CLARIS®

Technical Note

Apple Events in Claris® Resolve™ 1.0

See also:	How to Write a Claris Resolve™ External Function
Written by:	Jon Thatcher
Date:	September 13, 1991

This document describes the Apple Events which Claris Resolve 1.0 responds to, including the details of events specific to Resolve.

Resolve supports inter-application communication through Apple Events, a feature of Apple's System 7 software. Resolve responds to standard Apple Events which allow opening and printing documents, as well as custom Apple Events which allow getting data from and putting data into worksheet cells.

This document lists the standard Apple Events supported by Resolve 1.0, and describes the Resolve-specific Apple Events and how to use them (including MPW C source code).

The reader of this document should be familiar with Chapter 6, The Apple Event Manager, of Inside Macintosh Volume VI, and should definitely have read the section "Introduction to Apple Events" within that chapter.

Summary of Apple Events in Resolve

Apple Events are grouped into different functional classes. There may be one or many events with similar functions which are grouped into the same class, so each event within a class has a separate identifier. The following table summarizes the Apple Events which are supported by Claris Resolve.

Resolve 1.0 only *responds* to the Apple Events in this list. It includes no built-in support for *sending* Apple Events, nor does it offer a built-in mechanism for responding to *generic* Apple Events (as HyperCard 2.1 does). These features may be offered in a future version of Resolve.

Required ("core") Apple Events

Event Class	Event ID	Description
kCoreEventClass	kAEOpenApplication	Open the application
kCoreEventClass	kAEOpenDocuments	Open the document(s) specified
kCoreEventClass	kAEPrintDocuments	Print the document(s) specified
kCoreEventClass	kAEQuitApplication	Quit the application

Publish/Subscribe Apple Events

Event Class	Event ID	Description
sectionEventMsgClass	sectionReadMsgID	Read the specified edition
sectionEventMsgClass	sectionWriteMsgID	Write the specified edition
sectionEventMsgClass	sectionScrollMsgID	Scroll to the specified edition

Miscellaneous Apple Events

Event Class	Event ID	Description
kAEMiscStdSuite	kAEDoScript	Execute the specified text as a script

Claris-specific Apple Events

Event Class	Event ID	Description
kCoreEventClass	CLAECloseActiveDoc	Close the current document
kCoreEventClass	CLAESaveActiveDoc	Save the current document
kClarisEventClass	CLAEGetValue	Get data from the current worksheet
kClarisEventClass	CLAEPutValue	Put data into the current worksheet
kClarisEventClass	CLAEScript	Execute a Resolve Script file or function

The Required and Publish/Subscribe Apple Events are described in Inside Macintosh Volume VI. The Miscellaneous Apple Events are described in the Apple Event Registry. The identifiers used for event classes and IDs in the lists above may be found in the documents mentioned, or for Claris-specific events, later in this document.

The kAEDoScript Apple Event is documented as accepting either a text string (typeText) or an alias as its direct object. When used with Resolve 1.0, the text string must be less than 255 characters in size, or an error will be returned. If an alias is specified, the file it refers to should be a Resolve Script file (compiled or text-only).

Claris-specific Apple Events in Resolve

The first two Claris-specific events in the last summary list above are extensions to the class of “core” Apple Events. They are intended for use *only* with a document opened using the kAEOpenDocuments Apple Event. The CLAESaveActiveDoc Apple Event saves any changes made to the current document. The CLAECloseActiveDoc Apple Event closes the current document and all of its windows.

The other three Claris-specific events in the list above form their own class, kClarisEventClass. These events are specific to certain Claris applications, and will almost certainly not be improved in future versions of Resolve. Instead, the Apple Object Support Library will be used in future versions of Resolve to implement more of the standard classes of events described in the Apple Event Registry.

The Claris-specific Apple Events of most interest in Resolve are CLAEGetValue, CLAEPutValue, and CLAEScript. The CLAEGetValue (GVAL) event is used to get a range of data or an object from the current Resolve worksheet and return it to the requesting application. The CLAEPutValue (PVAL) event copies specified data into a range within the current Resolve worksheet. Finally, the CLAEScript (SCPT) event allows a Resolve Script file or function to be executed.

The rest of this document describes these three Apple Events in detail.

The SCPT Event

The SCPT event allows a Resolve Script function or file to be executed by the sending program. Similar functionality is also provided by the standard kAEDoScript event in the miscellaneous event class, which Resolve also supports.

There are two parameters which may be supplied with the SCPT event, indicated with the keywords keyScpFile (FILE) and keyScpFunc (FUNC). One or both parameters may be supplied.

Parameters of SCPT event:

Keyword	Data Type	Description
keyScpFile	typeAlias	optional, alias of Resolve Script file
keyScpFunc	typeChar	optional, name of Resolve Script function

If the FILE and FUNC parameters are both supplied, Resolve will load the script file specified with the FILE parameter, then call the function named by the FUNC parameter. If only the FILE parameter is supplied, Resolve will simply run the entire Resolve Script file specified. Finally, if only the FUNC parameter is supplied, Resolve will attempt to call the specified function, which must exist within a previously loaded Resolve Script file.

If the Resolve Script successfully executes, only the noErr status code will be returned in the SCPT event reply. If an error does occur during execution of the script, the status code CLAEerrCantComplete will be returned, possibly with a text string which may help identify the error. Other error codes may be returned if the event itself has a problem (e.g., if the file specified does not exist).

The GVAL and PVAL Events

The PVAL event is used to put data into a cell range of the current Resolve worksheet, while the GVAL event is used to get data or a graphic object from the current Resolve worksheet. The PVAL and GVAL events use a similar parameter structure for specifying the data to be put into or gotten from the worksheet. The PVAL and GVAL parameters are placed within an Apple Event record (typeAERecord), which is itself placed in the direct object parameter of the event. There is no way of specifying the target worksheet; the **current** worksheet is **always** used with PVAL and GVAL.

For a GVAL event which requests data from a cell range, the AERecord will be of type keyTABL and will contain three parameters. The reply to this event will be of similar structure, except that the keyRetType (return type) parameter will be replaced with the returned data. The returned data is in the same format as TEXT on the clipboard: tabs are used in the text to delimit cells in the same row, and carriage returns delimit the rows.

Contents of keyTABL record for GVAL event:

Keyword	Data Type	Description
keyTopLeft	typeInteger	specifier of top left cell of range
keyBotRight	typeInteger	specifier of bottom right cell of range
keyRetType	typeEnum	return type (must be typeChar)

Contents of keyTABL record for GVAL reply (and for PVAL event):

Keyword	Data Type	Description
keyTopLeft	typeInteger	specifier of top left cell of range
keyBotRight	typeInteger	specifier of bottom right cell of range
keyRetValue	typeChar	tab-delimited text being returned

A PVAL event only allows data to be put into cells, so a PVAL event is essentially the inverse of a GVAL event with a keyTABL record. Thus, it's not surprising that the parameters specified with a PVAL event are placed in a keyTABL record, with exactly the same keywords and format as the GVAL reply shown above.

The keyTopLeft and keyBotRight parameters used in the keyTABL record shown above are similar in structure to the standard QuickDraw Point data type. The keyTopLeft parameter is a long integer value which contains the top coordinate and the left coordinate as two separate short (16-bit) integers. Similarly, the keyBotRight parameter contains the bottom and right coordinates. The coordinates use a zero origin, so cell address A1 is represented by zeros for the top and left coordinates. Thus, the cell range A3..B7 is represented with the coordinates top=2, left=0, bottom=6, right=1.

A GVAL event may also be used to get a graphic object from the current Resolve worksheet. In this case, an AERecord of type keyOBJ is specified as a parameter to the event. The object to be gotten is specified by using the number that Resolve assigns to the object upon its creation. The object number is also returned by the Number() Resolve Script function. The object is returned in the PICT format in the reply to the GVAL event.

Contents of keyOBJ record for GVAL event:

Keyword	Data Type	Description
keyObjNum	typeInteger	object number in the Resolve worksheet
keyRetType	typeEnum	return type (must be typePict)
keyRetType	typeEnum	return type (must be typePict)

Contents of keyOBJ record for GVAL reply:

Keyword	Data Type	Description
keyObjNum	typeInteger	object number in the Resolve worksheet
keyRetValue	typePict	PICT-format data being returned

The next section of this document contains an example of creating a PVAL event. Don't fret if you couldn't figure out these events from the descriptions above. That would be a miraculous feat.

Resolve Apple Events example code

Following is an MPW C source code example of creating a PVAL Apple Event. The example is in the form of a Resolve external tool. This allows the two functions SetAddress and PutValue to be called using Resolve Script. Thus, a worksheet could be created which has push buttons to execute the two functions.

SetAddress simply calls the toolbox PPCBrowser function to allow the user to select the target of the Apple Event. Once the target is selected, the PutValue function may be called with the string to put, and the coordinates of the target range. For example, the Resolve Script command

call PutValue("foo"&char(9)&"bar",11,1,11,2)

could be used to place “foo” into cell B12 and “bar” into cell C12.

PVAL source code example

```
/* This example was tested using MPW 3.1 C Include files with System 7
*      updates.  The latest MPW 3.2 Include files should be compatible.
*/

#include <AppleEvents.h>
#include <DatabaseAccess.h>
#include <Processes.h>

#define numfuncs 2

#include "ResolveTools.h"

#define  keyTABL       'TABL'
#define  keyOBJ        'OBJ '
#define  keyTopLeft    'TLPT'
#define  keyBotRight   'BRPT'
#define  keyRetType    'RTRN'
#define  keyObjNum     'OBJ#'
#define  keyRetValue   'VAL '

#define  keyScpFile    'FILE'
#define  keyScpFunc    'FUNC'

#define  kClarisEventClass          'CLRS'
#define  CLAEGetValue               'GVAL'
#define  CLAEPutValue               'PVAL'
#define  CLAEScript                 'SCPT'
```

```
/* union for specifying keyTopLeft and keyBotRight values */
typedef union
        {
        struct {
            short       vert;
            short       horz;
            } p;

        long        coord;
        } COORD;

ROUT                        rout;
char                        SetAddressName[] = "\pSetAddress";
char                        PutValueName[] = "\pPutValue";

Boolean                     goodTarget;
AEAddressDesc               myTarget;



void SetAddress(PVAL pret, PVAL parg)
{
OSErr               reterr;
PortInfoRec     portInfo;
TargetID        target;

#pragma unused (pret, parg)

if ((reterr = PPCBrowser(nil, nil, false, &target.location,
                                    &portInfo, nil, nil)) == noErr)
    {
    target.name = portInfo.name;

    myTarget.descriptorType = typeTargetID;
    reterr = PtrToHand((Ptr) &target, &myTarget.dataHandle, sizeof(target));
    }

goodTarget = (reterr == noErr);
}



void PutValue(PVAL pret, PVAL parg)
{
OSErr               reterr,  ret2;
COORD               topLeft, botRight;
AppleEvent      message, reply;
AEDescList      tablRec, reqList;
short               gotmess, gotrepl, gottabl, gotlist;

#pragma unused (pret)

if (goodTarget && parg[0].flag == STRING &&
        parg[1].flag == NUMERIC && parg[2].flag == NUMERIC &&
        parg[3].flag == NUMERIC && parg[4].flag == NUMERIC)
    {
    gotmess = gotrepl = gottabl = gotlist = false;

    while (1)
        {
        /*
```

```
 * create the AERecord to contain keyTABL parameters, then put them in
 */
if ((reterr = AECreateList(nil, 0, true, &tablRec)) != noErr)
      break;
gottabl = true;

topLeft.p.vert = (short) parg[1].val.numeric;
topLeft.p.horz = (short) parg[2].val.numeric;
if ((reterr = AEPutKeyPtr(tablRec, keyTopLeft, typeLongInteger,
                                          (Ptr) &topLeft, sizeof(topLeft))) != noErr)
      break;

botRight.p.vert = (short) parg[3].val.numeric;
botRight.p.horz = (short) parg[4].val.numeric;
if ((reterr = AEPutKeyPtr(tablRec, keyBotRight, typeLongInteger,
                                          (Ptr) &botRight, sizeof(botRight))) != noErr)
      break;

if ((reterr = AEPutKeyPtr(tablRec, keyRetValue, typeChar,
                                          &parg[0].val.string[1],
                                          parg[0].val.string[0])) != noErr)
      break;

/*
 * create the AERecord which will be the direct object,
 *  then put the keyTABL AERecord into it
 */

if ((reterr = AECreateList(nil, 0, true, &reqList)) != noErr)
      break;
gotlist = true;

if ((reterr = AEPutKeyDesc(reqList, keyTABL, tablRec)) != noErr)
      break;

if ((reterr = AEDisposeDesc(&tablRec)) != noErr)
      break;
gottabl = false;
```

```
            /*
         * create the Apple Event message, then put the direct object into it
         */

        if ((reterr = AECreateAppleEvent(kClarisEventClass, CLAEPutValue,
                                                        myTarget, 0, 0, &message)) != noErr)
            break;
        gotmess = true;

        if ((reterr = AEPutParamDesc(message, keyDirectObject, reqList))
                != noErr)
            break;

        if ((reterr = AEDisposeDesc(&reqList)) != noErr)
            break;
        gotlist = false;

        /*
         * send the Apple Event, but NEVER wait for the reply
         *      (since Resolve is currently recalculating)
         */

        reply.dataHandle = nil;
        if ((reterr = AESend(message, &reply, kAENoReply, kAENormalPriority,
                                            0, nil, nil)) != noErr)
            break;
        gotrepl = true;

        break;

        }       /* end of while (1) */

    /*
     * dispose of any and all remaining AE records
     */

    if (gotmess)
        {
        ret2 = AEDisposeDesc(&message);
        if (reterr == noErr)
            reterr = ret2;
        }

    if (gotrepl)
        {
        ret2 = AEDisposeDesc(&reply);
        if (reterr == noErr)
            reterr = ret2;
        }
```

```
	if (gottabl)
		{
		ret2 = AEDisposeDesc(&tablRec);
		if (reterr == noErr)
			reterr = ret2;
		}

	if (gotlist)
		{
		ret2 = AEDisposeDesc(&reqList);
		if (reterr == noErr)
			reterr = ret2;
		}

	}		/* end of if (goodTarget...) */
}


PROUT main()
{
	rout.nrout = numfuncs;
	rout.exitfunc = nil;

	rout.relts[0].pfunc = SetAddress;
	rout.relts[0].name = SetAddressName;
	rout.relts[0].narg = 0;

	rout.relts[1].pfunc = PutValue;
	rout.relts[1].name = PutValueName;
	rout.relts[1].narg = 5;

	goodTarget = false;
	InitGraf(&qd.thePort);

	SysBeep(11);
	return(&rout);
}
```

GVAL source code fragment

Following is an MPW C source code fragment of handling the reply from a GVAL Apple Event. This code can't be used as part of the preceding Resolve external example since it requires an Apple Event reply.

```
while (1)
    {

    /* code left out here would prepare the Apple Event and call AESend */

    /*
     * handle the reply if no error occured, or try to get the error string
     */

    reterr = AEGetParamPtr(reply, keyErrorNumber, typeLongInteger, &theType,
                                        (Ptr) &errNumber, sizeof(errNumber), &theSize);

    if ((reterr == noErr && errNumber == noErr)
          || reterr == errAEDescNotFound)
        {
        if ((reterr = AEGetParamDesc(reply, keyDirectObject, typeAERecord,
                                                    &reqList)) != noErr)
            break;
        gotlist = true;

        if ((reterr = AEGetKeyDesc(reqList, keyTABL, typeAERecord,
                                                &tablRec)) != noErr)
            break;
        gottabl = true;

        if ((reterr = AEGetParamPtr(tablRec, keyRetValue, typeChar, &theType,
                                                    (Ptr) theString, sizeof(theString) - 1,
                                                    &theSize)) != noErr)
            break;
        }

    else
        {
        if ((reterr = AEGetParamPtr(reply, keyErrorString, typeChar,
                                                    &theType,
                                                    (Ptr) theString, sizeof(theString) - 1,
                                                    &theSize)) != noErr)
            break;
        }

    theString[theSize] = '\0';

    /* code left out here would return or display 'theString' */
```