Return Value Pattern

The return value pattern standardizes the way we communicate with the caller through return values.

Since the C programming language does not have exceptions, return values are our standard way to communicate a result of an operation.

We must therefore have a standardized set of return codes and we must follow a clear pattern with our return values to make sure that we can have clear expectations of how a function would behave and what value it will return in case of an error.

In this module we are going to standardize our return values.

Definition

- **Signed integer return value**: we use standard way of reporting status by returning a status value as a signed int in C.
- **Zero means no error**: a zero return value shall always mean that there was no status to return - meaning that the operation has been successful.
- **Negative value means error**: a negative status means that something negative has happened and the value of the negative number that is returned shall specify what error has occurred.
- **Positive value means status**: a positive status means that the operation was at least partially successful and the return code shall be function specific success return code. For functions that read and write data, this value could for example mean number of items successfully written or read.

Use Cases

- **Every non-trivial function**: standardized return values should be used at all times when a function does any operation that may fail or which can produce different status.
- **Every function that can fail**: any function that can fail must return a negative error code upon failure condition.
- **Every function that has status to report**: any function that may succeed in at least two ways.

Benefits

- **Standardized expectations**: we can have standard expectations about every function that returns a status code. We don’t need to open documentation each time.
- **Clean code**: our code can be clean and easy to understand.

Drawbacks

- **Pressure on programmers**: this pattern puts extra pressure on programmer to write clean code. Return values are often not understood and much less often checked. If you are using tools like PC-Lint you will be inclined to check all return values because otherwise it would be a lint error - but for programmers who are not used to top quality programming this can be a difficult pattern to follow.

- **Longer code**: since we always check return values we need additional 'if' statements and blocks which make our code longer. There is no good way around this.
- **Limited level of detail**: sometimes we have to return status codes that do not accurately describe the actual error since status codes are generic. However, in most cases this is not an issue since we simply treat a negative code as a generic error.

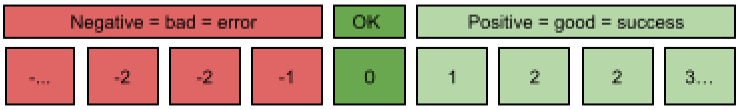
Implementation

We implement the return value pattern in C in the following way:

- **Every non-trivial function must return integer status code**: C does not support exceptions with stack unwinding like C++ so the easiest way is to pass status code through return value.
- **Return values must be checked**: every call of a function that has a return value must be checked when the function is called.
- **Error codes are always constant**: upon getting an error code, we try to handle the error and if we can not handle it we always return a well defined error code to the caller.

```
int my_object_method(struct my_object 'self){
    if(there_is_error){
        return -ERRNOCODE;
    }
    return 0;
}
```

Error convention



- **Negative error codes correspond to ERRNO codes**: we always use the errno.h for negative error codes.
- **Zero always means success**: returning a zero status code always means operation completed without issues.
- **Positive error codes are function specific**: the meaning of positive values must always be documented in documentation of the function that returns them.

Standard Error Codes

Below is a table of existing errno codes and their possible uses. The meanings of each errno code on this list can be further defined and described in more detail to indicate what kind of condition that particular errno code can indicate in the context of the application:

Table 1. ERRNO codes

EPERM	Not owner
ENOENT	No such file or directory
ESRCH	No such context
EINTR	Interrupted system call
EIO	I/O error
ENXIO	No such device or address
E2BIG	Arg list too long
ENOEXEC	Exec format error
EBADF	Bad file number
ECHILD	No children
EAGAIN	No more contexts
ENOMEM	Not enough core
EACCES	Permission denied
EFAULT	Bad address
ENOTBLK	Block device required
EBUSY	Mount device busy
EEXIST	File exists
EXDEV	Cross-device link
ENODEV	No such device
ENOTDIR	Not a directory
EISDIR	Is a directory
EINVAL	Invalid argument
ENFILE	File table overflow
EMFILE	Too many open files
ENOTTY	Not a typewriter
ETXTBSY	Text file busy
EFBIG	File too large
ENOSPC	No space left on device
ESPIPE	Illegal seek
EROFS	Read-only file system

EMLINK	Too many links
EPIPE	Broken pipe
EDOM	Argument too large
ERANGE	Result too large
ENOMSG	Unexpected message type
EDEADLK	Resource deadlock avoided
ENOLCK	No locks available
ENOSTR	STREAMS device required
ENODATA	Missing expected message data
ETIME	STREAMS timeout occurred
ENOSR	Insufficient memory
EPROTO	Generic STREAMS error
EBADMSG	Invalid STREAMS message
ENOSYS	Function not implemented
ENOTEMPTY	Directory not empty
ENAMETOOLONG	File name too long
ELOOP	Too many levels of symbolic links
EOPNOTSUPP	Operation not supported on socket
EPFNOSUPPORT	Protocol family not supported
ECONNRESET	Connection reset by peer
ENOBUFS	No buffer space available
EAFNOSUPPORT	Addr family not supported
EPROTOTYPE	Protocol wrong type for socket
ENOTSOCK	Socket operation on non-socket
ENOPROTOOPT	Protocol not available
ESHUTDOWN	Can’t send after socket shutdown
ECONNREFUSED	Connection refused
EADDRINUSE	Address already in use
ECONNABORTED	Software caused connection abort
ENETUNREACH	Network is unreachable

ENETDOWN	Network is down
ETIMEDOUT	Connection timed out
EHOSTDOWN	Host is down
EHOSTUNREACH	No route to host
EINPROGRESS	Operation now in progress
EALREADY	Operation already in progress
EDESTADDRREQ	Destination address required
EMSGSIZE	Message size
EPROTONOSUPPORT	Protocol not supported
ESOCKTNOSUPPORT	Socket type not supported
EADDRNOTAVAIL	Can’t assign requested address
ENETRESET	Network dropped connection on reset
EISCONN	Socket is already connected
ENOTCONN	Socket is not connected
ETOOMANYREFS	Too many references: can’t splice
ENOTSUP	Unsupported value
EILSEQ	Illegal byte sequence
EOVERFLOW	Value overflow
ECANCELED	Operation canceled

Keep in mind that even if the exact error you encounter in your function may not map exactly to one particular standard errno code, it is still better to return the generic errno code rather than have to deal with custom error status. In 99.9% of cases you will therefore be returning a generic errno code (and often printing out the error to log) instead of writing a custom error status to a parameter passed to your function.

Best Practices

- **Always check return codes**: not doing so must be configured to be an error in your lint setup.
- **Always return errno**: If your function fails then return a negative value using one of the predefined errno codes.
- **Error status always well defined**: If a function that your function calls fails then return a negative errno code that best describes the state of your object but do not directly return the return code of the function you have called. Instead return an error code that is constant for your function.

- **Zero means success**: If your function succeeds and does not require reporting partial success then always return 0.
- **Partial success always uses positive numbers**: If your function requires partial success indication (such as number of items processed) then return this number as a positive integer.
- **Use parameters for complex return values**: If you need to return a complex return value then simply return zero to indicate success and write this complex value into a structure pointed to by one of your function’s parameters. Do not return complex structures as return values.

Pitfalls

- **No clear definition of standard return values**. The most serious pitfall with return values is not defining their meaning. If every function returns values that can be potentially mixed up with other values then you can never really know what to expect. This is why we have adopted errno codes as standard way to indicate errors - because they are defined as macros and they are ubiquitous - they are available everywhere - they are standardized. This means we always can deduce what the problem may be by looking at the errno code.
- **Returning structs**. When you need to return a struct, it is better to pass it as a parameter and then simply return an integer status code signaling whether the struct parameter was filled with valid data or not.

Alternatives

- **Exceptions**: this is only possible in CPP or Rust. Unfortunately if you are suing C, the return status codes of your functions is the only way to pass exceptions back to the caller.
- **Logging**: another way to handle status is by generating a log message. You should always combine this method with status return code.
- **Panic**: an assertion should be used for fatal errors and the application should call system panic function which in turn should reboot the device after logging the reason for the panic.
- **Long jump**: this is a mechanism of switching context during an exception and returning to an arbitrary place in the application where a corresponding "setjump" call was made. Do not use long jump because it does not give you an easy way of cleaning up resources. Instead it is better to panic or reboot if an exception occurs.

Conclusion

In this module we have covered a pattern for returning status of an operation.

We have covered meaning of negative, zero and positive status codes and what to do when the operation needs to "return" a complex value.

Quiz

- What status code should a function return upon success when there is no other information that needs to be conveyed?
    1. The function should return 1 to indicate success.
    2. The function should return zero to indicate success.
    3. Any value that is non-zero indicates success.
- What status code should a function return on failure?
    1. Any non-zero value indicates failure.
    2. A function must return negative errno code on failure.
    3. The function must always return -1 on failure.
- Why is it not a good idea to forward a return status from a function you have called and return it without changes?
    1. Because forwarding error status code makes it non-constant and thus may interfer with expectations up the call stack.
    2. Because it may result in stack corruption.
    3. Because that status code has no meaning to the caller.
- When should you return a positive status code?
    1. When there was an error.
    2. When the operation was successful.
    3. Only when the operation was successful and there are multiple ways in which a function can succeed.