NCR CONTROL DATA CORPORATION

ADVANCED SYSTEMS LABORATORY

CHAPTER 11

MAJOR INTERFACE AREAS

Doc. No. ASL00282

Rev. 04

Copy No. 87

TABLE OF CONTENTS

1.0 INTRODUCTION

1.0 INTRODUCTION

To be supplied

2.0 MESSAGE PROTOCOL

2.0 MESSAGE PROTOCOL

To be supplied

3.0 PROCESS STATE SWITCHING

To be supplied

4.0 COMMAND LANGUAGE (MACROS)

To be supplied

5.0 CONTROL LANGUAGE (MACROS)

5.0 CONTROL LANGUAGE (MACROS)

To be supplied

6.0 LOGICAL NAME SPACE

To be supplied

7.0 OBJECT CODE ENVIRONMENT AND FORMATS

7.0 OBJECT CODE ENVIRONMENT AND FORMATS

7.1 INTRODUCTION

Object information for any module in IPLOS may reside in two different formats: object modules and load modules. Object modules are the output of compilers and the input to both the loader and OBLIGE, the library generator. Load modules are only output by OBLIGE and may be input to the loader and OBLIGE as well. Two formats are provided for approximately the same purpose to allow one of the formats (object module) to be amenable to compiler code generation, and the other to be amenable to operating system purposes (i.e. sharing of code).

Table 7-1 summarizes the differences between object modules and load modules.

7.0 OBJECT CODE ENVIRONMENT AND FORMATS
7.1 INTRODUCTION

TOPIC	OBJECT MODULE	LOAD MODULE
structure	binary file of record descriptor-record pairs; each pair representing a logically discrete entity	a virtual memory segment
access	SWL standard I/O	directly addressed
output from	compilers	library generator (OBLIGE)
inout to	loader OBLIGE	loader OBLIGE
code shared?	no : code section is in record format which must be copied for each instance of execution	yes : code is in executable form; same physical image can be given to each instance of execution
residency	binary files only	IPL library segments only

TABLE 7-1
Object module-load module summary

7.0 OBJECT CODE ENVIRONMENT AND FORMATS
7.1 INTRODUCTION

Any program executing under IPLOS may have three major kinds of components which reside in different segments having different characteristics and protection attributes. The IPLOS object and load module structures are designed to support that three part environment. The three kinds of components are code, working storage and binding. Table 7-2 summarizes the protection, contents and characteristics of the three sections.

	PROTECT	CONTENTS	CHARACTERISTICS
CODE	read execute	.reentrant instructions .constant data	.sharable among all activations; if module is in load module format, every activation shares the same physical copy .only one per module
WORKING STORAGE	section dependent	.all unshared or modifiable data	.nonshared among all activations; a new copy is provided for each instance of execution .typically multiple sections per module .protection attributes are compiler specified
BINDING	read binding	.pointers or pointer pairs .word aligned	.nonshared among all activations; a new copy is provided for each instance of execution .unstructured; no ordering relationship may exist between pointers or pointer pairs .binding attribute is administered by IPLOS; may not be compiler specified .only one per module

TABLE 7-2
Program components

A more detailed description of the object and load modules and the object environment is provided in the ensuing sections.

7.0 OBJECT CODE ENVIRONMENT AND FORMATS
7.2 OBJECT MODULE SUMMARY

7.2 OBJECT MODULE SUMMARY

The object module is a file of binary records with the following topology:

```
<record descriptor 1>
     <record 1>
<record descriptor 2>
     <record 2>
          ...
<record descriptor n>
     <record n>
```

Each record descriptor contains two fields which define the ensuing record: 1.)record type and 2.)length (record type dependent). The length field is used chiefly to fix the lengths of adaptable arrays.

For the sake of simplicity, the record descriptor-record pairs will be referred to as records in the remainder of this document.

The following is a list and explanation of each variant record of the object module:

Record ID	Explanation
IDR	Identification record; first record of the module; specifies module name and attributes.
SDC	Section definition record; specifies length and attributes of every object module section (code, working storage, and binding) and all common blocks.
TEX	Text record; specifies data to be placed into any section.
RPL	Replication record; specifies data to be repetitively inserted into any section.
BIT	Bit insertion; inserts a specified subset of a byte into any section.
EPT	Entry point definition; defines an address in any section as a named externally accessible value.
RIF	Relocation information; defines those areas in each section which must be modified by OBLIGE when binding modules together; not processed by the loader.
AIN	Address insertion; replaces or adds the address of a section of the module to another location within a module; allows the construction of full PVAs at load

time; required since the ring number, segment number, and offset of each section are only determined at load time.

AOI Address offset insertion; essentially the same as AIN above except that a (section,offset) may be replaced or added.

XRL External reference linkage; specifies a list of addresses in the containing module into which the address of the named external is to be placed.

TRA Transfer record; specifies a.)the primary entry point and b.)the end of the object module.

The object module records must be arranged in the following order:

1.)IDR record
2.)SDC records for all object module sections
3.)TFX,RPL,BIT,EPT,RIF,AIN,AOI, and XRL records in arbitrary order
4.)TRA record

The records in group three are not required to be in any order, however they will be processed by the loader in the order that they are received. Therefore some concern must be given to the order in which they are generated.

7.3 LOAD MODULE - LIBRARY SEGMENT SUMMARY

7.3.1 LIBRARY SUMMARY

A library is a directly addressible virtual memory structure which contains a number of modules plus the module and entry point dictionaries required to retrieve them.

When an object module is placed on a library it is reformatted into a load module. The most significant difference between an object module and a load module is that the code section of the load module is in executable form. This means that the library segment containing the load module need only be INITIATEd in the address space of the requesting user and the code section will be ready to execute (the working storage and binding sections must still be allocated and initialized). Furthermore any other task in the same or any other job requesting the use of that library will receive the same copy of the code section although not neccessarily INITIATEd in the same process segment number.

The components which organize and define the load modules on a library are as follows:

COMPONENT	EXPLANATION
Library header	Identifies the library; contains relative pointers to the library dictionaries.
Subprogram dictionary	Contains the name, attributes and relative pointer to the load module for each subprogram module. Used for calls to subprograms.
Procedure dictionary	Contains the name, attributes and relative pointer to the load module for each procedure module; used by OBLIGE during library generation.
Entry point : module dictionary	Contains the relative pointer to the load module for each entry point in each procedure module; used by the loader to load modules associated with a particular entry point.

7.3.2 LOAD MODULE SUMMARY

A detailed format of the load module is not available at this time, however an outline of the major elements of the load module and of the components of each element is provided below.

The load module consists of six major elements: the module header, the code element, the linkage element, the working storage element, the entry point definitions, and the information element. The components of each element are summarized below.

7.3.2.1 Module header

The load module header identifies and organizes the load module. The header is pointed to by one of the module dictionaries of the library in which it resides. It contains the following items:

- Module header header
  - Primary entry point name
  - Back pointer to procedure or subprogram module dictionary entry for this module
  - Module generator code
  - Module generator name and version
  - Time/date created
  - Pointer (relative to module header header) of section definition list
  - Number of section definitions
  - Pointer (relative to module header header) of load module map
  - Number of load module map entries
  - Commentary supplied at module generation time
- Section definition list entry - one for each section defined in the load module
  - Section type
  - Section attributes
  - Section length
  - Maximum length for extensible sections
  - Section alignment
  - Name for common blocks
- Load module map entry - one for each element of the load module
  - Element type
  - Pointer (relative to the library segment) of the element

  - Element length

7.3.2.2 Code element

The code element may contain constants, instructions and relative pointers to other sections. This element must be nonselfmodifying to enable the element to be shared by all activations of the load module.

7.3.2.3 Linkage element

The linkage element contains the names and linkage chains for all external references made by this load module.

7.3.2.4 Working storage element

The working storage element contains the interpretive initialization information for all the working storage sections and common blocks defined in the module. Each of these sections is allocated and initialized for every activation of the module. Since the code section is not modified at load time, all modifiable data and full address pointers must reside in working storage sections.

The kinds of interpretive initialization records supported in the working storage element are as follows:

Record ID	Record Name
TEX	Text insertion record
RPL	Replication record
BIT	Bit insertion record
AIN	Address insertion record
AOI	Address offset insertion record

Since the System/hardware calling convention only provides a called module with the address of its binding section, the binding section must contain self-referencing links which allow the code section to find the appropriate working storage sections

at execution time.

7.3.2.5 Entry point definitions

The entry point definitions are a list of all the named, externally accessible addresses in the load module.

7.3.2.6 Information element

The information element contains information which does not belong in the other four elements, notably relocation information for library generation, module information, compiler generated symbol tables, etc. The element consists of a header and several tables which make up the body of the information element.

7.3.2.6.1 INFORMATION ELEMENT HEADER

The information element header organizes the information element. It contains the list headers for the other components of the information element.

7.3.2.6.2 COMPONENT IDENTIFICATION

A load module may consist of several object and load modules bound together by OBLIGE. In this case a component identification entry is maintained for each of the original components of the module. Each entry consists of the following items:

- Module name
- Module generator code
- Module generator name and version
- Time/date created
- User supplied commentary

7.3.2.6.3 RELOCATION INFORMATION

Relocation information is used by OBLIGE when binding object or load modules together. It identifies every offset relative to the base of either the code or binding section which must be altered to reflect the offset in the new bound module. Relocation information is not used by the loader when the module is loaded.

Each relocation information item consists of the following:

- Section and offset containing field to be relocated
- Section to which field is to be relocated (i.e. code or binding)
- Size and kind of field
- Sign and type of offset contained within the field

7.3.2.6.4 BINDING SECTION TEMPLATE

The binding section template is produced by OBLIGE whenever it creates a load module. It identifies the contents of each word in the binding section for the load module.

7.4 OBJECT ENVIRONMENT AND CONVENTIONS

7.4.1 OBJECT COMPONENTS

Any program executing in IPLOS has an object environment consisting of three types of components: code sections, working storage sections, and binding sections. A single module (object or load) consists of a single code section, a single binding section, and multiple working storage sections. The code section is separated from the working storage sections in order to allow the sharing of the code section among multiple activations of the module. The binding section is separated from the working storage sections in order to a.)allow controlled transfer between rings of protection and b.)allow the binding sections of several load modules to be combined during library generation by the elimination of matching entry point-external pairs and redundant external references.

7.4.1.1 Code sections

All code produced by standard IPL compiler products must be reentrant to allow it to be shared among all activations of the module. The code section of every module, therefore, should only contain nonselfmodifying instructions, constant data and relative pointers to other sections. There may be no more than one code section per module.

Since the object module may contain code that has been generated discontiguously (i.e. not in the order in which it is to be executed), the code sections of object modules are allocated in a segment INITIATEd at load time for every activation of the module. However, should that module be processed by the library generator and reformatted into a load module, its code section will be in executable image and will be shared among all activations of the module.

7.4.1.2 Working storage sections

Working storage sections contain all modifiable, nonshared data used during a modules execution. There may be an arbitrary number of working storage sections per module each having its own

attributes specified by the program that generated the module.

The attributes that may be specified for each working storage section are as follows:

SECTION TYPE

• Working storage	Fixed length known at allocation time and unchanging during execution; always allocated and initialized when module is loaded.
• Common block	Named data section equivalent to FORTRAN common, SWL external, PL/1 static external, and COBOL (ATG) global; allocated once for every task.
• Extensible working storage	Like working storage except length may increase during execution; maximum length is specifiable.
• Extensible common	like common block except length may increase during execution; maximum length is specifiable.

PROTECTION ATTRIBUTES

• Read	Indicates section is readable
• Write	Indicates section is writable
• Execute	Indicates section is executable
• Binding	Indicates section is a binding section; this attribute is administered by the system and may therefore only be specified in the binding section (c.f. 7.4.1.3).

ALLOCATION ATTRIBUTES

• Length	Section length; initial allocation for extensible sections.
• Maximum length	Maximum length for extensible sections.
• Alignment	Byte alignment of section; section is allocated starting at a 0 MOD alignment byte address.

7.4.1.3 Binding sections

The binding section may be thought of as a special class of working storage section that is administered by the Operating

System. There may be only one binding section per module just as there may be only one code section. Binding sections are allocated in a segment that has the read and binding protection attributes and does not have the write attribute in user rings. Furthermore only the Operating System may INITIATE a segment with a binding attribute; users may not.

In order to insure the efficacy of the binding section, several conventions concerning it must be adhered to by all modules executed in IPLOS. These conventions have been established to achieve the following ends:

1.) Assure the integrity of crossing ring of protection boundaries; one of the requirements of protecting one piece of code from another is that the protected code only be entered from points at which it is prepared to receive control. Branching to arbitrary entry points within a piece of code could cause undefined, possibly destructive results.
2.) Allow the binding sections of several modules to be combined at library generation time in such a fashion that:
   a.) Further processing of matching entry-external references at load time is eliminated in some cases.
   b.) Redundant external references are removed thereby reducing the overall size of the combined binding section of the new module.
3.) Provide a consistant mechanism for all pure procedure code (user and system) to discover the data base associated with the appropriate instance of execution.

The conventions associated with the binding section are as follows:

1.) Only the Operating System may INITIATE a segment with the binding protection attribute.
2.) The binding section is readable but not writable in the users ring of protection.
3.) The only data that may be stored in the binding section are pointers, and they must be in one of the three following forms:
   a.) Forty eight bit data pointer, right justified in a full word with the two unused bytes zero filled; aligned in a full word.
   b.) A sixty four bit internal procedure code base pointer; aligned in a full word
   c.) A 128 bit external procedure code base-binding section pointer pair; aligned in two full words.

   Despite the fact that the binding section is readable in the

user ring, placement of constant data is explicitly disallowed to prevent generation of erroneous entry points to more privileged rings under false pretenses.
4.) The only data that must be stored in the binding section (i.e. hardware requirement) are internal and external procedure descriptors (i.e. 3b and c above).
5.) The binding section must be unstructured; that is no predetermined order can be assumed between binding section entries since a given entry's relative location within the binding section may change independently of any other entry during library generation. This implies that address arithmetic (indexing) or assumptions about pointer contiguity are not permitted with regard to the binding section.
6.) The Operating System/hardware calling convention only provides a procedure with the address of its binding section. This implies that the base address of at least one of the module's working storage sections must be stored in the binding section.

7.5 SECTION - SEGMENT ALLOCATION

Table 7-3 summarizes the segment allocation that takes place for each type of module section when a program is established as a task.

SECTION	SEGMENT ALLOCATION
CODE	.one segment per ring for the code sections of all the object modules in the object file list .one segment per system for each library in the library segment list
WORKING STORAGE	.one segment per ring per access attribute set in which all non-extensible working storage sections and common blocks are allocated .one segment for each extensible working storage section and common block
BINDING	.one segment for all binding sections of all the modules in a task

TABLE 7-3
Module-segment allocation

7.6 MODULE CONVENTIONS

The conventions to which all modules (object and load) must conform in order to be processed by the loader and library generator are summarized below:

- Object modules must be generated in the following order: IDR; all SDCs; any combination of TEX,RPL,BIT,EPT,RIF,AIN,AOI, and XRL records; TRA.
- Object module records are processed as they are read by the loader and library generator. Overlapping records are not detected.
- Each module may contain at most one code section and one binding section but an arbitrary number of working storage sections and common blocks.
- Section definition ordinals in every module must start at zero and be numbered consecutively.
- The code section of every module must be "pure" to allow it to be shared with all activations of the module.
- The binding section of every module must adhere to the following conventions:
  - Only the O.S. may INITIATE a segment with the binding attribute.
  - The binding section is readable and <u>not</u> writable in the user's ring
  - Only right justified, word aligned pointers and procedure descriptors may be stored in the binding section; constant data is explicitly <u>disallowed</u> .
  - Only procedure descriptors <u>must</u> be stored in the binding section.
  - The binding section must be unstructured; that is no order may be presumed to exist among binding section entries.
  - The O.S./hardware calling convention only supplies a called procedure with the base address of its binding section; the base addresses of all working storage sections must be bootstrapped from the binding section.
- Code section protection attributes are read and execute.
- FORTRAN blank common should be an extensible common block with a name of all blanks.
- Entry-externals and common blocks having the same names are specifically allowed.
- For dynamic linking - address arithmetic instruction sequences using external addresses must load the pointer to the external address into an A register before performing any computation to allow the dynamic linking fault to be processed correctly.

7.7 DETAILED MODULE FORMATS

A detailed listing of the current version of the type definitions of the object module is available in catalog GSB in an ASCII list on file OBJTEXT and in the output listing of ISWL on the file OBJSWL.

A detailed listing of the load module is not currently available.

8.0 ACCESS METHODS

8.0 ACCESS METHODS

To be supplied

9.0 INSTRUMENTATION

To be supplied