Manuel français du runner

La commande `run` permet d'installer une configuration de travail quelconque (ouverture de dossier/fichier, incrémentation de version, surveillance de fichier, code quelconque, etc.).

Installation d'une configuration de travail définie

Une fois la création d'une nouvelle configuration définie, on peut la lancer, soit en jouant la commande `run` puis en la choisissant dans le menu s'affichant, soit en lançant la commande `run` avec l'identifiant de cette configuration

```
run id-configuration
```

> Cf. à quoi correspond l'identifiant de la configuration de travail

---

Création de la nouvelle configuration de travail

Jouer dans un Terminal (n'importe où) :

```
> run
```

... choisir "Nouvelle configuration de travail" et suivre la procédure.

Aide

Pour obtenir de l'aide, on peut jouer `run manuel` ou `run open` puis choisir "Manuel" pour ouvrir ce manuel (ajouter l'option `-dev` pour ouvrir la version éditable qui peut être modifiée dans Typora.

Lancer une configuration de travail

Pour lancer une configuration de travail, i.e. ouvrir les dossiers, lancer un fichier, rejoindre une URL, etc. il suffit de jouer la commande **`run`** en console et de choisir la configuration de travail voulue.

> Note : la dernière configuration de travail est toujours au-dessus.

On peut aussi lancer cette configuration directement grâce à son identifiant de configuration. Par exemple :

```
> run nom-fichier-sans-extension
```

Rejouer seulement certaines étapes d'une configuration de travail

Parfois, on peut avoir à ne relancer ou à ne lancer que certaines étapes d'une configuration de travail (par exemple un fichier de notes). Dans ce cas, il suffit de lancer la commande `run` avec l'option `-c/--choose`. Par exemple :

```
> run ma-config -c
```

Avec la commande ci-dessous, ***Run*** passera en revue toutes les étapes en demandant s'il faut les jouer.

> Noter que la touche `Return` permet alors de passer une étape et que pour en jouer une autre, il faut explicitement entrer `y` ou `yes`.

Configuration d'un travail

Nom humain de la configuration de travail

```yaml
---
name: "Nom du travail"
# ...
```

> C'est ce nom qui apparait dans la liste pour choisir un travail.

Dossier par défaut de la configuration

Si ce dossier est défini, il sera utilisé dès qu'un chemin d'accès ne sera pas trouvé tel quel.

```yaml
---
# ...
folder: "path/to/existing/folder"
# ...
```

Définition des étapes de la configuration

Tout ce qu'il y a à installer se trouve dans la propriété `setup` du fichier `YAML` de la configuration de travail.

> Pour ouvrir ce manuel, jouer `run -h/--help`

Ce paramètre contient les étapes (`steps`) à jouer pour obtenir une installation complète.

Configuration d'une étape

Une étape est avant tout définie par son `:type` (cf. ci-dessous tous les types d'étape). C'est la première chose à définir.

```yaml
:setup:
  - type: <type de l'étape>
```

Chemin d'accès dans une étape (dossier par défaut)

De nombreuses étapes nécessitent de définir un `:path`, chemin d'accès à un fichier ou un dossier. L'écriture de ce path peut être absolu, mais il peut également être simplifié en définissant un dossier par défaut dans la propriété générale `:folder` de la configuration du travail.

Par exemple :

```yaml
---
:name: "Configuration du travail à faire"
:folder: "path/absolu/to/folder/default
:setup:
  - type: open
    name: "Ouverture du dossier par défaut
    path: "."
    description: "Cette étape ouvrira le dossier :folder"
```

Étapes optionnelles

Par défaut, toutes les étapes sont jouées. Mais une étape peut être optionnelle, dans ce cas, le programme demande s'il faut la jouer avant de le faire ou non.

Pour indiquer qu'une étape est optionnelle, on ajoute `opt: true` à sa configuration. Pour pouvoir être désignée, il lui faut également un `:name`. Si elle contient aussi une `:description`, celle-ci sera discrètement affichée pour préciser l'étape.

Donc une étape optionnelle peut ressembler à :

```yaml
:setpup:
  - type: open
    path: path/mon/fichier
    opt: true
    name: "Ouverture du fichier d'aide"
    description: |
      Ce fichier contient des renseignements qui peuvent
      aider à comprendre l'étape et savoir ce qu'elle va
      faire précisément.
```

Tous le types d'étapes

Ouverture d'un fichier

Pour ouvrir un fichier spécifique :

```yaml
:setup:
  - type: open
    path: path/to/file # peut être relatif à :folder
    app: application optionnelle # sinon l'application par défaut
    bounds: [top, left, width, height]
```

Ouverture d'un dossier

```
:setup:
  - type: open
    path: path/to/folder # peut être relatif à :folder
    bounds: [top, left, width, height]
```

Ouverture d'un dossier dans l'IDE

On peut ouvrir un dossier dans l'IDE courant qui doit être défini dans `./lib/constants.rb` dans la constante `IDE` et définir la commande à utiliser dans `IDE_CMD`.

```
IDE = "Nom de l'IDE"
IDE_CMD = "<commande> '%s'" # %s sera remplacé par le :path défini
```

Par exemple :

```
IDE = "Sublime Text"
IDE_CMD = "subl -n '%s'"
```

Définition dans le fichier setup du travail :

```
setup:
  - type: open
    app:  IDE
    path: path/to/folder
```

Rejoindre une URL

```
setup:
  - type: url
    path: https://path/to/site
    # Valeurs optionnelles
    name: "Nom précis de l'étape"
    app: Firefox # ou le navigateur par défaut
    args: {key: value, key2: value, ...} # query string
    opt: true # pour en faire une étape optionnelle
```

> Les arguments (`args`) peuvent aussi être fournis sous forme de vraie table :

```
---
# ...
setup:
  - type: untype
    # ...
    args:
      key1: valuekey1
      key2: valuekey2
      # etc.
```

Jouer un script utilitaires

Ce script doit se trouver dans le dossier `scripts` du dossier `Run` ou être spécifié par chemin d'accès complet. Il reçoit toujours les arguments définis par `:args` comme un json-string, donc comme une table avec des clés string. Si `args` n'est pas défini, il appelle le script sans argument.

```
:setup:
  - type: script
    path: mon_script
    args: '{"key":"value", "key2":"value2",...}' # format JSON ou Hash
```

Jouer du code à la volée

```
:setup:
  - type: code
    lang: ruby # le langage ('ruby', 'bash', 'python', 'applescript')
    cmd: "puts('le code ruby à evaluer')"
```

> Vous trouverez en annexe la liste complète des script

Lexique

Identifiant de configuration

L'identifiant de la configuration (souvent symbolisé par `<id-configuraion>`) correspond à l'affixe du fichier qui la définit dans le dossier des travaux, donc **le nom du fichier du travail sans l'extension**.

Dossier des travaux

Dossier contenant la fiche de la définition précise de chaque configuration de travail. On le trouve dans le dossier de l'application ***Run***, à la racine, avec le nom `_travaux_`.

La commande `run open`, en choisissant l'item "Ouvrir le dossier des travaux", permet de l'ouvrir dans l'éditeur

---

Annexe

Liste complète des scripts

Suivre un fichier avec `backup`

Le script `backup` permet de suivre un fichier et d'en faire un backup de sécurité chaque fois qu'il est enregistré.

> Ce script a été inauguré suite à la perte de plusieurs heures de travail avec la version 2 de Publisher, fortement instable (peut-être avec les anciens fichiers).

Pour l'utiliser dans un set-up :

```
---
# ...
folder: /path/to/mon/dossier
setup:
  - type: script
    path: 'backup'
    args:
      file: /v([0-9]+)\.([0-9]+)\.([0-9]+)/BOOK\.afpub/
    description: "suivre le fichier principal de développement"
```

La donnée la plus délicate, si elle doit varier, concerne l'argument à transmettre, qui doit spécifier le fichier à suivre. Ici, par exemple, il se trouve dans un dossier versionné.

Upgrader la fichier d'un fichier

Le script `upgrade_version` permet de faire monter le fichier/dossier spécifié à la version supérieure (tout en faisant une sauvegarde de la version courante.

Pour l'utilliser dans le script dans un set-up :

```
---
# ...
folder: /path/to/mon/dossier
setup:
  type: script
  path: upgrade_version
  args: {format: "v([0-9]+)\.([0-9]+)\.([0-9]+)"}
  description: Pour passer à la version suivante.
```

> Noter que, ici, la donnée `format` doit absolument être un `String`. Dans le cas contraire, une exception sera levée.

Utilisation de Run avec MyTaches

On peut lancer une configuration de travail depuis une tâche de l'application `MyTaches`. Il suffit de définir dans la tâche le code à exécuter et de mettre la valeur de lancement d'une configuration :

```
code à exécuter : 'run <id-configuration>'
```

> Cf. à quoi correspond l'identifiant de la configuration de travail