

# Packaging Extensions with ADOBE® EXTENSION MANAGER CS5



© 2010 Adobe Systems Incorporated. All rights reserved.

Using Adobe® Extension Manager CS5 for Windows® and Mac OS

This user guide is protected under copyright law, furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

This user guide is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the user guide for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the user guide; and (2) any reuse or distribution of the user guide contains a notice that use of the user guide is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>.

Adobe, the Adobe logo, Adobe Bridge, Adobe Premiere Pro, Contribute, Creative Suite, Dreamweaver, Fireworks, Flash, Illustrator, InCopy, InDesign, Kuler, and Photoshop are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Windows is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries. Mac and Mac OS are trademarks of Apple Inc., registered in the U.S. and other countries. All other trademarks are the property of their respective owners.

Updated Information/Additional Third Party Code Information available at <http://www.adobe.com/go/thirdparty>.

Portions include software under the following terms:

This product contains either BSAFE and/or TIPEM software by RSA Security, Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Flash video compression and decompression is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved. <http://www.on2.com>.

Portions licensed from Nellymoser ([www.nellymoser.com](http://www.nellymoser.com)).

**Sorenson  
Spark**

Sorenson Spark® video compression and decompression technology licensed from Sorenson Media, Inc.

MPEG Layer-3 audio compression technology licensed by Fraunhofer IIS and THOMSON multimedia (<http://www.iis.fhg.de/amm/>).

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

Notice to U.S. Government End Users: The Software and Documentation are “Commercial Items,” as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Packaging Extensions with Extension Manager CS5

This document describes how to package extensions with Extension Manager CS5. The development and creation of extensions is covered in other documents.

To package extensions, you need to do the following:

- 1 Choose the extension package format
- 2 Create the extension installation file
- 3 Create the extension package from the Extension Manager user interface or command line.

In this document, most instructions are for the command-line interface. See [Use Extension Manager from the command line](#).

For information on packaging extensions using the Extension Manager user interface, see [Use Extension Manager from the graphical user interface \(GUI\)](#).

## Choose the extension package format

Extension Manager CS5 supports various extension formats.

Extensions can be packaged into two file formats:

- MXP extensions (with filename extension `.mxp`) for CS4 and earlier.
- ZXP extensions (with filename extension `.zxp`) for CS5. The ZXP format is based on the ZIP standard, and extensions in this format can be digitally signed to verify the identity of the extension author. For more information, see [Creating a ZXP extension package \(version 5.0 only\) on page 56](#)

From extension function perspective, there are three kinds of extensions:

- Ordinary extensions:  
Any Adobe product-specific extension or plug-in that extends the functionality of an Adobe application— such as a Dreamweaver extension or a Photoshop C++ plug-in. Ordinary extensions were previously packaged as MXP files with Extension Manager CS4 or before, and they require an extension installation (`.mxi`) file.
- Creative Suite extensions (hereafter called *CS extensions*):  
Flash-based extensions that can be installed and run in multiple Creative Suite applications, built using the Creative Suite SDK. A CS extension must be a ZXP file, but it doesn't require an extension installation (`.mxi`) file. You can't create a CS extension with Extension Manager, but such an extension can be installed or removed by Extension Manager CS5.
- Hybrid extensions:  
A package that contains both files that can be contained in an ordinary extension and a CS extension. Hybrid extensions are used when the feature developed needs both a Creative Suite Flash-based component and a native C++ plug-in or script file. This allows developers to build extensions with rich Flash-based interfaces and still take advantage of the extended native integration with the application. For more information, see [Creating Hybrid Extension packages \(version 5.0 only\) on page 57](#).

Consider the following when determining whether to use the ZXP or MXP format to package your extensions:

- If you want to create an ordinary extension, you can use either ZXP or MXP.
- If your extension is not designed for CS4 and earlier, you should use ZXP.
- If you want to create a hybrid extension, you should use ZXP.
- If you want to digitally sign the extension, you must use ZXP. For more information, see [Creating a ZXP extension package \(version 5.0 only\) on page 56](#)

For information on digitally signing extensions, refer to the Creative Suite SDK:

<http://www.adobe.com/devnet/creativesuite/sdk>

## Create the extension installation file

An extension installation file is an XML file (with the extension .mxi) that provides the following information about the extension for Extension Manager:

- extension name and version number
- information about each file included in the extension, including where each file is installed
- information about how users access the extension from an Adobe application, such as product-specific information about menu items or other user interface items to add
- information about language-specific files for multilingual extensions
- information about update information for updatable extension packages

When you create your extension installation file, give it a filename that is valid on both Windows and Mac OS, which is no more than 20 characters long, and contains no spaces.

This section describes the tags used in the installation file. For a list of each tag and compatible Adobe applications, see [Tags and their compatible products on page 51](#). After reading about these tags, you can examine the sample installation file in the Extension Manager's Samples folder, or you can make a copy of the blank installation file and fill in values for the attributes.

This section contains the following:

- [About careful XML coding on page 7](#)
- [MXI tag summary on page 7](#)
- [MXI tag descriptions on page 9](#)
- [Tags and their compatible products on page 51](#)
- [Example MXI file on page 53](#)
- [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#)
- [Creating a ZXP extension package \(version 5.0 only\) on page 56](#)
- [Creating Hybrid Extension packages \(version 5.0 only\) on page 57](#)
- [Creating plug-in extension packages for InDesign CS5 and InCopy CS5 \(version 5.0 only\) on page 58](#)
- [Creating updatable extension packages \(version 5.0 only\) on page 59](#)

## About careful XML coding

XML files have fairly strict syntax requirements. When you're creating or editing an extension installation file, make sure that you use correct XML syntax:

- Every attribute value must be enclosed in a single pair of double quotation marks. For example, `version = 1.0.0` and `version = "1.0.0"` are both incorrect syntax; instead, use `version = "1.0.0"`.
- A tag defined as an empty tag (a tag with no contents) must end with `/>`. Do not include any spaces between the slash and the closing angle bracket.
- Each attribute name must be preceded by a space (or other form of white space). In particular, if you use more than one attribute in a tag, you must put a space between each attribute's value and the next attribute's name.
- XML does not support special characters such as ampersands (&). To include an ampersand within a tag, you must use the code `&amp;` (for instance, to use ampersands in menu items or other UI elements).

### Encoding characters

If Extension Manager 5.0 can't get the explicit encoding information from the MXI file, Extension Manager assumes that the MXI is encoded with the operating system default code page. To avoid confusion, it is recommended to encode MXI with UTF-8 and explicitly declare the encoding of MXI as UTF-8.

To declare that the MXI is encoded with UTF-8, do the following:

- (Windows) Put the UTF-8 representation of the BOM at the head of the MXI file. The easiest way to make sure of the presence of the BOM is to open the MXI file with Windows built-in program Notepad and select File > Save As, then set Encoding as UTF-8. Note that it is not enough to just specify `<?xml version="1.0" encoding="UTF-8"?>`
- (Mac OS) Put the UTF-8 encoding declaration at the head of the MXI file. It is `<?xml version="1.0" encoding="UTF-8"?>`.

### MXI tag summary

The following table lists the primary tags available in the MXI file, briefly describes each tag, and specifies whether the tag contains child tags. Use this table to get an overview of what tags are available and what functions they perform.

Tag	Description	Contains Child Tags?
<code>macromedia-extension</code>	Main tag for extension installation file.	Yes
<code>defaultLanguage</code>	Default for multilingual extensions (version 2.1 and later).	No
<code>author</code>	Name of the extension's author.	No
<code>description</code>	Describes what the extension does.	No
<code>update</code>	Specifies extension update information (version 5.0 only).	No

<b>Tag</b>	<b>Description</b>	<b>Contains Child Tags?</b>
products	Container tag that contains tags specifying an extension's compatibility.	Yes
license-agreement	Allows a third-party developer to include a license agreement that is displayed at installation.	No
ui-access	Specifies the text that will appear in the Extension Manager window when the extension is selected.	No
files	Container tag that contains tags describing the files an extension installs.	Yes
configuration-changes	Container tag for tags that modify the application's configuration. These include menus, shortcuts, server behaviors, and data sources.	Yes
documenttype-changes	Describes changes made to the MMdocumentTypes.xml file.	Yes
toolpanel-changes	Marks the beginning of Flash toolpanel changes.	Yes
ftp-extension-map-changes	Specifies a change to the FTPExtensionMap.txt file. This defines whether the file is downloaded or uploaded as an ASCII or binary file from Dreamweaver to an FTP server.	Yes
insertbar-changes	Specifies changes to be made to the insertbar.xml file and add new toolbars files.	Yes
server-behavior-changes	Container tag for changes to menus in the menus.xml file in any of the Dreamweaver MX Configuration/ServerBehaviors/document_type folders.	Yes
server-format-changes	Container tag for changes to menus in the menus.xml file in any of the Dreamweaver MX Configuration/ServerFormats/document_type folders	Yes
data-source-changes	Container tag for changes to menus in the menus.xml file in any of the Dreamweaver MX Configuration/DataSources/document_type folders.	Yes
menu-remove	Provides information about a menu bar, menu, menu item, or format to remove during installation of an extension.	No
menu-insert	Specifies where in the application's menus to insert a menu bar, menu, menu item, or format during installation of an extension.	No
menubar	Provides information about a menu bar to be inserted into the application's menu structure during installation of this extension.	No
menu	Describes a menu or submenu to be inserted into the application's menu structure during installation of an extension.	No

Tag	Description	Contains Child Tags?
<code>menuitem</code>	Describes the menu item to be inserted into the application's menu structure during installation of an extension.	No
<code>format</code>	Describes the data format to be inserted into the Dreamweaver Format menu during installation of the extension.	No
<code>separator</code>	Specifies that a separator be inserted into a menu at the location indicated by the containing <code>menu-insert</code> tag.	No
<code>comment</code>	Provides a comment about an item being inserted into the menu structure. The Extension Manager inserts this comment (in the form of an XML comment tag) into the <code>menus.xml</code> file as it installs the extension.	No
<code>shortcut-remove</code>	Indicates that the specified keyboard shortcut should be removed from the <code>menus.xml</code> file.	No
<code>shortcut-insert</code>	Indicates that a keyboard shortcut should be added to the <code>menus.xml</code> file.	No
<code>shortcut</code>	Specifies a keyboard shortcut to be added to the <code>menus.xml</code> file.	No
<code>taglibrary-changes</code>	Describes changes to be made to the <code>TagLibraries.vtm</code> file.	Yes
<code>toolbar-changes</code>	Inserts the specified tag library at the end of the file.	Yes
<code>extensions-changes</code>	Container tag that describes any changes to the <code>Extensions.txt</code> file, such as adding or removing extensions that you can open in Dreamweaver.	Yes
<code>Tags and their compatible products</code>	Container tag that allows you to specify tokens. Tokens let you specify the destination folder of one or more files from your extension during installation or provide a dialog box for the user to choose a destination folder for certain files.	Yes
<code>token</code>	Defines a custom token for an extension. Custom tokens let you specify the destination folder of one or more files from an extension during installation, or provide a dialog box for the user to choose a destination folder.	No

## MXI tag descriptions

The tags used in the extension installation file are described below. Attribute names enclosed in curly braces ( { } ) are optional. The tags are listed according to their position with the MXI file hierarchy. For example, the `macromedia-extension` tag is the main tag within the file, and is the first tag described.

## macromedia-extension

### Description

Main tag for extension installation file.

### Attributes

`id`, `name`, `version`, `{type}`, `{requires-restart}`, `{ismultilingual}`, `{name_resid}`, `{plugin-manager-type}`, `{show-files}`, `{force-quit}`

`id` A unique extension ID, to be created by Adobe after you submit your extension. Never add or modify this attribute yourself.

`name` The name of the extension. This must be a `VARCHAR` data type with a limit of 255 characters.

`version` The version number of the extension, in the format `a{.b{.c}}`, where `a`, `b`, and `c` are all positive integers. For example, valid version numbers include 1.3.6, and 10.0.1. The first number is the major version number, incremented when you make substantial changes to the extension; the second number is the minor version number, incremented for smaller changes. The third number is incremented for each new “build” of the extension between releases; for example, after you submit version 4.1 of your extension to Adobe, it may be returned to you for minor corrections. You might label the fixed version 4.1.1; after a couple of rounds of corrections, the version number of the posted extension might be 4.1.3.

`mxiversion` Attribute for version 5.0 that degrades gracefully. Indicates the version of the MXI specification. Extension Manager 5.0 supports `mxiversion` 5.0 and earlier. If `mxiversion` is unspecified, the default value is 1.0.

*Note:* If the specified `mxiversion` is more recent than the current Extension Manager, an alert appears during installation, indicating that a newer Extension Manager is required.

`xmanversion` Attribute for version 5.0 that degrades gracefully. Indicates the oldest version of Extension Manager that can install this extension.

*Note:* Specify the `xmanversion` attribute only if a newer version of Extension Manager doesn’t support the extension. To indicate the application versions that are compatible with an extension, see the `maxversion` attribute for [product on page 16](#).

`icon` Attribute for version 5.0 that does not degrade gracefully. Indicates the path to customized extension icon displayed by Extension Manager. To specify this relative path, use the `$ExtensionSpecificEMStore` attribute. For more information, see the `destination` attribute for [file on page 19](#). If `icon` is unspecified, the default icon is used.

*Note:* The `icon` attribute applies only to CS4 applications and later. To specify an icon for CS3 or earlier applications, use the `type` attribute.

`type` Indicates the kind of extension. This optional attribute applies only to Dreamweaver, Fireworks, and Flash.

- Valid values for Dreamweaver:

behavior, browserprofile, codehint codesnippet, coloringscheme, command, connection, datasource, dictionary, documenttype, encoding, flashbuttonstyle, flashelement, floater, insertbar, jsextension, keyboard shortcut, object, plugin, propertyinspector, report, referencebook, samplecontent, serverbehavior, serverformat, servermodel, site, suite, taglibrary, template, thirdpartytags, toolbar, translator, utility, query

- **Valid values for Fireworks:**

autoshape, command, commandpanel, dictionary, keyboard shortcut, library, pattern, texture

- **Valid values for Flash:**

actionsript, flashcomponent, flashcustomaction, flashimporter, flashpanel, flashtemplate, generatorobject, keyboardshortcut, lesson, library, publishtemplate, sample, smartclip, utility

**Extensions of type "generatorobject" are supported only by Flash 5 and earlier. Values are not case-sensitive; "object" is equivalent to "Object".**

*Note:* The value "suite" denotes a set of items that are released as a unit, in a single MXP file, with a single MXI file. For example, you can create a set of objects, a command, a palette, and behaviors to make a process such as layer alignment easier to complete. Specify a single name and version for the entire suite.

`requires-restart` Indicates whether the Adobe application must be restarted after the extension is installed. Valid values are "true" and "false". A new attribute `force-quit` introduced in CS5 has a similar function; it is recommended to use the new attribute `force-quit`.

`ismultilingual` If this attribute is false, all multilingual elements are ignored. If it is true, multilingual elements install language-specific files and apply related configuration changes. If this attribute is not specified, it is considered false. For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

`name_resid` References string with value of "name\_resid" in resource file (see `isresourcefile` attribute for [file on page 19](#)), and displays that string in the "extension" field of the user interface. For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

`plugin-manager-type` Indicates the type of plug-ins included in the extension. Valid values are "all-users" and "current-user". For more information, see [Creating plug-in extension packages for InDesign CS5 and InCopy CS5 \(version 5.0 only\) on page 58](#).

`show-files` This attribute only works for extensions for InDesign CS5 and InCopy CS5. If this attribute is "true", path information for all of the files installed with the extension is shown in the "Advanced" tab in the bottom portion of the Extension Manager workspace. If this attribute is "false", no path information is shown. The default value is "true".

`force-quit` Specifies whether the extension target application needs to quit before an operation such as installation or removal. Operations like installation or removal of some extensions may conflict with the running target application. Setting this attribute as "true" informs Extension Manager to make sure the target application is not running before performing the operation. If Extension Manager finds that the target application is running, it prompts the user to quit the application first. For many applications, the user must manually quit them. For Dreamweaver CS5, the user can click the button Exit Application to request it to quit. The default value is "false".

## Contents

This tag must contain a `description` tag, a `ui-access` tag, a `products` tag, and an `author` tag. If you're changing the menus, this tag must also contain a `configuration-changes` tag. If you're installing files, this tag must also contain a `files` tag.

## Container

None.

## Example

```
<macromedia-extension
  name = "FrobSquigger Command"
  version = "1.0.0"
  type = "command"
  requires-restart = "false" >
  mxversion = "5.0"
  xmanversion = "5.0"
  icon = "command.png"
  <!-- description, ui-access, products, author, configuration-changes, and
  files tags go here -->
</macromedia-extension>
```

*Note:* The `macromedia-extension` tag must be located at line 1 of your file.

## defaultLanguage

### Description

This tag specifies the default language for installed files. Extension Manager determines the correct language by completing these steps, listed in order of priority:

- 1 The language of the point product (defined in `XManConfig.xml` file for point product).
- 2 The language of the Extension Manager interface.
- 3 The language selected by the user when prompted by Extension Manager.
- 4 If the extension doesn't provide files for the user-selected language, Extension Manager installs files specified by the "defaultLanguage" tag.

If Extension Manager can't find any files belonging to the above languages, it installs files for all languages.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\)](#) on page 54.

Possible values for the `defaultLanguage` tag are listed below:

Language	Value
American English	en_US
British English	en_GB
Danish	da_DK
Dutch	nl_NL
Finish	fi_FI
French	fr_FR

Language	Value
German	de_DE
Italian	it_IT
Norwegian	nb_NO
Portugese	pt_BR
Spanish	es_ES
Catalan	ca_ES
Swedish	sv_SE
Ukrainian	uk_UK
Chinese	zh_CN
Taiwanese	zh_TW
Japanese	ja_JP
Korean	ko_KR

#### Example

```
<defaultLanguage>fr_FR</defaultLanguage>
```

## description

### Description

Describes what the extension does or is used for.

### Attributes

{href}, {resid}, {source}

**href** Attribute for version 5.0 that degrades gracefully. Indicates online URL that will be displayed as the description of the extension. The value must start with either “http://” or “https://”.

**resid** References string with value of “resid” in resource file. When users select an extension in Extension Manager, this string is displayed in the lower-right area of the user interface. For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

**source** Attribute for version 5.0 that does not degrade gracefully. Indicates the relative path to the HTML file on the local computer, specified by the `$ExtensionSpecificEMStore` attribute. For more information, see the [destination attribute for file on page 19](#).

*Note:* If href is specified and computer is online, Extension Manger displays the page pointed to by the URL in the description field. If source is specified, the specified local page is displayed for the description. If neither href nor source are specified, text in the Contents tag below is displayed.

### Contents

This tag must contain a CDATA section, which you can format with any HTML tags. If text colors are unspecified, the background is gray (#626262), and the text is black.

## Container

This tag must be contained in a `macromedia-extension` tag.

## Example

```
<description><![CDATA[This command converts a frob into a squig.<br>Be sure not to use it on a grickle]]></description>
```

## update

### Description

This tag allows third-party developers to include an update link with the extension. When Extension Manager starts, it checks for an extension update. If an update available, Extension Manager notifies the user to update the extension.

### Attributes

`url`, {method}

`url` Indicates online link for extension update information file. The value must start with either “`http://`” or “`https://`”. For more information, see [Creating updatable extension packages \(version 5.0 only\) on page 59](#)

`method` Reserved for future use. Now it has to be ‘directlink’.

### Contents

None.

### Container

This tag must be contained in a `macromedia-extension` tag.

## Example

```
<update method="directlink" url="http://www.foobar.com/update.xml" />
```

## license-agreement

### Description

This tag lets third-party developers include a license agreement with extensions they develop. The contents of this tag are displayed under the heading Third Party License, at the end of the new extension install license.

If the `license-agreement` tag is omitted from the MXI file, only the default extension disclaimer is displayed.

### Attributes

{resid}

`resid` References string with value of “resid” in resource file and displays that string after the License Agreement when installing the extension. For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

This tag must contain a CDATA section, which you can format with any HTML tags. If text colors are unspecified, the background is gray (#585858) and the text is nearly white (#E0E0E0).



#### Attributes

None.

#### Contents

This tag must contain one or more `product` tags.

#### Container

This tag must be contained in a `macromedia-extension` tag.

#### Example

```
<products>  
  <!-- product tags go here -->  
</products>
```

## product

#### Description

Specifies which Adobe application or applications your extension is compatible with. List each application in a separate `product` tag.

#### Attributes

name, {version}, {primary}, {required}, {maxversion}, {familyname}, {platform}, {bit}

**name** The name of an Adobe application. This attribute uses a `VARCHAR2` data type with a limit of 64 characters. Valid values appear below:

- Bridge
- Contribute
- Dreamweaver
- Fireworks
- Flash
- Illustrator
- InCopy
- InDesign
- Photoshop32 (32-bit Photoshop)
- Photoshop64 (64-bit Photoshop)
- Premiere

*Note:* To specify Photoshop for Mac OS and Windows, see the “familyname” attribute below. When that attribute is specified, the “name” attribute isn’t required.

**version** The version number of the specified Adobe application. Valid version numbers:

<b>Product</b>	<b>Version number</b>
Bridge CS4	3
Bridge CS5	4
Contribute CS4	5
Contribute CS5	6

<b>Product</b>	<b>Version number</b>
Dreamweaver MX 2004	7
Dreamweaver 8	8
Dreamweaver CS3	9
Dreamweaver CS4	10
Dreamweaver CS5	11
Fireworks MX 2004	7
Fireworks 8	8
Fireworks CS3	9
Fireworks CS4	10
Fireworks CS5	11
Flash MX 2004	7
Flash 8	8
Flash CS3	9
Flash CS4	10
Flash CS5	11
Illustrator CS4	14
Illustrator CS5	15
InCopy CS4	6
InCopy CS5	7
InDesign CS4	6
InDesign CS5	7
Photoshop CS4	11
Photoshop CS5	12
Premiere Pro CS5	5

For example, if your extension is for Dreamweaver CS5, specify `version = "11"`. The extension can be installed in any version of the product greater than or equal to the specified version number. This attribute uses a `VARCHAR` data type with a limit of 8 characters. Note Extension Manager CS5 supports CS5 products only. To install extension for CS4 products, Extension Manager CS4 is necessary.

`primary` Indicates whether the specified Adobe product is the one the extension was primarily intended to be used with. For example, if the extension's user interface appears in Dreamweaver but the extension also uses Fireworks, Dreamweaver is the primary product. For example, `<product name = "Dreamweaver" version = "11" primary = "true" />` indicates that this extension is primarily intended for Dreamweaver; however, it might be used in another product that supports the Extension Manager. (If you set the primary attribute to "true" for multiple products, Extension Manager can install the extension into each product.)

**required** Indicates whether the specified Adobe product is required for the extension to function properly. If the extension will function without the indicated product, even if it won't function as well without it, specify "false" or omit the `required` attribute. If you don't specify `required = "true"` for any product tag, the product specified in the first product tag listed is assumed to be required.

For example, `<product name = "Dreamweaver" version = "11" required = "true" />` indicates that Dreamweaver is necessary to use this extension.

**maxversion** Specifies the latest product version that an extension can be installed in. For example, if a Dreamweaver extension can be installed with CS5 only, you would specify `<product name="Dreamweaver" version="11" maxversion="11"/>`

**familyname** Combines Photoshop products together. For example, to combine the standard and Extended versions, specify `<product version="11" familyname="Photoshop" primary="true" />`

**platform** Indicates on which platform the extension can be installed. Valid values are "mac" or "win". If set to "mac", the extension can only be installed for the product on Mac OS. If set to "win", the extension can only be installed for the product on Windows. If not set, the extension can be installed for the product on both platforms.

**bit** Indicates whether the extension's target product is a 32-bit product or 64-bit product. Valid values are "32" or "64". If set to "32", the extension can only be installed for the 32-bit product. If set to "64", the extension can only be installed for the 64-bit application. If not set, the extension can be installed for both the 32-bit and the 64-bit versions of the application.

#### Contents

None.

#### Container

This tag must be contained in a `products` tag.

#### Example

```
<product name = "Dreamweaver" version = "11" primary = "true" />
```

## author

#### Description

Name of the author of the extension.

#### Attributes

`name`, {`author_resid`}

`name` The author's name. This attribute uses a `VARCHAR` data type with a limit of 255 characters.

`author_resid` References string resource with value of 'author\_resid' in resource file and displays that string in "Author" field of the user interface. For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

None.

## Container

This tag must be contained in a `macromedia-extension` tag.

## Example

```
<author name = "Jambalaya Joe"/>
```

## files

### Description

Container tag for all file tags.

### Attributes

```
{xml:lang}, {default-file-type}
```

`xml:lang` Specifies the language for the group of files. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, the files are installed; if not, the files are ignored. If Extension Manager can't determine the user language, it copies all files regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

`default-file-type` Specifies the type of files wrapped in the `files` tag. Valid values are "csxs", "plugin" and "ordinary". The value "csxs" flags the file as a CS extension package. For more information, refer to [Creating Hybrid Extension packages \(version 5.0 only\) on page 57](#). The value "plugin" flags the file as a plug-in. For more information, refer to [Creating plug-in extension packages for InDesign CS5 and InCopy CS5 \(version 5.0 only\) on page 58](#). The default value is "ordinary". If you specify "ordinary" the files will be packaged up without any special processing into an ordinary extension. Use the "ordinary" flag for all Extensions for CS4 or earlier releases.

### Contents

This tag must contain one or more file tags.

## Container

This tag must be contained in a `macromedia-extension` tag.

## Example

```
<files>  
  <!-- file tags go here -->  
</files>
```

## file

### Description

Provides information about a specific file to be installed as part of the extension.

*Note:* Use `menu-insert` tags to explicitly add your item to menus even if your extension is an object or a command; don't rely on the Adobe application to automatically add objects and commands to its menus. See [menu-insert](#) for details.

## Attributes

source, destination, {platform}, {shared}, {systemfile}, {win-extension}, {isresourcefile}, {file-type}, {minVersion}, {maxVersion}

**source** The name of the file. It can include a path relative to the location of the installation file; the extension's files don't all have to be in the same folder. The filename must be a valid name in both Windows and Mac OS, unless you specify a value for the `platform` attribute. You can use a colon (:), slash (/), or backslash (\) as the separator between folder names (and before the filename) in the path. Note that in some operating systems, filenames are case-sensitive; make sure to use the same capitalization in the `source` attribute as you use for the corresponding file and folder names on your disk. Filenames should be a total of 30 characters or less.

Do not use the same filenames as Adobe extensions unless your extension is intended as a substitute for an Adobe extension.

To create an extension as part of a bundle or framework on Mac OS, use either of the following formats, without wildcards:

- `<files><file source="sourceFolder" destination="$photoshop/" /></files>`
- `<files><file source="sourceFolder/" destination="$photoshop/" /></files>`

**destination** The name of the folder the file will be copied to. If the folder doesn't exist, the Extension Manager creates it during installation. Note that this attribute should contain a folder name, not a filename. The filename is specified by the `source` attribute.

Attributes you can use to refer to installation folders include the following:

Various applications and system folders

Attribute	Description
<code>\$dreamweaver</code>	Specifies the Dreamweaver installation folder
<code>\$dreamweaver/Configuration</code>	Specifies the Dreamweaver configuration folder under the user home folder
<code>\$fireworks</code>	Specifies the Fireworks installation folder
<code>\$flash</code>	Specifies the Flash installation folder
<code>\$System</code>	Specifies the System or System32 folder
<code>\$Fonts</code>	Specifies the Font folder on the computer's hard disk
<code>\$ExtensionSpecificEM Store</code>	Attribute for version 5.0 that degrades gracefully. Specifies the folder that stores extension-specific file.

## Photoshop

<b>Attribute</b>	<b>Description</b>
\$photoshoppfolder	Specifies the installation folder
\$pluginsfolder	Specifies the top level of the Plug-ins folder
\$presetsfolder	Specifies the top level of the Presets folder
\$scripts	Specifies the Scripts folder
\$actions	Specifies the Actions folder
\$brushes	Specifies the Brushes folder
\$matlab	Specifies the MATLAB folder

## Adobe Bridge

<b>Attribute</b>	<b>Description</b>
\$bridgeappfolder	Specifies the Bridge installation folder
\$pluginsfolder	Specifies the Bridge Plug-ins folder
\$presetsfolder	Specifies the Bridge Presets folder
\$bridge	Specifies the Bridge ExtensionManager Config folder
\$startupscripts	Specifies the global (suite) startup scripts folder
\$bridgestartupscripts	Specifies the Bridge global startup scripts folder
\$extensions	Specifies the Bridge namespace extensions folder
\$workspaces	Specifies the Bridge user workspaces folder
\$extensionworkspaces	Specifies the Bridge namespace extensions workspaces folder
\$userscripts	Specifies the Bridge user startup scripts folder

## Illustrator

<b>Attribute</b>	<b>Description</b>
\$illustrator	Specifies the installation folder
\$plugin	Specifies the Plug-ins folder
\$scripting	Specifies the Scripting folder
\$presets	Specifies the Presets folder

## InDesign

<b>Attribute</b>	<b>Description</b>
\$indesign	Specifies the installation folder
\$indesign_user	Specifies the per-user folder under the user home folder

## InCopy

Attribute	Description
\$incopy	Specifies the installation folder
\$incopy_user	Specifies the per-user folder under the user home folder

## Contribute

Attribute	Description
\$contribute	Specifies the installation folder
\$contribute_user	Specifies the per-user folder inside the user home directory

The Extension Manager picks the appropriate system and font folder on the user's disk, based on the user's platform and operating system. If none of these options suits your needs, you can define your own custom tokens for the destination of your files. For information about writing custom tokens, see [token on page 24](#).

Generally, destination folders should be inside the application's Configuration folder. The `destination` attribute is not case-sensitive; `configuration` is the same as `Configuration`. The folder name must be a valid name on both Windows and Mac OS, unless you specify a value for the `platform` attribute. You can use a colon (:), slash (/), or backslash (\) as the separator between folder names in the path. Note that in some operating systems, folder names are case-sensitive; make sure to use the correct capitalization for your folder names.

If your extension for Dreamweaver contains multiple files, such as help files, many images, or a suite of items, the destination folder for your files should be a folder in the Configuration/Shared folder. The folder name should be related to your company or product—for example, Configuration/Shared/MagicTricks.

You do not need to include the Configuration folder in the path name for Flash extensions. The folder specified in the `destination` tag is automatically created in the Configuration folder if it does not already exist.

`platform` Indicates what platform the file is intended for. If you specify a platform, the file is installed only on that platform; for instance, you can provide two versions of a file, one for Windows and one for Mac OS, and specify a platform value for each. Valid values are "win" and "mac". If you don't specify this attribute, the file is installed on both platforms.

`shared` Indicates whether the file is used by more than one extension. When you use the Extension Manager to remove an extension, a shared file associated with that extension is not deleted as long as other installed extensions refer to that file. Valid values are "true" and "false". If you don't specify this attribute, its default value is "false".

*Note:* If you install a newer version of a shared file and another extension is using the old version of the file, the new shared file either must be backward compatible with the other extension, or must have a new filename so that the other extension continues to work properly.

`systemfile` Indicates whether the file is used by anything other than extensions. For example, some extensions provide new versions of DLLs or other system files, or files that are used by other applications. If a file is specified as `systemfile = "true"`, it is not deleted when the extension is removed, even if no other extensions use the file. When `systemfile` is set to true, the `shared` attribute is ignored.

`win-extension` Used when a file is generated on Mac OS that does not include the Windows extension, such as `.fla` or `.htm`.

For example, a FLY file named “shoo” created on Mac OS installs with the correct creator and file-type information on another Mac OS. However, in a Windows system, it requires the following to install properly in the Configuration\Shared\Thingies folder:

```
<file source="shoo" destination="$Dreamweaver/configuration/shared/thingies/"
win-extension="fly" />.
```

This adds the extension to the filename and installs “shoo.fly”.

If you create a file on Windows that does include the extension, such as “heeble.fla” or “frob.htm”, and install it on Mac OS, the `win-extension` attribute does not need to be added to the file tag.

*Note:* If the `platform` attribute is included, the `win-extension` attribute is ignored.

`isresourcefile` If set to `true`, this attribute flags the file as a resource file containing language-specific text strings. Place resource files in a folder with the name [*installer prefix*].mxi\_Resources. When the MXI file is loaded, Extension Manager copies this folder into following location, where it then looks for text strings:

- Win XP: C:\Documents and Settings\All Users\Application Data\Adobe\Extension Manager CS5
- Vista/Win7: C:\ProgramData\Adobe\Extension Manager CS5
- Mac OS: /Library/Application Support/Adobe/Extension Manager CS5

If this attribute is not specified, it is considered false.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

`file-type` Specifies the file type. Valid values are `"csxs"`, `"plugin"` and `"ordinary"`. Refer to attribute `default-file-type` in `files` tag.

If this attribute is not specified, any file wrapped in `files` tag use the value of the attribute `default-file-type` in `files` tag. If attribute `default-file-type` is not specified either, its default value is `"ordinary"`. The value specified by this attribute has more priority than value specified by the attribute `default-file-type` in wrapping `files` tag.

`minVersion`, `maxVersion` Specifies the minimum and maximum version of the product to which the file will be installed. For example, if `minVersion` is 9 and `maxVersion` is 10, the file won't be installed in product version 8 or 11, but will be in product version 9. The correct format for this attribute is `Major_Version_Number.Minor_Version_Number.Build_Number`.

## Contents

None.

## Container

This tag must be contained in a `files` tag.

## Example

```
<file source = "frob2squig.htm" destination = "$Dreamweaver/configuration/
commands/common" platform = "mac" shared = "false" />
```

## file-tokens

### Description

Container tag that indicates any custom tokens.

### Attributes

None.

### Contents

One or more token tags for defining custom tokens.

### Container

This tag must be contained in a `macromedia-extension` tag.

### Example

```
<file-tokens>
  <!-- token tags go here -->
</file-tokens>
```

## token

### Description

Defines a custom token for an extension.

Custom tokens let you specify the destination folder of one or more files from your extension during installation or provide a dialog box for the user to choose a destination folder for certain files. For example, you might use a custom token if your extension contains items that must be installed in a specific directory as well as a file, such as a tutorial, that can be installed anywhere on the hard disk. In this case, you could use a custom token tag to allow the user to select the destination folder for the tutorial while still installing the other files in the proper directories. If several files need to be grouped in the same directory, but that directory location is not important, you can allow the user to select the directory location.

Custom tokens are useful even if you don't allow the user to specify the destinations of files. You can easily change the destination directory of multiple files without having to manually change each destination path in the MXI file. In this case, you would use a custom token as you would use the `$Dreamweaver`, `$Fireworks`, `$Flash`, `$fonts`, or `$system` token. For example, if your extension contains multiple files that must be installed in `C:\program files\trailer`, you can use a token tag to define a custom token called `airstream`; all of the files that use this token are installed in `C:\program files\trailer`. If you want to change the destination folder of the files using the `$airstream` token, you have to make only one change in the token tag rather than change every instance of the path to the new destination in your MXI file.

*Note:* You cannot redefine the `$Dreamweaver`, `$Fireworks`, `$Flash`, `$fonts`, or `$system` token with a custom token.

### Attributes

name, {prompt}, {default}, {definition}

name The name of your custom token. This must be a unique name. Do not include the dollar sign (\$) in the name.

`prompt` Describes the kind of file to be installed in a folder. When you include this attribute, the user is prompted to specify a destination, and the value you provide is added to the dialog box's title. For example, if the attribute is `prompt="Sample Files"`, the dialog box displays "Select Folder for Sample Files".

`default` Defines the default folder path if the `prompt` attribute is used. If you do not define the `default` attribute, the path box is blank. You can use a token in this attribute, such as `default="$Dreamweaver"`.

`definition` Defines the file path of the token when you do not use the `prompt` attribute. This prevents the Select Folder dialog box from appearing, so that the user cannot choose a destination path. In the example below, all files using the token `$airstream` are installed in `C:\program files\trailer.`:

```
<token name="airstream" definition="C:\program files\trailer" />
```

*Note:* If you use the `prompt` attribute, do not use the `definition` attribute.

#### Contents

None.

#### Container

This tag must be contained in a `file-token` tag.

#### Example

This example is for Windows platforms, which use backslashes (\) to delimit directories:

```
<token name="airstream" prompt="Sample File"
  default="$Dreamweaver\Configuration\Shared\trailer" />
```

This example is for Mac OS, which uses colons (:) to delimit directories:

```
<token name="airstream" prompt="Sample File"
  default="$Dreamweaver:Configuration:Shared:trailer" />
```

## configuration-changes

#### Description

Container tag for changes to menus, shortcuts, server behaviors, server formats, and data sources. Extensions for Dreamweaver can specify several product-specific changes—such as where in the user interface they are installed—using tags within the `configuration-changes` tag.

#### Attributes

None.

#### Contents

This tag may contain any combination of `data-source-changes`, `menu-insert`, `menu-remove`, `server-behavior-changes`, `server-format-changes`, `server-format-definition-changes`, `shortcut-insert`, and `shortcut-remove` tags.

#### Container

This tag must be contained in a `macromedia-extension` tag.

### Example

```
<configuration-changes>
  <!-- ftp-extension-map-changes, menu-remove, menu-insert, shortcut-remove,
  shortcut-insert, server-behavior-changes, server-format-changes, server-
  format-definition-changes, and data-source-changes tags go here -->
</configuration-changes>
```

## documenttype-changes

### Description

Describes changes to be made to the MMDocumentTypes.xml file.

### Attributes

None.

### Contents

This tag contains the `documenttype-insert` and `documenttype-remove` tags.

### Container

This tag must be contained in a `configuration-changes` tag.

### Example

This example illustrates the syntax of the tags that can be contained by the `documenttype-changes` tag.

```
<documenttype-changes>
  <documenttype-insert>
    <documenttype>
      ...
    </documenttype>
  </documenttype-insert>
  <documenttype-remove id="remove_id" />
</documenttype-changes>
```

## documenttype-insert

### Description

Insert the specified tag library at the end of file.

### Attributes

{xml:lang}

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

## Contents

The `documenttype` tag describes the document type to be inserted. The Extension Manager verifies only that the XML structure is valid.

## Container

This tag must be contained in a `documenttype-changes` tag.

## Example

```
<documenttype-insert>
  <documenttype>
    ...
  </documenttype>
</documenttype-insert>
```

## documenttype-remove

### Description

Removes the specified document type.

### Attributes

`id`, `{xml:lang}`

`id` ID of the document type to remove.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

None.

### Container

This tag must be contained in a `documenttype-changes` tag.

### Example

```
<documenttype-remove id="remove_id" />
```

## toolpanel-changes

### Description

Marks the beginning of Flash tool panel changes.

### Attributes

None.

### Contents

This tag may contain the `toolpanel-item-insert` tag.

## Container

This tag must be contained in a `configuration-changes` tag.

## Example

```
<configuration-changes>
  <toolbar-changes>
    ...
  </toolbar-changes>
</configuration-changes>
```

## toolbar-item-insert

### Description

Inserts the tool with the specified name into the Flash tool panel.

### Attributes

`name`, `position`, `depth`, `{xml:lang}`

`name` The name of the tool to insert. This is a required attribute.

`position` The 0-based position at which to insert the tool. Valid positions are 0 through 17. If the attribute is missing, or has a value beyond 17, the tool assumes the last position in the toolbar by default. This is an optional attribute.

`depth` The 0-based depth of the tool is its position in the toolbar, where 0 specifies the top. If the `depth` attribute is missing, or has a value beyond the bottom of the menu as the tool's position, the tool depth defaults to the bottom of the menu. This is an optional attribute.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage](#) on page 12. If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\)](#) on page 54.

### Contents

None.

### Container

This tag must be contained in a `toolbar-changes` tag.

### Example

```
<toolbar-changes>
  <toolbar-item-insert name="polystar" position="7" />
</toolbar-changes>
```

## ftp-extension-map-changes

### Description

Specifies a change to the `FTPExtensionMap.txt` file located in the Configuration folder.

#### Attributes

None.

#### Contents

This tag may contain an `ftp-extension-insert` tag and an `ftp-extension-remove` tag.

#### Container

This tag must be contained in a `configuration-changes` tag.

#### Example

```
<ftp-extension-map-changes>  
  <!-- ftp-extension-insert, ftp-extension-remove tags go here-->  
</ftp-extension-map-changes>
```

### ftp-extension-insert

#### Description

Specifies a change to the `FTPExtensionMap.txt` file. This defines whether the file is downloaded or uploaded as an ASCII or binary file from Dreamweaver to an FTP server.

#### Attributes

`extension`, `type`, `mac-creator`, `mac-file-type`

`extension` The file extension, such as `.gif` or `.jpg`.

`type` The format used when you upload a file to the FTP server. The current valid values are "ASCII" and "Binary".

`mac-creator` The Mac OS creator code. If you do not know the creator code, use "????".

`mac-file-type` The Mac OS file type. If you do not know the file type, use "????".

#### Contents

None.

#### Container

This tag must be contained in an `ftp-extension-map-changes` tag.

#### Example

```
<ftp-extension-insert extension="JPG" mac-creator="MKBY" mac-file-type="JPEG"  
  type="ASCII"/>
```

### ftp-extension-remove

#### Description

Indicates the extension that is removed from `SourceFormat.txt` in the Configuration folder.

#### Attributes

`extension`

`extension` The file extension, such as `.gif` or `.jpg`.

## Contents

None.

## Container

This tag must be contained in an `ftp-extension-map-changes` tag.

## Example

```
<ftp-extension-remove extension="JPG" />
```

## insertbar-changes

### Description

Marks the beginning of changes to `Insertbar.xml`. Note that `InsertBar.xml` is automatically updated when objects are installed into Dreamweaver MX. Modifying the file explicitly from the MXI file is not required.

### Attributes

None.

### Contents

The `insertbar-insert`, `insertbar-item-insert`, and `category` tags describe the category to be inserted. The Extension Manager verifies only that the XML structure is valid.

### Container

This tag must be contained in a `configuration-changes` tag.

### Example

```
<insertbar-changes>
  <insertbar-insert insertBefore|insertAfter="category_id">
    <category ...>
      <itemtype.../>
    </category>
  </insertbar-insert>
  <insertbar-remove id="category_id" />
  <insertbar-item-insert
    insertBefore|insertAfter|appendTo|prependTo="category_or_item_id"
    category="category_id"> <itemtype.../>
  </insertbar-item-insert>
  <insertbaritem-remove id="item_id" />
</insertbar-changes>
```

## insertbar-insert

### Description

Inserts the specified category at the end of file.

### Attributes

`insertBefore` | `insertAfter`, {xml:lang}

`insertBefore` | `insertAfter` = `category_id` of the existing category to insert before or after. You can specify only one of the two attributes; either `insertBefore` or `insertAfter`.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

The `category` tag that describes the category to insert. The Extension Manager verifies only that the XML structure is valid.

#### Container

This tag must be contained in an `insertbar-changes` tag.

#### Example

```
<insertbar-insert insertBefore | insertAfter = category_id>
  <category ...>
    <itemtype...>
  </category>
</insertbar-insert>
```

## insertbar-remove

#### Description

Removes the specified category.

#### Attributes

`category_id`, {`xml:lang`}

`category_id` ID of the category to be removed.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

None.

#### Container

This tag must be contained in the `insertbar-changes` tag.

#### Example

```
<insertbar-remove id="category_id" />
```

## insertbar-item-insert

### Description

Inserts the specified item at the specified location.

### Attributes

`insertBefore` | `insertAfter`, `appendTo` | `prependTo`, `category`, `{xml:lang}`

`insertBefore` | `insertAfter` ID of the existing item before or after which the specified item should be inserted.

`appendTo` | `prependTo` ID of the existing category to which the specified item is appended or prepended.

`category` ID of the category to append to if the `insertBefore|insertAfter` item isn't found.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

A tag that describes the item to insert. The Extension Manager verifies only that the XML structure is valid.

### Container

This tag must be contained in the `insertbar-changes` tag.

### Example

```
<insertbar-item-insert
  insertBefore | insertAfter | appendTo | prependTo = category_or_item_id
  category = category_id
  <itemtype...>
</insertbar-item-insert>
```

## insertbar-item-remove

### Description

Removes the specified `insertbar` item.

### Attributes

`id`, `{xml:lang}`

`id` ID of the item to be removed.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

None.

#### Container

This tag must be contained in an `insertbar-changes` tag.

#### Example

```
<insertbar-item-remove id = item_id />
```

## server-behavior-changes

#### Description

Container tag for changes to menus in the `ServerBehaviors.xml` file in any of the Dreamweaver MX Configuration/ServerBehaviors/*servermodel* folders.

#### Attributes

`servermodelfolder`

`servermodelfolder` The name of the server model folder in which the changes are to be made. The name of any installed server model (such as "ASP.NET\_Csharp", "ASP.NET\_VB", "ASP\_Js", "ASP\_Vbs", "ColdFusion", "UD4-ColdFusion", "PHP\_MySQL" or "JSP") is a valid value. Note that the attribute value must precisely match the name of the corresponding server model folder.

#### Contents

This tag may contain any combination of `menu-remove` and `menu-insert` tags.

#### Container

This tag must be contained in a `configuration-changes` tag.

#### Example

```
<server-behavior-changes servermodelfolder = "ASP_VB">  
  <!-- menu-remove, menu-insert tags go here -->  
</server-behavior-changes>
```

## server-format-changes

#### Description

Container tag for changes to menus in the `Formats.xml` file in any of the Dreamweaver MX Configuration/ServerFormats/*servermodel* folders.

### Attributes

servermodelfolder

servermodelfolder The name of the server model folder in which the changes are to be made. The name of any installed server model (such as "ASP.NET\_Csharp", "ASP.NET\_VB", "ASP\_Js", "ASP\_Vbs", "ColdFusion", "UD4-ColdFusion", "PHP\_MySQL" or "JSP") is a valid value. Note that the attribute value must precisely match the name of the corresponding server model folder.

### Contents

This tag may contain any combination of menu-remove and menu-insert tags.

### Container

This tag must be contained in a configuration-changes tag.

### Example

```
<server-format-changes servermodelfolder = "ASP_VB">  
  <!-- menu-remove, menu-insert tags go here -->  
</server-format-changes>
```

## server-format-definition-changes

### Description

Container tag for changes to menus in the ServerFormats.xml file in any of the Dreamweaver MX Configuration/ServerFormats/*servermodel* folders.

### Attributes

servermodelfolder

servermodelfolder The name of the server model folder in which the changes are to be made. The name of any installed server model (such as "ASP.NET\_Csharp", "ASP.NET\_VB", "ASP\_Js", "ASP\_Vbs", "ColdFusion", "UD4-ColdFusion", "PHP\_MySQL" or "JSP") is a valid value. Note that the attribute value must precisely match the name of the corresponding server model folder.

### Contents

This tag may contain any combination of menu-remove and menu-insert tags.

### Container

This tag must be contained in a configuration-changes tag.

### Example

```
<server-format-definition-changes servermodelfolder = "ColdFusion">  
  <!-- menu-remove, menu-insert tags go here -->  
</server-format-definition-changes>
```

## data-source-changes

### Description

Container tag for changes to menus in the DataSources.xml file in any of the Dreamweaver MX Configuration/DataSources/*servermodel* folders.

## Attributes

`servermodel`

`servermodel folder` The name of the server model folder in which the changes are to be made. The name of any installed server model (such as "ASP.NET\_Csharp", "ASP.NET\_VB", "ASP\_Js", "ASP\_Vbs", "ColdFusion", "UD4-ColdFusion", "PHP\_MySQL" or "JSP") is a valid value. Note that the attribute value must precisely match the name of the corresponding server model folder.

## Contents

This tag may contain any combination of `menu-remove` and `menu-insert` tags.

## Container

This tag must be contained in a `configuration-changes` tag.

## Example

```
<data-source-changes servermodel = "ASP_VB">
  <!-- menu-remove, menu-insert tags go here -->
</data-source-changes>
```

## menu-insert

### Description

Specifies where in the application's menus to insert a menu bar, menu, menu item, or format during installation of this extension.

Use `menu-insert` tags to explicitly add your extension to menus even if your extension is an object or a command; don't rely on the Adobe application to automatically add objects and commands to its menus. To ensure that your extension is not automatically added to the menus, add `<!-- MENU-LOCATION=NONE -->` to the top of each of your extension's HTML files. If you do this, you must make an entry for your file in the `menus.xml` file.

### Attributes

`insertAfter`, `insertBefore`, `appendTo`, `prependTo`, `{skipSeparator}`, `{xml:lang}`

You can specify only one of the following four attributes: `insertAfter`, `insertBefore`, `appendTo`, or `prependTo`.

`insertAfter` Indicates that the new item should be inserted immediately following the item with the specified ID. (The ID can be the ID of a menu bar, a menu, a menu item, or a format.)

*Note:* No menu can appear to the right of the Help menu in Dreamweaver. If you insert a new menu after the Help menu, the application displays the new menu to the left of the Help menu.

`insertBefore` Indicates that the new item should be inserted immediately before the item with the specified ID. (The ID can be the ID of a menu bar, a menu, a menu item, or a format.)

`appendTo` Indicates that the new item should be inserted immediately after the last item in the specified menu or menu bar. (The specified ID can be the ID of a menu bar or a menu only, not a menu item or format.)

`prependTo` Indicates that the new item should be inserted before the first item in the specified menu or menu bar. (The specified ID can be the ID of a menu bar or a menu only, not a menu item or format.)

`skipSeparator` Applies only if the `insertAfter` attribute is also specified. It indicates that the new item should be inserted after the separator that immediately follows the item specified in `insertAfter`. If there is no separator there, or if `insertAfter` is not used, this attribute is ignored. Valid values are "true" and "false"; the default value is "false".

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage](#) on page 12. If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\)](#) on page 54.

## Contents

In most uses, this tag must contain one or more `menu` tags and/or one or more `menuitem` tags. (When this tag appears inside a `server-format-definition-changes` tag, it contains `format` tags instead of `menuitem` tags.) It may also optionally contain `separator` tags and `comment` tags. Everything inside the `menu-insert` tag is inserted as a block, retaining its order; for example, if you list four menu items inside a `menu-insert` tag, those four items are inserted at the specified location so that they appear in the menu structure in the same order.

You can insert as many menus, menu items (or formats), separators, and comments as you want in a single `menu-insert` tag, but they can't be nested. That is, you can't insert a new menu and its contents by listing the items inside the `menu` tag. Instead, insert the menu first with one `menu-insert` tag, then insert all of the items into the new menu using another `menu-insert` tag. In the example below, the menu item `Animals` is inserted in the `Insert` menu after the `Get More Objects` menu item. The `Animals` menu item contains the submenus `Cat` and `Dog`. The `Dog` menu contains yet another submenu called `Poodle`. This is the resulting menu structure:

Insert

```
...
Get More Objects
Animals
  Dog
  Poodle
  Cat
```

```
<menu-insert insertAfter="DWMenu_Insert_GetMoreObjects">
  <menu name="Animals" id="DWMenu_Insert_Animals" />
</menu-insert>
<menu-insert appendTo="DWMenu_Insert_Animals">
  <menu name="Dog" id="DWMenu_Insert_Animals_Dog" />
  <menuitem name="Cat" id="DWMenu_Insert_Animals_Cat" />
</menu-insert>
<menu-insert appendTo="DWMenu_Insert_Animals_Dog">
  <menuitem name="Poodle" id="DWMenu_Insert_Animals_Dog_Poodle" />
</menu-insert>
```

## Container

This tag must be contained in a `configuration-changes`, `server-behavior-changes`, `server-format-changes`, `server-format-definition-changes`, or `data-source-changes` tag.

### Example

```
<menu-insert insertAfter = "DWMenu_Commands_SortTable" skipSeparator = "true">
  <!-- menu, menuitem (or format), separator, and comment tags here -->
</menu-insert>
```

## menu-remove

### Description

Provides information about a menu bar, menu, menu item, or format to remove during installation of this extension.

*Note:* If the user removes an installed extension, the menus, menu items, and formats that were removed when that extension was installed are not restored.

### Attributes

`id`, `{xml:lang}`

`id` The menu ID of the item to be removed. Menu bars and menus are not removed unless they're empty. To find the menu ID of an item in a Dreamweaver menu, look for the item in the `menus.xml` files. (To find the menu ID of a format, look for the item in the `Formats.xml` files.)

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

None.

### Container

This tag must be contained in a `configuration-changes`, `server-behavior-changes`, `server-format-changes`, `server-format-definition-changes`, or `data-source-changes` tag.

### Example

```
<menu-remove id = "DWMenu_Commands_FrobSquigger-beta" />
```

## menubar

### Description

Provides information about a menu bar to be inserted into the application's menu structure during installation of this extension.

### Attributes

`name`, `id`, `{platform}`

`name` The name of the menu bar to insert.

`id` ID for the new menu bar. Your menu IDs should start with a company name or some other namespace prefix to ensure uniqueness. In particular, don't start your menu IDs with `DW`, which is the prefix used by Dreamweaver menu IDs.

`platform` Indicates that the menu bar should appear only on the given platform. Valid values are "win" and "mac".

#### Contents

None.

#### Container

This tag must be contained in a `menu-insert` tag.

#### Example

```
<menubar name = "Mugwump Context menu" id = "JMMugwumpContext"
platform = "mac"></menubar>
```

## menu

#### Description

Describes a menu or submenu to be inserted into the application's menu structure during installation of an extension.

#### Attributes

`name`, `id`, {`platform`}

`name` The name of the menu to insert, as it will appear in the menu bar. To set the menu's access key (mnemonic) on Windows, use an underscore (`_`) in front of the access letter. The underscore is automatically removed on Mac OS.

`id` The menu ID of the new menu. Your menu IDs should start with a company name or some other namespace prefix to ensure uniqueness. In particular, don't start your menu IDs with `DW`, which is the prefix used by the Dreamweaver menu IDs. One useful approach is to prefix every ID with your domain name (with the elements reversed); for example, if your domain name is `joe.com`, you could start every ID with `com.joe.` to ensure uniqueness.

`platform` Indicates that the menu should appear only on the given platform. Valid values are "win" and "mac".

#### Contents

None.

*Note:* Always use a `</menu>` tag to close a `<menu>` tag. Although the `<menu>` tag in the MXI file has no contents, it corresponds to the `<menu>` tag in the `menus.xml` file, which does have contents. Therefore, the `<menu>` tag is not defined as an empty tag, so you can't use the `</>` XML syntax to close the tag.

#### Container

This tag must be contained in a `menu-insert` tag.

#### Example

```
<menu name = "Recent Frobs Converted" id = "JMMenu_Commands_RecentFrobs"
platform = "mac" >
</menu>
```

## menuitem

### Description

Describes the menu item to be inserted into the application's menu structure during installation of this extension.

### Attributes

name, id, {key}, {platform}, {file}, {command}, {enabled}, {checked}, {dynamic}, {arguments}, {resid:name}

**name** The menu item name that you want to appear in the menu. To set the menu item's access key on Windows, use an underscore (`_`) in front of the access letter. The underscore is automatically removed on Mac OS. If two menu items have the same access key, the access key works only for the first of the two.

*Note:* To make an underscore character appear in a menu item, precede it with a percent sign—that is, use `%_` instead of just an underscore.

**id** The menu ID of the new item. Your menu IDs should start with a company name or some other namespace prefix to ensure uniqueness. In particular, don't start your menu IDs with `DW`, which is the prefix used by the Dreamweaver menu IDs. One useful approach is to prefix every ID with your domain name (with the elements reversed); for example, if your domain name is `joe.com`, you could start every ID with `com.joe.` to ensure uniqueness.

**key** The shortcut key for the menu item. For syntax details that apply to Dreamweaver, see "About customizing Dreamweaver menus" in the "Customizing Dreamweaver" chapter of *Using Dreamweaver*.

**platform** Indicates that the menu should appear only on the given platform. Valid values are "win" and "mac".

**file** The name of an HTML or JavaScript file that contains JavaScript code determining the behavior of the menu item. The path specified in the `file` attribute is relative to the Configuration folder. The `file` attribute overrides the `command`, `enabled`, and `checked` attributes. Either `file` or `command` must be specified for each menu item. Note that in some operating systems, filenames are case-sensitive; make sure to use the same capitalization in the `file` attribute as you use for the corresponding file and folder names on your hard disk.

**command** JavaScript code specifying the action to be taken when the user chooses the menu item.

**enabled** JavaScript code that the application executes before displaying the menu, to determine whether the menu item is enabled. The code should return a value of `true` or `false`, indicating that the menu item should be enabled or dimmed, respectively.

**checked** JavaScript code that the application executes before displaying the menu, to determine whether the menu item should have a check mark next to it. The code should return a value of `true` or `false`, indicating that the menu item should be checked or unchecked, respectively.

**dynamic** Indicates whether the menu item's text and state are to be determined dynamically, by an HTML file that contains JavaScript code (specified in the `file` attribute). Valid values are "true" and "false". If you don't specify the `dynamic` attribute, its default value is "false".

**arguments** Provides arguments to pass to the specified command file. This attribute is used only in conjunction with the `file` attribute.

`resid:name` References string with value of `resid:name` in resource file, and changes the Dreamweaver configuration file to display that string as a menu item. If Extension Manager can't find the string in the resource file, the value specified in the `name` attribute is displayed as the menu item.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

None.

#### Container

This tag must be contained in a `menu-insert` tag that is not inside a `server-format-definition-changes` tag.

#### Example

```
<menuitem name = "Convert Frobs to Squigs" id = "JMMenu_Commands_ConvertFrobs"
key = "Cmd+Alt+Shift+F" platform = "mac" file = "commands/common/
  frob2squig.htm"
dynamic = "false" />
```

## format

#### Description

Describes the data format to be inserted into the Dreamweaver Format menu during installation of this extension.

#### Attributes

This tag's attributes are difficult to write by hand. The best way to create a `format` tag is to use the interface inside Dreamweaver. After you create a format, open the appropriate `Formats.xml` file in a text editor and copy the appropriate `format` tag (generated by Dreamweaver). Paste this tag into the appropriate place in your extension installation file.

Then add an `id` attribute. Each format ID must be unique; your IDs should start with a company name or some other namespace prefix to ensure uniqueness. In particular, don't start your IDs with `DW`, which is the prefix used by the Dreamweaver IDs. One useful approach is to prefix every ID with your domain name (with the elements reversed); for example, if your domain name is `joe.com`, you could start every ID with `com.joe.` to ensure uniqueness.

#### Contents

None.

#### Container

This tag must be contained in a `menu-insert` tag that is inside a `server-format-definition-changes` tag.

#### Example

```
<format
file = "Date/Time"
title = "Date/Time - 14:35"
expression = "<%\s*=\sDoDateTime\(.*, 4, 1033\)\s*%"
strNamedFormat = "shortTime"
```

```
nLCID = 1033
id = "JMMenu_ServerFormatDef_ASP_2_DT18" />
```

## separator

### Description

Indicates that a separator should be inserted into a menu at the location specified by the containing `menu-insert` tag.

### Attributes

`id`, {platform}

`id` The ID for the separator; each separator ID must be unique. Your separator IDs should start with a company name or some other namespace prefix to ensure uniqueness. In particular, don't start your IDs with `DW`, which is the prefix used by the Dreamweaver menu IDs. One useful approach is to prefix every ID with your domain name (with the elements reversed); for example, if your domain name is `joe.com`, you could start every ID with `com.joe.` to ensure uniqueness.

`platform` Indicates that the separator should appear only on the given platform. Valid values are "win" and "mac".

### Contents

None.

### Container

This tag must be contained in a `menu-insert` tag.

### Example

```
<separator id = "JMMenu_Commands_ConvertFrobs_Separator" platform = "win" />
```

## comment

### Description

Provides a comment about an item being inserted into the menu structure. The Extension Manager inserts this comment (in the form of an XML comment tag) into the `menus.xml` file as it installs the extension.

### Attributes

None.

### Contents

The text of a comment.

### Container

This tag must be contained in a `menu-insert` tag.

### Example

```
<comment>This command is part of the Mugwump extension.</comment>
```

## shortcut-insert

### Description

Indicates that a keyboard shortcut or shortcut list should be added to the `menus.xml` file.

### Attributes

`list_Id`, {`xml:lang`}

`list_Id` The ID of the shortcut list into which the shortcut should be inserted. Use this attribute only when inserting a single shortcut into a list. Don't use it when inserting an entire list.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

This tag must contain a `shortcut` tag or a `shortcutlist` tag.

### Container

This tag must be contained in a `configuration-changes` tag.

### Example

```
<shortcut-insert list_Id = "DWMainWindow">  
  <!-- shortcutlist or shortcut tag goes here -->  
</shortcut-insert>
```

## shortcut-remove

### Description

Indicates that the specified keyboard shortcut be removed from the `menus.xml` file.

### Attributes

`id`, {`xml:lang`}

`id` ID of the shortcut or shortcut list to remove. A shortcut list is removed only if it's empty.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

None.

## Container

This tag must be contained in a `configuration-changes` tag.

## Example

```
<shortcut-remove id = "DWMainWindow" />
```

## shortcutlist

### Description

Specifies a shortcut list to be added to the `menus.xml` file.

### Attributes

`id`, `{platform}`

`id` ID for the new shortcut list. It should be the same as the menu ID for the menu bar that represents a window in Dreamweaver with which the shortcuts are associated. Currently supported IDs are `DWMainWindow`, `DWMainSite`, `DWTimelineInspector`, and `DWHTMLInspector`.

`platform` Indicates that the shortcut list should appear only on the given platform. Valid values are "win" and "mac".

### Contents

None.

### Container

This tag must be contained in a `shortcut-insert` tag.

### Example

```
<shortcutlist id = "NewMenuBar" platform = "win" />
```

## shortcut

### Description

Specifies a keyboard shortcut to be added to the `menus.xml` file.

`key`, `id`, `{command}`, `{file}`, `{platform}`

`key` The key combination used to activate the keyboard shortcut. For syntax details that apply to Dreamweaver, see "About customizing Dreamweaver menus" in the "Customizing Dreamweaver" chapter of *Using Dreamweaver*.

`id` A unique identifier for a shortcut. Your shortcut IDs should start with a company name or some other namespace prefix to ensure uniqueness. One useful approach is to prefix every ID with your domain name (with the elements reversed); for example, if your domain name is `joe.com`, you could start every ID with `com.joe.` to ensure uniqueness.

`command` The JavaScript code to execute when the user issues the keyboard shortcut.

`file` A file containing the JavaScript code to execute when the user issues the keyboard shortcut. The `file` attribute overrides the `command` attribute; either `file` or `command` must be specified for each shortcut.

`platform` Specifies that the shortcut works only on the indicated platform. Valid values are "win" and "mac".

#### Contents

None.

#### Container

This tag must be contained in a `shortcut-insert` tag.

#### Example

```
<shortcut key = "Shift+F5" command = "dw.newDocument()" id = "ShortCutTest"
platform = "win" />
```

## taglibrary-changes

#### Description:

Describes changes to be made to the `TagLibraries.vtm` file.

#### Attributes

None.

#### Contents

This tag may contain `taglibrary-insert` and `taglibrary-remove` tags.

#### Container

This tag must be contained in a `configuration-changes` tag.

## taglibrary-insert

#### Description

Inserts the specified tag library at the end of file. Order is not important.

#### Attributes

```
{xml:lang}
```

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

This tag may contain `taglibrary` tags that describe a tag library to be inserted. The Extension Manager verifies only that the XML structure is valid.

#### Container

This tag must be contained in a `taglibrary-changes` tag.

## taglibrary-remove

### Description

Removes the specified tag library.

### Attributes

`id`, {`xml:lang`}

`id` ID of the tag library to be removed.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage](#) on page 12. If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\)](#) on page 54.

### Contents:

None.

### Container

This tag must be contained in a `taglibrary-changes` tag.

## toolbar-changes

### Description

Marks the beginning of toolbar changes.

{`file`}

`file` This optional attribute specifies the name of the toolbar file to edit. If not supplied, the default is `Toolbars.xml`.

### Contents

This tag may contain the `toolbar-insert`, `toolbar-item-insert`, `toolbar-remove`, and `toolbar-item-remove` tags.

### Container

This tag must be contained in the `configuration-changes` tag.

### Example

This example illustrates the syntax and hierarchy of all tags that can be contained by the `toolbar-changes` tag.

```
<toolbar-changes [file="file_name"]>
  <toolbar-insert>
    <toolbar ...>
      ...
    </toolbar>
  </toolbar-insert>
  <toolbar-remove id="toolbar_id" />
  <toolbar-item-insert
```

```
insertBefore|insertAfter|appendTo|prependTo="toolbar_or_item_id"
toolbar="toolbar_id">
<itemtype.../>
</toolbar-item-insert>
<toolbar-item-remove id="toolbar_item_id" />
</toolbar-changes>
```

## toolbar-insert

### Description

Inserts the specified toolbar at the end of file.

### Attributes

{xml:lang}

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

The `toolbar` tag, which describes the toolbar to be inserted. The Extension Manager verifies only that the XML structure is valid.

### Container

This tag must be contained in the `toolbar-changes` tag.

### Example

```
<toolbar-insert>
  <toolbar ...>
    ...
</toolbar>
```

## toolbar-remove

### Description

Removes the specified toolbar.

### Attributes

`id`, {xml:lang}

`id` ID of the toolbar to be removed.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

None.

#### Container

This tag must be contained in the `toolbar-changes` tag.

#### Example

```
<toolbar-remove id="toolbar_id" />
```

## toolbar-item-insert

#### Description

Inserts the specified toolbar item at the specified location.

#### Attributes

`insertBefore|insertAfter, appendTo|prependTo, toolbar, {xml:lang}`

`insertBefore|insertAfter` ID of the existing toolbar item before or after which the specified item is inserted.

`appendTo|prependTo` ID of the existing toolbar to which the specified item is appended or prepended.

`toolbar` ID of the toolbar to append to if the `insertBefore|insertAfter` item isn't found.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

#### Contents

A tag that describes a toolbar item to be inserted. The Extension Manager verifies only that the XML structure is valid.

#### Container

This tag must be contained in the `toolbar-changes` tag.

#### Example

```
<toolbar-item-insert
  insertBefore|insertAfter|appendTo|prependTo="toolbar_or_item_id"
  toolbar="toolbar_id">
  <itemtype.../>
</toolbar-item-insert>
```

## toolbar-item-remove

### Description

Removes the specified toolbar item.

### Attributes

`id`, `{xml:lang}`

`id` ID of the toolbar item to remove.

`xml:lang` Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

None.

### Container

This tag must be contained in the `toolbar-changes` tag.

### Example

```
<toolbar-item-remove id="toolbar_item_id" />
```

## extensions-changes

### Description

Container tag that describes any changes to the `Extensions.txt` file, such as adding or removing extensions that you can open in Dreamweaver.

### Attributes

None.

### Contents

This tag may contain an `extension-insert` tag and an `extension-remove` tag.

### Container

This tag must be contained in a `configuration-changes` tag.

### Example

```
<extensions-changes>  
  <!--extension-insert and extension-remove tags go here-->  
</extensions-changes>
```

## extension-insert

### Description

Describes a new extension that Dreamweaver can open.

### Attributes

extension, description, {xml:lang}

extension Specifies name of the extension, such as .gif or .htm.

description Describes what the extension is used for.

xml:lang Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

### Contents

None.

### Container

This tag must be contained in an `extensions-changes` tag.

### Example

```
<extension-insert extension="PHP" description="PHP files"/>
```

## extension-remove

### Description

Indicates an extension to remove from the Extensions.txt file.

### Attributes

extension, {description}, {xml:lang}

extension The name of the extension, such as .gif or .htm.

description This optional tag allows you to specify a description of the extension being removed. If no description is provided, the extension is removed from all lines of the Extensions.txt file.

xml:lang Specifies the language for the listed file. Extension Manager compares this language with the user language, which is determined by the process outlined in [defaultLanguage on page 12](#). If the languages match, configuration changes are applied; if not, they are ignored. If Extension Manager can't determine the user language, it applies all configuration changes regardless of their specified language.

For more information, see [Creating multilingual extension packages \(version 2.1 and later\) on page 54](#).

## Contents

None.

## Container

This tag must be contained in an `extensions-changes` tag.

## Example

```
<extension-remove extension="PHP" description="PHP files"/>
```

or

```
<extension-remove extension="PHP" />
```

## Tags and their compatible products

The following table lists each tag and its compatible Adobe applications. An X in the application column indicates that the tag is compatible with that application. If there is no X in the column for that application, then the tag is not compatible with the application. If a tag is not supported by an application, but is included in the MXI file, it will be ignored during installation of the extension

Tag	Dreamweaver CS5	Fireworks CS5	Flash CS5	Other CS5 applications
macromedia-extension	X	X	X	X
description	X	X	X	X
license-agreement	X	X	X	X
ui-access	X	X	X	X
products	X	X	X	X
product	X	X	X	X
author	X	X	X	X
files	X	X	X	X
file	X	X	X	X
configuration-changes	X			
documenttype-changes	X			
documenttype-insert	X			
documenttype-remove	X			
toolpanel-changes			X	
toolpanel-item-insert			X	
ftp-extension-map-changes	X			
ftp-extension-insert	X			
ftp-extension-remove	X			
insertbar-changes	X			
insertbar-insert	X			
insertbar-remove	X			
insertbar-item-insert	X			
insertbar-item-remove	X			
server-behavior-changes	X			
server-format-changes	X			
server-format-definition-changes	X			
data-source-changes	X			

<b>Tag</b>	<b>Dreamweaver CS5</b>	<b>Fireworks CS5</b>	<b>Flash CS5</b>	<b>Other CS5 applications</b>
menu-insert	X			
menu-insert	X			
menu	X			
menu-insert	X			
format	X			
separator	X			
comment	X			
shortcut-remove	X			
shortcut-insert	X			
shortcutlist	X			
shortcut	X			
taglibrary-changes	X			
taglibrary-insert	X			
taglibrary-remove	X			
toolbar-changes	X			
toolbar-insert	X			
toolbar-remove	X			
toolbar-item-insert	X			
toolbar-item-remove	X			
extensions-changes	X			
extension-insert	X			
extension-insert	X			
Tags and their compatible products	X	X	X	
token	X	X	X	

## Example MXI file

The following is an example of an MXI file that creates a Dreamweaver extension called Dog and Cat Extension Suite. This particular extension installs an object and a command, and modifies menu.xml and insertbar.xml to add menu items to the application's interface.

```
<macromedia-extension
  name="Dog and Cat Extension Suite"
  version="1.0.1"
  mxiversion="5.0"
  xmanversion="5.0"
  icon="icon.png"
  requires-restart="true"
  type="suite">

  <author name="Macromedia" />

  <products>
    <product name="Dreamweaver" version="11" primary="true" />
  </products>
  <description href="http://www.diamond.com/calendarobject/description.htm"
    source="com.diamond/calendarobject/description.htm">
    <![CDATA[This extension installs an object and a command while modifying
    menus.xml and insertbar.xml.]]>
  </description>
  <ui-access>
    <![CDATA[Access the Dog command by selecting Commands > Dog. Access the
    Cat object by selecting the Cat category within the Insert bar.]]>
  </ui-access>
  <license-agreement>
    <![CDATA[SAMPLE THIRD PARTY LICENSE TEXT:<br>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy
    nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut
    wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit
    lobortis nisl ut aliquip ex ea commodo consequat.]]>
  </license-agreement>
  <files>
    <file source="cat.htm"
      destination="$dreamweaver/configuration/objects/cat"/>
    <file source="cat.gif"
      destination="$dreamweaver/configuration/objects/cat"/>
    <file source="description.htm"
      destination="$ExtensionSpecificEMStore/com.diamond/calendarobject/
description.htm"/>
    <file source="icon.png"
      destination="$ExtensionSpecificEMStore/icon.png"/>
    <file source="dog.htm"
      destination="$dreamweaver/configuration/commands"/>
    <file source="dog.js"
      destination="$dreamweaver/configuration/commands"/>
  </files>
  <configuration-changes>
    <menu-insert insertAfter="DWMenu_Commands_SortTable"
      skipSeparator="true">
      <menu id="DWMenu_Commands_Dog" name="_Dog">
        </menu>
      </menu-insert>
    <menu-insert appendTo="Animals_Menu_Dog">
      <menuItem id="DWMenu_Commands_Dog_Dog1" name="_Insert Dog" />
      <menuItem id="DWMenu_Commands_Dog_Dog2" name="Insert _Dog Again" />
    </menu-insert>
  </configuration-changes>
</macromedia-extension>
```

```

    <menuitem id="DWMenu_Commands_Dog_Dog3" name="Insert Dog _Again Again"
      file="commands dog.htm" />
  </menu-insert>
</insertbar-changes>
<insertbar-insert>
  <category folder="Cat" id="DW_Inserbar_Cat">
    <button file="cat/cat.htm" id="DW_Inserbar_Cat_Cat1" image="cat
      cat.gif" />
  </category>
</insertbar-insert>
</insertbar-changes>
</configuration-changes>
</macromedia-extension>

```

You can get more examples here:

[http://help.adobe.com/en\\_US/extensionmanager/cs/using/sample.zip](http://help.adobe.com/en_US/extensionmanager/cs/using/sample.zip).

The sample.zip file contains three examples, including MXI files and other necessary files.

- <sample.zip>/Samples/html illustrates how to use an HTML file for extension description.
- <sample.zip>/Samples/multilingual combines multiple language versions of files in one extension.
- <sample.zip>/Samples/net update demonstrates an updateable extension.

## Creating multilingual extension packages (version 2.1 and later)

Since version 2.1, Extension Manager supports multilingual extension packages, which let you combine multiple language versions in one ZXP/MXP file. During installation, the appropriate language is determined by the process outlined in [defaultLanguage on page 12](#). Then, language-specific files and text strings are identified using attributes you've included in the MXI file.

### Enabling multilingual support in an MXI file

To enable multilingual support in an MXI file, you need to specify the attribute `ismultilingual="true"` to load strings from external files. If you want to localize the name of the extension, you can specify a `name_resid` attribute in the `<macromedia-extension>` tag. If no corresponding language tag is found for the id, "name\_ID", Extension Manager falls back to the name in the name attribute.

```

<macromedia-extension
  name="Extension Name"
  name_resid="name_ID"
  version="1.0.0"
  ismultilingual="true" >

<defaultLanguage>en_US</defaultLanguage>
<author name="FooBar" author_resid="author_ID"/>
<description resid="description_ID">
<![CDATA[ Inline Description ]]>
</description>

```

## Setting up the folder hierarchy for localized XML files

At the same location as your .mxi file, create a folder with the exact name of your .mxi file and append "Resources" to it. For example if you have an .mxi file named "Calendar.mxi", create a folder named "Calendar.mxi\_Resources". In this folder, you add an XML file for each language you are localizing the extension into. Each file will have a two-letter ISO language code followed by an underscore character, and then the two-letter ISO country code in upper case. For example, for English you specify "en\_US.xml" and for French "fr\_FR.xml".

If your extension needs to install localized files, you may want to create a subfolder for each of the languages. The folder hierarchy should look as follows:

Calendar.mxi - MXI File

Calendar.mxi\_Resources (folder)

    en\_US.xml - XML File containing English strings

    fr\_FR.xml - XML File containing French Strings

en\_US (Folder Containing English files)

fr\_FR (Folder Containing French files)

## Formatting XML for language-specific strings

The Extension Manager looks up localized strings from XML files that you provide for each language. Each XML file should use Adobe's zstring format. Below is an example of the French xml file. The locale (in this case, "fr\_FR") should match the locale of the language. Extension Manager looks up the localized strings based on the name="" attribute and substitutes the localized strings in the <val> tags. In the following example, we use name\_ID for the localized Extension Name, "French Extension Name". And description\_ID will be used for the Description displayed when you click the Extension in the Extension Manager.

Example

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE asf SYSTEM "http://ns.adobe.com/asf/asf_1_0.dtd">
<asf locale="fr_FR" version="1.0" xmlns="http://ns.adobe.com/asf">
  <set name="DefaultSet">
    <str name="name_ID">
      <val>French Extension Name</var>
    </str>
    <str name="description_ID">
      <val>
        French Extension Description.
      </val>
    </str>
  </set>
</asf>
```

## Installing localized files

To specify that a set of localized files get installed for a particular language, use the xml:lang attribute on the files tag containing files for that language.

Example

```
<files xml:lang="en_US">
  <file source="en_US/Jacket.htm" destination="$dreamweaver/configuration"/>
</files>

<files xml:lang="fr_FR">
```

```

<file source="fr_FR/Jacket.htm" destination="$dreamweaver/configuration"/>
</files>

<files>
<file source="styles.css" destination="$dreamweaver/configuration" />
</files>

```

## Configuration changes

For Dreamweaver, you can provide localized strings for changes to the XML files in the `<configuration-changes>` section of the mxi file by using the `resid:attributeName` and providing a string id in the string resource files. For example, to provide a localized menu item string, you add `resid:name="menuItem_ID"` to the `<menuItem>` tag.

### Example

```

<configuration-changes>
  <menu-insert appendTo="DWMenu_Insert">
    <menu id="MyItem_Insert" name="Menu Item Default" resid:name="menuItem_ID" >
  </menu-insert>
</configuration-changes>

```

If no localized string is available for the language being installed, the default string in the `name=attribute "Menu Item Default"` is inserted. Using this method, you can provide localized strings for the following attributes:

Menu Items: `<menuItem resid:name="menuItem_ID" />`

Button Names: `<button resid:label="buttonLabel_ID" />`

## Creating a ZXP extension package (version 5.0 only)

In addition to the traditional proprietary MXP format, Extension Manager CS5 supports the ZXP package format. ZXP is based on the public ZIP standard. A ZXP format extension package is named with the `.zxp` file-name extension.

There are three kinds of ZXP format extensions: ordinary extensions, CS extensions, and hybrid extension. CS extensions don't contain an MXI file, whereas the other two kinds of extension must contain a MXI file. Extension Manager can install and remove CS extensions, but it cannot package them. For information on how to package CS extensions, see the Creative Suite SDK: <http://www.adobe.com/devnet/creativesuite/sdk> .

To create ordinary ZXP or hybrid ZXP extensions, you can select **File > Package ZXP Extension** from the Extension Manager workspace. You can also use command line to create ZXP format extensions. Refer to [Use Extension Manager from the command line on page 60](#)

## Sign ZXP format extension

Users want to feel confident that the software they're installing comes from a reliable developer, and that what they're installing hasn't been modified since that developer released it. It is recommended that you sign ZXP extensions so that people installing them can verify that the extension has been published by you and that it has not been tampered with.

For information on signing extensions, see <http://www.adobe.com/devnet/creativesuite/sdk> .

## Validate ZXP format extension

Upon installation of ZXP format extension, Extension Manager validates its digital signature. For some validation results, it prompts the user to decide whether to continue with the installation.

## Creating Hybrid Extension packages (version 5.0 only)

Extension Manager CS5 supports hybrid extension , which let you package two types of files in one extension: CS extension package file with name ".zxp", and files can be contained in hybrid and ordinary extensions. These two types of files are installed in different ways. CS extension files are installed to a fixed folder, while hybrid and ordinary extension files are installed to the folder specified by destination attribute. So to create hybrid extension packages, you have to specify file type. You can do it in two ways: use `file-type` attribute in `<file>` tag, or use `default-file-type` attribute in `<files>` tag.

### Use `file-type` attribute

For CS extension package file, you need to specify `file-type` attribute as `csxs`. For ordinary file, you need to specify `file-type` attribute as `ordinary` or `plugin`. For more information, refer to `file-type` attribute in `<file>` tag.

#### Example

In the below example, file "MyCreativeSuiteExtension.zxp" is a CS Extension package file, and file "foo" is ordinary file. "MyCreativeSuiteExtension.zxp" is installed according to a fixed folder, so you don't need to specify the `destination` attribute. Even you do, it is ignored. While for file "foo", you has to specify `destination` like below example illustrates.

```
<files>
  <file source="MyCreativeSuiteExtension.zxp" file-type="csxs" />
  <file source="foo" destination="$dreamweaver/configuration" file-
    type="ordinary" />
</files>
```

### Use `default-file-type` attribute

You can specify `default-file-type` attribute in `<files>` tag, so any file wrapped in that files tag use that file type if the file doesn't have attribute `file-type`. For more information, refer to `default-file-type` attribute in `<files>` tag.

#### Example

In the below example, files "MyCreativeSuiteExtension1.zxp" and "MyCreativeSuiteExtension2.zxp" are CS extension package, while files "foo1" and "foo2" are ordinary files.

```
<files default-file-type="csxs">
  <file source="MyCreativeSuiteExtension1.zxp" />
  <file source="MyCreativeSuiteExtension2.zxp" />
</files>

<files default-file-type="ordinary">
  <file source="foo1" destination="$dreamweaver/configuration" />
```

```
<file source="foo2" destination="$dreamweaver/configuration" />
</files>
```

## Creating plug-in extension packages for InDesign CS5 and InCopy CS5 (version 5.0 only)

Extension Manager CS5 supports plug-in extension package, which let you to include plug-ins for InDesign CS5 or InCopy CS5 in extensions. On Mac OS, one plug-in is one package. On Windows, one plug-in includes one binary file(usually have file extension “.pln”) and one resource directory. To allow Extension Manager recognize plug-in, you need to specify its file type. On Mac OS, specify the plug-in file’s type as "plugin", while on Windows, you need to specify the type of binary file(with file extension pln) as "plugin", but not for resource directory. File type of resource directory should be "ordinary". To specify file type, you need to specify attribute "file-type" in <file> tag or attribute "default-file-type" in <files> tag.

There are two kinds of plug-in extensions: Enable\_for\_all\_disable\_for\_one extension and Enable\_for\_one\_disable\_for\_one extension.

### Creating Enable\_for\_all\_disable\_for\_one extension packages

Plug-ins included in an Enable\_for\_all\_disable\_for\_one extension works for every user on the machine after one user installs this extension. While if one user disables or even removes the extension, plug-ins still work for other users in operating system.

To create Enable\_for\_all\_disable\_for\_one extension packages, you need to specify the attribute "plugin-manager-type" in tag <macromedia-extension> as "all-users". And the destination of plug-ins needs to be inside the folder \$indesign/Plug-Ins.

#### Example

```
<macromedia-extension
  name="Enable_for_all_disable_for_one_sample"
  version="1.0.0"
  plugin-manager-type="all-users" >

<files>
  <file source="sample.pln" destination="$indesign/Plug-Ins/sample/
  sample.pln" platform="win" file-type="plugin" />
  <file source="(sample Resources)" destination="$indesign/Plug-Ins/sample"
  platform="win" />
  <file source="sample.InDesignPlugin" destination="$indesign/Plug-Ins/
  sample/sample.pln" platform="mac" file-type="plugin" />
</files>
```

### Creating Enable\_for\_one\_disable\_for\_one extension packages

Plug-ins included in an Enable\_for\_one\_disable\_for\_one works only for the user who installs this extension. And when user disables or removes this extension, no user in operating system could use plug-ins any more.

To create Enable\_for\_one\_disable\_for\_one extension packages, you need to specify the attribute "plugin-manager-type" in tag <macromedia-extension> as "current-user". And the destination of plug-ins should not be inside the folder "\$indesign/Plug-Ins".

#### Example

```
<macromedia-extension
```

```

name="Enable_for_one_disable_for_one_sample"
version="1.0.0"
plugin-manager-type="current-user" >

<files>
  <file source="sample.pln" destination="$indesign_user/Plug-Ins/sample/
sample.pln" platform="win" file-type="plugin" />
  <file source="(sample Resources)" destination="$indesign_user/Plug-Ins/
sample" platform="win" />
  <file source="sample.InDesignPlugin" destination="$indesign/Plug-Ins/
sample/sample.pln" platform="mac" file-type="plugin" />
</files>

```

## Creating updatable extension packages (version 5.0 only)

Extension Manager CS5 supports updatable extension packages, which allows end user of extension to update extension once there is new version available. To enable this feature, you need to include in MXI file a link which points to update information in extension. This link actually points to an online xml file, called update information file, which contains two information: whether new update is available, and how to update to the latest extension.

### Creating update information file

This file should be encoded with utf-8. It contains tag “version”, “download” and “description”

#### version

Specifies the version of the latest extension. It needs to have same format with the attribute “version” in the “`macromedia-extension`” tag in MXI.

#### download

This tag specifies a url starting with "http" or "https". This url can link to the latest extension itself, in this case, it must end with ".zxp" or ".mxp". It can also link to a webpage containing descriptions and instructions about how to get and install the latest extension, in this case, it must not end with ".zxp" or ".mxp".

#### description

This tag specifies a short description about what's new for the latest extension. It can include an optional attribute "url" which links to a webpage with detailed release notes.

#### Example

```

<ExtensionUpdateInformation>
  <version>1.5.0</version>
  <download>http://www.foobar.com/extensions/foo.zxp</download>
  <description url="http://www.foobar.com/extensions/fool5releasenotes.htm">
  <![CDATA[
The 1.5 version fix a lot of bugs<br>
It also adds great new things.]]>
</description>
</ExtensionUpdateInformation>

```

## Checking update upon launch of Extension Manager

If you specify the update information link in tag "update", Extension Manager tries to follow the link to retrieve the update information file when checking update upon the launch of Extension Manager. If version in the retrieved file doesn't match the version of installed extension, Extension Manager notifies end user there is an update available. Then if end user chooses to update extension, Extension Manager starts update procedure which is depending on url specified by tag "download" in the retrieved xml file. If tag "download" specifies a url ending with ".zxp" or ".mxp", Extension Manager downloads and installs the new version automatically. Otherwise, Extension Manager opens the url in an operating system default web browser.

## Use Extension Manager from the command line

Extension Manager can run command through both CLI and BridgeTalk. To run command through CLI, start a command shell and invoke the command described below. To run command through BridgeTalk, you need to specify the first parameter of the command as "-EMBT", then send the command through BridgeTalk to Extension Manager.

### Parameter definitions

- "-install" installs the extension.
- "-package" packages the extension.
- "-remove" removes the extension.
- "-enable" enables the extension.
- "-disable" disables the extension.
- "-locate" locates point product in Extension Manager window.
- "-from" specifies command originator when command line parameter is passed through BridgeTalk. After executing command, Extension Manager sends result back to specified originator. Extension Manager CS5 only.
- "-quit" quit Extension Manager. Extension Manager CS5 only.
- "-suppress" suppresses the UI of Extension Manager.
- "-locale" specifies language for Extension Manager at startup.
- "-EMBT" used only when command line parameter is passed through BridgeTalk. "-EMBT" should appear before all other command line parameters.

### Attribute definitions

- "mxi" specifies the name and location of the extension installation file.
- "mxp" specifies the name and location of the package file in MXP format.
- "zxp" specifies the name and location of the package file in ZXP format. Extension Manager CS5 only.
- "product" specifies the application that originate the command if following parameter "-from". Specifies the application that uses the extension if following other parameters.
- "extension" specifies the name of the extension (as specified in the MXI file).
- "lang" specifies the locale language code, such as en\_US.

- "timeout" specifies the maximum seconds the Extension Manager waits for the application to quit before executing the command. If the attribute force-quit is set to true, then the extension can't be installed, disabled, re-enabled, or removed until the application is quit. This attribute is an optional attribute which value must be a positive integer from 0 through 1000. Extension Manager CS5 only.
- "pcdentry" specifies the BridgeTalk identifier of command originator. Extension Manager CS5 only

## Parameter formats

Required parameter	Following parameters/attributes allowed
-install	mxi="" [timeout=""], mxp="" [timeout=""], zxp="" [timeout=""]
-package	mxi="" mxp="", mxi="" zxp=""
-remove	product="" extension="" [timeout=""]
-enable	product="" extension="" [timeout=""]
-disable	product="" extension="" [timeout=""]
-locate	product=""
-from	product="", pcdentry=""
-locale lang=""	-from, -locate, -install, -package, -remove, -enable, -disable
-quit	None
-suppress	-quit, -from, -locate, -install, -package, -remove, -enable, -disable
-EMBT	any parameter above

Extension Manager CS5 can execute multiple requests in one command, which means you can specify multiple "-install", "-package", "-remove", "-enable", "-disable" groups in one command.

## Command execution result

Extension Manager returns the result of command execution in two scenarios.

- Command is executed in headless mode.
- "-from" parameter is specified in command.

## Headless mode

Headless mode is a special command execution mode. In this mode, there is no UI shown during the command execution. After executing the command, Extension Manager returns "0" if execution succeeds. Otherwise, Extension Manager returns non-zero error code, and localized UTF-16(Windows)/UTF-8(Mac OS) encoded error message is returned through std:error.

To execute command in headless mode,

- Windows: "XManCommand.exe" -suppress ...
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -suppress ...

## "-from" parameter

If "-from" parameter is specified in the command. Extension Manager passes execution result through BridgeTalk to command originator specified by "product" or "pcdentry" attribute. If execution succeeds, the result is "0". Otherwise, the result is "'error code' 'localized UTF-16(Windows)/UTF-8(Mac OS) encoded error message'".

## Return code(Extension Manager CS5 only)

```
0 -- Succeeded
1 -- Install extension failed
2 -- Remove extension failed
3 -- Enable extension failed
4 -- Disable extension failed
5 -- Package extension failed
101 -- Incorrect CLI Format
102 -- The specified product not exist
103 -- The specified extension not exist
104 -- The specified extension is already enabled,
105 -- The specified extension is already disabled.
7 -- There is already an instance of Extension Manager.
```

## Execute command through BridgeTalk

Extension Manager can execute command passed through BridgeTalk. All command sent through BridgeTalk must begin with "-EMBT" parameter. To send command to Extension Manager, specify the target BridgeTalk Identifier as "exman-5.0".

## Examples:

### Install extension through CLI

- Windows: "Adobe Extension Manager CS5.exe" -install mxp="d:\test.mxp"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -install zxp="/Volumes/x1/forcequit.zxp" timeout=30

### "Package extension through CLI

- Windows: "Adobe Extension Manager CS5.exe" -package mxi="d:\test.mxi" zxp="d:\test.zxp"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -package mxi="/Volumes/x1/test.mxi" mxp="/Volumes/x1/test.mxp"

### "Remove extension through CLI

- Windows: "Adobe Extension Manager CS5.exe" -remove product="Dreamweaver CS5" extension="Sample"  
"Adobe Extension Manager CS5.exe" -remove productfamily="Photoshop-12" extension="Sample"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -remove product="Dreamweaver CS5" extension="Sample"

### "Enable extension through CLI

- Windows: "Adobe Extension Manager CS5.exe" -enable product="Dreamweaver CS5" extension="Sample"  
"Adobe Extension Manager CS5.exe" -enable productfamily="Photoshop-12" extension="Sample"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -enable product="Dreamweaver CS5" extension="Sample"

### "Disable extension through CLI

- Windows: "Adobe Extension Manager CS5.exe" -disable product="Dreamweaver CS5" extension="Sample"  
"Adobe Extension Manager CS5.exe" -disable productfamily="Phtshop-12" extension="Sample"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -disable product="Dreamweaver CS5" extension="Sample"

### "Locate the point product in Extension Manager window through CLI

- Windows: "Adobe Extension Manager CS5.exe" -locate product="Dreamweaver CS5"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -locate product="Dreamweaver CS5"

### "Install extension and suppress EULA dialog and "Installation succeeded" confirmation through CLI

- Windows: "Adobe Extension Manager CS5.exe" -suppress -install mxp="d:\test.mxp"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -suppress -install mxp="/Volumes/x1/test.mxp"

### "Package extension through BridgeTalk

Run the below example script in ExtendScript Toolkit CS5 to send package command to Extension Manager through BridgeTalk.

- Windows:

```
var bt = new BridgeTalk();  
bt.target = "exman-5.0";  
bt.body = '-EMBT -package mxi="D:\\test.mxi" zxp="D:\\test.zxp";'  
bt.send();
```

- Mac OS:

```
var bt = new BridgeTalk();  
bt.target = "exman-5.0";  
bt.body = '-EMBT -package mxi="/Volumes/x1/test.mxi" zxp="/Volumes/x1/  
test.zxp";'  
bt.send();
```

### "Specify en\_US as language at startup

- Windows: "Adobe Extension Manager CS5.exe" -locale lang="en\_US" -install zxp="d:\test.zxp"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -locale lang="en\_US" -install mxp="/Volumes/x1/test.mxp"

### Invalid parameters

- Eight attributes (mxi, mxp, zxp, product, extension, lang, timeout, pcdentry) do not allow "-" at the front of them.
- -locale parameter must be followed by lang="en\_US", otherwise it is an invalid parameter.

### Valid irregular parameters (which support dragging or double-clicking to install and package)

#### Install

- Windows: "Adobe Extension Manager CS5.exe" "d:\test.mxp"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" "/Volumes/x1/test.mxp"

#### Package

- Windows: "Adobe Extension Manager CS5.exe" "d:\\test.mxi"  
"Adobe Extension Manager CS5.exe" -suppress "d:\\test.mxi"
- Mac OS: "/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" "/Volumes/x1/test.mxi"  
"/Applications/Adobe Extension Manager CS5/Adobe Extension Manager CS5.app/Contents/MacOS/Adobe Extension Manager CS5" -suppress "/Volumes/x1/test.mxi"

"If a string (for example, extension name, or mxi/mxp/zxp file name) does not contain a space character, it does not need to be included in ""

For example, both below are valid:

- -enable product="Flash CS5" extension=Sample
- -enable product="Flash CS5" extension="Sample"

## Use Extension Manager from the graphical user interface (GUI)

To package an extension in Extension Manager, do the following:

- 1 Start Adobe Extension Manager CS5.
- 2 Choose File > Package MXP Extension or Package ZXP Extension.
- 3 Browse to the extension's installation file, select it, and click OK (Windows) or Open (Mac OS).

