[ID] エクステンションの開発手順

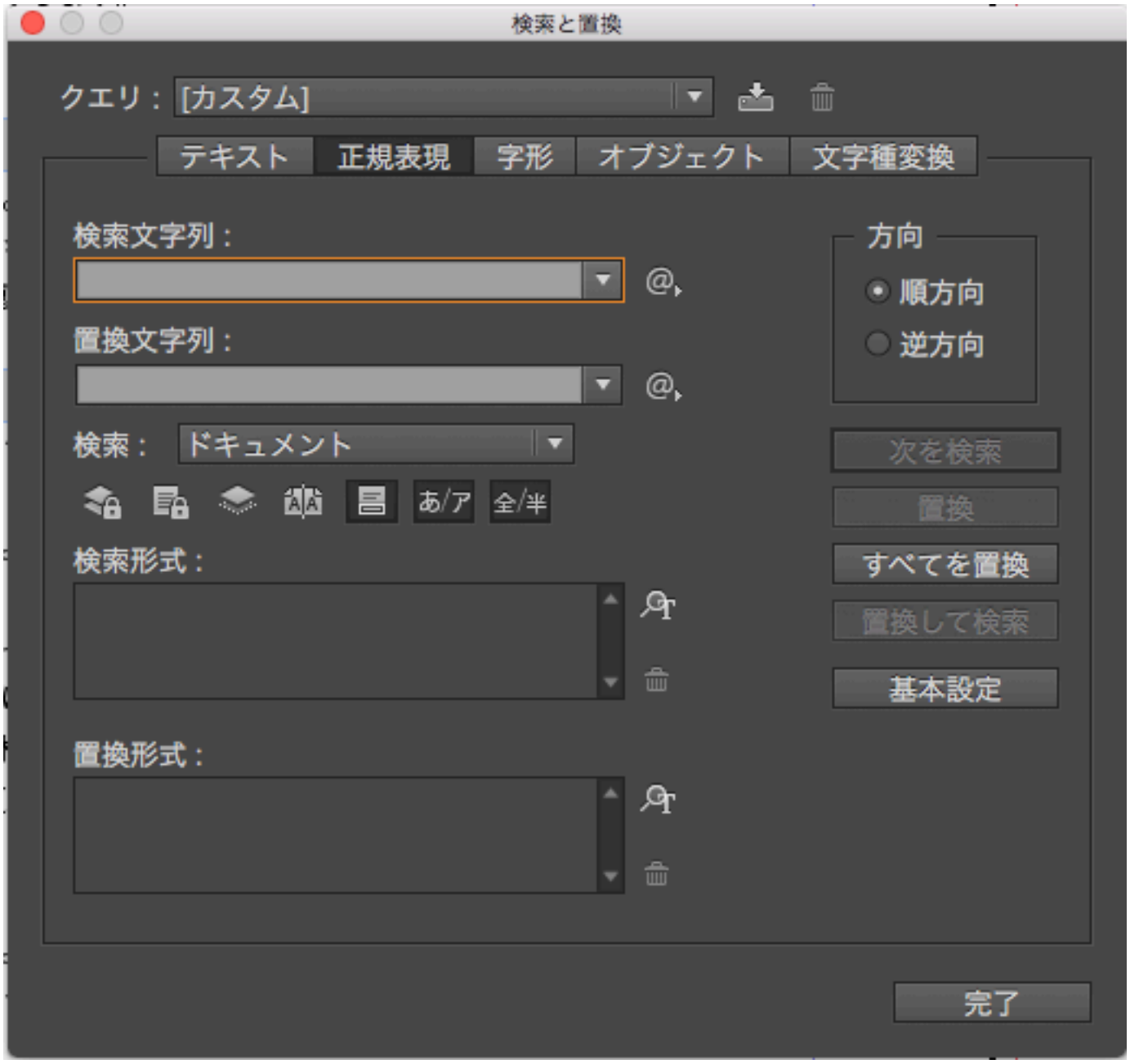
Posted by 10 A in アプリケーション自動化総合コミュニティフォーラム (Japan) on Mar 30, 2018 8:45:00 PM

```
01.
01.
```

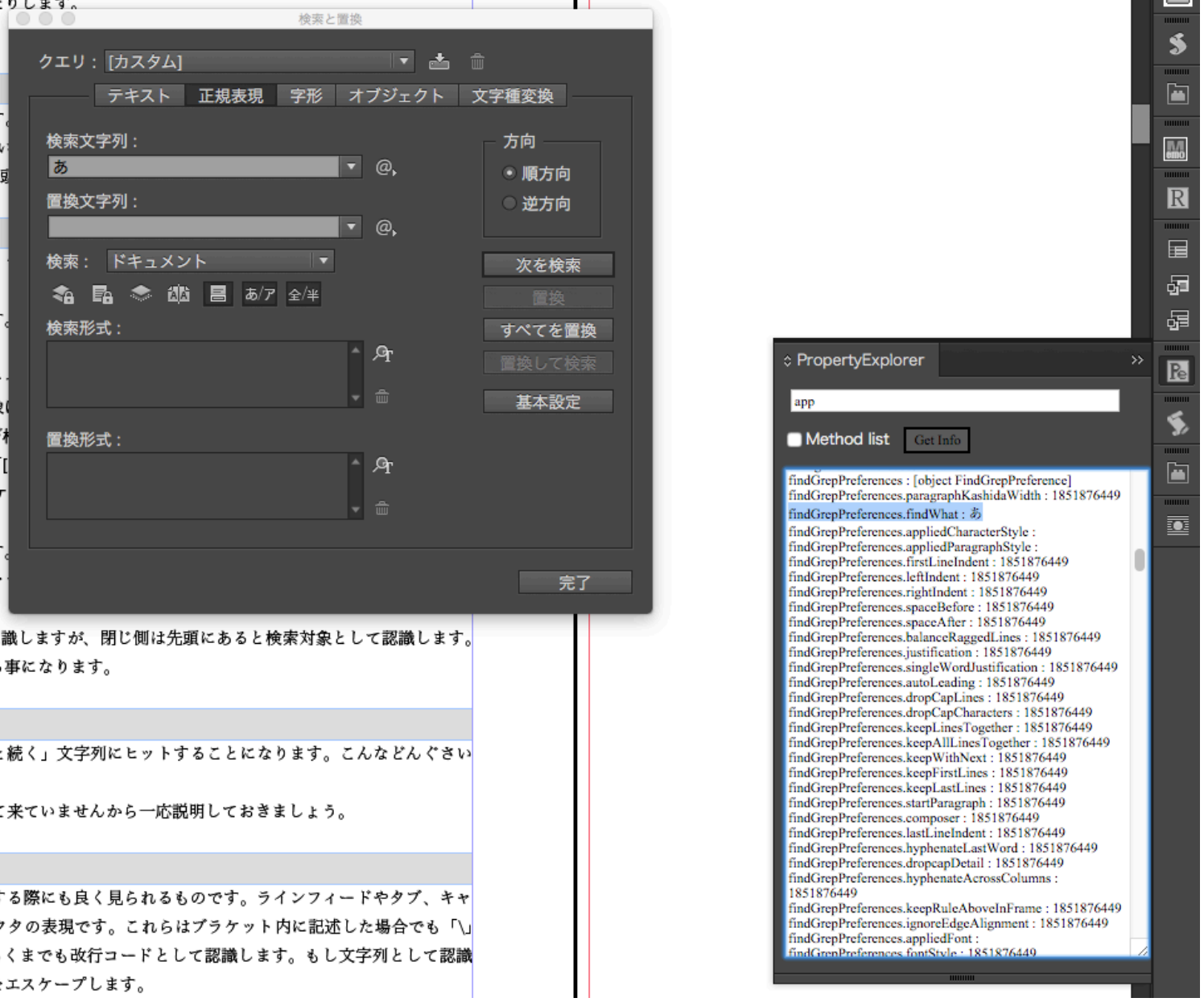
はじめに

事の発端はSNSでInDesignの作業中に自動検索置換スクリプトで途中マニュアルステップで一つ一つ確認しながら置換したいと言う声が聞こえてきた事でした。よくよく確認すると現状はス クリプトで一気に複数のキーワードを置換しているが中には慎重に作業を行う方が良いものもあってという事のようでした。
いちいち確認する必要があるなら標準装備の検索置換パネルで良いやんってなるのですが、確かにひとつずつ検索文字と置換文字を設定していくのは数が多い場合は手間かなあと思いました。で、どう するかなんですが、各文字をワンアクションで切り替えていくような仕組みはどうかと思った次第です。

基本となる処理について



こちら標準の検索置換パネルです。正規表現を基本にしますので、ここの検索文字列（正規表現）と対象文字列を切り替える手法から見ます。
上の検索置換パネルの各項目については基本的に各設定項目をスクリプトからコントロール可能です。
まず、検索の方から見てみます。検索置換パネルはアプリケーションの機能ですからドキュメントの切り換えによって内容が変わることがありません。この事からこれらのプロパティが**app**コレクション直下にあるであろうことが予想できます。では、実際にオブジェクトモデルを走査してみます。
こういう場合**PropertyExplorer**を利用すると手早く調べることが出来ます。

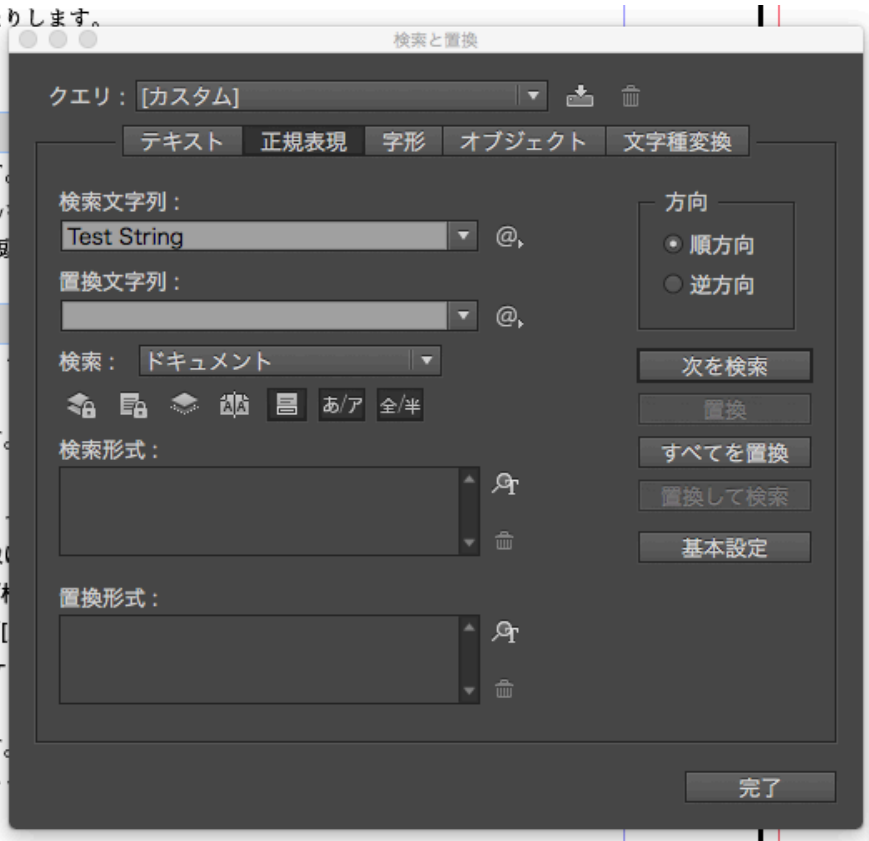


こちらのように検証対象を**app**としてその持つプロパティを全てリストアップします。結果の中に**findGrepPreferences**配下に**findWhat**というプロパティを見つけることができました。また、現在設定されている「あ」も見えているのが分かると思います。
では実際にこれを変えるスクリプトを書いてみます。

```
01. app.findGrepPreferences.findWhat = "Test String";
```


```
01. app.findGrepPreferences.findWhat = "Test String";
```


実際に実行してみます。



でも認識しますが、閉じ側は先頭にあると検索対象として認識します。
閉じる事になります。

def] と続く」文字列にヒットすることになります。こんなどんぐさい

だ出て来ていませんから一応説明しておきましょう。

この様に検索文字列入力欄を書き換える事が出来ました。ここでは**CodeEditor**エクステンションを利用してInDesign上で実行していますが**ESTK**を利用しても良いでしょう。
同様にして調べると置換対象文字列は**app.changeGrepPreferences.changeTo**である事が容易に調べられます。
以上のように**findGrepPreferences.findWhat**に正規表現を**changeGrepPreferences.changeTo**には置換文字列を当ててやればよいということが分かりました。また、グルーピングしたものはインターフェイス利用時同様に$を利用して取り出すことも可能です。
さて、ここまで調べるとあとはインターフェイスです。
今回は**CEP**を利用します。所謂Adobe Extension SDKってやつです。こちらは**HTML**で**UI**を作成しそのインターフェイスから **ExtendScript**の関数を実行する仕組みになっています。

要件の整理と仕様について

基本的な事が決定したので要件を書き出してみましょう。

1. 検索文字列（正規表現）は複数設定可能。
2. 置換文字列も検索文字列と同数を設定。
3. ユーザーのアクションで標準の検索置換パネルの検索文字列と置換文字列を切換える。

こう言った所です。
出来るだけインターフェイスを単純にする為に各文字列は複数行同時に入力できる**textarea**を利用します。また、基本的に一度置換処理を行うと再度実行する必要がありませんから、頭から順番に検索置換

文字列を切り替えていく動作とし、そのための**button**をひとつ用意します。

UIの構成

UIはHTMLとCSSで構成されます。今回のものは比較的単純な構成となります。まずは全容からご覧ください。

```
<!doctype html>
head>
<meta charset="utf-8">
<link rel="stylesheet" href="css/styles.css"/>
<link id="hostStyle" rel="stylesheet" href="css/theme.css"/>
<title>RE quick loader</title>
<script src="js/libs/CSInterface-4.0.0.js"></script>
<script src="js/libs/jquery-2.0.2.min.js"></script>
<script src="js/themeManager.js"></script>
<script src="js/main.js"></script>
</head>
<body class="ps-cs6">
<div class="clearfix">
   <div class="leftbx">
   <label for="findWhat">Find Words:</label><br />
      <textarea name="text1" id="findWhat" cols="20" rows="11" onchange="getwords()"></textarea>
   </div>
   <div class="leftbx">
   <label for="changeTo">Change Words:</label><br />
      <textarea name="text2" id="changeTo" cols="20" rows="11" onchange="getwords()"></textarea>
   </div>
   </div>
<div class="leftbx">
<button id="aply" onclick="aply()">load expression</button>
</div>
</body>
</html>
```

一般的なHTMLドキュメントと何ら変わりないことが分かると思います。また、bodyの構成を見て頂けると分かりますが、とても単純な構成です。

```
.leftbx {
    margin:5px;
     float:left;
     }


.clearfix:after {
```

```
  display: block;
  clear: both;
  content: "";
}
```

```
.leftbx {
    margin:5px;
     float:left;
     }


.clearfix:after {
  display: block;
  clear: both;
  content: "";
}
```

CSSの方もそう複雑なことはしていません。左寄せに２つのtextareaを横並びに配置してその下にボタンを置くための処理をfloatとclearを利用して構成しています。

UIのコントロール

HTML上の各form要素については**イベントを割り当て**、**textarea**には**内容が変更された時に各文字列を取得するための関数**を、**ボタン**には**クリックされた時に検索・置換文字列を切り換えする為の関数**を設定しています。
これらUI側のメインロジックは**main.js**に書かれています。

```
'use strict';
var csInterface = new CSInterface();
var counter = 0;
var fw = [], ct = [];


function init() {
themeManager.init();
}
init();


function getwords(){
fw = $("#findWhat").val().split("\n");
ct = $("#changeTo").val().split("\n");
counter = 0;
}


function aply(){
if (fw.length!=ct.length) return;
if (fw.length==counter) counter = 0;
csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
counter = counter + 1;
}
```

```
'use strict';
var csInterface = new CSInterface();
var counter = 0;
var fw = [], ct = [];
```

```
05.
06.
07.  function init() {
08.  themeManager.init();
09.  }
10.  init();
11.
12.
13.  function getwords(){
14.  fw = $("#findWhat").val().split("\n");
15.  ct = $("#changeTo").val().split("\n");
16.  counter = 0;
17.  }
18.
19.
20.  function aply(){
21.  if (fw.length!=ct.length) return;
22.  if (fw.length==counter) counter = 0;
23.  csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
24.  counter = counter + 1;
25.  }
```

機能を絞り込んでいるためコード自体は非常に短くなっています。頭から４行はエクステンショングローバルに各種インスタンスや変数を定義しています。
themeManagerは**ExtensionBuilder for Bracket**で追加されるアプリケーションテーマとエクステンションのアピアランスを同期させるためのライブラリです。アピアランスの同期についてはCSInterfaceクラスに盛り込まれていますから 自分で構成することも可能です。
では、UIに設定した各関数について解説しましょう。
まずは各textareaに設定されたgetwords関数です。

```
01.  function getwords(){
02.  fw = $("#findWhat").val().split("\n");
03.  ct = $("#changeTo").val().split("\n");
04.  counter = 0;
05.  }
```


```
01.  function getwords(){
02.  fw = $("#findWhat").val().split("\n");
03.  ct = $("#changeTo").val().split("\n");
04.  counter = 0;
05.  }
```


fw、**ct**それぞれに**textarea**に入力されたテキストを抜き出します。その際行ごとに分割して配列としています。この関数は双方の**textareaが編集されるたびに更新**されます。また、**counter**変数がリセットされるために**textareaを編集した場合は毎回一番最初の行から文字列を渡す**ことになります。
続いてaply関数です。

```
01.  function aply(){
02.  if (fw.length!=ct.length) return;
03.  if (fw.length==counter) counter = 0;
04.  csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
05.  counter = counter + 1;
06.  }
```


```
01.  function aply(){
02.  if (fw.length!=ct.length) return;
03.  if (fw.length==counter) counter = 0;
```

```
04.  csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
05.  counter = counter + 1;
06.  }
```

こちらは**ボタンをクリックした時にアプリケーションDOMに対して各文字列を渡す**役割をします。こちらは各行を細かく見ていきましょう。

```
01.  if (fw.length!=ct.length) return;
01.  if (fw.length!=ct.length) return;
```

入力データの検証を行っています。検索・置換双方の行数が合わない場合はデータを渡しません。

```
01.  if (fw.length==counter) counter = 0;
01.  if (fw.length==counter) counter = 0;
```

次はcounterが最後まで達している場合です。この場合、counterのみをリセットして再度最初の行から処理を行うようにしています。

```
01.  csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
01.  csInterface.evalScript('loadArgs("'+fw[counter]+'","'+ct[counter]+'")');
```

実際に**ExtendScriptへと情報をわたすにはここで使われるevalScriptと言う関数だけが利用可能**です。引数に**ExtendScript側の関数**と**それに利用する引数**を盛り込む為にクォート処理が複雑になりがち（ExtendScriptの関数名とそれに付随する引数をCEP側の変数と共にevalScriptの引数（String）として構成する必要が ある為）です。

```
01.  counter = counter + 1;
01.  counter = counter + 1;
```

最後にcounterをひとつ進めて一連の流れが終了します。

DOMの操作

ExtendScript側では先のevalScriptから実行される関数と引数を正しく受け取れる形で構成する必要があります。

```
01.  function loadArgs(fw, ct) {
02.  app.findGrepPreferences.findWhat = fw;
03.  app.changeGrepPreferences.changeTo = ct;
04.  }
01.  function loadArgs(fw, ct) {
02.  app.findGrepPreferences.findWhat = fw;
03.  app.changeGrepPreferences.changeTo = ct;
04.  }
```

DOMの操作に関しては調べた結果から実装はこの様になり、それぞれのプロパティにCEP側から受け取った引数を割り当てます。
以上で主要な部分の構成はおしまいです。その他編集が必要なものにmanifest.xmlがありますが、これについては以下を参照下さい。

CEP Extension Primer 2 Build up Projects 1 | CC Labo

関連情報

CEP関連の情報やサンプルエクステンションについては以下のページを一読しておくことをお勧めします。

https://forums.adobe.com/docs/DOC-8785

Adobe CEP · GitHub

また、今回のエクステンションの完全なソースは以下から入手可能です。

CreativeSuiteSDK_Experimentals/REQuickLoader at master · ten-A/CreativeSuiteSDK_Experimentals · GitHub

インストール可能なzxpパッケージは以下から入手可能です。

[IDエクステンション] REQuickLoader.zxp

以下はこのエクステンションの動画です。

REloader - YouTube

今回はエクステンションの制作課程を企画・リサーチから記述してみました。ある程度初歩的な部分から解説したつもりですが、ご要望等ありましたらお気軽にメッセージ入れて下さい。

1078 Views Tags : indesign, extension, cep

0 Comments