**ruche** is a package manager for Windows. It’s designed to help developers to implement a clean and portable development environment.

This guide is divided in two parts: The *User Documentation* in which you will find information and tutorials about the installation and the usage of **ruche**, and the *Developer Documentation* which contains an API reference and some technical information.

- *User Documentation*
- *Developer Documentation*
User Documentation
2.1 API Reference

Table of Contents

- Main Module
  - ruche.alternatives
  - ruche.help
  - ruche.install
  - ruche.uninstall
  - ruche.version
- Command-line Interface
  - cli.alternatives
  - cli.help
  - cli.install
  - cli.uninstall
  - cli.version
- Utilities: ruche
  - util.download
  - util.emitter
  - util.error
  - util.extract
  - util.match
  - util.register
  - util.remove
  - util.untilde
  - util.valid
- Utilities: CLI
  - cliUtil.argv
  - cliUtil.error
- Internal Events

2.1.1 Main Module

You can use ruche from your own projects. Achieving that is as simple as requiring ruche from within your script.
ruche Documentation, Release 0.0.1

ruche.alternatives

This function allows you to switch between different versions and installations of the same package. So both a bleeding edge and a legacy version are availables at your fingertips.

ruche.alternatives(package, callback)

Arguments

- **package** *(string)* – The package you want to manipulate. It could be either in a **short format**: curl or in a **long format**: curl-7.36.0-win64. To actually change the aliases in the path you have to specify a long format package name.

- **callback** *(function)* – It gets two arguments *(err, data)* where data is a flat array of locally availables packages (in **long format**: curl-7.36.0-win64).

Fires

- alt-show
- alt-choice

Throws

- The version specified is not valid

Exemple

```javascript
var ruche = require('ruche');
// Show local alternatives
ruche.alternatives('curl', function(err, data) {
    if (err) {
        // handle your error
    }
    console.dir(data); // ['curl-7.35.0-win32', 'curl-7.37.0-win64']
});
// Use one
ruche.alternatives('curl-7.37.0-win64', function(err, data) {
    if (err) {
        // handle your error
    }
    console.dir(data); // ['curl-7.35.0-win32', 'curl-7.37.0-win64']
});
```

ruche.help

Get the local text help file content according to the specified context. This function gets one or two arguments, in all cases the last is a callback function. When only one argument is given it gets the global ruche help.

ruche.help([context], callback)

Arguments

- **context** *(string)* – The context of the wanted help. It could be either ‘global’, undefined, or a valid ruche command.

- **callback** *(function)* – A callback function that is executed when the help file is loaded. It gets two arguments *(err, data)* where data is the content of the help file.

Throws

- 6 Chapter 2. Developer Documentation
• Can’t read the help file

Exemple

```javascript
var ruche = require('ruche');
// one argument
ruche.help(function (err, data) {
  if (err) {
    // handle your error
  }
  console.log(data); // output the ruche help
});
// two arguments
ruche.help('install', function (err, data) {
  if (err) {
    // handle your error
  }
  console.log(data); // output the install command help
});
```

---

**ruche.install**

Install a list of new packages.

**ruche.install(packages, callback)**

**Arguments**

- **packages** *(array)* – The list of packages you want to install. This argument is an array of strings that could be either in a *short format*: curl or in a *long format*: curl-7.36.0-win64.

- **callback** *(function)* – A callback function that is executed when all packages are installed. It gets two arguments *(err, data)* where data is a flat array of successfully installed packages *(in long format: curl-7.36.0-win64)*.

**Throws**

- Can’t reach URL

Exemple

```javascript
var ruche = require('ruche');
ruche.install(['git', 'curl-7.35.0-win32'], function (err, data) {
  if (err) {
    // handle your error
  }
  console.dir(data); // [ 'git-1.9.4-win32', 'curl-7.35.0-win32' ]
});
```
rucheDocumentation, Release 0.0.1

ruche.uninstall

Uninstall a list of packages.

ruche.uninstall(packages, callback)

Arguments

• packages (array) – The list of packages you want to remove. This argument is an array of strings that could be either in a short format: curl or in a long format: curl-7.36.0-win64.

• callback (function) – A callback function that is executed when all packages are removed. It gets two arguments (err, data) where data is a flat array of successfully installed packages (in long format: curl-7.36.0-win64).

Throws

• Can’t reach URL

Exemple

```javascript
var ruche = require('ruche');
ruche.uninstall(['git', 'curl-7.35.0-win32'], function (err, data) {
    if (err) {
        // handle your error
    }
    console.dir(data); // [ 'git-1.9.4-win32', 'curl-7.35.0-win32' ]
});
```

ruche.version

This function parse the package.json file of the current ruche installation to read its version number.

ruche.version(callback)

Arguments

• callback (function) – A callback function that is executed when the version number is loaded. It gets two arguments (err, data) where data is the version number: a string like this ‘0.0.1’.

Throws

• Can’t read the ruche package.json file

• Can’t parse the ruche package.json file

Exemple

```javascript
var ruche = require('ruche');
ruche.version(function (err, data) {
    if (err) {
        // handle your error
    }
    console.log(data); // output the ruche version number
});
```
2.1.2 Command-line Interface

The command-line interface (or cli) is a module of ruche. It is a wrapper around the main ruche module that is called when the user use the binary from a prompt.

cli.alternatives

This function is a wrapper around ruche.alternatives. It displays the available to stdout versions of a package and prompt the user on which one to use.

cli.alternatives(argv[, callback])

Arguments

• argv (object) – An usable object of arguments and options.
• callback (function) – A callback function that is executed when the output has been written. It gets one argument (data).

Fires

• cli-alt-show
• cli-alt-choice

See ruche.alternatives

Exemple

```javascript
var cli = require('./lib/cli');
cli.alternatives(argv, function (data) {
    console.log(data); // output a second time
});
```

cli.help

This function is a wrapper around ruche.help. It output the appropriate ruche help to stdout according to the context.

cli.help(context[, callback])

Arguments

• argv (string) – The context of the wanted help. It should be a valid ruche command or ‘global’.
• callback (function) – A callback function that is executed when the output has been written. It gets one argument (data).

See ruche.help

Exemple

```javascript
var cli = require('./lib/cli');
cli.help(function (data) {
    console.log(data); // output a second time
});
```
cli.install

This function is a wrapper around ruche.install. It install the wanted packages and display some usefull information about the state of the process.

cli.install(argv[, callback])

Arguments

• argv (string) – An usable object of arguments and options.

• callback (function) – A callback function that is executed when the output has been written.
  It gets one argument (data).

See ruche.install

Exemple

```javascript
var cli = require('./lib/cli');
cli.install({ packages: ['curl'] }, function (data) {
  console.log(data); // output a second time
});
```

cli.uninstall

This function is a wrapper around ruche.uninstall. It remove the wanted packages and display some usefull information about the state of the process.

cli.uninstall(argv[, callback])

Arguments

• argv (string) – An usable object of arguments and options.

• callback (function) – A callback function that is executed when the output has been written.
  It gets one argument (data).

See ruche.uninstall

Exemple

```javascript
var cli = require('./lib/cli');
cli.uninstall({ packages: ['curl'] }, function (data) {
  console.log(data); // output a second time
});
```
cli.version

This function is a wrapper around ruche.version. It output the ruche version number to stdin.

cli.version([callback])

Arguments

- **callback (function)** – A callback function that is executed when the output has been written. It gets one argument (data) where data is the version number: a string like this ‘0.0.1’.

See ruche.version

Exemple

```javascript
var cli = require('./lib/cli');
cli.version(function(data) {
    console.log(data); // output a second time
});
```

2.1.3 Utilities: ruche

The following functions are called by the main module.

util.download

Download a ruche package from a repository. This uses rc module to determine the source repository and the output directory.

util.download(match, i, callback)

Arguments

- **match (object)** – An object of parameters about the package that will be downloaded. Typically this value come from a ruche.json file.
- **i (number)** – This identify the current download. This is usefull for catching events when multiple downloads are required. i is used in the name of the events that are fired.
- **callback (function)** – A callback function that is executed when the file is downloaded and written to the file system. It gets two arguments (err, location) where location is the local path to the file.

Fires

- dl-start-i
- dl-data-i
- dl-done-i

Throws

- Unaccessible URI

Exemple
var rucheUtil = require('./lib/ruche/util');
rucheUtil.download(match, 0, function (err, location) {
  if (err) {
    // handle your error
  }
  console.log(location); // where the file is located
});

util.emitter

Require this to get access to the ruche specific events. This inherits directly from the node native class EventEmitter. The extends method in use is the Underscore.js one.

util.emitter()

Returns (object) – The ruche internal event emitter.

Exemple

var rucheUtil = require('./lib/ruche/util');
rucheUtil.emitter.on('dl-start-0', function (length) {
  console.log('This download is %sB long', length);
});

util.error

It is never a good idea to let a process continue on unknown error. This function helps you handling this situation by emitting an error event.

util.error()

Fires

• error

Exemple

var error = require('./lib/ruche/util/error');
error();

util.extract

This method use a precedent download and extract it. This uses rc module to determine the the tpm and share directories, respectively where download are stored and where to extract them.

util.extract (match, i, callback)

Arguments

• match (object) – An object of parameters about the package that will be extracted. Typically this value come from a ruche.json file.
• **i (number)** – i This identify the current operation. This is usefull for catching events when *multiple extractions* are required. i is used in the name of the events that are fired.

• **callback (function)** – A callback function that is executed when the file is extracted and written to the file system. It gets two arguments *(err, location)* where *location* is the local path to the file.

**Fires**

• gz-start-i
• gz-data-i
• gz-done-i
• tar-start-i
• tar-data-i
• tar-done-i

**Throws**

• No downloaded package to extract

**Exemple**

```javascript
var rucheUtil = require('./lib/ruche/util');
rucheUtil.extract(match, 0, function (err, location) {
  if (err) {
    // handle your error
  }
  console.log('Package extracted to %s');
});
```

---

**util.match**

This function parse a *ruche.json*. Those files describes all the packages available. By passing a *wanted* argument you choose the specifications of the package you want.

**util.match (wanted, data)**

**Arguments**

• **wanted (object)** – Package and spec you want. It’s an object with at least one key *package*. Optionnals keys can be added: *version* and *platform*.

• **data (object)** – The content of a *ruche.json* file. is used in the name of the events that are fired.

**Returns (object)** – Return the info of the best package.

**Exemple**

```javascript
var rucheUtil = require('./lib/ruche/util');
var data = {}; // content of a ruche.json file
var match = rucheUtil.match({ package: 'curl' }, data);
```
util.register

Read a entry of a ruche.json file and copy and make aliases for all files under the bin category. Those aliases are placed in the bin local folder indicated by the rc configuration.

**util.register** *(match, i, callback)*

**Arguments**

- **match (object)** – An object of parameters about the package that will be downloaded. Typically this value come from a ruche.json file.
- **data (number)** – This identify the current operation. This is usefull for catching events when *multiple operations* are required. i is used in the name of the events that are fired.
- **callback (function)** – A callback function that is executed when the aliases are created. It gets one argument binaries.

**Fires**

- reg-start-i
- reg-data-i
- reg-done-i

**Exemple**

```javascript
var rucheUtil = require('./lib/ruche/util');
rucheUtil.register(match, 0, function (binaries) {
  console.log(binaries); // where the aliases are located
  // [ {
  //   bin: 'run', file: 'path/to/run'},
  //   { bin: 'stop', file: 'path/to/stop'}
  // ]
});
```

util.remove

Remove a package from the local ruche installation. It uses the rc module to find the directories to empty and a ruche.json file to find the binaires to unregister.

**util.remove** *(match, i, callback)*

**Arguments**

- **match (object)** – An object of parameters about the package that will be removed. Typically this value come from a ruche.json file.
- **data (number)** – This identify the current operation. This is usefull for catching events when *multiple operations* are required. i is used in the name of the events that are fired.
- **callback (function)** – A callback function that is executed when all is done.

**Fires**

- unreg-start-i
- unreg-done-i

**Exemple**
var rucheUtil = require('./lib/ruche/util');
rucheUtil.remove(match, 0, function () {
  console.log('Done!');
});

util.untilde

Parse a string and convert it into a valid path. Different symbols can be added: ~ represents the home directory of the user (like: C:/Users/MyProfile). @ represents the installation folder of the current ruche installation.

util.untilde(path)

Arguments

- path (string) – The string to parse.

Returns (string) – Return the resolved path.

Exemple

```javascript
var rucheUtil = require('./lib/ruche/util');
var home = rucheUtil.untilde('~'); // user home directory
var root = rucheUtil.untilde('@'); // ruche root
```

util.valid

Returns an object of valid options for each commands or context. Each key in the object is a context and each value an array of valid options.

util.valid()

Returns (object) – An object of valid options for each context.

Exemple

```javascript
var rucheUtil = require('./lib/ruche/util');
// Get the valid options of the 'alternatives' command
rucheUtil.valid().alternatives; // [ 'choice' ]
```

2.1.4 Utilities: CLI

The following functions are called by the cli module.
**cliUtil.argv**

This function parses the command from stdin into an object that is usable by the Cli.

```javascript
cliUtil.argv(argv)
```

**Arguments**

- `argv (object)` – From stdin.

**Returns**

- `(object)` – An usable object of arguments and options.

**Exemple**

```javascript
var cliUtil = require('./lib/cli/util');
var args = cliUtil.argv(process.argv);
```

**cliUtil.error**

This function logs a colorful error to stdout.

```javascript
cliUtil.error(err)
```

**Arguments**

- `err (Error)` – The error to log.

**Returns**

- `(string)` – The message of the error.

**Exemple**

```javascript
var cliUtil = require('./lib/cli/util');
var e = new Error('Nop!');
cliUtil.error(e);
```

### 2.1.5 Internal Events

Both `ruche` and `cli` modules emit and listen to events. You can also hook into this events to implement your own business logic.

**emit (`cli-alt-show`)**

- `Event` Emitted when the alternatives are printed to stdout.

**emit (`cli-alt-choice`, number)**

- `Event` Emitted when the user has chosen an alternative.

**Arguments**

- `answer (number)` – The number chosen by the user.

**emit (`alt-show`, alternatives)**

- `Event` Emitted when alternatives are availables.

**Arguments**

- `alternatives (array)` – Locally availables versions of the package.
emit ('alt-choice', package)

**Event** Emitted when the user chose an alternative version.

**Arguments**
- **package (number)** – The package name in *long format*.

emit ('install-i', match)

**Event** Emitted when got a match. \( i \) is an identifier.

**Arguments**
- **match (object)** – Info about the package.

emit ('uninstall-i', match)

**Event** Emitted when got a match. \( i \) is an identifier.

**Arguments**
- **match (object)** – Info about the package.

emit ('dl-start-i', length)

**Event** Emitted when starting download. \( i \) is an identifier.

**Arguments**
- **length (number)** – The total length of the file.

emit ('dl-data-i', chunk)

**Event** Emitted when receiving data. \( i \) is an identifier.

**Arguments**
- **chunk (object)** – Chunk of data.

emit ('dl-done-i')

**Event** Emitted when the file is downloaded and written to the file system. \( i \) is an identifier.

emit ('error', err)

**Event** Emitted when an uncaughtException occurs.

**Arguments**
- **err (Error)** – The error to handle.

emit ('gz-start-i', length)

**Event** Emitted when starting unzipping. \( i \) is an identifier.

**Arguments**
- **length (number)** – The total length of the file.

emit ('gz-data-i', chunk)

**Event** Emitted when receiving data from unzipping. \( i \) is an identifier.

**Arguments**
- **chunk (object)** – Chunk of data.

emit ('gz-done-i')

**Event** Emitted when unzipping is done. \( i \) is an identifier.
emit ('tar-start-i', length)
    Event Emitted when starting un-tar. i is an identifier.
    Arguments
    • length (number) – The total length of the file.

emit ('tar-data-i', chunk)
    Event Emitted when receiving data from tar. i is an identifier.
    Arguments
    • chunk (object) – Chunk of data.

emit ('tar-done-i')
    Event Emitted when the package is copied into share. i is an identifier.

emit ('reg-start-i', length)
    Event Emitted when start registering. i is an identifier.
    Arguments
    • length (number) – The number of aliases to register

emit ('reg-data-i', chunk)
    Event Emitted when an alias is created. i is an identifier.
    Arguments
    • chunk (object) – Identifier: registered file index.

emit ('reg-done-i')
    Event Emitted when all aliases are created. i is an identifier.

emit ('unreg-start-i')
    Event Emitted when start removeing. i is an identifier.

emit ('unreg-done-i')
    Event Emitted when removing is done. i is an identifier.

2.2 Servers

Most of the commands require to be connected to a server. You can edit your configuration to any valid ruche server. The servers are designed to be simple, you need nothing more than a HTTP server that serve an directory statically.

To build your own server you need to valid three things:

• Filesystem Hierarchy
• All the ruche.json files
• Your archives
2.2.1 Filesystem Hierarchy

Each new package gets a dedicated directory. We recommend to the name as simple as possible (Ex: git/ or ruby/). In this package repository create a dist/ and a lib/ folder. The first one contains the archives that will be downloaded by the client. The second directory contains additional scripts that can be useful for some packages (Ex: installation of MySQL as a Windows service).

Finally create a ruche.json file in the package directory.

Exemple:

```
+ curl/
  | + dist/
  |   | + curl-7.36.0-win32.tar.gz
  |   | + curl-7.36.0-win64.tar.gz
  |   | + curl-7.37.0-win32.tar.gz
  |   | + curl-7.37.0-win64.tar.gz
  | + lib/
  |   | + script1.js
  |   | + script2.js
  | + ruche.json
  | + php/
  ...
```

2.2.2 Specifications of ruche.json Files

This file lists the available versions of a package and describes them. It’s a JSON array which contains objects. Each object is an available version and has the following keys:

- **package**: The package name. Must be the same as the parent folder.
- **version**: The version number. 1.35.2 is a good value.
- **platform**: Either win32 or win64.
- **bin**: [optional] Lists the executables that have to be registered in the user PATH. It is an object where keys are the name in the PATH and values the path to the executable is the archive.
- **homepage**: A link to the author page.

Exemple:

```
[
    {
        "package": "curl",
        "version": "7.35.0",
        "platform": "win32",
        "bin": {
            "curl": "bin/curl.exe"
        },
        "homepage": "http://curl.haxx.se/"
    },
    {
        "package": "curl",
        "version": "7.35.0",
        "platform": "win64",
        "bin": {
            "curl": "bin/curl.exe"
        }
    }
]
```
2.2.3 Building an archive

The archives must be named as follow: package-version-platform.tar.gz.

**Note:** We recommend 7-Zip for building archives.

1. Put all your file in a folder named package-version-platform.
2. **Use a software to Tar this** folder into a package-version-platform.tar file.
3. **Use a software to Gzip this** file into a package-version-platform.tar.gz file.
4. Put this file is the package/dist/ folder of your server.

Exemple:

```
| + curl/
|   + dist/
|   |   + curl-7.36.0-win32.tar.gz
|   |   + curl-7.36.0-win32.tar
|   |   + curl-7.36.0-win32
|   |   + bin
|   |   + curl.exe
| ...```