Library

Jan 25, 2021

Contents

CHAPTER 1

.NET

1.1 Cherry.Common

1.2 Cherry.Syntax

1.3 Cherry.Desktop

1.4 Cherry.Web

1.5 Cherry.Android

CHAPTER 2

Java / Kotlin

2.1 cherry.common

2.2 cherry.syntax

2.3 cherry.android

2.4 cherry.fx

CHAPTER 3

Prototype

3.1 Introduction

```
-- let's go with image first --

[IMAGE logo]

-- then simple edit field --

[EDIT 'Name']

-- and button to go --

[BUTTON 'Go']
```

3.2 Usage

3.2.1 Simple prototyping

It is easy to write down cherry script using just a piece of paper.

3.2.2 Code generation

Libraries in this project will help with generating UI layout in format specific for the platform. You may generate XML for Android as well as HTML or even builder code for GUI toolkit you need.

3.2.3 Embedded systems

Cherry script may be converted to bytecode which will result in tiny amount of data needed to store or transmit over network.

3.3 Definition

Definition of interface appearance consists of a list of elements, also called components, together with attributes assigned to them such as name, text or other properties.

Each element is of a specific kind and this type is written in square brackets together with the attributes. For example button is represented simply by `[BUTTON]`.

Attributes are defined as a set of *key*=*value* pairs. If a value contain any special characters or spaces, it should be surrounded with citation marks.

When you're insane you might also write something like `['BUTTON' "name"=`value`]` but as you can see, it doesn't look well.

Every text outside square brackets is ignored. One line `//` and multiline `/* */` comments are allowed.

```
[TEXT text='Hello, world.']

//[TEXT text='This element is commented out']

This is ignored static text.

/*
 * Inside multiline comments you may also use
 * [TEXT text='Nothing'] and it will be ignored too.
 */

[BUTTON text='Thank you']
```

Component names and attribute keys are considered to be case insensitive. For nicer look component names are usually written with capital letters. Attribute key names are also case insensitive but it is recommended to use lower case letters for them. And finally it is nice to write predefined values in capital letters.

Values that need to be surrounded by quotation marks might be written using apostrophes (which is recommended) or double quotes.

Elements are packed into groups. Each group defines vertical (default) or horizontal layout.

3.3.1 Escaping

When in need to use of quotation character inside value content, doublets might be used like in folllowing examples.

```
[LABEL text='That''s my piece of floor']
```

3.4 Set

Root components.

Layout components.

Basic components.

3.5 Unicode

For values there is full unicode support.

```
[TEXT text='']
```

```
[TEXT ]
```

3.6 Sticking

Elements are sticked to previous elements and there is no need to fold them in definition although they are represented as tree of elements.

For example `[GROUP]` begin new group of components until next `[GROUP]`. So if you need to put some elements vertically and horizontally you have to separate them by several `[GROUP]` elements.

```
[EDIT type=SELECT]
[OPTION text='First option']
[OPTION text='Second option']
```

3.7 Parser

3.7.1 Outer expression

Take this one as a test.

```
//[IGNORE][TEXT text=]
[IMAGE
icon='logo'
//hint='[IMAGE]'
text='
That''s my job
'
]
/*
[GROUP orientation='VERTICAL']
[EDIT text='' hint='User']
//*/
[TEXT x="""""""]
///*
/* [EDIT text='' type='PASSWORD' hint='Password'] */
[CHECK text='Remember me']
[GROUP]
[TEXT text='Need account?']
[LINK text='Register here' target='http://target.link']
//
```

```
PCRE (PHP) + ECMAScript (JavaScript)
```

```
/\/\/[^\r\n]*(?:\r?\n|$)|\/\*(?:.|[\r\n])*?(?:\*\/|$)|(\[(?:\/\/[^\r\n]*(?:\r?\n|$)|\/
↪\*(?:.|[\r\n])*?(?:\*\/|$)|'[^']*'|\"[^\"]*\"|[^\]])*\])/g
```

C#

```
new Regex(@"
\/\/[^\r\n]*(?:\r?\n|$)
|
\/\*(?:.|[\r\n])*?(?:\*\/|$)
|
(
\[
(?:
  \/\/[^\r\n]*(?:\r?\n|$)
  |
  \/\*(?:.|[\r\n])*?(?:\*\/|$)
  |
  '[^']*'
  |
  \""[^\""]*\""
  |
  [^\]]
)*
\]
)
", RegexOptions.IgnoreWhitespace | RegexOptions.Multiline);
```

Python

```
r"\/\/[^\r\n]*(?:\r?\n|$)|\/\*(?:.|[\r\n])*?(?:\*\/|$)|(\[(?:\/\/[^\r\n]*(?:\r?\n|
↪$)|\/\*(?:.|[\r\n])*?(?:\*\/|$)|'[^']*'|\"[^\"]*\"|[^\]])*\])"g
```

Golang

```
`\/\/[^\r\n]*(?:\r?\n|$)|\/\*(?:.|[\r\n])*?(?:\*\/|$)|(\[(?:\/\/[^\r\n]*(?:\r?\n|$)|\/
↪\*(?:.|[\r\n])*?(?:\*\/|$)|'[^']*'|\"[^\"]*\"|[^\]])*\])`g
```

https://regex101.com/r/IK8B3r/1

3.8 Bytecode

It is also possible to convert UI definition into bytecode. All texts are UTF-8 encoded.

Header starts with 4 bytes BB BB 00 01, where last one indicate version of bytecode.

Data contain only definitions and text values.

It will use snappy compression by default.

Each definition contains one character indicating type, like “T” for text, number of elements limited to 255, and a list of key - value pairs. We are however limited to 65535 elements of key and values.

All keys are converted to lowercase and there is a limitation of 65535 elements in one bytecode file.

‘T’ 02 00 01 00 02 00 03 00 04

00 01 : 00 04 ‘text’ 00 02 : 00 07 ‘My text’ 00 03 : ‘key’ 00 04 : ‘X’

There should be two tables, one for keys and one for values, because there is no point in having more than 255 keys I guess.

```
[E option='Option 1' option='Option 2' display=select]
```

3.9 Example

3.9.1 Sign in

This is example of sign in screen.

```
[IMAGE icon='logo']

[GROUP orientation='VERTICAL']
[EDIT text='' hint='User']
[EDIT text='' type='PASSWORD' hint='Password']
[CHECK text='Remember me']

[GROUP orientation='HORIZONTAL']
[BUTTON text='Sign in']
[BUTTON text='Forgotten password']

[GROUP]
[TEXT text='Need account?']
[LINK text='Register here' target='http://target.link']
```

The same using short version.

```
[I 'logo']

[G V]
[E hint='User']
[E type=PASSWORD hint='Password']
[C 'Remember me']

[G H]
[B 'Sign in']
[B 'Forgotten password']

[G]
[T 'Need account?']
[L 'Register here' target='http://target.link']
```

And here is its JSON representation.

```
[
    {
        "_": "IMAGE",
        "icon": "logo"
    },
    {
        "_": "GROUP",
        "orientation": "VERTICAL"
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "_": "EDIT",
        "text": "",
        "hint": "User"
    },
    {
        "_": "EDIT",
        "text": "",
        "type": "PASSWORD",
        "hint": "Password"
    },
    {
        "_": "CHECK",
        "text": "Remember me"
    },
    {
        "_": "GROUP",
        "orientation": "HORIZONTAL"
    },
    {
        "_": "BUTTON",
        "text": "Sign in",
    },
    {
        "_": "BUTTON",
        "text": "Forgotten password"
    },
    {
        "_": "TEXT",
        "text": "Need account?"
    },
    {
        "_": "LINK",
        "text": "Register here",
        "target": "http://target.link"
    }
]
```

Simple and not so smart conversion from JSON to XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <element>
      <_>IMAGE</_>
      <icon>logo</icon>
   </element>
   <element>
      <_>GROUP</_>
      <orientation>vertical</orientation>
   </element>
   <element>
      <_>EDIT</_>
      <hint>User</hint>
      <text />
   </element>
   <element>
      <_>EDIT</_>
      <hint>Password</hint>
```

(continues on next page)

(continued from previous page)

```
      <text />
      <type>PASSWORD</type>
   </element>
   <element>
      <_>CHECK</_>
      <text>Remember me</text>
   </element>
   <element>
      <_>GROUP</_>
      <orientation>vertical</orientation>
   </element>
   <element>
      <_>BUTTON</_>
      <text>Sign in</text>
   </element>
   <element>
      <_>BUTTON</_>
      <text>Forgotten password</text>
   </element>
   <element>
      <_>TEXT</_>
      <text>Need account?</text>
   </element>
   <element>
      <_>LINK</_>
      <target>http://target.link</target>
      <text>Register here</text>
   </element>
</root>
```

3.9.2 Tabbed view

```
[USE 'tabs']
[BUTTON]
[FRAGMENT 'tabs']
[PAGE 'General']
[TEXT 'Name']          [EDIT]
[TEXT 'Description']   [EDIT]
[PAGE 'Additional']
[TEXT 'Modified by'] [TEXT]
```

3.9.3 Multiple screens

```
[SCREEN]
[SCREEN]
```

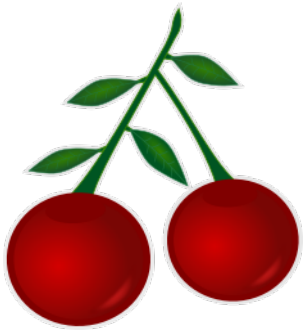
3.9.4 File browser

```
[SCREEN]
[DATA]
```

(continues on next page)

(continued from previous page)

```
[GROUP v]
[BUTTON 'Ok']
[BUTTON 'Cancel']
```



CHAPTER 4

Welcome

Visual Cherry Project contains set of libraries and programs for making user interface definitions quick and simple way.