CLARIS®

Technical Note

How to Write a Resolve™ External Function

Written by: Ken Walter
Date: September 13, 1991

This document describes the creation of a Claris Resolve External Function in C.

Resolve includes a large number of built-in functions that you can use in worksheets and scripts. You can also create custom functions using Resolve Script. If additional functionality is required, external functions can be created using a compilable programming language such as C. These functions can be installed in Resolve external tools files and also in scripts and worksheets.

This document describes how you can create such functions in MPW C and Think C.

Using External Functions in Resolve

– Load the external using the GET EXTERNAL command.

  • The GET EXTERNAL command also makes the resources of the file containing the external available for use by the external or other Resolve Script commands.

```
GET EXTERNAL "HardDisk:Externals:MyExternals"
```

  • Note that if the external is installed in a worksheet or script, you still need to load it using the GET EXTERNAL command.

– Once it is loaded, use the external just as you would a custom function, either as a command function or computational function.

  • Once an external is installed, you refer to it by file name only, not full path name.

  • For external command functions, use the CALL keyword.

  • The following example shows a command function DialPhone() with an argument that is a string representing a phone number.

```
CALL MyExternals:DialPhone("(408)555-1212")
```

  • You can use computational external functions wherever an expression could be used, as long as it returns an appropriate value type. The following examples use an imaginary external function "Foo" which takes two numeric arguments.

Assign the value returned to a variable.

```
DEFINE x, y, z
y = 1
z = 1
x = MyExternals:Foo(y, z)
```

Store the return value in a cell.

```
= MyExternals:Foo(A1, A2)
```

Display the returned value in a message.

```
MESSAGE MyExternals:Foo(y, z)
```

– Remove the external using the REMOVE EXTERNAL command

```
REMOVE EXTERNAL "MyExternals"
```

Creating an External Function in C

– Although it is possible to create an external tool in many compiled programming languages, this section discusses writing an external tool in MPW C and Think C.

– In MPW C, creating an external tool in C is similar to writing any C program.

– In Think C, you will have to build a code resource with a custom header for your external to work properly.

Header

– In the file containing your function main, define the macro numfuncs to be the number of external functions in your program.

```
#define numfuncs 2
```

– If you are in Think C, define the macro ThinkC

```
#define ThinkC
```

– In MPW, include ResolveTools.h. ResolveTools.h is included on the "Resolve Samples" disk.

– In Think C, you must include ThinkCGlue.h, which should be customized for your program as outlined in the section Building in Think C below. ThinkCGlue.h is included on the "Resolve Samples" disk. ThinkCGlue.h will include ResolveTools.h, so you do not need a separate include statement.

Functions

You will need to write a main function, a function for each external Resolve Script function you wish to support, and an optional cleanup function.

Main

– The function main is declared as follows.

```
PROUT main()
```

– Your main function will need to use two structures declared in ResolveTools.h.

– The routine list (ROUT) structure is used to pass information to Resolve concerning the location and number of external functions in your program.

```
typedef struct _ROUT
{
short               nrout;
#ifdef ThinkC
ProcPtr  exitfunc;
#else
void                (*exitfunc)();
#endif
RELT                relts[numfuncs];
} ROUT, *PROUT;
```

• A pointer to a routine list structure will be returned back by your main function when the command GET EXTERNAL is executed.

• The nrout field should contain the number of externals in your program (should be the same as the macro numfunc).

• The exitfunc field is a pointer to an optional routine to “clean up” (free all memory allocated, close files, etc.) when a tool is removed with the REMOVE EXTERNAL command . This function is of type void and has no parameters.

• The relts field is an array of numfuncs routine list elements (RELT’s), defined below.

– Each routine list element contains information about an external function supported in your program.

```
typedef struct _RELT
{
#ifdef ThinkC
ProcPtr  pfunc;
#else
void                (*pfunc)(PVAL, PVAL);
#endif
char                *name;
short               narg;
} RELT, *PRELT;
```

• The pfunc field is a pointer to the function in your program supporting this external.

• The name field is the name of the routine (Pascal string). All names should be allocated as globals or in non-relocatable blocks on the heap. This is the name the user will use to call the routine in a script or formula.

• The narg field is number of arguments to your routine.

• When a function is called in a script or worksheet, Resolve will automatically check for the correct number of arguments, and set an appropriate error result if necessary.

– Your main function is called by Resolve whenever the GET EXTERNAL command is called:

• The function main initializes a routine list which must be allocated as a global or as a non-relocatable block on the heap, and does any other initialization, allocation, or opening of drivers required by your external functions.

• If you are going to draw anything with your external routines (including windows), you must call the Macintosh toolbox routine InitGraf in the function main.

• For access to globals in ThinkC, you will need to set up and restore A4 at the beginning and end of your function.

```
PROUT main()
{
```

```
static ROUT gRout;          /* must be a global or in a
                                    * non-relocatable block on heap
                                     */
PROUT           returnPRout;

#ifdef ThinkC
RememberA0();           /* Set up access to globals */
SetUpA4();
InitGraf(&thePort);     /* must be called if drawing */
#else
InitGraf();             /* must be called if drawing */
#endif
/* initialize our routine list */
gRout.nrout = numfuncs;
gRout.exitfunc = CleanUp;
gRout.relts[0].pfunc = Foo;
gRout.relts[0].name = "\pFoo";
gRout.relts[0].narg = 2;
gRout.relts[1].pfunc = Bar;
gRout.relts[1].name = "\pBar";
gRout.relts[1].narg = 3;

/* read address of our routine list into a local
 * variable before calling RestoreA4 in Think C,
 * which will trash globals
  */
returnPRout = &gRout;
#ifdef ThinkC
RestoreA4();            /* Restore previous A4 value */
#endif
/* return pointer to routine list to resolve */
return returnPRout;
}
```

- **Note**: In the Think C example shipped with Resolve 1.0, assembly glue was used to return the routine list. Not only is this unnecessary, but *externals using this method may not be supported in future versions of Resolve*.

External Functions

– In the main function, each external function has a pointer to a C function assigned to it in its corresponding routine list element.

– All such functions use the value (VAL) structure declared in ResolveTools.h.

```
typedef struct _val
{
union
{
#ifdef ThinkC
        short double    numeric;
#else
        double          numeric;
#endif
        char                *string;
        short                   err;
        unsigned long   private[3];
} val;

short           flag;
```

```
} VAL, *PVAL;
```

• The flag field tells the type of the value in the val field.

The following macros are defined in ResolveTools.h for use in this field:

```
#define NUMERIC         0
#define STRING      1
#define ERR             2
```

• The val field is a union.

The numeric field is used if the value to be returned is numeric. Note that its type is double (64 bits), not extended (80 or 96 bits, depending on whether 68881 code is used).

The string field is a pointer to a Pascal string.

The error field is a short corresponding to the Resolve Script error message numbers in Appendix E of the *Claris Resolve Functions and Scripts* manual.

Do not use the private field.

– All external functions are of type void and have two arguments, like the example declaration below.

```
void MyExternal(PVAL pReturn, PVAL pArgs);
```

– The second argument is a pointer to an array. This array is used to pass the Resolve Script arguments to the external function. The size of the array was passed to Resolve by the main function in the narg field of the routine list element.

• The Resolve Script arguments are passed from left to right, so pArgs[0] corresponds to the leftmost argument, pArgs[narg - 1] to the rightmost.

• Your function should check the flag field of each value structure in the array to make sure the user has passed the proper type of argument, and return an appropriate error if necessary.

– The first argument is a pointer to the value structure that you will set up and pass back to Resolve. This determines the return value of your declared in ResolveTools.h.

• Make sure to set the flag field of your return structure appropriately, using the macros defined in ResolveTools.h.

• If a pointer to a string is returned, the string must be allocated as a global or a non-relocatable block on the heap.

– For access to globals in ThinkC, you will need to set up and restore A4 at the beginning and end of your function.

Exit Function

– The exit function is called when the REMOVE EXTERNAL command is executed. For functions installed in a worksheet or script, it is also called whenever the file is saved.

– The exit function should deallocate anything allocated by your program.

– For access to globals in ThinkC, you will need to set up and restore A4 at the beginning and end of your function.

Building in MPW

– Use the C command to compile your program as you would any other program.

– Your Link command should set the type of your external tool to 'RsTl' and the creator to 'Rslv'.

Building in Think C

– Set the type of your program to 'RsTl' and the creator to 'Rslv'.

– Set the project type to be a code resource with type CODE and ID 0.

– Because of the way Think C initializes its global space, you must build a code resource instead of an application. Since Resolve launches the external as an application, you must make the code resource look and act like an application, by creating a jump table at the beginning of your code resource.

– To better understand the steps below, read the discussion of "The Jump Table" in *Inside Mac II*, pp. 60-62.

• You need to customize the header function in ThinkCGlue.h for your code resource. This becomes the header of your CODE 0 resource.

The example below is a commented version of ThinkCGlue.h. You should make the necessary changes in the ThinkCGlue.h file on the Resolve Samples disk, which has the wrong declaration for the main function.

Note that you must modify ResolveTools.h whenever you make changes to your program.

```
#include "ResolveTools.h"
/* This is the code header.  The sizes below must be
* adjusted whenever any code is added
* or taken away.  Build and use ResEdit to find the
* size of your CODE resource, then rebuild.
*/
extern PROUT main();

header()
{
asm
{
valid_A4:
        DC.L    7032    ; "Above A5 size" (CODE size+32)
        DC.L    0       ; "Below A5" size, always 0 since
                        ; we have no A5 globals
        DC.L    7000    ; your CODE size
        DC.L    32      ; Offset to jump table from location
        ; pointed to by A5 (Always 32)
        DC.W    0       ; Always 0
        LEA             @valid_A4,a0
        JMP             main
}
}

#include "SetUpA4.h"
```

Installing in Scripts or Worksheets

– Using ResEdit or Resorcerer or other resource editing application, move the resources of your tool into the resource fork of the script or worksheet.

– Use GET EXTERNAL and REMOVE EXTERNAL to load and unload your external program and call the external in the usual way.

– *See warnings about globals below*.

Resources

– The GET EXTERNAL command will also make all resources in a file available for use by the external and other Resolve Script commands that use resources (sound commands, etc.).

– Resolve always calls UseResFile with the refnum of your external before calling one of your functions. If you open another resource file, you will have to call UseResFile in each function that uses it to make it the current resource file.

– *You must be careful to avoid using the same resource ID as any resource used in Resolve or Resolve Scripts.*

– *FONT resources will not be displayed in font menu.*

Globals

– When your external is installed from a Resolve external tool, globals will retain their values on multiple calls to your routines until REMOVE EXTERNAL is called. *This is not true for worksheets and scripts.*

– *When a worksheet or script is saved, any externals installed are removed and reinstalled. At this time, all global data will be destroyed and set back to its initial state. For this reason, do not depend on globals keeping their values in external programs installed in worksheets. Do not even depend on opening the same resource file.*

– In Think C, all functions that use globals should begin with the statement:

```
SetUpA4()
```

– You cannot access any Resolve globals or functions.

– Likewise, Resolve cannot access your global data or functions, except through the routine list returned by your main function and the value structure returned by external functions. For this reason, do not install completion routines or vbl tasks that continue after your function returns.
– In Think C, all functions that use globals should have the statement below before returning:

```
RestoreA4()
```

Other Limitations

– Strings returned from routines must be allocated as global variables or as non-relocatable blocks on the heap.

– You can safely use up to 2K of stack space. If you need more memory, allocate objects on the heap using the Macintosh Memory Manager.

– In MPW, you may have up to 15 code segments in your program. Anything more may cause collisions with swappable Resolve code segments and may cause Resolve to hang or crash.

– In Think C, you are limited to a single code segment, which must have an ID of 0.

Examples

– The following is a simple external program that will work in Think C or MPW:

```
/*
        FILE:           ExternalSample.c

        DESCRIPTION:    Simple Resolve™ External Function Example in MPW C

        LEGAL:          Copyright © 1990 by Claris Corporation.
                        All Rights Reserved.

        REMARKS:        In MPW, build using the following commands:
                                C  MyExternal.c
                                Link -t RsTl -c Rslv ∂
                                        MyExternal.c.o ∂
                                        "{CLibraries}"CInterface.o ∂
                                        "{CLibraries}"CRuntime.o ∂
                                        "{Libraries}"Interface.o ∂
                                        -o MyExternal

                        In Think C:
                                Define macro ThinkC below.
                                Set up a project with this file and MacTraps
                                library.
                                Build project as CODE resource with ID 0, custom
                                header, type 'RsTl' and creator 'Rslv'.
*/

/* defined here because it's used by ResolveTools.h */
#define numfuncs 1

/*** Includes ***/

/* define ThinkC if you are using Think C */
/* #define ThinkC     */

#ifdef ThinkC
#include "mactypes.h"
#include "memorymgr.h"
/* THE HEADER IN ThinkCGlue.h MUST BE CHANGED WHENEVER ANY CODE IS ADDED */
#include "ThinkCGlue.h"
#define nil 0L
#else
#include "types.h"
#include "memory.h"
#include "ResolveTools.h"
#endif

/*** Prototypes ***/
static void Reverse(PVAL, PVAL);
static void CleanUp(void);

/*** Globals ***/
/* strings returned to Resolve must be allocated as a globals or as
* non-relocatable blocks on the heap
*/
Str255gReturnString;        /* for return from our external routine */
PROUT               gRout;          /* pointer to our routine list structure */

/*** Functions ***/

/*
        FUNCTION:  Main
        ARGUMENTS:      None
```

```
	RETURN:		PROUT		pointer to routine list structure
	DESCRIPTION:
		Called when GET EXTERNAL is executed.
		Does any initialization, allocation required.
		Initializes our ROUT structure, and returns it to Resolve.
*/

PROUT main()
{
	PROUT saveGRout;	/* This is what is returned to Resolve.
				 * We can't return gRout in ThinkC since it gets blown
				 * away by RestoreA4() call.
				 */
#ifdef ThinkC
	RememberA0();	/* Set up access to globals */
	SetUpA4();
#endif

	/* Call InitGraf() if external will do any drawing */

	/* Allocate and set up routine list and
	 * its routine list elements.
	 * It could easily be a global, but
	 * we're allocating it here just to
	 * give our CleanUp function something to do.
	 */
	saveGRout = gRout = (PROUT)NewPtr(sizeof (ROUT));
	if (gRout != nil)
	{
		gRout->nrout = numfuncs;
#ifdef ThinkC
		gRout->exitfunc = (ProcPtr)CleanUp;	/* called when REMOVE EXTERNAL
							 * is executed
							 */
		gRout->relts[0].pfunc = (ProcPtr)Reverse;	/* Function that handles
								* this external
								*/
#else
		gRout->exitfunc = CleanUp;		/* called when REMOVE EXTERNAL
							* is executed
							*/
		gRout->relts[0].pfunc = Reverse;  /* function that handles this
							* external */
#endif
		gRout->relts[0].name = "\pReverse";	/* must be a Pascal string,
							* allocated on heap */
		gRout->relts[0].narg = 1;
	}
#ifdef ThinkC
	RestoreA4();					/* Restore previous A4 value */
#endif
	return (saveGRout);				/* Pointer to routine list */
}

/*
	FUNCTION:	CleanUp
	ARGUMENTS:		None
	RETURN:			None
	DESCRIPTION:
		Called when REMOVE EXTERNAL is executed.
		Deallocates anything allocated by program.
```

```
*/
void CleanUp()
{
#ifdef ThinkC
	SetUpA4();				/* Set up access to globals */
#endif
	if (gRout)
		DisposPtr((Ptr)gRout);		/* Get rid of ptr allocated in main() */
#ifdef ThinkC
	RestoreA4();				/* Restore previous A4 value */
#endif
}


/*
	FUNCTION:		Reverse
	ARGUMENTS:		PVAL returnVal		points to return value structure
				PVAL arguments		points to arguments array (in
								this case only 1 argument)
	RETURN:			None
	DESCRIPTION:
		Useless example function that reverses order of characters
		in string passed in
*/

void Reverse(returnVal, arguments)
PVAL returnVal;
PVAL arguments;
{
	unsigned char	length;
	short		errMessage;
	char		*sourceChar;

#ifdef ThinkC
	SetUpA4();			/* Set up access to globals */
#endif
	errMessage = 0;
	if (arguments[0].flag == STRING)
	{
		length = arguments->val.string[0];
		sourceChar = arguments->val.string + 1;
		gReturnString[0] = length;
		do
		{
			gReturnString[length] = *sourceChar;
			sourceChar++;
		}
		while (--length);

		/* set up the value structure */
		returnVal->flag = STRING;
		returnVal->val.string = (char *)gReturnString;
	}
	else
	{
		errMessage = 12;	/* bad argument */
		/* set up an error value structure */
		returnVal->flag = ERR;
		returnVal->val.err = errMessage;
	}
#ifdef ThinkC
```

```
        RestoreA4();                    /* Restore previous A4 value */
#endif
}
```