

7.1.6 Linking a C Program that uses the ALIGN Option

When a source file is compiled with the ALIGN option, it is important that

- all source files that are linked into the program are either alignment independent, or also compiled with the ALIGN option, and
- the program is linked with the aligned run-time library

The aligned library is accessed by specifying 'WCOS.CLIB.OBJALIGN' as the first entry in the SYSLIB DD, as follows:

```
//SYSLIB DD DSN=WCOS.CLIB.OBJALIGN,DISP=SHR
// DD DSN=other C libraries
```

There are five object libraries that comprise the Waterloo C run-time library (not including the Waterloo C Panel Library). In the case where a reentrant, aligned program is linked with the RMODE ANY pre-linked library, all five libraries can be concatenated in the following order:

```
//SYSLIB DD DSN=WCOS.CLIB.OBJALIGN,DISP=SHR
// DD DSN=WCOS.CLIB.OBJRENT,DISP=SHR
// DD DSN=WCOS.CLIB.OBJVECSA,DISP=SHR
// DD DSN=WCOS.CLIB.OBJVEC,DISP=SHR
// DD DSN=WCOS.CLIB.OBJ,DISP=SHR
```

'WCOS.CLIB.OBJALIGN' must always be specified before the other libraries, or the program's behaviour may be undefined.

7.2 Waterloo C Run-time Conventions

The following sections describe several aspects of the Waterloo C run-time environment and how they can be modified according to the specific requirements of a particular application.

7.2.1 Command-line Parameters and the Program Return Code

C programs that use the usual Waterloo C run-time environment should define a function `main()` which the library startup code calls when the program is run. As part of the library initialization procedure, command line arguments are converted to a conventional `argc/argv` style, depending on how the program was run, and subject to the value of the external variable `_parms`. The integer value that is returned by `main()` is returned to the system as the program's return code.

There are two parameter formats that the C run-time environment can process. When a program is started with the TSO CALL or TEST commands, or if it is run in a batch environment, a standard OS parameter format is received. If the program is started as a TSO command processor, the program is passed a command processor parameter list (CPPL). The library initialization code determines which parameter format has been received and formats the arguments into the C argument style.

Several variations of command formatting are available and can be selected by defining the initialized value of the constant extern symbol `_parms`. The possible values of this symbol are defined in the library header file `<setup.h>`. Note that the value `NO_REDIRECTION` is a mask value that can be included with any of the other formats to suppress command line redirection. Only one of `EXTEND`, `UNTOKENIZED`, `EXTEND_NOPARENS` and `C_STRINGS` should be specified. The following is a summary of the functions performed by the library initialization for each format value:

UNTOKENIZED

The entire argument string is passed to the program as `argv[0]` and `argc` is always 1. If the program was not passed a CPPL, a command name of 'CALL ' is prefixed to the argument string. This facilitates consistent processing of untokenized format arguments, regardless of how a program is run.

EXTEND

Blanks and parentheses (' ', '(' and ')') are considered to be token

Reference

delimiters. Blanks are removed and parentheses are passed as separate entries in the `argv` vector. Note that this format is provided for compatibility with the "extended" parameter list format provided in the VM/CMS environment.

EXTEND_NOPARENS

The program argument is tokenized in a manner similar to the tokenization for `EXTEND`, with the exception that only blanks delimit tokens. This is the normal library default processing, as defined in the library source file `'WCOS.CLIB.C($PARMS)'`.

C_STRINGS

For this style, the program argument is tokenized as for `EXTEND`, with several extensions. A backslash character (`\`) in the argument string is processed as for a backslash in a C character constant. The following backslash sequences are recognized:

`\0xh, \xhh, ...`

denote a single character with an EBCDIC character code of `0x0h, 0xhh`, etc. ('h' is a hexadecimal digit); the first non-hexadecimal character terminates the character constant

`\o, \oo, \ooo`

denote a single character with an EBCDIC character code of `00o, 0oo` or `ooo` ('o' is an octal digit)

`\a, \b, \f, \n, \r, \t` and `\v`

denote the equivalent C character constant

`\c`

denotes the character 'c', such as a quotation mark (for use within quoted tokens), or the backslash character

Quotation marks (") are treated as explicit tokenization directives, which allow imbedded blanks to be included in a token. Two adjacent quotes within a string are interpreted as a string concatenation operation, and are ignored. Trigraph sequences, as recognized by the C compiler, are replaced with their single character equivalent.

For all tokenization styles, the `main()` function is passed a number of command line arguments (`argc`) and a pointer to a list of character string pointers (`argv`) that comprise the text of the command line. In all cases, `argv[argc]` is `NULL` and,

with the exception of the UNTOKENIZED style, if the command that was used to run the program is unknown, `argv[0]` is the empty string ("").

The following program, 'WCOS.CLIB.C(TARGS)', can be used to test command line processing.

```
#include <setup.h>
#include <stdio.h>

/* const int _parms = C_STRINGS; */
const int _parms = UNTOKENIZED;

int main( int   argc,
          char **argv )
{
    int count;

    printf( "argc = %d\n", argc );
    for( count = 0; count < argc; ++count )
        printf( " argv[ %d ] = \"%s\"\n", count, *argv++ );
    printf( " argv[ %d ] = 0x%08x\n", argc, *argv );
    return( 0 );
}
```

In this example, the string array argument is defined as `char **argv`. An equivalent definition that is often used is `char *argv[]`. The former definition is used throughout this manual, although the latter can be used instead.

If 'WCOS.EXAMPLE.LOAD(TARGS)' contains an executable version of this program, it is possible to experiment with various command line arguments and processing styles. For example, the output from a typical TSO CALL command would be the following:

```
call example.load(targs) ' Command-line arguments '
argc = 1
argv[ 0 ] = "CALL  COMMAND-LINE  ARGUMENTS  "
argv[ 1 ] = 0x00000000
READY
```

Note that the TSO CALL command converts the arguments to upper case. If this program is run from JCL, mixed case arguments are available:

Reference

```
//RUN EXEC PGM=TARGS,PARM=' Hello, there. ',REGION=256K
. . .
argc = 1
argv[ 0 ] = "CALL Hello, there. "
argv[ 1 ] = 0x00000000
```

If the same program is run as a command processor using the TSO TEST command, mixed case command arguments are available:

```
test example.load(targs) cp
ENTER COMMAND FOR CP
  targs  Command-line  arguments
argc = 1
argv[ 0 ] = "  targs  Command-line  arguments"
argv[ 1 ] = 0x00000000
READY
```

7.2.2 Initialization

In all of the linking configurations where the standard C run-time library is used, the program's `main()` function will eventually be given control when the program is run. So that the linking procedure can resolve the call from the library into the program, `main()` should be defined in a manner similar to the following:

```
int main( int  argc,
          char **argv )
{
    /* a useful program */
    return( 0 );
}
```

In particular, `main()` must not be defined with the `static` attribute; its scope must be known outside the source file.

Before the application is executed, the Waterloo C run-time environment must be set up. Execution begins in the library at the entry point `$$STARTUP` that is contained in the source file `'WCOS.CLIB.ASM(STARTUP)'`. From this point, the following functions are performed by the library initialization procedure:

- memory is allocated from the system for the run-time stack, the library internal read-write data area and the global variable (AUX) data area
- an optional, pre-linked run-time library is loaded from a system link area, the job library (JOBLIB DD), or the step library (STEPLIB DD) for the program
- if the program is running above 16M on an MVS/XA system, the 24-bit mode dependent portion of the library is relocated
- pre-main program configuration parameters, such as the initial value of the I/O error message display flag, and the default standard file name definitions are set
- library internal read/write data is initialized, in preparation for calls to functions such as `rand` and `strtok`.
- the initial values for reentrant (read/write) data are set

Reference

- system-level signal handlers and exception stacks are allocated; default signal handlers are defined; program check handling is enabled
- system-level file I/O blocks are allocated and initialized
- device drivers for the "_NULL" and "_TERMINAL" devices are defined
- the program arguments are processed, subject to a configuration parameter
- the standard input, output and error stream files (`stdin`, `stdout` and `stderr`) are opened, optionally subject to command line redirection
- `main()` is called

As complicated as this sequence appears, each function has been carefully designed and optimized for both flexibility and efficiency.

7.2.3 File Support

Waterloo C provides three general techniques for directing input/output operations to a system device. If a standard OS DD name is specified, an OPEN using that DD is executed. If an MVS data set name is specified, a DD name is dynamically allocated for that data set and the file is processed in the same manner as if the DD is specified. Alternatively, a device driver name can be specified. For the current version of Waterloo C, two device names are provided: "_TERMINAL." and "_NULL".

Specifying a file by DD name

The use of an OS DD name allows access to the file represented by the DD name. The format for opening a file specified by a DD name is

```
ddname ( options
```

A left parenthesis '(' is used to introduce the file options when needed.

Recognized options are:

RECFM F	set the MVS record format to "fixed", which must agree with the file format
RECFM V	set the MVS record format to "variable", which must agree with the file format.
LRECL nnn	set the MVS record length to the value nnn, which must agree with the file's logical record length.

Specifying a file by Data Set Name

An MVS data set may be accessed by specifying the following:

```
dsname(member) ( options
```

The member name is optional and the '(' is used to introduce any needed options. A fully qualified data set name can be specified by enclosing the name in single quotes.

Reference

The options recognized are:

RECFM options

The options consist of a series of one letter format specifiers:

- F set the MVS record format to "fixed"
- V set the MVS record format to "variable"
- B set the MVS record format to "blocked"
- M indicates the presence of machine control characters in the MVS record
- A the first character of each record is an ANSI carriage control character
- S sets the MVS record format to "spanned"

LRECL nnn

sets the MVS record length to the length nnn

BLKSIZE nnn

sets the MVS block size to the length nnn

DSORG dsorg

where dsorg is PS for a sequential file or PO for a partitioned data set.

SPACE type, primary { , secondary } { , dirblocks }

specifies the amount of space to allocate for the data set; type determines the unit of space in which the data set will be allocated. If it is numeric, it specifies an average block size for unit of allocation space. The other valid options are TRK for tracks or CYL for cylinders as the allocation unit. The number of space units of type in the primary and secondary extents is given by the primary and secondary parameters. The dirblocks parameter allocates that number of directory blocks for a partitioned data set.

```
DISP status {, normdisp } {, conddisp }
```

specifies the disposition of the data set. `status` is one of OLD, MOD, NEW, or SHR. `normdisp` specifies the data set disposition if the job terminates normally. It can be one of CATLG, UNCATLG, DELETE, or KEEP. `conddisp` specifies the data set disposition the same way as `normdisp` should the job end abnormally.

If a parameter is not specified, the following defaults are used:

- If the data set has a disposition of NEW or does not exist, the record format defaults to fixed length, 80 character records. The block size defaults to the record size. The data set organization defaults to partitioned if a member is specified, and sequential otherwise. Any new data set will be cataloged. When no space specification is given the primary and secondary extents will each be allocated one unit of the type specified. When no type is specified the data set will be allocated in tracks.
- If the status of the data set is not specified, then the data set is created if it does not exist, using the above defaults. If the data set is opened for append, the disposition defaults to MOD; otherwise, for read the default is SHR, and otherwise OLD. A partitioned data set may not be created without specifying a member name and a member must exist if it is opened for read.

If a data set cannot be allocated successfully, an error message giving a "dynamic allocation" error code may be issued. These error codes are listed in the *System Programming Library: Job Management* (GC28-0627).

Specifying a file by Device Driver Name

`_NULL` When this file name is used, any output to the file is ignored, and any attempt to read from the file results in an end-of-file condition.

`_TERMINAL`.

This device driver is intended primarily for use for the standard input, output and error files, but it can be used by any application program for

Reference

general purpose, mixed case, terminal input/output as well. The general form of a file name using this device driver name is the following:

```
__TERMINAL.ddname
```

When the file is opened, if a DD name is provided and has been defined, input/output will be directed to that DD using the usual DD file access. If no DD is specified, or if it has not been defined, input/output operations will be directed to the TSO terminal using TGET and TPUT SVC's. If the program is not executing in the TSO foreground, the file open operation will fail.

For example, the normal standard input, output and error files are opened with calls similar to the following:

```
fopen( "__TERMINAL.STDIN", "r" );  
fopen( "__TERMINAL.STDOUT", "w" );  
fopen( "__TERMINAL.STDERR", "w" );
```

For each file, a C program that is run as a TSO foreground program does not need to have DD's for STDIN, STDOUT or STDERR, but if they are defined, they will be used. When a C program is run from batch, the DD's must be specified. The library standard file names are defined in 'WCOS.CLIB.C (\$STDIOFN) '.

Common Options

Several options are common to all methods of accessing a file:

- BIN** The newline character (`\n`) has no special significance when character input/output functions are used (`fputc`, `fgetc`, `fgets`, `fputs`, `fprintf`, `fscanf`, etc.).

- TEXT** The newline character is used to signify the end of a record. For fixed format files, the remaining portion of the record is padded with blanks. When files are read using the character input/output functions, a newline is returned as the last character in each record.

- CC For a printer file, the first character of each record is a carriage control character. If this option is not specified, a blank is inserted at the start of each record.
- RAW The system-level input/output functions `read` and `write` operate on a single record for each call. (`read` will read at most one record, no matter what count is specified.) This allows programs to take advantage of the record structure of disk files.
- DIRECT allows random access to a sequential file, but not to a partitioned data set member. The file must have fixed length records. Each block is considered a record because no deblocking is done. New records may not be created using this method but may be read and updated.

For compatibility with previous versions of Waterloo C, the special filename `terminal` is recognized to allow input/output to a TSO terminal. All data is converted to upper case.

For example,

```
fopen( "TEST.FILE", "w" )
```

specifies a file named 'TEST.FILE' and creates it with fixed format records and a record length of 80, if the file does not already exist.

```
fopen( "XYZ (BIN LRECL 1024 RECFM F", "w" )
```

specifies a file with a DD name of "XYZ" which must have fixed format records and a length of 1024 bytes.

Standard File Redirection

Programs developed using Waterloo C support file redirection from the command line of the stream files `stdin` and `stdout`.

If the characters '<' or '>' are found, the rest of the command line is taken to be a file name, and the appropriate standard I/O stream is associated with that file. Thus, the line

```
PROG Parm <INFILE >OUTFILE (RECFM F LRECL 132
```

Reference

would associate stdin with the file INFILE, stdout with OUTFILE (RECFM F LRECL 132 and pass main a parameter list consisting of the two strings "PROG" and "Parm").

7.2.4 Run-time Stack

The size of the run-time stack is 10K bytes by default. All auto variables, parameters and register save areas for a C program are allocated from this stack, which is defined before the program's main function is called. (A C program can determine how much stack space is available by calling the `stakleft` function, which is described in the *Waterloo C Run-time Library Reference*.) The stack size can be modified by changing the initialized value of the external variable `_staksize` (defined in the library source file `'WCOS.CLIB.C($STAKSIZ)'`).

The following example program, `BIGSTACK`, defines a run-time stack of 50Kb:

```
#include <stdio.h>

/* BIGSTACK -- a program with a large stack */

const int _staksize = (50*1024);

int main()
{
    printf( "hello, world\n" );
    return( 0 );
}
```