Date: 10 April 75

To: Distribution

From: M. D. MacLaren

Subject: Converting from ios_ to iox_

This MTB contains a draft of a proposed cookbook for conversion from ios_ to iox_. Not all the recipes are kitchen tested. It is planned to make the information in this document available to users.

This document contains information relevant to converting programs that currently call the old I/O system (ios_) so that they will call only the new I/O system (iox_). The reader may also wish to consult the following documents:

1) Interim supplement to Multics Programmers' Manual for MR 2.1.

2) MTB-136, tape_mult_ I/O Module.

Comments on this document should be addressed to Don MacLaren; information about the use of iox_ itself can be obtained from Steve Herbst; information about individual I/O modules can be obtained from their authors.

Note that no conversion is required for programs that do all their i/o through the i/o facilities in a programming language and/or subroutines such as ioa_. Only programs that directly call ios_ need be converted.

For purposes of converting to iox_, dims (device interface modules) supported through ios_ may be divided into three classes:

1) dims for which the Multics system contains a compatible iox-type I/O module. Conversions involving these are covered in this document.

2) dims for tape I/O. For conversions involving such dims another document will be available sometime in the

future.

3) all other dims. When a program uses a dim in this class, the dim must be replaced by an iox-type I/O module, and the program must be converted to call this module.

"Read_ptr and Write_ptr"

Programs that use only ios_$read_ptr and ios_$write_ptr are an especially simple case.

The calls:

```
call ios_$write_ptr(buff_ptr, 0, buff_len);
call ios_$read_ptr(buff_ptr, buff_len, nelemt);
```

should be mapped into:

```
call iox_$put_chars(iox_$user_output, buff_ptr, buff_len,
     code); /* replaces call to write_ptr */
call iox_$get_line(iox_$user_input, buff_ptr, buff_len,
     nelemt, code); /* replaces call to read_ptr*/
```

The applicable declarations are:

```
dcl iox_$user_output external ptr;
dcl iox_$user_input external ptr;
dcl buff_ptr ptr, code fixed bin(35);
dcl (buff_len, nelemt) fixed bin(21);
dcl iox_$put_chars entry(ptr, ptr, fixed bin(21),
     fixed bin(35));
dcl iox_$get_line entry (ptr, ptr, fixed bin (21),
     fixed bin(21), fixed bin(35));
```

The argument code is a standard Multics status code, not a 72-bit status code as with ios_. The code should be tested. A non-zero code indicates an error, except that for a call to iox_$get_line, the code error_table_$long_record is returned if the buffer is filled without encountering a read delimiter. Note that nelemt (the number of characters actually read into the buffer) must be redeclared as "fixed bin(21)" to match the parameter descriptor in iox_$get_line.

The mapping above does not cover the case where a call to write_ptr specifies a nonzero offset in the buffer. Suppose we have:

```
dcl offset fixed bin;
     /*other declarations as above*/
call ios_$write_ptr(buff_ptr, offset, buff_len);
```

The call should be replaced by:

```
dcl buffer(0:1048575) char(1) based; /*01048575
     is the longest possible buffer*/
call iox_$put_chars (iox_$user_output,
     addr(buff_ptr-> buffer(offset)), buff_len, code);
```

Switches, Streams and Control Blocks

In the terminology used for iox_, an ios_ stream is called an I/O switch. Each I/O switch has an associated control block (called an iocb for short), and most calls to iox_ pass a pointer to the iocb to specify the switch that is the source/target for input/output. Given a switchname (i.e., a stream name in ios terminology), a pointer to the iocb may be obtained by calling iox_$find_iocb. For example:

```
declare iox_$find_iocb entry(char(*), ptr, fixed bin(35));
call iox_$find_iocb("foo", iocb_ptr, code);
call iox_$put_chars(iocb_ptr, buff_ptr, 20, code);
```

writes 20 characters from a buffer (pointed to by buff_ptr) through the switch named "foo". The routine iox_$put_chars consists of a few instructions that transfer to the actual I/O routine through a transfer vector in the iocb. In most programs, only one call is needed to iox_$find_iocb to cover a sequence of I/O requests. Thus repeated table lookups on the switchname are avoided.

External pointers are provided for the standard switches, so there is no need to call iox_$find_iocb when using these switches.

```
declare iox_$user_io ptr external; /*user_i/o*/
declare iox_$user_input ptr external; /*user_input*/
declare iox_$user_output ptr external; /*user_output*/
declare iox_$error_output ptr external; /*error_output*/
```

Switches and ioa

There are two new entries in ioa_ that take a pointer to an iocb.

```
dcl ioa_$ioa_switch entry options (variable);
dcl ioa_$ioa_switch_nnl entry options (variable);
```

```
call ioa_$ioa_switch (iocb_ptr, control_string,
          argl,...argn);
call ioa_$ioa_switch_nnl (iocb_ptr, control_string,
          argl,...argn);
```

Except for taking an iocb pointer rather than a switch/name (stream name), these entries do the same things as ioa_$ioa_stream and ioa_$ioa_stream_nnl, respectively.

Compatible Dims

The following dims have compatible iox-type I/O modules: syn, tw_, ntw_, absentee_dim_, mrd_, oc_, tek_, exec_com_, and discard_output_. For these dims there is a simple mapping of calls to ios_ into calls to iox_. Such a mapping also exists for file_ in cases where it: 1) is used with default delimiters and default element sizes, 2) reading and writing are not both done in a single attachment, and 3) ios_$seek and ios_$tell are not used.

The mapping of a call to ios_$attach depends on the dim as is explained in the next section of this info segment. All other calls map independently of the dim involved (except for detaching a syn attach).

Attachment

The call:

```
call ios_$attach (stream_name, dim_name, device,
     modes_1, status);
```

should be mapped into the following calls
(not all used in all cases)

```
call iox_$attach_ioname (stream_name,
     iocb_ptr, attach_descrip, code);
call iox_$open (iocb_ptr, opening_mode, "0"b, code);
call iox_$modes (iocb_ptr, modes_2, "", code);
```

where:

1) attach_descrip is a character string depending on the dim. It is described below.

2) iocb_ptr points to the I/O control block. It is set by the call to iox_$attach_ioname and is used by all other calls.

3) opening_mode is an integer specifying the use of the attachment as follows:

opening_mode = 1, stream_input, which corresponds to the ios mode "read".
opening_mode = 2, stream_output, which corresponds to the ios mode "write".
opening_mode = 3, stream_input_output, which corresponds to the ios modes "read" and "write".

4) code is a standard Multics status code.

5) modes_2 is a string containing those modes other than "read" and/or "write" that were specified by modes_1 in the call to ios_$attach.

Notes on Attach Calls

1) The call to iox_$open is omitted when the dim involved is syn.

2) The call to iox_$modes is made only when modes_1 specifies modes other than "read" and/or "write".

3) If the dim is file_, opening mode = 3 is not allowed.

4) If the iocb_ptr is already known, the call to iox_$attach_ioname may be replaced by:

```
call iox_$attach_iocb(iocb_ptr, attach_descrip, code);
```

Declarations for Attach Calls

```
declare iox_$attach_ioname entry(char(*), ptr,
     char(*), fixed bin(35));
declare iox_$attach_iocb entry(ptr, char(*), fixed bin(35));
declare iox_$open entry(ptr, fixed bin, bit(1) aligned,
     fixed bin(35));
declare iox_$modes entry(ptr, char(*),
     char(*), fixed bin(35));
```

Attach Descriptions

The attach descriptions to be used in the calls to iox_$attach_ioname are now given in the form dim_name corresponding_attach_description. Note that "device" is the string that is an argument to ios_$attach.

```
syn                 syn_
tw_                 tty_        device
ntw_                netd_       device
absentee_dim_       abs_        device
mrd_                mr_         device
oc_                 ocd_        device
tek_                tekd_       device
exec_com_           ec_         device
discard_output_     discard_
file_               vfile_      device      -extend
```

For example:

```
call ios_$attach ("foo", "file_", "my_file",
     "read", status);
```

should be mapped into:

```
call iox_$attach_ioname ("foo", iocb_ptr,
     "vfile_, my_file -extend", code);
call iox_$open (iocb_ptr,1, "o"b, code);
```

Mapping Other Calls

The remainder of this info segment explains how the other calls to ios_ (for compatible dims) are mapped into calls to iox_. The declarations of buff_ptr, buff_len, nelemt, and code are as given above under "Read_ptr and Write_ptr". The argument iocb_ptr is a pointer to the I/O control block for the stream_name given in the calls to ios_.

The calls:

```
call ios_$read(stream_name, buff_ptr, 0,
     buff_len, nelemt, status);
call ios_$write(stream_name, buff_ptr, 0,
     buff_len, nelemt, status);
```

should be mapped into:

```
call iox_$get_line(iocb_ptr, buff_ptr, buff_len,
     nelemt, code);
call iox_$put_chars(iocb_ptr, buff_ptr, buff_len,
     code);
```

If a nonzero offset is given as the third argument of the call to ios_, the method given for ios_$write_ptr may be used to pass the correct buff_ptr to iox_.

The call:

```
call ios_$detach(stream_name, device, disposal, status);
```

should be mapped into:

```
call iox_$close(iocb_ptr, code);
call iox_$detach_iocb(iocb_ptr, code);
```

The call to iox_$close is omitted when the dim involved is syn.

The declarations for the above calls are:

```
declare iox_$get_line entry(ptr, ptr, fixed bin(21),
     fixed bin(21), fixed bin(35));
declare iox_$put_chars entry (ptr, ptr,
     fixed bin(21), fixed bin(35));
declare iox_$close entry(ptr, fixed bin(35));
declare iox_$detach entry(ptr, fixed bin(35));
```

The calls:

```
call ios_$resetread(stream_name, status);
call ios_$resetwrite(stream_name, status);
call ios_$abort(stream_name, ""b, status);
call ios_$order(stream_name, order, info_ptr, status);
```

should be changed to, respectively,

```
call iox_$control(iocb_ptr, "resetread", null, code);
call iox_$control(iocb_ptr, "resetwrite", null, code);
call iox_$control(iocb_ptr, "abort", null, code);
call iox_$control(iocb_ptr, order, info_ptr, code);
```

The call:

```
call ios_$changemode(stream_name, new_modes,
     old_modes, status)
```

should be changed to:

```
call iox_$modes(iocb_ptr, new_modes, old_modes, code);
```

The declarations for the above calls are:

```
declare iox_$control entry(ptr, char(*), ptr,
     fixed bin(35));
declare iox_$modes entry(ptr, char(*), char(*),
     fixed bin(35));
```