Welcome

To Advance through Presentation
Use Page Up and Page Down Keys

99 | Worldwide Developers Conference

Extending AppleScript

Jason Yeo

AppleScript Technology Manager

99 Worldwide Developers Conference

Extending AppleScript

Christopher Nebel
AppleScript Engineering

Andy Bachorski
AppleScript Guru

What’s in This Session

- Native Scripting Additions
- Unit Types
- Attaching and Embedding Scripts

Scripting Additions

Surgeon General's Warning:

Scripting additions cannot implement the object model, execute inside other applications, and override application terminology. Improperly written additions can cause system instability and general consternation.

A Brief History

- 1.0–1.1.2: 68K code resources
- 1.3 (Mac OS 8.5): native shared library or accelerated code resources
- 1.3.7 (Mac OS 8.6): native shared library designed correctly!

Native Additions in 8.6

- Additions are just shared libraries with a terminology resource—no extra baggage
- AppleScript prepares and releases your addition’s code fragment
- No AppleScript overhead
- Added bonus—code sharing!

Initialization

- Install your handlers here!
- You can be smart: install different routines depending on context
- If you fail, undo everything you did
- Be careful about what you link to—you will be loaded at system startup time!

Initialization

```
static AEEventHandlerUPP myHandlerUPP;

OSErr MyFragInit(const CFragInitBlock *initBlock)
{
    myHandlerUPP = NewAEEventHandlerUPP(MyHandler);
    AEInstallEventHandler(‘blah’, ‘zoot’, myHandlerUPP,
        myRefcon, true);
    …any other initialization you need…
    if (err != noErr) {
        AERemoveEventHandler(‘blah’, ‘zoot’,
                myHandlerUPP, true);
    }
    return err;
}
```

Initialization—Linking

```
pascal OSErr CountVoices(short *numVoices)
{
    typedef OSErr (*CountVoicesPtr)(short *);

    static Boolean              attempted = false;
    static CountVoicesPtr fn = NULL;

    if (!attempted) {
        fn = (CountVoicesPtr) Bind("\pSpeechLib",
                "\pCountVoices");
        attempted = true;
    }
    return fn ? (*fn)(numVoices) : paramErr;
}
```

```
Ptr Bind(ConstStr63Param library, ConstStr255Param symbol)
{
    OSErr                   err;
    CFragConnectionID       connectionID;
    Ptr                     addr;

    // Must bind symbol in the system context!
    THz                     savedZone;
    savedZone = LMGetTheZone();
    SetZone(LMGetSysZone());

    err = GetSharedLibrary(library, kPowerPCCFragArch, kFindCFrag,
                &connectionID, NULL, NULL);

    /* If we couldn't find it, load it. */
    if (err == cfragNoLibraryErr || err == cfragLibConnErr)
        err = GetSharedLibrary(library, kPowerPCCFragArch,
                kLoadCFrag, &connectionID, NULL, NULL);
    …next slide, please…
```

Bind, continued…

```
    if (err == noErr) {
        CFragSymbolClass        symClass;

        FindSymbol(connectionID, symbolName, &addr, &symClass);
    }

    SetZone(savedZone);
    return addr;
}
```

Runtime

- Just one extra thing: keep a use count

```
UInt32 gAdditionReferenceCount = 0;

OSErr MyEventHandler(…)
{
    gAdditionReferenceCount++;

    …do your thing…

    gAdditionReferenceCount--;

    return err;
}
```

Termination

- Remove your handlers using AERemoveEventHandler or AERemoveCoercionHandler
- Pass your routine's UPP—uninstall only your handler, not somebody else's!

Termination

```
static AEEventHandlerUPP myHandlerUPP;

void MyFragTerm()
{
    AERemoveEventHandler('blah', 'zoot',
        myHandlerUPP, true);
}
```

Standing on Your Own

- Open your own resource fork

```
FSSpec myFSS;

OSErr MyFragInit(const CFragInitBlock *initBlock)
{
    myFSS = *initBlock->fragLocator.u.onDisk.fileSpec;

    …install handlers, etc…
}
```

Standing on Your Own

- Open your own resource fork

```
extern FSSpec myFSS;

pascal OSErr MyAppleEventHandler(...)
{
	SInt16		savedResFile, myRefNum;

	savedResFile = CurResFile()
	myRefNum = FSpOpenResFile(&myFSS, fsRdPerm);
	...do that voodoo that you do so well...
	CloseResFile(myRefNum);
	UseResFile(savedResFile);
	return result;
}
```

Standing on Your Own

- Check for remote events

```
Boolean IsRemoteEvent(const AppleEvent *theEvent)
{
    OSErr err; DescType typeCode;
    SInt16 eventSource; Size actualSize;

    err = AEGetAttributePtr(theEvent, keyEventSourceAttr,
                typeShortInteger, &typeCode, &eventSource,
                sizeof(eventSource), &actualSize);

    return (err == noErr &&
            eventSource == kAERemoteProcess);
}
```

Carbon

- Mac OS 8 additions can’t link to Carbon
    - Carbon not available at system startup
    - You can be called from non-Carbon apps
- Mac OS X additions must use Carbon
    - See Session 157:
      “AppleScript, Mac OS X, and Carbon”

Unit Types

- What They Are
- How They Can Be Used
- Adding New Unit Types

What Are Unit Types

- Real number values (doubles) with associated type information

What Are Unit Types

- Real number values (doubles) with associated type information
- Families of unit types with common base type

What Are Unit Types

- Real number values (doubles) with associated type information
- Families of unit types with common base type
- Coercions from one type to another within a family

Base Unit Types

- Meters
- Square meters
- Cubic meters
- Liters
- Kilograms
- Degrees Celsius

Unit Type Families

- Meters
- Metres
- Inches
- Feet
- Yards
- Miles
- Kilometres
- Kilometers
- Centimetres
- Centimeters

Using Unit Types

- Assigning unit values

```
set x to 10 as inches
    --> inches 10
```

Using Unit Types

- Assigning unit values

```
set x to 10 as inches
      --> inches 10
```

- Converting between unit types

```
set y to x as centimeters
       --> centimeters 25.4
```

Adding New Unit Types

- Extend an existing family
- Add a new family of types
- Usually implemented as scripting addition
- Can be installed from an application
    - Should be installed as system hander
    - Must be removed when app quits

Extending a Unit Family

- Define four coercion handlers for the unit to be added
  - typeWildCard to unitType
  - unitType to typeWildCard
  - unitType to baseType
  - baseType to unitType

Adding a New Unit Family

- Define two coercion handlers for the base unit type
  - typeWildCard to baseType
  - baseType to typeWildCard
- Define other unit types for the family

typeWildCard to unitType

```
pascal OSErr WildToType(…)
{
    switch (fromType)
    {   // For intrinsic types, first coerce to a double
        case typeInteger: case typeChar: case typeFloat:
            err = AECoercePtr(fromType, dataPtr, dataSize,
                typeFloat, result);
            // Then change to the unitType type
            if (err == noErr) result->descriptorType = toType;
        break;
        …
```

typeWildCard to unitType

```
// For other types, first try to coerce to base unit,
default:
	err = ConvertWildToBase(fromType, dataPtr, &baseDesc);
	// then to the unitType type
	if (err == noErr) {
		err = ConvertBaseToType(baseDesc, toType, result);
		AEDisposeDesc(&baseDesc);
	}
	if ( err != noErr ) err = errAECoercionFail;
	break;
}
```

Key Points

- Base types know nothing about derived types
- Coercing to types other than intrinsic types involve an intermediate coercion to the base type
- Handle coercion to typeObjectSpecifier so you can be displayed in the Result and Log windows

Attaching and Embedding Scripts

- Scripts menu

Attaching and Embedding Scripts

- Scripts menu
- Attachability: attaching scripts to existing commands

Attaching and Embedding Scripts

- Scripts menu
- Attachability: attaching scripts to existing commands
- Tinkerability: replace existing commands with scripts

Attaching and Embedding Scripts

- Scripts menu
- Attachability: attaching scripts to existing commands
- Tinkerability: replace existing commands with scripts
- Embedding: attach scripts to documents or other objects

Executing Scripts

- Open connection to scripting component
- Load and prepare the script for execution
- Execute the script
- Retrieve any results and deal with errors
- Save changes to the script

Open Connection to Scripting Component

- Generic or specific
- Resulting component instance used for all other OSA calls

Load and Prepare the Script

- Scripts typically stored as resources of type ‘scpt’
- Load the script resource
- Call OSALoad to prepare the script for execution

Execute the Script

- OSAExecute
- OSADoScript
- OSAExecuteEvent
- OSADoEvent

OSAExecute

- Simplest way to execute a script
- Executes a compiled script
- Call the script's run handler
- Returns a scriptID containing any results

OSADoScript

- Compiles and executes source text rather
- Call the script's run handler
- Returns textual representation of any results

OSAExecuteEvent

- Executes a compiled script
- Tries to handle the input Apple event
- Returns a scriptID containing any results

OSADoEvent

- Executes a compiled script
- Tries to handle the input Apple event
- Returns a reply event
- Most closely matches application supplied handler functionality

Other Methods

- OSALoadExecute
- OSACompileExecute

Retrieving Results

- Call OSADisplay to convert a scriptID to text
- Use returned text
- Extract directly from reply event

Saving Script Changes

- Execution can cause script context to change
- Call OSAGetScriptInfo to see if script has changed
- Call OSAStore to convert script to an AEDesc
- Replace original script resource with contents of the dataHandle

Other Consideration

- Call OSASetActiveProc to install an active function
- Call OSASetSendProc to install a send function
- Pass an idle function, and optionally a reply filter, in your send function

Key Points

- Allows customers to:
  - Add new functionality
  - Augment or change existing functionality
  - Automate repetitive tasks

Other Resources

- Inside Macintosh: Interapplication Communications
- AppleScript SDK: Sample code for working with Apple events and AppleScript, available in developer section of Apple ftp site

Roadmap

AppleScript Feedback Forum For interactive feedback and discussion with the team	Hall C **Thur., 4:00pm**
AppleScript Birds of a Feather For community-building with other developers	Hall C **Thur., 5:30pm**

AppleScript Kitchens

London, U.K.:
June 15 through 17, 1999

Cupertino, CA:
August 17 through 19, 1999

For more information:
Email Jason Yeo at jason@apple.com

Think different.™

Welcome

To Advance through Presentation
Use Page Up and Page Down Keys

99 | Worldwide Developers Conference