PHILIPS

PHILIPS INTERACTIVE MEDIA

Technical Note #71

Resource Compiler/Manager for CD-I Applications

Ken Ellinwood

December 2, 1991

This technical note describes a method for loading and accessing pre-initialized data at run time. This method is intended to aid in the development of data-driven CD-I applications.

Resource Compiler/Manager for CD-I Applications

Writing Data-Driven Software for CD-I

The concept of data-driven software has become popular among engineers in the CD-I software community. For example, this method was used with success on the CD-I Ready Kit program, where the Macintosh resource compiler *rez* was used to generate resource assets for inclusion in disc builds. Essentially, a *resource asset* is a block of statically initialized data that can be loaded into memory at run-time. The data elements within the resource asset are then accessed through the use of a resource manager; the resource manager returns a pointer to the data element based on a symbolic parameter.

For example, an engineer whose title uses a large number of hotspots might want to load data describing the hotspots into memory at run-time rather than compiling the hotspot structures in the source code. The engineer would develop a resource description file that describes the structure, symbolic names, and data for the hotspots. Then, the engineer would compile the resource description to generate the resource asset. The resource asset could be included in an OS-9 data module or could be included as FORM1 sectors in the disc build for loading by a play manager. Once the data is loaded into memory at run-time, the resource manager returns pointers to the data structures defined in the resource description file based on a symbolic name.

There are several drawbacks to using *rez* on the Macintosh as the resource compiler. Using *rez*, the engineer develops *rez* header files to describe the data structures that may be declared and initialized at a later time in a resource description file. The data description mechanism is not rendered in the C language, but in a *rez* specific language. This requires the engineer also to write an analogous description of the data structure in the C language for use in the run-time code which, although it seldom presented a problem on the CD-I Ready Kit project, is problematic because two definitions of the data are required—one for *rez* and one for the C language source code. Another drawback arises when data structures contain pointer references to other structures. Although no one should expect that global or external pointer references will be resolved by a resource manager of this type, by using *rez* the engineer is forced to write code to resolve pointer references that are local to his own data descriptions. Of course, the most severe drawback to using *rez* is that it is available only on the Macintosh.

A better solution is to write the resource definitions the C programming language. Thus, the same data description files (*#include* files) may be used at build-time to generate the resource asset files and in the run-time code where the data is actually used. If the resource compiler is written in portable C code, then it can also be compiled to run on a variety of development platforms, including the Sun SPARC stations, the Sun/3, and the Macintosh. Of course, a good feature would also be the resolution of pointers local to the resource asset at run-time by the resource manager. Does this sound like too much? Well, as the saying goes: "We have the technology."

The Resource Compiler Program

A program called **rc** (for resource compiler) has been developed that makes the above functionality possible. Use of the program involves writing C language *#include* files and then declaring and initializing data in a C source file; compiling the files with the OS-9 compiler; and processing the output of r68 with **rc** to generate the resource asset file. Information in the **r68** output file about symbolic names, pre-initialized data, and pointer references is preserved and made useful in the format of the resulting resource asset file. A mechanism for including raw data files as resources in the resource asset file has also been included.

The resource compiler/manager package consists of one build-time utility, **rc**, two compiler macros, **RmMakeID()** and **RmStr2ID()**, and two run-time functions, **RmGetData()** and **RmGetSize()**.

Code Example

On the following page is a code example for **rc**.

```
#include <cursor.h>

CPCALLBACK cpcb[] = {

        /* Event mask,                                  function, param */
        { EV_COUT,                                      NULL,     0},
        { EV_MOVE,                                      NULL,     1},
        { EV_CIN,                                       NULL,     2},
        { EV_B0D | EV_B1D,                              NULL,     3},
        { EV_B0U | EV_B1U | DISP_LASTONE,               NULL,     4}
};

HOTSPOT hspt = {
        HOT_OPAQUE,             /* Flags */
        NULL,                   /* Cursor */
        0, 0, 768, 480,         /* Rectangle */
        NULL,                   /* Region */
        NULL,                   /* Sibling */
        NULL,                   /* Child */
        cpcb                    /* Cp Callbacks */
};
```

This code is easily compiled using the OS-9 cross compiler. Once the **r68** phase has been completed, **rc** processes the information in the **.r** file into a resource asset file. By default (unless the **-s** option is used) , symbolic names in the files are truncated or padded out to four characters and then transformed into four byte identifiers for use at run time. The names **cpcb** and **hspt** in the example above remain unchanged because they are already four characters long. During the first call to **RmGetData()**, the resource manager resolves any locally unresolved references within the resource asset. In the example above, if the pointer to **hspt** is requested, the callback pointer field in the **HOTSPOT** structure would be resolved to point to the array, **cpcb**, by the time **RmGetData()** returns. This would also be true for the cursor, region, sibling, and child fields if pointers to local structures are also declared and then referenced in the **HOTSPOT** structure in the resource description file. The resource manager cannot resolve references to data that lives outside the scope of the resource asset file; thus, the actual functions for the **CPCALLBACK** structures must be assigned at run-time.

Program Description

Name: **rc**

Usage: **rc [options] [<output file from r68>] [<resource asset file>]**

Options:

-s	use variable length strings instead of four byte identfiers as resource symbols.
-f=<symbol,datafile>	read *datafile* as a resource accessible by *symbol*
-v=<verbose level>	-1 : suppress warnings 0 : default 1 : verbose
-m=<string memory>	1024 default for reading variable length strings
-o=<options file>	read more options/filenames from the named file
-r=<resource file>	output resource asset file

This utility reads the output from the OS-9 compiler (pass **r68**) and/or raw data files (**-f=** option) and writes a resource asset file. When **rc** reads the r68 compiled input file, only pre-initialized data symbols and the pre-initialized data are kept. Uninitialized data symbols, code symbols, and unresolved external code and data symbols are discarded. Warnings are issued when these symbols are encountered by the resource compiler. The reserved word "static" in the C language resource description file should not be used, because the associated symbols do not appear in the output of **r68** and, therefore, will not be accessible by the resource manager. Unless the **-s** option is used, symbolic names must be four characters in length and are padded or truncated as necessary.

The **-s** option may be used to store resource names as variable length strings instead of four byte integers. A resource asset compiled with the **-s** option will always be larger than one compiled without it, so if storage space is an issue, you may not want to use it.

Raw data files may be included in the resource asset using one or more occurrences of the **-f=** option. This option binds a symbol in the header of the resource asset file to the data read from the named data file so that it may be

accessed through the resource manager using the symbol at run-time. In the event that the -f= option is used exclusively to provide input for **rc** (i.e. no r68 compiled file is specified as input), then the **-r=** option must be used to specify the output file.

Warning Messages

discarding uninitialized data symbol

This warning is issued when a data object in the C language source file has been declared, but not initialized. This warning indicates that the data object will not be available in the resource asset file.

discarding code symbol

This warning is issued when a code symbol is encountered. It means that one or more functions were present in the C language source file.

discarding unresolved external symbol

This warning is issued when unresolved external code or data symbols are encountered. For data symbols, this usually means that the data was declared using the C language reserved word "extern."

unknown type in unresolved reference list

This warning is issued when an unknown record type is encountered in the unresolved local symbol list. At the time of this writing, **rc** issues this warning when the C language source file has been compiled with the debug flag, **-g**.

data name has been truncated to four characters

Unless the **-s** option is used, data names (symbols) must be four characters long. This warning is issued to indicate that this rule has been enforced.

data name padded with spaces out to four characters

Unless the **-s** option is used, data names (symbols) must be four characters long. This warning is issued to indicate that this rule has been enforced.

Macro Descriptions

Name: **RmMakeID()**

Usage:
```
#include <resource.h>

int RmMakeID( c1, c2, c3, c4)
char c1, c2, c3, c4;
```

This macro constructs a four-byte integer from the four character parameters. Unless the **-s** option is used, resource names are forced to be four characters long so that storage and searching use only four bytes. This macro provides a source code translation mechanism between the character representation of the name used in the resource description file to the integer ID used at run-time.

Name: **RmStr2ID()**

Usage:
```
#include <resource.h>

int RmStr2ID( idStr)
char *idStr;
```

This macro casts the idStr into a four-byte integer. The idStr must be four characters long. Unless the **-s** option is used, resource names are forced to be four characters long so that storage and searching use only four bytes. This macro provides a source code translation mechanism between the character representation of the name used in the resource description file to the integer ID used at run-time.

Run-Time Functions

Name: **RmGetData()**

Usage:

```
#include <resource.h>

void *RmGetData( asset_ptr, id)
void *asset_ptr;
int id;

void *RmGetData( asset_ptr, name)
void *asset_ptr;
char *name;
```

This function returns a pointer to the named data. The parameter **asset_ptr** is a pointer to the resource asset created by the **rc** utility. The **id** parameter is a four-byte identifier that symbolically names the data (used when the resource asset was compiled without the **-s** option). The **name** parameter is a pointer to a variable length string that symbolically names the data (used when the resource asset was compiled with the **-s** option).

Name: **RmGetSize()**

Usage:

```
#include <resource.h>

int RmGetSize( asset_ptr, id)
void *asset_ptr;
int id;

int RmGetSize( asset_ptr, name)
void *asset_ptr;
char *name;
```

This function returns the total size of the named data in bytes. For arrays, the size returned is the size of each element multiplied by the number of elements in the array. The parameter **asset_ptr** is a pointer to the resource asset created by the **rc** utility. The **id** parameter is a four-byte identifier that symbolically names the data (used when the resource asset was compiled without the **-s** option). The **name** parameter is a pointer to a variable length string that symbolically names the data (used when the resource asset was compiled with the **-s** option).

More Information

These resource compiler/manager functions are available to P.I.M.A. co-producers. Please contact Drew Topel at (310) 444-6516 to obtain a copy.