```
/*
 *
------------------------------------------------------------------------------------------
 *  main.c
 *  LocalizeString XFcn
 *
 *  Created by Bruce Martin on 3/26/06.
 *  Copyright Martin Solution 2006. All rights reserved.
 *
------------------------------------------------------------------------------------------
 */


/*
 *
------------------------------------------------------------------------------------------
 *  Includes
 *
------------------------------------------------------------------------------------------
 */

#include "commonExt_460.h"
#include "InternalsToolbox_460.h"


/*
 *
------------------------------------------------------------------------------------------
 *  Defines
 *
------------------------------------------------------------------------------------------
 */

#define kMinNumParams   1               // Fill in your minimum number of parameters
here
#define kMaxNumParams   3               // Fill in your maximum number of parameters
here
#define kVersionNumber  "2.0"           // Fill in your current version number here
#define kSyntaxMsg      "LocalizeString(<Key>[,<StringFileName>][,ContextualComment])"
#define kCopyrightMsg   "LocalizeString XFcn by Bruce Martin \2512006 Martin Solution"
```

```
/*
 *
------------------------------------------------------------------------------------------
--------------------
 *  Prototypes
 *
------------------------------------------------------------------------------------------
--------------------
 */

int                 main                (XCmdPtr);
void                Fail                (char *);
Boolean             FalseAlarm          (void);
void                SetResult           (char *);
void                doMyFunction        (void);
void                CFSetResult         (CFStringRef message);
CFStringRef         ParamToCFString     (Handle theParam);

/*
 *
------------------------------------------------------------------------------------------
--------------------
 *  Static and Global Variables
 *
------------------------------------------------------------------------------------------
--------------------
 */

XCmdPtr             par;                // Public pointer to parameter block
//XCmdPtr             msgConduit;
Boolean             fatalError;         // Had any fatal errors yet?


/*
 *
------------------------------------------------------------------------------------------
--------------------
 *  Script Entry Point
 *
------------------------------------------------------------------------------------------
--------------------
 */

int main(XCmdPtr xcmdPtr)
{
    par = xcmdPtr;
    fatalError = false;
    par->returnValue = 0;
  /*   CFStringRef expression;
```

```
    handle h;
    char buffer; */

    if (!FalseAlarm())                  // Deal with kSyntaxMsg, author, & bad
paramCounts
/*expression = CFSTR("get the environment");

    CFStringGetCString(expression,buffer,CFStringGetLength(expression)+1,kCFStringEncodi
ngMacRoman)
        h=EvalExpr(msgConduit,buffer);
        */
        doMyFunction();

    return(0);
}


/*
 *
-------------------------------------------------------------------------------------
--------------------
 *  Put Your Code Here
 *
-------------------------------------------------------------------------------------
--------------------
 */

void doMyFunction(void)
{
    //#define kSyntaxMsg        "Localize(<Key>,<StringFileName>[,<CommentTag>])"
    CFStringRef theKey,theFileName,theComment; //,theBundleStr;
    //CFBundleRef theBundle;
    char theResult,cStrComment;
    CFStringRef cfsResult;
    int paramHasContent[kMaxNumParams];
    int i;
    //Size len;

    //init my paramHasContent variable to all false
    for(i=0;i<kMaxNumParams;i++){
        paramHasContent[i]=0;
    }
    int doWithParams;
    //check the size
    for(i=0;i< par->paramCount;i++){
        if(GetHandleSize(par->params[i])>1){
            paramHasContent[i]=1;
        }
    }
```

```
    if((paramHasContent[1]) && (paramHasContent[2])){
        //Use all params
        doWithParams = 3;
    }else if((paramHasContent[1])&&(paramHasContent[2]==false)){
        //Has no comment
        doWithParams=2;
    }else if((paramHasContent[1]==false)&&(paramHasContent[2])){
        //theFileName=empty but has a comment
        doWithParams=1;
    }else{
        //just use the key
        doWithParams=0;
    }

    //char theComment;
    theKey=ParamToCFString(par->params[0]);
    //theKey=CFStringCreateWithBytes(kCFAllocatorDefault,(const UInt8*)*par->params[0],
GetHandleSize(par->params[0])-1, kCFStringEncodingMacRoman,false);

    switch(doWithParams){
    case 0:{
        cfsResult=CFCopyLocalizedString (theKey,NULL);

    if(CFStringGetCString(cfsResult,&theResult,CFStringGetLength(cfsResult)+1,kCFStringE
ncodingMacRoman))
        {
            SetResult(&theResult);
        }else{
            CFSetResult(CFSTR("Error\r"));
        }
        break;
    }
    case 1:{
        /*len = GetHandleSize(par->params[2]);
        char theComment[len];
        //BlockMoveData(*par->params[2], &theComment, len);
        strcpy(theComment,**par->params[2]); */
        theComment=ParamToCFString(par->params[2]);
        //theComment=CFStringCreateWithBytes(kCFAllocatorDefault,(const UInt8*)*par->par
ams[2], GetHandleSize(par->params[2])-1, kCFStringEncodingMacRoman,false);

    if(CFStringGetCString(theComment,&cStrComment,CFStringGetLength(theComment)+1,kCFStr
ingEncodingMacRoman))
        {
            cfsResult=CFCopyLocalizedString (theKey,theComment);

    if(CFStringGetCString(cfsResult,&theResult,CFStringGetLength(cfsResult)+1,kCFStringE
ncodingMacRoman))
```

```
            {
                SetResult(&theResult);
            }else{
                CFSetResult(CFSTR("Error\r"));
            }
        }else{
            CFSetResult(CFSTR("Error\r"));
        }
        break;
    }
    case 2:{
        theFileName=ParamToCFString(par->params[1]);
        //theFileName=CFStringCreateWithBytes(kCFAllocatorDefault,(const UInt8*)*par->pa
rams[1], GetHandleSize(par->params[1])-1, kCFStringEncodingMacRoman,false);
        cfsResult=CFCopyLocalizedStringFromTable(theKey, theFileName,NULL);

    if(CFStringGetCString(cfsResult,&theResult,CFStringGetLength(cfsResult)+1,kCFStringE
ncodingMacRoman))
        {
            SetResult(&theResult);
        }else{
            CFSetResult(CFSTR("Error\r"));
        }
        break;
    }
    case 3:{
        /*len = GetHandleSize(par->params[2]);
        char theComment[len];
        //BlockMoveData(*par->params[2], &theComment, len);
        strcpy(theComment,**par->params[2]);
        */
        theComment=ParamToCFString(par->params[2]);
        theFileName=ParamToCFString(par->params[1]);
        //theComment=CFStringCreateWithBytes(kCFAllocatorDefault,(const UInt8*)*par->par
ams[2], GetHandleSize(par->params[2])-1, kCFStringEncodingMacRoman,false);

    if(CFStringGetCString(theComment,&cStrComment,CFStringGetLength(theComment)+1,kCFStr
ingEncodingMacRoman))
        {
            //cfsResult=CFCopyLocalizedString (theKey,theComment);
            cfsResult=CFCopyLocalizedStringFromTable(theKey, theFileName,theComment);

    if(CFStringGetCString(cfsResult,&theResult,CFStringGetLength(cfsResult)+1,kCFStringE
ncodingMacRoman))
            {
                SetResult(&theResult);
            }else{
                CFSetResult(CFSTR("Error\r"));
            }
```

```
        }else{
            CFSetResult(CFSTR("Error\r"));
        }
        break;
    }
    }


}



/*
 *
----------------------------------------------------------------------------------------------------
 *  Boilerplate Utilities
 *
----------------------------------------------------------------------------------------------------
 */

/************************************** Fail() ************************************/
// Load up par->returnValue with "false" and an error message
void Fail(char *message)
{
    Handle  h;
    fatalError = true;
    if (par->returnValue) DisposeHandle(par->returnValue);
    if (!*message) message = kSyntaxMsg;
    par->returnValue = (PtrToHand("false\r", &h, 6) ||
                        PtrAndHand(message, h, strlen(message)+1)) ? nil : h;
}

/************************************ ParamToCFString() ***************************/
// Convert a param to a CFStringRef

CFStringRef ParamToCFString(Handle theParam)
{
    CFStringRef theString;
    theString=CFStringCreateWithBytes(kCFAllocatorDefault,(const UInt8*)*theParam,
GetHandleSize(theParam)-1, kCFStringEncodingMacRoman,false);
    return theString;
}

/************************************ SetResult() *********************************/
// Load up par->returnValue with a string
void SetResult(char *message)
{
```

```
    PtrToHand(message, &par->returnValue, strlen(message) + 1);
}

void CFSetResult(CFStringRef message)
{
    //char theMsg;
    char theMsg[CFStringGetLength(message)+1];
    if(CFStringGetCString(message,theMsg,CFStringGetLength(message)+1,kCFStringEncodingM
acRoman))
    {
        SetResult(theMsg);
    }else{
        SetResult("Error!\rThere was an error converting a CFString to a C string.");
    }
}

/************************************ FalseAlarm() *******************************/
// Handle requests for kSyntaxMsg or kCopyrightMsg & bad paramCounts
Boolean FalseAlarm(void)
{
    Boolean result = false;
    short   pCnt = par->paramCount;

    if(pCnt == 1) {
        if ((result = !strcmp((char*)*par->params[0], "!")))
            SetResult(kCopyrightMsg);
        else if ((result = !strcmp ((char*)*par->params[0], "?")))
            SetResult(kSyntaxMsg);
        else if ((result = !strcmp ((char*)*par->params[0], "??")))
            SetResult(kVersionNumber);
        else if ((result = !strcmp ((char*)*par->params[0], "???")))
            SetResult(__DATE__"  "__TIME__);
    }
    if (!result && (result = ((pCnt < kMinNumParams) || (pCnt > kMaxNumParams))))
        Fail(kSyntaxMsg);

    return(result);
}
```