

# Args Parser

## GitHub repo

Nous avons tous plus ou moins déjà été amenés à parser des arguments en entrée d'un programme.

L'objectif de ce TP est de fournir un utilitaire sous forme de classe, permettant de faire ce travail à notre place.

Les arguments passés au programme consistent en des **flags** et des **values**.

- Un flag contiendra toujours un signe moins (-) suivi d'un caractère (**[a-zA-Z]**).
- Un flag doit avoir **zero** ou **une value** associée.

Vous devez écrire un **parser** pour ce genre d'arguments.

Ce parser doit prendre un **schéma** détaillant les arguments auxquels le programme peut s'attendre.

Le schéma spécifie **précisément** les flags possibles ainsi que le **type associé**.

Une fois le schéma spécifié, l'utilisateur doit pouvoir **passer une liste d'arguments** au parser.

Le parser se charge alors de **vérifier** qu'il peut parser toutes les valeurs (selon le schéma spécifié).

Si c'est OK, l'utilisateur doit alors pouvoir **récupérer** les valeurs du parser en utilisant les **noms** des flags.

Si l'utilisateur essaie de récupérer la valeur pour un **flag inexistant**, une valeur doit être renvoyée par défaut

- false pour un boolean
- 0 pour un number
- "" (chaîne vide) pour une string

---

Un exemple d'utilisation est donné en Typescript :

```
// Schema explanations:
// -> char   - Boolean arg
// -> char#   - Number arg
// -> char*   - String arg

const schema = 'd,p#,h*';
try {
  const args = new Args(schema);
  args.parse(`-d -p 42 -h 'Vincent Vega'`);
```

```
const detach = args.getBoolean('d');
const port = args.getNumber('p');
const hero = args.getString('h');
executeApplication(detach, port, hero);
} catch (e) {
  console.error(`Parse error: ${e.message}`);
}

const executeApplication = (d, p, h) => {
  console.log(`Application running - detached (${d}), port: (${p}), hero is (${h})`);
}
```

---

Revision #1

Created 9 May 2022 21:24:15 by Noé Larrieu-Lacoste

Updated 9 May 2022 21:26:22 by Noé Larrieu-Lacoste