MOSES Coding Specifications

The MOSES Project:

Meta Operating System And Entity Shell

Daniel J. Pezely

11 April 1991

---

1 Programming Language

The programming language syntax used here follows the ANSI X3J11 C programming language specifications and will be referred to as C.

2 Function Specifications

2.1 Calling Conventions

The C calling conventions for functions (parameter evaluation order) shall be maintained for all non-private kernel routines for all implementations in all programming languages used.

2.2 Return Codes

All routines described in this document return a signed integer value for success status, unless otherwise specified. The status value shall be a code corresponding to success, failure, or any errors which may have occurred.

Success is context dependent and will usually mean successful assignment or allocation of a resource, and it will always mean the task has completed.

When an error code is returned, the calling routines are responsible for recognizing the errors and taking appropriate action. Fatal errors cannot be forced and are not graceful. That is, no return value will cause a fatal error, and when fatal errors occur, no return value will be sent. User source code may have its own concept of a fatal error which is handled delicately.

The status codes shall always be referenced by name, and the corresponding values should be irrelevant with the sole exceptions of the first three values. `MOSES_Success`, `MOSES_Failure`, and `MOSES_OK` shall have the values `2`, `0`, and `1`, respectfully, to coincide peacefully with boolean values, even though a strict comparison of return values is strongly advised.

2.2.1 Status Return Codes:

1. `MOSES_Success == 2` – Everything in this task worked completely
2. `MOSES_Failure == 0` – Something in this task was unable to be completed
3. `MOSES_OK == 1` – Neutral; the task completed, period

2.2.2 Error Return Codes:

1. `MOSES_Close_Failed` – Cannot close file descriptor
2. `MOSES_Memory_Full` – Out of memory; cannot complete allocation
3. `MOSES_Message_Bad` – Message was malformed or incomplete
4. `MOSES_Open_Failed` – Cannot open specified file descriptor
5. `MOSES_Parameter_Bad` – Parameter was malformed or out of range
6. `MOSES_Parameter_Null` – A non-null parameter was expected
7. `MOSES_Parameter_Size` – Invalid size parameter: too large or small
8. `MOSES_Read_Failed` – Cannot read from file descriptor
9. `MOSES_Select_Failed` – Selection could not be made; generic

10. `MOSES_Select_Address_Stale` – Presumed actual address is incorrect
11. `MOSES_Task_Killed` – Task execution forced to terminate
12. `MOSES_Task_Suspended` – Task not executing temporarily
13. `MOSES_Task_Terminated` – Task exited, reason unknown
14. `MOSES_Task_Failed` – New task could not be started
15. `MOSES_Write_Failed` – Cannot write to file descriptor

2.3 Parameters

To ensure consistency with existing C libraries, function parameters conform to the traditional standard C programming convention of `function(destination, source)`. This is not directly expandable to C++ class methods, so *inline* methods should be used.

In the cases of `GroupleSymbolClass` parameters where sizes are required, the corresponding sizes shall always follow the parameter they apply to.

All functions outlined in this document shall be passed references to `GroupleHeadClass` structures which contain the data to be manipulated. This includes functions called via pointers. Such a restriction forces all kernel data, static and temporary, to be in the DataSpace at all times, provided the `New` routines are implemented according to their specifications.

2.3.1 Valid Parameters

Any or all reference parameters may be null; however, any non-null parameter is expected to be a valid pointer. Null destination references will cause most functions to return `MOSES_Failure`. No further action will be taken by the kernel or its libraries.

2.3.2 Grouples as Parameters

An *empty* grouple refers to a properly allocated `GroupleHeadClass` structure which simply references no terms (`GroupleTermClass` structures). A *null* grouple refers to a `GroupleHeadClass` structure pointer which has not been allocated, thus the pointer value should be null.

2.3.3 Parameters to Function Pointers

This section also extends to the functions within the DataSpace and called via the `fn` function pointer within the `GroupleTermClass` structure: the `GroupleFnPtrClass` *typedef*. That is, a reference to a grouple is passed as the parameter; however, these functions only take one parameter and return that parameter, if modified.
