NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note TN1075

Some Techniques for Handling Variables in Apple Guide

CONTENTS

Apple Guide has many advantages over other help systems but also has several drawbacks. One is that it is not easy to keep track of what the user has already done. This becomes a particular problem if you are writing a tutorial. For example, your tutorial might have 10 different lessons which the user may need to complete over several days. How can the user be sure that he or she has gone back to the point they left the tutorial and not skipped stages? One way to do this is to store variables within an application.

This Technote also discusses using helper applications with Apple Guide.

The following enclosed code sample demonstrates several important techniques that can be used in the creation of a guide file. It demonstrates how to write a background-only helper application and how to use it to store and access variables which are used by the guide. In addition, it shows you how to add a coach mark handler. To download it, just click on the icon below:

Updated: [Oct 01 1996]

Storing Apple Guide Variables in an Application

Storing the variables within your application is often the easiest place to put them. You need to do several things:

- Create storage for the variables and initialize them
- Install Apple Event handlers

- Install custom context check handlers

The examples below just store an array of longs and initialize everything to 0 each time the application is run. For simplicity sake, the codes only going to set them to 0 or 1 and test whether they are 0 or non 0. An unregistered suite of Apple Events `'Vals'` is going to be used. This is for demonstration purposes only and you should use your own ID's. The Apple Guide accesses variables 1 to 3, however, as arrays in 'C' start from 0, indexes from the guide are mapped from 1-3 to 0-2.

Installing and Initializing

You need to create storage for variables and then initialize them. You may want to store and recall these from a preference file or clear them each time the application is run. Listing 1 contains storage for 3 variables and a context check reference. It also shows how to initializes them and install the handlers.

```
// Maximum number of values that can be stored by app
#define kMaxAGEntries 3

// Storage for values
long gAGValues[kMaxAGEntries];

// Storage for context check references
AGContextRefNum contextRefNum;

void ClearAllAGValues()
{
    long    count;
    for (count=0;count < kMaxAGEntries;count++)
        gAGValues[count] = 0;
}


OSErr InitAGVars()
{
    OSErr theErr = noErr;
    short count;

// Clear all the values stored by the application
    ClearAllAGValues();

// Install context check handler. MyContextCallBack will get called when a
// context check of 'Vals' is queried by the Guide

    theErr = AGInstallContextHandler( NewContextReplyProc(MyContextCallBack),
        'Vals', 0, &contextRefNum );
    if (theErr)
        DebugStr("\pGot an Err - AGInstallContextHandler ");

// set up the dispatch table for the Apple Events
    theErr = AEInstallEventHandler( 'Vals', 'Cler',
       NewAEEventHandlerProc(ClearAGValues), 0, false) ;

    theErr = AEInstallEventHandler( 'Vals', 'Clr1',
       NewAEEventHandlerProc(ClearAGValues1), 0, false) ;
    theErr = AEInstallEventHandler( 'Vals', 'Clr2',
       NewAEEventHandlerProc(ClearAGValues2), 0, false) ;
    theErr = AEInstallEventHandler( 'Vals', 'Clr3',
```

```
            NewAEEventHandlerProc(ClearAGValues3), 0, false) ;

        theErr = AEInstallEventHandler( 'Vals', 'Set1',
            NewAEEventHandlerProc(SetAGValues1), 0, false) ;
        theErr = AEInstallEventHandler( 'Vals', 'Set2',
            NewAEEventHandlerProc(SetAGValues2), 0, false) ;
        theErr = AEInstallEventHandler( 'Vals', 'Set3',
            NewAEEventHandlerProc(SetAGValues3), 0, false) ;

        return theErr;
    }
```

Listing 1. Initializing the guide variables and handlers.

Setting and Clearing Variables

Setting and clearing of variables is done by Apple Events. Each event has a specific ID and no passed parameters. You will probably need to make these obvious by their ID. For instance, to set and clear value 1, use the event ID's `'Set1'` and `'Clr1'` . The source for the handlers can be found in Listing 1.

```
pascal OSErr SetAGValues1(const AppleEvent *message, const AppleEvent
     *reply, long refcon)
{
    gAGValues[0] = 1;
    return( noErr );
}

pascal OSErr ClearAGValues1(const AppleEvent *message, const AppleEvent *reply,
     long refcon)
{
    gAGValues[0] = 0;
    return( noErr );
}
```

Listing 2. Setting and clearing a variable.

For ease, you may want to clear all the variables. This is done by event ID 'Cler'.

```
pascal OSErr ClearAGValues(const AppleEvent *message, const AppleEvent
     *reply, long refcon)
 {
    ClearAllAGValues();
    return( noErr );
}
```

Listing 3. Clearing all the variables.

Interrogating Variables

Checking the state of the variables is handled by Apple Guide context checks. There are mechanisms to pass data from a guide file into a context check; therefore, you only need one check which can be used to interrogate all the values. In this case, two

longs are passed in the pInputData parameter. The easiest way to access them is to create a structure.

```
struct PassedAGParamType
{
    long command;
    long index;
};

typedef struct PassedAGParamType PassedAGParamType,
    *PassedAGParamPtr, **PassedAGParamHdl;
```

Listing 4. Passed data.

If command contains a 0, then a check is made to see if the value is 0. If it contains a 1, then the check sees it if the value is non-0. Index is used to define which variable to check.

```
pascal OSErr MyContextCallBack( Ptr   pInputData,
                        Size  inputDataSize,
                        Ptr   *ppOutputData,
                        Size  *pOutputDataSize,
                        AGAppInfoHdl hAppInfo )
{
    OSErr   theErr = noErr;
    Boolean checkResult = false;

// Check the index is within range
    if (((PassedAGParamPtr)pInputData)->index <= kMaxAGEntries)

// Work out whether we need to return true or false
         if (((PassedAGParamPtr)pInputData)->command == 1)
           checkResult = gAGValues[ ((PassedAGParamPtr)pInputData)->index - 1 ] != 0;
        else
           checkResult = gAGValues[ ((PassedAGParamPtr)pInputData)->index - 1 ] == 0;

// Set up the return values ppOutputData and pOutputDataSize

    theErr = MySetContextResult ( checkResult, ppOutputData, pOutputDataSize );

    return theErr;
}
```

Listing 5. Context check.

Once the result of the check has been determined, MySetContextResult() is used to fill in the ppOutputData and pOutputDataSize. The result is returned in this way because it is part of a far wider mechanism. MySetContextResult creates a pointer and moves the value in result into to it. Next, it returns the newly created pointer and the size of the data within it. Apple Guide is responsible for deallocating this pointer.

```
OSErr MySetContextResult( Boolean result, Ptr *outMessage,
                    Size *outSize )
{
    Ptr   pOut;
    Size  theSize = sizeof ( Boolean );
    OSErr theErr = noErr;

    pOut = NewPtr (theSize);
    if ( pOut )
    {
        BlockMove( (void *)&result, pOut, theSize );
        *outSize = theSize;
        *outMessage = pOut;
    }
    else
        theErr = MemError();
    return theErr;
}
```

Listing 6. Setting the context check result.

Don't forget to clear up when you quit

Before your application quits, you should remove the context check handlers using `AGRemoveContextHandler`. You also may want to save the variables into a preference file.

```
void RemoveAGStuff()
{
    OSErr theErr;
    theErr = AGRemoveContextHandler( &contextRefNum );
}
```

Listing 7. Removing the context check.

Back to top

Making Use of These Calls

Once the necessary calls have been added to the application, it's time to make changes to the guide's source files as follows:

- Define events
- Define context checks
- Mark sequences that are completed
- Skip completed sequences

Defining the Events

To make use of the events defined in the application, you need to define the events in the guide.

```
<Define Event> "ClearValues", 'AgHp', 'Vals', 'Cler'
<Define Event> "SetFlag1",    'AgHp', 'Vals', 'Set1'
<Define Event> "ClearFlag1",  'AgHp', 'Vals', 'Clr1'
```

Listing 8. Defining the events.

ClearValues sends the Apple Event `'Cler'` of event class `'Vals'` to the application with the creator type `'AgHp'`. This is defined in the application to clear all variables to 0.

Defining the Context Checks

As previously mentioned, context checks have a mechanism for passing data. They are defined in the additional parameters of the <DCC> command. In this case, if we want to check that a variable has been set,we would use the command:

```
<DCC> "IsFlagSet", 'Vals', 'AgHp', LONG: 1, LONG
```

Listing 9. Defining the context checks.

`'Vals'` is the context check in the application with the identifier `'AgHp'`. The first additional parameter is a long with a default value of one. The second is a long which will need a value entered each time the check is used. These two parameters correspond to the data structure `PassedAGParamType` described above; therefore, to check to see if variable one has been set:

```
<If> (IsFlagSet(1))
```

If you want to be more specific, for example, a check to see if lesson one has been completed could be:

```
<DCC> "IsLesson1Done", 'Vals', 'AgHp', LONG: 1, LONG: 1
<If> (IsLesson1Done())
```

Note that because both parameters have defaults, you don't need to enter a parameter when the check is used.

Marking Sequences Completed

To mark a sequence completed, use an <On Panel Hide> command with a suitable event to set the variable.

```
<Define Panel> "Lesson 1 - last panel"
Panel text....
<On Panel Hide> SetFlag1()
<End Panel>
```

Listing 10. Marking a sequence complete.

Skipping Completed Sequences

To skip sequences, create a containing sequence that will take you through your tutorial. You'll probably want to put an intro and outro panel into the sequence. To decide whether to show a lesson sequence, use the <If> command as shown below. This will skip lessons that have already been completed.

```
<Define Sequence> "Tutorial"
  <Panel> "Tutorial Intro"
  <If> NOT Flag1Set()
    <Jump Sequence> "Lesson 1"
  <End If>
  <If> NOT Flag2Set()
    <Jump Sequence> "Lesson 2"
  <End If>
  <If> NOT Flag3Set()
    <Jump Sequence> "Lesson 3"
  <End If>
  <Panel> "Tutorial Outro"
<End Sequence>
```

Back to top

Helper Applications

It's not always possible for a guide author to have access to the applications source. Therefore, it is not always possible to add the features described above to an application. Also, some applications use Apple Events but are not necessarily scripted. Setting up these events may be very difficult or impossible from Apple Guide. In these cases, it's possible to write a small background-only application which can be used to store variables and from simple Apple Events create and trigger far more sophisticated events. This small application could be used to store variables and handle the code described above.

To trigger it, you need to add a compiled Apple Script to the application.

```
tell application "AGHelperApp"
    launch
end tell
```

Listing 11. Apple Script to trigger helper application.

When you compile the script, make sure the helper application is in the same folder as the script. Also, if you change the creator type of the helper, you will need to recompile the script.

You will probably need to start the helper from the first panel of the tutorial sequence. In the example below, the compiled script is called StartHelperApp.

```
<Define Panel> "Tutorial Intro"
Tutorial Intro text...
<On Panel Create> DoAppleScript( "StartHelperApp" )
<End Panel>
```

Listing 12. Starting the helper application from within the guide.

Back to top

Summary

Guide files and tutorials can seriously add to the usability of your product. Keeping track of how far the user has gone through your tutorial will help the user learn your product. The techniques in this Technote will help you to implement this. Use them!

Back to top

References

Apple Guide Complete, published by Addison-Wesley, is an essential volume if you are writing Apple Guide help systems.

AGVariable sample code is an example implementation using the techniques described in this Technote. It contains a 'C' file that can be added to your application however it based around a helper application.

Back to top

Downloadables

	Acrobat version of this Note (K).	Download
	Apple Guide Variables Code Sample Folder (305K)	Download

Back to top
