RPC Stub Compiler
Version 0.1.3

0. About this release

This release of the RPC stub compiler is the first one, as such I'm sure there are lots of bugs, I've hammered on it as much as possible, but for what tries to be a very orthoganal piece of code it's hard to poke in all the corners, I'm planning a new major release probably in mid June 1991 - if people report bugs back to me - I'll try and bring out small maintainance releases in the meantime, I can be reached at taniwha!paul@mtxinu.com (on the internet, this I read every day) and at CAMPBELL.P on AppleLink (this I read about once a week, sometimes more, sometimes less).

The RPC stub compiler is shareware, this doesn't mean it's free, just that you don't have to pay for it unless you intend to use it. If you do so please donate $10 (for individuals) or $50 (if you are using it on behalf of a company) to Amnesty International or the ACLU

thanks

Paul Campbell
May 24th 1991

1. Introduction

This document describes the RPC Stub Compiler called 'rpc' which is an MPW tool to create Apple Events glue for program to program communication.

An RPC Stub Compiler is a compiler that reads a description of a procedural interface (like a subroutine call) and produces the necessary code so that the call may be made to a subroutine in a remote process. In this case the transport used is AppleEvents and the compiler makes all the AppleEvents calls required to round up all the arguments, send them to the remote process, unpack them and call the server subroutine, and return the resulting value (if any) back to the calling routine.

This compiler is based around C and it accepts a C-like syntax for its subroutines, it produces as output MPW C source which can be compiled to produce the resulting AppleEvents glue. Since it uses AppleEvents as a transport you must be running System 7.0 or later of the Mac OS.

To Install it simply drag the MPW Tool rpc into your MPW 'Tools' directory.

The MPW tool takes the form:

rpc [-v] [-o *output-file*] *input-file*

where:

-v is an optional flag which turns off verbose mode (the copyright and version message)

-o *output-file* allows you to optionally specify the output file name, by default this will be the *input-file* with '.h' appended to it.

input-file this is the name of the input file and usually ends with the suffix '.rpc'

Chapter 3 of this manual describes the syntax of the *input-file* in greater depth.

The *output-file* is a C include file, it may be included into many different C source files, it contains 4 parts:

- Subroutine prototypes for the client and server action subroutines (client routines have the same names as in the *input-file* , server action routines have the suffix '_SERVER' appended to them).

- The source to the client access routines - these are only included if the #define RPC_CLIENT is defined and this should only be done in one of the files that the *output-file* is included in. You should make sure that all data structures that end up being referenced by this file are defined BEFORE it is included - this mostly applies to

parameters to routines you define. If you are just compiling a server these don't have to be included at all.

- The source to the server access routines - these are only included if the #define RPC_SERVER is defined and this should only be done in one of the files that the *output-file* is included in. You should make sure that all data structures that end up being referenced by this file are defined BEFORE it is included - this mostly

applies to parameters to routines you define. If you are just compiling a client these don't have to be included at all. The server access routines all have the suffix '_HANDLER' and call the server action routines that you must provide which have the suffix '_SERVER'.

- Source to a routine, called 'rpc_server_init', that will register the server access routines with AppleEvents - you should include this once in your sources (by defining RPC_SERVER_INIT) and call it during your application's initialization.

The AppleEvents glue code use some basic assumptions about timeouts etc these may change in future releases of the rpc compiler, or of course you can edit the resulting sources to change them yourself.

If you want to provide an idle event loop to AESend when when you define RPC_CLIENT you can #define the symbol CLIENT_IDLE_FUNCTION to be the name of your idle function (this is important if you intend to be able to make rpc calls to yourself).

Since this rpc package is based on AppleEvents, in your server you must dispatch incoming AppleEvents (events of type kHighLevelEvents) by calling AEProcessAppleEvent (which will in turn call your server routines). Check out Inside Mac IV pp6-26/6-27 for more detail.

2. Q/A

Q: Why isn't it in Pascal rather than C?

A: Because I program in C and I wrote it :-) ... actually all the routines are Pascal callable you just have to compile them using C.

Q: What if I use Think C rather than MPW?

A: Same sort of answer .... I use MPW C from day to day and don't have Think C so I can't compile it.

Q: How do I turn off the annoying copyright message?

A: Use the '-v' flag, if it's annoying you you've probably got the message.

3. Syntax

Input data is similar to C, both sorts of comment (/* ... */ and // to the end of the line) are supported.

If a '#' is discovered in the input text it and the following characters to the end of the current line are copied to the output, you should only do this BETWEEN declarations, if you don't the results are undefined.

Text is case sensitive, just like C. Anywhere where you can input a '????' style value you can also use the corresponding symbolic name, rpc knows about all the names in the MPW include files and in the Apple Events Registry from the Golden Master 7.0 CD - where they differ values are resolved in favor of the MPW files.

The following are reserved words and can't be used for *name* s

char	cstring	double	
enum	extended		float
handle	list		long
of	option		pstring
short	struct		union
unsigned			

Error reporting is rather tacky - the basic error is 'parse error' with the line number and file it occured on.

The text consists of a list of declarations of the form:

[*type*] *name* (*class* , *id*) ([*param* ,...]) ;

Where:

type	is an optional return type, currently you are limited to one return value and it's types are restricted to 'simple' types (see below).
name	is the name used to name all the generated routines (make sure it doesn't clash with anything else you have declared).
class	is the event class that the server responds to
id	is the event ID that the server responds to
param,...	is an optional list of parameters separated by commas.

Parameters have one of 7 formats:

c-type	*name*		[*key*]
c-type	*name*	[]	[*key*]
s-type	*name*		[*key*]
s-type	*name*	[]	[*key*]
list of *c-type*	*name*		[*key*]
list of *s-type*	*name*		[*key*]
list of (*param* ,...)			[*key*]

Where:

c-type this is a C style type, valid types are:

char
unsigned char
short
unsigned short
long
unsigned long
float
double
extended
struct *name* *
union *name* *
enum *name*
cstring
pstring
handle

the last two are null terminated and counted strings resp.

s-type This one of the following types:

typeBoolean
typeChar
typeSessionID
typeLongInteger
typeShortInteger
typeLongFloat
typeShortFloat
typeExtended
typeComp
typeKeyword
typeType

typeEnumerated
typeMagnitude
typeApplSignature
typeAlias
typeAppParameters
typeAEList
typeAERecord
typeFSS
typeTargetID

typeProcessSerialNumber

(other types may be added in future releases of the compiler - what do you want?)

key Is a key (either a keyword or a string of 4 characters surrounded by single quotes) and corresponds to the key used for the parameter within an AppleEvent, it is optional and if you don't define it the compiler will generate one for you.

param ,... This corresponds to a list of parameters separated by commas, parameter lists in 'list of ( )' like this may not have *key* options.

Options

Input can also contain lines of the form:

option *option,...* ;

Where:

option,... is a list of option of one of the following forms:

name This sets a boolean option

- *name* This clears a boolean option

name = *string* This sets a string options to the value (inside of double quotes) given by the *string* option.

4. Parameters

Parameters passed to rpc stub routines depend on how you declare them, the various different options allow you some control on how the parameters are packed up and sent to the remote server, wherever possible the standard type codes for parameters are used, where this is not possible RPC will make some for you (where this happens is noted below).

c-type *name*

For this type you can use any of the C-style types described above.

If you use the 'struct *name* *' or 'union *name* *' options you must pass the addresses of the structures or unions you wish to pass, since these may be of any type the compiler doesn't use a standard type code but instead uses one of it's own - this makes it hard to send arbitrary structures to known servers.

If you use the 'cstring' or 'pstring' types you must pass the address of a null terminated or counted string resp. These parameters are passed as type 'TEXT' parameters.

If you pass something of type 'handle' it must be a genuine Mac memory manager handle - since the glue uses GetHandleSize to find out how big it is.

c-type *name* []

This type is similar to the previous type except that the address of an array of the appropriate type is passed to the rpc routine as well as an integer describing the number of entries in the array (the next parameter after the address of the array).

This is a relatively efficient way to send data since it all gets packed up into one parameter, however it's not suitable for using to pass data to well known servers since they expect descriptor lists (see below)

You can't use types 'cstring' and 'pstring' with [] options, this is because such strings have variable sizes and you can't make arrays of them.

s-type *name*

For this type you can use any of the standard types described above.

If you use the type typeChar it is treated exactly like a 'pstring' described above.

If you use a typeAlias you must pass an alias handle as a parameter.

s-type *name* []

This passes arrays of standard types to remote servers and works in the same manner as for C-types above.

You can't use types typeChar and typeAlias with [] options, this is because such objects have variable sizes and you can't make arrays of them.

list of *c-type* *name*

This construct creates an AppleEvents Descriptor list of the standard c-types being passed, unlike the [] construct above you can make arrays of 'cstring's and 'pstring's (by passing the address of an array of pointers to them). Like the [] construct you have to pass a count of the number of entries in the next parameter to the routine.

list of *s-type* *name*

This construct creates an AppleEvents Descriptor list of the standard c-types being passed, unlike the [] construct above you can make arrays of 'typeChar's and 'typeAlias's (by passing the address of an array of pointers to them). Like the [] construct you have to pass a count of the number of entries in the next parameter to the routine.

list of (*param* ,...)

This allows you to group the parameters within the parentheses into an AppleEvents descriptor list all with the same key.

When a server routine is called all the parameters are passed into it in the same manner as they were passed into the client, in the case of dynamic parameters (for example arrays, strings or aliases) the server stub will deallocate the space used when the call completes - don't attempt to do this for it.

The type of the result returned (if any) from an RPC routine is currently restricted to simple *s-types* and *c-types* (with the exception of the variable length ones such as cstrings, pstrings and typeAliases - this restriction will probably be removed in future compiler releases - along with the limitation of being able to return just one value).

Finally whenever you call an rpc stub you must pass as the first parameter the address of an AEAddressDesc descriptor describing the remote process you wish to send the message to (you can use the PPCBrowser or other PPC utilities to generate this record).

5. Options

There are two types of options - boolean options have the values on and off and are set and cleared by simply using their name within an optionsstatement (or the name preceded by a minus sign to clear them). String options have string values and are set by an entry in an option statement of the form 'name = "string" ', they are usually used to replace some text in the output file.

The following boolean options are defined:

Name	Function	Default Value
client	Enables output of client stub code	on
server	Enables output of server stub code	off

The following string options are defined:

Name	Function	Default Value
send_timeout	sets the timeout value for client calls to AESend	"kAEDefaultTimeout"
send_options	sets the options value for client calls to AESend	"kAEWaitReply | kAEAlwaysInteract | kAECanSwitchLayer"
send_prio	sets the prority value for client calls to AESend	"kAENormalPriority"
send_idle	sets the idel function for client calls to AESend	"CLIENT_IDLE_FUNCTION"
send_filter	sets the filter function for client calls to AESend	"NULL"

6. Examples

```
//
//
//              This file is an example of the rpc stub compiler
//
//
//      Notice that whenever a # sign is discovered everything from it to the end of it's line
//              is included in the output file
//

#include <aliases.h>

/*
*
*               The following is an example showing all the different types of parameter
*                       passing and what sort of real parameters things get converted to in
*                       the stubs
*/

short fred ('abcd' : 'defg') ('long' a[],
                        // becomes      - long *a, unsigned long a_count
                                struct b *b,
                                  //                    struct b *b
                                list of float c,
                                        //                      float *c, unsigned long c_count
                                list of (long d, short e) 'QQQQ');
                                        //                      long d, short e
//
//
//              Which ends up looking like:
//
//      For the client                  pascal OSErr fred(AEAddressDesc *server,
//                                                      short *result,
//                                                      long *a, unsigned long a_count,
//                                                      struct b *b,
//                                                      float *c, unsigned long c_count,
//                                                      long d, short e);
//
//
//      For the server                  pascal OSErr fred_SERVER(short *result,
//                                                       long *a, unsigned long a_count,
//                                                      struct b *b,
```

```
//                                      float *c, unsigned long c_count,
//                                       long d, short e);
//
```

```
//
//
//      here are some stub declarations for the core events for things that only
//              handle aliases
//
//


core_open_doc(kCoreEventClass:kAEOpenDocuments)
                        (list of typeAlias THINGS keyDirectObject);

core_print_doc(kCoreEventClass:kAEPrintDocuments)
                        (list of typeAlias THINGS keyDirectObject);

core_open_app(kCoreEventClass:kAEOpenApplication)
                        (list of typeAlias THINGS keyDirectObject);

core_quit_app(kCoreEventClass:kAEQuitApplication)
                         ();

//
//      for example core_open_doc ends up as
//              pascal OSErr core_open_doc(AEAddressDesc *server,
//                      AliasHandle **THINGS, unsigned long THINGS_count);
//
//



//
//      Here are some stub declarations for the same things with fsspecs ... (commented
//              out here because you don't want to declare 2 handlers for the same event)
//

//core_open_doc_fs(kCoreEventClass:kAEOpenDocuments)
//                      (list of typeFSS THINGS keyDirectObject);

//core_print_doc_fs(kCoreEventClass:kAEPrintDocuments)
//                      (list of typeFSS THINGS keyDirectObject);

//core_open_app_fs(kCoreEventClass:kAEOpenApplication)
//                      (list of typeFSS THINGS keyDirectObject);

//
//      for example core_open_doc_fs ends up as
//              pascal OSErr core_open_doc_fs(AEAddressDesc *server,
```

```
//                      FSSpec *THINGS, unsigned long THINGS_count);
//
```

```
//
//      finally here are some examples of the options keywords, firstly using the
//      server/client options to create a server that gets FSSpecs and a client that sends
//      aliases, also set the send_timeout to be 400 ticks
//

option server -client send_timeout = "400";

test_open_app_alias('test':kAEOpenApplication)
                (list of typeAlias THINGS keyDirectObject);

option -server client;
test_open_app_fs('test':kAEOpenApplication)
                (list of typeFSS THINGS keyDirectObject);

option server client;
```