Guide du développeur de scripts d'extension

Chewing Word 1.6.4

Aristide Grange

Table des matières

Table des matières

1 Introduction

À partir de la version 1.6, l'utilisateur de Chewing Word a la possibilité d'exécuter des scripts extérieurs à l'application.

Il le fait en composant des mots spéciaux, appelés **commandes** et commençant par `cw`. Lorsqu'une commande est reconnue, le système effectue les opérations suivantes, dans cet ordre :

1. il efface le nom de la commande, antéfixe et suffixe inclus ;
2. il sélectionne une partie (éventuellement vide) du texte de l'éditeur ;
3. il appelle le script en lui transmettant son environnement ;
4. il récupère le résultat de l'exécution du script sous la forme d'une liste de directives : insertion de texte, ouverture d'une fenêtre de dialogue, etc. ;
5. il traite ces directives l'une après l'autre ;
6. enfin, il repasse en mode normal.

Tout comme Chewing Word lui-même, les scripts sont écrits en Python 2.7. En fait, dans le but d'épargner à l'utilisateur toute installation supplémentaire, l'interpréteur Python chargé de les exécuter n'est autre que celui qui est empaqueté avec le logiciel. Bien sûr, un script Python peut lui-même très bien faire appel à des scripts écrits dans d'autres langages, ou même à des exécutables.

L'inspiration de cette fonctionnalité provient :

– d'une part, du shell Unix, pour le principe de commandes textuelles lisant un canal d'entrée, accédant à un environnement et écrivant leur résultat dans un canal de sortie ;
– d'autre part, de l'éditeur de texte Textmate, d'Allan Oogard, pour la notion d'argument textuel automatique et le post-traitement du résultat par l'application.

2 Anatomie d'un script

Un script se présente sous la forme d'un sous-dossier du dossier dont l'emplacement est défini dans l'onglet « Scripts » du panneau de Préférences. Par convention, le nom de ce sous-dossier débute par `"cw"` et est formé exclusivement de caractères ASCII minuscules. Un même sous-dossier peut contenir la définition de plusieurs scripts (p. ex., `"cwpage"` contient les définitions de `cwpage`, `cwpages` et `cwdraft`.)

Pour fixer les idées, intéressons-nous au sous-dossier `"cwmail"`. Il contient non moins de 9 fichiers, que l'on peut regrouper en différentes catégories :

– les méta-données : ici, `"cwmail.json"` et `"fr_cwmail.json"`. Le premier fichier contient les méta-données « internationales » du script ; le second est dévolu à leur adaptation dans différentes langues (on parle de **localisation**), ici le français ;
– le script proprement dit, ou programme : ici, `"cwmail.py"` ;
– d'éventuelles annexes : ici, `"template.txt"`, `"archive_template.txt"`, et leurs versions locales : `"fr_template.txt"`, `"fr_archive_template.txt"`, de simples fichiers texte contenant les formulaires générés pour la composition et l'archivage d'un mail (respectivement) ;
– les tests unitaires : ici, `"cwmail_tests.py"`. Ils permettent, de façon automatique et sans lancer Chewing Word, de s'assurer du fonctionnement correct du script dans un certain nombre de cas prévus ;
– la documentation, accessible au sein même de Chewing Word par la commande `cwAide` : ici, `"fr_cwmail.t2t"`.

Penchons-nous maintenant tour à tour sur chacune ces catégories dans le cas général.

2.1 Méta-données

2.1.1 Méta-données internationales

Lorsque Chewing Word indexe les scripts (soit au lancement, soit à la demande de l'utilisateur), il les identifie en se basant sur la présence d'au moins un fichier dont le nom commence par `"cw"` et finit par `".json"`. Ce fichier est donc obligatoire.

Si votre script tourne sur une seule plateforme, c'est aussi dans le nom du fichier de méta-données que vous le préciserez, en insérant `"_mac"`, `"_win"` ou `"_lin"` avant l'extension (p. ex., `"cwadium_mac.json"`). S'il tourne sur exactement deux plateformes,

vous devrez actuellement créer deux versions du fichier, même si elles ne diffèrent que par leur nom. S'il est multiplateforme (par défaut), vous n'avez rien de spécial à faire.

À partir du contenu de ce fichier, Chewing Word crée un dictionnaire, c'est-à-dire une table associant des clefs à des valeurs. Par exemple :

```
"names": ["cwMail", "cwEmail"],
"type": "external",
"abstract": "Send your transcription, in whole or in part, as an electronic message.",
```

Les clefs sont `"names"`, `"type"`, `"abstract"`, suivies de leur valeur. Vous remarquerez que ces valeurs sont exprimées dans un sabir lointainement réminescent de l'anglais (la langue commune de l'informatique). Elles n'ont pas vocation à être vues par l'utilisateur final, mais à être traduites dans différentes langues (y compris en bon anglais, par un locuteur natif).

2.1.2 Méta-données localisées

Pour le moment, la seule langue officiellement prise en charge est le français. Une fois le dictionnaire créé, Chewing Word essaie donc de le **localiser** en français. Pour ce faire, il préfixe le nom du fichier de méta-données internationales (ici, `"cwmail.json"`) par deux lettres représentant la locale, suivies d'un soulignement : en l'occurrence `"fr_cwmail.json"`, qui est effectivement un fichier existant. Ce fichier est donc lu, et utilisé pour mettre à jour le dictionnaire. Cela donnera dans notre exemple :

```
"names": ["cwMail", "cwEmail", "cwMél", "cwCourrier"],
"abstract": "Envoyez votre transcription comme courrier électronique.",
```

Remarquez que la clef `"type"` est absente de ce fichier localisé. Sa valeur est en effet utilisée seulement en interne, et non destinée à être affichée. Elle ne nécessite donc pas d'être traduite.

La mise à jour est effectuée selon les règles suivantes :

- si une clef est présente à la fois dans le fichier international et le fichier local, la valeur locale « écrase » la valeur internationale ;
- si une clef n'est présente que dans un seul des deux fichiers, elle est conservée dans le résultat.

Remarquez également que les mises à jour sont « profondes » : dans le cas où une valeur consiste en un dictionnaire, celui-ci est non pas écrasé, mais mis à jour avec le nouveau dictionnaire, et ceci récursivement.

Le dictionnaire localisé contiendra finalement les entrées suivantes :

```
"names": ["cwMail", "cwEmail", "cwMél", "cwCourrier"],
"type": "external",
"abstract": "Envoyez une partie de votre transcription comme courrier électronique.",
```

2.1.3 Méta-données utilisateur

Il faut encore savoir que l'utilisateur a la possibilité de « patcher » un script à sa convenance sans modifier quoi que ce soit dans le dossier de celui-ci. Chewing Word, immédiatement après avoir lu un fichier de méta-données, cherchera s'il en trouve un de même nom dans un dossier situé au même niveau que celui des scripts et nommé `"scripts (user)"` (dossier créé automatiquement s'il n'existe pas). Le cas échéant, le dictionnaire actuel sera mis à jour avec son contenu.

2.1.4 Ordre des mises à jour

La constitution des méta-données d'un script se fait donc en cascade selon la séquence suivante :

1. création du dictionnaire « international » à partir d'un fichier JSON multi- ou mono-plateforme ;
2. mise à jour éventuelle à partir d'un fichier de même nom dans le répertoire `"scripts (user)"` ;
3. mise à jour éventuelle à partir d'une version localisée ;
4. mise à jour éventuelle à partir d'un fichier localisé de même nom dans le répertoire `"scripts (user)"`.

2.1.5 Méta-données communes

Au même niveau de l'arborescence que les dossiers de scripts, se trouve un dossier `"cw"` qui, lui, ne contient aucun script, mais seulement des fichiers de méta-données. Celles-ci sont mises à la disposition de tous les scripts, et constituent de ce fait un « pot commun » de valeurs d'intérêt général, réparties en `"static"` et `"prefs"`.

Exemples. `cwdate`, `cwgooglesearch`, `cwhelp`, `cwmail`, `cwsavevoice`.

2.2 Programme

2.2.1 Structure

À l'instar de la plupart des programmes Python, un script d'extension de Chewing Word commence en général par quelques importations : vous trouverez en référence la liste des modules disponibles.

Il doit obligatoirement comporter une fonction `main` à un argument, nommé `env` par convention, qui doit renvoyer une liste de directives, éventuellement vide.

Voici par exemple à quoi pourrait se réduire le programme `"cwip.py"` :

```
import urllib2

def main(env):
  return [("append",urllib2.urlopen("http://whatismyip.org").read()+" ")]
```

Remarquez que l'argument `env` est mentionné par obligation dans l'en-tête de `main`, mais ignoré dans le corps de la fonction.

La liste de directives est réduite à un élément. C'est un couple, avec à gauche le nom de la directive (`"append"`) et à droite une chaîne consistant au contenu d'une page web, concaténé à un espace.

2.2.2 Environnement

Bien sûr, au contraire de ce premier exemple, la plupart des scripts tiennent compte de l'environnement `env`. Pour avoir une idée de ce que votre script « connaît » au moment de son appel, nous vous conseillons de lancer Chewing Word et de valider la commande `cwEnvironnement`. Vous verrez apparaître dans une fenêtre une longue liste de champs répartis selon les rubriques suivantes :

`env["context"]`. Contexte de l'éditeur. Ces variables consistent en fragments de texte « remarquables » extraits automatiquement de l'éditeur, et accompagnés de leurs indices de début et de fin.

`env["meta"]`. Méta-données du script en cours d'exécution. Elles se divisent en valeurs dynamiques et statiques.

- Les valeurs dynamiques sont sans doute les plus utiles : elles permettent au script de savoir sous quelle forme il a été invoqué. L'utilisateur a en effet la faculté de lui passer des arguments supplémentaires en « antéfixant » ou en « suffixant » ceux-ci au nom de la commande. Pour les voir remplis, essayez par exemple de valider `3.14cwEnvironnementEtPatatiEtPatata`.
- Parmi les valeurs statiques, on retrouve les champs définis dans les fichiers en `.json` du script (à l'exception de `"prefs"` et de `"statics"` qui font chacun l'objet d'une rubrique à part entière), auxquels s'ajoutent quelques champs remplis automatiquement par Chewing Word lors de l'indexation du script.

`env["private"]`. Variables connues des scripts, mais pas de Chewing Word. Ce dictionnaire, vide au lancement du logiciel, offre une zone dans laquelle tout script en cours d'exécution peut librement créer, stocker et modifier des données destinées à être exploitées lors d'une exécution ultérieure, soit par le même script, soit par un autre.

`env["static"]`. Variables statiques connues par le script. Elles regroupent deux dictionnaires `"static"` : celui des fichiers de méta-données du script lui-même (notez que ce dictionnaire est réduit à une entrée pour `cwEnvironnement`) ; mais aussi, celui des méta-données communes, définies dans le fichier `"cw.json"` et sa version localisée.

`env["prefs"]`. Variables du panneau de Préférences.

  - En tête sont regroupées les variables ajoutées dans l'onglet « Scripts » du panneau de Préférences au moment de la lecture du fichier de méta-données des différents scripts. Cependant — et c'est sans doute un choix de conception plus inattendu — elles sont également accessibles de façon transversale par les autres scripts. Ceux actuellement livrés avec Chewing Word ne tirent pas parti de cette possibilité, mais on peut très bien imaginer des scripts interdépendants.
  - Ensuite viennent les autres réglages. Même si beaucoup ne concernent que l'aspect de l'interface, le fait qu'ils soient exposés aux modifications de l'utilisateur dénote un caractère relativement « externe ». Par exemple, c'est dans cette rubrique que le script `cwAudio` va récupérer la commande de synthèse vocale `env["prefs"]["speechCommandLine"]` qui génère un fichier audio à partir d'une sélection de texte. Notez que seuls les réglages du profil en cours sont accessibles par les scripts externes.

`env["constants"].` Définies dans le fichier `"constants.json"` et sa version locale, ou encore au moment du lancement du logiciel, la plupart de ces constantes contrôlent l'aspect visuel du logiciel (couleurs, dimensions, fontes par défaut, etc.) et n'auront donc *a priori* qu'un rôle interne. Cependant, certaines pourront être exploitées dans les scripts externes, par exemple le répertoire de travail (`env["constants"]["workingDirectory"]`) ou le langage de l'interface (`env["constants"]["language"]`).

Les modifications qu'un script apporterait à ces dictionnaires au cours de son exécution sont perdues à sa sortie, à deux importantes exceptions près, `env["prefs"]` et `env["private"]`, que nous étudierons en présentant les différentes techniques de persistance.

2.2.3 Directives

Le résultat de la fonction `main` doit donc être une liste de directives, que Chewing Word aura charge d'interpréter pour assurer le post-traitement du script.

Si cette liste est vide, la sélection de texte opérée, soit manuellement, soit automatiquement avant le lancement du script, restera telle quelle dans l'éditeur au terme de son exécution.

Une directive consiste en un couple dont le premier terme est le nom de la directive (donnée sous forme de chaîne) et le second terme, son argument. Le type de cet argument (unique) dépend de la directive : chaîne, nombre, dictionnaire (équivalent à plusieurs arguments), voire `None` pour simuler une directive sans argument.

Se reporter à la liste des directives disponibles.

2.2.4 Exceptions

Si le script termine sur une erreur, Chewing Word la rattrape et la met en forme pour l'afficher dans un dialogue. Cette issue est à éviter autant que possible en appliquant nos recommandations sur le traitement des erreurs.

2.3 Fichiers annexes

2.3.1 Accès

Au moment de l'exécution d'un script, le répertoire de travail (renvoyé par `os.getcwd()`) devient le dossier du script. Cela signifie que vous pouvez accéder aux fichiers qu'il contient en mentionnant simplement leur nom.
Par exemple, dans `"cwmail.py"` :

```
form = codecs.open(env["static"]["cwMail_template"],"r","utf8").read()
```

... où l'entrée `"cwMail_template" est définie dans “"fr_cwmail.json"` :

```
"static": {
  ...
  "cwMail_template": "fr_template.txt",
  ...
},
```

Il est bien sûr possible d'accéder à des fichiers situés au-dehors de ce répertoire, mais ce faisant vous perdrez le caractère *self-contained* de votre script, ce qui risque de limiter son déploiement ou compliquer sa distribution.

2.3.2 Nature

Les fichiers annexes peuvent consister, entre autres, en :

- des feuilles de style ;
- des formulaires (p. ex., `"initial_template.txt"`, `"template.txt"`, `"archive_template.txt"` et leurs versions localisées dans `"cwmail"`) ;
- des fichiers-journaux (p. ex., `"log.scpt"` dans `"cwcopyinto"`) ;
- des fichiers de données, textes (p. ex., `"mots faciles.txt"` et `"mots difficiles.txt"` dans `"cwhangman"`), images (p. ex., `"matrix.gif"` dans `"cwpage"`) ;
- des scripts ou des exécutables invoqués par le script Python lui-même (p. ex. `"send.scpt"`, `"launch.scpt"`, `"switch.scpt"` dans `"cwadium"`, ou `"keystrokes.py"`).

Exemples. `cwcopyinto.py`, `cwenvironment.py`, `cwhangman.py`, `cwhelp.py`, `cwkeys_mac.py`, `cwmail.py`.

2.4 Tests unitaires

2.4.1 Motivation

Idéalement, chaque script est fourni avec une batterie de tests qui couvrent l'ensemble des scénarios d'utilisation possibles. Ils ont principalement deux buts :

- contrôle du bon fonctionnement du programme ;
- documentation « vivante » du comportement attendu.

Nous vous renvoyons à cet article de Wikipédia pour un point d'entrée sur les tests unitaires et la méthodologie du développement guidé par les tests (TDD).

2.4.2 Environnement de tests

La plupart des scripts font un usage massif de leurs méta-données ou du contexte transmis par Chewing Word. Pour les tester, il convient donc de leur fournir tout ce dont ils ont besoin.

Reconstruction manuelle. L'approche minimale consiste à recréer explicitement la partie de l'environnement nécessaire au bon fonctionnement du programme testé.

Par exemple, dans `"cwdate_tests.py"` :

```
def testDay(self):
  env = {
    "prefs": {
        "cwDate_dateFormat": "%d",
        "cwDate_locale": ""
      },
    "static": {
        "cw_locale": ""
      }
    }
  self.assert_(0 < int(main(env)[0][1]) < 32)
```

Si elle offre le maximum de souplesse, ne perdez pas de vue que cette méthode découplera totalement les tests d'avec l'environnement réel du programme testé, ce qui peut les faire échouer à tort au cas où vous apporteriez ultérieurement des modifications dans les fichiers de méta-données. Relancez donc fréquemment les tests pour vérifier qu'ils sont encore à jour.

Exemples. `cwdate_tests.py`, `cwsavevoice_tests.py`, `cwimport_tests.py`, `cwstats_tests.py`, `cwunicode_tests.py`.

Construction à partir des méta-données. L'autre approche consiste à se baser sur les fichiers de méta-données livrés avec le script. Nous fournissons à cet effet dans le module « commun » `scriptgoodies` une fonction `getTestingEnvironment` qui facilite leur intégration. Si vous souhaitez l'utiliser, vous devrez rajouter ces trois lignes à votre programme de tests :

```
import sys
sys.path.append("../")
from scriptgoodies import getTestingEnvironment
```

Ensuite de quoi il vous suffira d'énumérer en argument les noms des fichiers de méta-données à exploiter, par exemple dans `"cwkeys_mac_tests.py"` :

```
  def setUp(self):
    env = getTestingEnvironment("cwkeys_mac.json","fr_cwkeys_mac.json")
    self.parse = MacKeyStrokes(env).parse
```

... L'environnement est créé en fonction de `"cwkeys_mac.json"` et mis à jour avec `"fr_cwkeys_mac.json"`. Les clefs `"context"`, `"constants"`, `"meta"` et `"private"` sont associées à des dictionnaires vides.

Exemples. `cwadium_tests.py`, `cwbrowse_tests.py`, `cwkeys_mac_tests.py`, `cwmail_tests.py`, `cwmenu_mac_tests.py`, `cwpastebin_tests.py`, `cwweather_tests.py`.

2.5 Documentation

La documentation d'un script est au format txt2tags, un langage de balises poids-plume facile à écrire et à lire. Chewing Word inclut la version 2.6 du moteur txt2tags. Celui-ci assure la conversion vers une multitude de formats, entre autres HTML (utilisé dans le prévisualiseur web intégré) et LaTeX (pour générer indirectement des PDF).

2.5.1 Sections

Le texte que vous insérez ou référencez sous la clef `"description"` apparaîtra dans une section de deuxième niveau intitulée :

```
== Description ==
```

Cela signifie que vous aurez droit à des divisions de troisième niveau et plus. Voici quelques exemples de divisions génériques de troisième niveau employées dans les scripts existants :

- Problématique
- Principe
- Tutoriel : découverte des différentes possibilités à travers une suite d'exemples
- Utilisation (— de base, — avancée)
- Méthodes alternatives
- Référence : toute liste un peu longue de fonctionnalités, d'options ou de mots-clés
- Astuces
- Avertissement / Remarque

Concernant le style, n'hésitez à recourir à de nombreux exemples concrets, avec des commandes prêtes à l'emploi qui pourront être copiées telles quelles dans l'éditeur de Chewing Word.

2.5.2 Inclusion d'images

Les figures et les copies d'écran sont très appréciées. Vous mettrez les fichiers correspondants dans le même dossier. Attention, leur insertion dans le texte obéit à une syntaxe des plus contournées. Ainsi, pour inclure une image intitulée `"nounours.png"`, vous écrirez :

```
[""%(path)s/nounours"".png]
```

Notez le `%(path)s/`, qui doit être écrit tel quel, ainsi que la position des deuxièmes doubles guillemets !

Afin de centrer l'image, la ligne doit commencer et finir par un espace, même s'ils ne sont pas visibles ci-dessus.

3 Techniques

3.1 Modalités du passage de paramètres

Strictement parlant, un script n'attend qu'un seul paramètre : l'environnement `env`. On verra qu'en réalité, celui-ci est un dictionnaire contenant plusieurs centaines de valeurs. Bien sûr, un script donné n'en exploitera que quelques-unes. Il appartient au concepteur du script de se mettre à la place de l'utilisateur pour déterminer lesquelles le conduiront le plus simplement possible à ses fins. Nous passons ici en revue les plus importantes, à savoir :

- la sélection, qui peut être réalisée manuellement ou automatiquement ;

– le suffixe, qui se présente sous forme appauvrie ou enrichie ;
– l'antéfixe ;
– le nom sous lequel la commande a été effectivement invoquée.

3.1.1 Sélection manuelle

Au moment où la commande est validée, Chewing Word commence par lui chercher une sélection explicite, que l'utilisateur aura réalisée manuellement. Celle-ci, si elle existe, se trouve immédiatement après le curseur, et consiste en un texte encadré d'une balise de début (parenthèse ouvrante et signe égal) et d'une balise de fin (parenthèse fermante), par exemple :

```
cwMaSuperCommande(=Ma sélection manuelle)
```

Pour créer ces balises, l'utilisateur aura procédé comme pour définir une abréviation : il aura sélectionné la zone à passer en argument, composé sa commande : dès la première lettre (en l'occurrence, un « c »), les balises seront apparues autour de la sélection.

3.1.2 Sélection automatique

Opérer une sélection manuelle peut s'avérer fastidieux ou délicat pour certaines personnes. C'est pourquoi le concepteur du script essaiera plutôt de définir une sélection automatique pertinente.

En l'absence d'argument textuel explicite, Chewing Word opère la sélection automatique spécifiée dans le champ `"autoSelect"` (défini dans le fichier de méta-données associé à la commande) : la page, le paragraphe en cours, la dernière phrase, le dernier mot, ou rien du tout.

3.1.3 Suffixe alphabétique

Indépendamment de la sélection (manuelle ou automatique), Chewing Word analyse le nom de la commande lui-même pour transmettre d'autres informations au script.

On sait qu'une commande prévue pour s'écrire `cwTradu` pourra également être invoquée par `cwTraduction`, `cwTraduis`, `cwTraduire`, etc. ; autrement dit, en suffixant son nom officiel d'une suite quelconque de lettres. L'intérêt de cette disposition va au-delà de la tolérance aux flexions. Chewing Word détache le **suffixe** ainsi constitué, et l'ajoute au dictionnaire `env["meta"]` du script sous deux formes :

Forme appauvrie. Le suffixe est réécrit à l'aide des lettres apparaissant sur les touches du cadran. Ainsi, si la commande composée est `cwTraduisMoiÇa`, `env["meta"]["poorSuffix"]` vaudra `ISMOICA`.

Forme enrichie. Le suffixe, pourvu de tous ses diacritiques, est découpé selon les changements de casse. Pour la même commande, `env["meta"]["poorSuffix"]` vaudra donc `"is Moi Ça"`.

Si vous ne vous intéressez qu'à la conservation des diacritiques, pour accéder à la forme originelle du suffixe (sans découpage) utilisez l'expression Python :

```
env["meta"]["richSuffix"].replace(" ","")
```

Pour vous débarrasser des capitales, évaluez :

```
env["meta"]["richSuffix"].lower()
```

Le découpage a été introduit pour pallier l'impossibilité de composer un nom de commande contenant des espaces : en effet, l'Espace est aussi la touche de validation ; au premier composé, la commande serait immédiatement interprétée. Nous vous déconseillons toutefois d'exploiter ce découpage pour autre chose que des fonctions non indispensables : en effet, la composition d'un mot incluant des changements de casse demande à l'utilisateur de saisir le mot complet (sans le valider par Espace), puis de repasser sur la gomme, puis dans la bulle, puis de cliquer toutes les lettres dont il souhaite changer la casse. Plutôt que de lui imposer toutes ces cérémonies, réfléchissez à lui faire passer le texte correspondant, non en suffixe, mais en argument textuel. Parmi les scripts énumérés ci-dessous, seuls `cwfont` et `cwpage` tirent effectivement parti du découpage. Par exemple, `cwPageRapportFinal` créera une page intitulée « Rapport Final ». Il n'était pas possible d'utiliser l'argument textuel pour le titre, puisque celui-ci sert au transfert de texte entre pages.

Exemples. `cwadium_mac`, `cwcopyinto`, `cwfont`, `cwhangman`, `cwhelp`, `cwkeys_mac`, `cwmail`, `cwmap`, `cwpage`, `cwpreferences`, `cwsudoku`.

3.1.4 Antéfixe non alphabétique

Le système de composition de Chewing Word interdit également de suffixer des caractères non alphabétiques à un nom de commande.

Pour contourner cette limitation, nous avons introduit la possibilité de les précéder d'un **antéfixe** (Chewing Word réserve le terme plus courant de *préfixe* au début d'un mot). Ainsi, lorsque vous composez :

```
Tour Eiffel, Paris 18cwcartesatellite
```

... en plus d'une sélection automatique (« Tour Eiffel, Paris ») et d'un suffixe (« satellite »), vous associez à `env["meta"]["antefix"]` la chaîne `"18"` que `cwCarte` interprétera comme le facteur de zoom.

Cet antéfixe peut être constitué de toute suite de caractères non blancs ni alphabétiques : signes de ponctuation, opérateurs mathématiques... ou même caractères chinois !

Notez qu'à l'instar du suffixe, l'antéfixe est effacé préalablement à l'appel de la commande. Il appartient au script lui-même de le restituer si besoin. C'est ce que fait `cwunicode` de façon à permettre la composition de deux caractères Unicode consécutifs :

```
("replace",env["meta"]["antefix"]+"".join(result))
```

Exemples. `cwfont`, `cwmap`, `cwunicode`.

3.1.5 Nom effectif de la commande

Un script peut tirer parti du nom sous lequel il a été effectivement invoqué. Par exemple, si l'utilisateur valide `cwHeure`, c'est la commande `cwDate` qui sera exécutée, mais avec la chaîne `"CWHEURE"` comme valeur de la clef `env["meta"]["actualPoorName"]`, ce qui permettra au script de formater le résultat comme une heure et non comme une date.

Exemples. `cwdate`, `cwpage`.

3.2 Critères de choix

Après avoir énuméré les cinq modalités de passage d'arguments : sélection manuelle, sélection automatique, préfixe, antéfixe, nom effectif, il convient sans doute de dégager quelques critères permettant de préférer l'un ou l'autre stratégie dans telle ou telle situation. L'idée est de trouver le meilleur compromis entre puissance et facilité d'utilisation.

3.2.1 Préférez l'implicite à l'explicite

Si le script a besoin d'un fragment de texte, essayez de le déterminer automatiquement plutôt que d'obliger l'utilisateur à le sélectionner. Délimiter une zone est une opération possible, mais non triviale pour beaucoup de personnes handicapées, principalement dans les modes les plus dégradés : clic automatique ou défilement. Choisissez donc avec soin la valeur que vous associerez à la clef `"autoSelect"`.

Par exemple, la commande `cwStats` affiche le nombre de signes, de mots, de phrases et de paragraphes présents dans la sélection. La sélection automatique sera donc de préférence `"page"`, parce que le plus souvent, les contraintes sur le nombre de signes portent sur la totalité du texte. Si, de temps à autre, c'est sur une autre zone que l'utilisateur souhaite avoir des renseignements, il pourra toujours la sélectionner explicitement :

en considération de la peine qu'elle a coûtée, toute sélection manuelle a priorité sur le traitement spécifié dans `"autoSelect"`. Ne vous résignez donc à choisir `"empty"` que dans le cas où le script n'a intrinsèquement pas besoin d'argument textuel.

3.2.2 Limitez le nombre de commandes

Toute dénomination est arbitraire. À part les geeks, qui ne sont de toute façon pas prêts à passer de Vim à Chewing Word, personne n'a envie de mémoriser des dizaines de mots-clefs pour piloter un ordinateur. Soyez astucieux, et exploitez à fond le contexte d'appel.

Par exemple, pour gérer une messagerie instantanée (ou *chat*), avons-nous vraiment besoin de : `cwLancerMessagerie`, `cwSélectionnerCorrespondant`, `cwEnvoyerMessage`, `cwChangerStatut` ? Toutes ces commandes seront avantageusement remplacées par une seule, comme le prouve `cwAdium` (sur Mac OS X seulement) :

- `cwAdium`, lancera le logiciel ;
- `cwAdiumAnatole`, passera à Anatole (opération qui peut être combinée avec la suivante) ;
- `Bonjour, ça va? cwAdium`, enverra le message (la forme la plus simple de la commande est réservée à l'opération la plus fréquente) ;
- `Je reviens tout de suite cwAdiumStatut`, modifiera le statut.

Autre exemple, lorsque nous avons conçu le système de pagination de Chewing Word, si nous n'avions réfléchi qu'en termes de fonctionnalités offertes, nous aurions abouti aux commandes suivantes : `cwCréerPage`, `cwDétruirePage`, `cwAllerVersPage`, `cwTransférerVersPage`, `cwRenommerPage`, `cwRevenirPagePrécédente`, `cwFusionnerPages`, `cwListerPages`, `cwModifierProfilPage`. Au lieu de quoi nous avons une seule commande, `cwPage`, qui couvre tous ces besoins, et même quelques autres que vous découvrirez dans l'aide.

Une autre façon d'épargner les noms est illustrée par ce que nous appelons les « commandes réentrantes », une technique suffisamment intéressante pour mériter sa propre section.

3.3 Commandes réentrantes

Il s'agit de commandes dont l'exécution insère dans le texte, soit la même commande, soit une autre, mais dans tous les cas avec un argument textuel pré-rempli. Cette nouvelle commande attend d'être validée, et sa validation permet de passer à l'étape suivante.

`CwMail`, par exemple, lorsque les informations sur le serveur d'envoi n'ont pas été fournies dans les Préférences, génère la commande `cwPréférences` avec les champs à remplir. De même, lorsqu'elle est invoquée sans argument textuel explicite, elle génère un formulaire d'envoi autour du contenu de la page en cours. Ce formulaire est encadré

de balises de sélection manuelle, et précédé d'une commande qui n'est autre que `cwMail`. Une fois ce formulaire complété, il suffira d'insérer un espace après cette dernière pour lui fournir son argument, ce qui déclenchera l'envoi du message.

Un autre exemple, assez rigolo, est la commande `cwPendu`, qui se regénère elle-même à chaque fois avec comme argument explicite le nouvel état du jeu (mot à deviner, dessin du pendu), jusqu'à ce que le joueur ait gagné, perdu, ou abandonné la partie.

Exemples. `cwhangman`, `cwmail`, `cwpreferences`.

3.4 Persistance

Techniquement, un script ne constitue pas un processus séparé : c'est un module Python qui est dynamiquement *importé* (par `__import__`) ou *rechargé* (par `reload`) à chaque nouvelle invocation. Cependant, comme un processus, il « naît » lors de son appel et « meurt » (héroïquement) en rendant son résultat. Au cours de son exécution, il n'a *per se* aucun souvenir de ses « vies » antérieures.

C'est pourtant parfois un besoin légitime, et diverses techniques ont été conçues pour y satisfaire.

3.4.1 Modifications textuelles

La technique la plus transparente consiste à « mémoriser » le résultat d'un appel en le présentant dans l'éditeur. On a vu par exemple, que chaque nouvelle invocation de `cwPendu` affichait un nouvel état du mot deviné et du progrès de la pendaison de façon à pouvoir l'utiliser en argument du prochain appel.

Exemples. `cwmail`, `cwhangman`.

3.4.2 Utilisation d'une variable privée

La limite de l'approche précédente est qu'elle ne permet pas de mémoriser une donnée que l'on voudrait cacher à l'utilisateur, ici en l'occurrence le mot à deviner. Pour atteindre ce but, cette donnée sera mise en réserve dans le dictionnaire `env["private"]` que nous avons décrit plus haut.

```
env["private"]["cwHangman_secret"] = random.choice(secrets).upper()
```

Concernant la forme du nom des clefs, utilisez les mêmes conventions que pour celles de `env["prefs"]` ou `env["static"]`.

Une application avancée de cette technique est illustrée par la commande réentrante `cwsudoku`. Au premier lancement, elle instancie une classe et stocke l'objet correspondant (qui contient tout l'état du jeu) dans une variable privée. Ce qui est intéressant, c'est qu'au sortir du script, le module importé est « délié », oublié en quelque sorte : cependant l'objet stocké survit avec toutes ses méthodes, prêt à être réutilisé lors des appels suivants (qui, eux, ne réinstancient pas la classe). Les avantages sont multiples : séparation plus claire de l'algorithme et de ses attributs ; calculs effectués une seule fois ; accès à l'état complet par une clef unique.

Attention : tout ce que vous mettez dans ce dictionnaire privé est perdu en quittant Chewing Word.

Exemples. `cwhangman`, `cwsudoku`.

3.4.3 Modification des préférences

Il est également possible de modifier le dictionnaire `env["prefs"]`. Ne pas oublier alors d'ajouter la directive de mise à jour des préférences (c'est inutile seulement si les entrées modifiées appartiennent toutes à des scripts).

Cette possibilité ouvre évidemment la porte à des modifications malveillantes du comportement de Chewing Word, mais il est probable que les méchants auront d'autres chats à fouetter que les utilisateurs d'un obscur clavier virtuel ! En attendant, faites bon usage du surcroît de puissance apporté par ce mécanisme.

Exemples. `cwfont`, `cwsudoku`.

3.4.4 Fichiers

Mentionnons finalement la technique la plus générale, qui consiste à stocker dans un fichier temporaire les données à mémoriser.

Par défaut, ce fichier sera placé dans le même dossier que le script. Cependant, cet emplacement n'est pas recommandé pour stocker quelque chose qui devrait survivre au remplacement du dossier des scripts, typiquement, par une version plus récente récupérée sur le site. Dans cette situation, préférez le dossier `"scripts (user)"` existant au même niveau que le dossier `"scripts"`, et qui accueille déjà les méta-données utilisateur. Pour sauvegarder un fichier `"rollmops.txt"` à cet endroit, vous écrirez quelque chose du genre :

```
path = os.path.join("../../scripts (user)/rollmops.txt")
codecs.open(path,"w","utf8").write("J'aime les rollmops!")
```

Cette technique est mise en œuvre dans le script `cwsudoku` pour éviter de proposer deux fois la même grille à l'utilisateur.

Exemples. `cwsudoku`.

3.5 Gestion des erreurs

3.5.1 Gardes

Un test préliminaire évite quelquefois d'exécuter des traitements qui mèneraient fatalement à une exception. Cela permet d'afficher un message ou d'effectuer un traitement plus adapté à la situation.
Par exemple, dans `"fr_cwsynonyms.py"`, on commence par vérifier que la sélection existe. Dans le cas contraire, plutôt que d'importuner le serveur en requérant les synonymes du mot vide, on termine directement sur le message approprié :

```
  if env["context"]["selection"] == "":
    return [("messageBox",env["static"]["cwSynonyms_emptyWordMessage"])]
```

Autre technique, lors de l'invocation de la commande `cwmail`, si l'une des clefs `"cwMail_smtp"`, `"cwMail_port"` ou `"cwMail_user"` de `env["prefs"]` se trouve vide, il n'y a aucune chance de réussir à finaliser l'envoi. Plutôt que de laisser l'utilisateur foncer dans le mur, le script génère un formulaire lui permettant de les renseigner :

```
cwPréférences(=
# Pour pouvoir envoyer des mails, commencez par compléter les champs suivants.

# cwMail - SMTP - serveur
cwMail_smtp =

... (etc.) ...

# Quand vous aurez fini, validez en insérant un espace avant la parenthèse ouvrante.
)
```

Exemples. `cwmail`, `fr_cwsynonyms`.

3.5.2 Filtrage des exceptions

Sachez que par défiance héréditaire envers la perfide Albion, une proportion non négligeable de vos compatriotes prennent les messages rédigés en anglais comme autant d'injures personnelles. Ménagez leur susceptibilité en filtrant au maximum ces messages offensants. Par exemple, la fonction `main` du script `"cwip.py"` que nous avions donnée plus haut dans une forme minimale, devrait plutôt s'écrire :

```
def main(env):
  try:
    return [("append",urllib2.urlopen("http://whatismyip.org").read()+" ")]
  except urllib2.URLError as err:
    if "[Errno 54]" in str(err) or "Error 403" in str(err):
      return [("messageBox",env["static"]["cwIp_connectionResetMessage"])]
    if "[Errno 8]" in str(err):
      return [("messageBox",env["static"]["cwIp_hostNameNotFoundMessage"])]
    raise err
```

... avec, dans "`fr_cwip.json`", les variables statiques suivantes :

```
    "cwIP_connectionResetMessage": "Requête refusée par le serveur, réessayez
                                    dans un instant.",
    "cwIP_hostNameNotFoundMessage": "Nom d'hôte introuvable, vérifiez votre
                                     connexion au réseau."
```

Exemples. `cwip`, `cwkeys_mac`, `fr_cwsynonyms`.

3.5.3 Limitation du temps d'exécution

Chewing Word n'est pas capable d'interrompre un script dont l'exécution durerait trop longtemps. Si le vôtre risque de se trouver dans ce cas, il vous appartient d'en gérer vous-même le *timeout*. Nous vous conseillons de définir le temps d'exécution maximal parmi les méta-données "`prefs`" de façon à laisser à l'utilisateur la possibilité de changer sa valeur.

Exemples. Aucun.

3.5.4 Fichier-journal

Certains scripts peuvent créer un fichier *log* documentant le déroulement de leur exécution. En cas de problème, le concepteur ou l'utilisateur d'un script aura ainsi plus de facilité à en comprendre la cause.

Exemples. `cwcopyinto`, `cwkeys_mac`.

4 Référence

4.1 Clefs du dictionnaire d'un fichier de méta-données

L'ordre des clefs est quelconque, on les énumère ici par ordre alphabétique.

4.1.1 `"abstract"`

Présence : conseillée.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : description du script en une ligne. Adressez-vous directement à l'utilisateur final, en le vouvoyant et en lui expliquant sans jargon, de la façon la plus concise possible, ce qu'il peut faire avec votre script. Cette ligne de description apparaîtra dans la liste affichée après réindexation des scripts, au lancement de la commande `cwAide` ainsi qu'au début de la documentation de chaque script.

4.1.2 `"autoSelect"`

Présence : facultative.
Valeur parmi : `"page"`, `"paragraph"`, `"sentence"`, `"word"`, `"empty"`.
Inclus dans la documentation : oui.
Rôle : il s'agit de la zone de texte de l'éditeur qui sera sélectionnée automatiquement si l'utilisateur n'a pas opéré de sélection manuelle. Pour plus d'informations, reportez-vous à la description de la sélection automatique et à la référence de l'auto-sélection.

4.1.3 `"authors"`

Présence : conseillée.
Type : liste de chaînes Unicode.
Inclus dans la documentation : oui.
Rôle : noms des auteurs du scripts. N'indiquez ici que les auteurs qui ont participé sciemment à l'écriture. Si vous vous êtes borné à adapter à Chewing Word un script existant, ou que vous faites appel à un service web, c'est dans la rubrique `"credits"` que vous l'expliquerez et citerez les auteurs originaux. Dans ce cas, assurez-vous que leur travail est placé sous une licence compatible avec la vôtre, ou à défaut contactez-les pour leur demander une autorisation.

4.1.4 `"bugs"`

Présence : facultative.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : indiquez ici, sous forme d'une liste txt2tags (chaque ligne commençant par un tiret), les bogues et limitations connues. Dans l'aide individuelle du script, cette rubrique se terminera systématiquement par un paragraphe donnant des instructions pour signaler une bogue ou suggérer une amélioration.

4.1.5 `"categories"`

Présence : conseillée.

Type : liste de chaînes Unicode.

Inclus dans la documentation : oui.

Rôle : identificateurs (en anglais) des catégories du script. L'utilisateur final les verra traduits selon la table suivante :

Identificateur	Traduction affichée
`"communication"`	communication
`"fun"`	divertissement
`"games"`	jeux
`"internet"`	internet
`"life"`	vie quotidienne
`"meta"`	méta
`"productivity"`	productivité
`"sound"`	son
`"system"`	système
`"geek"`	geek
`"utilities"`	utilitaires
`"writing"`	saisie

Dans l'avenir, si le nombre de scripts d'extension atteint des proportions effrayantes, les catégories serviront à en faciliter la recherche.

La liste des catégories est définie dans le fichier de méta-données communes `"cw.json"` et localisée en français dans `"fr_cw.json"`. Vous pouvez donc en définir vous-même de nouvelles, mais n'abusez pas de cette possibilité.

4.1.6 `"changes"`

Présence : conseillée.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : historique des versions, sous forme d'une liste txt2tags en ordre chronologique inversé, par exemple :

```
- **1.1.** Ajout d'une option pour sauver le monde.
- **1.0.** Version initiale.
```

Ne créez pas ce champ si la seule version est la version initiale.

4.1.7 `"contact"`

Présence : conseillée.

Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : indiquez les adresses des auteurs et/ou des personnes ou organismes en charge de la maintenance, avec toute précision éventuelle (forme libre). Attention, les adresses mentionnées se retrouveront en clair sur internet : utilisez donc des adresses dédiées et attendez-vous à du spam.

4.1.8 `"description"`

Présence : conseillée.

Type : chaîne Unicode.

Inclus dans la documentation : oui.

Rôle : c'est ici qu'est défini le « corps » de la documentation. Vous avez deux possibilités : si le script est suffisamment facile d'emploi pour que sa description tienne en quelques mots, insérez-les directement :

```
"description": "Cette commande ouvre une fenêtre avec une liste de synonymes du
                mot sélectionné (ou par défaut, du mot précédent). Cliquer sur
                OK remplace le mot original par le mot sélectionné.",
```

Alternativement, créez un fichier texte au format txt2tags et référérencez-le en insérant son nom (que Chewing Word reconnaîtra à son extension `".t2t"`) :

```
"description": "fr_cwmail.t2t",
```

Se reporter aux conseils de rédaction de la documentation.

4.1.9 `"license"`

Présence : conseillée.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : précisez sous quelle licence vous souhaitez placer votre script, en donnant un lien vers son site web. Si vous choisissez CeCILL 2.0 (la licence de Chewing Word, compatible avec la GPL), mettez simplement `"CeCILL"` : l'adresse correspondante sera automatiquement affichée dans les documentations générées.

4.1.10 `"maintainer"`

Présence : conseillée.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : nom des personnes ou des organismes en charge de la maintenance du script, possiblement distincts des auteurs. Par exemple, si l'auteur du script est un stagiaire, il est sans doute plus pérenne de mettre le nom de son employeur.

4.1.11 `"minCwVersion"`

Présence : facultative.
Type : chaîne Unicode couverte par l'expression régulière `"\d+(\.\d+){,2}"`.
Inclus dans la documentation : oui.
Rôle : numéro de version minimale de Chewing Word (1.6 par défaut), en deçà duquel le script sera silencieusement ignoré.

4.1.12 `"module"`

Présence : conseillée.
Type : identificateur Python.
Inclus dans la documentation : oui.
Rôle : nom du module contenant le script à exécuter : il s'agit du nom du script Python, privé de son extension `".py"`, par exemple :

```
"module": "cwmail",
```

Se reporter aux directives d'écriture des scripts en question.

4.1.13 `"names"`

Présence : requise.

Type : liste de chaînes Unicode en camelCase.

Inclus dans la documentation : oui.

Rôle : liste des noms de commande sous lesquels le script pourra être invoqué par l'utilisateur final. Le premier sera de préférence le plus naturel, c'est lui qui apparaîtra comme nom principal.

```
"names": ["cwMail", "cwEmail", "cwMél", "cwCourrier"],
```

La longueur de ces noms importe peu : au moment de l'indexation, Chewing Word les « apprend », et sera donc capable de les faire apparaître dans la bulle sitôt composé leur préfixe discriminant (à l'instar des prédictions normales).
Inversement, il faut savoir qu'une commande sera reconnue même si elle est suivie de lettres non prévues. Celles-ci forment un préfixe qui sera passé au script, lequel peut soit l'ignorer, soit le traiter comme un argument :

```
"names": ["cwVisiter", "cwVisite", "cwNavigue", "cwButine", "cwSurf"],
```

Ici, la même commande peut être invoquée par `cwSurf`, `cwSurfe` ou `cwSurfer`. Notez qu'en théorie, le premier nom, `"cwVisiter"`, est redondant avec `"cwVisite"` : nous l'avons intégré en tête à seule fin que le nom principal soit un verbe à l'infinitif.

De façon générale, si votre nom de script contient une forme verbale, privilégiez l'infinitif mais autorisez également l'impératif ou le substantif correspondants.

4.1.14 "prefs"

Présence : facultative.
Type : dictionnaire ordonné.
Inclus dans la documentation : oui.
Rôle : répertoire des variables à injecter dans le panneau de Préférences, sous l'onglet Scripts. Remarquez que l'ordre dans lequel vous énumérez les champs dans le fichier sera respecté dans la table du panneau de Préférences. Chaque entrée est un dictionnaire de trois entrées. En voici deux exemples tirés de "fr_cwmail.json" :

```
"cwMail_defaultSignature" : {
  "field": "cwMail - signature",
  "value": "\nEnvoyé avec Chewing Word.",
  "tip": "Copiez ici la signature qui apparaîtra en bas de chaque message, ou
          tapez-la directement (utilisez \\n pour aller à la ligne)."
},
"cwMail_smtp"  : {
  "field": "cwMail - SMTP - serveur",
  "tip": "Googlez « adresses serveurs FAI » pour les paramètres des serveurs
          d'envoi des principaux fournisseurs d'accès à internet."
},
```

La clef (ex. : "cwMail_defaultSignature") est écrite comme la concaténation du nom international de la commande en camelCase, d'un caractère de soulignement, de la partie discriminante de l'identificateur international souhaité, également en camelCase. Les trois champs sont :

"field" : Un libellé localisé donnant le nom du réglage sous une forme plus sympathique. Indiquez le nom principal de la commande, suivi d'un espace, d'un tiret court, d'un espace et du nom discriminant du réglage.

"value" : Le réglage par défaut. Notez que ce champ manque pour "cwMail_smtp", car il a été défini dans le fichier de méta-données international "cwmail.json".

"tip" : Le texte affiché dans la bulle d'aide au survol du champ « valeur » de la table. Notez qu'il est également au format txt2tags, ce qui permet par exemple d'insérer des tableaux (au prix parfois de quelques échappements).

4.1.15 "static"

Présence : facultative.
Type : dictionnaire.
Inclus dans la documentation : non.
Rôle : répertoire des variables statiques, localisables ou non. Les clefs obéissent aux mêmes conventions que celles du champ "prefs". Les valeurs, qui peuvent être d'un type quelconque, ne seront connues que du script en cours d'exécution, sauf en ce qui concerne les méta-données communes du répertoire "cw".

4.1.16 `"type"`

Présence : requise.
Valeur : `"external"`.
Inclus dans la documentation : oui.
Rôle : distingue les scripts externes, internes, et les options des menus. Votre script doit être marqué `"external"` pour fonctionner avec la distribution officielle de Chewing Word. Remarquez que les dossiers des scripts internes (`cwpage`, "cwpreferences", `cwhelp`) contiennent méta-données et ressources, mais pas les modules Python requis (ils font partie intégrante du logiciel lui-même).

4.1.17 `"url"`

Présence : conseillée.
Type : chaîne Unicode.
Inclus dans la documentation : oui.
Rôle : adresse d'un site où trouver des informations complémentaires ou les mises à jour du script.

4.1.18 `"version"`

Présence : conseillée.
Type : chaîne Unicode couverte par l'expression régulière `"\d+(\.\d+){,2}"`.
Inclus dans la documentation : oui.
Rôle : numéro de version du script (distinct du numéro de version de Chewing Word), à partir de 1.0 (valeur par défaut).

4.2 Sélection automatique

Valeurs possibles pour la clef `"autoSelect"`. Notez que les segmentations en paragraphes, phrases ou mots sont les mêmes que celles mises en œuvre lors de la synthèse vocale ou de la sélection par déclencheur.

4.2.1 `"page"`

Texte complet de la page en cours (le contenu des autres pages est actuellement inaccessible par les scripts).

Exemples. `cwbrowse.json`, `cwcopyinto_mac.json`, `cwmail.json`, `cwpastebin.json`, `cwsavevoice.json`, `cwstats.json`.

4.2.2 "paragraph"

Paragraphe, *éventuellement vide*, incluant le curseur, et contrôlé par l'expression régulière localisée `env["constants"]["paragraphRex"]`.

Exemples. `cwadium_mac.json`, `cwfiglet.json`, `cwgooglesearch.json`, `cwimport.json`, `cwkeys_mac.json`, `cwmap.json`, `cwmenu_mac.json`, `cwunicode.json`, `cwweather.json`.

4.2.3 "sentence"

Phrase incluant, ou à défaut précédant le curseur, et contrôlée par l'expression régulière localisée `env["constants"]["sentenceRex"]`.

Exemples. Aucun.

4.2.4 "word"

Mot incluant, ou à défaut précédant le curseur, et contrôlé par l'expression régulière localisée `env["constants"]["wordRex"]`.

Exemple. `cwsynonyms.json`.

4.2.5 "empty"

Réservé aux scripts qui n'ont jamais, ou que rarement besoin d'argument textuel en entrée.

- Dans la première catégorie, on trouve `cwdate`, qui se borne à insérer la date du jour, ou `cwhelp`, dont l'argument éventuel se passe en suffixe. De tels scripts ignoreront tout simplement l'éventuel argument textuel explicite défini par sélection manuelle.
- Dans la deuxième catégorie, il y a `cwpage`, qui permet principalement de changer de page (dont le nom est là encore passé en suffixe), mais peut aussi servir au transfert de texte entre pages : il faudra alors sélectionner ce texte manuellement avant de composer la commande.

Exemples. `cwdate.json`, `cwenvironment.json`, `cwhelp.json`, `cwdraft.json`, `cwfont.json`, `cwpage.json`, `cwpages.json`, `cwpreferences.json`.

4.3 Environnement du script : contexte de l'éditeur

Les clefs suivantes se trouvent dans le dictionnaire `env["context"]`.
Pour la définition précise de ces zones, on relira la référence de l'auto-sélection à la section précédente.

4.3.1 `"page"` et `"pageCursor"`

4.3.2 `"paragraph"` et `"paragraphCursor"`

4.3.3 `"sentence"` et `"sentenceCursor"`

4.3.4 `"word"` et `"wordCursor"`

4.3.5 `"selection"` et `"selectionCursor"`

Sélection définie, soit manuellement par l'utilisateur, soit automatiquement par Chewing Word en fonction de la valeur de la méta-donnée `"autoSelect"` du script en cours, et qui dans ce cas coïncidera avec l'une des quatre sélections ci-dessus.

4.4 Environnement du script : méta-données complémentaires

Les clefs suivantes se trouvent dans le dictionnaire `env["meta"]`.

4.4.1 `"actualPoorName"`

Le nom sous lequel la commande a été effectivement invoquée, dans sa variante appauvrie (p. ex., en français, plongement sur les 26 lettres de l'alphabet latin, sans diacritiques et en majuscules).

4.4.2 `"actualRichName"`

Le nom sous lequel la commande a été effectivement invoquée.

4.4.3 `"antefix"`

Les caractères non blancs ni alphabétiques collés à gauche du nom de la commande.

4.4.4 `"poorSuffix"`

Le suffixe du nom de la commande, dans sa variante appauvrie.

4.4.5 `"richSuffix"`

Le suffixe du nom de la commande, dans sa variante « aérée » par insertion d'un espace à chaque changement de casse.

4.4.6 `"identifier"`

Premier nom de la liste `"names"` avant localisation.

4.4.7 `"name"`

Premier nom de la liste `"names"`.

4.4.8 `"poorNames"`

Liste des noms de `"names"` réécrits à l'aide des lettres apparaissant sur les touches du cadran.

4.4.9 `"path"`

Chemin du script dans l'arborescence des fichiers.

4.4.10 `"platform"`

Une valeur parmi `"win"`, `"mac"`, `"lin"` et `"all"`, déduite du nom du fichier de métadonnées selon la technique présentée plus haut.

4.5 Directives de post-traitement d'un script

4.5.1 `"append"`

Insertion après le curseur. Si la sélection est non vide, cette opération équivaut à un appui sur la flèche droite (désélection) suivi d'une saisie de l'argument.

Argument. Chaîne Unicode à insérer.

Exemples. `cwadium_mac`, `cwconsole`, `cwdate`, `cwip`.

4.5.2 `"clipboard"`

Place son argument dans le presse-papiers.

Argument. Chaîne Unicode.

Exemples. Aucun.

4.5.3 `"deleteChars"`

Effacement de caractères. Si la sélection est non vide, cette opération équivaut à un appui sur la flèche gauche (désélection) suivi du nombre d'appuis spécifiés sur la touche « DEL ». Pour effacer la sélection, utiliser plutôt la directive `("replace","")`.

Argument. Nombre de caractères à effacer à gauche du curseur.

Exemples. Aucun.

4.5.4 `"htmlBox"`

Dialogue présentant une vue HTML, une vue textuelle, un bouton permettant de basculer de l'une à l'autre, deux boutons d'ascenseur, un bouton de copie dans le presse-papiers (pour la vue textuelle), un bouton « Visiter » (éventuellement) et un bouton « OK ».

Argument. Dictionnaire contenant les clefs suivantes :

`"label"` (requis) Libellé de la fenêtre de dialogue, en HTML.

`"html"` (facultatif) Chaîne HTML définissant le contenu de la vue éponyme. À défaut, le contenu est téléchargé depuis l'adresse `"url"`.

`"url"` (facultatif) Adresse web. Elle supplée à l'absence du champ `"text"` ou `"html"` en définissant le contenu de la vue textuelle ou HTML (respectivement). Par exemple, en supposant que `"url"` vaut `"http://chewingword.wikidot.com/"`, si `"text"` est manquant, la vue textuelle contiendra cette même adresse ; si `"html"` est vide, la vue HTML sera la page située à cette adresse. Dans tous les cas, la présence d'un champ `"url"` déclenche l'apparition d'un bouton « Visiter » dont l'appui ouvrira l'adresse en question dans le navigateur par défaut.

`"text"` (facultatif) Chaîne définissant le contenu de la vue textuelle. À défaut, c'est l'adresse `"url"` qui apparaîtra. Si le champ `"url"` est également manquant, le contenu textuel résultera de la conversion en texte brut du contenu HTML (le champ `"html"` doit donc exister dans ce cas).

`"width"` (facultatif) Largeur par défaut de la fenêtre. Les redimensionnements opérés par l'utilisateur sont mémorisés pour la suite de la session et pour les sessions suivantes.

"height" (facultatif) Hauteur par défaut de la fenêtre. *Idem* pour les redimensionnements.

On peut synthétiser les différents comportements attendus en fonction de la présence des champs "html", "text" et "url" dans la table ci-dessous (par commodité, on nomme box le dictionnaire, getPage la fonction téléchargeant une page et plainText la conversion en texte brut) :

"html"	"text"	"url"	vue HTML	vue textuelle
non	?	non	*illégal*	*illégal*
non	non	oui	getPage(box["url"])	box["url"]
non	oui	oui	getPage(box["url"])	box["text"]
oui	non	non	box["html"]	plainText(box["html"])
oui	non	oui	box["html"]	box["url"]
oui	oui	?	box["html"]	box["text"]

Exemples. cwgooglesearch, cwmap, cwpastebin, cwweather.

4.5.5 "listBox"

Dialogue permettant de choisir parmi une liste de chaînes Unicode, et comportant deux boutons de déplacement, un bouton « OK » et un bouton « Annuler ». La fermeture par « OK » substitue la chaîne choisie au texte sélectionné dans l'éditeur.

Argument. Dictionnaire contenant les clefs suivantes :

"label" (requis) Libellé de la fenêtre de dialogue, en HTML.

"items" (requis) Liste de chaînes Unicode parmi lesquelles l'utilisateur devra opérer son choix.

"current" (facultatif) Numéro de la chaîne initialement sélectionnée. Numérotation à partir de 0 (valeur par défaut).

Exemples. fr_cwsynonyms.

4.5.6 "messageBox"

Dialogue présentant un simple message d'information ou d'erreur, ainsi qu'un bouton « OK ».

Argument. Libellé du message, en HTML.

Exemples. cwgooglesearch, cwip, cwmail, cwpastebin, cwsavevoice, cwwallpaper, fr_cwsynonyms.

4.5.7 "moveCursor"

Déplacement relatif du curseur. Si la sélection est non vide, cette opération équivaut à un appui sur la flèche **droite** (désélection) suivi du nombre spécifié d'appuis sur la flèche gauche (s'il est négatif) ou droite (s'il est positif). La directive ("moveCursor",0) est employée dans de nombreux scripts simplement pour amener le curseur en fin de sélection.

Exemples. cwbrowse, cwcopyinto_mac, cwgooglesearch, cwhangman, cwmail, cwmap, cwpreferences, cwsavevoice, cwstats, cwsudoku, cwweather.

4.5.8 "remember"

Quand Chewing Word termine, il met à jour l'index personnel des mots transcrits par l'utilisateur. Des micro-adaptations de l'arbre lexicographique ont déjà été opérées au fur et à mesure de la session, mais à la fermeture le travail est repris à zéro afin d'éviter de garder mémoire des erreurs corrigées par l'utilisateur, des remords, des effacements, etc. : un index est alors établi par dépouillement de tout le texte présent sur les différentes pages (sauf les brouillons). Or, il se peut que certaines commandes effacent une sélection qui mériterait pourtant d'être prise en compte : par exemple, cwMail, après envoi et archivage d'un message, fait disparaître celui-ci de l'éditeur. C'est ici qu'intervient la directive "remember" : elle permet d'indexer ce contenu en anticipation de la fermeture.

Argument. Texte à indexer.

Exemples. cwmail.

4.5.9 "replace"

Remplacement de la sélection par le texte donné en argument. Si la sélection est vide, équivaut à une simple insertion. Utiliser ("replace","") pour effacer la sélection.

Exemples. cwadium_mac, cwfiglet, cwhangman, cwkeys_mac, cwmail, cwmenu_mac, cwpreferences, cwunicode, cwsudoku, cwwallpaper.

4.5.10 `"select"`

Sélection dans le texte d'une zone spécifiée par un indice de début et un indice de fin (en absolu). S'ils sont égaux, déplace simplement le curseur à cette position.

Argument. Un couple d'indices de début et de fin.

Exemples. Aucun.

4.5.11 `"setSpaceScript"`

Chewing Word offre la possibilité de répéter la dernière commande par un simple clic long sur l'Espace (déclencheur). Dans quelques cas, cette répétition n'a aucun intérêt : par exemple, si l'on vient de changer de police avec `cwPoliceTimesNewRoman`, il est inutile de le redemander ! Par contre, ce qui serait utile serait d'annuler le changement : c'est ce que permet (entre autres) la directive `"setSpaceScript"`. Elle remplace la dernière commande disponible sur l'Espace par la chaîne de votre choix. Dans notre exemple, en supposant que la police précédente était « Courier », `cwPolice` assignerait à l'Espace `cwPoliceCourier`.

Argument. Une chaîne Unicode quelconque.

Exemples. `cwfont`.

4.5.12 `"textBox"`

Dialogue présentant une vue textuelle, un bouton permettant de couper les lignes trop longues (*soft wrap*), deux boutons d'ascenseur, un bouton de copie dans le presse-papiers, un bouton « Fermer » et un bouton « Remplacer » (qui ferme le dialogue en substituant le texte à la sélection en cours).

La directive `"textBox"` peut être vue comme un `"replace"` sécurisé. Elle joue le rôle d'un « sas » qui laisse à l'utilisateur décider s'il veut ou non intégrer dans l'éditeur un texte renvoyé par le script.

Argument. Dictionnaire contenant les clefs suivantes :

`"label"` (requis) Libellé de la fenêtre de dialogue, en HTML.

`"text"` (requis) Chaîne Unicode à afficher.

`"width"` (facultatif) Largeur par défaut de la fenêtre. Les redimensionnements opérés par l'utilisateur sont mémorisés pour la suite de la session et pour les sessions suivantes.

"height" (facultatif) Hauteur par défaut de la fenêtre. *Idem* pour les redimensionnements.

Exemples. `cwenvironment`, `cwimport`.

4.5.13 "tooltip"

Affichage d'un message dans une bulle d'aide au voisinage du pointeur.

Argument. Chaîne Unicode.

Exemples. `cwfont`, `cwstats`.

4.5.14 "statusBar"

Affichage d'un message dans la barre de statut.

Argument. Message.

Exemples. Aucun.

4.5.15 "updatePrefs"

Il peut être intéressant d'utiliser un script pour modifier un ou plusieurs réglages du profil en cours. La commande interne `cwPréférences` répond à ce besoin de façon générale, qui plus est en effectuant un contrôle de validité des valeurs modifiées. Mais on peut fluidifier le processus pour certains réglages bien précis.

Il faut savoir que, dans la version actuelle du logiciel, le dictionnaire associé à la clef `env["prefs"]` n'est autre qu'une *référence* au dictionnaire de préférences utilisé en interne. Cela signifie que toute modification de celui-ci se répercutera dans l'environnement de travail de Chewing Word.

La directive `("updatePrefs", None)` garantit que la modification en question sera correctement prise en compte, ceci en mettant à jour toutes les stratégies associées. Pour prendre une image ferroviaire, l'application d'une *stratégie* consiste à manœuvrer un aiguillage le plus tôt possible en amont, de façon à n'avoir plus de décision à prendre à l'arrivée du train. Chaque mise à jour des préférences redessine complètement le réseau d'exécution interne de Chewing Word. À défaut, non seulement le train n'ira pas où l'on veut, mais il risque de dérailler pour cause d'incompatibilité entre certaines de ses caractéristiques (les valeurs modifiées) et les voies empruntées.

Notez toutefois que l'emploi de cette directive est inutile dans le cas où les entrées modifiées appartiennent toutes à des scripts (*i.e.*, sont listées dans l'onglet « Scripts » du panneau de Préférences) : comme les scripts sont complètement découplés du programme principal, et ne sont connus qu'au moment de leur exécution, leurs aiguillages ne sont pas manœuvrables à l'avance, mais seulement au passage du train.

Argument. Requis pour la cohérence du typage, il n'en est pas moins ignoré. La convention est de mettre `None`.

Exemples. `cwfont`, `cwhangman`, `cwsudoku`.

4.6 Modules disponibles

Les bibliothèques suivantes sont disponibles à l'importation.

4.6.1 Bibliothèque commune à tous les scripts

Le fichier `"scriptgoodies.py"` se trouve au même niveau que les dossiers des différents scripts. Pour l'instant, il ne contient que quelques fonctions, mais est destiné à s'étoffer de tout ce qui pourra se révéler utile à plusieurs scripts ou tests.

`getArchivePath` Renvoie le chemin complet d'un fichier horodaté dans le dossier archivant les transcriptions du jour. Les paramètres sont `env`, l'environnement habituel, et `name`, une chaîne de format comportant deux `"%s"` : au premier sera susbstitué l'heure formatée selon la chaîne `env["static"]["cw_archiveTimeFormat"]` ; au second, un suffixe numérique (1, 2, 3...) au cas où un fichier de même nom est déjà présent à cet endroit (p. ex. si deux mails sont envoyés dans la même minute). Exemples : `cwmail`, `cwsavevoice`.

`getTestingEnvironment` Cf. Construction à partir des méta-données.

`setTemporaryPrefs` Met à jour les préférences actuelles à l'aide du dictionnaire donné en argument. Sauvegarde au préalable les valeurs affectées dans l'environnement privé `env["private"]["cwXxx_backupPrefs"]` où `Xxx` est le nom de la commande appelante. Ne pas oublier d'ajouter après appel la directive de mise à jour des préférences.

`restoreBackupPrefs` Rétablit les préférences affectées par la commande appelante. Là encore, ajouter la directive de mise à jour.

4.6.2 Modules internes à l'application

Ils peuvent être consultés en clair en téléchargeant les sources (cf. distribution pour Linux).

Le module `assistivedialog` est utile pour afficher des fenêtres de dialogues accessibles sans terminer le script. Par exemple, un message d'avertissement peut apparaître au cours de l'exécution du script `cwsudoku` :

```
assistiveDialog.warning(env["static"]["cwSudoku_exhaustedMessage"] % level)
```

Exemples. `cwhangman`, `cwsudoku`.

4.6.3 Bibliothèques du langage

`__builtin__`, `__future__`, `_abcoll`, `_bisect`, `_codecs`, `_collections`, `_csv`, `_functools`, `_hashlib`, `_heapq`, `_io`, `_json`, `_locale`, `_random`, `_scproxy`, `_sha256`, `_sha512`, `_socket`, `_sre`, `_ssl`, `_struct`, `_warnings`, `_weakref`, `_weakrefset`, `abc`, `array`, `base64`, `binascii`, `bisect`, `cgi`, `chunk`, `codecs`, `collections`, `copy`, `copy_reg`, `cStringIO`, `csv`, `datetime`, `decimal`, `email`, `encodings`, `encodings.__builtin__`, `encodings.aliases`, `encodings.ascii`, `encodings.binascii`, `encodings.codecs`, `encodings.encodings`, `encodings.hex_codec`, `encodings.utf_8`, `errno`, `exceptions`, `fcntl`, `functools`, `gc`, `genericpath`, `getopt`, `gettext`, `gzip`, `hashlib`, `heapq`, `hmac`, `httplib`, `io`, `itertools`, `json`, `json._json`, `json.decoder`, `json.encoder`, `json.json`, `json.re`, `json.scanner`, `json.struct`, `json.sys`, `keyword`, `linecache`, `locale`, `marshal`, `math`, `mimetools`, `numbers`, `operator`, `os`, `os.path`, `pickle`, `pipes`, `platform`, `plistlib`, `posix`, `posixpath`, `pyexpat`, `pyexpat.errors`, `pyexpat.model`, `PyQt4`, `PyQt4.phonon`, `PyQt4.QtCore`, `PyQt4.QtGui`, `PyQt4.QtNetwork`, `PyQt4.QtWebKit`, `quopri`, `random`, `re`, `rfc822`, `select`, `shlex`, `shutil`, `signal`, `sip`, `site`, `sitecustomize`, `smtplib`, `socket`, `sre_compile`, `sre_constants`, `sre_parse`, `ssl`, `stat`, `string`, `strop`, `struct`, `subprocess`, `sys`, `sysconfig`, `tempfile`, `textwrap`, `thread`, `threading`, `time`, `traceback`, `types`, `unicodedata`, `urllib`, `urllib2`, `urlparse`, `UserDict`, `warnings`, `wave`, `weakref`, `webbrowser`, `xml`, `xml.parsers`, `xml.parsers.expat`, `xml.parsers.pyexpat`, `zipimport`, `zlib`.

Vous pouvez utiliser un module absent de cette liste en mettant dans `cw_pythonPath` (onglet « Scripts » du panneau de Préférences) le chemin d'une bibliothèque Python le contenant. Si vous souhaitez distribuer votre script, vous pouvez sans attendre le livrer avec une copie du module manquant. Contactez toutefois l'auteur de Chewing Word pour lui demander d'inclure celui-ci dans une prochaine distribution.

Notez bien que vous disposez de la plupart des modules de Qt, en particulier QtCore et QtGui.

Exemples. `cwwallpaper`.

5 Conclusion : « Read the code, Luke ! »

Ce guide n'est certainement ni complet, ni exempt d'erreurs. Nous l'améliorerons au gré des remarques et des questions de ses lecteurs. Cependant, il ne saurait vous apprendre à lui seul comment écrire des scripts d'extension de Chewing Word. Il doit être complété :

- par une bonne familiarité d'usage avec les scripts existants ;
- par la lecture de leur documentation (commande `cwAide`) ;
- par l'étude de leur code et leurs méta-données.

Au niveau interne, les scripts d'extension sont pris en charge par deux modules : la méthode `loadScripts` du programme principal `"chewing.py"` s'occupe de leur indexation ; le module `"scriptManager.py"` gère le pré-traitement, l'appel et le post-traitement. Leur analyse pourra éclairer de nombreux points de détail que nous n'avons pas jugé indispensable d'évoquer ici.