Contents

Overview

This cookbook is a guide to creating CEP 5.0 HTML/JavaScript Extensions for Creative Cloud applications.

CSXS is the old name before CS6, and CEP (Common Extensibility Platform) is new name from CS6. When we talk about CEP or CSXS, they refer to the same project.

CEP Extensions

CEP (formerly CSXS) Extensions extend the functionality of the host application in which they run. Extensions are loaded into applications through the PlugPlug Library architecture.

Starting from CEP 4.0, HTML/CSS and JavaScript (ECMAScript 5) can be used to develop extensions.

Extension Types

These extension types are supported by CEP. You need to specify an extension's type in its manifest.xml.

- Panel
    - The Panel type behaves like any other application panel. It can be docked, participates in workspaces, has fly-out menus, and is re-opened at start-up if open at shutdown.
- ModalDialog
    - A Modal Dialog type opens a new window and forces the user to interact with the window before returning control to the host application.
- Modeless
    - A Modeless Dialog type opens a new window but doesn't force the user to interact with the host application.
- Custom (Since CEP 5.0)
    - This type is for invisible extensions. An invisible extension remains hidden and never becomes visible during its whole life cycle. Read "Invisible HTML Extensions" for more details.

Applications Integrated CEP

These applications support CEP HTML extensions.

Appication	Host ID	CC Version	CC 2014 Version	CC 2015 Version	CC 2016 Version
Photoshop	PHSP	14	15	16	17
Photoshop Extended	PHXS	14	15	16	17
InDesign	IDSN	9	10	11	11
InCopy	AICY	9	10	11	11
Illustrator	ILST	17	18	19	20
Premiere Pro	PPRO	7	8	9	10
Prelude	PRLD	2	3	4	5
After Effects	AEFT	N/A	13	13	13
Flash Pro	FLPR	13	14	15	15
Audition	AUDT	N/A	N/A	8	9
Dreamweaver	DRWV	N/A	N/A	16	16

Chromium Embedded Framework (CEF)

CEP HTML engine is based on Chromium Embedded Framework version 3 (CEF3). You can find more information about CEF on http://code.google.com/p/chromiumembedded/. Here are the versions:

	CEP 5.2	**CEP 6.1 and CEP 7.0**
CEF 3	CEF 3 branch 1453, revision 1339	CEF 3 release branch 2272 Commit e8e1f98ee026a62778eb2269c8e883426db645ea
Chromium	27.0.1453.110	41.0.2272.104
Node.js	Node.js 0.8.22	IO.js 1.2.0
CEF/Node integration	cefnode ?	Node-WebKit 0.12.1 (nw.js)

Browser Features supported by CEP

HTTP Cookie

CEP supports two kinds of cookies:

- Session Cookies - Temporary in-memory cookie which will expire and disappear when user closes extension
- Persistent Cookies - No expiry date or validity interval, stored in user's file system

Persistent Cookies location:

- CEP 4.x
    - Windows: C:\Users\USERNAME\AppData\Local\Temp\cep_cookies\
    - Mac: /Users/USERNAME/Library/Logs/CSXS/cep_cookies/
- CEP 5.x
    - Windows: C:\Users\USERNAME\AppData\Local\Temp\cep_cache\
    - Mac: /Users/USERNAME/Library/Logs/CSXS/cep_cache/
- CEP 6.x and later releases
    - Windows: C:\Users\USERNAME\AppData\Local\Temp\cep_cache\
    - Mac: /Users/USERNAME/Library/Caches/CSXS/cep_cache/

Each persistent cookie is a file. File name is HostID_HostVersion_ExtensionName, such as PHXS_15.0.0_com.adobe.extension1.

Development and Debugging

Development Machine Setup

CEP HTML Extensions can be developed on both Windows and Mac platforms. The development machine needs to have the following applications in order to successfully develop CSXS extensions:

- Adobe Creative Suite applications supporting CEP HTML extensions.
- HTML Extension Builder (Nice to have, but not mandatory).
- Adobe ExtendScript Tool Kit (This tool kit is installed with all Creative Suite applications).
- Adobe Extension Manager.

HTML Extension Builder

HTML Extension Builder (under development) is a tool set built on top of Eclipse and can be used for developing and debugging HTML extensions. Please download the Extension Builder 3 [here](#).

Signing extensions

- Before you sign the extensions, you need to get or create the certificate file. Configurator and Adobe Exchange Packer can create certificates. Developers can get all information [here](#) after logging in.

- Three tools can be used to sign a HTML extension.

  1. [Extension Builder 3](#)
  2. CC Extensions Signing Toolkit (also on above labs web site)
     - Example of using CC Extension signing toolkit: ccextensionswin64.exe -sign "d:\Adobe Layer Namer\Adobe Layer Namer\"*(input extension path)* d:\AdobeLayerNamer.zxp*(output zxp path)* d:\sign.p12*(certificate path)* 1*(certificate password)*
  3. [Adobe Exchange Packer](#) (please sign in so that you can see it.)

Debugging Unsigned Extensions

If you are in the midst of development and are not using HTML Extension Builder for debug workflows and want to bypass the need to sign your extensions, you can bypass the check for extension signatures by editing the CSXS preference properties file, located at:

- Win: regedit > HKEY_CURRENT_USER/Software/Adobe/CSXS.7, then add a new entry **PlayerDebugMode** of type "string" with the value of "1".
- Mac: In the terminal, type: defaults write com.adobe.CSXS.7 PlayerDebugMode 1 (The plist is also located at /Users/<username>/Library/Preferences/com.adobe.CSXS.7.plist)

These entries will enable debug extensions to be displayed in the host applications.

Special notes for Mac 10.9 and higher

Staring with Mac 10.9, Apple introduced a caching mechanism for plist files. Your modifications to plist files does not take effect until the cache gets updated (on a periodic basis, you cannot know exactly when the update will happen). To make sure your modifications take effect, there are two methods.

- Kill **cfprefsd** process. It will restart automatically. Then the update takes effect.
- Restart your Mac, or log out the current user and re-log in.
- [More Information](#)

Remote Debugging

CEP supports remote debugging for HTML extensions from 5.0.

- Create a ".debug" file to the extension root directory such as Test_Extension\.debug. The .debug file contains remote debug ports. Developers must create this file and use valid debug ports because both remote debugging and dev tools are based on it.
- ".debug" file name is special for both Windows and Mac platforms, it has to be created via command line.
  - On Windows, use "copy con .debug" and "Ctrl+Z" to create an empty file.
  - On Mac, use "touch .debug" to create an empty file.
- The value of Port should be between 1024 and 65535 (not include 65535), otherwise remote debugging and dev tools will not work.

- One extension bundle may have multiple extensions. The .debug file can specify debug ports for each extension. Here is an example file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ExtensionList>
    <Extension Id="com.adobe.CEPHTMLTEST.Panel1">
        <HostList>
            <Host Name="PHXS" Port="8000"/>
            <Host Name="IDSN" Port="8001"/>
            <Host Name="AICY" Port="8002"/>
            <Host Name="ILST" Port="8003"/>
            <Host Name="PPRO" Port="8004"/>
            <Host Name="PRLD" Port="8005"/>
            <Host Name="FLPR" Port="8006"/>
```

```xml
                    <Host Name="AUDT" Port="8007"/>
            </HostList>
        </Extension>
        <Extension Id="com.adobe.CEPHTMLTEST.Panel2">
            <HostList>
                <Host Name="PHXS" Port="8100"/>
                <Host Name="IDSN" Port="8101"/>
                <Host Name="AICY" Port="8102"/>
                <Host Name="ILST" Port="8103"/>
                <Host Name="PPRO" Port="8104"/>
                <Host Name="PRLD" Port="8105"/>
                <Host Name="FLPR" Port="8106"/>
                    <Host Name="AUDT" Port="8107"/>
            </HostList>
        </Extension>
        <Extension Id="com.adobe.CEPHTMLTEST.ModalDialog">
            <HostList>
                <Host Name="PHXS" Port="8200"/>
                <Host Name="IDSN" Port="8201"/>
                <Host Name="AICY" Port="8202"/>
                <Host Name="ILST" Port="8203"/>
                <Host Name="PPRO" Port="8204"/>
                <Host Name="PRLD" Port="8205"/>
                <Host Name="FLPR" Port="8206"/>
                    <Host Name="AUDT" Port="8207"/>
            </HostList>
        </Extension>
        <Extension Id="com.adobe.CEPHTMLTEST.Modeless">
            <HostList>
                <Host Name="PHXS" Port="8300"/>
                <Host Name="IDSN" Port="8301"/>
                <Host Name="AICY" Port="8302"/>
                <Host Name="ILST" Port="8303"/>
                <Host Name="PPRO" Port="8304"/>
                <Host Name="PRLD" Port="8305"/>
                <Host Name="FLPR" Port="8306"/>
                    <Host Name="AUDT" Port="8307"/>
            </HostList>
        </Extension>
    </ExtensionList>
```

If you load an extension whose debug port is 8088, you can load the debugger through http://localhost:8088/ on Chrome.

Known Issues

- If you attempt to launch an extension using the requestOpenExtension API and the extension invoking the API has the same debug port as the target extension, the target extension will not load.
- Due to an issue which should be fixed in Webkit inspector, in the dev tools, you cannot inspect the variables or open watches. Please perform the following steps to work around this issue when it happens:

1. Open the Developer tools from Chrome browser main menu by choosing View->Developer->Developer Tools
2. Execute the following JS snippet in the Console

```
TreeElement.prototype.isEventWithinDisclosureTriangle =
function(event)
{
    var computedLeftPadding = 10;
    if(window.getComputedStyle(this._listItemNode).getPropertyCSSV
alue){
            computedLeftPadding =
    window.getComputedStyle(this._listItemNode).getPropertyCSSValue("
    padding-left").getFloatValue(CSSPrimitiveValue.CSS_PX);
        }

        var left = this._listItemNode.totalOffsetLeft() +
computedLeftPadding;
        return event.pageX >= left && event.pageX <= left +
this.arrowToggleWidth && this.hasChildren;
    }
```

- In CEP 6.1, inline JavaScript source maps will cause the debugging session to be terminated. To work around the issue, either remove the inline JavaScript source maps, or use source maps reference.
- [Can't scroll in Devtools console](#)
  To work around this, either use cefclient released by CEP for debugging, or use below workaround in Chrome/Chromium
    - Open Chrome/Chromium and connect to your extension's debugging port, say localhost:8088
    - Click the drop down menu on the upper-right corner on the Chrome/Chromium window, then click "More tools"->"Developer tools"
    - In the Console of the newly opened devtools panel, run:

```
(function()
{
    var styleElement = document.createElement("style");
    styleElement.type = "text/css";
    styleElement.textContent = "html /deep/ * { min-width: 0; min-
height: 0; }";
    document.head.appendChild(styleElement);
})();
```

Where are the Log Files

PlugPlug Logs

Log files with useful debugging information are created for each of the applications supporting CEP extensions. The platform-specific locations for the log files are as follows:

- Win: C:\Users\USERNAME\AppData\Local\Temp

- Mac: /Users/USERNAME/Library/Logs/CSXS

These files are generated with the following naming conventions:

- CEP 4.0 - 6.0 releases: csxs<versionNumber>-<HostID>.log. For example, PlugPlug in Illustrator generates log file csxs6-ILST.log.
- CEP 6.1 and later releases: CEP<versionNumber>-<HostID>.log. For example, PlugPlug in Illustrator generates log file CEP6-ILST.log.

Logging levels can be modified as per the following levels:

- 0 - Off (No logs are generated)
- 1 - Error (the default logging value)
- 2 - Warn
- 3 - Info
- 4 - Debug
- 5 - Trace
- 6 - All

The **LogLevel** key can be updated at the following location (The application should be restarted for the log level changes to take effect):

- Win: regedit > HKEY_CURRENT_USER/Software/Adobe/CSXS.7
- Mac: /Users/USERNAME/Library/Preferences/com.adobe.CSXS.7.plist

For example of Mac, in the terminal do:

```
defaults write com.adobe.CSXS.7 LogLevel 6
```

CEPHtmlEngine Logs

In CEP 6.1 and later releases, CEPHtmlEngine generates logs. Each CEPHtmlEngine instance usually genereate two log files, one for browser process, the other for renderer process.

These files are generated with the following naming conventions:

- Browser process: CEPHtmlEngine<versionNumber>-<HostID>-<HostVersion>-<ExtensionID>.log
- Renderer process: CEPHtmlEngine<versionNumber>-<HostID>-<HostVersion>-<ExtensionID>-renderer.log

For example:

- CEPHtmlEngine6-PHXS-16.0.0-com.adobe.DesignLibraries.angular.log
- CEPHtmlEngine6-PHXS-16.0.0-com.adobe.DesignLibraries.angular-renderer.log

They are also controlled by the PlugPlug log level.

CEF Log

In CEP 4.0 - 6.0, the Chromium Embedded Framework (CEF) in CEPHtmlEngine also generates a log:

- Win: C:\Users\USERNAME\AppData\Local\Temp\cef_debug.log
- Mac: /Users/USERNAME/Library/Logs/CSXS/cef_debug.log

In CEP 6.1 and later releases, this log is merged into CEPHtmlEngine log.

Extension Folders

CEP supports 3 types of extension folders.

- Product extension folder. Here is a suggestion, but each point product can decide where this folder should be.
  - ${PP}/CEP/extensions (PPs may use different folder.)
- System extension folder
  - Win(x86): C:\Program Files\Common Files\Adobe\CEP\extensions
  - Win(x64): C:\Program Files (x86)\Common Files\Adobe\CEP\extensions, and C:\Program Files\Common Files\Adobe\CEP\extensions (since CEP 6.1)
  - Mac: /Library/Application Support/Adobe/CEP/extensions
- Per-user extension folder
  - Win: C:\Users\{USER}\AppData\Roaming\Adobe\CEP/extensions
  - Mac: ~/Library/Application Support/Adobe/CEP/extensions

How does CEP decide which extension to load?

- CEP first searches the product extension folder, then the system extension folder, and finally per-user extension folder.
- Extensions without an appropriate host application ID and version are filtered out.
- If two extensions have same extension bundle ID, the one with higher version is loaded.
- If two extensions have same extension bundle ID and same version, the one whose manifest file has latest modification date is loaded.
- If two extensions have same extension bundle ID, same version and same manifest modification date, CEP loads the first one that is found.

Extension Installation:

- Point product installers should install extensions to product extension folder.
- Extension Manager and Exchange Plugin in Thor should install extensions to system extension folder or per-user extension folder.

Note:

- Character '#' is not allowed in extension folder path on both Windows and Mac OSX, since CEF treats '#' as a delimiter.

Extension Manifest

The manifest.xml file is required for every extension and provides the necessary information to configure a CEP extension. ExtensionManifest is the root element of the manifest.xml file. Extensions, ExtensionList, and DispatchList are the three child elements of the ExtensionManifest root element.

Extension Manifest XSD

All HTML extensions must use 5.0 or above version. XSD attached in this file.

To check if the extension's manifest is in sync with the latest schema, perform the following steps:

1. Download the latest schema (ExtensionManifest_<version>.xsd).
2. Navigate [here](#)
3. Upload the schema and your latest ExtensionManifest (from a real build to check the validity of the versions).
4. Hit validate

Important Manifest Change for CEP 5.0 Extensions

Make sure correct point product versions are used. Here is an example.

```
<HostList>
    <Host Name="PHXS" Version="[15.0,15.9]"/>
    <Host Name="PHSP" Version="[15.0,15.9]"/>
</HostList>
```

This will support Photoshop version 15.0 up to, and including, 15.9. If you use the following syntax then you are supporting releases up to 15.9 but not including 15.9

```
<HostList>
    <Host Name="PHXS" Version="[15.0,15.9)"/>
    <Host Name="PHSP" Version="[15.0,15.9)"/>
</HostList>
```

Make sure correct CEP version is used.

```
<RequiredRuntimeList>
    <RequiredRuntime Name="CSXS" Version="5.0"/>
</RequiredRuntimeList>
```

Important Manifest Change for CEP 7.0 Extensions

CEP 7.0 manifest files now support the use of a HostList specific to an extension in the bundle. For example, the following qualifies as the "default" HostList:

```
...
<ExecutionEnvironment>
  <HostList>
    <Host Name="DRWV" Version="15.0" />
    <Host Name="FLPR" Version="15.0" />
    <Host Name="IDSN" Version="11.0 "/>
    <Host Name="AICY" Version="11.0" />
    <Host Name="ILST" Version="19.0" />
    <Host Name="PHSP" Version="16.0" />
    <Host Name="PHXS" Version="16.0" />
    <Host Name="PPRO" Version="9.0" />
    <Host Name="PRLD" Version="4.0" />
    <Host Name="AEFT" Version="13.0" />
    <Host Name="DEMO" Version="1.0" />
    <Host Name="AUDT" Version="8.0" />
    <Host Name="LTRM" Version="7.0" />
    <Host Name="MUSE" Version="2015" />
  </HostList>
  ...
</Execution Environment>
...
```

However, you can also specify a custom HostList on a per extension basis as in the following example:

```
...
<DispatchInfoList>
  ...
  <Extension Id="com.adobe.CEPHTMLTEST.Panel1">
    <HostList>
      <Host Name="PHXS" />
    </HostList>
    ...
  </Extension>
  <DispatchInfo>
  ...
  </DispatchInfo>
  ...
</DispatchInfoList>
```

HostLists specified under extension tags will override the default HostList specified under the execution environment tag. It is important to note that adding HostList nodes with no child nodes is akin to turning that particular extension off for all host applications; this behavior is intentional. Please refrain from specifying both HostList tags and Host attributes in DispatchInfo tags for each extension; choose one or the other. Specifying both will more than likely result in unexpected behavior. The Host attribute is only maintained for backward compatibility. New extensions should use HostList tags, not the Host attribute.

Extension Size

You can specify extension size, max size and min size in extension manifest. Size is mandatory; max size and min size are optional.

A modal or modeless dialog is resizable if there are max size and min size, otherwise it is un-resizable. When you move mouse pointer over dialog border, CEP shows different cursor for resizable and un-resizable dialogs.

```
<Geometry>
    <Size>
        <Height>580</Height>
        <Width>1000</Width>
    </Size>
    <MaxSize>
        <Height>800</Height>
        <Width>1200</Width>
    </MaxSize>
    <MinSize>
        <Height>400</Height>
        <Width>600</Width>
    </MinSize>
</Geometry>
```

Customize Extension Menu

This is only supported in InDesign and InCopy.

You can customize the extension menu by editing <menu/> item in manifest. Here is an example. In this example, the Adobe Add-ons extension is displayed under Windows main menu, rather than extensions menu under Windows. You can customize the location of extension to somewhere else by changing the value of attribute Placement in <menu/> item.

```
<?xml version="1.0" encoding="UTF-8"?>
<ExtensionList>
    <Extension Id="Adobe Add-ons" Version="1.0"/>
</ExtensionList>
<ExecutionEnvironment>
    <HostList>
        <Host Name="IDSN" Version="8.0"/>
    </HostList>
    ...
</ExecutionEnvironment>
<DispatchInfoList>
    <Extension Id="com.adobe.CEPHTMLTEST.Panel1">
        <DispatchInfo>
            ...
            <UI>
              ...
             <Menu Placement="'Main:&amp;Window',600.0,'KBSCE Window
menu'">Adobe Add-ons</Menu>
...
```

High DPI Panel Icons

In high DPI display mode, panel extensions may want to use high DPI icons. You set these icons in extension's manifest.

```
<Icons>

 <Icon Type="Normal">./images/IconLight.png</Icon>

 <Icon Type="RollOver">./images/IconLight.png</Icon>

 <Icon Type="DarkNormal">./images/IconDark.png</Icon>

 <Icon Type="DarkRollOver">./images/IconDark.png</Icon>

</Icons>
```

You pack both normal icon files (IconLight.png and IconDark.png) and high DPI icon files (IconLight@2X.png and IconDark@2X.png) in your extension.

Host applications will be able to find and use

- IconLight.png and IconDark.png for normal display
- IconLight@2X.png and IconDark@2X.png for 200% high DPI display

@2X.ext is the industry standard. Please see more details on https://developer.apple.com/library/ios/qa/qa1686/_index.html.

Note: Photoshop supports **_x2.ext** format.

Dialog Size based on Screen Size

You can specify CEP dialog size as a percentage of screen size. Here is an example.

```
<UI>
    <Type>Modeless</Type>
    ...
    <Geometry>
        <ScreenPercentage>
            <Height>50%</Height>
            <Width>50%</Width>
        </ScreenPercentage>
        ...
    </Geometry>
</UI>
```

Shortcut Keys for HTML Extensions

(Since 5.2)

CEP 5.2 supports shortcut keys for HTML extensions. When focus is on HTML extensions, these shortcut keys are handled by extension.

Windows Keys	Mac Keys	Function
Ctrl + A	**Command + A**	Select All
Ctrl + C	**Command + C**	Copy
Ctrl + V	**Command + V**	Paste
Ctrl + X	**Command + X**	Cut

Other shortcut keys are handled by point products, such as pressing Ctrl + N to create a new document.

CEP JavaScript Programming

CEP JavaScript Libraries

CEP JavaScript Libraries are counterparts of the Flex CSXS Library and the CEP IMS Library. They provide JavaScript APIs to access application and CEP information.

- CSInterface.js
- Vulcan.js

To use them, please include these JavaScript files in your HTML extension.

API version

The CEP JavaScript APIs keep changing in each new CEP release. The changes are guaranteed to be backward compatible. For newly added APIs, a version tag like "Since x.x.x" is added to its API comments indicating since which CEP version the APIs is available.

You will need to check the version tag against the version of CEP integrated by the Adobe Product you are using to make sure the API you want to use is available. To do so, use CSInterface.getCurrentApiVersion() to retrieve the version of CEP integrated by the Adobe Product. Please note this API itself is available only since 4.2.0. If you get an error saying getCurrentApiVersion is undefined, then you are running in CEP 4.0 or 4.1. Otherwise, the value returned will tell you the version of CEP integrated by the Adobe product.

CEP Events

CEP supports sending and receiving events within an extension, among extensions in an application, and among extensions in different applications. Since both Flash and HTML extensions are based on the common communication layer with the same event data format, they can communicate with each other through CEP events, and even they can communicate with

native side as long as point products invoke PlugPlugAddEventListener/PlugPlugDispatchEvent accordingly.

Like what we have for Flash extensions, there are **dispatchEvent/addEventListener/removeEventListener** APIs available in JavaScript to dispatch and listen for events. Let's go through CEP event data format/structure, APIs to dispatch and listen event, and sample code snippet accordingly in JavaScript.

CSEvent

In terms of CSEvent, it just means CEP Event here. The data structure of CSEvent (CEP Event) in JavaScript is just the same as the one defined in Flash extension, as below.

```
/**
 * Class CSEvent.
 * You can use it to dispatch a standard CEP event.
 *
 * @param type            Event type.
 * @param scope           The scope of event, can be "GLOBAL" or "APPLICATION".
 * @param appId           The unique identifier of the application that
generated the event. Optional.
 * @param extensionId The unique identifier of the extension that generated
the event. Optional.
 *
 * @return CSEvent object
 */
function CSEvent(type, scope, appId, extensionId)
{
    this.type = type;
    this.scope = scope;
    this.appId = appId;
    this.extensionId = extensionId;
};
```

You could create a CSEvent object and dispatch it by using CSInterface.dispatchEvent. Also you could access its property in your callback of CSInterface.addEventListener. Refer to the section addEventListener/dispatchEvent below for more details.

Listen for and Dispatch CSEvent

dispatchEvent/addEventListener/removeEventListener APIs are available in JavaScript world to dispatch and listen for CSEvent.

addEventListener

Here is the definition for addEventListener. Refer to CSInterface.js for more information:

```
 /**
 * Registers an interest in a CEP event of a particular type, and
 * assigns an event handler.
```

```
 * The event infrastructure notifies your extension when events of this type
occur,
 * passing the event object to the registered handler function.
 *
 * @param type     The name of the event type of interest.
 * @param listener The JavaScript handler function or method.
 * @param obj      Optional, the object containing the handler method, if
any.
 *          Default is null.
 */
CSInterface.prototype.addEventListener = function(type, listener, obj)
```

One thing needs to be mentioned here is both named and anonymous callback functions are supported in CSInterface.addEventListener.

- An example of how to use named callback function in CSInterface.addEventListener.
- ```
  function callback(event)
  ```
- ```
  {
  ```
- ```
      console.log("type=" + event.type + ", data=" + event.data);
  ```
- ```
  }
  ```
-
- ```
  var csInterface = new CSInterface();
  ```
- ```
  csInterface.addEventListener("com.adobe.cep.test", callback); //invoke
  the function
  ```

- An example of how to use anonymous callback function in CSInterface.addEventListener.
- ```
  var csInterface = new CSInterface();
  ```
- ```
  csInterface.addEventListener("com.adobe.cep.test", function (event)
  ```
- ```
      {
  ```
- ```
          console.log("type=" + event.type + ", data=" + event.data);
  ```
- ```
      }
  ```
- ```
  ); // Anonymous function is the second parameter
  ```

Similarly in Flash, event.data can be an object (i.e. you could use an object as event.data).

Before CEP 6.1, we regarded every attribute in event.data object as a regular string, but from CEP 6.1, we revised the behavior that keep the type of each attribute in event.data as it was. If the value is a valid JSON string, CEP will parse it natively and convert it to an object.

- Here is an example on how to use it.

- ```
  var csInterface = new CSInterface();
  ```
- ```
  csInterface.addEventListener("com.adobe.cep.test", function (event)
  ```
- ```
      {
  ```
- ```
          var obj = event.data;
  ```
- ```
          console.log("type=" + event.type + ", data.property1=" +
  obj.property1 + ", data.property2=" + obj.property2);
  ```
- ```
      }
  ```
- ```
  ); // Anonymous function is the second parameter
  ```

dispatchEvent

Here is the definition for CSInterface.dispatchEvent. Refer to CSInterface.js for more details.

```
/**
 * Triggers a CEP event programmatically. Yoy can use it to dispatch
 * an event of a predefined type, or of a type you have defined.
 *
 * @param event A \c CSEvent object.
 */
CSInterface.prototype.dispatchEvent = function(event)
```

Here are three samples to demonstrate how to dispatch an event in JavaScript.

- An example of how to dispatch event in JavaScript.
- `var csInterface = new CSInterface();`
- `var event = new CSEvent("com.adobe.cep.test", "APPLICATION");`
- `event.data = "This is a test!";`
- `cSInterface.dispatchEvent(event);`
-
- Another example of creating event object and setting property, then dispatch it.
- `var csInterface = new CSInterface();`
- `var event = new CSEvent();`
- `event.type = "com.adobe.cep.test";`
- `event.scope = "APPLICATION";`
- `event.data = "This is a test!";`
  `cSInterface.dispatchEvent(event);`

- An example of dispatching an event whose data is an object.
- `var event = new CSEvent("com.adobe.cep.test", "APPLICATION");`
- `var obj = new Object();`
- `obj.a = "a";`
- `obj.b = "b";`
- `event.data = obj;`
  `cSInterface.dispatchEvent(event);`

Communication between Flash and HTML extensions

CEP event based communication between Flash and HTML extensions is simple, as long as the event data is string. You can use the APIs to dispatch and listen to events accordingly.

If you dispatch an event whose data is an object in Javascript and handle it in Flash, then you need a little bit conversion work to do in Flash extension. Because JavaScript objects are serialized JSON strings, you need to deserialize them to XML-based objects in Flash. Vice versa, if you dispatch an event whose data is an object in Flash and handle it in JavaScript, you need to deserialize it from XML to a JSON-based object in JavaScript side. Considering that usage of

Flash extensions has gone down, communication between Flash and CEP extensions is limited, and object-based event data is uncommon.

Handling Window State Change Events - Extensions

Unlike Flash extensions, CEP extensions do not support Window State Change Events.

Standard Events in Point Products

Following table lists the standard events supported by Point Products now:

(✅ = supported, ❌ = not supported)

Event Type	Event Scope	Description	Event Parameter	PS	ID	AI	FL	PR	PL	AU
documentAfterActivate	APPLICATION	Event fired when a document has been activated (after new/open document; after document has retrieved focus).	URL to the active document. If the doc was not save, the NAME will be set instead of the URL.	❌	✅	✅	❌	❌	❌	✅
documentAfterDeactivate	APPLICATION	Event fired when the active document has been de-activated. (after document loses focus)	URL to the active document. If the doc was not save, the name will be set instead of the URL.	✅	✅	✅	❌	❌	❌	✅

applicationBeforeQuit	APPLICATION	Event fired when the application got the signal to start to terminate.	none	✗ ⚠ ⚠ ✗ ✗	✗	✗
applicationActivate	APPLICATION	Event fired when the Application got an "activation" event from the OS.	none	✓ ✓ ✓ ✗	✓ on Mac; ✗ on Windows	✓ on Mac; ✗ on Windows	✓
documentAfterSave	APPLICATION	Event fired after the document has been saved	URL to the saved document.	✓ ✓ ✓ ✗ ✗	✗	✓

Note: CEP is unloaded from the point products ID and AI right after the event applicationBeforeQuit is emitted from the point products, therefore CEP may have no chance to get this event handled in HTML extensions.

Specific Events in Products

Photoshop

In Photoshop, the following specific events are defined:

- com.adobe.PhotoshopPersistent
- com.adobe.PhotoshopUnPersistent
- ~~com.adobe.PhotoshopCallback~~ => This event will be removed in Photoshop 17.0 (see below)
- com.adobe.PhotoshopWorkspaceSet
- com.adobe.PhotoshopWorkspaceGet
- com.adobe.PhotoshopWorkspaceAware
- com.adobe.PhotoshopWorkspaceData
- com.adobe.PhotoshopWorkspaceRequest
- com.adobe.PhotoshopRegisterEvent
- com.adobe.PhotoshopUnRegisterEvent
- com.adobe.PhotoshopLoseFocus
- com.adobe.PhotoshopQueryDockingState

For example, a CEP extension yields the mouse focus back to Photoshop by sending the com.adobe.PhotoshopLoseFocus event:

```
var csInterface = new CSInterface();
var event = new CSEvent("com.adobe.PhotoshopLoseFocus", "APPLICATION");
event.extensionId = csInterface.getExtensionID();
csInterface.dispatchEvent(event);
```

com.adobe.PhotoshopCallback will be removed in Photoshop 17.0 as adding a listener results in all CS Extensions receiving the event. As of Photoshop CC 2015 June release, developers can now use this alternative, which fixes the broadcast issue:

```
csInterface.addEventListener("com.adobe.PhotoshopJSONCallback" +
gExtensionID, PhotoshopCallbackUnique);
```

TBD: add more examples.

Better JSON Support in CEP Event JavaScript APIs

Defects in CEP 6.0 and Former Releases

- CEP 6.0 treats each attribute of the event.data object as a string. For example, if you pass the string below through a CEP event:

  ```
  {"myBoolKey": false,"myIntKey": 7,"myFloatKey": 5.4,"myStringKey":
  "testテスト测试","myArrayKey": [5.4,true,false,{"yellow": true,"green":
  false},7]}
  ```

- When you receive the event, event.data is an object with several attributes such as "myBoolKey" and "myIntKey". But the value of those attribute are all strings. For example, the value of "myBoolKey" is "false" rather than false; the value of "myIntKey" is "7" rather than 7. If the value of an attribute is an array or JSON string, they are all treated as regular strings. Besides, CEP does not support non-ASCII characters in CEP events, especially on Windows platform. So, the result is:

  ```
  { myBoolKey: "false", myIntKey: "7", myFloatKey: "5.400000",
  myStringKey: "testXXXXX", myArrayKey: "[5.4,true,false,{"yellow":
  true,"green": false},7]"}
  ```

Improvement in CEP 6.1

- CEP 6.1 keeps the type of all attributes in event.data as what they are. If the same string is passed through event.data by CEP 6.1, the result will be:

  ```
  {
      "myBoolKey": false,
      "myIntKey": 7,
      "myFloatKey": 5.4,
  ```

```
            "myStringKey": "testテスト测试",
            "myArrayKey": [
                5.4,
                true,
                false,
                {
                    "yellow": true,
                    "green": false
                },
                7
            ]
        }
```

- myArrayKey is an array object rather than a JSON string, and it has an anonymous object which includes "yellow" and "green" attribute.
- Therefore, Since CEP parses JSON and returns the parsed JavaScript object to you, you do not need to call JSON.Parse() to parse the result from event.data anymore.
- Besides, you can pass non-ASCII characters and binary data by CEP events.

Invoke point product's scripts from html extension

First, define a callback function in CEP extension:

```
function evalScirptCallback(result)
{
    // process the result string here.
}
```

Then call CSInterface.evalScript with the script you want to call and the callback function:

```
var script = "app.documents.add";  //Demo script
CSInterface.evalScript(script, evalScriptCallback);
```

Please be aware that the script in evalScript and the jsx file which is configured in <ScriptPath> in the extension's manifest are executed in host application's ExtendScript engine, which runs in host application's main thread. On the other hand, CEP event is also dispatched from host application's main thread. If the interaction between the script and CEP event is needed, please split the script into small parts and call them separately so that CEP event has a chance to be scheduled.

Vulcan messages

Starting with CEP 5.0, global CEP Events whose scope attribute is set to "GLOBAL" is no longer supported. Please use the APIs in Vulcan.js instead.

Vulcan message

The data structure of Vulcan message in JavaScript is as below.

```
/**
 * @class VulcanMessage
 * Message type for sending messages between host applications.
 * A message of this type can be sent to the designated destination
 * when appId and appVersion are provided and valid. Otherwise,
 * the message is broadcast to all running Vulcan-enabled applications.
 *
 * To send a message between extensions running within one
 * application, use the <code>CSEvent</code> type in CSInterface.js.
 *
 * @param type             The message type.
 * @param appId            The peer appId.
 * @param appVersion       The peer appVersion.
 *
 */
function VulcanMessage(type, appId, appVersion)
{
    this.type = type;
    this.scope = VulcanMessage.SCOPE_SUITE;
    this.appId = requiredParamsValid(appId) ? appId :
VulcanMessage.DEFAULT_APP_ID;
        this.appVersion = requiredParamsValid(appVersion) ? appVersion :
VulcanMessage.DEFAULT_APP_VERSION;
    this.data = VulcanMessage.DEFAULT_DATA;
}

VulcanMessage.TYPE_PREFIX    = "vulcan.SuiteMessage.";
VulcanMessage.SCOPE_SUITE    = "GLOBAL";
VulcanMessage.DEFAULT_APP_ID = "UNKNOWN";
VulcanMessage.DEFAULT_APP_VERSION = "UNKNOWN";
VulcanMessage.DEFAULT_DATA = "<data><payload></payload></data>";
VulcanMessage.dataTemplate = "<data>{0}</data>";
VulcanMessage.payloadTemplate = "<payload>{0}</payload>";
```

Listen for and Dispatch Vulcan message

addMessageListener, removeMessageListener, dispatchMessage and getPayload APIs are available to dispatch and listen for Vulcan messages. The API definitions are as below. Refer to Vulcan.js for more information.

```
/**
 * Registers a message listener callback function for a Vulcan message.
 *
 * @param type             The message type.
 * @param callback         The callback function that handles the message.
 *                         Takes one argument, the message object.
 * @param obj              Optional, the object containing the callback
method, if any.
 *                         Default is null.
 */
Vulcan.prototype.addMessageListener = function(type, callback, obj)

/**
```

```
 * Removes a registered message listener callback function for a Vulcan
message.
 *
 * @param type             The message type.
 * @param callback         The callback function that was registered.
 *                         Takes one argument, the message object.
 * @param obj              Optional, the object containing the callback
method, if any.
 *                         Default is null.
 */
Vulcan.prototype.removeMessageListener = function(type, callback, obj)

/**
 * Dispatches a Vulcan message.
 *
 * @param vulcanMessage    The message object.
 */
Vulcan.prototype.dispatchMessage = function(vulcanMessage)

/**
 * Retrieves the message payload of a Vulcan message for the registered
message listener callback function.
 *
 * @param vulcanMessage    The message object.
 * @return                 A string containing the message payload.
 */
Vulcan.prototype.getPayload = function(vulcanMessage)
```

Here is the example to demonstrate how to use the APIs in JavaScript.

```
var testVulcanMessage = new VulcanMessage(VulcanMessage.TYPE_PREFIX +
"test");
testVulcanMessage.setPayload("To be or not to be that is a question!");

var callback = function (message) {
        alert(VulcanInterface.getPayload(message));
};

VulcanInterface.addMessageListener(testVulcanMessage.type, callback);
VulcanInterface.dispatchMessage(testVulcanMessage);
...
VulcanInterface.removeMessageListener(testVulcanMessage.type, callback);
```

getEndPoints and getSelfEndPoint APIs are available to support point-to-point Vulcan message. The API definitions are as below. Refer to Vulcan.js for more information.

```
/**
 * Gets all available endpoints of the running Vulcan-enabled applications.
 *
 * Since 7.0.0
 *
 * @return                 The array of all available endpoints.
 * An example endpoint string:
 * <endPoint>
 *    <appId>PHXS</appId>
```

```
 *    <appVersion>16.1.0</appVersion>
 * </endPoint>
 */
Vulcan.prototype.getEndPoints = function()

/**
 * Gets the endpoint for itself.
 *
 * Since 7.0.0
 *
 * @return              The endpoint string for itself.
 */
Vulcan.prototype.getSelfEndPoint = function()
```

The steps to send point-to-point Vulcan message are as follows:

- Get all available endpoints.
- Select the destination endpoint and get appId and appVersion from it.
- Create Vulcan message with the destination appId and appVersion.
- Dispatch Vulcan message.

```
var endPointList = VulcanInterface.getEndPoints();
var destIndex = 0;
var appId = GetValueByKey(endPointList[destIndex], "appId");
var appVersion = GetValueByKey(endPointList[destIndex], "appVersion");
var message = new VulcanMessage(VulcanMessage.TYPE_PREFIX + "test", appId,
appVersion);
message.setPayload("blablabla...");
VulcanInterface.dispatchMessage(message);
```

Access Application DOM from Html Extension

There are two separate JavaScript engines here.

- JavaScript engine of host application - Application DOM/Extend script DOM
- JavaScript engine of CEP HTML runtime - HTML DOM

Application DOM is not available in CEP extension's engine and CEP DOM is not available in host application's engine.

To access Application DOM from CEP extensions, CEP JavaScript library provides an API, *CSInterface.evalScript,* to execute extend script to access the host application's DOM. Here is a brief diagram to indicate how to access Application DOM through this API.
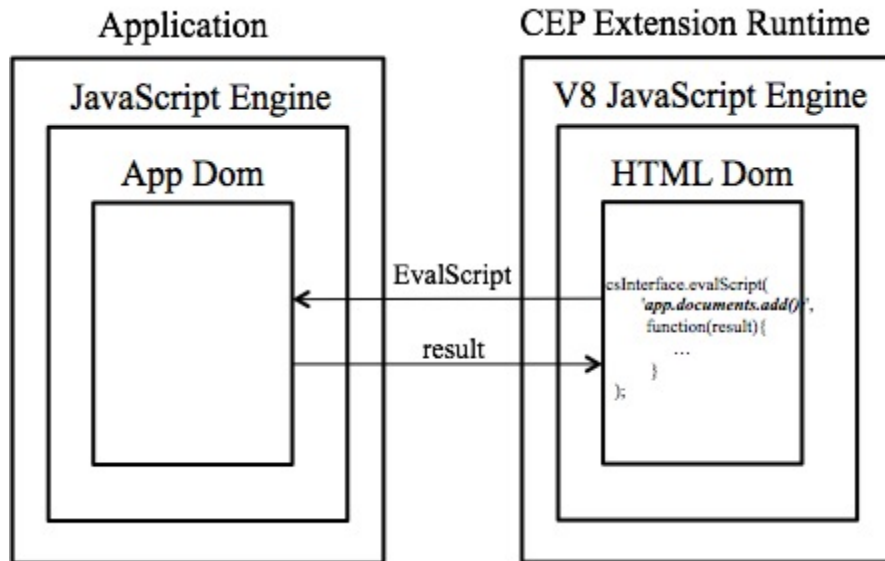
Here is the sample JavaScript code snippet in HTML extension.

```
var csInterface = new CSInterface();
csInterface.evalScript('app.documents.add();', function(result){
  alert(result);
});
```

Access HTML DOM from extend script

There is no way to access HTML extension's JavaScript DOM directly from Application's ExtendScript. If you need to access it, CEP event based communication can be used as a substitution.

CEP creates a library which uses External Object mechanism of ExtendScript to send CSXS events. The external object provides an ExtendScript class CSXSEvent for creating and dispatching application-level CSXS events. On HTML extension side, event listeners can be registered via the *addEventListener* API in *CSInterface.js* to listen to the events.

Some CC applications (Photoshop, Illustrator, Premiere Pro) integrate PlugPlugExternalObject library and start to support this functionality in CC 2014 release. Audition supports this functionality since CC 2015.1 release.

Sample Code

ExtendScript developers need to create external object instance first.

```
var externalObjectName = "PlugPlugExternalObject";
var mylib = new ExternalObject( "lib:" + externalObjectName );
```

And then create the CSXSEvent instance.

```
var eventObj = new CSXSEvent();
eventObj.type="documentCreated";
eventObj.data="blahblah";
```

At last use this instance to dispatch event:

```
eventObj.dispatch();
```

Below is the sample code of ExtendScript.

```
...
var cs = new CSInterface();

cs.addEventListener("documentCreated", function(event){
  alert('Cool!' + event.data);
});

var extendScript = 'var externalObjectName = "PlugPlugExternalObject"; var
mylib = new ExternalObject( "lib:" + externalObjectName );
app.document.add(); var eventObj = new CSXSEvent();
eventObj.type="documentCreated"; eventObj.data="blahblah";
eventObj.dispatch();'
cs.evalScript(extendScript);
```

Fly-out menu

For fly-out menu on the native panel of HTML extension, it has been supported.

Two new interfaces are added to CSInterface.

```
CSInterface.prototype.setPanelFlyoutMenu = function(menu){
  window.__adobe_cep__.invokeSync("setPanelFlyoutMenu", menu);
};

CSInterface.prototype.updatePanelMenuItem = function(menuItemLabel, enabled,
checked){
  var ret = false;
  if (this.getHostCapabilities().EXTENDED_PANEL_MENU){
       var itemStatus = new MenuItemStatus(menuItemLabel, enabled, checked);
       ret = window.__adobe_cep__.invokeSync("updatePanelMenuItem",
JSON.stringify(itemStatus));
  }

  return ret;
};
```

The "menu" parameter for "setPanelFlyoutMenu" is a XML string. Below is an example:

```
<Menu>
```

```
    <MenuItem Id="menuItemId1" Label="TestExample1" Enabled="true"
Checked="false"/>
    <MenuItem Label="TestExample2">
        <MenuItem Label="TestExample2-1" >
            <MenuItem Label="TestExample2-1-1" Enabled="false" Checked="true"/>
        </MenuItem>
        <MenuItem Label="TestExample2-2" Enabled="true" Checked="true"/>
    </MenuItem>
    <MenuItem Label="---" />
    <MenuItem Label="TestExample3" Enabled="false" Checked="false"/>
</Menu>
```

If user wants to be notified when clicking a menu item, user needs to register "com.adobe.csxs.events.flyoutMenuClicked" event by calling AddEventListener. When a menu item is clicked, the event callback function will be called. The "data" attribute of event is an object which contains "menuId" and "menuName" attributes.

To get notified when fly-out menu is opened and closed, register event listener for below event types respectively:

```
"com.adobe.csxs.events.flyoutMenuOpened"
"com.adobe.csxs.events.flyoutMenuClosed"
```

Customize Context Menu

Set and Update Context Menu

There are three APIs in CSInterface for developers to set and update the customized context menu.

```
CSInterface.prototype.setContextMenu = function(menu, callback){
  window.__adobe_cep__.invokeAsync("setContextMenu", menu, callback);
};

CSInterface.prototype.setContextMenuByJSON = function(menu, callback){
  window.__adobe_cep__.invokeAsync("setContextMenuByJSON", menu, callback);
};

CSInterface.prototype.updateContextMenuItem = function(menuItemID, enabled,
checked){
  var itemStatus = new ContextMenuItemStatus(menuItemID, enabled, checked);
  ret = window.__adobe_cep__.invokeSync("updateContextMenuItem",
JSON.stringify(itemStatus));
};
```

The "menu" parameter for "setContextMenu" is a XML string.

- Id - Menu item ID. It should be plain text.
- Icon - Menu item icon path. It is a path relative to the extension root path. For optimal display results please supply a 16 x 16px PNG icon as larger dimensions will increase the size of the menu item.

- Label - Menu item label. It supports localized languages.
- Enabled - Whether the item is enabled or disabled. Default value is true.
- Checkable - Whether the item can be checked/unchecked. Default value is false.
- Checked - Whether the item is checked or unchecked. Default value is false.
- The items with icons and checkable items cannot coexist on the same menu level. The former take precedences over the latter.

Here is an example.

```
<Menu>
    <MenuItem Id="menuItemId1" Label="TestExample1" Enabled="true"
Checked="false" Icon="./img/small_16X16.png"/>
    <MenuItem Id="menuItemId2" Label="TestExample2">
      <MenuItem Id="menuItemId2-1" Label="TestExample2-1" >
        <MenuItem Id="menuItemId2-1-1" Label="TestExample2-1-1"
Enabled="false" Checkable="true" Checked="true"/>
      </MenuItem>
      <MenuItem Id="menuItemId2-2" Label="TestExample2-2" Enabled="true"
Checkable="true" Checked="true"/>
    </MenuItem>
    <MenuItem Label="---" />
    <MenuItem Id="menuItemId3" Label="TestExample3" Enabled="false"
Checked="false"/>
</Menu>
```

The "callback" parameter is the callback function which is called when user clicks a menu item. The only parameter is the ID of clicked menu item.

If you prefer to using a JSON string to set context menu, you can achieve it by calling "setContextMenuByJSON".

The "menu" parameter for "setContextMenuByJSON" is a JSON string.

- id - Menu item ID. It should be plain text.
- icon - Menu item icon path. It is a path relative to the extension root path. For optimal display results please supply a 16 x 16px PNG icon as larger dimensions will increase the size of the menu item.
- label - Menu item label. It supports localized languages.
- enabled - Whether the item is enabled or disabled. Default value is true.
- checkable - Whether the item can be checked/unchecked. Default value is false.
- checked - Whether the item is checked or unchecked. Default value is false.
- The items with icons and checkable items cannot coexist on the same menu level. The former take precedences over the latter.

Here is an JSON example

{

```
"menu": [
    {
                    "id": "menuItemId1",
                    "label": "testExample1",
                    "enabled": true,
                    "checkable": true,
                    "checked": false,
        "icon": "./img/small_16X16.png"
        },
        {
                    "id": "menuItemId2",
        "label": "testExample2",
        "menu": [
            {
                "id": "menuItemId2-1",
                "label": "testExample2-1",
                "menu": [
                    {
                        "id": "menuItemId2-1-1",
                        "label": "testExample2-1-1",
                        "enabled": false,
                        "checkable": true,
                        "checked": true
                    }
                ]
            },
            {
                "id": "menuItemId2-2",
                "label": "testExample2-2",
                "enabled": true,
                "checkable": true,
                "checked": true
            }
        ]
    },
    {
        "label": "---"
    },
    {
        "id": "menuItemId3",
        "label": "testExample3",
        "enabled": false,
        "checkable": true,
        "checked": false
    }
  ]
}
```

If developers do not set context menu, CEP shows default items (Back, Forward, View Source, etc.). This is compatible with previous releases. If developers set context menu which has no any default id, CEP removes all default items and show customized items only. If one of the following default ids is set, the default menu item against that id will be shown.

id

"print"
"back"
"view source"
"forward"

Notes:

1. They are case-insensitive.
2. The quotation marks is not a part of an id.
3. The default callback associated to each id will be used and cannot be customised.

Disable Context Menu

To disable the context menu, you can call setContextMenu by null.

Another way is to add *oncontextmenu="return false;"* to the HTML tag. For example,

```
<body oncontextmenu="return false;">
```

Other Implementation of Context Menu

See [examples](#).

Get Display Status of HTML Extension Window

Two ways to get HTML extension window

- Register "com.adobe.csxs.events.panelWindowStatusChanged" CSXS event
- Call isWindowVisible JavaScript API.

Resister "com.adobe.csxs.events.panelWindowStatusChanged" CSXS event

- Observe "com.adobe.csxs.events.panelWindowStatusChanged" CSXS event, this is for PANEL extensions only. If user hides the panel window by clicking "X" or collapsing window, this event is going to be sent to observer with the "true" or "false" string in data attribute, while the event is not going to be sent if the extension is closed.
  That is to say, currently, only panel extensions which are running on Ai and persistent can receive this event, when the extension is hiding, event with "false" data is sent while the extension is shown, event with "true" is sent.

Call isWindowVisible API

- Call "isWindowVisible" JS interface. Both dialog and panel extension enable to access this API, but it always returns true for modal and modeless dialog extensions while it always returns false for invisible extensions.

Getting and Changing Extension Content Size

Getting Extension Content Size

Getting extension content size can be done using *window.innerWidth* and *window.innerHeight*. However, if you are accessing these properties from inside an IFrame, you are actually accessing the properties of the IFrame's window object, not the ones for the HTML document. To access the top-most one, you will need to do "parent.window.innerWidth" and "parent.window.innerWidth".

Changing Extension Content Size

Changing modal and modeless extension content size is supported in all Adobe applications that supports CEP. However, changing panel HTML extension size is not supported in Premiere Pro, Prelude, After Effects and Audition.

```
CSInterface.prototype.resizeContent = function(width, height)
```

The width and height parameters are expected to be unsigned integers. The function does nothing when parameters of other types are passed.

Please note that extension min/max size constraints as specified in the manifest file apply and take precedence. If the specified size is out of the min/max size range, the min or max bounds will be used. When a panel is docked with other panels, there are chances that it won't resize as expected even when the specified size satisfies the min and max constraints. The restriction is imposed by host applications, not by CEP.

Register invalid certificate error callback

(Since 6.1)

Register the invalid certificate error callback for an extension. This callback will be triggered when the extension try to access the web site that contains the invalid certificate on main frame. But if the extension does not call this function and try to access the web site containing the invalid certificate, a default error page will be shown:

```
CSInterface.prototype.registerInvalidCertificateCallback = function(callback)
```

Register an interest in specific key events

(Since 6.1)

Register an interest in some key events to prevent them from being sent to the host application:

```
CSInterface.prototype.registerKeyEventsInterest = function(keyEventsInterest)
```

This function works with modeless extensions and panel extensions. Generally all the key events will be sent to the host application for these two extensions if the current focused element is not text input or dropdown.

If you want to intercept some key events and you want them to be handled in the extension, please call this function in advance to prevent them being sent to the host application.

- keyEventsInterest: A JSON string describing those key events you are interested in. A null object or an empty string will lead to removing the interest

This JSON string should be an array, each object has following keys:

- keyCode: [Required] represents an OS system dependent virtual key code identifying the unmodified value of the pressed key.
- ctrlKey: [optional] a Boolean that indicates if the control key was pressed (true) or not (false) when the event occurred.
- altKey: [optional] a Boolean that indicates if the alt key was pressed (true) or not (false) when the event occurred.
- shiftKey: [optional] a Boolean that indicates if the shift key was pressed (true) or not (false) when the event occurred.
- metaKey: [optional] (Mac Only) a Boolean that indicates if the Meta key was pressed (true) or not (false) when the event occurred. On Macintosh keyboards, this is the command key. To detect Windows key on Windows, please use keyCode instead.

To learn all key codes:

- Windows
- Mac
    - /System/Library/Frameworks/Carbon.framework/Versions/A/Frameworks/HIToolbox.framework/Versions/A/Headers/Events.h
    - Install Key Codes from the Mac App Store.

An example JSON string:

```
[{
    "keyCode": 48
},
{
    "keyCode": 123,
```

```
    "ctrlKey": true
}]
```

Set and Get the title of the extension windows

(Since 6.1)

CEP 6.1 introduces two APIs to set and get the title of extension windows. Those functions work with modal and modeless extensions in all Adobe products, and panel extensions in Photoshop, InDesign, InCopy, Illustrator, Flash Pro and Audition:

```
CSInterface.prototype.setWindowTitle = function(title){
  window.__adobe_cep__.invokeSync("setWindowTitle", title);
};

CSInterface.prototype.getWindowTitle = function(){
  return window.__adobe_cep__.invokeSync("getWindowTitle", "");
};
```

HI-DPI display

CEP JavaScript library provides APIs for detecting the availability of HI-DPI display on the Mac platform.

- CSInterface.getScaleFactor()
  Use this function to retrieve the scale factor of the display on which the calling extension window is located.

  ```
  var scaleFactor = CSLibrary.getScaleFactor();
  ```

- CSInterface.setScaleFactorChangedHandler()
  Use this function to add a event handler that will be called when calling extension window is moved between HI-DPI and non-HI-DPI displays.

  ```
  window.scaleFactorHandler = function(){
    var scaleFactor = CSLibrary.getScaleFactor();
    if (scaleFactor === 2){
      imgSrc = "../img/PS_AppIcon_r.png"
    } else {
      imgSrc = "../img/PS_AppIcon.png"
    }
    document.getElementById("image").src = imgSrc;
  }

  CSLibrary.setScaleFactorChangedHandler(window.scaleFactorHandler);
  ```

CEP 5.2 has already supported HiDPI on Windows.

Other JavaScript APIs

The JavaScript engine in CEP HTML engine had been extended to provide some APIs, including:

- local file access
- native process
- others

These APIs are in JavaScript DOM and can be used as other built-in JavaScript APIs. You do **NOT** need to include any JavaScript files.

API reference is as below.

CEPEngine_extensions.js is actually a CEF extension that is built in CEPHtmlEngine to expand the DOM of CEPHtmlEngine, like create/delete folder, read/write file, create/quit process, and so on. You can invoke these built-in APIs directly in your HTML extension without any JavaScript file reference.

For example, you want to

1) create a folder.

```
var path = "/tmp/test";
var result = window.cep.fs.makedir(path);
if (0 == result.err){
  ...// success
} else {
  ...// fail
}
```

2) write a file.

```
var data = "This is a test.";
var path = "/tmp/test";
var result = window.cep.fs.writeFile(path, data);
if (0 == result.err){
  ...// success
} else {
  ...// fail
}
```

3) Write file with base64 encoding mode. To use this mode, you need to convert the input string to a base64-encoded string before calling writeFile(). The following is an example.

```
var data = "This is a test.";
var path = "/tmp/test";
```

```
data = cep.encoding.convertion.utf8_to_b64(data);

var result = window.cep.fs.writeFile(fileName, data, cep.encoding.Base64);
if (0 == result.err) {
  ...// success
} else {
  ...// fail
}
```

4) read a file.

```
var path = "/tmp/test";
var result = window.cep.fs.readFile(path);
if(result.err === 0){
  //success
  alert(result.data); //result.data is file content
} else {
  ...// fail
}
```

5) Read file with base64 encoding mode in which the read data after readFile called is converted to a base-encoded string. You need to decode this string to any format you want. The following is an example

```
var path = "/tmp/test";
result = window.cep.fs.readFile(path, cep.encoding.Base64);
if(result.err === 0){
  //success
  var base64Data = result.data;
  var data = cep.encoding.convertion.b64_to_utf8(base64Data);
} else {
  ...// fail
}
```

6) Create a process and check if it's running.

```
var result = window.cep.process.createProcess("usr/X11/bin/xterm");

if(result.err === 0){
  var pid = result.data;
  result = window.cep.process.isRunning(pid);
  if(result.data === true){
    // running
  }
}
```

You could use other APIs like delete folder, rename folder, set file permission, delete file, show file open dialog, quit process, etc.

To know them now, please look at
//csxs/main/projects/native/CEPHtmlEngine/common/CEPEngine_extensions.js

We have the following samples that demonstrate use of some of these APIs:

- https://github.com/Adobe-CEP/Samples/tree/master/Flickr
- https://github.com/Adobe-CEP/Samples/tree/master/Collage

Localization

In order to support localization, both the extension and the host application must provide locale information. There are two distinct types of locale information.

- The **License Locale** (returned as the applicationLocale by the AMT library)
- The **Effective/Language/UI** Locale (which is controlled by the user in the OS settings).

The extension must provide the list of supported locales for both the License Locale and the Language Locale via the HostEnvironment. This is particularly important in cases where the extension has features for a specific locale. PlugPlug library expects the host application to provide the locale information as part of the environmental data.

JavaScript API HostEnvironment has the appLocale property in place.

In the CSXS Reference Application, the applicationLocale can be changed in <userHome>/settings.txt. The default value is en_US for the Reference Application.

License Locale and locales supported by extension

CEP checks the **License Locale** of host application against supported locales declared in extensions locale list to determine if the extension is loadable for the host application.

Locale folder structure

The locale folder structure in HTML extensions is similar as Flash extensions. Each property file should be placed in its corresponding locale folder. For example, the en_US property file should be <YourExtension>/locale/en_US/messages.properties. Users can define a default property file (<YourExtension>/locale/messages.properties), which will be used when the corresponding locale file is not defined.

```
YourExtension/
      |-csxs/
      |-locale/                       <-- Directory for localized resources
         |-- messages.properties    <-- The one to fallback to if no
localized resources is provided for a locale
         |-- en_US/
         |         |- messages.properties
         |-- zh_CN/
                   |- messages.properties
```

Locale file format is similar to Flash extension. It contains multiple lines of <key>=<value>. There should be a new line below the last property key/value.

```
key1=value1
key2=value2
key3.value=value3
key4.innerHTML=value4
```

CEP provides a JS interface named **initResourceBundle** to initialize the locale resources. This should be called during the loading of the extension. CEP initializes the resource bundle for the extension with property values for the current application and UI locale. Then users can access the resource bundle (object) to get the localized strings.

```
var csInterface = new CSInterface();
csInterface.initResourceBundle();
```

Sharing localization resources across multiple locales

CEP 6.0 provided a mechanism to allow multiple locales to share the same localization resources (messages.properties).

For example, you want es_MX to use the messages.properties for es_ES. To do so, supply a file named fallback.properties in the es_MX folder as illustrated below.

```
YourExtension/
     |-csxs/
     |-locale/
          |-- messages.properties
          |-- es_ES/
          |          |- messages.properties
          |-- es_MX/
                     |- fallback.properties
```

In the fallback.properties file, you specify which locale's localized resources you want es_MX to use, in below format

```
fallback=es_ES
```

Side Notes:

- The fallback.properties file takes precedence when both messages.properties and fallback.properties exist at the same time.
- If fallback.properties is malformed, or it specifies a non-existent fallback locale, the messages.properties file in the same directory will be used.

Localized menu

In manifest, it supports to use locale string as menu. For example, in ShareOnBehance's manifest, it is using %UI_share_on_Behance. UI_share_on_Behance is defined as "UI_share_on_Behance=xxx" in messages.properties.

```
<UI>
    <Type>ModalDialog</Type>
    <Menu>%UI_share_on_Behance</Menu>
    ...
</UI>
```

Examples

Example 1

```
var cs = new CSInterface();
// Get properties according to current locale of host application.
var resourceBundle = cs.initResourceBundle();
// Use the localized strings.
<script type="text/javascript">document.write(resourceBundle.key1);</script>
```

Example 2

data-locale is the custom HTML element attribute and you can add to each HTML element that you want to localize.

In this example, there is "key3.value=value3" in the property file. In the HTML file, the input widget has attribute "data-locale" with "key3", then its value is set to "value3".

In this example, there is "key4.innerHTML=value4" in the property file. In the HTML file, the text area widget has attribute "data-locale" with "key4", then its innerHTML is set to "value4".

```
<script type="text/javascript">
  var cs = new CSInterface();

  // Get properties according to current locale of host application.
  var resourceBundle = cs.initResourceBundle();

  // Use the localized strings.
  document.write(resourceBundle.key1);
  document.write(resourceBundle.key2);
</script>

<input type="submit" value="" data-locale="key3"/>
<textarea rows="10" cols="80" data-locale="key4"></textarea>
```

Example 3

Use parameters ($1, $2, ...) in localized strings.

```
var localize = function(key){
  var cs = new CSInterface();
  var resourceBundle = cs.initResourceBundle();
  var localizedStr = resourceBundle[key];
  if (localizedStr){
    var index = 1;
    while (localizedStr.indexOf("$" + index) !== -1){
      localizedStr = localizedStr.replace("$" + index, arguments[index]);
      index++;
    }
    return localizedStr;
  } else {
    return '';
  }
};
```

Supporting MENA locales

MENA stands for "Middle East and North Africa". Support needs to be provided for Arabic, Hebrew, and NA French languages. Products supporting these languages are: ID, PS, AI, DW and Acrobat.

With MENA, new AMT locales have been added to AMT in CS6:

Language	ISO Code
Arabic (Middle East Enabled English Arabic)	en_AE
Hebrew (Middle East Enabled English Hebrew)	en_IL
NA French	fr_MA

If an extension needs to be loaded in host applications in MENA locales, MENA locales must be added to the supported locale list of the extension manifest file. For example:

```
<LocaleList>
    ...
    <Locale Code="en_AE"/>
    <Locale Code="en_IL"/>
    <Locale Code="fr_MA"/>
    ...
</LocaleList>
```

Extension localization for MENA locales

Suppose your extension has this directory layout

```
Extension/
        |-xxx.swf
        |-csxs/
```

```
            |-locale/                                   <-- Directory for localized
resources
                    |-- messages.properties        <-- The one to fallback to
if no localized resources is provided for a locale
                    |-- fr_FR/
                    |       |- messages.properties
                    |-- en_GB/
                            |- messages.properties
```

When CSInterface.initResourceBundle() is called, CEP uses the app UI locale (not app locale) reported by PP to load localized resources. If there is no localized resources for an app UI locale, for example fr_MA, then CEP will fall back to use messages.properties located under the "locale" folder.

With MENA feature, PPs map en_AE/en_IL to en_US and fr_MA to fr_FR for app UI locale. In this case, for en_AE and en_IL build of PP, en_US resources will be used if provided and for fr_MA build of PP, fr_FR resources will be used if provided. What extension team needs to do in this case is to provide en_US version of resources for en_AE/en_IL and fr_FR version of resources for fr_MA.

Video/Audio Playback

CEP 5.0 supports playing video and audio encoded in below formats

Format	MIME-Type	Misc.
MP4	video/mp4	MPEG 4 files with H.264 video codec and AAC audio codec
Ogg	video/ogg	Ogg files with Theora video codec and Vorbis audio codec
mp3	audio/mpeg	

Here is an example of playing video in your extension:

```
<video poster="http://www.html5rocks.com/en/tutorials/video/basics/star.png"
controls>
    <source
src="http://www.html5rocks.com/en/tutorials/video/basics/Chrome_ImF.mp4"
type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'/>
</video>
```

One thing to note is that because HTML extensions are hosted in integrating application's windows, video cannot be played in full-screen mode.

WebRTC

WebRTC is targeting to serve stream audio, video capture, like online video conference. WebRTC is not enabled by default. To enable it, the schema below need to be added in manifest file. For details, refer to Customize CEF command parameters.

```
<CEFCommandLine> <Parameter>--enable-media-stream</Parameter>
</CEFCommandLine>
```

For WebRTC related development, CEP runtime just keeps the same experiences as the usage in Chrome. Below is a sample script to demonstrate how to use it in HTML pages of extension:

```
<video id="basic-stream" autoplay></video>

<script>
var errorCallback = function(e){
  if (e.code === 1) {
    alert('User denied access to their camera');
  } else {
    alert('getUserMedia() not supported in your browser.');
  }
}

var video = document.querySelector('#basic-stream');
var localMediaStream = null;

if(navigator.getUserMedia){
  navigator.getUserMedia('video', function(stream){
  video.src = stream;
  video.controls = true;
  localMediaStream = stream;
  }, errorCallback);
} else if(navigator.webkitGetUserMedia){
  navigator.webkitGetUserMedia({ video: true }, function(stream){
    video.src = window.URL.createObjectURL(stream);
    video.controls = true;
    localMediaStream = stream;
  }, errorCallback);
}
</script>
```

More samples.

In addition, there are some limitations of WebRTC support in CEP 5.0 runtime due to the known issue in CEF3:

Issue. No	Description	Link
1065	Add support for webrtc based screen sharing/capturing	https://code.google.com/p/chromiumembedded/issues/detail?id=1065 http://www.magpcss.org/ceforum/viewtopic.php?f=6&t=10982
1139	cefclient w/OSR hangs on exit with the WebRTC Reference App	https://code.google.com/p/chromiumembedded/issues/detail?id=1139&q=Web

1144	CEF3 does not support RTCPeerConnection	https://code.google.com/p/chromiumembedded/issues/detail?id=1144
1151	CEF3: Mac: Process is blocked to quit when iframe is using WebRTC	https://code.google.com/p/chromiumembedded/issues/detail?id=1151&start=1
1153	CEF3: Win: Can not show https://apprtc.appspot.com	https://code.google.com/p/chromiumembedded/issues/detail?id=1153&start=1

Scroll bar tips

On Mac, scroll bars of panel are hidden by OS (since Lion by design). It can be always shown by settings as below.
1. Click the Apple menu at the top-left of the screen, then select System Preferences.
2. Next, select the General preferences pane; it's the very first one, up at the top.
3. Under the "Show scroll bars" heading, you'll find three options: "Automatically based on input device," "When scrolling," and "Always." Chose "Always."

Invisible HTML Extensions

An HTML extension can be invisible during its whole life cycle. This means

- It always runs in the background
- It is never visible

To make an HTML extension invisible

- Set extension manifest version to "5.0" or higher.
- Specify its window type as 'Custom' in the manifest file.
- Set <AutoVisible> to false in the manifest file.
- If you do not want the extension to appear in the Window->Extensions menu, do not add the <Menu> tag.
- If you want the extension to start on specific types of events, specify those events using <StartOn> tag.

Here is an example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ExtensionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ExtensionBundleId="IamInvisible" ExtensionBundleVersion="1.0" Version="5.0">
```

```xml
    <ExtensionList>
        <Extension Id="IamInvisible" Version="1.0"/>
    </ExtensionList>
    <ExecutionEnvironment>
        <HostList>
            <Host Name="PHXS" Version="13.0"/>
        </HostList>
        <LocaleList>
            <Locale Code="All"/>
        </LocaleList>
        <RequiredRuntimeList>
            <RequiredRuntime Name="CSXS" Version="5.0"/>
        </RequiredRuntimeList>
    </ExecutionEnvironment>
    <DispatchInfoList>
        <Extension Id="IamInvisible">
            <DispatchInfo>
                <Resources>
                    <MainPath>./html/index.html</MainPath>
                </Resources>
                <Lifecycle>
                    <AutoVisible>false</AutoVisible>
                    <StartOn>
                        <!-- Photoshop dispatches this event on startup -->
                        <Event>applicationActivate</Event>
                        <!-- Premiere Pro dispatches this event on startup -
->

<Event>com.adobe.csxs.events.ApplicationActivate</Event>
                        <!-- You can add more events -->
                        <Event>another_event</Event>
                    </StartOn>
                </Lifecycle>
                <UI>
                    <Type>Custom</Type>
                    <Geometry>
                        <Size>
                            <Height>1</Height>
                            <Width>1</Width>
                        </Size>
                    </Geometry>
                </UI>
            </DispatchInfo>
        </Extension>
    </DispatchInfoList>
</ExtensionManifest>
```

One important thing to note is that not all host applications support Invisible HTML Extension. See table below for more information:

PP	Supports Invisible Extension	Misc.

Photoshop	Yes	
Premiere Pro	Yes	
Prelude	Yes	
Flash Pro	Yes	
Audition	Yes	
InDesign	No	Doesn't work at all because InDesign doesn't support 'Custom' window type.
InCopy	No	Doesn't work at all because InCopy doesn't support 'Custom' window type.
Illustrator	No	The invisible extension is shown as a visible panel. Possibly that Illustrator treated the 'Custom' window type as 'Panel'. One possible workaround is to use the 'Modeless' window type instead of 'Custom'. However, under certain situations (someone calls PlugPlugLoadExtension for example), the extension will become visible.

Customize CEF Command Line Parameters

Chromium/CEF command line parameters can be passed to CEPHtmlEngine, like --enable-media-stream. Available Chromium command line [parameters](#).

CEP filters out some parameters due to various reasons:

Parameters	What is it filtered out?
--remote-debugging-port	This could overwrite the one in .debug file. Filter out to avoid conflict.
--ignore-certificate-errors	This ignores SSL certificate errors. It is a security concern to ignore invalid server certificate, which allows extensions to load files from malicious sites.

All other parameters are passed to underlying CEF. It is up to CEF to decide whether a parameter is supported and what is the behavior.

How to use CEF command line parameters

* Add <CEFCommandLine><Parameter>--param1<Parameter/> ... </CEFCommandLine> in manifest.

- For *key=value* parameter, add <CEFCommandLine><Parameter>--param1=value1<Parameter/> ... </CEFCommandLine> in manifest.

Here is an example:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ExtensionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ExtensionBundleId="xx.yy.zz" ExtensionBundleVersion="1.0" Version="5.0">
    <ExtensionList>
        <Extension Id="xx.yy.zz" Version="1.0"/>
    </ExtensionList>
    <ExecutionEnvironment>
        <HostList>
            <Host Name="PHXS" Version="13.0"/>
            <Host Name="PPRO" Version="6.0"/>
        </HostList>
        <LocaleList>
            <Locale Code="All"/>
        </LocaleList>
        <RequiredRuntimeList>
            <RequiredRuntime Name="CSXS" Version="5.0"/>
        </RequiredRuntimeList>
    </ExecutionEnvironment>
    <DispatchInfoList>
        <Extension Id="xx.yy.zz">
            <DispatchInfo>
                <Resources>
                    <MainPath>./html/index.html</MainPath>
                    <CEFCommandLine>
                        <Parameter>--enable-media-stream</Parameter>
                    </CEFCommandLine>
                </Resources>
                ...
            </DispatchInfo>
        </Extension>
    </DispatchInfoList>
</ExtensionManifest>
```

Commonly used CEF command parameters:

Parameters	Notes
--enable-media-stream	Enable media (WebRTC audio/video) streaming.
--enable-speech-input	Enable speech input (x-webkit-speech).
--persist-session-cookies	Persist session cookies.
--disable-image-loading	Disable loading of images from the network. A cached image will still be rendered if requested.
--disable-javascript-open-windows	Disable opening of windows via JavaScript.
--disable-javascript-close-windows	Disable closing of windows via JavaScript.

--disable-javascript-access-clipboard	Disable clipboard access via JavaScript.
--enable-caret-browsing	Enable caret browsing.
--proxy-auto-detect	This tells Chrome to try and automatically detect your proxy configuration. See more info on http://www.chromium.org/developers/design-documents/network-settings.
--user-agent	A string used to override the default user agent with a custom one.
--disable-application-cache	Disable the ApplicationCache.
--enable-nodejs	Enable Node.js APIs in extensions. Supported since CEP 6.1.
--disable-pinch	Disable compositor-accelerated touch-screen pinch gestures.
--mixed-conext	Enable the "mixed context" mode. Supported since CEP 7.0.

HTML Extension Persistent

The purpose of persistent is to force not reload HTML extension when it is closed or hidden. Photoshop has provided persistent since the version of 14.2.

To make an HTML extension persistent in Photoshop

- Upgrade Photoshop version to 14.2 or later
- Dispatch the event *com.adobe.PhotoshopPersistent* from HTML extension to Photoshop to request persistent

Sample code:

```
var Persistent = function(inOn){
  if(inOn){
    var event = new CSEvent("com.adobe.PhotoshopPersistent", "APPLICATION");
  } else {
    var event = new CSEvent("com.adobe.PhotoshopUnPersistent",
"APPLICATION");
  }
  event.extensionId = gExtensionId;
  csInterface.dispatchEvent(event);
}

Persistent(true); //persistent to prevent extension from unloading
…
Persistent(false); //unpersistent
```

FullScreen API in HTML Extension

CEP HTML extension runtime has not supported fullscreen API yet. The reason is its based library CEF3 does not support [fullscreen API](#).

Open URL link in default browser

In HTML extension, URL link could be opened in the default browser by calling *window.cep.util.openURLInDefaultBrowser('http://example.com')*:

```
<li><button
onclick="window.cep.util.openURLInDefaultBrowser('http://www.adobe.com')">Ope
n browser</button></li>
```

Using Node.js APIs (CEP 6.0 and prior releases)

Node.js Support

One of the most prominent feature in CEP 5.0 is allowing Node.js APIs to be used in HTML extensions. Most of the built-in APIs in Node.js version 0.8.22 are available to HTML extensions, with the below exceptions:

- **Cluster** APIs are not supported.
- **Console** APIs are not supported on Windows.

Other things to note:

- CEP injects following symbols into the root HTML DOM:
  - **global, GLOBAL, root** - same with the window object
  - **Buffer** - node's Buffer class
  - **process** - node's process class
  - **require** - the magic function that bring you node API
  - **module** - in node the main script is a special module, this is for the compatibility

- Conflicts with Web-Based Require Function

  - If your app uses libraries like RequireJS that inserts a require function into DOM, you should consider renaming CEP's require function to something else before migrating.
  - ```
    <script type="text/javascript">windows.nodeRequire=window.require &&
    window.require=undefined</script>
    <script type="text/javascript"
    src="your/require/js/file.js"></script>
    ```

- Conflicts with Web-Based module Function
  - If your app uses JQuery that is trying to register itself as nodejs plugin, then you will have to add the script below inside script tag to define window.module as undefined.

```
<script type="text/javascript">window.module =
undefined</script>
```

- Disable Node.js APIs in iframe
  Because of security consideration, CEP provides an option to disable Node.js APIs in
  iframe. To do so, add a **nodejs-disabled="true"** attribute to iframe tag. For example:

  ```
  <iframe id="xxx" class="xxxxx" nodejs-disabled="true">
  ```

- Forcing the environment implementation. If you are using RequireJS, and the text plugin
  tries to detect what environment is available for loading text resources, Node,
  XMLHttpRequest (XHR) or Rhino, but sometimes the Node or Rhino environment may
  have loaded a library that introduces an XHR implementation. You can force the
  environment implementation to use by passing an "env" module config to the plugin:

  ```
  requirejs.config({
      config: {
          text: {
              //Valid values are 'node', 'xhr', or 'rhino'
              env: 'rhino'
          }
      }
  });
  ```

Node.js Modules

JavaScript Modules

All third-party node JavaScript modules are supported. The root search path of third-party
modules is the directory which contains your HTML file. For example, when you do require
in file:///your_extension/index.html, CEP will lookup modules
under file:///your_extension/node_modules, this rule is exactly the same with upstream node.

Samples

Use Environment Variables

```
process.env.ENV_VARIABLE // ENV_VARIABLE is the name of the variable you want
to access.
```

Use Node.js to download files

- http://www.hacksparrow.com/using-node-js-to-download-files.html

Using Node.js APIs (CEP 6.1)

Node.js Support

CEP 6.1 upgraded its HTML engine to CEF 2272 (based on Chromium 41.0.2272.104) with IO.js version 1.2.0 integrated.

Known limitations:

- **Cluster** APIs are not supported.
- **Console** APIs are not supported on Windows.

Other things to note:

- Node.js APIs are disabled by default
  Due to security consideration, node.js APIs are disabled by default (prior to CEP 6.1, they were enabled by default) both on the extension level and IFrame level.
  To enable Node.js APIs:
  - Set ExtensionManifest version and RequiredRuntime version 5.0 or higher.
  - Specify '--enable-nodejs' in extension manifest. See section Customize CEF Command Line Parameters for details
  - To use Node.js APIs in IFrames, add property 'enable-nodejs' to it and to all its ancestor IFrames. If any of its ancestors don't have this property specified, Node.js APIs won't work

    ```
    <iframe id="xxx" class="xxxxx" enable-nodejs>
    ```

- The old 'nodejs-disabled' CEF command line parameter and IFrame property are no longer supported and ignored by the new HTML engine
- Node context and Browser context
  The way io.js was integrated into CEF introduced two types of JavaScript contexts, one for browser, the other for io.js. Global objects created in HTML pages are in browser context, while "required" js files run in io.js context. These two contexts don't have direct access to each other's data. To share data, pass reference to objects between the two contexts:
  - Accessing objects in io.js context from browser context
    For example, in browser context, "var backbone = require('backbone');" executes the backbone module's code and then pass the result object to browser context.
  - Accessing objects in browser context from io.js context
    Browser context's 'window' global object is injected to io.js context, providing a way to access objects in browser context from io.js context.
    For example, if you want to access a global object named 'localeStrings' defined in browser context from your io.js module, use 'window.localStrings' in your io.js module.

- CEP injects following symbols into the root HTML DOM:
  - **global, GLOBAL, root** - same with the window object
  - **Buffer** - node's Buffer class
  - **process** - node's process class
  - **require** - the magic function that bring you node API
  - **module** - in node the main script is a special module, this is for the compatibility

- Conflicts with Web-Based Require Function

  - If your app uses libraries like RequireJS that inserts a require function into DOM, you should consider renaming CEP's require function to something else before migrating.

    ```
    <script type="text/javascript">windows.nodeRequire=window.require &&
    window.require=undefined</script>
    <script type="text/javascript"
    src="your/require/js/file.js"></script>
    ```

- Conflicts with Web-Based module Function
  - If your app uses JQuery that is trying to register itself as nodejs plugin, then you will have to add the script below inside script tag to define window.module as undefined.

    ```
    <script type="text/javascript">window.module =
    undefined</script>
    ```

- Forcing the environment implementation. If you are using RequireJS, and the text plugin tries to detect what environment it is available for loading text resources, Node, XMLHttpRequest (XHR) or Rhino, but sometimes the Node or Rhino environment may have loaded a library that introduces an XHR implementation. You can force the environment implementation to use by passing an "env" module config to the plugin:

```
requirejs.config({
    config: {
        text: {
            //Valid values are 'node', 'xhr', or 'rhino'
            env: 'rhino'
        }
    }
});
```

Using Node.js APIs (CEP 7.0)

In addition to the Node.js support in CEP 6.1, CEP 7.0 provided a new "mixed context" Node.js mode. Unlike the "separate context" mode in CEP 6.1 where a "required" node module is in a separate JavaScript context, a "required" node module and the JavaScript code that "requires" it are in the same context in the new "mixed context" mode, eliminating all the inconveniences in the old "separate context" mode.

This mode is disabled by default. To enabled it, add command line parameter "--mixed-context" to your extension manifest.

```
<Parameter>--mixed-context</Parameter>
```

Node.js Modules

JavaScript Modules

All third-party node JavaScript modules are supported. The root search path of third-party modules is the directory which contains your HTML file. For example, when you do require in file:///your_extension/index.html, CEP will lookup modules under file:///your_extension/node_modules, this rule is exactly the same with upstream node.

Samples

Use Environment Variables

```
process.env.ENV_VARIABLE // ENV_VARIABLE is the name of the variable you want
to access.
```

Use Node.js to download files

- http://www.hacksparrow.com/using-node-js-to-download-files.html

Limitation of cep.process.stdout/stderr

There is a known limitation of cep.process.stdout/stderr which is targeting to capture one time of stdout/stderr output.

For applications that has not integrated CEP 5, there are two workarounds suggested as the following.

1. Embed cep.process.stdout/stderr:

```
var getSTDOutput = function(){
  console.log("getSTDOutput");
  window.cep.process.stdout(pid, function(output){
  console.log(output);
  // your code is here
  });

  var result = window.cep.process.isRunning(pid);
  if (result.data === true){
    setTimeout(getSTDOutput, 1000);
  }
}
```

2. Join all stdout output as one, like below

```
var str1 = 'abcdef';
var str2 = '12345';
var str3 = 'gghhtt';
console.log(str1 + str2 + str3);
```

An example on how to get curl downloading progress through stderr:

```
<script>
        var downloadPid = -1;
        function getStdErrOutput()
        {
            window.cep.process.stderr(downloadPid, function(progress) {
                                        var keys = progress.split(new
RegExp('[# ]', 'g'));
                                        for(i=0; i<keys.length; i++){
                                          if (keys[i] != '') {
                                           console.log(keys[i]);
                                          }
                                        }
                                    });
            var result = window.cep.process.isRunning(downloadPid);
            if (result.data == true)
            {
                setTimeout(getStdErrOutput, 100);
            }
        }
        var doDownload = function() {
            var qURL = 'http://code.jquery.com/jquery-1.11.0.min.js';
            var dest = '/tmp/test.js';
            console.log("ext download (curl) " + qURL + " " + dest);
            var result = window.cep.process.createProcess('/usr/bin/curl',
qURL, '-#', '-o', dest);
            downloadPid = result.data;
            console.log("download pid is " + downloadPid);
            getStdErrOutput();
        };

        doDownload();
</script>
```

Since CEP 5, Node.js is integrated into CEP runtime and users could invoke the standard APIs of Node.js in extension directly. For applications that has integrated CEP 5, refer to http://nodejs.org/api/process.html for how to use the global process object in Node.js.

Other JavaScript Information

Load Multiple JSX files

HTML extension can load jsx files which define functions and objects into ExtendScript environment, thus, the extension can refer them by evalScript.

There are two approaches to load jsx files:

1. Define <ScriptPath> node in manifest.xml, and the value of the node is the relative path for the jsx file. For example:

```
<Extension Id="com.adobe.CEPHTMLTEST.Panel1">
    <DispatchInfo>
        <Resources>
            <MainPath>./html/index.html</MainPath>
            <CEFCommandLine>
                <Parameter>--enable-speech-input</Parameter>
                <Parameter>--enable-media-stream</Parameter>
            </CEFCommandLine>
            <ScriptPath>./jsx/example.jsx</ScriptPath> <!--
ExtensionRootPath/jsx/example.jsx -->
        </Resources>
    ........................
    </DispatchInfo>
</Extension>
```

2. Call "$.evalFile(jsxFile)" to load other jsx files. Probably developers will refer to "$.fileName" to find out the jsx files path they expect. The value of "$.fileName" should be the current executed jsx file path. For example:

```
var extensionPath = $.fileName.split('/').slice(0, -1).join('/') + '/';
// The value of $.fileName should be ExtensionRootPath/jsx/example.jsx,
while the value of extensionPath should be "ExtensionRootPath/jsx/"
$.evalFile(extensionPath + 'example1.jsx');
$.evalFile(extensionPath + 'example2.jsx');
$.evalFile(extensionPath + 'example3.jsx');
```

But if the "$.fileName" is referred in the FIRST LOADED jsx file, the value is not correct. That is to say, if the snippet above runs in example.jsx which is referred in the manifest.xml, the error will arise. So, PLEASE AVOID using "$.fileName" in the FIRST LOADED jsx file, maybe this is a limitation in ExtendScript. The workaround is refer it in the second loaded and afterward jsx files. For example:

```
// After finishing loading the jsx file refered in the manifest.xml,
please use evalScript of CSInterface to load other jsx files.
// "anotherJSXFile" is not the first loaded jsx file, so the value of
"$.fileName" in it's stage is correct.
CSInterface.evalScript('$.evalFile(anotherJSXFile)', callback);

// Or in the first loaded jsx file, load another jsx file, and the
value of "$.fileName" is correct in this file.
// Given the code is running this example.jsx which is referred in the
manifest.xml.
// In the stage of "hardCodeJSXFile", the value of "$.fileName" is
correct too.
$.evalFile(hardCodeJSXFile);
```

3. Please use "namespace" if the developers want to define new variable/function/object in Global Space or "$" object.
   If the same name defined in multiple jsx files, the definition in the last loaded jsx file will take effect, and the definition in the previous loaded jsx files will be overridden. For example, "$.ext" is defined in a.jsx, b.jsx and c.jsx, and in a.jsx "$.ext" is a function, in b.jsx "$.ext" is a object and in c.jsx "$.ext" is a string, and the load sequence is a.jsx->b.jsx->c.jsx, after loading, "$.ext" is string, rather than a object or function. And this behavior will be across multiple extension running in the same point product, for example, if a.jsx, b.jsx and c.jsx belong to extension a,b,c separately , and extension loading order is extension a-> extension b-> extension c, "$.ext" will be still a string, rather than a function or an object.

Drag and Drop

Use Drag and Drop

CEP 5.2 support HTML 5 Drag and Drop. There are four types.

1. Drag and drop inside HTML extension.
2. Drag and drop between two HTML extensions
3. Drag and drop between HTML extension and its host application.
4. Drag and drop between HTML extension and operating system (e.g. Desktop or Browser).

To learn about HTML 5 Drag and Drop and how to use it by JavaScript, please refer to http://www.w3.org/TR/html5/editing.html#dnd.

Here are some demos.

- http://www.w3schools.com/html/html5_draganddrop.asp
- http://html5demos.com/

Disable Drag and Drop

Extension developers can disable the default behavior of DnD by JavaScript.

Method 1

```
<body ondragover="return false" ondrop="return false">
```

Method 2 (using jQuery)

```
$(document.body).on('dragover drop', function(e) {
  e.preventDefault();
});
```

Please read HTML 5 standard for more details.

http://www.w3.org/TR/html5/editing.html#event-dragenter

External JavaScript Libraries

CEP HTML Engine does not restrict using any extension JavaScript libraries. As long as a library can be used in CEF Client or Chrome browser, it should be usable in CEP HTML Engine.

Here are some JavaScript which had been used successfully

- JQuery - http://jquery.com/
    - Please refer to Node.js section about resolving symbol conflicts.
- RequireJS - http://requirejs.org/
    - Please refer to Node.js section about resolving symbol conflicts.
- spin.js - http://fgnass.github.com/spin.js/
- Modernizr - http://modernizr.com/
    - Modernizr is a JavaScript library that detects HTML5 and CSS3 features in the user's browser.

Increase/Decrease font size in HTML Panel

There are couples of JavaScript ways to increase or decrease font size in HTML panel either by plain JavaScript or JQuery. The following is the two pieces of sample snippet to achieve this. One is in plain JavaScript and the other uses JQuery library.

- Plain JavaScript
  Use document.body.style.fontSize to change font size in page.

```
<script type="text/javascript" language="javascript">
    window.onload = function() {
        var fontchange = document.createElement("div");
        var fontchangelink = function(fontsize, desc) {
            var a = document.createElement('a');
            a.href="#";
            a.style.margin = "5px";
            a.onclick = function() {
                document.body.style.fontSize = fontsize + 'pt';
            };
            a.innerHTML = desc;
            return a;
        };

        fontchange.appendChild(document.createTextNode("Change font
size:"));
        fontchange.appendChild(fontchangelink(9, "1"));
        fontchange.appendChild(fontchangelink(11, "2"));
        fontchange.appendChild(fontchangelink(13, "3"));
```

```
            document.body.insertBefore(fontchange,
    document.body.childNodes[0]);
        };
    </script>
```

For the full example, please refer to the HTML file - ResizeFont_plain_js.html

- JQuery
  Use $('html').css('font-size', size) to change font size in page.

```
    <script type="text/javascript" language="javascript">
        $(document).ready(function(){
            // Reset Font Size
            var originalFontSize = $('html').css('font-size');
            $(".resetFont").click(function(){
              $('html').css('font-size', originalFontSize);
            });

            // Increase Font Size
            $(".increaseFont").click(function(){
                var currentFontSize = $('html').css('font-size');
                var currentFontSizeNum = parseFloat(currentFontSize, 10);
                var newFontSize = currentFontSizeNum*1.2;
                $('html').css('font-size', newFontSize);
                return false;
            });

            // Decrease Font Size
            $(".decreaseFont").click(function(){
                var currentFontSize = $('html').css('font-size');
                var currentFontSizeNum = parseFloat(currentFontSize, 10);
                var newFontSize = currentFontSizeNum*0.8;
                $('html').css('font-size', newFontSize);
                return false;
            });
        });
    </script>
```

For the full example, please refer to the HTML file - ResizeFont_JQuery.html

The ways above are the common used solution for JavaScript developers to increase or decrease fonts.

JavaScript Tips

Check Internet Connection

```
if (navigator.onLine === true)
{
  //system is online
}
else
```

```
{
  //system is offline
}
```

Set Mouse Cursor

Please refer to http://jsfiddle.net/BfLAh/1390/

```
$(document).mousemove(function(e){
    $("#image").css({left:e.pageX, top:e.pageY});
});
```

De-obfuscate JavaScript

Please refer to http://jsbeautifier.org/

iframe

Due to the X-Frame-Options header a number of HTTPS websites are unavailable to host in an iframe. An alternative to displaying HTTPS content is to use the window.location.href e.g.

iFrame alternative for HTTPS content
```
// html
<body onLoad="onLoaded()">
<!--iframe src="https://www.trello.com"></iframe--> <!-- this line can be
deleted as HTTPS blocks content being displayed in an iframe -->
</body>

// javascript
function onLoaded() {
  window.location.href = "https://www.trello.com";
}
```

Tooltip

CEP 5.2 supports HTML title attribute to show the tooltip on Windows. However, it's not supported on Mac due to off-screen rendering. The alternative is use JavaScript instead, please refer to http://www.a2zwebhelp.com/bootstrap-tooltips for good examples.

Ports opened in CEPHtmlEngine

If you use TCPView to monitor CEPHtmlEngine process, you may see some ports are opened in localhost. Most of the ports are opened internally in Chromium code for websocket, which is initiated by HTML extension instead of CEPHtmlEngine itself. You can use RawCap to capture the data in *.pcap file and open it in Wireshark to examine the details.

CEF/Chromium Issues

- [HTML <datalist> tag is not supported](#)
- [Can't scroll in DevTools console](#)

More Information for Extension Developers

https://github.com/Adobe-CEP is for external developers, but it is also useful for Adobe internal developers.