contactsend

Guida alla personalizzazione dei messaggi in Contactsend

v 1.6-IT – marzo 2020

contactlab

Via Natale Battaglia, 12 - 20127 Milano
info@contactlab.com

Indice del documento

INTRODUZIONE

Personalizzazione

Contactsend permette di personalizzare i messaggi da inviare per ciascun destinatario. La personalizzazione è resa possibile dalla combinazione di un potente linguaggio di scripting, con speciali tag interpretati dal motore di spedizione in fase di generazione del messaggio.

Il presente documento descrive, oltre ai principi di base relativi al linguaggio di scripting in uso, i tag disponibili con relativi esempi di utilizzo. Laddove non diversamente specificato, le informazioni riportate nel documento sono da intendersi utilizzabili in tutti i motori di invio disponibili in Contactlab. La funzionalità SendImmediate di Contactsend può presentare delle eccezioni nel supporto a determinati tag: questi casi sono specificati all'interno della documentazione.

Eventuali approfondimenti sulla terminologia, le modalità d'uso e di configurazione di Contactsend, sono disponibili al seguente indirizzo: https://explore.contactlab.com/guida-send/.

Sorgenti dati disponibili

Alcuni dei tag illustrati nel presente documento hanno lo scopo di consentire – nel processo di personalizzazione – l'utilizzo di informazioni relative al destinatario (ad esempio: nome, cognome, città) estratte dalle sorgenti dati disponibili. Tali informazioni sono da intendersi come disponibili nel record del database di Contactsend

Panoramica delle classi di tag supportate

Di seguito una panoramica delle *classi di tag* supportate da Contactsend, con indicazione dei canali di invio per i quali sono disponibili.

Nota: per approfondimenti ed esempi d'uso dei singoli tag si rimanda ai successivi paragrafi.

	Sintassi			
Classe di tag	**Apertura**	**Chiusura**	**Canali**	**Descrizione**
USER_DATA_TAGS	${	}$	tutti	Consente di mostrare per ciascun destinatario un valore disponibile nella relativa sorgente dati, tipicamente il record del database utilizzato in Contactsend.
INLINE_CONTENT_TAGS	@(	)@	email	Consente di specificare la URL di un contenuto da incorporare nel corpo del messaggio (ad esempio un'immagine).
DYNAMIC_INLINE_CONTENT_TAGS	^(	)^	email	Similarmente a INLINE_CONTENT_TAGS consente di specificare la URL di un contenuto da incorporare nel corpo del messaggio, costruendo la URL a partire da valori generati da altri tag, quali: USER_DATA_TAGS e NOTES_TAGS
TRACKED_URL_TAGS	%{	}%	email	Consente la pre-tracciatura manuale di link presenti nel messaggio al fine di predisporre gli stessi al tracciamento dei click per fini statistici.
DYNAMIC_TRACKED_URL_TAGS	^{	}^	email	Consente, similmente a TRACKED_URL_TAGS, la pre-tracciatura manuale di link presenti nel messaggio, quando la URL è composta di elementi derivanti da altri tag, quali: USER_DATA_TAGS e NOTES_TAGS
ERB_TAGS	<%	%>	tutti	Consente di utilizzare il linguaggio di scripting ERB per permettere la generazione dinamica di parti del template.
EXTERNAL_SOURCE_TAGS	@{	}@	tutti	Consente di recuperare parte del contenuto di un messaggio da una sorgente esterna, tipicamente un frammento html disponibile a una determinata URL.
INCLUDE_URL_TAGS	<@	@>	tutti	Similmente ad EXTERNAL_URL_TAGS Consente di recuperare parte del contenuto di un messaggio da una sorgente esterna, quando la URL è composta di elementi derivanti da altri tag, quali: USER_DATA_TAGS e NOTES_TAGS
NOTES_TAGS	<{	}>	tutti	Permette di includere il valore di una delle etichette associate a una spedizione in Contactsend.

Ordine di esecuzione

Per comprendere al meglio l'utilizzo dei diversi tag, è necessario tenere conto dell'ordine in cui gli stessi vengono processati dai motori di spedizione. Ciò è particolarmente importante nel caso in cui si voglia utilizzare un tag come elemento nella costruzione di un altro tag, come ad esempio l'utilizzo di un parametro (l'email del destinatario) nella costruzione di un link dinamico da tracciare.

Di seguito è riportata la sequenza di estrazione/computo delle classi di tag per ognuna delle componenti di un messaggio.

EMAIL

CORPO HTML	CORPO SOLO TESTO	OGGETTO	MAIL-FROM	REPLY-TO
1. external_source_tags	1. external_source_tags	1. user_data_tags	1.user_data_tags	1. user_data_tags
2. inline_content_tags	2. tracked_url_tags	2. erb_tags		
3. tracked_url_tags	3. notes_tags			
4. notes_tags	4. user_data_tags			
5. user_data_tags	5. erb_tags			
6. erb_tags	6. include_url_tags			
7. include_url_tags	6a. tracked_url_tags			
7a. tracked_url_tags	6b. user_data_tags			
8. dynamic_inline_content_tags	7. dynamic_tracked_url_tags			
9. dynamic_tracked_url_tags	8. user_data_tags			
10. user_data_tags				

SMS

CORPO DEL MESSAGGIO
1. user_data_tags
2. include_url_tags
2a. tracked_url_tags
2b. user_data_tags
3. erb_tags

TAG

Di seguito, ciascuna delle classi di tag disponibili viene descritta dettagliatamente.

Classe "USER_DATA_TAGS"

Contactsend, utilizzabile in: *EMAIL (corpo html, corpo solo testo, oggetto, mail-from, reply-to), SMS.*

Descrizione: i tag della classe "USER_DATA_TAGS" permettono di inserire nel messaggio valori relativi al singolo destinatario. Nel caso di Contactsend, questi valori sono contenuti nei campi del record del database associati al gruppo email che si sta utilizzando per la spedizione.

Poniamo che un determinato contatto abbia i seguenti attributi, con i relativi valori:

Email	Nome	Cognome
email@example.com	Mario	Rossi

In questo caso potremo inviare al contatto un messaggio email, personalizzando l'oggetto del messaggio, il corpo in versione HTML o in versione testuale con i tag:

```
Gentile ${NOME}$ ${COGNOME}$,
```

In fase di generazione del messaggio, i tag saranno sostituiti con il corrispondente valore. Il valore risultante sarà dunque:

```
Gentile Mario Rossi,
```

Attenzione:

- *in Contactsend, caso di assenza del campo o del valore (ad esempio mancanza del campo nome per un certo destinatario), il tag restituisce una stringa vuota.*
- *Nelle SendImmediate, in caso di assenza del campo, il tag inserito non viene sostituito ma rimane inalterato.*

Modificatori di campo

È possibile modificare la visualizzazione dei valori restituiti dai tag della classe user_data_tags utilizzando dei "modificatori". La sintassi di un modificatore è la seguente:

```
${user_tag|nome_modificatore}$
```

dove il *nome_modificatore* è uno tra:

- default
- date_format

- date_parse
- string_format
- substring
- capitalize
- capitalize_words
- lower (o lowercase)
- upper (o uppercase)
- swapcase
- reverse
- urlencode
- urldecode
- replace
- md5 (o md5sum)
- base64encode
- base64decode
- fail_on_empty

Attenzione: sia i nomi dei modificatori che degli attributi (campi del database in Contactsend) sono "case sensitive", ovvero operano una distinzione tra lettere minuscole e maiuscole. Di conseguenza per essere valido, il nome degli attributi dovrà essere replica fedele di quanto presente a DB Contactsend. Anche i nomi dei modificatori sono da intendersi così come indicati nell'elenco qui sopra, ovvero indicati tutti in minuscolo.

Concatenare i modificatori in una pipeline

I modificatori possono eventualmente essere combinati tra loro in sequenza, creando una c.d. *pipeline*. In questo caso la trasformazione operata da un modificatore diverrà l'input per la successiva trasformazione del modificatore al primo concatenato.

Ad esempio, nell'ipotesi di un attributo "indirizzo" con valore vuoto, è possibile utilizzare la seguente pipeline:

```
${indirizzo|default:"Non specificato"|lowercase|urlencode}$
```

L'espressione risulterebbe in:

```
non+specificato
```

Nel caso invece di un attributo *data_nascita* con valore "2010/04/15", la pipeline

```
 ${data_nascita|date_parse:"%Y/%m/%d"|date_format:"%d-%m-%y"}$
```

darebbe come risultato:

```
15-04-10
```

Di seguito, ciascun modificatore viene descritto in dettaglio.

default

Consente di mostrare un valore predefinito da utilizzare laddove il tag ritorni un valore vuoto o nullo. Poniamo ad esempio di applicare il modificatore in questo modo:

```
${NOME|default:"utente"}$
```

Nel caso in cui l'attributo NOME non fosse valorizzato, verrà mostrato al suo posto la stringa indicata come valore predefinito (tra doppie virgolette). Nel nostro esempio la stringa: *utente*.

Quindi, a fronte di due destinatari:

Email	Nome	Cognome
email1@example.com	Mario	Rossi
email2@example.com		Bianchi

la pipeline,

```
Gentile ${nome|default:"utente"}$
```

avrà come risultato rispettivamente:

```
Gentile Mario
```

e

```
Gentile utente
```

date_format

Consente di formattare un campo data (disponibile in origine nel formato *yyyy-mm-dd*), secondo la seguente tabella:

%d	giorno del mese come numero decimale (range da 00 a 31)
%m	mese come numero decimale (range da 01 a 12)
%y	anno come numero decimale su due cifre (range da 00 a 99)
%Y	anno come numero decimale su quattro cifre

Ad esempio: a fronte di un ipotetico attributo del destinatario denominato *data_nascita* con valore "2010-04-15", il modificatore *date_format* così esplicitato:

```
${data_nascita|date_format:"%d/%m/%Y"}$
```

darà come risultato il valore

```
15/04/2010
```

Nota: nel caso la data non fosse disponibile in origine nel formato previsto (yyyy-mm-dd) è necessario utilizzare il modificatore date_format in pipeline con il modificatore date_parse (vedere di seguito).

date_parse

Consente di convertire una stringa in una data, opzionalmente formattabile con il modificatore *date_format* (vedere sopra). La sintassi per la determinazione del formato originario della data è il medesimo di *date_format*:

%d	giorno del mese come numero decimale (range da 00 a 31)
%m	mese come numero decimale (range da 01 a 12)
%y	anno come numero decimale su due cifre (range da 00 a 99)
%Y	anno come numero decimale su quattro cifre

Dato come esempio un ipotetico attributo *data_nascita* con valore "2010/04/15", il utilizzando il solo modificatore date_parse:

```
${data_nascita|date_parse:"%Y/%m/%d"}$
```

avremo come risultato:

```
 2010-04-15
```

mentre utilizzando il medesimo modificatore in pipeline con date_format:

```
${data_nascita|date_parse:"%Y/%m/%d"|date_format:"%d-%m-%y"}$
```

avremo come risultato:

```
15-04-10
```

string_format

Consente di formattare il valore preso da un attributo numerico, o da un attributo stringa che rappresenta un numero.

Ad esempio**:** per l'attributo *campo_1* con valore 23.5787446, il modificatore:

```
${campo_1|string_format:"%.2f"}$
```

darà come risultato:

```
 23.58
```

laddove il valore_1, coerentemente col formato impostato, è arrotondato al secondo decimale.

substring

Consente di estrarre una porzione della stringa indicando la posizione iniziale (inclusa) e la posizione finale (esclusa). Se la posizione finale è oltre la lunghezza della stringa, il risultato terminerà con la fine della stringa. L'inizio della stringa ha posizione "0" (zero). Dato ad esempio un ipotetico attributo *campo_1* con valore "abcdefg",

```
${campo_1|substring:"0":"5"}$
```

avrà come risultato:

```
abcde
```

mentre:

```
${campo_1|substring:"2":"5"}$
```

avrà come risultato:

```
cde
```

e

```
${campo_1|substring:"2":"100"}$
```

risulterà in:

```
cdefg
```

capitalize

Rende maiuscola la prima lettera della stringa e minuscole le rimanenti. Prendiamo l'ipotesi di un attributo *full_name* con valore "mario Rossi", il modificatore:

```
${full_name|capitalize}$
```

avrà come risultato:

```
Mario rossi
```

capitalize_words

Rende maiuscola la prima lettera di ogni parola e minuscole le rimanenti. Sono considerati separatori di parola i caratteri di spaziatura e i seguenti caratteri:

```
!"#$%&'()*+,-./:;<=>?@[]\^_`{|}~
```

Se ipotizziamo un attributo *full_name* con valore "dolcetto d'alba", il modificatore

```
${full_name|capitalize_words}$
```

avrà quindi come risultato:

```
Dolcetto D'Alba
```

lower (oppure lowercase)

Rende minuscola l'intera stringa. Prendiamo ad esempio un attributo *full_name* con valore " Mario Rossi ". In questo caso il modificatore:

```
${full_name|lower}$
```

avrà come risultato:

```
mario rossi
```

upper (oppure uppercase)

Rende maiuscola l'intera stringa. Nel caso di un attributo *full_name* con valore " Mario Rossi ", il modificatore

```
${full_name|upper}$
```

avrà quindi come risultato:

```
MARIO ROSSI
```

swapcase

Consente di invertire lettere minuscole in maiuscole e viceversa. Se ipotizziamo quindi un attributo *full_name* con valore "Mario Rossi", il modificatore

```
${full_name|swapcase}$
```

avrà come risultato:

```
mARIO rOSSI
```

reverse

Inverte la stringa. Nel caso di un campo *tipo_autoveicolo* con valore "Ambulanza", il modificatore

```
${tipo_autoveicolo|reverse}$
```

avrà di conseguenza come risultato:

```
aznalubmA
```

urlencode

Codifica la stringa in modo da renderla utilizzabile nella costruzione di una URL. Nel caso di un attributo *folder_name* con valore "/Utente/RossiMario", il modificatore

```
${folder_name|urlencode}$
```

avrà di conseguenza come risultato:

```
%2FUtente%2FRossiMario
```

urldecode

Modificatore che applica la trasformazione Inversa di urlencode, decodifica una stringa. Ad esempio, preso un attributo *folder_url* con valore "%2FUtente%2FRossiMario", il modificatore:

```
${folder_url|urlencode}$
```

darà come risultato:

```
/Utente/RossiMario
```

replace

Permette di operare una sostituzione di una stringa con un'altra (la c.d. operazione di "Trova e sostituisci"). Il primo parametro rappresenta la stringa da sostituire, il secondo la stringa da usare per la sostituzione.
Nota: il modificatore è *case sensitive*, ovvero opera una distinzione tra minuscole e maiuscole. Ad esempio, applicando ad un attributo *full_name* con valore "Mario Rossi", il modificatore:

```
${full_name|replace:"Mario":"Luigi"}$
```

otterremo una stringa quale:

```
 "Luigi Rossi"
```

md5 (oppure md5sum)

Calcola il checksum md5[1] di una stringa. Ad esempio, per l'attributo *full_name* con valore "Mario Rossi", il modificatore

[1] https://it.wikipedia.org/wiki/MD5

```
${full_name|md5}$
```

darà come risultato:

```
a5887a62d652d2b476e57f20bbbc8c2c
```

base64encode

Codifica una stringa secondo il sistema di codifica Base64[2]. Ad esempio, per l'attributo *full_name* con valore "Mario Rossi", il modificatore:

```
${full_name|base64encode}$
```

darà come risultato la stringa:

```
TWFyaW8gUm9zc2k=
```

base64decode

Applica l'operazione inversa di *base64encode*, ovvero decodifica una stringa codificata secondo il sistema di codifica Base64[2].

fail_on_empty

Modificatore che causa il fallimento dell'invio del messaggio nel caso in cui venga applicato ad un valore vuoto o nullo. Se applicato ad un valore non vuoto e non nullo, il modificatore ritorna il valore stesso dell'attributo. Ad esempio:

```
${nome|fail_on_empty}$
```

manderà un errore (causando il mancato invio del messaggio), nel caso l'attributo *nome* risulti vuoto o nullo.

INLINE_CONTENT_TAGS e DYNAMIC_INLINE_CONTENT_TAGS

Utilizzabile in: *Email (corpo html)*

[2] https://it.wikipedia.org/wiki/Base64

Questi tag vengono utilizzati per recuperare un'immagine presente all'URL indicato, e incorporarla nel messaggio. Ad esempio, rispettivamente:

```
@(http://nostrosito.com/img/foto1.jpg)@
```

e

```
^(http://nostrosito.com/img/${provincia}$/foto1.jpg)^
```

(NOTA: il secondo esempio non è gestito dalle SendImmediate)

L'uso di immagini incorporate all'interno delle email (o "embedded"), è una pratica che richiede cautela. Se da un lato rende le immagini immediatamente visibili (sono codificate all'interno del messaggio stesso, non allegate né scaricate dall'esterno), dall'altro provoca l'aumento delle dimensioni del messaggio e aumenta grandemente il rischio di blocco da parte di sistemi anti-spam.

TRACKED_URL_TAGS e DYNAMIC_TRACKED_URL_TAGS

Utilizzabile in: *Email (corpo HTML, corpo solo testo), SMS*

Questa sintassi viene di norma inserita automaticamente da Contactsend durante il processo di creazione di una spedizione email, per permettere poi il tracciamento dei link a fini di conteggio dei relativi click. La sintassi *TRACKED_URL_TAGS* può però essere inserita manualmente: si parla in questo caso di pre-tracciatura dei link. Supponiamo ad esempio di voler tracciare URL contenuti in frammenti inclusi da sorgenti esterne (tramite i tag della classe *include_url_tags*): in questo caso la pre-tracciatura dei link è necessaria per abilitare il conteggio dei risultanti click. Il formato da utilizzare è il seguente:

```
%{Descrizione[1]%%http://www.example.com/page.html}%
```

dove la "Descrizione" è una stringa di testo che sarà usata per etichettare l'URL, mentre il valore "[1]" rappresenta la categoria associata al link. Questi due valori (Descrizione e categoria) sono poi mostrati nell'area Controlla di Contactsend, nel set di statistiche della spedizione.

Informazioni sul tracciamento dei link sono anche disponibili nella seguente pagina della documentazione di Contactsend https://explore.contactlab.com/tracciare-a-mano-i-link/

I tag URL *DYNAMIC_TRACKED_URL_TAGS* vanno utilizzati nel caso in cui l'URL del link debba contenere parametri dinamici, e questi **non** siano posizionati in fondo all'URL stesso, come nel seguente esempio:

```
^{http://nostrosito.com/${provincia}$/docs/index.html}^
```

oppure nel caso che **l'intero** URL sia il valore di un attributo:

```
^{${dominio}$}^
```

ATTENZIONE: questa funzione è sconsigliata in caso vi sia una elevata cardinalità delle possibili permutazioni (in particolare per spedizioni inviate ad un grande numero di destinatari). In questi casi infatti verrebbero causati rallentamenti nella preparazione dei messaggi e nel successivo invio; in questo caso, superata una certa soglia di rallentamento la spedizione potrebbe fallire del tutto.

EXTERNAL_SOURCE_TAGS

Utilizzabile in: *Email (corpo HTML, corpo solo testo)*

Consente di includere nel messaggio un contenuto disponibile in un URL esterno. La sintassi è la seguente:

```
@{http://example.com/fragment.html}@
```

In fase di generazione del messaggio, il contenuto estratto dall'indirizzo web sarà incluso nel corpo del messaggio al posto del tag.

Attenzione: il contenuto vie estratto nel momento di generazione del messaggio, non in fase di composizione del medesimo. È quindi necessario accertarsi che in quel momento il file desiderato sia disponibile e coerente con quanto atteso.

INCLUDE_URL_TAGS

Utilizzabile in: *Email (corpo HTML, corpo solo testo), SMS*

Parimenti a EXTERNAL_URL_TAGS, permette di includere un file da un URL esterno, ma supporta la generazione dinamica dell'URL da includere. La sintassi è:

```
<@http://example.com/${attribute}$/fragment.html@>
```

In questo caso, al variare dell'attributo *attribute* verrà incluso un URL generato dinamicamente sulla base del valore dell'attributo medesimo per lo specifico destinatario.

NOTES_TAGS

Utilizzabile in: *Email (corpo HTML, corpo solo testo)*

Permette di includere il valore di una delle etichette spedizione salvate all’interno di Contactsend.

ERB_TAGS

Utilizzabile in: *Email (corpo HTML, corpo solo testo, oggetto), SMS*

Cos'è ERB

ERB (Embedded Ruby[3]) è un **linguaggio di templating** che permette di inserire codice Ruby all'interno di un template (ad esempio l'HTML di un'email) ed eseguirlo. Contactsend utilizza Ruby per la personalizzazione dei messaggi.

La versione utilizzata da Contactsend è la **1.8.6 patch 287**[4]. La documentazione è disponibile all'indirizzo: https://ruby-doc.org/stdlib-1.8.6/libdoc/erb/rdoc/ERB.html

ERB all'interno template

Nei template email inviati con Contactsend, si può usare ERB per eseguire codice Ruby allo scopo di personalizzare i messaggi, nonché per utilizzare attributi del destinatario di un messaggio (all'occorrenza operando sugli stessi operazioni di trasformazione).

Nota: non tutte le funzioni ERB sono rese disponibili. Le limitazioni applicate sono documentate nel paragrafo ***Errore. L'origine riferimento non è stata trovata.*** *di questo documento.*

Per accedere ai dati degli utenti, è possibile utilizzare il metodo `get_user_data` creato da Contactlab a questo scopo. Ad esempio, la sintassi:

```
<%= get_user_data('Gender') %>
```

restituisce il valore del campo Gender presente nel database, per ogni singolo destinatario a cui è indirizzato il template che si sta creando. Il tipo restituito dal metodo sarà il medesimo del campo utilizzato: stringa nel caso di un campo stringa, numero nel caso di un campo numerico, e così via. Il metodo non restituisce mai NULL: se il contenuto del campo è nullo, restituisce una stringa vuota.

Con Contactsend e le SendImmediate è possibile utilizzare una variante del metodo `get_user_data:` il metodo `get_typed_user_data` che funziona in modo del tutto simile, ma permette di convertire il valore del campo nel tipo di dato che si desidera. Ad esempio, la sintassi:

```
<%= get_typed_user_data('Gender', :string) %>
```

converte il valore presente nel campo Gender in una stringa di testo. Si possono specificare quattro diversi tipi di dato:

[3] Maggiori informazioni su ERB sono disponibili all'indirizzo: http://www.stuartellis.name/articles/erb/
[4] Le note di rilascio sono disponibili all'indirizzo: https://www.ruby-lang.org/en/news/2008/08/11/ruby-1-8-7-p72-and-1-8-6-p287-released/

:default

Equivale di fatto ad usare il metodo `get_user_data`. Restituisce il tipo di dato presente nel campo così come esso è rappresentato in origine. Non restituisce mai NULL: se il contenuto del campo è nullo, restituisce una stringa vuota.

:numeric

Restituisce un numero decimale (float) con separatore di decimale punto (es. 12.64). Se il valore non è un numero, o è mancante, o se l'operazione fallisce, viene mostrato 0.0

:string

Restituisce una stringa. Se il valore è mancante, o l'operazione fallisce, viene mostrata una stringa vuota. Se il valore è una cifra, questo verrà riportato come stringa, e quindi trattato non più come numero.

:nillable

Restituisce il tipo di dato presente nel campo così com'è, ma accetta anche NULL. Se il contenuto del campo è nullo, in questo caso viene restituito NULL.

Errori nel codice ERB

I codici ERB eventualmente presenti nel template vengono processati **dall'apposito interprete** dai motori di spedizione di Contactsend nel processo di generazione dei messaggi. Se la sintassi contiene degli errori, il singolo messaggio che li contiene va in errore e non viene inviato al destinatario.

Nel caso di Contactsend quei messaggi saranno riportati nelle statistiche della spedizione (area "Controlla > Archivio spedizioni") alla voce "Invii > Errori". Nel wizard di creazione delle spedizioni email presenti nell'interfaccia web di Contactsend, un messaggio contenente una sintassi ERB errata causerà il fallimento dell'anteprima nel Riepilogo spedizione.

Principali casi di utilizzo di ERB

Ci sono alcune funzioni offerte da ERB tipicamente utili nel template di un messaggio. Di seguito mostreremo a titolo di esempio alcuni dei casi d'uso tipici, con i relativi esempi.

Costrutto IF-ELSE

Questa sintassi consente di rendere visibile una porzione di testo o di codice HTML del template solo a fronte di determinate condizioni, ad esempio per i destinatari che hanno un certo valore all'interno di un campo del database.

Esempio 1

```
<% if get_user_data('sesso') == 'M' then %>
   Caro
<% elsif get_user_data('sesso') == 'F' then %>
   Cara
<% else %>
   Gentile
<% end %>
```

Nota: questa è la medesima sintassi prodotta nei template dall'interfaccia grafica della funzionalità "Condizionale" che è possibile attivata sullo strumento "PageBuilder" di Contactsend.

Esempio 2

```
<% if( (get_typed_user_data('OK_Promo', :numeric) == 1 ) %>
   Approfitta della nostra promozione di questa settimana:
   <a href="http://example.com/promo">clicca qui</a>
   per scoprire l'offerta!
<% end %>
```

In questo esempio la porzione di testo contenente il link verrà inclusa nei soli messaggi indirizzati agli utenti che hanno il valore numerico 1 nel campo "OK_Promo".

Esempio 3

```
<% if( (get_typed_user_data('Store', :string).downcase =~ /milano/) ) %>
  <img src="http://example.com/milano.png" width="280" height="140>
<% end %>
```

In questo secondo esempio, la porzione di codice HTML contenente l'immagine apparirà solo nei messaggi indirizzati ai destinatari per i quali il campo stringa "Store" contenga il valore "milano".

Si può notare che alla stringa viene applicato il modificatore *downcase* in modo che, in qualunque modo sia scritto nel campo di origine, il valore venga restituito tutto minuscolo. In questo modo si è certi che il match con la condizione indicata di seguito (in questo caso "contiene: milano") venga eseguito in modo corretto.

Utilizzo del costrutto "Case"

Questa sintassi risulta utile nel caso si debbano inserire molte condizioni di seguito, difficilmente gestibili con una serie di codici IF-ELSE consecutivi.

Esempio

```
<% case user_data['provincia'] when "MI" %>

       Ti aspettiamo a Milano!

   <% when "RM" %>

       Ti aspettiamo a Roma!

   <% else %>

        Ti aspettiamo in uno dei nostri punti vendita!

<% end %>
```

Ciclare gli elementi di un array

È possibile accedere a un array di oggetti e mostrarli all'interno del template. Questa operazione viene usata, ad esempio, per popolare delle email automatiche di riepilogo degli acquisti fatti su un sito e-commerce, o nella email agli utenti che hanno abbandonato il carrello.

Esempio 1

Supponiamo di avere disponibilità, in due attributi dell'utente, di valori relativi alle informazioni su prodotti abbandonati in un carrello, rispettivamente i nomi dei prodotti (nel campo PRODUCTS) ed i link alla pagina e-commerce dei medesimi (nel campo LINKS).

Nota: nell'esempio si ipotizza che i valori dei singoli prodotti siano presenti nel campo separati dal simbolo pipe (|).

Ecco l'esempio del codice HTML di una ipotetica mail di reminder del carrello:

```
<html>
<body>

<p>Hello ${CUSTOMERNAME}$</p>

<%
# Prepariamo le variabili che useremo nel ciclo for
product_names = get_user_data('PRODUCTS').to_s.split('|')
product_links = get_user_data('LINKS').to_s.split('|')

# Impostiamo i valori di defaul nel caso non vi sia un valore
default_name  = "Il tuo prodotto"
default_link  = "https://example.com"
%>

<table>
  <tr>
    <th>NAME</th>
    <th>LINK</th>
  </tr>

  <% product_names.each_with_index do | product_name, idx|%>

  <tr>
    <td>
        <%= product_name .nil? ? default_name : name_product %>
    </td>
    <td>
        <a href="<%= product_links[idx].nil? ? default_link : product_links[idx] %>">
           clicca qui
        </a>
    </td>
  </tr>

  <% end %>

</table>
</body>
</html>
```

Nel caso un nome di prodotto non sia presente (esempio: "Scarpa|Vestito||Sciarpa"), verrà mostrato nel template il testo specificato dalla variabile default_name; allo stesso modo in assenza di un link al prodotto, verrà utilizzato il link di default.

Esempio 2

Questo esempio è simile all'Esempio 1 visto più sopra, ma in questo caso solo il primo prodotto del carrello verrà preso in considerazione.

```
<html>
<body>

<p>Hello ${CUSTOMERNAME}$</p>

<%
# Impostiamo i valori di defaul nel caso non vi sia un valore
default_name  = "Il tuo prodotto"
default_link  = "https://example.com"
%>

<table>
  <tr>
    <th>NAME</th>
    <th>LINK</th>
  </tr>
  <tr>
    <td>
        <%= get_user_data('PRODUCTS').to_s.split('|')[0].nil? ? default_name :
get_user_data('PRODUCTS').to_s.split('|')[0] %>
    </td>
    <td>
        <a href="><%= get_user_data('LINKS').to_s.split('|')[0].nil? ? default_link :
get_user_data('PRODUCTS').to_s.split('|')[0] %>">
          clicca qui
        </a>
    </td>
  </tr>
</table>
</body>
</html>
```

Calcolare valori percentuali

ERB permette di eseguire e mostrare nel template delle operazioni aritmetiche su campi numerici presenti nel database. Tra queste operazioni, c'è il calcolo delle percentuali che può essere utilizzato – ad esempio - per mostrare all'interno del template prezzi scontati per i propri prodotti.

Esempio

```
<%= "%.02f" % [((100 - get_user_data('discount'))/100.0) *
get_user_data('amount')] %>
```

Questa sintassi prende il valore dello sconto dal campo chiamato "discount" (che deve contenere valori interi positivi), lo mette in percentuale, e poi lo applica al valore preso dal campo "amount", anch'esso contenente numeri interi positivi.

In alternativa è possibile usare il metodo *get_typed_user_data* in modo che il valore dei due campi risulti sempre numerico:

```
<%= "%.02f" % [((100 - get_typed_user_data('discount', :numeric))/100.0) *
get_typed_user_data('amount', :numeric)] %>
```

L'impostazione del formato "%.02f" fa sì che le il risultato dell'operazione riporti sempre 2 sole cifre decimali. Il separatore decimale è il punto, come nella notazione anglosassone. Se volessimo utilizzare la virgola, potremmo effettuare una sostituzione come nel seguente esempio:

```
<%= ("%.02f" % [((100 - get_typed_user_data('discount', :numeric))/100.0) *
get_typed_user_data('amount', :numeric)]).gsub(".",",") %>
```

Limitazioni

Limitazioni legate all'architettura di Contactsend

L'architettura di Contactsend impone alcuni vincoli che impattano sull'utilizzo di ERB. I dati relativi agli utenti vengono estratti dal database in cui si trovano i destinatari che riceveranno i template, e solo da quello. **Non sono supportate tabelle in join**, e non è altresì possibile appoggiarsi a tabelle di lookup contenenti degli attributi per creare una relazione "uno a molti". Allo stesso modo non è possibile importare dati dall'esterno, collegandosi a risorse terze attraverso la rete Internet.

Limitazioni legate alla configurazione dell'ambiente Ruby

Nei template utilizzati in Contactsend, il livello di sicurezza Ruby è quello massimo, il **livello 4** della variabile d'ambiente $SAFE[5].

Il livello quattro, oltre alle sue limitazioni specifiche, eredita anche quelle dei livelli inferiori, come descritto dalla documentazione ufficiale Ruby, che riportiamo di seguito in versione originale:

Livello $SAFE	Descrizione
0	No checking of the use of externally supplied (tainted) data is performed. This is Ruby's default mode.
>= 1	Ruby disallows the use of tainted data by potentially dangerous operations.
>= 2	Ruby prohibits the loading of program files from globally writable locations.
>= 3	All newly created objects are considered tainted.
>= 4	Ruby effectively partitions the running program in two. Nontainted objects may not be modified. Typically, this will be used to create a sandbox: the program sets up an environment using a lower $SAFE level, then resets $SAFE to 4 to prevent subsequent changes to that environment.

Il dettaglio delle operazioni vietate in ciascun livello è disponibile in appendice.

[5] A questo proposito si veda http://ruby-doc.com/docs/ProgrammingRuby/html/taint.html

Il dettaglio delle costanti e dei metodi non consentiti è disponibile in appendice.

Appendice

Livelli di SAFE e relative operazioni non consentite

Viene qui riportato il livello di SAFE impostabile in Ruby, con l'elenco delle relative operazioni non consentite.

$SAFE >= 1

- The environment variables RUBYLIB and RUBYOPT are not processed, and the current directory is not added to the path.
- The command-line options -e, -i, -l, -r, -s, -S, and -x are not allowed.
- Can't start processes from $PATH if any directory in it is world-writable.
- Can't manipulate or chroot to a directory whose name is a tainted string.
- Can't glob tainted strings.
- Can't eval tainted strings.
- Can't load or require a file whose name is a tainted string.
- Can't manipulate or query the status of a file or pipe whose name is a tainted string.
- Can't execute a system command or exec a program from a tainted string.
- Can't pass trap a tainted string.

$SAFE >= 2

- Can't change, make, or remove directories, or use chroot.
- Can't load a file from a world-writable directory.
- Can't load a file from a tainted filename starting with ~.
- Can't use File#chmod , File#chown , File#lstat , File.stat , File#truncate , File.umask , File#flock , IO#ioctl , IO#stat , Kernel#fork , Kernel#syscall , Kernel#trap . Process::setpgid , Process::setsid , Process::setpriority , or Process::egid= .
- Can't handle signals using trap.

$SAFE >= 3

- All objects are created tainted.
- Can't untaint objects.

$SAFE >= 4

- Can't modify a nontainted array, hash, or string.
- Can't modify a global variable.
- Can't access instance variables of nontainted objects.
- Can't change an environment variable.
- Can't close or reopen nontainted files.
- Can't freeze nontainted objects.
- Can't change visibility of methods (private/public/protected).
- Can't make an alias in a nontainted class or module.
- Can't get meta information (such as method or variable lists).
- Can't define, redefine, remove, or undef a method in a nontainted class or module.
- Can't modify Object.
- Can't remove instance variables or constants from nontainted objects.
- Can't manipulate threads, terminate a thread other than the current, or set abort_on_exception.
- Can't have thread local variables.
- Can't raise an exception in a thread with a lower $SAFE value.
- Can't move threads between ThreadGroups.
- Can't invoke exit, exit!, or abort.
- Can load only wrapped files, and can't include modules in nontainted classes and modules.
- Can't convert symbol identifiers to object references.
- Can't write to files or pipes.
- Can't use autoload.
- Can't use taint objects.

ruby_cop:

metodi non permessi: "__send__", "abort", "alias_method", "at_exit", "autoload", "binding", "callcc", "caller", "class_eval", "const_get", "const_set", "dup", "eval", "exec", "exit", "fail", "fork", "gets", "global_variables", "instance_eval", "load", "loop", "method", "module_eval", "open", "public_send", "readline", "readlines", "redo", "remove_const", "require", "retry", "send", "set_trace_func", "sleep", "spawn", "srand", "syscall", "system", "trap", "undef", "__callee__", "__method__"

costanti non permesse: "ARGF", "ARGV", "Array", "Base64", "Class", "Dir", "ENV", "Enumerable", "Error", "Exception", "Fiber", "File", "FileUtils", "GC", "Gem", "Hash", "IO", "IRB", "Kernel", "Module", "Net", "Object", "ObjectSpace", "OpenSSL", "OpenURI", "PLATFORM", "Proc", "Process", "RUBY_COPYRIGHT", "RUBY_DESCRIPTION", "RUBY_ENGINE", "RUBY_PATCHLEVEL", "RUBY_PLATFORM", "RUBY_RELEASE_DATE", "RUBY_VERSION", "Rails", "STDERR", "STDIN", "STDOUT", "String", "TOPLEVEL_BINDING", "Thread", "VERSION"