NAME

gcobol — GCC COBOL Front-end I/O function API

LIBRARY

libgcobol

SYNOPSIS

#include <symbols.h>
#include <io.h>
#include <gcobolio.h>

gcobol_io_t
gcobol_fileops();

```
class gcobol_io_t {
public:
  static const char constexpr marquee[64];
  typedef void (open_t)( cblc_file_t *file,
                         char *filename,
                         int mode_char,
                         int is_quoted );
  typedef void (close_t)( cblc_file_t *file,
                          int how );
  typedef void (start_t)( cblc_file_t *file,
                          int relop, // needs enum
                          int first_last_key,
                          size_t length );
  typedef void (read_t)( cblc_file_t *file,
                         int where );
  typedef void (write_t)( cblc_file_t *file,
                          unsigned char  *data,
                          size_t length,
                          int after,
                          int lines,
                          int is_random );
  typedef void (rewrite_t)( cblc_file_t *file,
                            size_t length, bool is_random );
  typedef void (delete_t)( cblc_file_t *file,
                          bool is_random );
  open_t      *Open;
  close_t     *Close;
  start_t     *Start;
  read_t      *Read;
  write_t     *Write;
  rewrite_t   *Rewrite;
  delete_t    *Delete;
  ...
};
```

DESCRIPTION

gcobol supplies replaceable I/O functionality via **gcobol_fileops**(). It returns a pointer to a structure of C function pointers that implement sequential, relative, and indexed file operations over files whose On Disk Format (ODF) is defined by **gcobol**. A user wishing to use another library that implements the same functionality over a different ODF must supply a different implementation of **gcobol_fileops**(), plus 7 functions, as described in this document. The pointers to those user-implemented functions are placed in a C++ object of type *gcobol_io_t* and an instantiation of that type is returned by **gcobol_fileops**(). The compiled program initializes I/O operations by calling that function the first time any file is opened.

Each function takes as its first argument a pointer to a *`cblc_file_t`* object, which is analogous to a *`FILE`* object used in the C **stdio** functions. The *`cblc_file_t`* structure acts as a communication area between the compiled program and the I/O library. Any information needed about the file is kept there. Notably, the outcome of any operation is stored in that structure in the *file_status* member, not as a return code. Information about the *operation* (as opposed to the *file*) appear as parameters to the function.

`cblc_file_t` has one member, not used by **`gcobol`**, that is reserved for the user:
`void *` *implementation*.

User-supplied I/O functions may assign and dereference *implementation*. **`gcobol`** will preserve the value, but never references it.

The 7 function pointers in *`gcobol_io_t`* are

Open *`void`* **`open_t`**(*`cblc_file_t *file`*, *`char *filename`*, *`int mode_char`*, *`int is_quoted`*)
parameters:

`filename` is the filename, as known to the OS

`mode_char` is one of

‘r’	OPEN INPUT: read-only mode
‘w’	OPEN OUTPUT: create a new file or overwrite an existing one
‘a’	EXTEND: append to sequential file
‘+’	modify existing file

`is_quoted` If **true**, *`filename`* is taken literally. If**false**, *`filename`* is interpreted as the name of an environment variable, the contents of which, if extant, are taken as the name of the file to be opened. If no such variable exists, then *`filename`* is used verbatim.

Close *`void`* **`close_t`**(*`cblc_file_t *file`*, *`int how`*)
parameters:

`how` A value of 0x08 closes a “REEL unit”. Because no such thing is supported, the function sets the file status to “07”, meaning *not a tape*.

Start *`void`* **`start_t`**(*`cblc_file_t *file`*, *`int relop`*, *`int first_last_key`*, *`size_t length`*)
parameters:

`relop` is one of

`0`	means ‘<’
`1`	means ‘<=’
`2`	means ‘=’
`3`	means ‘!=’
`4`	means ‘>=’
`5`	means ‘>’

`first_last_key`
is the key number (starting at 1) of the key within the *`cblc_file_t`* structure.

`length` is the size of the key (TODO: per the START statement?)

Read *`void`* **`read_t`**(*`cblc_file_t *file`*, *`int where`*) parameters:
`where`

`-2`	PREVIOUS
`-1`	NEXT

`N` represents a key number, starting with 1, in the *`cblc_file_t`* structure. The value of that key is used to find the record, and read it.

Write *`void`* **`write_t`**(*`cblc_file_t *file`*, *`unsigned char *data`*, *`size_t length`*, *`int after`*, *`int lines`*, *`int is_random`*)
parameters:

data	address of in-memory buffer to write
length	length of in-memory buffer to write
after	has the value 1 if the AFTER ADVANCING n LINES phrase was present in the **WRITE** statement, else 0
lines	may be one of `-666` ADVANCING PAGE `-1` no **ADVANCING** phrase appeared 0 ADVANCING 0 LINES is valid >0 the value of *n* in ADVANCING *n* LINES
is_random	is **true** if the *access mode* is RANDOM

Rewrite *void* **rewrite_t**(*cblc_file_t *file*, *size_t length*, *bool is_random*) parameters:

length	number of bytes to write
is_random	**true** if *access mode* is RANDOM

Delete *void* **delete_t**(*cblc_file_t *file*, *bool is_random*) parameters:

is_random	**true** if *access mode* is RANDOM

The library implements one function that the **gcobol**-produced binary calls directly:

*gcobol_io_t ** **gcobol_fileops**()
This function populates a *gcobol_io_t* object with the above function pointers. The compiled binary begins by calling **gcobol_fileops**(), and then uses the supplied pointers to effect I/O.

RETURN VALUES

I/O functions return **void**. **gcobol_fileops**() returns *gcobol_io_t**.

STANDARDS

The I/O library supplied by **gcobol**, **libgcobolio.so**, supports the I/O semantics defined by ISO COBOL. It is not intended to be compatible with any other ODF. That is, **libgcobolio.so** cannot be used to exchange data with any other COBOL implementation.

The purpose of the *gcobol_io_t* structure is to allow the use of other I/O implementations with other ODF representations.

CAVEATS

The library is not well tested, not least because it is not implemented.

BUGS

The future is yet to come.