INVECE"), o nessuno. Diversamente vi può essere uno ed un solo **[else]** .

Istruzioni e blocchi di istruzioni

In generale, al verificarsi di una condizione, il costrutto [if] esegue l'istruzione oppure il blocco di istruzioni che segue. All'interno di un tag si possono nidificare altri costrutti condizionali, in effetti anche altri [if]. Quindi possiamo avere casi come questo:

```
[IF]

  [CONDITION] [% ... %] [/CONDITION]

  [THEN]

    [IF]

      [CONDITION] [% ... %] [/CONDITION]

      [THEN] ... [/THEN]

      [ELSE] ... [/ELSE]

    [/IF]

  [/THEN]

  [ELSE] ... [/ELSE]

[/IF]
```

Costrutti iterativi

Se i costrutti condizionali visti nella precedente sezione rispondono all'esigenza di eseguire diverse parti di un codice subordinatamente ad una determinata condizione, i costrutti iterativi ci consentono di **eseguire ripetutamente un determinato blocco di codice** (che, al solito, potrà essere una singola linea oppure un intero blocco contenuto in tag EFTL© innestati).

I costrutti iterativi definiti in EFTL© sono sostanzialmente 2, comunemente denominati in base alle keywords **while-do** e **for**.

TAG WHILE/DO

Il **ciclo while** esegue una istruzione o un blocco di codice finché rimane verificata una certa condizione o non viene superato il numero massimo di cicli impostato dalla soglia massima con l'attributo non obbligatorio "**threshold**"="32766" (32766 è il limite massimo impostato di default). In italiano diremmo: "fino a quando la condizione è vera esegui il blocco" ecco un esempio:

```
[WHILE (threshold="32766")]
  [CONDITION] [% ... %] [/CONDITION]
  [DO]
    <codice iterato>
  [/DO]
[/WHILE]
```

Questo tag al momento NON prevede attributi.

dove [CONDITION] deve essere una variabile o una *espressione di tipo boolean*. In questo caso, quindi, esprimiamo la volontà di eseguire tutte le istruzioni del blocco ripetutamente fino a quando condizione ha valore **true**.

Interessante capire cosa succede la prima volta che si accede al ciclo. Se [CONDITION] è **false** quando l'esecuzione arriva per la prima volta allo statement [while] il blocco di codice **non** sarà eseguito, neanche una volta.

ATTENZIONE: se la variabile valutata nella condizione si dovesse comportare come una costante, ossia a tutti gli effetti non cambia valore o per un errore programmatico o per un errore logico, quello che si ottiene è un ciclo infinito.

Per ovviare a questa possibilità, verrà sollevata una **StackOverflowError** contenente il testo della **condition** che l'ha provocata, secondo il noto algoritmo "**Squared digit sum**": se la somma dei quadrati delle cifre di un contatore risulta duplicato in una cache locale dedicata, allora abbiamo il verificarsi dell'atteso errore.

TAG FOR

Il **ciclo for** è un costrutto tra i più conosciuti, comune praticamente a tutti i linguaggi e, pur servendo come i precedenti ad eseguire ripetutamente un blocco, fornisce una semplice sintassi per accomodare:

```
[FOR varName="doc" iterable="docs"]
  <codice iterato>
[/FOR]
```

In EFTL®, viene utilizzato con variabili di tipo *"iterabile"*, quali array, collezioni e liste.

Automaticamente, il costrutto cicla sulla variabile iterabile e assegna il valore corrente alla variabile con nome definito in attributo **"varName"**, che viene innestata nel contesto utente per tutta l'esecuzione del ciclo così da renderla disponibile e ri-usabile al blocco di codice eseguito; finito il ciclo la variabile viene distrutta.

BLOCCHI DI CODICE

I blocchi rappresentano delle istruzioni formate da espressioni logiche. Possono essere di tre tipi:

BLOCCO DI ASSEGNAZIONE

```
[% expr = expr; ...; expr_n = expr_n; %]
```

L'assegnazione prevede la possibilità di modificare il valore di una variabile, durante l'esecuzione di un blocco o uno statement.

Viene identificato dalla sequenza di apertura "[%" e chiusura "%]"; l'espressione deve essere seguita dal carattere ";" per stabilire la fine dell'istruzione!

BLOCCO VALUTATIVO

```
[% expr == expr %]
```

Il blocco valutativo esegue espressioni, durante l'esecuzione di un blocco o uno statement e valutandone il risultato in uscita.

Viene identificato dalla sequenza di apertura "[%" e chiusura "%]" viene tipicamente utilizzato all'interno di tag condizione [CONDITION].

BLOCCO DI ESPOSIZIONE

```
[%= <var name> %]
```

Il blocco di esposizione o di "output" permette di scrivere nel documento target il valore attuale di una variabile del contesto utente. Se la variabile non esiste viene sollevato un errore di esecuzione.

Examples

EFTL examples

```
<html>
<head>
</head>
<body
  style="font-family: Tahoma, Verdana, Segoe, sans-serif; font-size: 12px; font-style: normal; line-
height: 19.6px;">
<p style="text-align: center; font-size: 13pt; padding-top: 5pt;"><a href="#ITALIANO">VERSIONE ITALIANA</a> -
<a href="#DEUTSCH">DEUTSCHE FASSUNG</a> - <a href="#ENGLISH">ENGLISH VERSION</a></p>
<p style="padding-bottom: 5pt;"> </p>
<h2 align="center"><strong>LIBERA UNIVERSITÀ DI BOLZANO</strong><br /> <strong>FREIE UNIVERSITÄT BOZEN<
/strong><br /> <strong>FREE UNIVERSITY OF BOZEN-BOLZANO</strong></h2>
<p style="text-align: center; font-size: 14pt; padding-top: 25pt;"><strong>PROCEDURA di VALUTAZIONE COMPARATIVA<
/strong></p>
```

<p style="text-align: center; font-size: 14pt;">per il conferimento di</p>
 <p style="text-align: center; font-size: 14pt;">[TAG]SCHEMAID,374,COL0004,ID_OBJECT, , [/TAG]</p>
 <p style="text-align: center; font-size: 14pt;">ASSEGNO/I a tempo determinato per la COLLABORAZIONE ad
 ATTIVITÀ di RICERCA </p>
 <p style="text-align: center; font-size: 14pt; padding-top: 5pt;">Decreto del Rettore</p>
 <hr />
 <p style="text-align: center; font-size: 14pt; padding-top: 20pt;">VERGLEICHENDES BEWERTUNGSVERFAHREN<
 /strong></p>
 <p style="text-align: center; font-size: 14pt;">für die Besetzung von</p>
 <p style="text-align: center; font-size: 14pt;">[TAG]SCHEMAID,374,COL0004,ID_OBJECT, , [/TAG]</p>
 <p style="text-align: center; font-size: 14pt;">Stelle/n als FORSCHUNGSASSISTENT</p>
 <p style="text-align: center; font-size: 14pt; padding-top: 5pt;">Dekret des Rektors</p>
 <hr />

[EFTL]

<tag>un tag</tag>

testo no tag

[VAR name="i" type="number"] [% i = 0; %] [/VAR]

<p>un po' di testo qui</p>

[IF]

[CONDITION]

[% lang == "IT" %]

[/CONDITION]

[THEN]

[WHILE]

[CONDITION] [% docs != null && i < docsSize %] [/CONDITION]

[DO]

<p> value of var "i" is: [%= i %]</p>

[FOR varName="doc" iterable="docs"]

doc corrente: [%= doc %]

ASSEGNO/I a tempo determinato per la

COLLABORAZIONE

ad ATTIVITÀ di RICERCA

[/FOR]

[% i = i + 1; %]

<p> value of var "i" after increment is: [%= i %]</p>

[/DO]

[/WHILE]

[/THEN]

[ELSE]

fixed-time RESEARCH ASSISTANT CONTRACT/S for
 COLLABORATION at RESEARCH ACTIVITY

[/ELSE]

[/IF]

[/EFTL]

<p>un po' di testo là</p>

[EFTL]

Decreto del rettore del [TAG]SCHEMAID,202,COL0030,ID_OBJECT, , [/TAG]
 nr. [TAG]SCHEMAID,202,COL0022,ID_OBJECT, , [/TAG]

Decreto del rettore nr.

[TAG]GETVALUEBYTAG, NUM_PROT, REQUEST, IUQOID[FILTER]

ACTION_NAME=pers_a_inviaPerFirmaRemotaBando_facEcon[/FILTER][TAG]

del

[TAG]GETVALUEBYTAG, PROV_V_DATE, REQUEST, IUQOID[FILTER]

ACTION_NAME=pers_a_inviaPerFirmaRemotaBando_facEcon[/FILTER],, , dd/MM/yyyy[/TAG]

<tag>altro tag</tag>

[/EFTL]

[EFTL]

Test EWF-107:

```
[VAR name="numbers" type="iterable"] [SPLIT regex=" "]123 1456 1244 2134[/SPLIT] [/VAR]
<p>
```

```
    <ol>
      [FOR varName="number" iterable="numbers"]
        <li>number corrente: [%= number %]</li>
      [/FOR]
    </ol>
```

```
</p>
```

```
[/EFTL]
```

```
<p align="center">
  <em>Referring provisions</em>
</p>
```

```
<ol>
```

```
  <li>For anything that is not expressly considered in the present
    announcement, reference shall be made, where applicable, to the norms
    cited in the preamble of the present decree and any related laws in
    force.</li>
```

```
</ol>
```

```
<p> </p>
```

```
<p>The Rector</p>
```

```
<p>Prof. Paolo Lugli</p>
```

```
<p>Digitally signed</p>
```

```
<p>Bozen-Bolzano, date of registration</p>
```

```
</body>
```

```
</html>
```

```
[EFTL]
```

Test EWF-106:

```
[VAR name="protocolNumber" type="string"][% protocolNumber = "3333"; %][[/VAR]
```

```
<p>Il numero di protocollo di test per il controllo è [%= protocolNumber; %]</p>
```

```
[WHILE]
```

```
  [CONDITION] [% protocolNumber == "3333"; %] [/CONDITION]
```

```
  [DO]
```

```
    <p>Nr. protocollo delibera (nr./anno): [%= protocolNumber %]</p>
```

```
    [!-- comment this line below, if you would like to cause a StackOverflowError in while
      cycle if you would like to test "Squared digit sum" algorithm --]
```

```
    [% protocolNumber = null; %]
```

```
  [/DO]
```

```
[/WHILE]
```

```
[/EFTL]
```

```
[EFTL]
```

Test EWF-128:

```
[VAR name="nomi"] [SPLIT regex="::"]nome1::nome2::nome3::nome4[/SPLIT] [/VAR]
```

```
[VAR name="cognomi"] [SPLIT regex="::"]cognome1::cognome2::cognome3::cognome4[/SPLIT] [/VAR]
```

```
<table>
```

```
  <tr>
```

```
    <th>Nome</th>
```

```
    <th>Cognome</th>
```

```
  </tr>
```

```
[IF]
```

```
  [CONDITION]
```

```
    [IS_NOT_EMPTY varname="nomi" /]
```

```
  [/CONDITION]
```

```
  [THEN]
```

```
    [VAR name="cognomiSize"] [SIZE_OF varname="cognomi" /] [/VAR]
```

```
    [VAR name="i"] [% i = 0; %] [/VAR]
```

```
    [WHILE]
```

```
      [CONDITION] [% i < cognomiSize %] [/CONDITION]
```

```
      [DO]
```

```
        <tr>
```

```

                                <td>[VALUE_OF varname="nomi" index="i" /]</td>
                                <td>[VALUE_OF varname="cognomi" index="i" /]</td>
                            </tr>
                            [% i = i + 1; %]
                        [/DO]
                    [/WHILE]
                [/THEN]
            [/ELSE]
        <tr><td>Nessun nome</td><td>Nessun cognome</td></tr>
    [/ELSE]
[/IF]
</table>
[/EFTL]

```