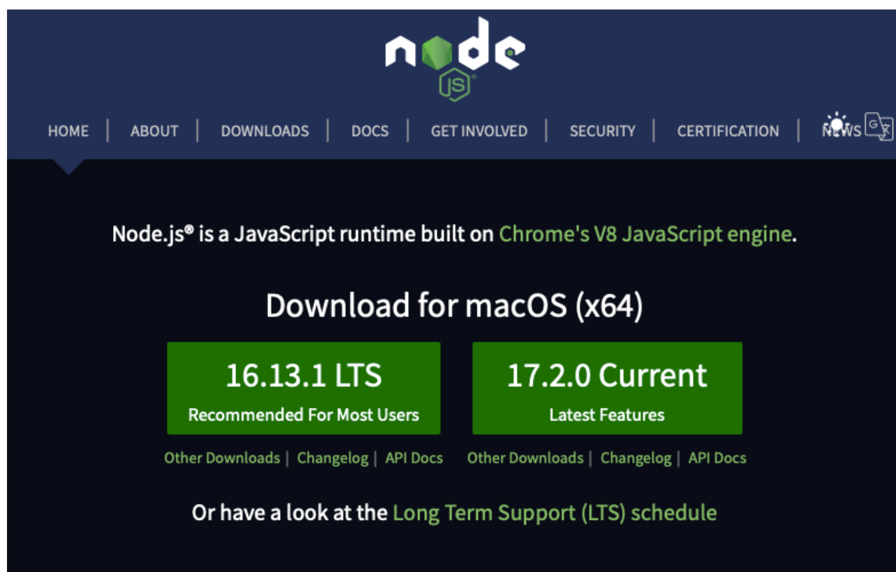
付録 B

SmrAI 開発ガイド

1. 開発環境の構築

1.1 Node.js のインストール

https://nodejs.org より Windows 用のインストーラーをダウンロードする．LTE 版をダウンロードしてインストールすれば良い．

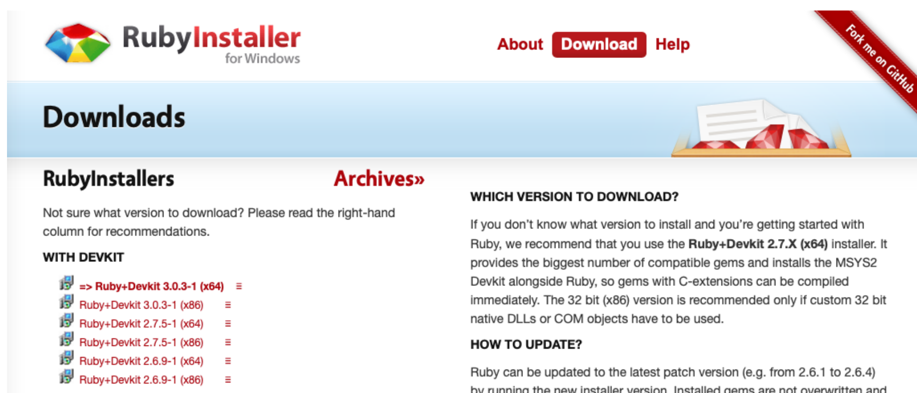


コマンドプロンプトを立ち上げ，node, npm のバージョンを確認しておく．

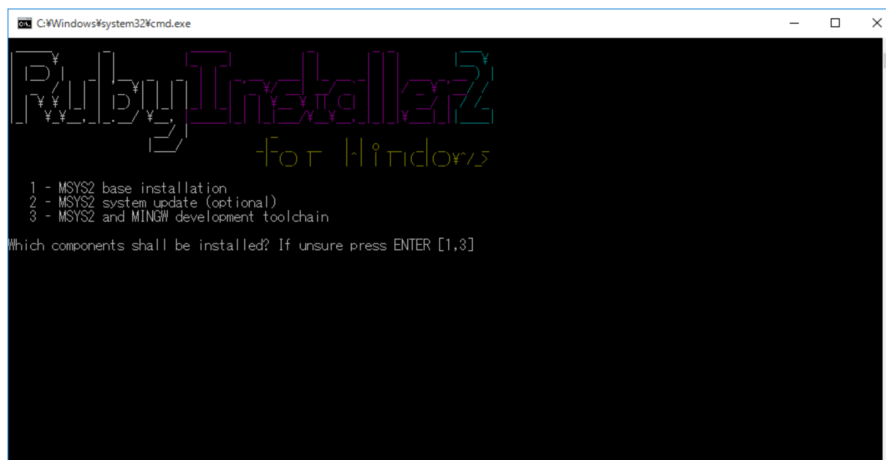
```
> node --version

> npm --version
```

1.2 Ruby のインストール

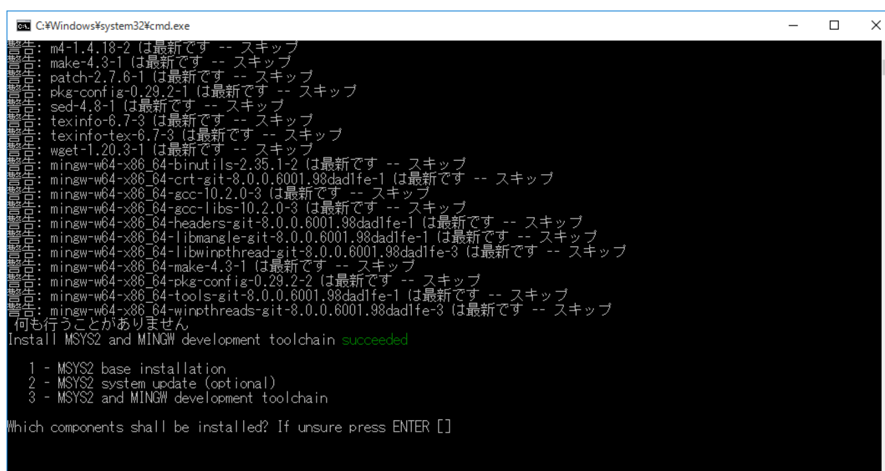
https://rubyinstaller.org/downloads/ より Windows 用のインストーラーをダウンロードする．WITH DEVKIT の **Ruby+Devkit2.7.5-1(x64)** をインストールする。



インストール後にコマンドプロンプトが起動し、以上のような画面になるので、そのままEnter キーを押せばよい（Enter キーを押すことで、1:MSYS のインストールと 3:開発に必要なツールチェーンのインストールが実行される）



「succeeded」と表記されたら無事にインストール終了



1.3 Smalruby3 のセットアップ(Web アプリ)

コマンドプロンプトを立ち上げ、以下のコマンドを入力する

```
>git clone https://github.com/gfd-dennou-club/smalruby3-gui-SmrAI.git
```

必要なモジュールを取得する

```
>cd smalruby3-gui-iot

>npm install
```

Smalruby3 を動かす。ファイアウォールの許可をする。

```
>npm start
```

ブラウザで http://localhost:8601/ にアクセスすると Smalruby3 を利用できる.

.1.4 SmrAI(Electron)のセットアップ(Win アプリ)

コマンドプロンプトを立ち上げ、必要なリポジトリをクローンする

```
>git clone https://github.com/gfd-dennou-club/electron_SmrAI.git
```

必要なモジュールを取得する

```
>cd electron_SmrAI

>npm install
```

SmrAI を動かす

```
>npx electron app
```

2. 開発方法

2.1 カテゴリの追加

Smalruby3 のブロックは「動き」「見た目」「イベント」などのカテゴリで分けられている

例として「テスター」カテゴリを追加した際の方法を紹介する

- smalruby3-gui-SmrAI¥src¥lib¥ruby-generator 直下に「tester.js」というファイルを新規作成し、以下のコードを書く(ruby-generator 直下にある js ファイルの内容をコピペしてもよい)

```
export default function (Generator) {
    const getUnquoteText = function (block, fieldName, order) {
        const input = block.inputs[fieldName];
        if (input) {
            const targetBlock = Generator.getBlock(input.block);
            if (targetBlock && targetBlock.opcode === 'text') {
                return Generator.getFieldValue(targetBlock, 'TEXT') || '';
            }
        }
        return Generator.valueToCode(block, fieldName, order);
    };

     //新しくブロックを追加した際は、ここにブロックと Ruby コードの対応を定義する
     //記述方法は後述する

}
```

- smalruby3-gui-SmrAI¥src¥lib¥ruby-generator¥index.js を開き、以下のコードを記述して、先ほど作った tester.js を TesterBlocks という名前で import する

```
import TesterBlocks from './tester.js';
```

また、index.js ファイルをスクロールして最後部に以下のコードを記述する

```
TesterBlocks(RubyGenerator);'
```

```
1   import _ from 'lodash';
2
3   import Blockly from 'scratch-blocks';
4   import Generator from '../generator';
5
6   import MathBlocks from './math.js';
7   import TextBlocks from './text.js';
8   import ColourBlocks from './colour.js';
9   import MotionBlocks from './motion.js';
10  import LooksBlocks from './looks.js';
11  import SoundBlocks from './sound.js';
12  import EventBlocks from './event.js';
13  import ControlBlocks from './control.js';
14  import SensingBlocks from './sensing.js';
15  import OperatorsBlocks from './operators.js';
16  import DataBlocks from './data.js';
17  import ProcedureBlocks from './procedure.js';
18  import RubyBlocks from './ruby.js';
19  import TesterBlocks from './tester.js';
20  import TestersecondBlocks from './tester_second.js';
21  import MusicBlocks from './music.js';
22  import PenBlocks from './pen.js';
23  import VideoBlocks from './video.js';
24  import Text2SpeechBlocks from './text2speech.js';
25  import TranslateBlocks from './translate.js';
26
```

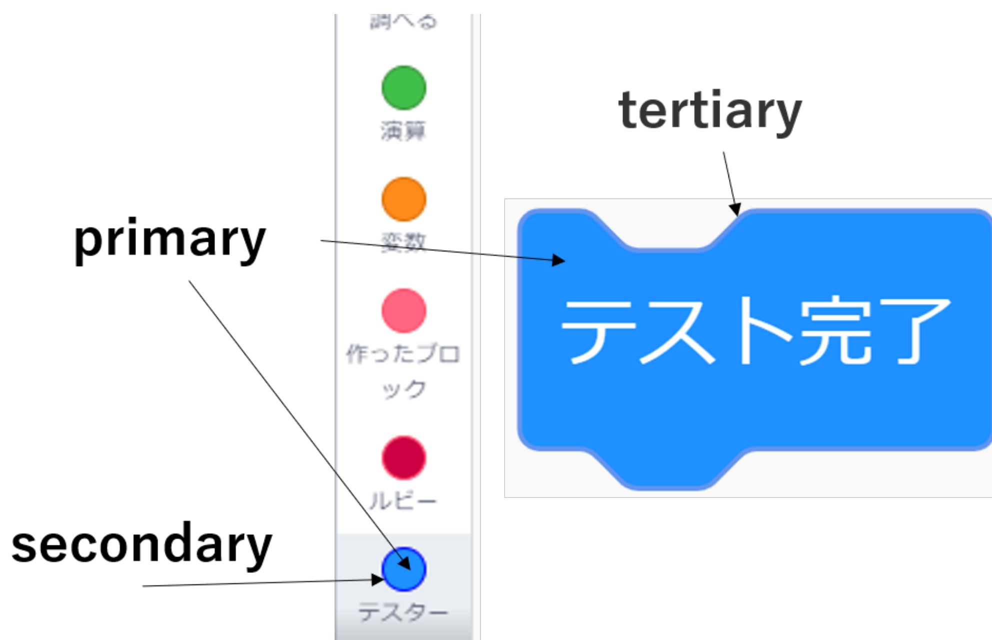
```
421
422 MathBlocks(RubyGenerator);
423 TextBlocks(RubyGenerator);
424 ColourBlocks(RubyGenerator);
425
426 MotionBlocks(RubyGenerator);
427 LooksBlocks(RubyGenerator);
428 SoundBlocks(RubyGenerator);
429 EventBlocks(RubyGenerator);
430 ControlBlocks(RubyGenerator);
431 SensingBlocks(RubyGenerator);
432 OperatorsBlocks(RubyGenerator);
433 DataBlocks(RubyGenerator);
434 ProcedureBlocks(RubyGenerator);
435 RubyBlocks(RubyGenerator);
436 TesterBlocks(RubyGenerator);
437 TestersecondBlocks(RubyGenerator);
438 MusicBlocks(RubyGenerator);
439 PenBlocks(RubyGenerator);
440 VideoBlocks(RubyGenerator);
441 Text2SpeechBlocks(RubyGenerator);
442 TranslateBlocks(RubyGenerator);
443
444 export default RubyGenerator;
445
```

- smalruby3-gui-SmrAI¥src¥lib¥define-ruby-blocks.js に以下のコードを追加した

```
const name2 = 'tester'

if (ScratchBlocks.Categories.hasOwnProperty(name2)) {
        return ScratchBlocks;
}

ScratchBlocks.Categories[name2] = name2;
ScratchBlocks.Colours[name2] = {
        //ブロックの色指定
        primary: '#1E90FF',
        secondary: '#0000FF',
        tertiary: '#6495ED'
};

ScratchBlocks.Extensions.register(
        `colours_${name2}`,
        ScratchBlocks.ScratchBlocks.VerticalExtensions.colourHelper(name2)
);
```

ブロックの色指定については、以下の図のようになっている。

primary はブロックのメイン色、secondary は円の際の色、tertiary はブロックの周りの色



- smalruby3-gui-SmrAI¥src¥lib¥make-toolbox-xml.js に以下のコードを追加した

```
ScratchBlocks.Msg.CATEGORY_TESTER = 'tester';
ScratchBlocks.ScratchMsgs.locales.en.CATEGORY_TESTER = 'tester';
ScratchBlocks.ScratchMsgs.locales.ja.CATEGORY_TESTER = 'テスター';
ScratchBlocks.ScratchMsgs.locales['ja-Hira'].CATEGORY_TESTER = 'テスター';
```

また、tester という名前の関数を定義し、その関数の戻り値に xml を記述するため、以下のコードを追加した

```
ScratchBlocks.Msg.CATEGORY_TESTER = 'tester';
const tester = function () {
    return `
    <category
        name="%{BKY_CATEGORY_TESTER}"
        id="tester"
        colour="#1E90FF"  //primary カラーと同じにする
        secondaryColour="#0000FF">  //secondary カラーと同じにする

        //ブロックを新しく追加した際はここにブロックの見た目を xml 形式で記述していく
        //記述方法は後述する

    </category>
    `;
};
```

また、最後までスクロールし、everything 定数の末尾に、以下を追加した

```
tester(isStage, targetId)
```

2.2 ブロックの追加

ここでは例として、先ほど作ったテスターカテゴリにブロックを追加する場合を説明する

- smalruby3-gui-SmrAI¥src¥lib¥define-ruby-blocks.js にブロックの定義を書いていく
  テンプレートは以下のようなコードである

```
ScratchBlocks.Blocks.ブロックの名前 = {
    init: function () {
        this.jsonInit({
            type: 'ブロックの名前',
            message0: 'ブロックの表示名',
            args0: [//ブロックの引数
            ],
            category: ScratchBlocks.Categories.ruby,
            extensions: [ブロックの色, ブロックの形状]
        });
    }
};
```

✓ 「ブロックの名前」はプログラム上でブロックを区別するための名前である
ブロックの名前を付ける際、【カテゴリ名_ブロックの役割】 のように命名するとよい

✓ 「ブロックの表示名」は実際に画面に表示されるブロックの役割を表すものである．「時計合わせを行う」と書けば、以下のように「「時計合わせを行う」と書かれたブロックが作られる

✓ 「ブロックの色」は p5 に示した define-ruby-blocks.js の ScratchBlocks.Extensions.register の引数にある「colors_${変数}」が入る．ここでは、変数が name2 になっており、name2 には tester が代入されているため、「colors_tester」と記述する．このようにすることで、define-ruby-blocks.js の ScratchBlocks.Colours で指定した色がブロックに反映される

✓ 「ブロックの形状」は 3 種類ある
shape_statement を指定すると、以下の四角いブロックが作られる．

output_number を指定すると、以下の丸いブロックが作られる．主に変数として扱われる．

output_boolean を指定すると、以下のダイヤ型のブロックが作られる．主に条件式として

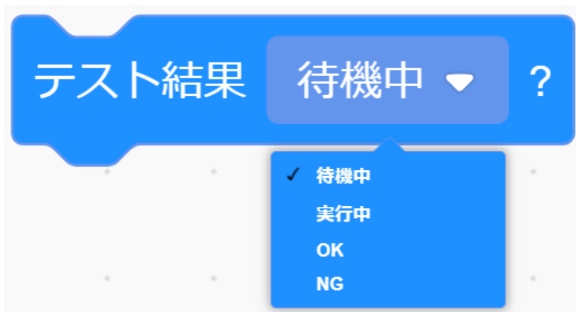
扱われる．

✓　「ブロックの引数」は 2 種類ある

type に field_dropdown を指定すると、ドロップダウンリストがブロックに追加される．

options には、リストの項目名とその項目名に対応させる文字列や数字を指定する．

```
args0: [
    {
        type: 'field_dropdown',
        name: '適当な名前',
        options: [
             //例えば…
             ['待機中', 'RESULT_WAIT'],
             ['実行中', 'RESULT_DOING'],
             ['OK', 'RESULT_OK'],
             ['NG', 'RESULT_NG']
        ]
    }
],
```



type に input_value を指定すると、文字入力ができたり変数型ブロックをはめ込めるスペースがブロックに追加される．name はその引数をプログラム上で判別するために付けるものである．

```
args0: [
     {
         type: 'input_value',
         name: '適当な名前'
     }
],
```

もちろん、1 つのブロックに field_dropdown タイプと input_value タイプの両方を引数と

して追加することもできる

```
args0: [
    {
        type: 'input_value',
        name: 'result'
     },
     {
         type: 'field_dropdown',
         name: 'ok_ng',
         options: [
              ['成功', 'true'],
              ['失敗', 'false'],
         ]
      }
],
```

また、引数を必要としない場合は、プロパティ args0 を指定しなければよい(書かなければよい)

- smalruby3-gui-SmrAI¥src¥lib¥ruby-generator¥tester.js にブロックと Ruby プログラムの対応を書いていく

テンプレートは以下のようなコードである

```
Generator.ブロックの名前 = function (block) {
        return `対応させる Ruby プログラム`;
};
```

ブロックの名前は、define-ruby-blocks.js で定義した「ブロックの名前」と同じにする

ブロックの引数の内容を、Ruby プログラムに対応させるためには、以下のように書く

```
Generator.ブロックの名前 = function (block) {
       //引数が input_value タイプの場合
       //取得した文字列 or 数値にダブルクォーテーションをつけないで変数に格納する場合
       const value = getUnquoteText(block,'引数の name',Generator.ORDER_NONE);
       //取得した文字列 or 数値にダブルクォーテーションをつけて変数に格納する場合
       const value = Generator.valueToCode(block,'引数の name',Generator.ORDER_
                                            NONE);
       //引数が field_dropdown タイプの場合
       const value2 = Generator.getFieldValue(block, '引数の name') || null;
       return `${value}${value2}`;
};
```

引数が input_value タイプについて、Hello World という文字列を getUnqouteText で受け取ると上の行のように出力される．また Generator.valueToCode で受け取ると下の行のように出力される

```
Hello World
"Hello World"
```

- smalruby3-gui-SmrAI¥src¥lib¥make-toolbox-xml.js にブロックの見た目を表す xml を書いていく

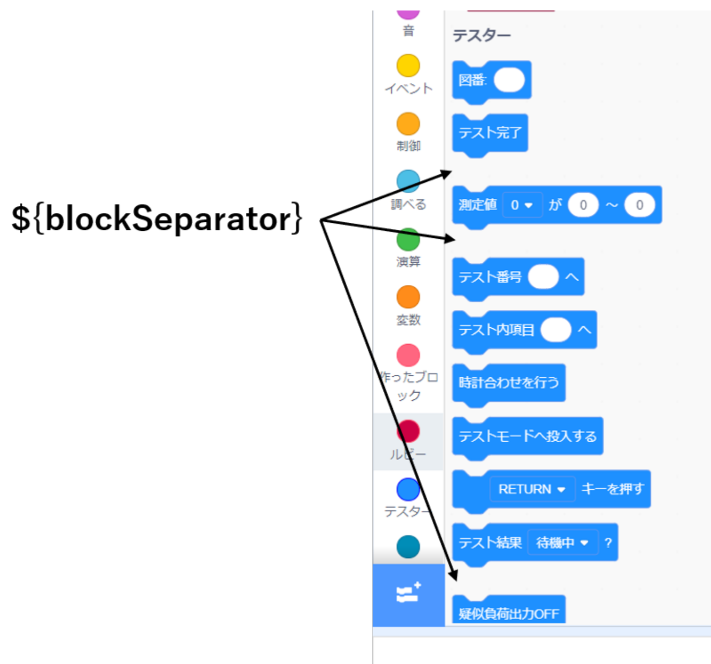
  テンプレートは以下のようなコードである

```
<block type="ブロックの名前 ">
           <value name="引数の name">
               //引数が数字の場合
               <shadow type="math_number">
                   <field name="NUM"></field>
               </shadow>
           </value>
           <value name="引数の name">
               //引数が文字の場合
               <shadow type="text">
                   <field name="TEXT"></field>
               </shadow>
           </value>
</block>
```

<block>~</block>タグ内に 1 つのブロックの見た目を書いていく

また、このように${blockSeparator}を記述することで、ブロックとブロックの間にすきまを開けることができる

```
<block>
     …
</block>
${blockSeparator}
<block>
     …
</block>
```



2.3 Smalruby3 の変更を Electron に反映させる

- コマンドプロンプトを開き、smalruby3-gui-SmrAI ディレクトリ内で以下のコマンドを実行すると、webpack によって smalruby3 をいくつかのファイルに圧縮することができる

```
>npm run build
```

- 圧縮されたファイルは smalruby3-gui-SmrAI/build に出力されるので、このディレクトリ内のファイル(static ディレクトリも含む)をすべて electron_SmrAI/app 直下にコピーすることで、開発内容を反映することができる

2.4 Smalruby3 の緑の旗からプログラムを実行させて結果を表示する

プログラムの実行は electron アプリ側で行う．そのため、electron アプリを形成するメインプロセス(アプリを制御する唯一のプロセス)とレンダラープロセス(画面描画用プロセス)間で IPC(プロセス間通信)を用いて、Smalruby3 の画面からプログラムを実行させる部分を実装する．IPC の実装は electron API に用意されている

- 緑の旗がクリックされたら、メインプロセスへメッセージを送る
smalruby3-gui-SmrAI¥src¥containers¥controls.jsx 内の handleGreenFlagClick メソッド内に以下のコードを書く

```
window.ipc.send('greenflag-click','Go');
```

window.ipc.send はレンダラープロセス→メインプロセスへメッセージを送るときに使う

メソッドである．定義は electron_SmrAI¥app¥preload.js に記述してある．
第 1 引数がチャンネル名、第 2 引数が送信内容である．上記のコードでは、レンダラープロセス上の greenflag-click チャンネルに Go という文字列を送っている．

- メインプロセスから Ruby プログラムを実行する
  electron_SmrAI¥app¥main.js 内に以下のコードを書く

```
ipcMain.on('greenflag-click', (event,arg) => {
}
```

これは、electron API で用意されている、レンダラープロセスから送られたメッセージを受け取るためのメソッドである．greenflag-click チャンネルに何か送られてきたら、{}内の処理を実行するというものである．
そして、その{}内に以下のコードを記述した．

```
exec('cd app/ruby & ruby test.rb', (error,stdout,stderr) => {
}
```

これは node.js API で用意されているプログラムやコマンドを実行するメソッドである．第 1 引数に指定したコマンドの実行が終了したら、{}内の処理が実行される．error には、コマンドの実行が失敗したときにエラーメッセージが格納される．stdout にはコマンドが実行された後の標準出力が格納される．stderr にはコマンドが実行された後の標準エラー出力が格納される．
このようにすることで、緑の旗がクリックされたら Ruby プログラムを実行するようになる．

- 実行結果をレンダラープロセスに送る
  上記の exec メソッドの{}内に

```
subWindow.webContents.send('exec-finish', stderr + stdout + "<br><<実行終了>>
");
```

これは、electron API で用意されている、メインプロセス→レンダラープロセスへメッセージを送るメソッドである．第 1 引数がチャンネル名、第 2 引数が送信内容である．
上記のコードでは、レンダラープロセス上の exec-finish チャンネルに stderr + stdout + "
<<実行終了>>"を送っている．

- 実行結果を表示する
  electron_SmrAI¥app¥render.js に以下のコードを記述した

```
window.ipc.on('exec-finish', (arg) => {
}
```

window.ipc.on はメインプロセスから送られてきたメッセージを受け取るメソッドである．定義は electron_SmrAI¥app¥preload.js に記述してある．
第 1 引数がチャンネル名、第 2 引数が受け取った内容である．上記のコードでは、メインプロセスから exec-finish チャンネルあてに何かメッセージが送られてきたら、{}内の処理を実行する．
そして、その{}内に以下のコードを書いた．

```
let result = arg.replace(/¥n/g,"<br>");
p.textContent += result;
p.textContent += "<br>";
div.innerHTML = p.textContent;
```

ここでは、レンダラープロセスが受けとった実行結果を HTML に成形して表示させている．

2.5 ログファイルの生成

ログファイル生成に必要な処理はすべて electron アプリのメインプロセス上で実行される

- ログファイルの置き場をつくる

  electron_SmrAI¥app¥main.js 内に記述した、exec メソッドの{}内に以下のコードを記述した．

```
if(!fs.existsSync('./log')) {
  fs.mkdirSync('log');
}
```

fs.existSync は node.js API で用意されている、ファイルもしくはフォルダが存在するかどうかを調べるメソッドである．
fs.mkdirSync は node.js API で用意されている、フォルダを作るメソッドである．このように書くことで、もし、log フォルダが存在しなかったら新しく log フォルダを作るようになる．

- ログファイルの生成

  上記の if 文の下に、以下のコードを記述した

```
time = new Date();
now = String(time.getFullYear())+String(time.getMonth()+1)+String(time.getDat
e());
fs.appendFile(`log/${now}_testlog.txt`,time+'¥n'+log+'¥n', (err, stdout) => {
      if(err) console.log(err);//ファイル書き込みに関するエラー
      else console.log('write end');
});
```

ここでは､ログファイルを生成した時刻を取得するために Date クラスを利用している．

fs.appendFile メソッドを使うことで第 1 引数に指定したファイルに、第 2 引数で指定した内容を書き込んでいる.

2.6 Electron アプリをパッケージングする方法

作成した electron アプリは以下のコマンドを入力することでパッケージングできる

```
>cd electron_SmrAI
> npx electron-packager ソースディレクトリ(app) 新しいアプリ名 --platform=プラットフォーム(x64) --arch=アーキテクチャ(win32) --version バージョン

  platform ・・・ all, linux, win32, darwin のいずれかを選択。
         「--all」は全部入りのパッケージング。
         「darwin」は mac のこと。複数選択はカンマ区切り。
  arch ・・・ all, ia32, x64 のいずれかを選択。
  version ・・・ Electron のバージョンを指定。(npx electron -v で確認)
```

なお、electron_SmrAI¥package.json に以下を記述してあるため、

```
"package": "electron-packager app SmT-2020 --platform=win32 --arch=x64 --electron-version=7.1.14"
```

以下のコマンドでも、パッケージングすることができる

```
>npm run package
```

パッケージング後のアプリ名やプラットフォーム、アーキテクチャ、electron バージョンを変える場合は、electron_SmrAI¥package.json の package を編集すればよい