Reference of mvUCCU

nodejs compatible APIs

fs

cwd = is same as RPGMV.exe at first

We support following APIs which have same interfaces as nodejs,

- accessSync
- readdirSync
- readFileSync
- writeFileSync
- appendFileSync
- realpathSync
- renameSync
- rmdirSync
- statSync
- existsSync
- unlinkSync
- mkdirSync

And some special functions work with Buffer

- readFileBufferSync
- writeFileBufferSync
- writeFileLenSync
- appendFileBufferSync
- appendFileLenSync

`fs` can access both file on disk and file in at resources. It's implemented by `QFile`

Class Stats

- isFile
- isDirectory
- isSymbolicLink

and properties

- uid
- gid
- size

process

- exit
- cwd

and properties

- argv
- execPath

  path of the executable file
- pid
- version
- env
- platform

  one of `win32`, `darwin`, `linux` and `unsupported`.

vm

Just to ensure some basic nodejs modules can run.

util

Just to ensure some basic nodejs modules can run.

assert

Just to ensure some basic nodejs modules can run.

path

Just to ensure some basic nodejs modules can run.

module

Just to ensure some basic nodejs modules can run.

debugging

You can use chrome console to debug your script. All these things can be configured in `uccu.json` under `mvuccu` folder.

console

- log

  `console.log(format, ...)`

  `format` is qt format string, for example

```
console.log("Hello, %1. Today is %2", name, day)
```

- err

  same as log

debug

- break

  `debug.break()`, trigger a debug interrupt at anywhere.

  can be used if and only v8 debug is enabled.

mvUCCU

uccu.json

`uccu.json` is a configure file for mod system. It contains few sections and some of them are obsoleted.

`EnableLang` and `LangFile` set a language file replacement.

`EnableConsole` controls console (for windows).

When `EnableLog` is set, mcuccu write log file to temporary directory.

`CategoryMode` modified log level, usually you don't need to change it.

`v8` control v8 related behaviors, currently, debugger only.

mod.json

In `mod.json`, we defined some basic information of the mod.

```
{
  "name": "essentials",
  "version": "0.0.0",
  "description": "Essential mod for all other mods",
  "dependencies": {"$rmmv":"=1.*", "$uccu":"=1.*"},
  "author": "xxx",
  "homepage": "yyy"
}
```

In dependencies section, you can make your mod depend on some other mods, especially, you can use `$rmmv` to refer the version of editor and `$uccu` for mod system.

platform

Constants

- qt_version

  now, typically, `5.4.2`
- os_type
- os_version
- os
- kernel
- kernel_version
- cpu
- app_home
- locale

Functions

- registerResourceBuffer(buffer, root)
- registerResourceRCC(filename, root)
- replaceTranslatorFile(originalfile, replacedfile)
- addTraslatorFile(filename)

modapi

Functions,

- get(file)
- update(file, buffer/string)
- add(file, buffer/string)
- each(path, callback)
- node(code)

  Create a `QmlNode` with code.
- doc(code)

  Create a `QmlDocument` with code
- qml(file)

  Create a `QmlDocument` with file

Constant,

- api
- rmmv
- build

Class QmlDocument

Properties,

- node/root

  get root object of the document
- imports/parmas

  get `imports/parmas` of the document. returns

Functions,

- toString
- save([filename])

  if the `QmlDocument` is created by `ModAPI.qml("xxx")`, `filename` would be set to `"xxx"` by default.

Class QmlNode

Properties,

- vars
- name

  Get field of this objected assigned relative to its parent
- type
- parent

Functions,

- obj

  Get objects not assigned to field.

  ```
  XXX {
   //yyy
  }
  ```
- all

  Get all objects, including those assigned to fields
- names

  Get all field names
- exists(name)
- on(name, [val])
- binded(name)
- get(name)
- ref(name)

Create a reference of field `name`

- end

  return parent

- clr(name)

  remove value of name

- set(name, value)

  set value of name

  return reference of this field.

- def(name, type, [value])

  type should be one of `object`, `function`,`property`,`signal`

  value here can be `rawCode` only. aka a String that will be embedded into final qml file.

- _default(name [, val])
- readonly(name [, val])
- info(name)

  ```
  {
    "name": name,
    "kind": "Property", "Function", "Signal" or "Object",
    // property onle
    "_default": true/false,
    "readonly": true/false,
    "ret": "type",
    "binded": true/false
  }
  ```

- ret(name[, value])
- add(object)
- addBefore(object)

  `x.addBefore` will add object to **parent of `x`** before the object `x`.

- makeObject(name)

  mark `name` object, that it to say `xxx: YYY {...}` for qml

- makeProperty

  mark `name` property

- kill

  remove self from parent

- remove(name)

  remove children by name

- getObjectById(id [, max_deep])
  recursively find a object have `id: xxx`
- getObjectsByType(type [, max_deep])
  recursively find objects have given `type`
- getValueByName(name [, max_deep])
  recursively find references of given `name`
- select(field, value [, max_deep])
  find all objects `x` have `x[field]=value`

Class QmlRef

Properties,

- name

Functions,

- remove
  remove referenced object
- val([value])
- info
- _default
- readonly
- ret
- isNull
- isNameExists
- isValueExists
- end
  return to `qmlnode`

Differences between `fs.readFileSync` and `ModAPI.get`

Both `fs.read` and `ModAPI.get` can read file inside qt package. However, their are slight difference between them.

1. `fs.readFileSync` can read file on drivers, `ModAPI.get` is designed to read file under `qml/` folder in qt package only.
2. `fs.write` cannot update final data write to qt package, that is won't work for qmls.
3. After `ModAPI.update`, we will cache your modify, but not write them to qt package instantly. Others won't be able to read the modified qml file by `fs.read`. While `ModAPI.get` will search in the cache first. So, if you want the original version of some qml file, you should use `fs.readFileSync`, or you can use `ModAPI.get` to read

   qml modified by other mods. But you should always use `ModAPI.update` for modifying qml file.
4. `ModAPI`'s root folder is `:/qml/`, for example, you can use `Main/MainWindow.qml` to get `:/qml/Main/MainWindow.qml`.

buffer

`Buffer` is used for holding `QByteArray`. It prevent the type convert between `QByteArray` and `v8::String` .

- ToString
- clone
- reserve(size)
- resize(size)
- [x]
- [x] = y

property,

- length

static function

- fromString(string)

3rdparty module(s)

lodash