User guide

Jump to bottom

Grégory Mantelet edited this page on 29 Jun 2018 · 4 revisions

The below documentation has been written for the version 1.1 of the UCIDY library.

Introduction

This library has no dependency and does not required any Internet connection.

Java 1.7 or newer is required.

It can be used as:

- a standalone (in command line)
- or integrated within another Java application.

Definitions

- A UCD word is said **valid** if *its syntax is correct* according to the IVOA standard (An IVOA Standard for Unified Content Descriptors - Version 1.1). In other words: it must contain only letters, digits, underscores or hyphens, and it can start only with a letter or a digit.
- A UCD word is said **recognised** if it *can be found in the list provided to the UCD parser*.
- A UCD word is said **recommended** if it is one of the UCD1+ words validated by the IVOA (The UCD1+ controlled vocabulary).
- A **recognised** word is always **valid**.
- By definition, a **recommended** word is always **valid** and **recognised**.

- A UCD is said **fully valid** if all its words are **recognised** and that the order of the words is conform to their respective syntax code (e.g. `meta.main` must always be used in second position).

Standalone usage

For the moment, in its standalone version, a small interactive program lets validate any given UCD. No other action is possible and so no parameter is required. Once started, the user will have to write a UCD string and hit `Enter` to see a validation report.

The command

```
java -jar ucidy-{version}.jar
```

or

```
java -cp ucidy-{version}.jar ari.ucidy.UCDParser
```

Output description

Once a UCD has been entered and submitted, a validation report is returned. It will list the following information:

Output	Possible values	Description
`All words valid?`	`true` / `false`	It tells whether all words composing the given UCD are **syntactically valid**.
`All words recognised?`	`true` / `false`	It tells whether all words are known. In the case of standalone usage, this will *always* mean that the word is validated by the IVOA ; it is then a synonym of the following piece of information.
`All words recommended?`	`true` / `false`	It tells whether all words are validated by the IVOA.

Output	Possible values	Description
`UCD fully valid?`	`true` / `false`	`true` only if all words are recognised (and so also valid) and that their order is syntactically correct (e.g. `meta.main` must always be in second position).
`REASON`		list of all issues preventing your UCD to be *fully valid* ; in brief, a list of errors.
`CORRECTION SUGGESTION`		It suggests a valid UCD from the given one after some automatic corrections to the errors listed above (e.g. re-order words in function of their syntax code, replace by the closest match in case of typo, suggestion a replacement for deprecated words, …) ***WARNING!*** *This is only an automatic suggestion. Before any hasty copy/paste, it should be carefully checked by a human to ensure that the original meaning of the UCD is kept.*
`ADVICE for improvement of your UCD`		These are just hints/advice aiming to improve the readability or precision of your UCD.

Integrated within another Java application

Note: *the following part describes some library features by presenting partially its API. As an additional help, you can use the generated UML diagram and Javadoc.*

Once the jar `ucidy-{version}.jar` is included in the classpath, you can use the following classes in function of your need:

1. To parse a UCD: `UCDParser`
2. To manage the list of all known UCD words and/or to search for UCD words: `UCDWordList`
3. To manage the list of all deprecated UCD words: `DeprecatedUCDWordList`

`UCDParser`

Static function

The function you will use most of the time is the static function `parseUCD(String): UCD`. Since it is a static function, you do need to create any instance of `UCDParser` and it is already able to resolve any UCD1+ word validated by the IVOA and to detect deprecated UCD1+ words.

The list of *UCD1+ words validated by the IVOA* is loaded from the resource file: `ucd1p-words.txt`. This file is provided by the CDS at http://cdsweb.u-strasbg.fr/UCD/ucd1p-words.txt.

The list of *deprecated UCD1+ words* is loaded from the resource file: `ucd1p-deprecated.txt`. This file is also provided by the CDS at http://cdsweb.u-strasbg.fr/UCD/ucd1p-deprecated.txt.

The class `UCD`

Whatever if the given UCD is correct or not, this function will always return an object of type `UCD`. As in the standalone execution, thanks to its attributes you will be able to know whether:

- the whole UCD is valid : `fullyValid`
- all words are valid : `allValid`
- all words are recognised : `allRecognised`
- all words are recommended : `allRecommended`.

With the function `getErrors()`, `getAdvice()` and `getSuggestion()` you will also get all the other explanations provided by standalone execution output.

The class `UCDWord`

And obviously, it is possible to access all the words composing the parsed UCD. In addition of the word itself, `UCDWord` let's you get the specified namespace (by default: 'ivoa' if none is specified), and the syntax code and description if the word is recognised.

`UCDWordList`

`UCDParser` has also a non-static function `parse(String): UCD` doing exactly the same thing as the static one. The only difference is that you must create a new instance of `UCDParser` to use it. By doing that, you will have the possibility to provide your own list of UCD words through an object of type `UCDWordList`.

Creating its own UCD words list

This goal can be merely achieved thanks to the following functions:

add(UCDWord): boolean
Add the given word. If the word is not valid, is not flagged as recognised or already exists (case insensitive and namespace ignored), the word will not be added and `false` will be returned.

addAll(Collection<UCDWord>): int **or** *addAll(UCDWord[]): int*
Add all given words. This function returns the number of added words (see `add(UCDWord): boolean` for the conditions).

addAll(File): int, *addAll(File, boolean): int*, *addAll(InputStream, boolean): int* **or** *addAll(Reader, boolean): int*
Parse a Pipe Separated-Value file following the same columns as the list provided by the IVOA: http://cdsweb.u-strasbg.fr/UCD/ucd1p-words.txt. This function returns the number of added words (see `add(UCDWord): boolean` for the conditions). If the syntax of a file line is incorrect, an error will be displayed on the standard error stream (i.e. stderr). If a word can not be added for any reason, a warning will be displayed on the standard output stream (i.e. stdout) with a line number indication.

remove(String): UCDWord
Search for the given UCD word (case insensitive and namespace ignored) and remove it. The removed word is returned or `null` if none can be found.

clear()`
Remove all UCD words of the list.

Searching for words

This class contains all the research facilities of the *ucidy library*. By default, all words are sorted by ascending alphabetic order case insensitively and while ignoring namespace prefixes.

Why ignoring namespace prefixes? *Because a single UCD word should have a* ***precise***, ***unique*** *and* ***explicit*** *meaning. Allowing two UCD words identical but with a different namespace would be too much ambiguous for users and machines.*

The following functions lets you search a word in different ways:

get(String): UCDWord **or** *get(String, boolean): UCDWord*
These are to perform an **exact match** research (case insensitive). With the first one the namespaces will be always ignored. With the second one, you can choose by setting the second parameter to the desired value (`true` to also check the namespace). If no word is found, `null` will be returned.

startingWith(String): SortedSet<UCDWord>

All UCD words starting with the given string will returned in a sorted set. Again, comparisons are case insensitive and the namespaces are ignored. If no word can be found, the given collection will be empty.

`getClosest(String): UCDWord[]`

This function is a permissive version of `get(String…): UCDWord`. Indeed, the given word may contain typographical errors (i.e. missing/additional/wrong letters). Then, all closed words (i.e. matches having an "acceptable" score) will be returned in a sorted array. If no word can be found, the returned array will be empty.

`search(String): Set<UCDWord>`

Not yet implemented! This function lets search UCD words using their description thanks to the keywords given in parameter.

DeprecatedUCDWordList

This class extends UCDWordList. On the contrary to UCDWordList, this list is aimed to contain only deprecated UCD words.

These words are valid, but can never be recognised or recommended. They may have no syntax code and no description, but they must have a `suggestedReplacement`. This value is the UCD substring to substitute the deprecated UCD word with. *The function `add(UCDWord): boolean` has been overwritten in order to ensure these conditions.*

The functions `addAll(File): int`, `addAll(File, boolean): int`, `addAll(InputStream, boolean): int` or `addAll(Reader, boolean): int` have been modified so that adapting the input parsing to the format of `ucd1p-deprecated.txt`. In this file, lines can be either a comment (i.e. starting with #) or 2 strings separated by a space:

1. the deprecated UCD word *(it must have a valid syntax)*
2. the UCD substring replacing the deprecated word *(all words of this substring must be recognised)*

In order to ensure the *recognised* status of all words in the UCD substrings, this class must be created with an instance of UCDWordList containing recognised words (so, not an instance of DeprecatedUCDWordList).

Clone this wiki locally

```
https://github.com/gmantele/ucidy.wiki.git
```