CEP/HTML5 Extension

Cookbook for Adobe Bridge CC 8.0 and later

Last updated on

Oct 04, 2017

Contents

Creating CEP 8 Extensions for Bridge

This cookbook is a guide of tips and samples for creating CEP/HTML5 extensions for **Adobe Bridge CC 8.0** and later.

CEP now supported by Bridge

Common Extensibility Platform CEP (formerly known as CSXS - Creative Suite Extensible Services) is a shared technology, which provides a rich platform to create and run HTML5-based extensions across Adobe Creative Cloud products. The extensions created using CEP framework, extend the functionality of the host application that they run in, using HTML5, JavaScript, ExtendScript, and CSS.

With the CEP support, the users can now create CEP extensions for Bridge as well.

Get started with the development

What all is required?

- Adobe Bridge CC version 8.0 and later
- Bridge Sample Extensions: https://github.com/Adobe-CEP/Samples/tree/master/Bridge
- The ZXPSignCmd signing utility: To sign the extensions and create signed .zxp bundles for add-ons or direct distribution

Steps to get started

- Download or clone the github repository.
- Add **PlayerDebugMode** setting for quick testing of unsigned applications during development. Refer to the section Debugging the extension for detailed instructions.
- Copy the samples to the CEP Extension folder and update it as your own extension. Refer to the section Extension Folders for CEP Extensions locations on Win and Mac.
- Launch Bridge. From application menu, go to *Window* > *Extensions*. You can find list of extensions there.
- Sign and package the Extension. Refer to the section Signing and Packaging the Extension for details.

Extension types supported by Bridge

Currently, these three types of CEP extensions that can be used in Bridge - Embedded, Modal Dialog, Modeless Dialog.

- The **Embedded** type of extension behaves like any other Bridge panel. It can be docked, participates in workspaces, has fly-out menus, and is re-opened at start-up if open at application quit. The corresponding type identifier in the extension manifest is **Embedded**.
- The **Modal Dialog** type opens a new window and forces the user to interact with the window before returning control to the host application. The corresponding type identifier in the extension manifest is **ModalDialog**.
- The **Modeless** Dialog type opens a new window but doesn't force the user to interact with it. The corresponding type identifier in the extension manifest is **Modeless**.

CEP runtime supported

Adobe Bridge supports CEP 8.0.

Important Manifest changes for Bridge CEP extensions

- Make sure that you use the correct ExtensionManifest version in the *manifest.xml* of the extension.

```
<ExtensionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ExtensionBundleId="com.adobe.Samples2" ExtensionBundleVersion="1.0" Version="7.0">
```

- Make sure to use the correct HostID and product version for Bridge in the *manifest.xml* of the extension.

```
<HostList>
<Host Name="KBRG" Version="[8.0,99.9]"/
> </HostList>
```

This will support Bridge version 8.0 through 99.9.

- Make sure that you use the correct CEP runtime version.

```
<RequiredRuntimeList>
<RequiredRuntime Name="CSXS" Version="7.0"/>
</RequiredRuntimeList>
```

Sample Manifest XML for Bridge

manifest.xml

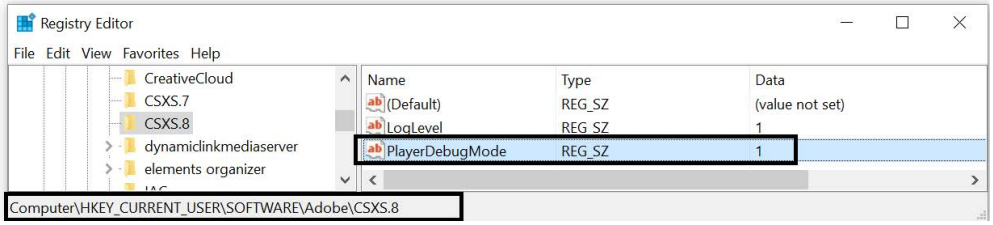
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ExtensionManifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ExtensionBundleId="com.adobe.Samples" ExtensionBundleVersion="1.0" Version="7.0">
    <ExtensionList>
        <Extension Id="BridgeHelpPage" Version="1.0"/>

    </ExtensionList>
    <ExecutionEnvironment>
        <HostList>
            <Host Name="KBRG" Version="[8.0,99.9]" />
        </HostList>
        <LocaleList>
            <Locale Code="All"/>
        </LocaleList>
        <RequiredRuntimeList>
            <RequiredRuntime Name="CSXS" Version="7.0"/>
        </RequiredRuntimeList>
    </ExecutionEnvironment>
    <DispatchInfoList>
        <Extension Id="BridgeHelpPage">
            <DispatchInfo>
                <Resources>
                    <MainPath>./index.html</MainPath>

                </Resources>
                <Lifecycle>
                    <AutoVisible>true</AutoVisible>
                </Lifecycle>
                <UI>
                    <Type>Embedded</Type>
                    <Menu>Bridge Help</Menu>
                    <Geometry>
                        <Size>
                            <Height>580</Height>
                            <Width>1020</Width>
                        </Size>
                        <MaxSize>
                            <Height>800</Height>
                            <Width>1200</Width>
                        </MaxSize>
                        <MinSize>
                            <Height>400</Height>
                            <Width>600</Width>
                        </MinSize>
                    </Geometry>
                </UI>
            </DispatchInfo>
        </Extension>

    </DispatchInfoList>
</ExtensionManifest>
```

Debugging the extension

Using unsigned extensions during development

If you are in the midst of development and want to bypass the need to sign your extensions, you can do so by editing the CSXS preference properties file or registry entry.

Windows



To edit the registry entry:

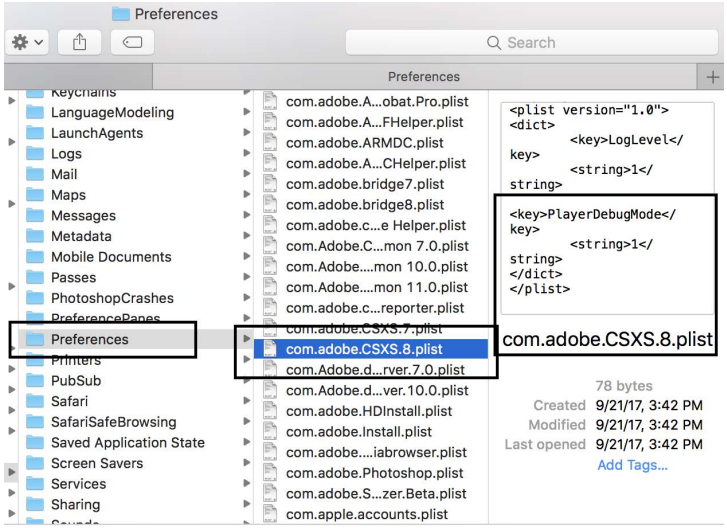
1. Open Registry Editor.

2. Go to HKEY_CURRENT_USER/Software/Adobe/CSXS.8, then add a new entry PlayerDebugMode of type *string* with the value of *1*.

3. Relaunch Bridge.

The extension will work without signing.

Mac



1. In the terminal, type: *defaults write com.adobe.CSXS.8 PlayerDebugMode 1*

2. Restart Mac and Launch Bridge.

The Extension should work without signing.

Note: The plist is also located at /Users/<username>/Library/Preferences/com.adobe.CSXS.8.plist.

Debugging in Chrome

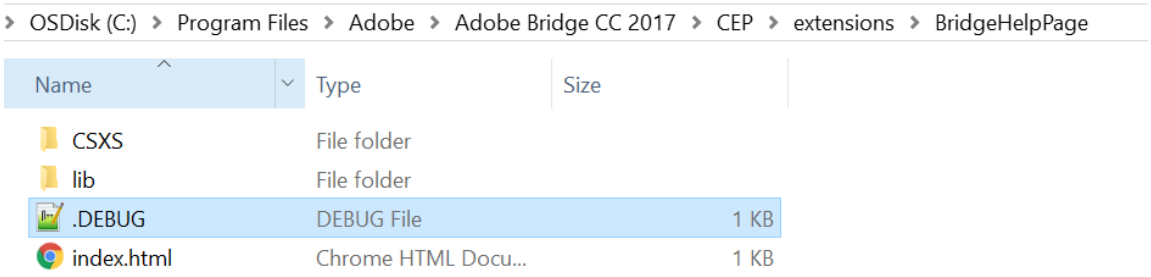
CEP supports remote debugging for HTML extensions. You can use Chrome developer tools to debug your Extension.

Step 1. Create a ".debug" file in 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.

This file name is special for both Windows and Mac platforms, it has to be created via command line.

- (Windows) Open cmd, go to the path of your extension and type "copy con .debug" and "Ctrl+Z" to create an empty file.
- (Mac) Use "touch .debug" to create an empty file.

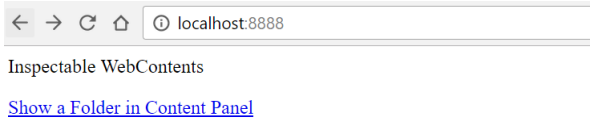


Step 2. Edit the .debug file. It should resemble the values shown below and the Extension ID must match the one in the panel's manifest. The value of Port should be between 1024 and 65535 (excluding 65535), otherwise remote debugging and the dev tools will not work.

.debug

```
<?xml version="1.0" encoding="UTF-8"?>
<ExtensionList>
    <Extension Id="ShowFolder">
        <HostList>
           <Host Name="KBRG" Port="8888"/>
        </HostList>
    </Extension>
</ExtensionList>
```

Step 3. Launch your application and open the Extension.Open Chrome and navigate to http://localhost:8888. Your Extension should be visible.



Log locations

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 convention:

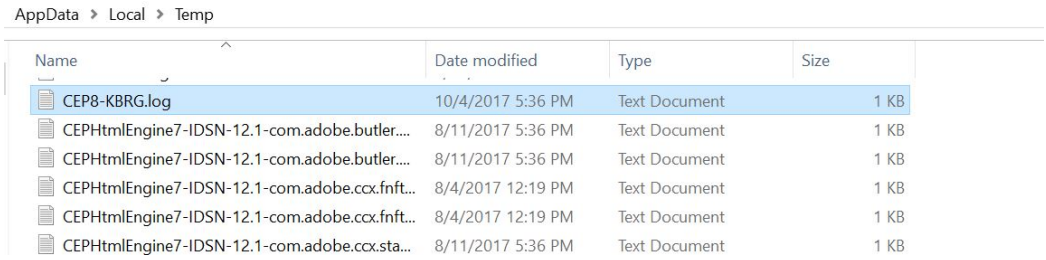
CEP<versionNumber>-<HostID>.log. For example- CEP8-KBRG.log

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.8
- Mac: /Users/USERNAME/Library/Preferences/com.adobe.CSXS.8.plist



Signing and Packaging the Extension

After testing your extension thoroughly, you must package and sign your extension so that the users can install it on their systems.

Adobe provides a command-line tool *ZXPSignCmd*, which can be used to package and sign extensions so they can be installed in Adobe desktop applications. The signature verifies that the package has not been altered since its packaging.

You can use **ZXPSignCmd** to do the following:

Create a self-signed certificate

*Example: ./ZXPSignCmd -**selfSignedCert** US NY MyCompany MyCommonName abc123 MyCert.p12*

This generates a file named MyCert.p12 in the current folder. You can use this certificate to sign your extension.

Create a signed ZXP package

*Example: ./ZXPSignCmd -**sign** myExtProject myExtension.zxp MyCert.p12 abc123*

This generates the file myExtension.zxp in the current folder, adding these two files to the packaged and signed extension in the final ZXP archive:

1. mimetype : A file with the ASCII name of mimetype that holds the MIME type for the ZIP container (application/vnd.adobe.air-ucf-package+zip).

2. signatures.xml: A file in the META-INF directory at the root level of the container file system that holds digital signatures of the container and its contents.

Verify an existing ZXP package

Example: ./ZXPSignCmd -verify myExtProject myExtension.zxp

This shows a message 'Signature verified successfully' if the signature is valid.

More information about signing and packaging can be found at Extension Signing-Tech Note.

Extension Folders

CEP supports three types of extension folders. For Bridge, the location of these folders is mentioned below:

Product extension folder

- Win(x86) : C:\Program Files\Adobe\<Bridge Product Folder>\CEP\extensions
- Win(x64) : C:\Program Files\Adobe\<Bridge Product Folder>CEP\extensions
- Mac: /Applications/<Bridge Product Folder>/CEP/extensions

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

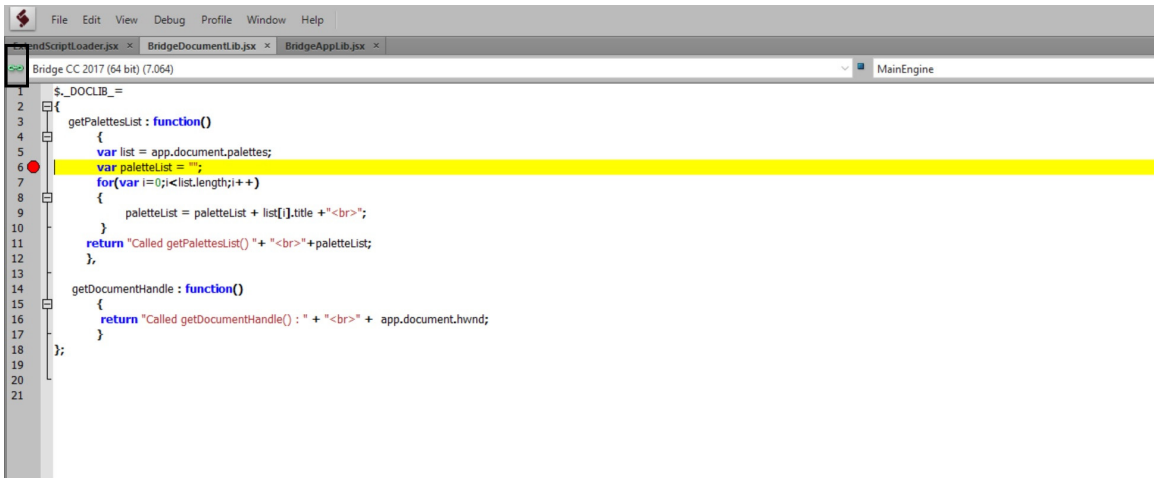
- Win: C:\Users\<USERNAME>\AppData\Roaming\Adobe\CEP\extensions
- Mac: ~/Library/Application Support/Adobe/CEP/extensions

Using ExtendScript in a CEP Extension

ExtendScript code for the Host Application can be called from a CEP Extension inside the HTML/Java script code in many ways using the EvalScript() function. ExtendScript code opens up all the Scripting API of the Host Application to CEP Extension developers. The ExtendScript code (.JSX) can span across multiple files and there are many ways to load them inside the CEP Extension.

To debug ExtendScript using the ExtendScript ToolKit CC:

1. Open the JSX script in ExtendScript Toolkit CC.
2. Connect it with Adobe Bridge CC 8.0 or later and put a debug point in script.
3. Open the CEP Extension and call the EvalScript code. You should be able to debug the ExtendScript code.



Using NodeJS in a CEP Extension

In CEP, Node.js is disabled by default. To enable NodeJS, you need to add the parameters listed below in your CEP extension Manifest file.

Under the *resources* tag, add the following:

```
<CEFCommandLine>
<Parameter>--enable-nodejs</Parameter>
</CEFCommandLine>
```

Example:

```
<DispatchInfo>
<Resources>
<MainPath>./index.html</MainPath>
<CEFCommandLine>
<Parameter>--enable-nodejs</Parameter>
</CEFCommandLine>
</Resources>
```

Important resources

Resource name	**Link**
Adobe CEP Resources	https://github.com/Adobe-CEP/CEP-Resources
Adobe ExtendScript Tools Guide	https://www.adobe.com/content/dam/Adobe/en/devnet/scripting/pdfs/javascript_tools_guide.pdf
Application Signing TechNote	http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/creativesuite/pdfs/SigningTechNote_CC.pdf
Bridge CEP Samples	https://github.com/Adobe-CEP/Samples/tree/master/Bridge
Bridge Scripting Reference	http://www.adobe.com/devnet/bridge.html
CEP 7.0 HTML CookBook	https://github.com/Adobe-CEP/CEP-Resources/blob/master/CEP_7.x/CEP_7.0_HTML_Extension_Cookbook.pdf