PACKAGING, SIGNING, AND DEPLOYING EXTENSIONS

WITH EXTENSION MANAGER CS5: TECHNICAL NOTE

Technical Note: Packaging, Signing, and Deploying Extensions with Extension Manager CS5

Adobe Systems Inc., 345 Park Avenue, San Jose, California 95110, USA.

Contents

Packaging, Signing, and Deploying Extensions

This document provides extension developers with the information necessary to package and sign extensions so they can be installed in Adobe® Creative Suite® applications using Adobe Extension Manager CS5. This document comes with the UCF tool, a command-line tool used to create Universal Container Format (UCF) packages.

If you are creating an Enterprise Deployment of Creative Suite that includes extensions, you can use Adobe Extension Manager CS5 to silently deploy the signed package; see "Installing an extension programmatically" on page 9. The package you create for deployment with Adobe Extension Manager CS5 is an archive of type ZXP.

There are three types of extension that you can package and sign using the UCF tool:

- Ordinary extensions

  Any Adobe product-specific extension or plug-in that extends the functionality of an Adobe product. Ordinary extensions were previously packaged as MXP files via Extension Manager, and require an MXI file, the extension installation file (such as a Dreamweaver® extension or an InDesign® or Photoshop® C++ plug-in).

- Creative Suite extensions

  Flash-based extensions that can be installed and run in multiple Creative Suite products, built using the Creative Suite SDK.

- Hybrid extensions

  A package that contains both an ordinary extension and a Creative Suite extension. Hybrid extensions are used when the feature developed needs both a Creative Suite Flash-based component and a native or script extension. This allows developers to build extensions with rich Flash-based interfaces and still take advantage of the extended native integration with the application.

How to package and sign an extension

After testing your extension thoroughly, you must package and sign your extension so users can install it in their systems using Extension Manager. To prepare for this step, it is recommended that you copy all of the files for your extension to a staging folder for ease of packaging.

- Ordinary extension

  Create an extension installation file (a filename ending in `.mxi`) for your extension. The MXI file is an XML file that specifies attributes of the extension, including the extension name, a description of the extension, version number, and type. The file also specifies each file included in the extension, including any custom icon you want to use.

  For more information, see the Extension Installation File Format document, which you can download from the Adobe website at http://www.adobe.com/go/em_file_format.

- Creative Suite extension

  If you are building Flash-based Creative Suite extensions using the Creative Suite SDK, we recommend you export your extension using the provided tools. Follow the instructions in the Creative Suite SDK documentation on how to deploy an extension.

  If you prefer to package and sign the extension manually (see "Using the UCF tool " on page 5) or from a build system, make sure the staging folder contains a subfolder names `csxs/`, which contains the `manifest.xml` file:

  ```
  /csxs/manifest.xml
  ```

  You can add any extra resources to the root or to a folder within the root folder; verify that the relative paths in the manifest (`manifest.xml`) are correct. For example:

  ```
  /HelloWorld.jsx
  /HelloWorld.png
  /HelloWorld.swf
  ```

- Hybrid extension

  To package a hybrid extension, you must prepare the ordinary extension portion for packaging as described above. Package and sign the Creative Suite extension portion separately, then add the signed package to the root of the staging folder for the ordinary extension.

  - Add the following line to the `<files>` element in the MXI file, specifying the source file for the Creative Suite extension package.

    ```
    <file source="myCSExtension.zxp" destination="" file-type="CSXS" />
    ```

  - Make sure that the extension name in the MXI file describes the hybrid extension, not just the ordinary extension component.

Using the UCF tool

To package an extension manually, use the Universal Container Format (UCF) command-line tool. The resulting UCF files can optionally be signed.

The Adobe UCF tool is implemented in Java; the JAR file is packaged with this document. Running the tool requires that the `java` command is available in your shell's path. UCF requires JRE 1.5 or newer to run, but JRE 6 is recommended. This is the default in MacOS X; in Windows, you must install JRE 1.5 or better.

Invoke UCF directly using the JAR file:

```
java -jar ucf.jar ...
```

Packaging options

```
java –jar ucf.jar -package name (fileOrDir*)|( -C dir fileOrDir+ )|( -e file path ))*)
```

You can provide the following packaging options:

`-package` *`name fileOrDir*`*	Directs UCF to construct the package. The arguments specify the name of the UCF file to be created, followed by the list of files and folders to include. This list can be omitted if the `-C` or `-e` flags are used, as described below. When the file list is provided directly, path specifications are relative to the current working directory, and only the relative paths are placed in the package. Absolute paths are not permitted in UCF files. Any directories are added recursively; any files marked as "hidden" are ignored.

`-C` *`dir fileOrDir`*	Use to specify files relative to a directory other than the current working directory. The option is followed first by the path to that directory, then by one or more files or directories to be added relative to that location. This flag can be specified any number of times.
	You cannot use shell wildcard character such as * with this option, because the shell always expands wildcards in the current directory, not in the target directory of the -C option. To recursively include the contents of a directory other than the current directory, use the syntax:
	`-C` *`dir`* `.`
	The final dot instructs UCF to include the specified directory and all of its contents. For example:
	`java -jar ucf.jar -package ordinary.zxp -C "./ordinary" .`

`-e`	Use to include files, but remap them to different paths within the package. For example, the following includes the file `/tmp/out.tmp` but stores it in the `main` directory within the package: `java -jar ucf.jar -package package.zxp -e /tmp/out.tmp main` You can only remap files; you cannot use this option to remap directories.

Signing options

You can provide the following options to sign the created package. These are all optional and can be specified in any order.

`-alias`	If specified, the alias of a particular key in the specified keystore. If not specified, the first key returned by the keystore is used.
`-storetype`	The type of keystore to use. Common values are `JKS`, `PKCS11`, and `PKCS12`. Values are case-insensitive. If not specified, the default `JCA` provider is used.

`-keystore`	Applies only to file-based keystore types, such as JKS and PKCS12. For applicable keystore types, specifies the path to the file containing the keystore. Default depends on the keystore type. Behavior when specified with non-file-based keystores is undefined.
`-storepass`	The password used to protect the keystore; this can be different from the password used to protect a specific key. If not specified, UCF prompts at the console for a keystore password.
`-keypass`	The password used to protect the selected key.; this can be different from the password used to protect the keystore. If not specified, UCF prompts at the console for a key password.
`-providerName`	Selects a particular provider of the given keystore type. If not specified, the default provider for the specified type is used.
`-tsa`	TheURL for the RFC3161-compliant time-stamping server, used to timestamp the signature. Default is https://timestamp.geotrust.com/tsa. To disable time stamping, use the special value "none".

For example, suppose you want to package and sign an extension of any type that has been staged for packaging in a folder called `myExtension` in the current working directory. Use this shell command:

```
java -jar ucf.jar -package -storetype PKCS12 -keystore myCert.pfx -storepass mypasswd
   myExtension.zxp -C "./myExtension" .
```

This create a package named `myExtension.zxp`, signed with the `myCert.pfx` certificate.

After packaging and signing the extension these two files are added to the final ZXP archive:

- `mimetype`

  A file with the ASCII name of `mimetype`, which holds the MIME type for the Zip container (`application/vnd.adobe.air-ucf-package+zip`).

- `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.

Installing a packaged and signed extension

The Adobe Extension Manager CS5, a tool that is included with all Creative Suite applications, installs extensions that are properly packaged and signed. Adobe Extension Manager CS5 is installed at the same time as CS5 applications; you can also launch it from the Start menu in Windows or the Applications folder in Mac OS.

Using the Extension Manager CS5

To install the signed ZXP file follow these steps:

1. Open Extension Manager CS5 and click **Install**.
2. Browse to the location where your ZXP file is saved, select it, and click **Open** to start the installation process.

3. The Extension Manager attempts to validate the package against the signature. For some validation results, it prompts the user to decide whether to continue with the installation; for example, if it cannot verify the publisher, you can choose to install the extension anyway.
4. Once the installation has completed, check that your extension appears in all of the products that it supports.

Troubleshooting the installation

If your package fails to install properly:

- Verify that you have built your extension with the correct structure.
- Verify that your extension package contains the correct files in the correct locations.
- For a Creative Suite extension, verify that the extension's manifest file has been configured correctly; In particular, check that the host name and version match the application in which you are looking for it.
- For ordinary and hybrid extension, verify that the MXI file has the correct configuration.
- Verify that the package has not been modified since being properly signed.

Because the ZXP is an archive file, you can rename the package with the `.zip` extension to examine its contents and verify that it contains all needed files. If you change anything in it, however, the signature no longer matches the content, and the Extension Manager cannot load the package. If you need to make changes, you must create and sign a new package.

Extension Manager behaviour based on the certificate used

The signature verifies that the package has not been altered since its packaging. When the Extension Manager tries to install a package, it validates the package against the signature, and checks for a valid certificate. For some validation results, it prompts the user to decide whether to continue with the installation. These are the possible validation results:

Signature	**Signing certificate**	**Extension Manager action**
No signature	N/A	For Creative Suite extensions, shows error dialog and aborts installation. For other extensions/plug-ins and hybrid extensions, depending on Extension Manager preference settings, either silently installs the extension or prompts user for permission to continue the installation.
Signature invalid	Any certificate	Shows error dialog and aborts installation.
Signature valid	Adobe certificate	Silently installs extension.
	OS-trusted certificate	Silently installs extension.
	Other certificate (including self-signed)	Prompts user for permission to continue the installation.

For a better user experience we recommend you use an OS-trusted certificate. This is a certificate that is stored in the system certificate store as a trusted certificate, or for which it is possible to establish a certificate chain from the signing certificate to some trusted certificate in the system store. Certificates issued by a known trusted certificate authority (TCA) are normally OS-trusted certificates.

Installing an extension programmatically

You can invoke the Extension Manager programmatically, as part of an Enterprise Deployment of Creative Suite. By invoking it with the correct options, you can do so without opening the UI window, and use the tool to automatically install an extension package.

Invoking the Extension Manager

In the folder in which it is installed, invoke the Extension Manager executable from the command line. Follow the executable name with the invocation options that perform the desired startup behavior and configuration:

IN WINDOWS:

To run the Extension Manager without the UI, and returns error codes that report the result, open a command shell in the installation folder and run this command:

```
XManCommand.exe -suppress options
```

To invoke the Extension Manager normally, without returning error codes, you can run the product executable :

```
"Adobe Extension Manager CS5.exe" options
```

IN MAC OS:

Use the full installation location; note that the name contains spaces, so you must use quotes. :

```
"/Applications/Adobe Extension Manager CS5/Adobe Extension Manager
CS5.app/Contents/MacOS/Adobe Extension Manager CS5" options
```

Command-line options

You can provide the following options:

`-suppress`	Does not show the Extension Manager’s UI window. If another instance of the tool is already running, this command is performed by that instance, but without the EULA and other dialogs related to the request. When used, this must be specified before the operation parameter.
`-install mxi|mxp|zxp ="name"`	Installs the specified extension.

`-remove`
`product|productfamily="name" extension="name"`
	Removes the specified extension from the specified product or product family.

`-enable`
`product|productfamily="name" extension="name"`
	Enables the specified extension in the specified product or product family.

`-disable`
`product|productfamily="name" extension="name"`
	Disables the specified extension in the specified product or product family.

`-package`
`mxi="name"`
`mxp|zxp="name"`
	Repackages the specified MXI as an unsigned MXP or ZXP package.
	It is recommended that use the UCF tool to create signed packages for Creative Suite extensions.

`-locate product="name"`	Brings up the Extension Manager's UI showing the specified product. Do not use with `-suppress`.
`-locale lang="locale_code"`	Brings up the Extension Manager's UI in a specific language. Do not use with `-suppress`.

Examples

The first example shows both platform versions; the tool invocation and file references should always be platform-appropriate.

Use silently for deployment tasks

These examples operate without launching the Extension Manager UI window; or, if the window is already shown, perform their tasks without bringing up any additional dialogs.

Task	Command

Install a signed extension	▶ In Windows
	`XManCommand.exe" -suppress -install zxp="C:\test.zxp"`
	▶ In Mac OS
	`"/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -suppress -install zxp="Volumes/x1/test.zxp"`

Remove an extension	`XManCommand -suppress -remove product="Dreamweaver CS5" extension="Sample"`
	`XManCommand -suppress -remove productfamily="Photoshop-12" extension="Sample"`

Enable/disable an extension	`XManCommand -suppress -enable product="Dreamweaver CS5" extension="Sample"`
	`XManCommand -suppress -disable productfamily="Photoshop-12" extension="Sample"`

Launch the Extension Manager

You can also use the full product executable for these operations.

Task	Command
Show specific product	`XManCommand.exe -locate product="Dreamweaver CS5"`
In specific language, install extension at the same time	`XManCommand.exe -locale lang="en_US" -install zxp="C:\test.zxp"`

Invoking the Extension Manager through interapplication communication

The Extension Manager can execute a command passed from an application through the `BridgeTalk` object. When you send a command this way, specify the target as with application identifier `exman-5.0`, and pass `-EMBT` (a `BridgeTalk` option) as the first parameter.

For example, you can run this script in ExtendScript Toolkit CS5 to install an extension in Mac OS without bringing up the Extension Manager UI:

```
var bt = new BridgeTalk();
bt.target = "exman-5.0";
bt.body = '-EMBT -suppress -from product="Dreamweaver CS5"
    -install zxp="Volumes/x1/test.zxp"
bt.send();
```

Errors

When you suppress the Extension Manager UI and use the command-line tool to operate directly on extensions as part of your deployment, the tool can return these error codes, while sending a detailed error message to standard error output:

Error code	Description
0	Success.
1	Install operation failed.
2	Remove operation failed.
3	Enable operation failed.
4	Disable operation failed.
5	Package operation failed.
7	Another instance of Extension Manager is already active.
101	Command-line format incorrect.
102	The specified product is not found.
103	The specified extension is not found.

104	The specified extension is already enabled.
105	The specified extension is already disabled.