

Photoshop Actions File Format

- Contents
- Actions file format
 - Actions palette file
 - Actions file
 - Action set
 - Action
 - Command
- Action Manager format
 - Boolean
 - Byte string
 - Class
 - Double
 - Enumerated
 - File alias
 - File path
 - ID
 - Identifier
 - Index
 - Integer
 - Item
 - Key-item pair
 - Large integer
 - List
 - Name
 - Object
 - Offset
 - Property
 - Raw data
 - Reference
 - Reference item
 - Unicode string
 - Unit double
- Object array format
 - Object array
 - Object array key-item pair
 - Unit doubles
 - Specific handling
- Descriptor parsing
- Dialog mode
 - Decision table
 - Dialog recording options
- Parsing actions files

Contents

This document is intended to provide missing information about the format of Photoshop actions files. It complements and slightly corrects the original "official" information found in the [Actions](#) section (Additional File Formats) of the page [Adobe Photoshop File Formats Specification](#).

Note: unless indicated otherwise, all multi-byte values, i.e., integer numbers (including C-style 4-character constants), floating-point (double) numbers, and Unicode characters are coded in [big-endian](#) format.

Actions file format

Actions palette file

Name	Type	Kind	Description
			Adobe Photoshop preferences file containing all the actions sets available in the Actions Palette.

Actions Palette.psp	'8BPF'	Actions palette file	Warning: like most preferences files, the actions palette file is not updated in real-time: it is read by the application only once at start-up (launch) time and written back at shut-down (quit) time.
---------------------	--------	----------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Length (in bytes)	Description	Comments
4	Version (= 16)	32-bit integer.
4	Number of action sets	32-bit integer.
Variable	Sequence of action sets	Each in Action set format.

Actions file

Name	Type	Kind	Description
*.atn	'8BAC'	Actions file	Adobe Photoshop actions file containing a set of related actions; generally produced by saving an action set from the Actions Palette.

Length (in bytes)	Description	Comments
4	Version (= 16)	32-bit integer.
Variable	Action set	Action set format.

Action set

Length (in bytes)	Description	Comments
Variable	Action set name	Unicode string format.
1	Expanded	Boolean; true if the action set is expanded in the Actions palette.
4	Number of actions	32-bit integer.
Variable	Sequence of actions	Each in Action format.

Action

Length (in bytes)	Description	Comments
2	Function key number	16-bit integer; function key used as keyboard shortcut; 0 if not used.
1	Shift key	Boolean; true if shift key needed for keyboard shortcut.
1	Command/Control key	Boolean; true if command/control key needed for keyboard shortcut.
2	Color index	16-bit integer; color values: <ul style="list-style-type: none"> • 0 (None) • 1 (Red) • 2 (Orange) • 3 (Yellow) • 4 (Green) • 5 (Blue) • 6 (Violet) • 7 (Gray)
Variable	Action name	Unicode string format.
1	Expanded	Boolean; true if the action is expanded in the Actions palette.
4	Number of commands	32-bit integer.
Variable	Sequence of commands	Each in Command format.

Command

Length (in bytes)	Description	Comments
1	Expanded	Boolean; true if the command is expanded in the Actions palette.
1	Enabled	Boolean; true if the command is enabled.
1	With dialog	Boolean; true if dialogs should be displayed (cf. Dialog mode).
1	Dialog options	8-bit integer; options for displaying dialogs (cf. Dialog mode).
4	Event ID format selector	C-style 4-character constant: either 'long' (CharID) or 'TEXT' (StringID).
4 or variable	Event ID (CharID or StringID)	C-style 4-character constant if CharID; Byte string format if StringID.
Variable	Dictionary name	Byte string format.
4	Has descriptor	32-bit integer; either 0 (no descriptor) or -1 (descriptor).
0 or variable	Descriptor	Object format, if present.

Action Manager format

Boolean

Length (in bytes)	Description	Comments
1	Boolean value	8-bit integer.

Byte string

Length (in bytes)	Description	Comments
4	Number of characters	32-bit integer.
Variable	String of characters	One byte per character.

Class

Length (in bytes)	Description	Comments
Variable	Class name (usually empty)	Unicode string format.
Variable	Class ID	ID format.

Double

Length (in bytes)	Description	Comments
8	Double	64-bit double.

Enumerated

Length (in bytes)	Description	Comments
Variable	Enumerated type ID	ID format.
Variable	Enumerated value ID	ID format.

File alias

Length (in bytes)	Description	Comments
4	Length (in bytes) of the alias structure	32-bit integer.

Variable	Alias structure	Opaque data used on Mac OS.
----------	-----------------	-----------------------------

File path

Length (in bytes)	Description	Comments
4	Length (in bytes) of the path structure	32-bit integer.
4	Unicode text signature (= 'utxt')	C-style 4-character constant, in little-endian format.
4	Length (in bytes) of the path structure	32-bit integer, in little-endian format.
4	Number of Unicode characters	32-bit integer, in little-endian format.
Variable	String of Unicode characters	Two bytes per character, in little-endian format; includes terminating null.

ID

Length (in bytes)	Description	Comments
4	Length	32-bit integer; CharID if 0, StringID otherwise.
4 or variable	CharID or StringID	C-style 4-character constant if CharID; string of characters if StringID.

Identifier

Length (in bytes)	Description	Comments
4	Identifier value	32-bit integer (unsigned).

Index

Length (in bytes)	Description	Comments
4	Index value	32-bit integer (unsigned).

Integer

Length (in bytes)	Description	Comments
4	Integer number	32-bit integer (signed).

Item

Length (in bytes)	Description	Comments
4	Type	C-style 4-character constant: <ul style="list-style-type: none"> • 'bool' (Boolean) • 'type' or 'GlbC' (Class) • 'doub' (Double) • 'enum' (Enumerated) • 'alis' (File alias) • 'Pth ' (File path) • 'long' (Integer) • 'comp' (Large integer) • 'VLLs' (List) • 'Objc' or 'Glb0' (Object) • 'tdta' (Raw data) • 'obj ' (Reference) • 'TEXT' (Unicode string) • 'UtfF' (Unit double) or: <ul style="list-style-type: none"> • 'ObAr' (Object array)

Variable	Value	Depending on type: <ul style="list-style-type: none"> • Boolean format • Class format • Double format • Enumerated format • File alias format • File path format • Integer format • Large integer format • List format • Object format • Raw data format • Reference format • Unicode string format • Unit double format or: <ul style="list-style-type: none"> • Object array format (cf. Specific handling)
----------	-------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Key-item pair

Length (in bytes)	Description	Comments
Variable	Key ID	ID format.
Variable	Item	Item format.

Large integer

Length (in bytes)	Description	Comments
8	Large integer number	64-bit integer (signed).

List

Length (in bytes)	Description	Comments
4	Number of items in the list	32-bit integer.
Variable	Sequence of items	Each in Item format.

Name

Length (in bytes)	Description	Comments
Variable	Name string	Unicode string format.

Object

Length (in bytes)	Description	Comments
Variable	Class	Class format.
4	Number of items in the object	32-bit integer.
Variable	Sequence of key-item pairs	Each in Key-item pair format.

Offset

Length (in bytes)	Description	Comments
4	Offset value	32-bit integer (signed).

Property

Length (in bytes)	Description	Comments
Variable	Key ID	ID format.

Raw data

Length (in bytes)	Description	Comments
4	Length (in bytes) of the raw data value	32-bit integer.
Variable	Raw data value	Sequence of bytes.

Reference

Length (in bytes)	Description	Comments
4	Number of items in the reference	32-bit integer.
Variable	Sequence of reference items	Each in Reference item format.

Reference item

Length (in bytes)	Description	Comments
4	Form	C-style 4-character constant: <ul style="list-style-type: none">'Class' (Class)'Enmr' (Enumerated)'Idnt' (Identifier)'indx' (Index)'name' (Name)'rele' (Offset)'prop' (Property)
Variable	Desired class	Class format.
Variable (0 for Class)	Value	Depending on form: <ul style="list-style-type: none">Class: nothingEnumerated formatIdentifier formatIndex formatName formatOffset formatProperty format

Unicode string

Length (in bytes)	Description	Comments
4	Number of Unicode characters	32-bit integer.
Variable	String of Unicode characters	Two bytes per character; includes terminating null.

Unit double

Length (in bytes)	Description	Comments
4	Unit ID	C-style 4-character constant: <ul style="list-style-type: none">'#Ang' (Angle)'#Rsl' (Density)'#Rlt' (Distance)'#Nne' (None)

		'#Prc' (Percent) <ul style="list-style-type: none"> '#Px1' (Pixels) or, for text code only: <ul style="list-style-type: none"> '#Mlm' (Millimeters) '#Pnt' (Points)
8	Double	64-bit double.

Object array format

Object array

Length (in bytes)	Description	Comments
4	Number of objects in the array	32-bit integer.
Variable	Object class	Class format.
4	Number of items in each object	32-bit integer.
Variable	Sequence of key-item pairs	Each in Object array key-item pair format.

Object array key-item pair

Length (in bytes)	Description	Comments
Variable	Key ID	ID format.
4	Item type	C-style 4-character constant: <ul style="list-style-type: none"> 'UnF1' (Unit doubles)
Variable	Item value	Depending on item type: <ul style="list-style-type: none"> Unit doubles format

Unit doubles

Length (in bytes)	Description	Comments
4	Unit ID	C-style 4-character constant: <ul style="list-style-type: none"> '#R1t' (Distance) '#Prc' (Percent) '#Px1' (Pixels)
4	Number of doubles	32-bit integer; should be the same as the number of objects in the array.
Variable	Sequence of doubles	Each as 64-bit double.

Specific handling

The 'ObAr' item type (assumed to be an "Object Array") can be found in some actions files, for instance in the file "Frames.atn" located in the "Presets/Actions" folder alongside the Photoshop application. This specific item type is not only undocumented, but also cannot appear directly in an ActionDescriptor since there are no associated methods putObjectArray or getObjectArray to handle it with.

In addition, it should be noted that the 'UnF1' item type (assumed to be a "Unit Floats") is not defined either (not to be confused with the standard 'UnTF' item type). Consequently, the object array is altogether a very special case which requires further conversion/translation.

In practice, the object array seems to be only used to describe a polygon made of a list of points (recorded while setting a selection using the Lasso tool, Polygonal Lasso tool, or Magnetic Lasso tool). It is clearly a "packed" format, used only in serialized form inside an actions file, in order to save space by factorizing unnecessarily repeated data, such as the class (point) and keys (horizontal and vertical) of each object, as well as the common unit (distance/percent/pixels) for all double coordinates, seeing that the number of points can be quite important, especially when doing a freehand selection with the Lasso tool.

Here is a quick-and-dirty JavaScript code snippet defining a function which computes how an object array would actually appear in a byte stream in "unpacked" format; it assumes that the current position in the parsed actions file is set just after a recognized 'ObAr' item type; it returns a byte stream beginning with a 'VL1s' item type, indicating that it represents a standard ArrayList:

```

function toListStream (file)
{
  function toBEIntValue (bytes)
  {
    var intValue = 0;
    for (var index = 0; index < bytes.length; index++)
    {
      intValue = (intValue << 8) + bytes.charCodeAt (index);
    }
    return intValue;
  }
  function readUnicodeString (file)
  {
    var unicodeLength = file.read (4);
    var unicodeLengthValue = toBEIntValue (unicodeLength);
    var unicodeString = file.read (unicodeLengthValue * 2);
    return unicodeLength + unicodeString;
  }
  function readId (file)
  {
    var idLength = file.read (4);
    var idLengthValue = toBEIntValue (idLength);
    var id = file.read ((idLengthValue) ? idLengthValue : 4);
    return idLength + id;
  }
  function readClass (file)
  {
    var classUnicodeString = readUnicodeString (file);
    var classId = readId (file);
    return classUnicodeString + classId;
  }
  //
  var objCount = file.read (4);
  var objCountValue = toBEIntValue (objCount);
  var objClass = readClass (file);
  var itemCount = file.read (4);
  var itemCountValue = toBEIntValue (itemCount);
  var itemKeyId = [ ];
  var itemUnitId = [ ];
  var itemDoubles = [ ];
  for (var itemIndex = 0; itemIndex < itemCountValue; itemIndex++)
  {
    itemKeyId.push (readId (file));
    var itemType = file.read (4);
    if (itemType !== 'UnFl')
    {
      throw new Error ("Unrecognized item type!");
    }
    itemUnitId.push (file.read (4));
    var doublesCountValue = toBEIntValue (file.read (4));
    if (doublesCountValue !== objCountValue)
    {
      throw new Error ("Inconsistent object count!");
    }
    itemDoubles.push (file.read (doublesCountValue * 8));
  }
  //
  var listStream = 'VLLs' + objCount;
  for (var objIndex = 0; objIndex < objCountValue; objIndex++)
  {
    listStream += 'Objc' + objClass + itemCount;
    for (var itemIndex = 0; itemIndex < itemCountValue; itemIndex++)
    {
      listStream += itemKeyId[itemIndex] + 'Unf' + itemUnitId[itemIndex];
      listStream += itemDoubles[itemIndex].substr (objIndex * 8, 8);
    }
  }
  return listStream;
}
}

```


Descriptor parsing

An interesting parsing strategy relies on the fact that each command's descriptor data found in the actions file matches the serialized format expected by the `ActionDescriptor.fromStream` method (in JavaScript), or by the corresponding `HandleToDescriptor` routine of the `ActionDescriptor` suite (in C/C++), except that it lacks the 4-byte header whose value in [big-endian](#) format is identical to the version number of the actions file itself (= 16).

Once the missing header is inserted at the beginning, each command's descriptor data can be easily decoded into an `ActionDescriptor`. It should be noted that object array ('ObAr') item types are correctly handled, i.e. automatically unpacked and converted to appropriate `ActionList` items.

The main drawback of that parsing method lies in the fact that each command's descriptor data is just a byte stream whose length is not known beforehand; as a result, it is still necessary to pre-decode each descriptor data by skipping all its fields, in order to compute its entire length and keep synchronized with the rest of the actions file. Surprisingly enough, the 32-bit field preceding the descriptor data is used as a boolean (0: no descriptor, or -1: descriptor) while it would have just been the right place to hold that useful length information...






Dialog mode

Decision table

Parsing an actions file can be used in different ways, for instance to play an action without the need to load the file in the Actions Palette.

For each command in the action, the `app.executeAction ()` method can be called with the "event ID" and the "action descriptor" directly available from the resulting data, whereas the dialog mode requires both "dialog options" and "with dialog" to be taken into account.

The "dialog options" value is set once and for all at the time of recording the action, while the "with dialog" flag can be interactively modified by the user in the Actions palette, by clicking in the toggle dialog visual mark located on the left of each command of the action, provided the mark is both visible and enabled.

Dialog options	With dialog	Toggle dialog visual mark (Actions palette)	Dialog mode for <code>app.executeAction ()</code>	
0	false	OFF		<code>DialogModes.NO</code>
	true	ON		<code>DialogModes.ALL</code>
1	false	Always ON		<code>DialogModes.ALL</code>
	true			
2	false	Always OFF		<code>DialogModes.NO</code>
	true			
3	false	Always OFF		<code>DialogModes.ALL</code>
	true			

Dialog recording options

While recording an action from the Actions palette, running a dedicated automation plug-in such as the [JSON Event Listener](#), part of the [JSON Action Toolbox](#), is very useful to monitor the event, descriptor, and dialog options associated with each recorded command.

Here are the different values of dialog options that can be encountered in practice:

Recording options	Value	Description
Optional	0	Display dialog only if necessary or requested by user.
Required	1	Always display dialog.
None	2	No dialog to display.
Specific	3	Display dialog (or not) in specific conditions.
Stop/Continue	4	Display stop dialog (with continue or not).

Notes:

- Values 3 and 4 are undocumented; what they stand for is a guess based on extensive tests.
- Examples of user actions resulting in specific conditions (value 3):
 - [Actions palette] > Insert Menu Item... (only for some menu items: Blur, Free Transform, Zoom In, etc.)
 - Image > Adjustments > Equalize (or Equalize... when active selection)
 - File > Import > Scanner XYZ...
 - Filter > Liquify...
 - Filter > Vanishing Point...

- User action bringing on a stop/continue dialog (value 4):
 - [Actions palette] > Insert Stop...
- The value 4 is never recorded as such in the actions file, it is always changed to 0.

Parsing actions files

A practical set of JavaScript functions for parsing actions files is contained in the module [jamActions](#), which is part of the [JSON Action Manager](#) scripting library. It is used by three utility scripts:

- [Convert Actions File](#): [Photoshop CS3 or later] convert an actions file (.atn) into a folder of directly executable scripts (.js) which can be further edited.
- [Parse Actions File](#): [Photoshop CS2 or later] parse an actions file (.atn) or an actions palette file (Actions Palette.psp) into a JSON text file.
- [Play Actions File Action](#): [Photoshop CS3 or later] play a specific action contained in an actions file (.atn), without the need to load the file in the Actions Palette.

All files are open-source and licensed under [GPLv3](#); the utility scripts have been successfully tested in Photoshop CS4 on Mac OS X, but should be platform agnostic.

Doc version: 2.5

Date: 2017-03-23

Copyright: © 2012-2017 Michel MARIANI

Disclaimer: this information is provided 'as is' without warranty of any kind, express or implied; use it at your own risk.