

AppleScript System Additions (OSAX)

by Donald Olson
AppleLink: D.OLSON
(version 0.3)
December 27, 1992

This is a very preliminary document. Feedback on the form
and content (or lack thereof) is greatly desired.

Copyright © 1992 Apple Computer Inc.
All rights reserved.

Overview

OSAX are extensions to the language and functionality of AppleScript much like XCMDs and XFCNs extend the usefulness of HyperTalk. Simply put, OSAX are Apple Event handlers or

coercions that are dynamically loaded by the AppleScript Runtime Engine.

Details

There are two basic types of OSAX: language extension (AE handlers) and data coercions (AE coercions). Each is contained in a file of type 'osax' and creator 'ascr'. An OSAX resource file may contain a combination of three basic resource forms. They are: 1) a code resource of type 'osax' that contains the executable code for the OSAX; 2) an aete resource that describes the functionality of the OSAX; and 3) any owned resources such as dialog definitions, strings, sounds, etc.

Each of the two basic types of 'osax' resource has a specific naming convention that tells the OSAX loader mechanism what type of OSAX it is.

1. Apple Event Handler OSAX (OSAX Handlers)

'osax' resources for Apple Event handlers follow the following convention:

resource type	'osax'
resource id	an identifier (e.g.. 6991)
resource name	'AEVTclssidid'

The 'AEVT' in the name above indicates that this is an OSAX using the event handler prototype. The next eight characters represent the events class and id.

The 'osax' code resource for an OSAX handler are in the form of an AppleEvent handler so the entry point for the code resource must follow the Apple Event handler function prototype as follows:

In C,

```
pascal OSErr MyAEHandlerFunction(AppleEvent *theEvent,  
                                   AppleEvent *theReply,  
                                   long *theRefCon);
```

In Pascal,

```
FUNCTION MyAEHandlerFunction( theEvent, theReply : AppleEvent; theRefCon:  
LONGINT): OSErr;
```

The OSAX handler is an extension to the AppleScript language so it needs to have an 'aete' resource that describes the syntax of the OSAX to AppleScript in a human readable form. The 'aete' can also be used to describe any objects or properties used by the OSAX. The resource id of the 'aete' must be the dialect code the 'aete' is written for. For example, the 'aete' for the example Flash OSAX in Appendix A is 0 indicating it is for the English dialect.

2. Apple Event Coercions OSAX (OSAX Coercions)

'osax' resources for Apple Event coercions follow the following convention:

resource type	'osax'
resource id	a unique identifier (e.g.. 3069)
resource name	'CSDSfromtoto' or 'CSPTfromtoto'

The 'CSDS' in the name above indicates that this is an Apple event coercion using the "from descriptor" interface and the alternate naming form 'CSPT' indicates that this is an Apple event coercion using the "coerce from pointer" interface. The next eight characters represent the from and to types.

The 'osax' code resource for OSAX coercions are in the form of AppleEvent coercion handlers so the entry point for the code resource must follow the Apple Event coercion function prototype for the particular coercion form as follows:

In C, the coerce from pointer coercion form ('CSPT')

```
pascal OSErr MyCoercePtr( DescType  fromType,
                          Ptr        dataPtr,
                          Size       dataSize,
                          DescType  toType,
                          long       theRefCon,
                          AEDesc     *theResult);
```

In Pascal, the coerce from pointer coercion form ('CSPT')

```
FUNCTION MyCoercePtr (  typeCode:  DescType;
                        dataPtr:   Ptr;
                        dataSize:  Size;
                        toType:    DescType;
                        refcon:    LongInt;
                        VAR addressDesc:  AEDesc): OSErr;
```

In C, the coerce from descriptor coercion form ('CSDS')

```
pascal OSErr MyCoerceDesc( AEDesc  theFromDesc,
                           DescType toType,
                           long     theRefCon,
                           AEDesc    *theResult);
```

In Pascal, the coerce from descriptor coercion form ('CSDS')

```
FUNCTION MyCoerceDesc ( theFromDesc: AEDesc;  
                        toType:      DescType;  
                        theRefCon:   LongInt;  
                        VAR addressDesc: AEDesc): OSErr;
```

Using other resources with OSAX

The OSAX loading mechanism adds the invoked OSAX 's resource file to the top of the target applications resource chain. For example, if we write the script:

```
tell application "Foobar"  
    beep 3  
end tell
```

The resource chain would be: Beep -> Foobar -> System (assuming Foobar hasn't added anything to the chain). This guarantees that any resources found in the OSAX's resource file will be found before any of the same name/id in the application or System resource files. In the case where an OSAX is called outside of a tell block, the resource chain would be as above without the application.

Creating aete's for replies as records

Some OSAX might need to return more than one piece of data in their replies. If we return a list, we can refer to the elements of the list by index. But, if we wish to return a record with named fields, we need to use a trick: while an 'aete' allows you to specify the type of a return value, it does not provide any additional information about it, such as the names of its fields if it is a record.

The trick we use to get around this limitation is to create a class that has properties for each of the fields in our record. We then declare the return type of the reply to be the id of this class.

In the 'aete' code snippet below, we define a class with the id 'hack'. This is the code we place in the return type field of our events reply. The property ids are the keywords for the records fields.

(For an example of this technique in action, see the reply returned from the display dialog OSAX included with AppleScript.)

```
{ /* array Classes: 1 elements */
```

```

/* [1] */
"my record names",
'hack',
"",
{
    /* array Properties: 2 elements */
    /* [1] */
    "button returned",
    'rone',
    ' hack',
    "",
    reserved,
    singleItem,
    notEnumerated,
    readOnly,
    reserved, reserved, reserved, reserved, reserved, reserved,
    reserved, reserved, reserved, reserved, reserved, reserved,
    /* [2] */
    "text returned",
    'rtwo',
    'hack',
    "",
    reserved,
    singleItem,
    notEnumerated,
    readOnly,
    reserved, reserved, reserved, reserved, reserved, reserved,
    reserved, reserved, reserved, reserved, reserved, reserved
},
},

```

Following is a simple OSAX to demonstrate the basic structure of an OSAX handler and its associated 'aete'. It is called "Flash OSAX" and is inspired by the demo XCMD "Flash" found in the HyperCard Script Language Guide. It is written in MPW C.

```

////////////////////////////////////
//
//                      Flash.c written by Donald Olson
//
//                      The Flash OSAX
//                      Copyright ©1992 Apple Computer Inc.
//                      All rights reserved
//
//          Based on the "Flash XCMD"
//
////////////////////////////////////

#include <Quickdraw.h>
#include <AppleEvents.h>

////////////////////////////////////
//
//  FlashEntry ()
//
//          Flash the port twice the number of requested times so that
//          we invert it *AND* restore it to its original state.
//
//  //////////////////////////////////////

pascal OSErr FlashEntry( AppleEvent *theAEEEvent,
                        AppleEvent *theReply,
                        long *theRefCon)

{
    OSErr          theErr = noErr;
    long           timesToFlash = 0;
    DescType       typeCode;
    Size           actualSize;
    GrafPtr        flashPort;

    GetPort(&flashPort);    // We'll flash the port rect of the
                          // current port.

```

```

/* Get the number of times to flash */

theErr = AEGetParamPtr( theAEEvent, keyDirectObject,
                        typeLongInteger, &typeCode,
                        (Ptr)&timesToFlash,
                        sizeof(timesToFlash), &actualSize);

if(theErr != noErr){
    // No parameters so just flash once.
    InvertRect(&flashPort->portRect);
    InvertRect(&flashPort->portRect);
}
else{
    // Flash the number of times asked
    if(timesToFlash < 1)
        timesToFlash = 3;    // default if negative
    while(timesToFlash--){
        // Flash until done.
        InvertRect(&flashPort->portRect);
        InvertRect(&flashPort->portRect);
    }
}

return noErr;
}

```

Here's the resource file for the above source code:

```

/*****

```

```

    Resource file for Flash.c
    Copyright ©1992 Apple Computer Inc.
    All rights reserved
    Written by Donald Olson

```

```

*****/

```

```
#include "Types.r"
#include "SysTypes.r"
#include "AEUserTermTypes.r"
```

```
resource 'vers' (1) {
    0x1,
    0x0,
    alpha,
    0x0,
    verUS,
    "1.0",
    "1.0, Copyright © 1992 Apple Comput"
    "er, Inc. All rights reserved."
};
```

```
resource 'vers' (2) {
    0x1,
    0x0,
    alpha,
    0x0,
    verUS,
    "1.0",
    "(by Donald Olson)"
};
```

```
resource 'aete' (0, "Flash OSAX") {
    0x0,
    -0x70,
    english,
    roman,
    { /* array Suites: 1 elements */
        /* [1] */
        "System Object Suite",
        "",
        'syso',
    /* System Object Suite */
        1,
        1,
        {
            /* array Events: 1 elements */
            /* [1] */
            "flash",
```

```

    "",
    'aevt',
    'flsh',
    noReply,
    "",
    replyOptional,
    singleItem,
    notEnumerated,
    reserved, reserved, reserved, reserved, reserved,
    reserved, reserved, reserved, reserved, reserved,
    reserved, reserved, reserved,
    'long',
    "Number of times to Flash",
    directParamOptional,
    singleItem,
    notEnumerated,
    doesntChangeState,
    reserved, reserved, reserved, reserved, reserved,
    reserved, reserved, reserved, reserved, reserved,
    reserved, reserved,
    {          /* array OtherParams: 0 elements */
    },
    {
        /* array Classes: 0 elements */
    },
    {
        /* array ComparisonOps: 0 elements */
    },
    {
        /* array Enumerations: 0 elements */
    }
};

```

And finally, the build commands:

```

C -b "Flash.c" -d SystemSevenOrLater
Rez -a -o "Flash OSAX" -t osax -c ascr 'Flash.r'
Link -p -w -t osax -c ascr -rt osax=1000 -m FLASHENTRY 0
-sg "AEVTaevtflsh" -ra "AEVTaevtflsh"=resSysHeap,resLocked 0
    "Flash.c.o" 0
    "{CLibraries}"StdCLib.o 0
    "{Libraries}"Runtime.o 0

```

```
"{Libraries}"Interface.o ␣  
-o "Flash OSAX"
```