

|  |
| --- |
| Tutoriel d’utilisation |
| Création d’un modèle de message visuel  2017.11.27 |

Contents

[Préambule 4](#_Toc496184326)

[Accessibilité 5](#_Toc496184327)

[Utilisation 7](#_Toc496184328)

[Éléments statiques ou dynamiques ? 8](#_Toc496184329)

[Structure unique ou multiple ? 8](#_Toc496184330)

[1. Créer un texte dynamique 9](#_Toc496184331)

[2. Créer une date dynamique 13](#_Toc496184332)

[3. Créer une image dynamique 15](#_Toc496184333)

[ Dans une compagnie n’ayant pas le module d’hébergement 15](#_Toc496184334)

[ Dans une compagnie ayant le module d’hébergement activé 16](#_Toc496184335)

[4. Créer un style CSS dynamique 18](#_Toc496184336)

[5. Créer un sous-titre 22](#_Toc496184337)

[6. Créer un texte dynamique personnalisé 23](#_Toc496184338)

[7. Créer un élément multiple 26](#_Toc496184339)

[8. Créer un élément dynamique personnalisé (drop custom) 33](#_Toc496184340)

[9. Utiliser du contenu externe 36](#_Toc496184341)

[1. Utiliser une variable de type : données de tables relationnelles 37](#_Toc496184342)

[2. Utiliser des éléments multiples de données de tables relationnelles 43](#_Toc496184343)

[3. Utiliser les listes de définitions de tables relationnelles 52](#_Toc496184344)

[4. Utiliser des variables ou des blocs d’éléments d’OpenField Contenu 55](#_Toc496184345)

[a. Différence avec les TRs 55](#_Toc496184346)

[9. Utiliser des blocs avec conditions d’affichages 56](#_Toc496184347)

[10. L’Édition de zone WYSIWYIG directe 60](#_Toc496184348)

[11. L’édition Live du texte 63](#_Toc496184349)

[12. Bloc vidéo Youtube 63](#_Toc496184350)

[13. Les outils SU 64](#_Toc496184351)

[1. Les alias 64](#_Toc496184352)

[2. Import/export de variables, Blockset, etc. 65](#_Toc496184353)

**Historique des révisions**

|  |  |  |
| --- | --- | --- |
| Date | **Version** | **Commentaires** |
| 2014.10.01 | 1.0 | Version originale |
| 2014.10.16 | 1.1 | Ajout d’explications au point 6 |
| 2014.10.23 | 1.2 | Ajout d'explications au point 1, révisions mineures |
| 2015.09.17 | 2.0 | Relecture complète, mise à jour des textes et des captures d’écrans afin de refléter l’évolution de l’application. |
| 2015.12.01 | 2.1 | Ajout des points 8 à 11 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Préambule

Lorsque l’on souhaite envoyer un message attrayant et intéressant à ses contacts, il est important d’avoir de nombreuses compétences. Il faut tout d’abord avoir des talents de graphisme pour mettre en valeur le visuel, d’intégration afin de créer le corps du message, de rédaction afin d’écrire du contenu pertinent, et de gestion afin de procéder aux envois. Toutes ces tâches peuvent difficilement être attribuées à la même personne, et pour remédier à cela, il est possible de créer des gabarits d’éditions visuelles de messages.

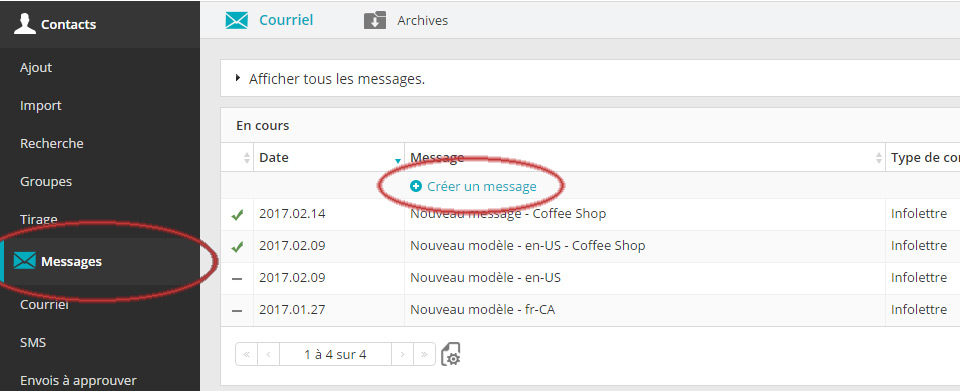
Les gabarits sont des outils de configurations qui permettent : de distinguer les zones éditables des zones statiques du message, de définir le visuel général, de fournir la provenance des données, etc. Le but de cette configuration est de pouvoir séparer les tâches plus simplement. Par exemple, les articles peuvent être écrits dans une base de contenus par une équipe de rédaction, les images peuvent être ajoutées dans une banque d’images par une autre équipe, pendant qu’un technicien définira les zones d’éditions du message, la provenance des données, et la disposition de ces articles selon ses règles visuelles. Ainsi, une équipe marketing pourra créer un message par simple glissé-déposé, qui respectera les standards visuelles, sans se soucier du contenu intrinsèque et sans connaissances du HTML.

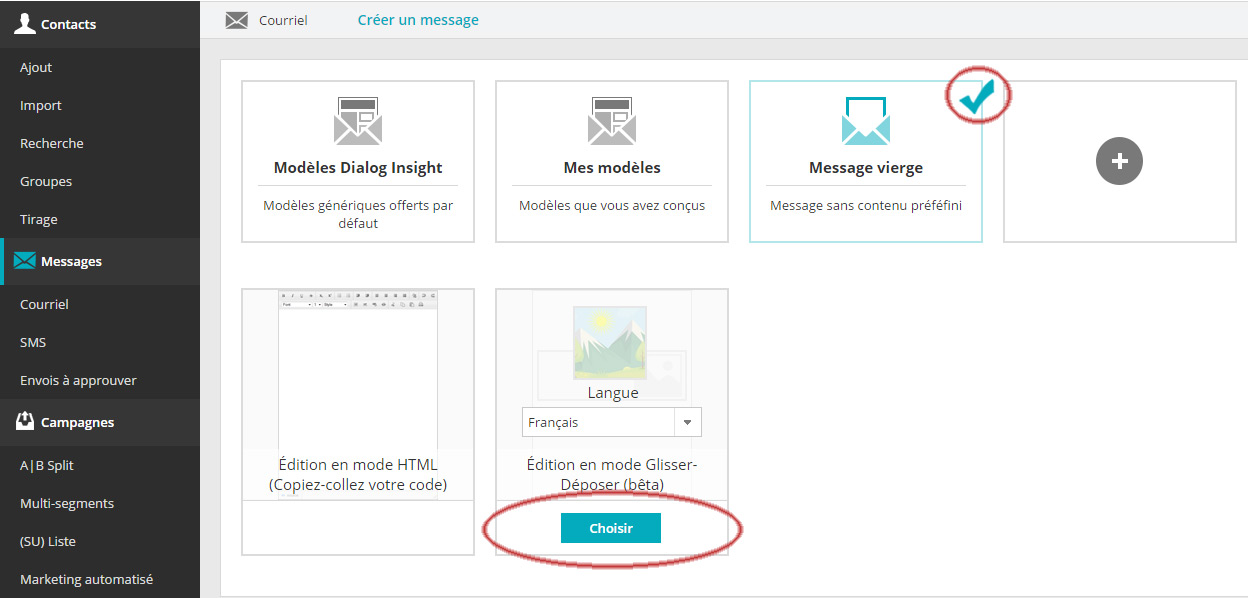
L’autre avantage des gabarits de messages est de pouvoir être réutilisé. Ainsi en dupliquant un message ou sa version, il sera possible de créer un nouveau message respectant de nouvelles règles visuelles, sans affecter les zones d’éditions ou le contenu du message.

Dans la suite de ce document, nous allons voir comment prendre en main cet outil en y décrivant ses principales fonctionnalités, et en voyant concrètement comment créer un simple message visuel.

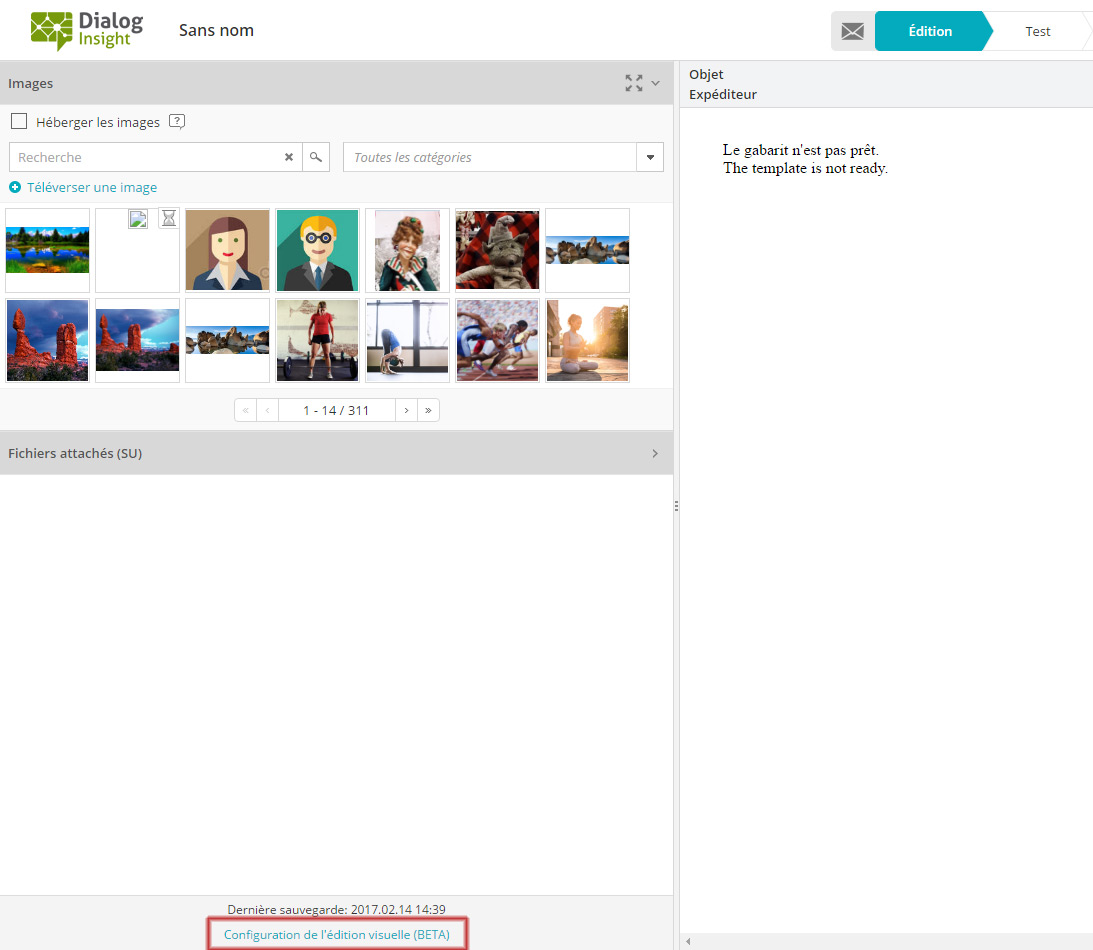
# Accessibilité

Une fois connecté sur Openfield, dans la section Openfield Contacts, aller dans la catégorie Messages > Courriel > Créer un message. Sélectionnez Message vierge et spécifier la langue dans laquelle le contenu sera généré « Édition en mode glisser-Déposer (béta) », puis cliquer sur ensuite Choisir. On décide ensuite le type de communication désiré.





Une fois dans l’interface de l’édition du message, un lien permet de basculer vers les interfaces de configurations de l’édition visuelle :



# Utilisation

Afin de prendre en main cet outil de configuration visuelle, nous allons créer un gabarit de message comprenant des éléments courant comme une bannière d’entête, un titre, un message de politesse, une date, un ou des article(s) avec photos, etc.

Voici un exemple de message fictif que nous allons créer :



## Éléments statiques ou dynamiques ?

La première chose à se demander lorsque l’on crée un gabarit de message, c’est la distinction entre les éléments statiques : les zones que l’on n’offre pas à l’édition et que la personne qui crée le message ne pourra pas modifier; et les éléments dynamiques : les zones qui changeront à chaque envoi.

Un message visuel est un document HTML dans lequel on ajoute des définitions d'édition visuelle. Avant d'aller plus loin, il est important de déterminer ce qui doit être éditable (dynamique) et ce qui n'a pas besoin de l'être (statique). Pour chaque partie éditable il faudra préparer des variables et leurs rendus. Cette étape d'analyse de vos besoins est cruciale, limitez-vous à ce que vous avez réellement besoin, trop en faire sera inutilement complexe à préparer, et deviendrait lourd à utiliser par la suite.

## Structure unique ou multiple ?

Les éléments statiques et dynamiques peuvent être constants, c’est-à-dire qu’à chaque message ils seront présent une seule et unique fois, ou bien ils peuvent être variables : ils seront présent zéro, une ou plusieurs fois.

Dans notre exemple d’infolettre, on suppose qu’à chaque envoi :

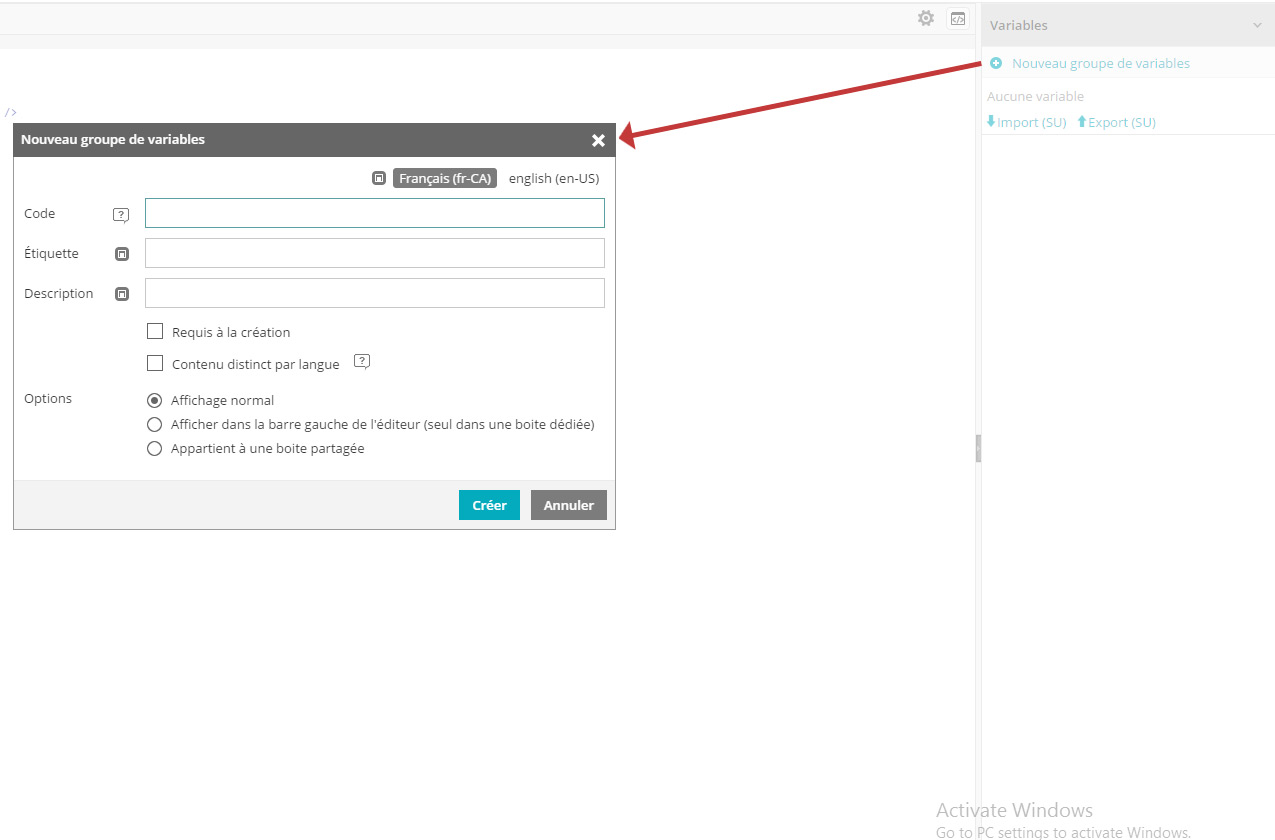
* Le message de salutation sera toujours présent, mais non-éditable : c’est donc un élément statique, malgré le fait qu'il contienne des composantes variables tel que le nom du contact. Ici, le terme "statique" fait référence au fait que l'usager n'aura pas à saisir de texte pour cette zone lorsque viendra le temps de créer un nouveau message à partir de ce modèle.
* La bannière d’entête, le titre et la date du bulletin seront toujours présent mais peuvent différer d’un message à un autre : ce sont des éléments dynamiques.
* Des articles seront potentiellement présent, en quantité variable et ils pourront être modifiables : c’est une structure multiple dont le contenu sera statique ou dynamique

Parmi les éléments dynamiques de votre message certain seront unique (entête, pied de page), d'autre pourront être présent plusieurs fois (articles, nouvelles, promotions). C'est une information importante à savoir car elle affecte la préparation des variables.

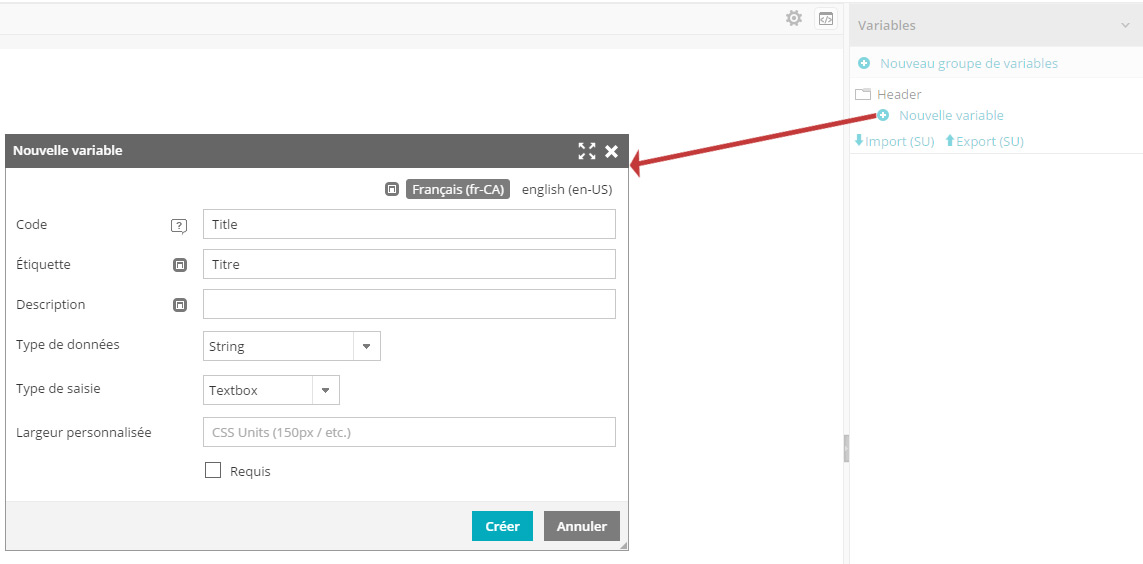
Voyons maintenant comment créer des zones d’éditions de champs.

## Créer un texte dynamique

Le premier élément que l’on a défini comme dynamique est le titre du bulletin. Il nous faut donc créer un groupe de variables pour l’entête qui contiendra des variables comme le titre ou la date. Pour cela, ajouter dans les variables du message un nouveau groupe de variables, avec un code qui servira dans le HTML du message et une étiquette qui servira à présenter les champs à saisir au client.



Une fois le groupe de variables d’entête créé, il ne reste plus qu’à ajouter ses variables. Ici, on veut juste un champ de type chaîne de caractères, dont la saisie s’effectuera dans un simple champ texte. De même, le code servira dans le HTML du message et l’étiquette servira à indiquer au client ce qu’il doit saisir.



Notez que lorsque des variables sont créées, leurs définitions sont sauvegardées en base de données. Leurs références sont ensuite utilisées à divers endroits afin de créer leurs contenus associés. Il n’est pour l’instant pas possible d’altérer le code ou le type des variables après leur création. Ainsi lorsqu’une variable est créée, son code et son type ne doivent jamais être modifiés. En cas d’erreur, il est préférable de créer une nouvelle variable.

On veut maintenant utiliser ce champ « Title » dans le message. Pour cela il suffit simplement de l’ajouter en code de template de cette façon : Version.MaVariable.MonChamp, soit :

<table align="center" width="600">

<tr>

<td>

[[=Version.Header.Title;]]

</td>

</tr>

</table>

Note : Le caractère « = » est un raccourci de la méthode output.write().

Pour permettre à l’utilisateur de saisir ce titre, il faut spécifier que cette zone de texte est éditable :

<table align="center" width="600" [[VisualEditor.EditZone(Version.header);]]>

<tr>

<td>

[[=Version.Header.Title;]]

</td>

</tr>

</table>

Cette balise est désormais cliquable, mais étant présentement vide, elle ne sera pas visible à l’utilisateur. Il faut donc vérifier si cette variable a une valeur, et le cas échéant, demander à l’utilisateur d’effectuer une action. Cette vérification se fait toujours en 2 étapes : la variable doit contenir le champ que l’on vérifie**(1)** et ce champ ne doit pas être nul**(2)**. Dans le cas d’une chaîne de caractère, il faut aussi vérifier que la valeur est plus grande que zéro caractère**(3)**. On peut aussi afficher un texte alternatif incitant l’utilisateur à saisir la valeur du champ. Ce texte doit être affiché uniquement en mode édition de message afin qu’il ne soit pas utilisé dans l’envoi du message final**(4)**.

Ce qui donne :

**3**

string TitleHeader = "";

if (Version.ContainsField("Header") && Version.Header != null)

{

if (Version.Header.ContainsField("Title")   
&& Version.Header.Title != null   
&& Version.Header.Title.Length > 0)

{

TitleHeader = Version.Header.Title;

}

}

]]

<html>

<head>

</head>

<body>

<table align="center" width="600" [[VisualEditor.EditZone(Version.Header);]]>

<tr>

<td>

[[if(TitleHeader != ""){]]

[[=TitleHeader;]]

[[}else if(VisualEditor.IsDesignMode()){]]

Svp entrer le titre de votre infolettre

[[}]]

</td>

</tr>

</table>

</body>

</html>

[[

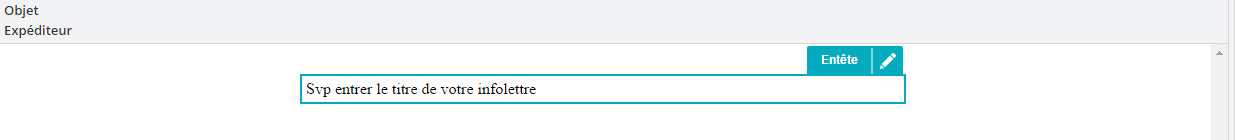
**1**

**2**

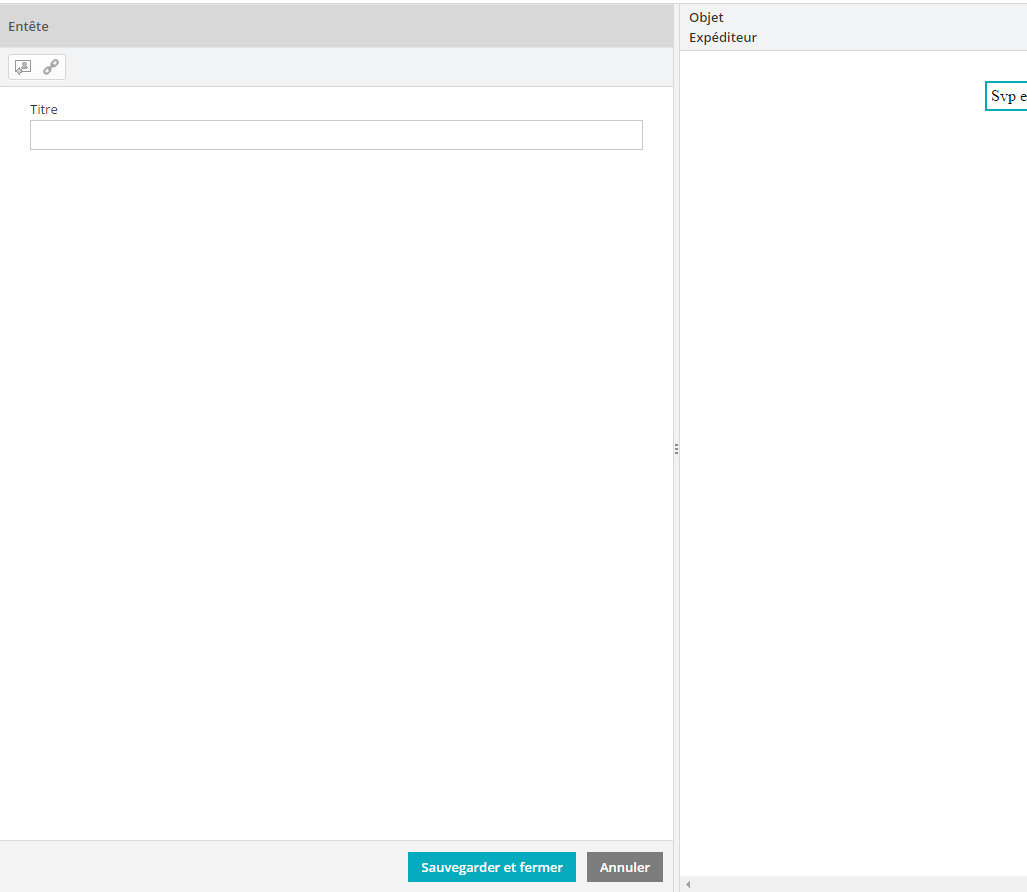
**4**

Ainsi, dans l’interface de création d’un message, un double-clic sur la balise <div> ouvrira une fenêtre permettant l’édition des champs de la variable « Header ».

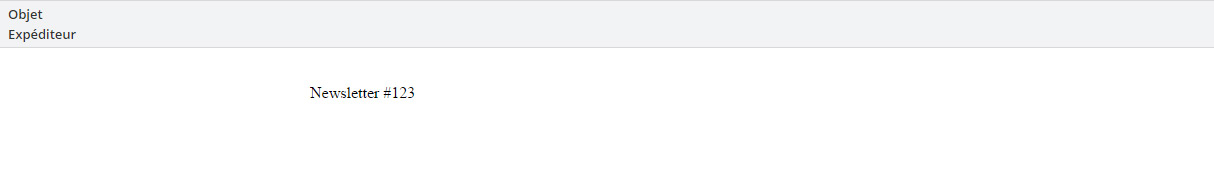
Voici un exemple d’utilisation de ce champ dans l’interface d’édition de message :



Étiquette de la variable

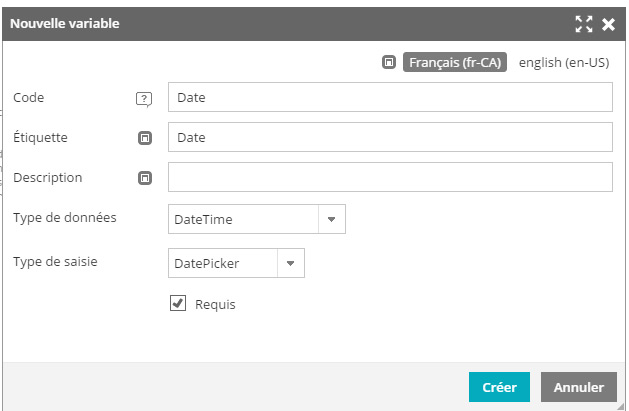


Étiquette du champ



## Créer une date dynamique

Pour ajouter un élément dynamique de type date, la procédure est presque la même que pour ajouter un champ texte. Ajouter un champ de type DateTime au groupe de variables   
« Header », et choisir un mode de saisie. Ici le plus approprié sera évidemment le Sélecteur de date :



Pour afficher correctement la date, il suffira simplement de placer l’affichage au bon endroit, ici dans une colonne à droite du titre, et de formater l’affichage de la date selon un format définit comme « yyyy.MM.dd » ou « MMMM d, yyyy » par exemple.

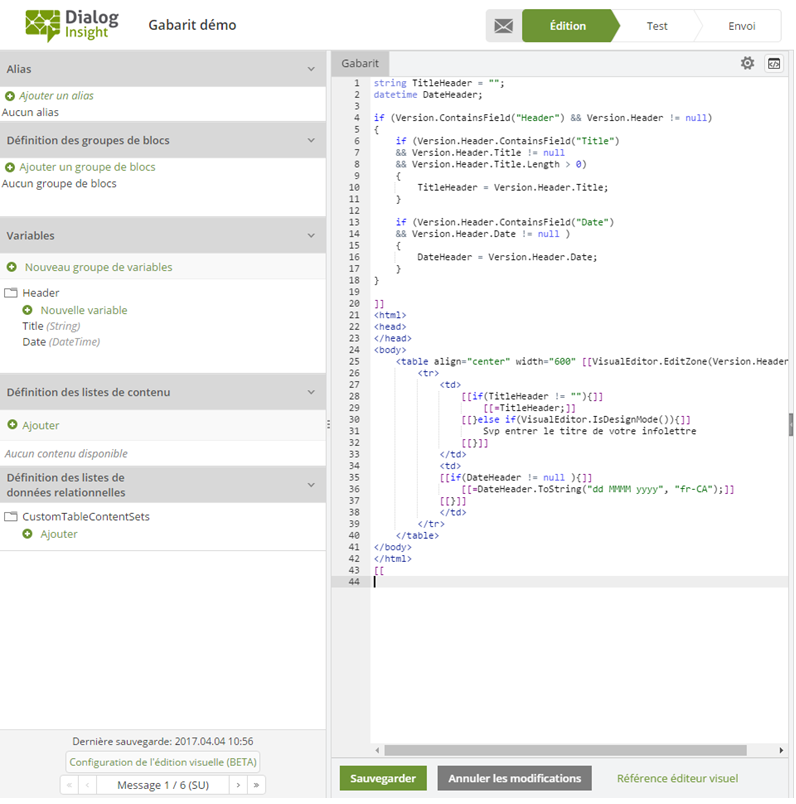
Soit : **[[=DateHeader.ToString("dd MMMM yyyy", "fr-CA");]]**

Note : Lors de la prévisualisation du message, la date s’affichera dans la langue du gabarit sélectionné pour le message, et non dans la langue d'interface-usager.

Utilisation :



Le code complet de ce début de gabarit :



string TitleHeader = "";

datetime DateHeader;

if (Version.ContainsField("Header") && Version.Header != null)

{

if (Version.Header.ContainsField("Title")

&& Version.Header.Title != null

&& Version.Header.Title.Length > 0)

{

TitleHeader = Version.Header.Title;

}

if (Version.Header.ContainsField("Date")

&& Version.Header.Date != null )

{

DateHeader = Version.Header.Date;

}

}

]]

<html>

<head>

</head>

<body>

<table align="center" width="600" [[VisualEditor.EditZone(Version.Header);]]>

<tr>

<td>

[[if(TitleHeader != ""){]]

[[=TitleHeader;]]

[[}else if(VisualEditor.IsDesignMode()){]]

Svp entrer le titre de votre infolettre

[[}]]

</td>

<td>

[[if(DateHeader != null ){]]

[[=DateHeader.ToString("dd MMMM yyyy", "fr-CA");]]

[[}]]

</td>

</tr>

</table>

</body>

</html>

[[

## Créer une image dynamique

Pour ajouter une bannière dynamique à notre message, il faut comme pour le titre ajouter un champ à notre groupe de variables « Header ». Le choix du type de saisie de ce champ et son utilisation va dépendre de la manière dont le client utilise les services de Dialog Insight :

### Dans une compagnie n’ayant pas le module d’hébergement

Le client héberge les images chez lui et insère le lien de ses images, hébergées sur ses propres serveurs. Dans ce cas, un simple champ texte est proposé à l’utilisateur ou il y insérera le lien de son image. Ce lien sert à remplir la source d’une balise <img>, stockée dans un champ de type chaîne de caractères et utilisée dans le gabarit comme ceci :

string BannerHeader= "";

if (Version.ContainsField("Header") && Version.Header != null){

if (Version.Header.ContainsField("Banner") && Version.Header. Banner != null && Version.Header. Banner.Length > 0){

BannerHeader = Version.Header. Banner;

}

}

]]

<html>

<head>

</head>

<body>

<table align="center" width="600" [[VisualEditor.EditZone(Version.Header);]]>

<tr>

<td colspan="2">

[[if(TitleHeader != ""){]]

<img src="[[=BannerHeader;]]" alt="" width="600" style="height:auto;display:block;" border="0" />

[[}else if(VisualEditor.IsDesignMode()){]]

<img src="https://placeholder-repsolution.netdna-ssl.com/S(600x100)/T(Bannière 600px)/image.png" alt="" width="600" style="display:block;height:auto;" border="0" />

[[}]]

</td>

</tr>

…

### Dans une compagnie ayant le module d’hébergement activé

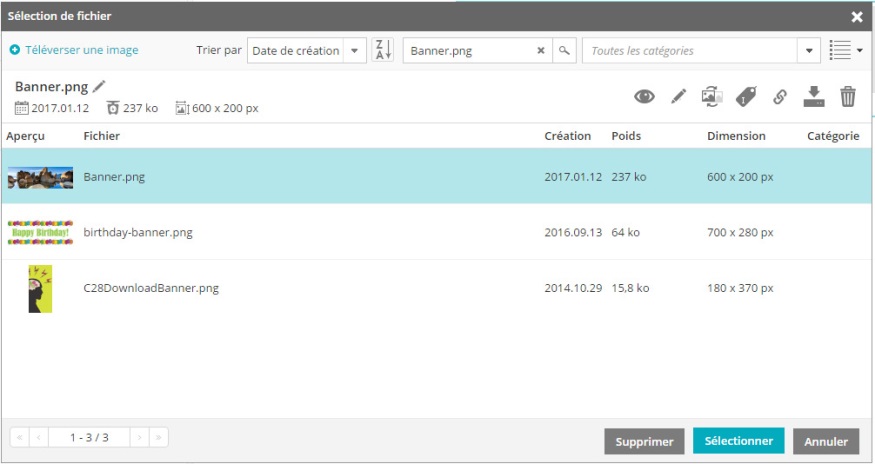
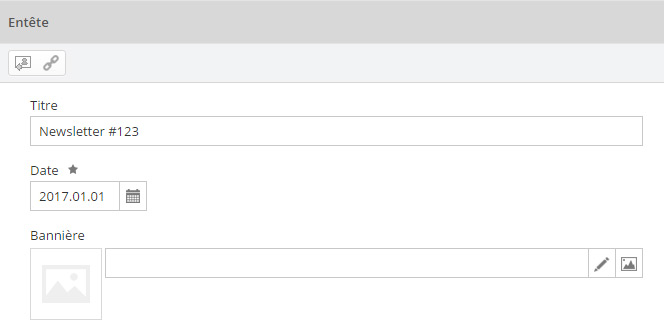
Dans ce cas, le client héberge ses images dans l’application et les utilise dans son message. Ces images sont gérées au niveau de la compagnie, et peuvent donc être utilisées dans tous les projets. Pour lui faciliter cette tâche, le type de saisie de ce champ doit être un sélecteur d’image. Ce dernier permet lors de la saisie, de choisir dans une banque d’images le fichier à utiliser dans le message. Dans ce cas, la valeur retournée par ce sélecteur est soit un entier (qui sera l’identifiant de l’image choisit), soit une chaîne de caractère (le chemin complet de l’image choisie).

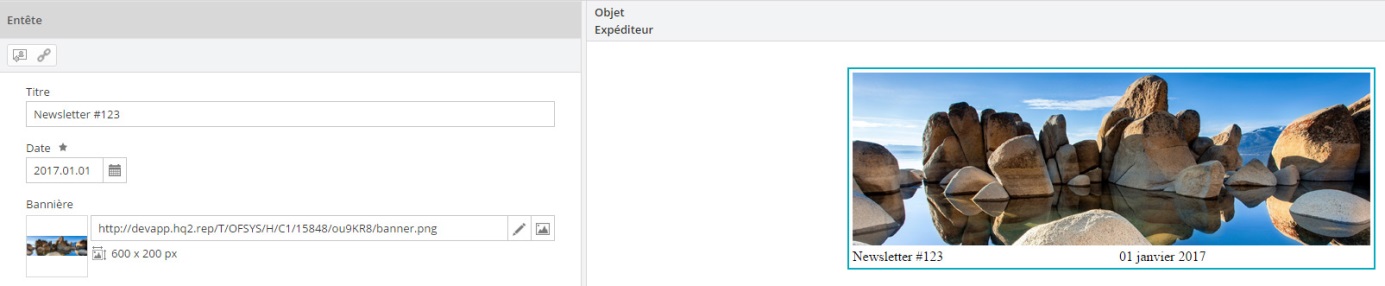
Tant que les images sont hébergées dans Dialog Insight, ces deux types de données amènent au même résultat. Mais l’utilisation d’un entier est préconisée afin de facilité les duplications de messages.



Voici comment le sélecteur d’image sera utilisé par l’utilisateur :



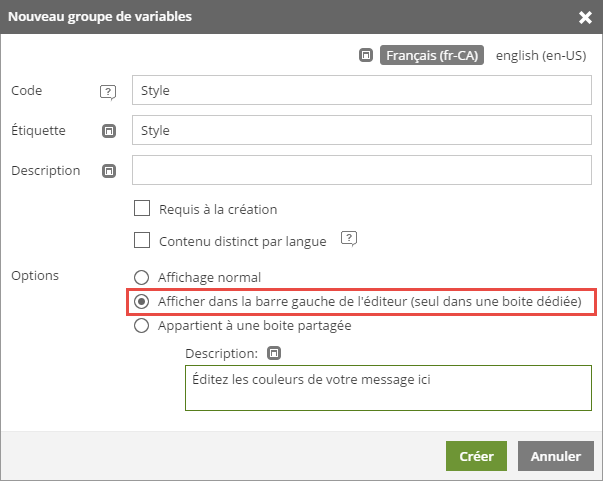




## Créer un style CSS dynamique

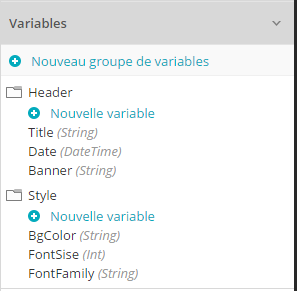
Pour styliser le message, nous pouvons créer une nouvelle variable « Style » qui contiendra tous les champs que l’on veut rendre éditable. Plutôt que d'ajouter des champs à une variable existante, le choix d'une nouvelle variable permet de contrôler la façon dont ces valeurs seront saisies indépendamment des autres données.

Nous avons vu lors de la création de l’entête qu’un clic sur la zone à éditer ouvre une fenêtre d’édition des variables du groupe « Header ». Or ici, les styles CSS peuvent s’appliquer à tout le message et il n’existera pas de zone cliquable pour permettre leurs éditions. Dans ce genre de cas, il faut préciser lors de la création d’une variable que l’on souhaite la voir afficher parmi les autres outils de l’éditeur de message:



On ajoute ensuite tous les champs que l’on rend éditables. Ici, le choix du type de champ est simple : les attributs CSS sont généralement des nombres entiers ou des chaines de caractères. Leurs saisies peuvent donc se faire via un champ texte, ou bien, dans le cas d’une couleur, via un « Sélecteur de couleurs ».

Voici un exemple d’une structure de données de styles :



Voyons maintenant comment utiliser ces champs dans le code du gabarit de message.

Pour cela, il faut ajouter un attribut « style » à toutes les sections que l’on veut styliser. Or, comme nous l’avons vu, nous devons vérifier l’existence de chaque variable avant de l’utiliser, ce qui alourdit énormément le code si nous devions ajouter ces contenus conditionnels à l’intérieur des balises. De plus, à chaque utilisation de ces champs, il faudrait refaire ces vérifications…

Pour remédier à cela, il est possible de définir des variables dans le code du gabarit. Ainsi, nous pouvons déclarer une variable dans le gabarit, qui contiendra une valeur par défaut ou la valeur saisie par l’utilisateur :

string BgColor= "#ffffff";

if (Version.ContainsField("Style") && Version Style != null){

if (Version. Style.ContainsField("BgColor") && Version. Style. BgColor != null && Version. Style. BgColor.Length > 0){

BgColor = Version. Style. BgColor;

}

}

Il ne reste plus qu’à construire le style dans une balise avec cette variable :

<table align="center" width="600" bgcolor="**[[**=BgColor;**]]**">

Voici dans notre cas comment utiliser dans le code du gabarit nos 3 champs saisies par l’utilisateur dans un champ texte ou à l’aide d’un sélecteur de couleurs :

**[[**

string BgColor= "#ffffff";

string FontSise= 11;

string FontFamily= " Arial, Helvetica, sans-serif";

if (Version.ContainsField("Style") && Version.Style != null){

if (Version.Style.ContainsField("BgColor") && Version.Style.BgColor != null && Version. Style. BgColor.Length > 0){

BgColor = Version.Style.BgColor;

}

if (Version.Style.ContainsField("FontSise") && Version.Style.FontSise != null && Version. Style. FontSise.Length > 0){

FontSise = Version.Style.FontSise;

}

if (Version.Style.ContainsField("FontFamily") && Version.Style.FontFamily != null && Version.Style.FontFamily.Length > 0){

FontFamily = Version.Style.FontFamily;

}

}

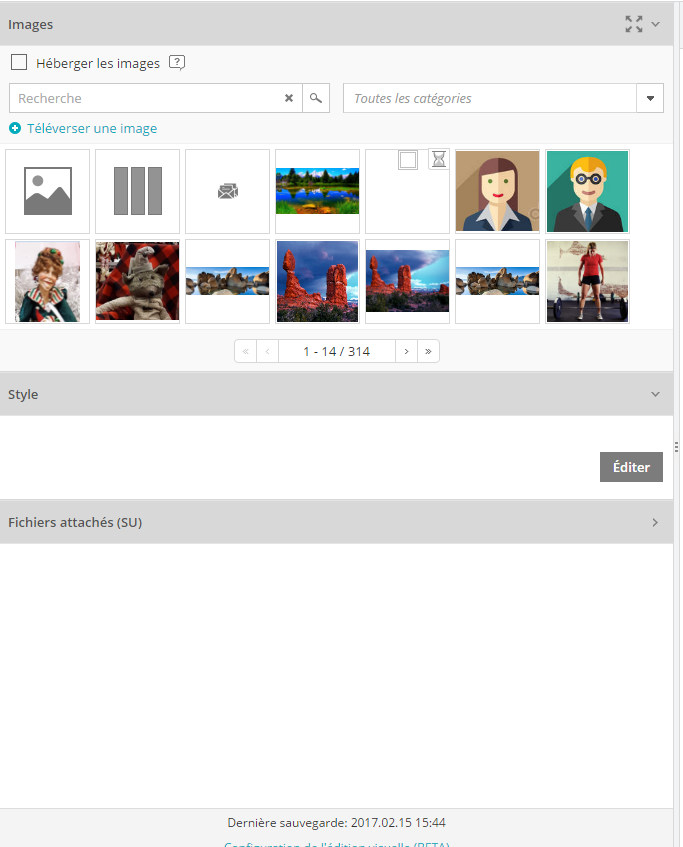
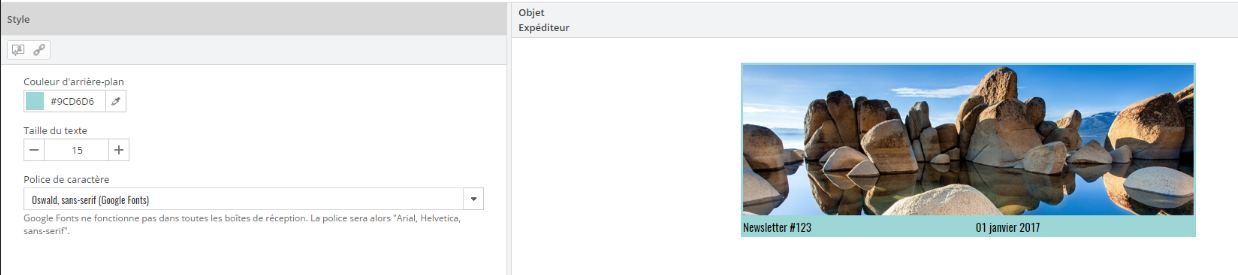
**]]**

<table align="center" width="600" [[VisualEditor.EditZone(Version.Header);]]>

<tr>

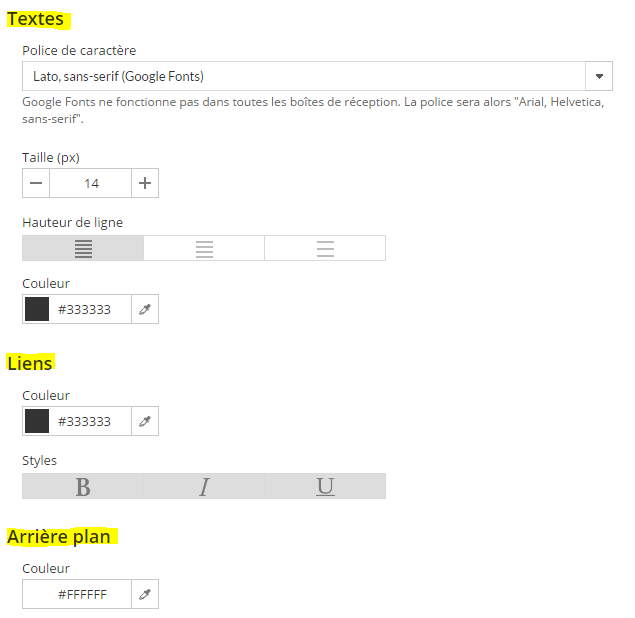
<td style="font-family:[[=FontFamily;]];font-size:[[=FontSise;]];">

Et voici visuellement ce que verra l’utilisateur :

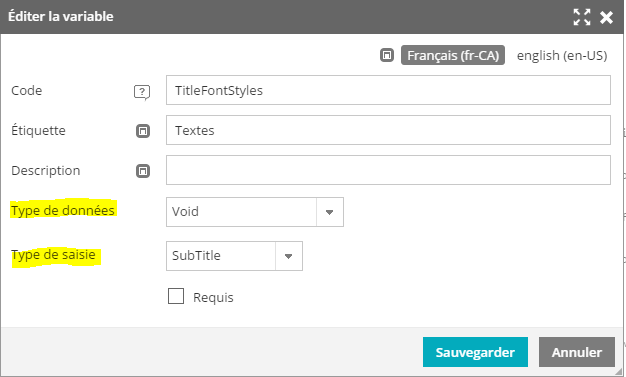
 

## Créer un sous-titre

Il est possible de créer des variables sous-titre pour hiérarchiser ou structurer la prise de contenu lorsque celle-ci devient plus complexe. Voici un exemple complexe de prise de données :



Les textes en noir et de taille plus grande sont des sous-titres. Ces variables se créent de la même façon que les autres.



## **Créer un texte dynamique personnalisé**

Lorsque le rendu du message s’effectue, le code qui se trouve entre double crochets est exécuté. Donc [[=Message.header.title;]], qui est simplement un raccourci à [[output.write(Message.header.title);]], va afficher la valeur contenue dans le champ « title ».

Ainsi, pour afficher le prénom du contact lorsque l’on compose un message, on utilise tout simplement [[=Contact.f\_FirstName;]]. Au moment de créer le rendu du message, le code entre crochets se fera exécuter, et la valeur affichée sera ce que contient Contact.f\_FirstName, soit le prénom du contact.

Dans un gabarit d’un message visuel, il y a une pré-exécution. Le gabarit doit être exécuté une première fois pour générer la source du message éditable, cette source sera exécutée une deuxième fois lors de la prévisualisation ou de l’envoi du message. Ainsi, dans le gabarit il n’est pas possible d’écrire directement [[=Contact.f\_FirstName;]], car le contact n’est disponible qu’à la seconde exécution.

Pour remédier à cela, il suffit d’afficher le code qui affichera le prénom du contact, soit : [[="[[=Contact.f\_FirstName;]]";]]

Ainsi, tout ce qui touche à Contact doit être fait dans une double exécution.

Donc l’affichage du prénom, en vérifiant s’il existe, dans un message comme ceci :

[[if (Contact.f\_FirstName != null && Contact.f\_FirstName.Length > 0){]]  
 Hello [[output.write(Contact.f\_FirstName);]],  
[[} else {]]  
    Hello,   
[[}]]

Doit donc être fait comme ceci :

[[output.write("  
  [[if (Contact.f\_FirstName != null && Contact.f\_FirstName.Length > 0) {]]  
    Hello [[output.write(Contact.f\_FirstName);]],  
   [[} else {]]  
       Hello,   
   [[}]]");]]

Cela affichera dans l’éditeur de message ceci :



Ce qui affichera finalement dans le message envoyé :

Cette double exécution doit donc être faite pour tous les éléments de personnalisation. Cela vaut pour les informations des contacts, mais aussi les liens de désabonnements, les marqueurs d’ouverture de message(les readTags), les requêtes, etc.

Si ce code est gênant pour celui qui édite le message, il est possible de masquer la condition en utilisant la balise de commentaire html (<!-- -->). Attention de ne pas masquer le contenu !

Voici l’affichage dans l’éditeur de message de la personnalisation en commentant le code conditionnel :

<tr>

<td colspan="2" style="padding:10px;font-family:[[=FontFamily;]];font-size:[[=FontSise;]];">  
    [[output.write("  
   
     <!--[[if (Contact.f\_FirstName != null && Contact.f\_FirstName.Length > 0)  {]]-->  
           Hello [[output.write(Contact.f\_FirstName);]],  
     <!--[[} else {]]-->  
           Hello,   
     <!--[[}]]-->")  
   
     ;]]  
 </td>

</tr>



## Créer un élément multiple

Nous avons vu comment ajouter des éléments individuels à notre message, mais dans bien des cas il sera souhaitable de réutiliser des fragments de code. Pour cela, nous allons voir comment créer une série de blocs de contenu, ainsi qu'un contenant – ou "groupe de blocs" (blockset en anglais) – pour les regrouper.

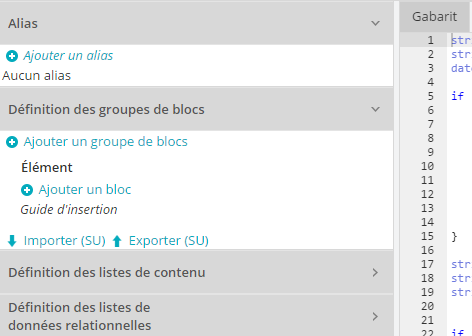
Commençons par ajouter un nouveau groupe de blocs. Comme toujours, le code servira à utiliser le groupe de blocs dans le gabarit, et l’étiquette sera affichée à l’utilisateur. Le type de contenu sert ici à déterminer dans quel contexte sera utilisé ce groupe de bloc. Cela pourrait être dans une cellule ou une ligne d’un tableau, mais dans la plupart des cas, il s’agit d’un bloc d’éléments du type div, table, etc. On rend aussi disponible ce groupe dans les éléments autorisés à être glissés/déposés dans le message.

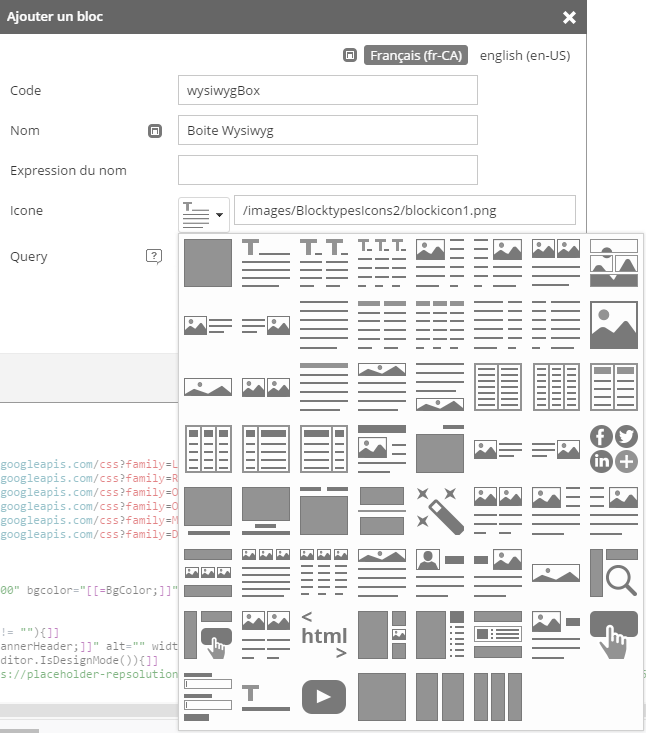




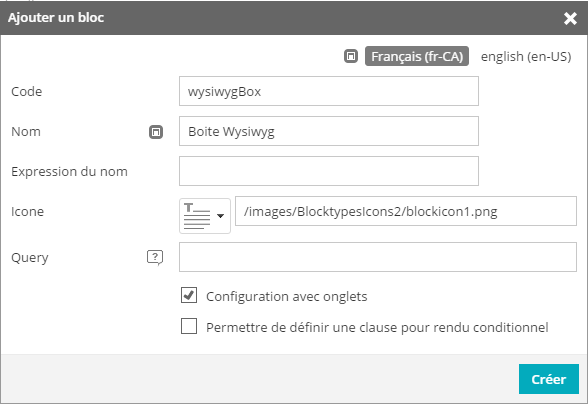
Nous avons maintenant un contenant pour nos blocs de contenu, il nous faut donc créer ce contenu. Dans notre exemple d’infolettre, nous voulions un texte d’introduction avec son titre. Nous allons donc créer un bloc qui contiendra ces éléments.

On créer pour cela un nouveau bloc avec son code et son étiquette, et on lui choisit un icône existant, ou on lui fournit l’URL d’une image, qui le représentera dans la librairie. Dans notre cas, nous choisiront une simple zone d’édition de texte WYSIWYG :





Une option permet également de repartir les groupes de variables par onglets lors de la saisie de l’utilisateur. Cela est surtout pratique lorsqu’il y a plusieurs valeurs à saisir, on peut alors les répartir en plusieurs groupes de variables par sujet (contenu d’un article d’un côté et style CSS de l’autre par exemple).



Dans la zone d’édition du bloc, nous allons modifier le contenu du <table> déjà présent comme exemple de code, afin d’afficher un titre et le contenu du texte. Comme toujours, on vérifie la présence de chaque variable avant de les utiliser :

]]

<table width="100%" align="center" cellpadding="0" cellspacing="0" border="0" [[ VisualEditor.EditZone(block\_Element);]]>

<tr>

<td align="center">

[[if(block\_Content.ContainsField("Content") && block\_Content.Element != null && block\_Content.Element.ContainsField("HtmlCode") && block\_Content.Element.HtmlCode != null && block\_Content.Element.HtmlCode.Length > 0){]]

[[=block\_Content.Element.HtmlCode;]]

[[}else{]]

<div style="color:#aaaaaa; font-style:italic; text-align:center;">

[[=VisualEditor.UICultureTranslate("Insérer du contenu HTML", "Insert HTML content");]]

</div>

[[}]]

</td>

</tr>

</table>

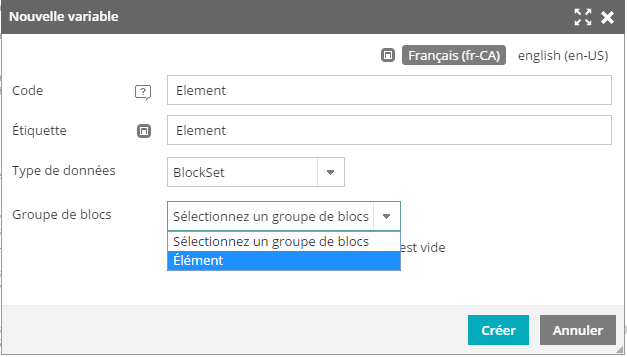
[[

Par praticité pour l’utilisateur, on rend le contenu du bloc plus grand ou égal à la taille du guide d’insertion (le placeholder de drag and drop).

Une fois le groupe de blocs créé, il faut l’ajouter dans les variables du message. On commence donc par se créer un groupe de variables « content ».



Puis, on créer dans ce groupe une variable nommée « element » de type « BlockSet » afin de pouvoir l'utiliser pour stocker l'ensemble de la liste des blocs que nous définirons pour notre bulletin.



Il ne reste plus qu’à effectuer le rendu du groupe de blocs dans le code du gabarit :

<table align="center" width="600">

<tr>

<td>

[[if(Version.ContainsField("Content") && Version.Content.ContainsField("Element") && Version.Content.Element != null){ Version.Content.Element.Render(); }]]

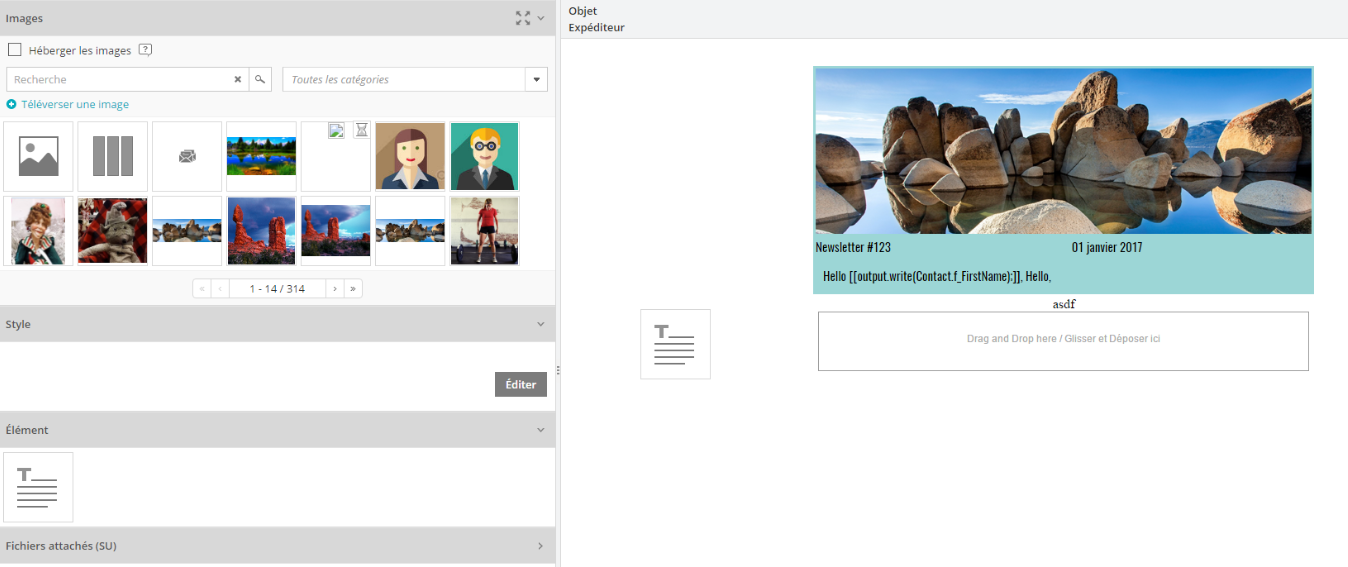
</td>

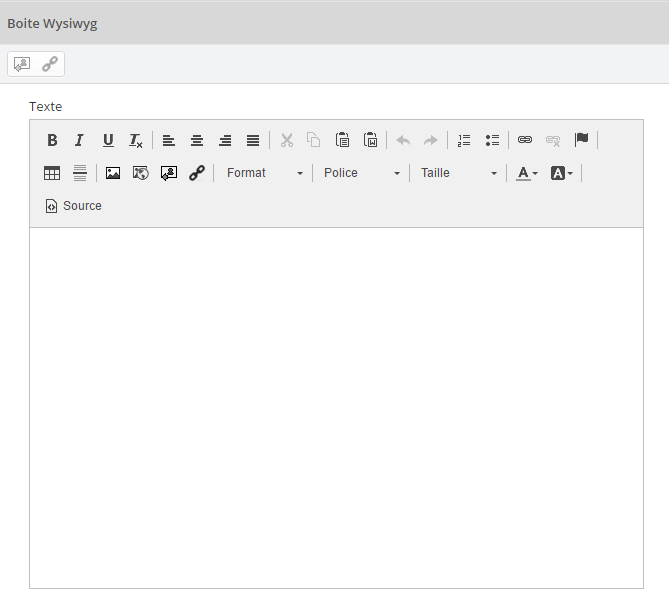
</tr>

</table>

Notez qu’ici, il n’y a pas besoin d’afficher ce bloc avec un [[=Message… ou un [[output.write(Message… car la methode Render() n’a pas de retour, il s’agit d’une méthode de type void.

Ainsi, l’utilisateur verra dans son message une zone de glisser/déposer lui indiquant qu’il peut y déposer le bloc que nous avons créé. Ce bloc est visible dans l’onglet « librairie ». Après avoir déposé ou ajouté un nouveau bloc, l’utilisateur sera incité à y mettre du contenu :



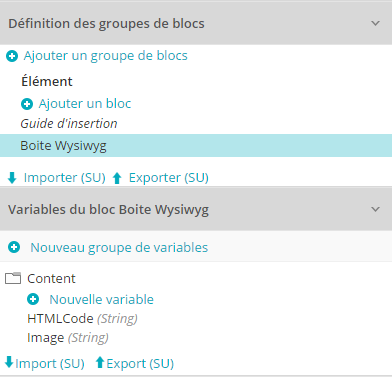


L’utilisateur pourra ainsi créer autant de blocs de ce contenu qu’il le souhaite, les réordonner par glisser/déposer, les modifier, etc. Le style et la disposition des éléments resteront tels que définit par le gabarit :



## Créer un élément dynamique personnalisé (drop custom)

Pour aller plus loin dans l’édition du message, on peut permettre la création automatique d’un bloc sur le glisser/déposer d’une image par exemple. Pour cela, on ajoute à notre groupe de variables « Content » du bloc «Wysiwyg Box » un champ de type «image» éditable à l'aide d'un contrôle "FilePickerImage" :



Puis, on utilise ce champ comme vu précédemment dans le code du bloc :

<tr>

<td>

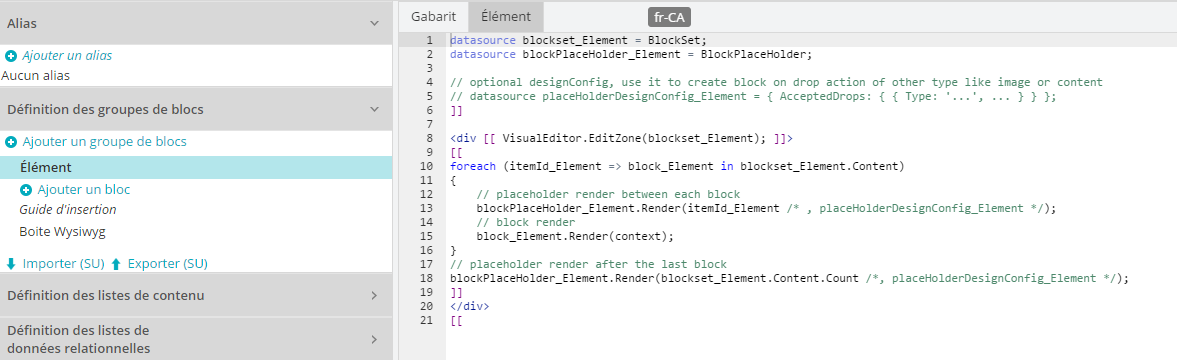
[[if(block\_Content.ContainsField("Content") && block\_Content.Element != null && block\_Content.Element.ContainsField("Image") && block\_Content.Element.Image != null && block\_Content.Element.Image.Length > 0){]]

<img src="[[=block\_Content.Element.Image;]]" width="600" alt="" style="display:block;height:auto;" border="0" />

[[}]]

</td>

</tr>

 A présent, il ne nous reste plus qu’à permettre la création de notre bloc lors de l’ajout d’une image dans notre guide d’insertion. Pour cela, aller dans l’édition du groupe de blocs «Element». Un exemple de code présente comment utiliser les dépôts personnalisés.

On va le modifier pour obtenir ceci :

**5**

**4**

**3**

**2**

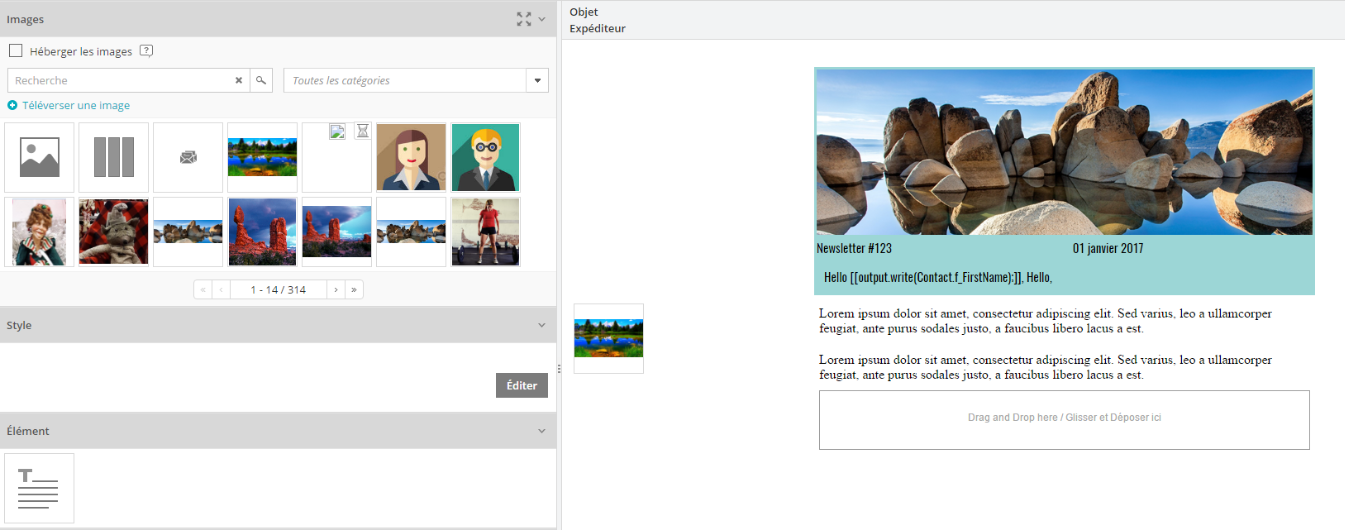
**1**

datasource placeHolderDesignConfig\_element =   
{  
    AcceptedDrops:  
    {  
        {  
            Type: "image",  
            CreateBlock:  
            {  
                BlockSet: blockset\_Element,  
                BlockType: "wysiwygBox",  
                Assignations:  
                {  
                    {  
                        ProviderName: "Content",  
                        ProviderField: "Image"  
                    }  
                }  
            }  
        }  
    }  
}

Explications :

Lorsqu’un élément est déposé dans la zone du groupe de blocs « Element », si c’est un élément de type image**(1)**, alors on créer un nouveau bloc. Ce nouveau bloc fait partie de la variable blockset\_element**(2)** (le groupe «Element») et est un bloc du type wysiwygBox**(3)**. Une fois créée, l’image qui vient d’être déposée remplira le champ «image» **(5)** contenu dans la variable «content» **(4)**. Les autres champs n’étant pas assignés et étant requis, une fenêtre d’édition sera présentée à l’utilisateur afin qu’il saisisse le titre et le texte manquant.

Ainsi, lors d’un glisser/déposer d’une image dans le groupe de bloc Element, un nouveau bloc sera automatiquement créer avec l’URL de l’image pré-remplit :

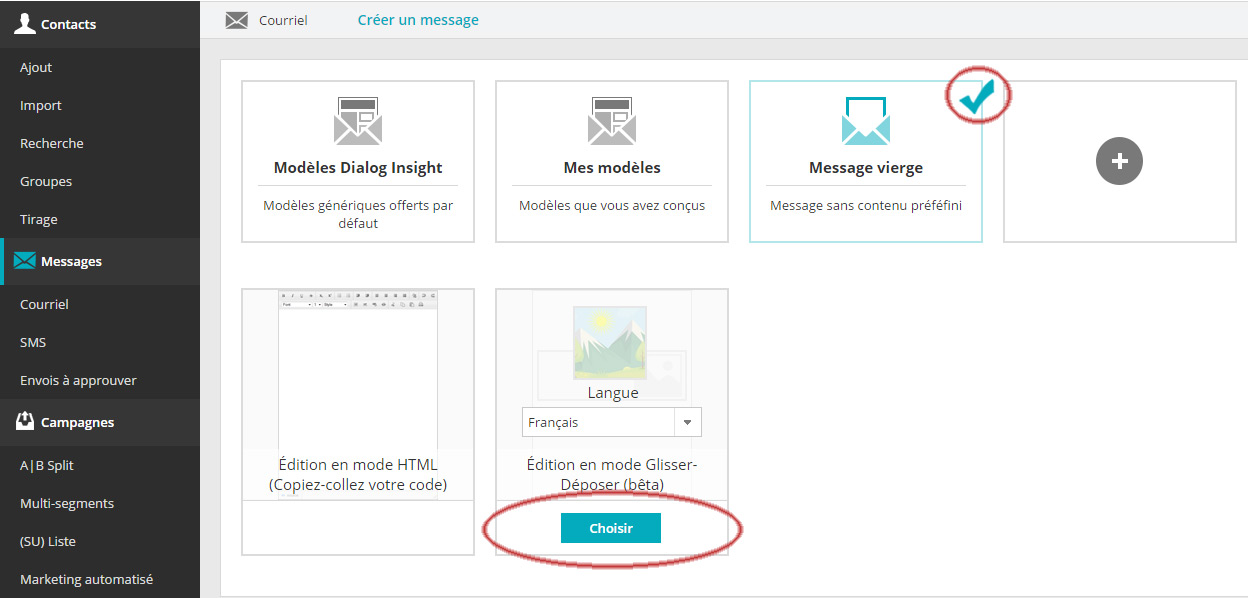


# Utiliser du contenu externe

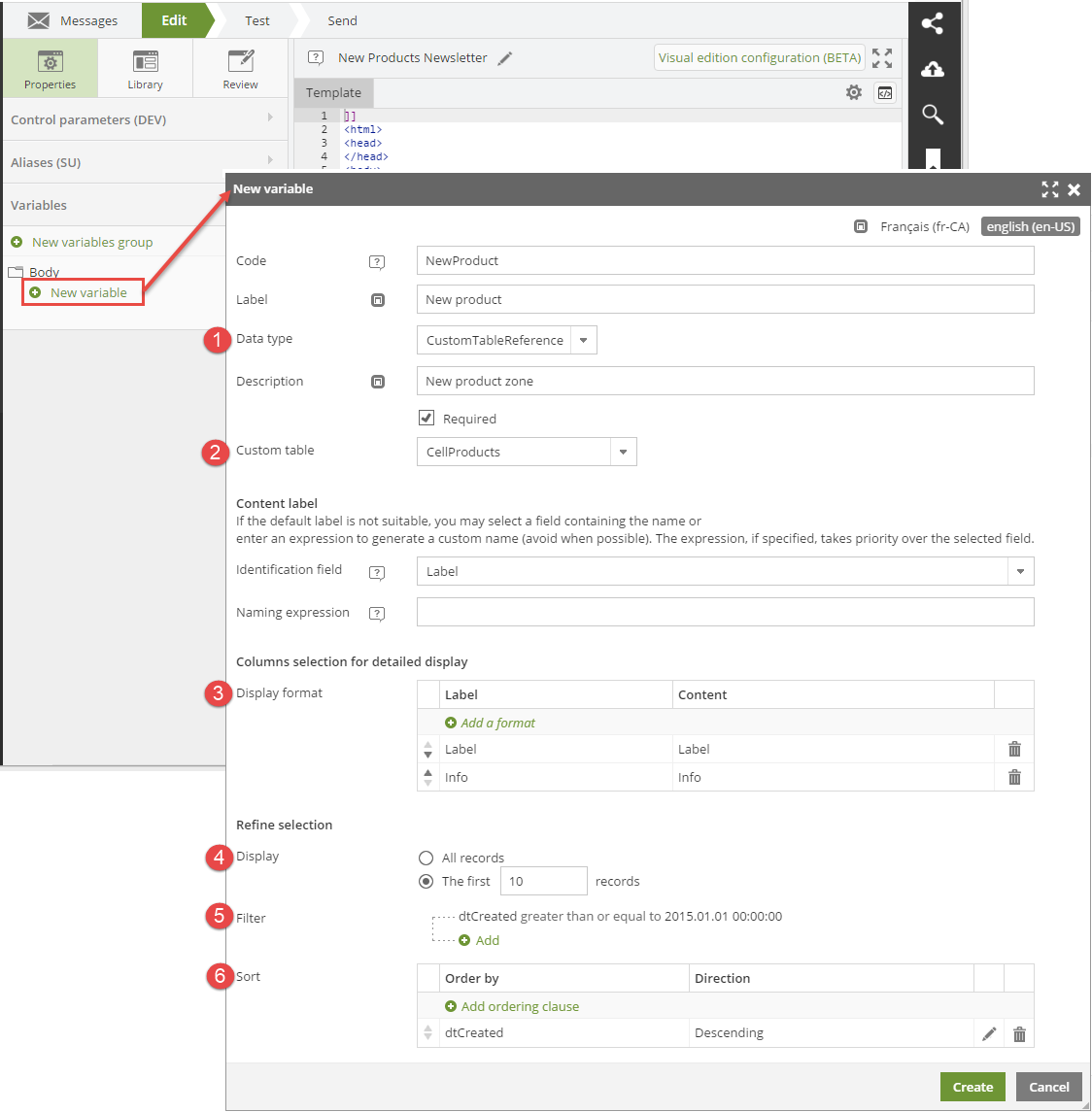
Il est possible de rendre la création d’un courriel complexe encore plus simple pour l’utilisateur, en lui proposant d’utiliser du contenu externe dans son message. Par exemple, une compagnie voulant envoyer un top 3 de ses produits, ou ses derniers articles, pourrait vouloir figer la présentation de ses produits et ne laisser à la personne qui prépare les courriels que la possibilité de choisir les items….

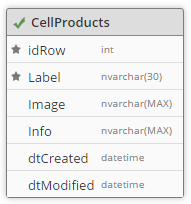
Pour illustrer ces fonctionnalités, nous allons prendre un exemple relativement courant : une infolettre de produits. Le but ici est de permettre à celui qui créer les courriels de pouvoir choisir les produits d’une liste, sans se soucier du visuel.

Pour cela, on créera un nouveau message par glissé-déposé :



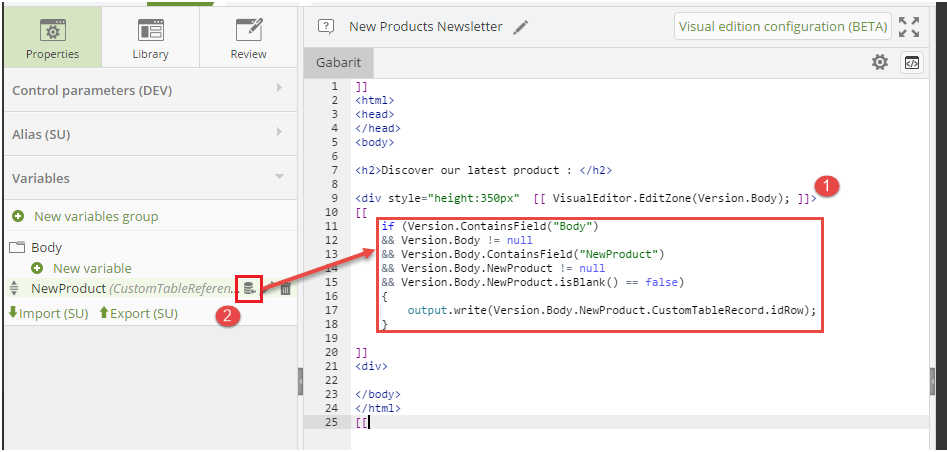
### Utiliser une variable de type : données de tables relationnelles

On souhaite rendre la création d’une présentation de produit soit aussi simple que la création d’une image dynamique. Pour cela, on va utiliser une variable de message qui ne sera pas d’un type simple comme un sélecteur de date ou de texte (voir points 3 et 4), mais d’un type plus complexe : une CustomTableReference [1].

Comme son nom l’indique, il s’agit d’une variable de message qui est liée à une table relationnelle. Dans notre cas, nous allons associer cette variable à la table CellProducts [2], cela nous permettra de choisir un produit pour notre message.

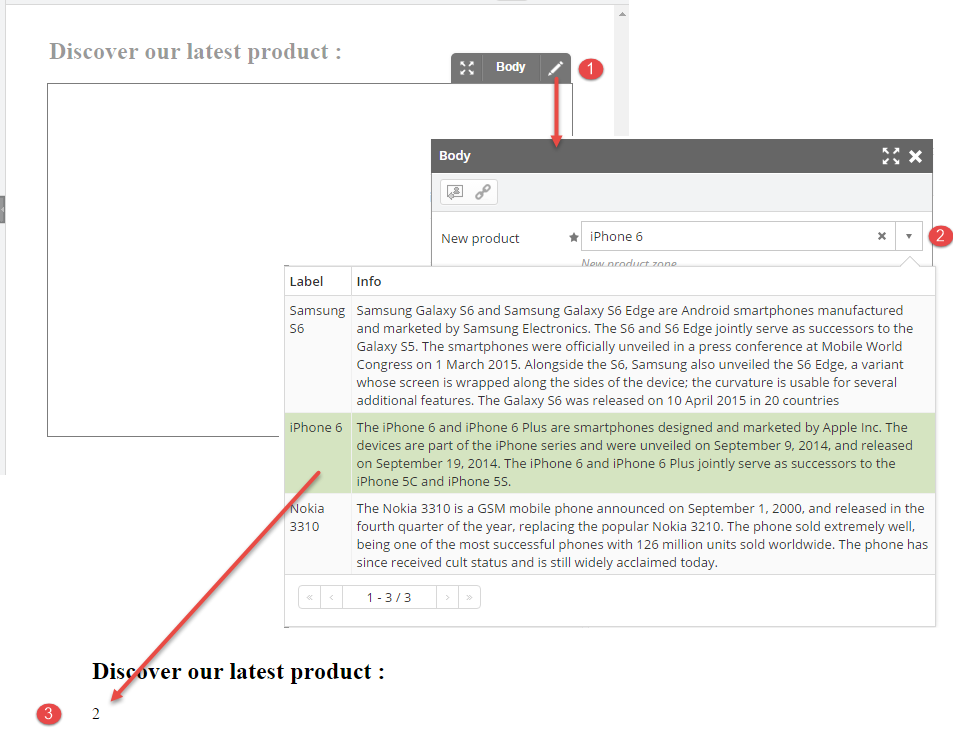
Cette table provient de la section «Donnée relationnelle» d’Openfield, voici la structure de table que nous allons utiliser dans notre exemple :

Afin de facilité cette sélection, nous allons choisir les champs à afficher pour l’utilisateur (l’étiquette et/ou la description par exemple) [3].Pour éviter de chercher dans une masse de données trop importante, on peut choisir d’affiner la sélection d’items disponibles pour la newsletter. Dans notre exemple, il peut être pertinent de n’afficher que les 10 derniers produits [4] ajoutés après le 1er janvier 2015 [5] en ordre descendant [6].

La variable des produits étant créés, il ne reste plus qu’à la rendre disponible pour l’utilisateur. Pour cela, il faut tout d’abord définir la zone qui va servir à sélectionner les variables du message (dans notre cas, la sélection d’un produit) [1] :

<div> [[ VisualEditor.EditZone(Version.Body); ]]</div>

Lorsque notre variable NewProduct aura un produit d’assigné, il faut gérer son affiche dans la page, en faisant attention à n’afficher le rendu du produit que s’il existe. Pour cela, il est possible de générer le code qui vérifie la présence de variable via un bouton [2]

Voici le résultat de l’utilisation de cette variable : le message contient une zone de 350px de haut tel que défini dans le code du gabarit permettant de choisir un produit par double-clics [1]. Cette sélection de produit affiche le top 10 des derniers produits ajoutés après le 1er janvier 2015 en ordre descendant, tel que défini dans la variable [2].

Il est maintenant possible de choisir un produit, et d’afficher son idRow (code généré automatiquement). Version.Body.NewProduct.CustomTableRecord contient donc l’ensemble des champs de l’enregistrement choisi. On peut donc récupérer les autre informations pertinentes de ce produit afin d’afficher une vraie fiche produit standard pour toutes les autres infolettres de ce type :

<div style=**"height:350px; width:750px;"** [[ VisualEditor.EditZone(Version.Body); ]]>

**[[**

**if** **(**Version**.**ContainsField**(**"Body"**)**

**&&** Version**.**Body **!=** **null**

**&&** Version**.**Body**.**ContainsField**(**"NewProduct"**)**

**&&** Version**.**Body**.**NewProduct **!=** **null**

**&&** Version**.**Body**.**NewProduct**.**isBlank**()** **==** **false)**

**{**

string ProductLabel **=** Version**.**Body**.**NewProduct**.**CustomTableRecord**.**Label**;**

string ProductImage **=** Version**.**Body**.**NewProduct**.**CustomTableRecord**.**Image**;**

string ProductInfo **=** Version**.**Body**.**NewProduct**.**CustomTableRecord**.**Info**;**

**]]**

<table align=**"center"** cellpadding=**"0"** cellspacing=**"0"** width=**"100%"**>

<tr height=**"37"**>

<td colspan=**"2"** height=**"37"**>

<h3>**[[=ProductLabel;]]**</h3>

</td>

</tr>

<tr>

<td style=**"font-size:11px; font-family: Arial; "**>

<img src=**"[[=ProductImage;]]"** width=**"240"** height=**"220"**>

</td>

<td style=**"font-size:11px; font-family: Arial"**>

**[[=ProductInfo;]]**

</td>

</tr>

</table>

**[[**

**}**

**else**

**{**

output**.**write**(**"Select a product."**);**

**}**

**]]**

<div>

**1**

**2**

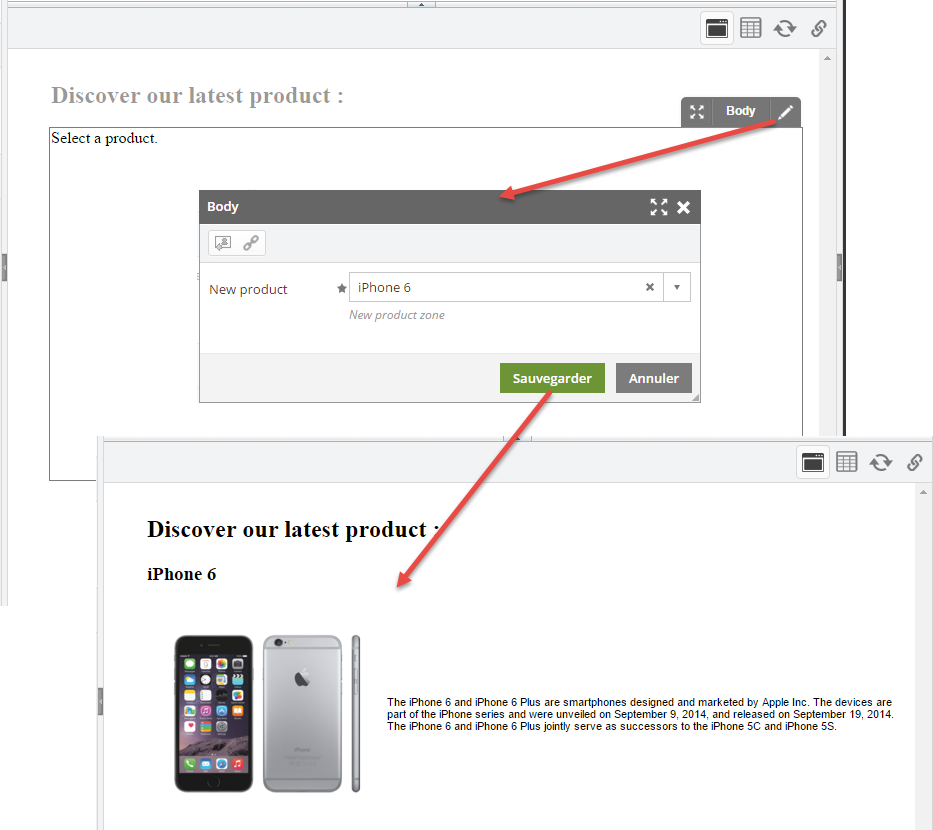
**3**

Explications:

* Si un produit a été sélectionné, on peut getter toutes les propriétés de ce produit dans la table relationnelle des produits

**3**

* Dans ce cas, on gère un affichage type pour ce genre de produit
* Et on gère également le cas où aucun produit n’a été sélectionné

Coté utilisation, un message invite l’utilisateur à choisir son produit. Après en avoir sélectionné un, l’affiche de ce produit devient fixe, il est possible de changer de produit, mais pas d’en altérer l’affichage :

Nous avons vu comment utiliser une variable qui est une référence d’un enregistrement d’une table relationnelle. Mais dans le cas d’un ou l’on voudrait une quantité variables d’enregistrement, l’utilisation de variables devient trop contraignante. Dans ce cas, une solution est d’utiliser des blocs de contenu (blockset en anglais), tel que vu dans les parties précédentes. Cela va nous permettre de définir une zone de dépôt personnalisée (zone de drop custom) qui pourra contenir plusieurs de produits.

Dans la partie suivante, nous allons prendre l’exemple d’une infolettre qui enverrait son top produits du mois. Le but est donc d’offrir à l’utilisateur une interface «figée» qui ne lui permettra que de sélectionner un ou plusieurs produits, l’affichage de ces derniers étant définit dans le gabarit.

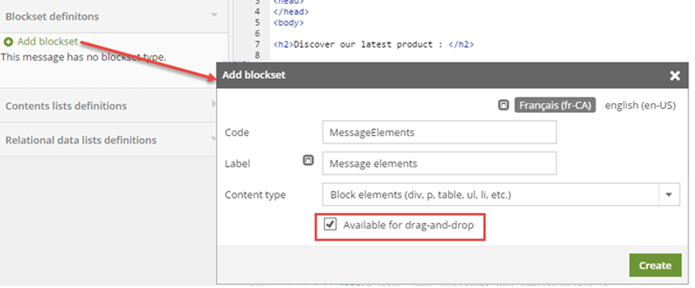
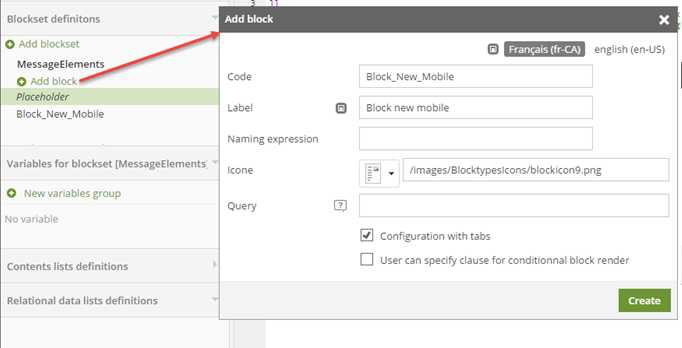
### Utiliser des éléments multiples de données de tables relationnelles

Comme nous l’avons vu dans les parties précédentes, un bloc de

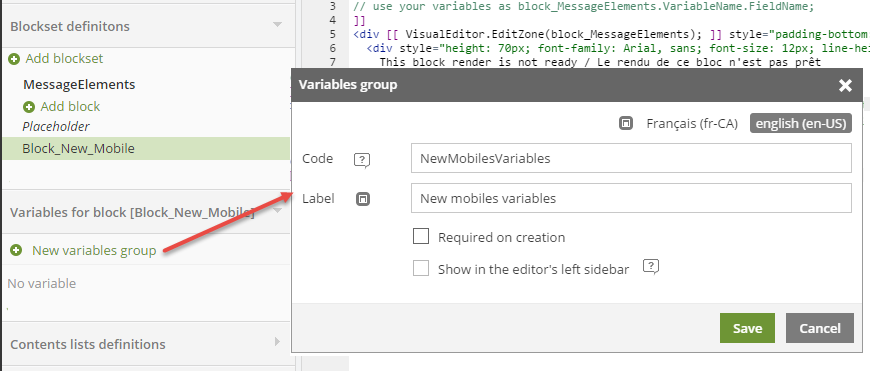
Cette définition de la table relationnelle des produits est pratique pour mettre à jour plus facilement la variable de produits précédemment définit, mais va surtout être utile pour remplir le bloc de contenu que nous allons créer afin de pouvoir afficher plusieurs produits.

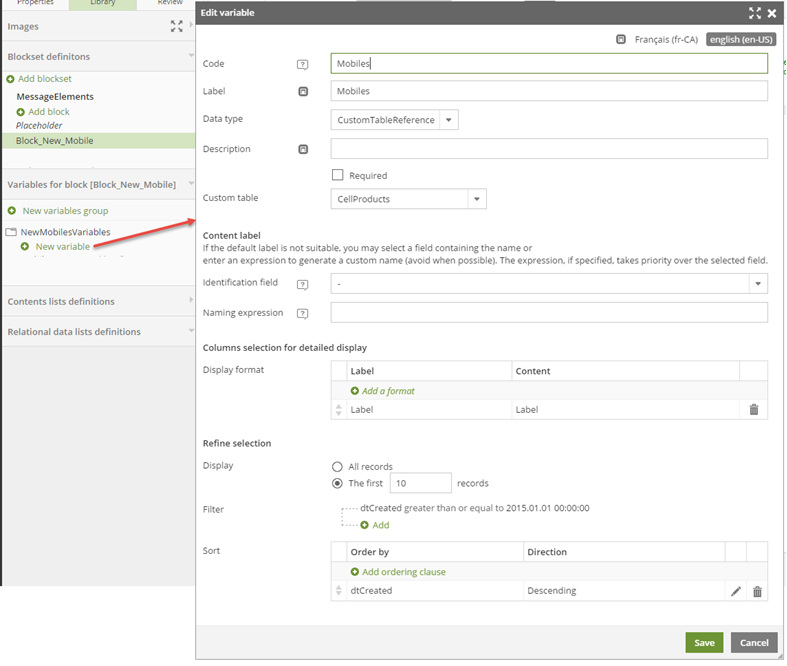
Pour cela, il faut :

1. Créer une nouvelle définition de bloc de contenu, qui servira à accueillir les différents blocs de contenus
2. Créer un nouveau bloc de contenu qui servira à gérer le rendu de son contenu
3. Créer une nouvelle variable de type CustomTableReference, spécifique à ce bloc.

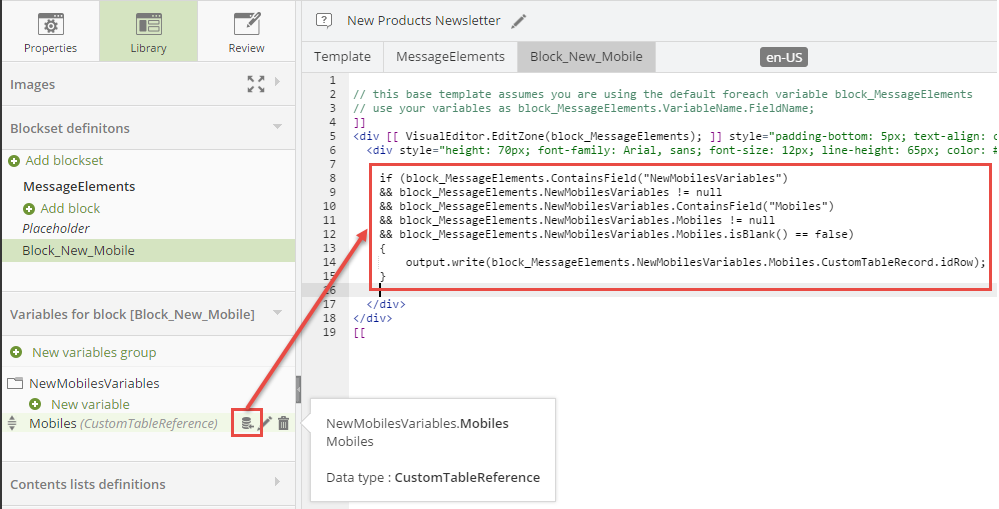
  
Cette zone permettra d’y déposer des blocs de contenus. On créer donc un nouveau bloc de contenu pour les produits de notre table relationnelle :

Après avoir sélectionné ce bloc, il est possible de lui ajouter des variables. On va donc ajouter





Maintenant que la variable est créée, on va pouvoir l’utiliser dans le bloc de contenu précédemment créé afin de gérer l’affichage du produit qui sera sélectionné. Pour cela, on peut générer un exemple d’utilisation, et le modifier avec de gérer l’affichage de chaque produit du « Block\_New\_Mobile » de la même façon que l’affichage d’un unique produit vu précédemment :



**// this base template assumes you are using the default foreach variable block\_MessageElements**

**// use your variables as block\_MessageElements.VariableName.FieldName;**

**]]**

<div [[ VisualEditor.EditZone(block\_MessageElements); ]] style=**"text-align: center;width:750px;"**>

**[[**

**if (block\_MessageElements.ContainsField("NewMobilesVariables")**

&& **block\_MessageElements.NewMobilesVariables != null** &&

**block\_MessageElements.NewMobilesVariables.ContainsField("Mobiles")** &&

**block\_MessageElements.NewMobilesVariables.Mobiles != null** &&

**block\_MessageElements.NewMobilesVariables.Mobiles.isBlank() == false)**

**{**

**string ProductLabel = block\_MessageElements.NewMobilesVariables.Mobiles.CustomTableRecord.Label;**

**string ProductImage = block\_MessageElements.NewMobilesVariables.Mobiles.CustomTableRecord.Image;**

**string ProductInfo = block\_MessageElements.NewMobilesVariables.Mobiles.CustomTableRecord.Info;**

**]]**

<div style=**"margin:10px; padding:10px; border: 2px solid #EEEEEE; box-shadow: 5px 5px 3px #EEEEEE;"**>

<table align=**"center"** cellpadding=**"0"** cellspacing=**"0"** >

<tr height=**"20"**>

<td colspan=**"2"** >

<h3>**[[=ProductLabel;]]**</h3>

</td>

</tr>

<tr>

<td style=**"font-size:11px; font-family: Arial"**>

**[[=ProductInfo;]]**

</td>

<td style=**"font-size:11px; font-family: Arial; "**>

<img style=**"display:block;"** src=**"[[=ProductImage;]]"** width=**"240"** height=**"220"**>

</td>

</tr>

</table>

</div>

**[[**

**}**

**else**

**{**

**output.write("No product selected.");**

**}**

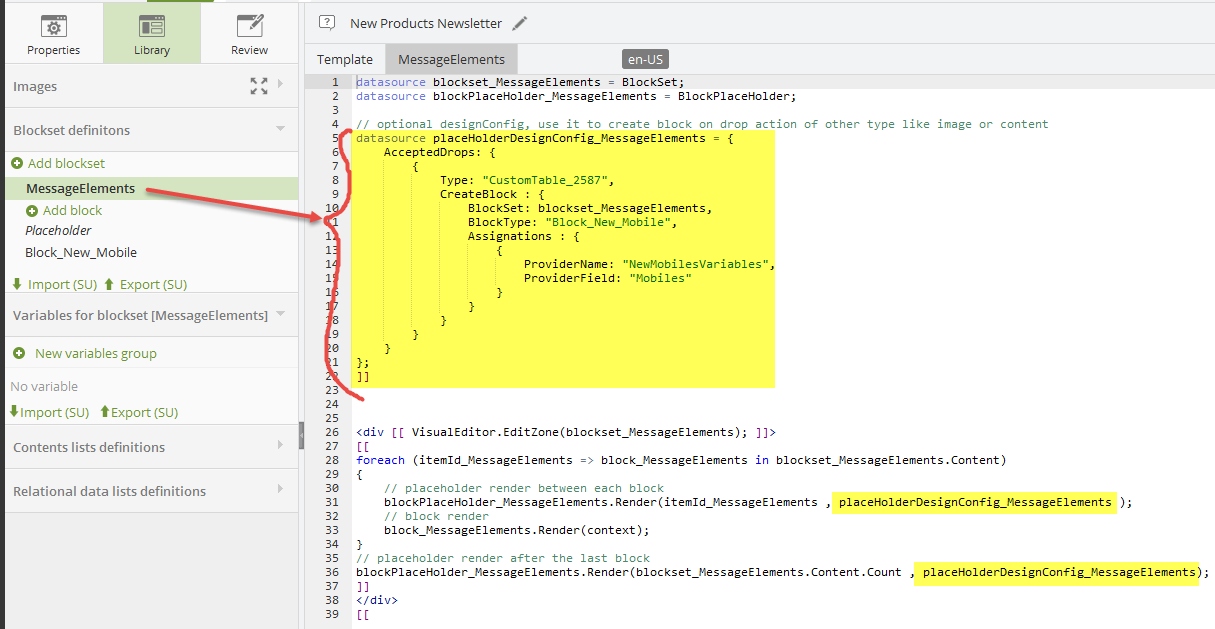
**]]**

</div>

**[[**

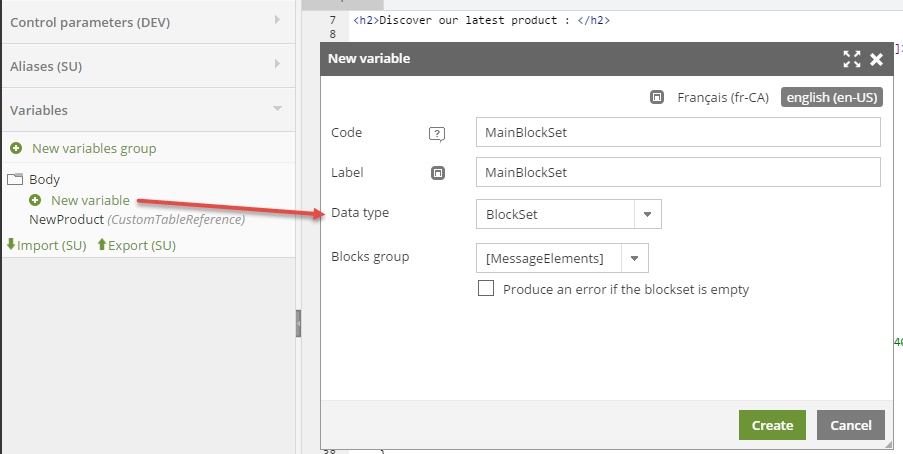
Maintenant que le rendu de chaque produit dans ce bloc est définit, il faut configurer le groupe de bloc précédement créer afin qu’il accepte seulement les elements du type de bloc que l’on vient de créer : Block\_New\_Mobile

Pour cela, il suffit de cliquer sur le groupe de blocs MessageElements et de configurer les datasource qu’il accepte. Nous choisirons ici un seul type de datasource correspondant à la table relationnelle des produits. Une fois un produit ajouté à ce groupe, on assignement automatiquement la variable précédemment créer avec ce choix :

Voici en jaune les parties du code générer à modifier afin que le groupe de blocs MessageElements accepte le dépôt de bloc de type Block\_New\_Mobile :

Explications :

Le groupe de bloc ***MessageElements*** accepte le dépôt d’un nouvel élément de type référence à la table relationnelle CellProduct (id=2587). L’ajout d’un de ces éléments va créer dans le groupe de bloc ***MessageElements*** un nouveau bloc du type ***Block\_New\_Mobile***. La variable ***Mobiles*** du groupe de variables ***NewMobilesVariables*** sera alors assigné avec la référence du produit déposé.

Après avoir fini de configurer l’affichage des blocs de produits, ainsi que le groupe de bloc qui les contiendra, il ne reste plus qu’à utiliser ce groupe de bloc dans une variable de message :

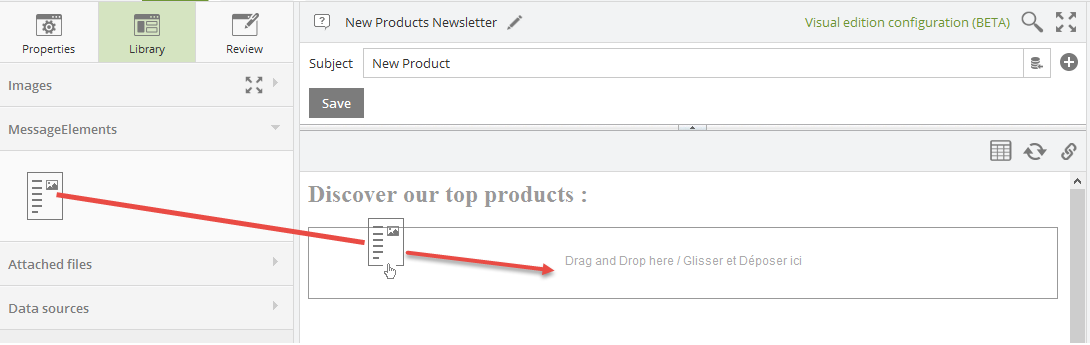
Une fois créée, cette variable peut être utilisée dans le gabarit de message afin de définir la zone dans laquelle nous pourrons déposer des produits :

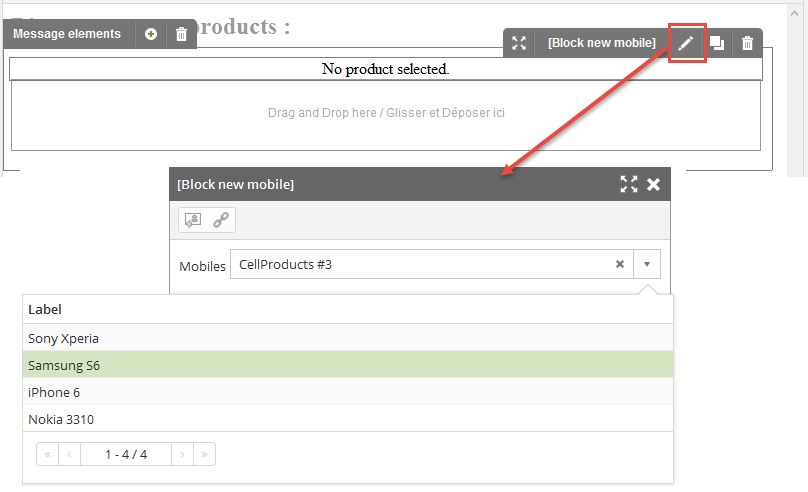
**<br/>**

[[Version.Body.MainBlockSet.Render();]]

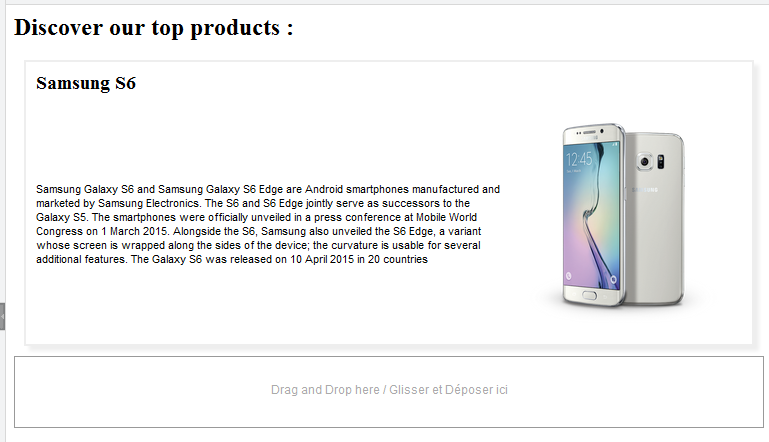
**<br/>**

Ainsi, coté utilisateur, il est possible de créer une infolettre de produit par simple sélection de ces derniers :



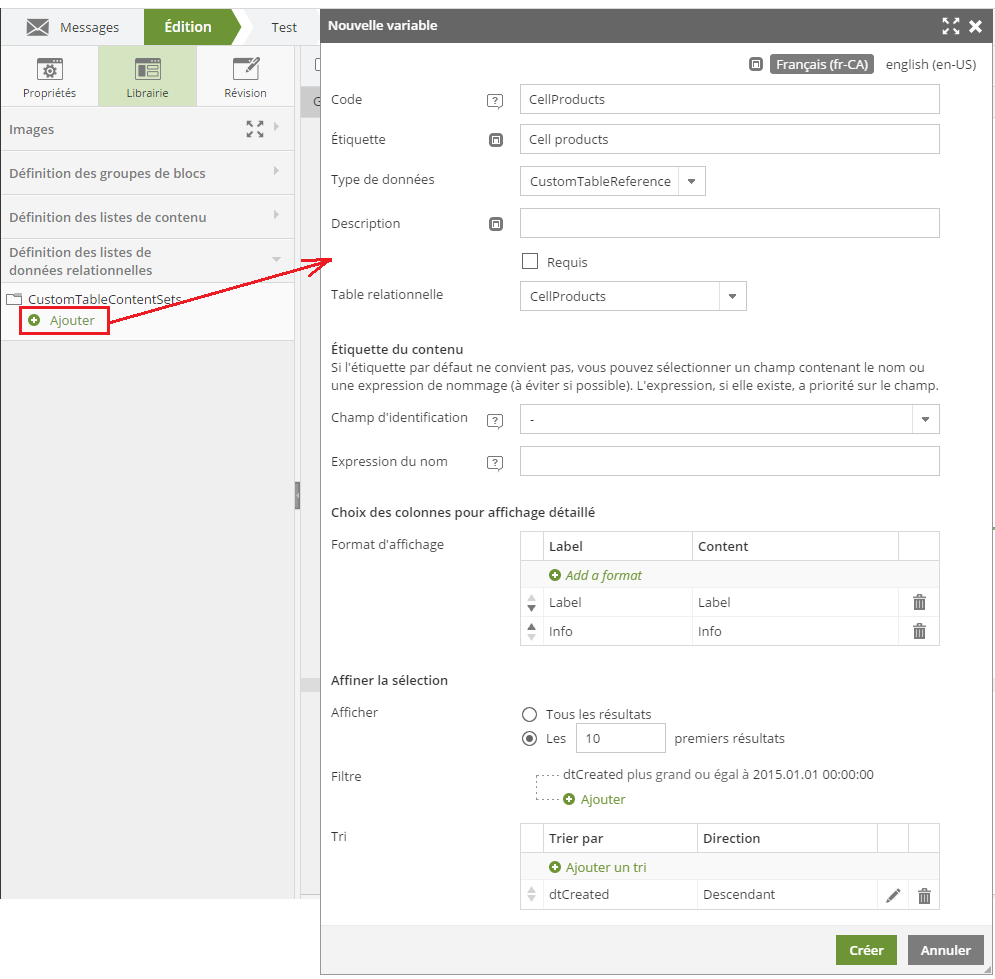


Apres avoir sélectionné un produit, son affiche est gérer par le groupe de bloc dans lequel il est contenu :

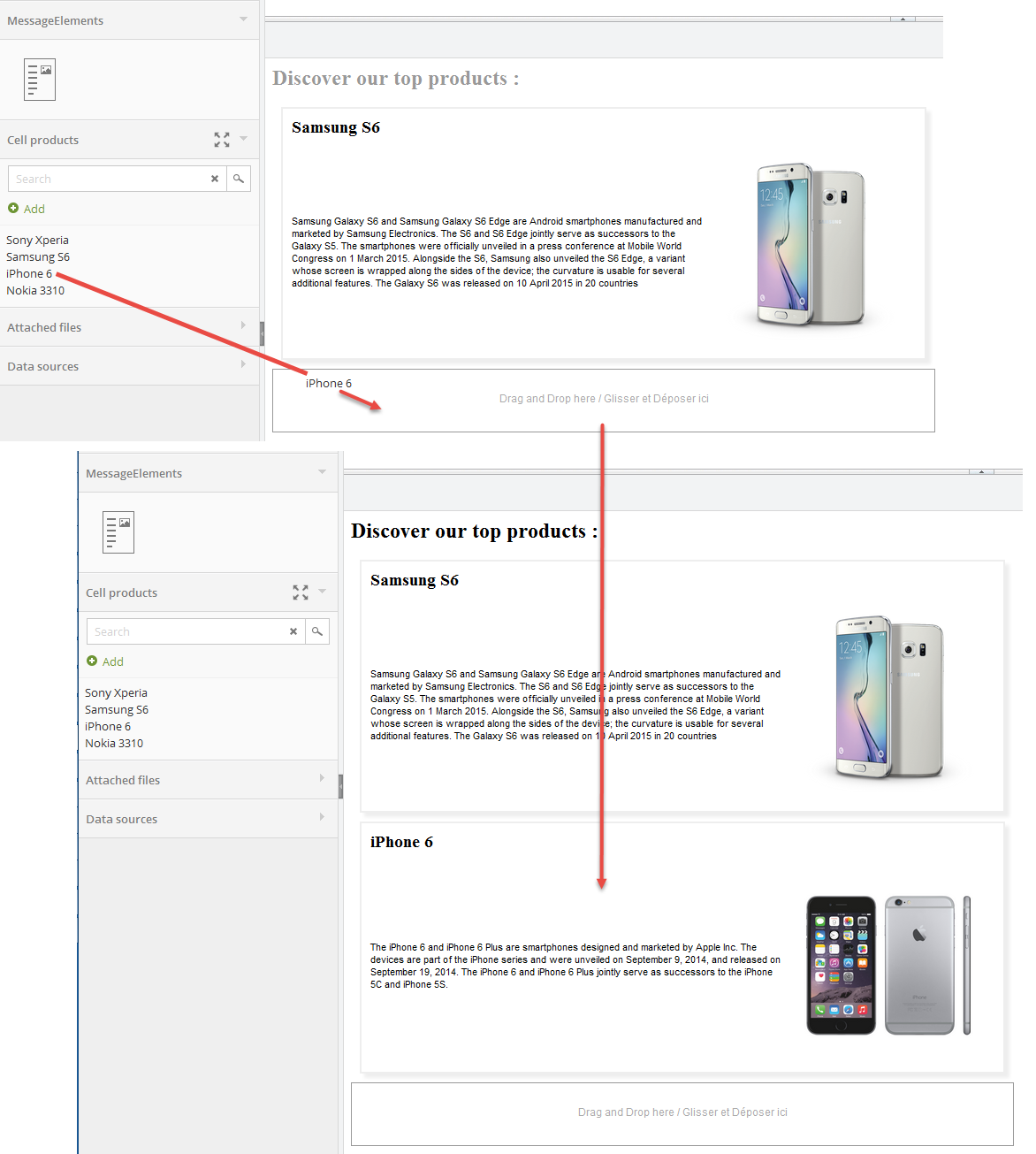


Il est ensuite possible d’ajouter une multitude de produits, d’en dupliquer, supprimer, etc.

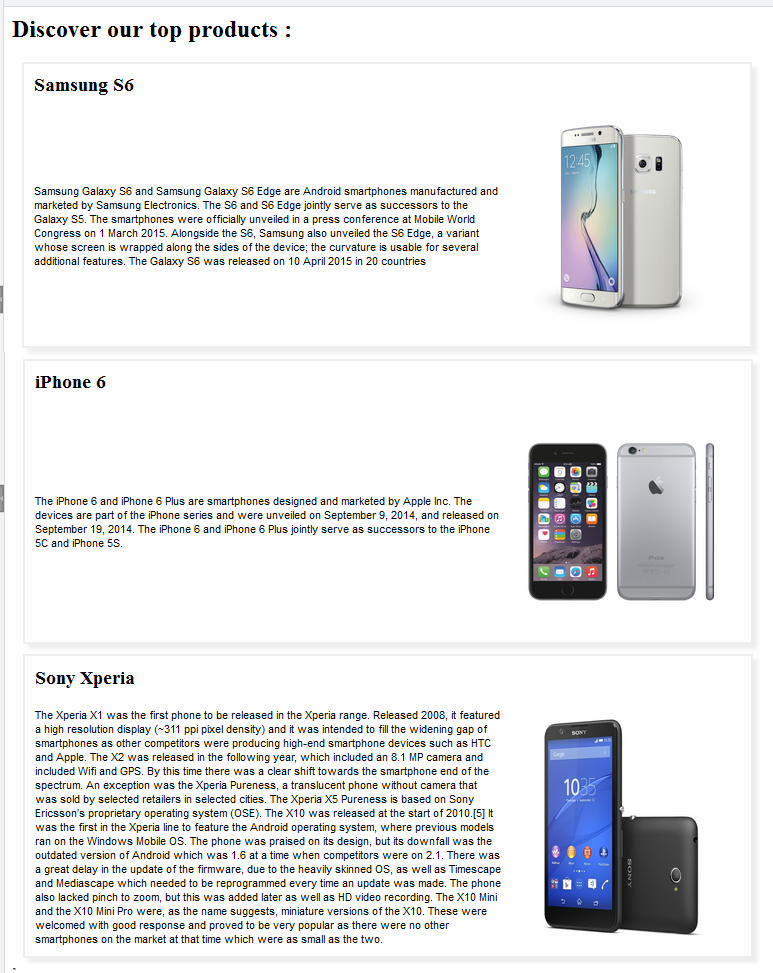
### Utiliser les listes de définitions de tables relationnelles

Un moyen encore plus pratique d’ajouter des items dans notre infolettre est d’utiliser une définition de listes de tables relationnelle. Le but de cette définition est le même que lorsqu’on définit une variable de table relationnelle : permettre de lister les éléments pertinents de la table relationnelle qui nous intéresse. L’intérêt est surtout d’éviter de devoir naviguer entre la gestion des données dans la table relationnelle et son utilisation dans l’éditeur de message. Ainsi en ajoutant une définition de notre table de produits directement dans l’éditeur de message, on peut afficher un nombre restreint de colonne de nos produits, déjà filtrés et ordonnés :

Une fois cette définition de notre table de produits terminées, il devient possible depuis l’éditeur de message de rechercher un produit dans la table relationnelle (parmi les items ciblé par la définition), de l’éditer, d’en ajouter et surtout de pouvoir en choisir un afin de le glisser-déposer dans la zone produit précédemment utilisé :



Résultat d’un infolettre présentant son top 3 produits :



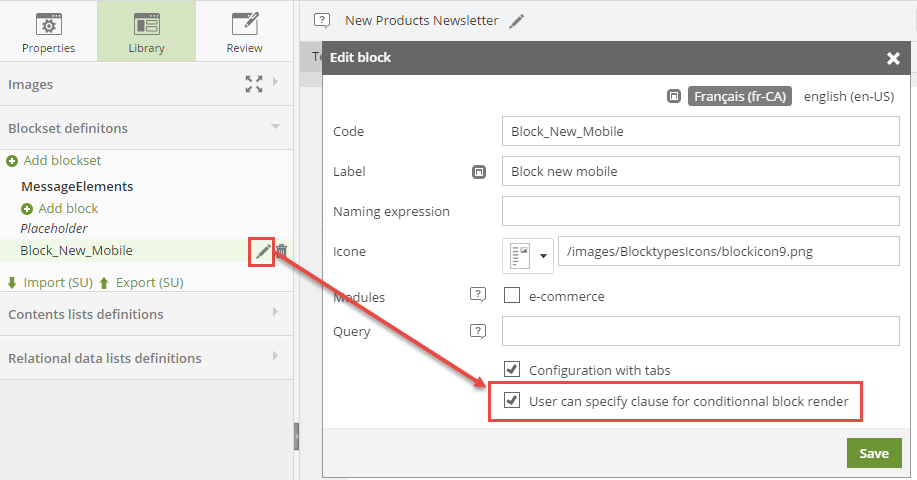
### Utiliser des variables ou des blocs d’éléments d’OpenField Contenu

### Différence avec les TRs

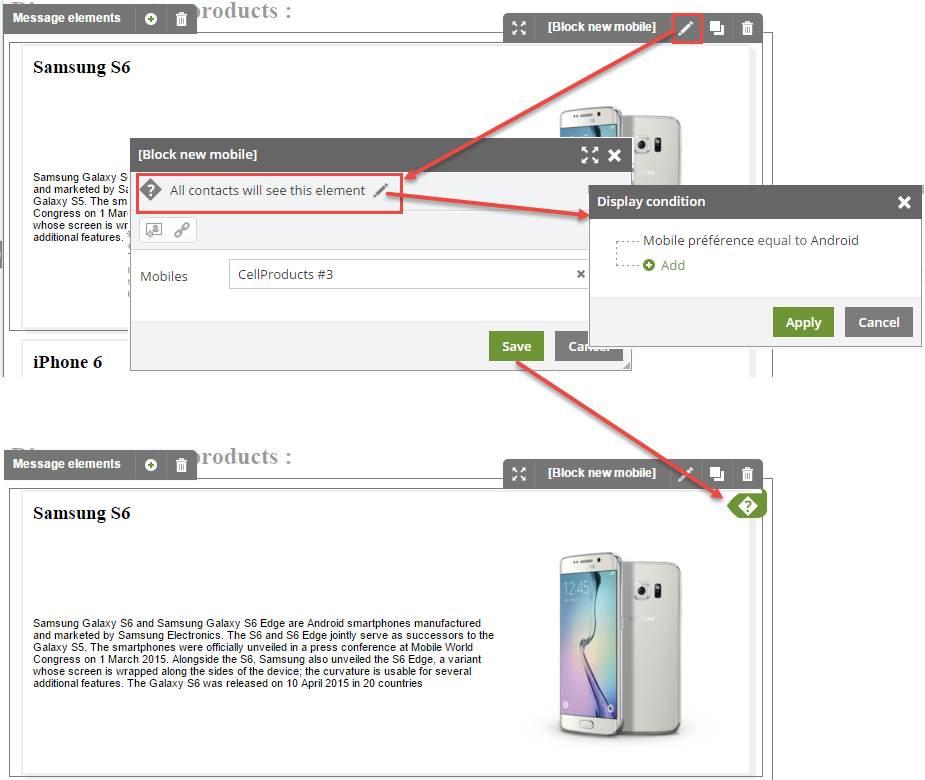
## Utiliser des blocs avec conditions d’affichages

Nous avons vu la possibilité de créer plusieurs bloc de contenu en glisser/déposé dans notre message, mais pour le moment, tous les messages contiendront les mêmes blocs pour tous les contacts ciblé. Il peut néanmoins être pertinent de n’afficher que certains blocs pour une partie des contacts.

* **Configuration :**

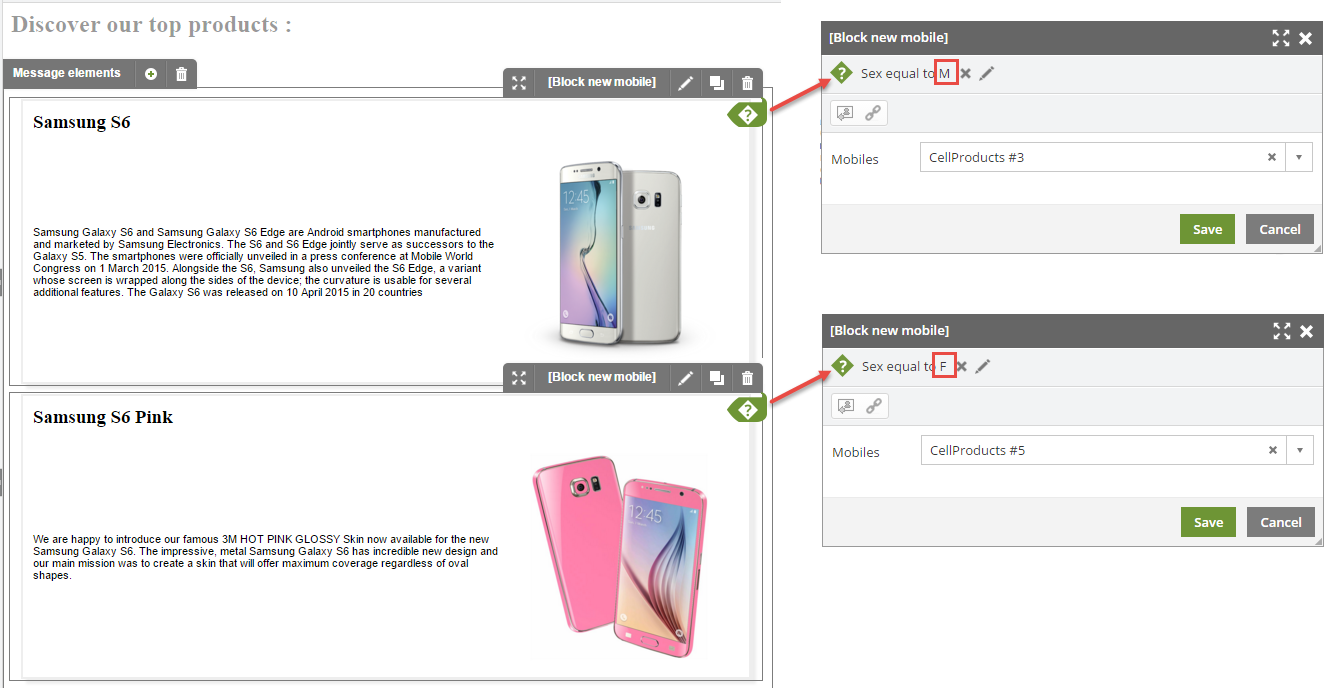
Pour cela il est possible de rendre certain bloc optionnel, basé sur les données du contact. Dans l’édition de notre bloc de produit, cocher la case «Permettre de définir une clause pour rendu condition»

Cette option va offrir à l’utilisateur la possibilité de définir un critère d’affichage pour chacun de ces types de blocs.

* **Utilisation**

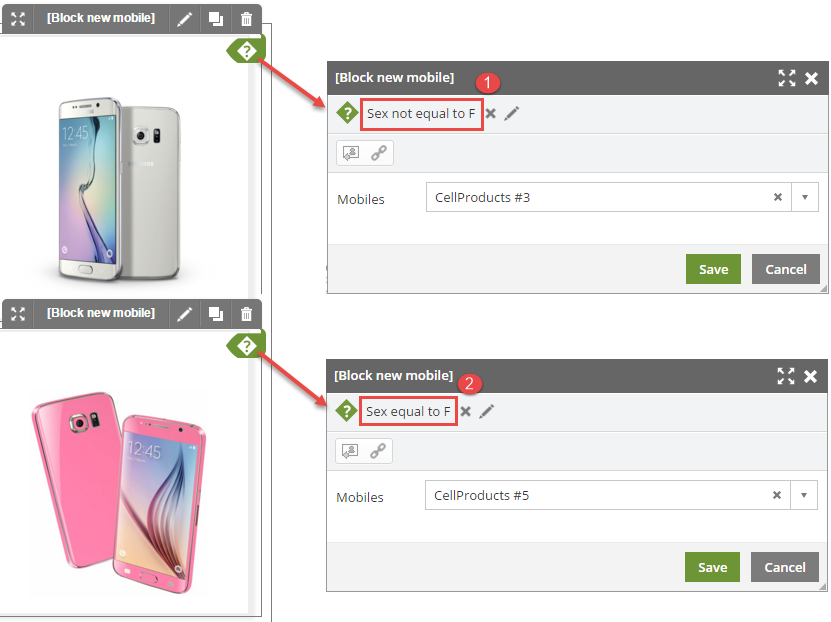
Pour utiliser une condition sur un bloc, il suffit d’éditer ce dernier en ajoutant un ou plusieurs critères d’affichages :

Le bloc ainsi configuré affiche un icone indiquant que ce bloc peut ne pas être affiché si la condition n’est pas respectée. Ainsi dans cet exemple, seuls les contacts dont le champ «Mobile preference» est égale à Android verront le premier bloc de ce message. Les autres blocs de ce message, n’ayant pas de conditions spécifié restent visible pour tous les contacts.

Habituellement, on utilise deux blocs ciblant des contacts qui ne peuvent pas se recouper. Par exemple : un bloc ayant du contenu ciblant les hommes, et un second bloc ayant du contenu ciblant les femmes. En spécifiant une condition sur ces deux blocs, on s’assure d’avoir du contenu personnalisé selon le sexe, en étant certain qu’aucun contact ne reçoit les deux contenus.

Attention toutefois aux contacts n’ayant pas de valeur spécifiée dans le champ utiliser dans la condition. Ce dernier pourrait recevoir un message n’ayant aucun bloc d’affiché.

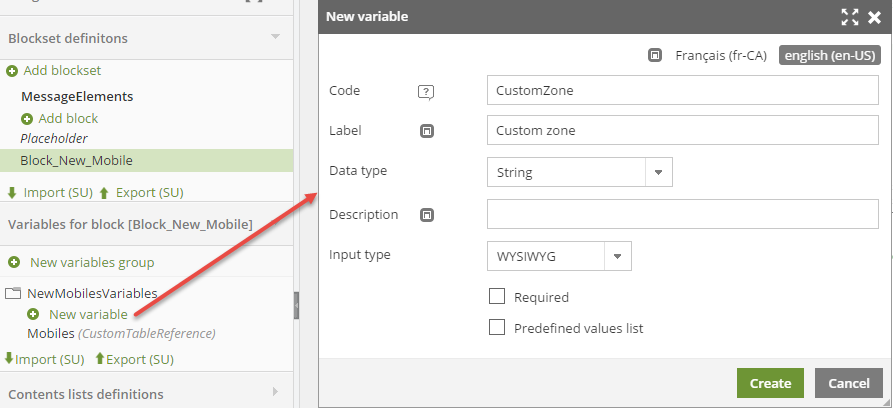
Pour être sûr qu’au moins un bloc soit affiché, quel que soit la condition appliquée, il est possible d’afficher un 3ème bloc ayant pour condition «*Sex not equal to F And Sex not equal to M*», ce dernier bloc sera affiché si on ne connait pas le sexe du contact. Une autre solution plus simple peut être de définir un des deux blocs comme «Bloc par défaut». Ce sera celui que l’on voudra afficher si on ne connait pas le sexe du contact. Ainsi, il suffit de lui appliquer la condition contraire de l’autre bloc comme ceci :



Ainsi, le second bloc sera affiché pour tous les contacts dont le champ Sex est égale à «F» [2], et le premier bloc sera affiché dans tous les autres cas [1] : champ égale à «M», champ vide, champ égale à une autre valeur (une erreur de saisie par exemple), champ null, etc.

## L’Édition de zone WYSIWYIG directe

Afin de rendre la création du message encore plus facile, il est possible de rendre certaines zones éditables directement dans la partie visuelle du message. L’utilisateur à juste à double-cliquer sur la portion de texte qu’il souhaite éditer, et peut y changer la valeur directement sans passer par des popups d’édition.

Cette zone d’édition directe ne peut être faite que dans un bloc, et seulement pour les variables de types WYSIWYG. On ajoute donc dans notre bloc produit une nouvelle variable :

Il ne reste qu’à ajouter le code qui gère l’affichage de cette variable dans le code du bloc de produit. Le VisualEditor doit absolument être sur un élément <Div> ou du même type.

<div [[ VisualEditor.EditZone(block\_MessageElements.NewMobilesVariables,

{Label:"", EditedSingleProviderField: "CustomZone", UseInlineWysiwygEditor : true,

ToolSide: "Top", ToolAlign: "Right", ToolPosition: "Inside"});]]>

[[if(block\_MessageElements.ContainsField("NewMobilesVariables")

&& block\_MessageElements.NewMobilesVariables.ContainsField("CustomZone")

&& block\_MessageElements.NewMobilesVariables.CustomZone != null

&& block\_MessageElements.NewMobilesVariables.CustomZone.length > 0)

{]]

[[=block\_MessageElements.NewMobilesVariables.CustomZone;]]

[[}else{]]

<div style="color: #aaaaaa; font-style: italic; text-align: center;">

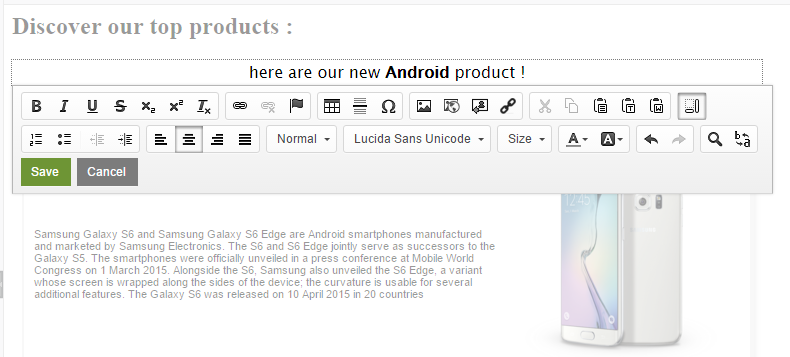
Insert Content

</div>

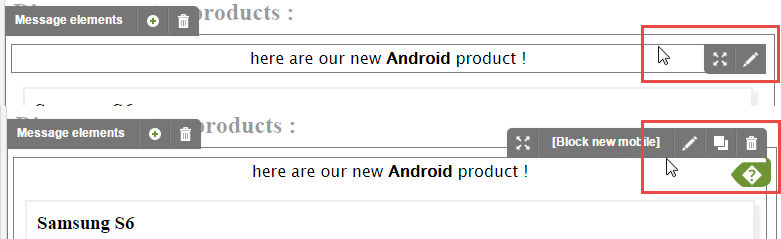
[[}]]

</div>

Coté utilisateur, un simple double-clique sur cette zone permettra d’éditer le texte :



En revanche, l’utilisation d’une zone d’édition wysiwyg en plus de l’édition du bloc peut rendre plus difficile le clique sur les boutons d’édition, car les deux zone se superposent :



Pour remédier à cela, nous ajouterons en mode édition visuel seulement un petit en-tête dans les blocs produits : le clique dans cet entête permettra de sélectionner le bloc, le clique sur la zone wysiwyg permettra d’éditer cette zone.

Pour cela, on ajoute au début du bloc de produit un élément de type DIV qui sera visible qu’en mode création de message :

[[ VisualEditor.IfDesignMode("<div class=\"BlockDragGripHeader\"></div>");]]

Puis, dans le corps du message, on ajoute le CSS de cet entête. Par habitude, on utilisera une barre hachurée grâce à ce CSS :

]]

<html>

<head>

<style type="text/css">

[[ VisualEditor.IfDesignMode("

.BlockDragGripHeader {

height: 15px;

border: 1px solid #bbbbbb;

background-color: #d7d7d7;

background-image: repeating-linear-gradient(-45deg, transparent,

transparent 20px, rgba(255,255,255,.5) 20px, rgba(255,255,255,.5) 40px);

}");

]]

</style>

</head>

<body>

<h2>Discover our top products : </h2>

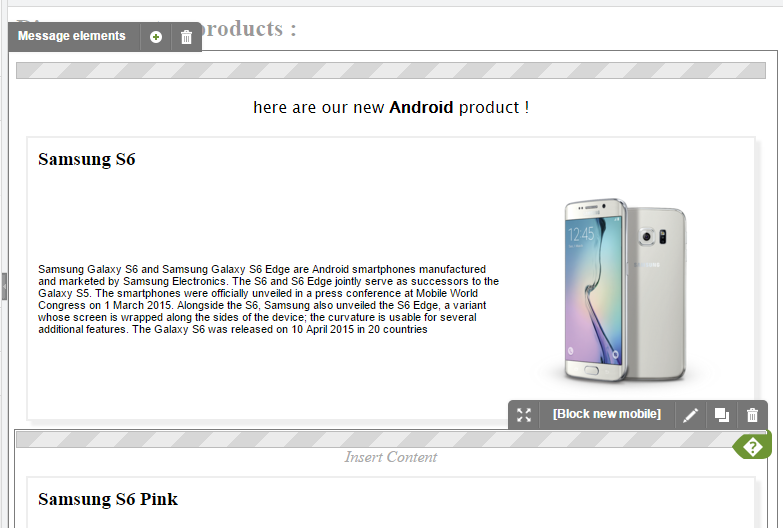
[[Version.Body.MainBlockSet.Render();]]

</body>

</html>

**[[**

Résultat :



## L’édition Live du texte

Il est aussi possible de rendre live l’édition de variables de type string dans des inputs de type textbox. Cette option est utile dans le cas où l’on veut seulement éditer un titre ou une amorce pour voir instantanément le rendu.

<td>

[[ VisualEditor.LivePreviewContainer(block\_Content.Content, "TextHTML"); ]]

</td>

## Bloc vidéo Youtube

Nous pouvons maintenant aller chercher l’image de prévisualisation d’un vidéo Youtube par l’aide d’un outil maison : lib.Youtube.GetMediumVideoThumbnail(UrlDuVideo).

Ceci retourne une datasource avec l’url du video et une moyenne et petite image de celui-ci.

datasource myVideo;

string myVideoLink = "";

myVideoLink = block\_Content.Content.Video;

myVideo = lib.Youtube.GetMediumVideoThumbnail(block\_Content.Content.Video);

Nous pouvons ensuite utiliser la datasource pour afficher l’image avec le lien Youtube

[[if(myVideo != null){]]

<a href="[[=myVideoLink;]]"[[ if(ImgAltText != ""){ ]] OFSYS\_LinkName="[[=ImgAltText;]]">[[ } ]]

<img src="[[=myVideo.URL;]]" width="[[=ImgWidth;]]" border="0" alt="[[=ImgAltText;]]" style="display:block; width:100%; max-width:[[=ImgWidth;]]px; height:auto; font-family:[[=BodyFontFamily;]]; color:[[=BodyTextColor;]]; font-size:18px; text-align:center;" />

</a>

[[}else{]]

<img src="https://placeholder-repsolution.netdna-ssl.com/S([[=ImgWidth;]]x100)/T(URL invalide)/image.png" width="[[=ImgWidth;]]" alt="" style="display:block; width:100%; max-width:[[=ImgWidth;]]px; height:auto;" />

[[}]]

## Les outils SU

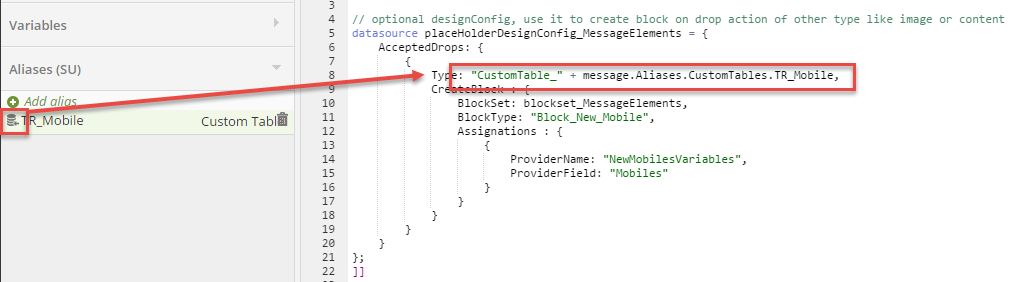
Cette partie s’adresse plus particulièrement aux employés de Dialog Insight qui ont des droits avancés sur l’application. Les options marquées «SU» sont surtout utilisés pour améliorer la création de gabarits de messages pour les clients (lorsqu’un intégrateur doit créer un gabarit de message réutilisable pour une compagnie par exemple)

### Les alias

### 

Idéalement, lorsqu’on utilise dans un gabarit de message qui utilise des données issues de tables relationnelles, d’OpenField Contenu ou d’une requête, on éviter d’utiliser les identifiant de ces éléments. En effet, si ce gabarit doit être dupliqué dans un autre projet ou une autre compagnie, les identifiants des tables dupliquées également ne correspondront pas à celle utiliser dans le message. Pour pallier à cela, on utilisera des alias de tables relationnelles, ou de requêtes :

L’alias «TR\_Mobile» correspond ici à l’identifiant de la table relationnelle. Il peut donc être utilisé pour remplacer cet ID dans le code du groupe de blocs :

Utilisation avec l’alias créé :

### Import/export de variables, Blockset, etc.

Certaines boites d’outils comme les Variables globales, les définitions de groupes de blocs ou les variables des blocs disposent de bouton d’imports ou export. Le but est surtout de gagner en productivité lorsque l’on souhaite copier ces données depuis un autre gabarit de message.

* Le bouton d’export permet d’afficher la structure des variables par exemple du gabarit d’origine au format XML.
* Le bouton d’import permet d’ouvrir un poup qui recevra une chaine XML.

**/!\** Attention, il n’est pas possible de supprimer une variable ou une définition de groupe de blocs qui est utilisé dans le gabarit du message. Cela se traduirait par un crash du message lors de son utilisation, il ne serait ainsi plus utilisable. L’importation de la structure de variables en XML a pour conséquence de remplacer toute la structure existante. Ainsi, si des variables existaient déjà et étaient utilisées dans le gabarit de message, cette importation aurait pour conséquence d’invalider le message. Il est donc nécessaire d’utiliser cette option sur un message vierge, ou du moins n’ayant pas encore de variables de créé.

