DataSpace Function Specifications

The MOSES Project:

Meta Operating System And Entity Shell

Daniel J. Pezely

11 April 1991

---

1 The Grouple Symbol Module

Grouple symbols are generic for holding characters larger then just one byte. So, it is important to note again here that to maintain consistency, the network byte-order shall be used for message-passing.

The following functions are described below:

1. `NewSymbol()`
2. `NewSymbol_fromCstring()`
3. `NewSymbol_nByte()`
4. `DeleteSymbol()`
5. `CopySymbol()`
6. `CopySymbol_fromCstring()`
7. `CopySymbol_toCstring()`
8. `CopySymbol_nByte()`
9. `EvaluateSymbol()`
10. `SelectSymbol()`
11. `SubstituteSymbol()`

The symbol equivalents for the standard C string manipulation routine are listed after the descriptions of the routines in this list.

1.1 NewSymbol

```
GroupleSymbolClass * NewSymbol( int length )
```

- Parameter: length (number of characters) of symbol to create
- Attempt to allocate memory for a symbol string
- Memory allocated will be at least the requested size, if successful
- Passing a negative or zero length will fail
- Returns:

1. If successful, returns an allocated symbol reference with all characters assigned to zero
2. If unsuccessful (memory is full or unavailable), returns null

1.2 NewSymbol_fromCstring

```
GroupleSymbolClass * NewSymbol_fromCstring( char * string,
                                            int * length   )
```

- Parameters:
  1. C string
  2. referenced length (number of characters) of string to be assigned
- Attempts to allocate a symbol string to contain the specified string
- Memory allocated will be at least the requested size if successful
- If allocation succeeds, contents of C string will be copied to the symbol string
- Each C string character will occupy the lowest byte of the symbol character, and no delimiting token character will be assigned
- Passing a null string will return a null symbol
- Passing a null reference to the length will fail
- Return:
  1. If successful, returns an allocated, assigned symbol reference
  2. If unsuccessful (memory full or unavailable), returns null

1.3 NewSymbol_nByte

```
GroupleSymbolClass * NewSymbol_nByte( GroupleSymbolClass * string,
                                      int numBytes,
                                      int length                  )
```

- Parameters:
  1. referenced symbol character or symbol string
  2. number of bytes per string character
  3. length of string (number of characters)
- Attempt to allocate a new symbol string
- Memory allocated will be at least the requested size if successful
- If allocation succeeds, contents of string will be copied to the symbol such that each string parameter character will occupy the lowest bytes of the symbol character.
- If the string parameter characters occupy more bytes than do symbol characters, wrap-around of characters will occur such that string parameter characters will remain in network byte-order
- Passing a null string or zero length will fail
- Returns:
  1. If successful, returns an allocated, assigned symbol reference
  2. If unsuccessful, returns null

1.4 DeleteSymbol

```
int DeleteSymbol( GroupleSymbolClass * oldSymbol )
```

- Parameter: old symbol to be destroyed
- Deallocate the symbol
- Note: *a free-list may be maintained, but such internal features are not required for compatibility*
- Passing a null symbol will always succeed
- Return success status

1.5 CopySymbol

```
int CopySymbol( GroupleSymbolClass * destSymbol,
                GroupleSymbolClass * sourceSymbol,
                int length                        )
```

- Parameters:
  1. referenced destination symbol character or string
  2. referenced source symbol character or string
  3. length of symbol string
- Duplicate the source symbol string
- If the destination symbol is non-null, it is assumed to be large enough to contain the source
- If the destination symbol is null, attempt to allocate it to be at least as large as the specified length
- Copy the symbol strings up to the specified length
- Passing a null source or zero length is always successful
- Passing a null source or zero length and a non-null destination will cause the destination to be zeroed
- Return success status

1.6 CopySymbol_fromCstring

```
int CopySymbol_fromCstring( GroupleSymbolClass * destSymbol,
                            char * sourceSymbol              )
```

- Parameters:
  1. referenced destination symbol character or string
  2. source C string
- Duplicate the source symbol string
- If the destination symbol is non-null, it is assumed to be large enough to contain the source
- If the destination symbol is null, attempt to allocate it to be at least as large as the source length
- Copy the symbol strings up to, but not including, the delimiting token character
- The C string characters shall occupy the lowest bytes of the symbol characters in a character-by-character copy
- Passing a null source is always successful
- Passing a null source and a non-null destination will cause the destination to be zeroed
- Return success status

1.7 CopySymbol_toCstring

```
int CopySymbol_toCstring( char * destSymbol,
                          GroupleSymbolClass * sourceSymbol,
                          int length,
                          boolean truncate                  )
```

- Parameters:
  1. destination C string
  2. referenced source symbol character or string
  3. length of source symbol string
  4. boolean flag for truncation mode
- Duplicate the source symbol string
- If the destination symbol is non-null, it is assumed to be large enough to contain the source
- If the destination symbol is null, attempt to allocate it to be at least as large as the source length plus one C character for the delimiting character
- Note: *the destination length is dependent upon the truncation mode*
- Copy the symbol strings as determined by what value is passed for the truncation mode flag
- Passing a null source or zero length is always successful
- Passing a null source or zero length and a non-null destination will cause the destination to be zeroed
- Passing `TRUE` for the truncation mode flag will force only the lowest byte of each symbol character to be copied into C string
- Passing `FALSE` for the truncation mode flag will convert the symbol string to a C string in network byte-order
- Return success status

1.8 CopySymbol_nByte

```
int CopySymbol_nByte( GroupleSymbolClass * destSymbol,
                      int destNumBytes,
                      GroupleSymbolClass * sourceSymbol,
                      int sourceNumBytes,
                      int length,
                      boolean truncate                  )
```

- Parameters:
  1. referenced destination symbol character or string
  2. number of bytes per destination character
  3. referenced source symbol character or string
  4. number of bytes per source character
  5. length of source string
  6. boolean flag for truncation mode
- Duplicate the source string
- If the destination symbol is non-null, it is assumed to be large enough to contain the source
- If the destination symbol is null, attempt to allocate it to be at least as large as the source length
- Note: *the destination length is dependent upon the truncation mode*
- Copy the symbol strings as determined by what value is passed for the truncation mode flag
- Passing a null source or zero length is always successful
- Passing a null source or zero length and a non-null destination will cause the destination to be zeroed
- Passing `TRUE` for the truncation mode flag will force only the lowest bytes of each source character to be copied into each destination character

- Passing `FALSE` for the truncation mode flag will convert the destination string characters to network byte-order since they will overflow the destination character size
- Return success status

1.9 EvaluateSymbol

```
int EvaluateSymbol( GroupleHeadClass * resultGrouple,
                    GroupleSymbolClass * expressionSymbol,
                    int length
                    GroupleHeadClass * spaceSearchOrder   )
```

- Parameters:
  1. grouple to write result in
  2. referenced expression symbol character or sting
  3. length of symbol string
  4. space search order grouple (contains list of spaces to search and order)
- Called by `EvaluateGrouple()` and grouple function pointer routines
- Calls `SelectSymbol()` with spaces which might contain matches for the expression symbol
- Passing a null result grouple will fail since result cannot be written
- Passing a non-empty result grouple will be substituted with results, destroying previous contents
- Passing an empty or null expression symbol is considered a successful evaluation: evaluates to an empty grouple
- Passing a null space search order grouple will select from the entire DataSpace
- Passing an empty space search order grouple will always fail since nothing can be searched
- Returns success status

1.10 SelectSymbol

```
int SelectSymbol( GroupleHeadClass * matchingGrouple,
                  GroupleSymbolClass * matchingAddress,
                  int * addressLength,
                  GroupleSymbolClass * patternSymbol,
                  int patternLength,
                  GroupleSymbolClass * searchSpace     )
```

- Parameters:

  1. pre-allocated grouple to be linked with matching
  2. referenced symbol character or string describing physical address of the grouple sought and/or physical address of the matching
  3. referenced length of symbol describing physical address
  4. referenced symbol character or sting containing the pattern to search for
  5. length of the pattern symbol string
  6. grouple specifying which space (or spaces) to search

- Traverse each nested grouple (sublist) within the search space until the head of the grouple is a non-sublist
- compare each grouple within the search space for a match of the pattern
- the first match found will be substituted into the matching grouple
- Passing a non-empty matching will be assumed to be the previous matching and specifies where to continue the search
- Passing both a non-empty matching and a non-null physical address starts searching for the next match, assuming the given matching is at the given physical address
- Passing a null matching address and/or null address length prevents assigning both parameters
- Passing a null pattern or zero pattern length matches everything

- Passing a null search space will select from the entire DataSpace
- Passing an empty search space will always fail since nothing can be matched; this over-rides all other cases
- Returns success status

1.11 SubstituteSymbol

```
int SubstituteSymbol( GroupleSymbolClass ** oldSymbol,
                      GroupleSymbolClass * newSymbol  )
```

- Parameters:
  1. referenced pointer to old symbol character
     or referenced string to be overwritten
  2. referenced new symbol character or string
- Attempt to delete the old symbol
- Set the old reference point to the new reference
- Note: *this is a copying of pointers and not a copying of symbol characters*
- Passing a null old reference will fail since there would be no place to substitute in to
- Returns success status

Standard String Routines: The following prototypes are for the `GroupleSymbolClass` version of the standard C string routines: strcat, strncat, strdup, strcmp, strncmp, strcpy, strncpy, strchr, strrchr, strpbrk, strspn, strcspn, strstr, and strtok.

There is no equivalent for strcasecmp() since symbols deal with more than just ASCII characters, and there is no equivalent for strlen() since the lengths of strings must be maintained separately from the string data.

Descriptions are omitted; however, the verbose parameter names should provide sufficient details. The routines in this section return the same value as all symbol routines, except where noted, but all routines return an integer.

When cases arise where a string pointer needs to be returned, as referenced pointer variable (double indirection) should be passed as a parameter to that function. Such parameters are denoted as `destSymbol`.

```
int Symbol_cat( GroupleSymbolClass * destSymbol, int destSize,
                GroupleSymbolClass * sourceSymbol, int sourceSize )


int Symbol_dup( GroupleSymbolClass ** destSymbol, int * destSize
                GroupleSymbolClass * sourceSymbol, int sourceSize )
```

The Following two function return same as strcmp():

```
int Symbol_cmp( GroupleSymbolClass * symbol1, int size1,
                GroupleSymbolClass * symbol2, int size2 )


int Symbol_cpy( GroupleSymbolClass * destSymbol, int * destSize
                GroupleSymbolClass * sourceSymbol, int sourceSize )


int Symbol_chr( GroupleSymbolClass * destSymbol, int * destSize
                GroupleSymbolClass * sourceSymbol, int sourceSize,
                GroupleSymbolClass symbolChar, int numBytes        )


int Symbol_chr_nByte( GroupleSymbolClass * destSymbol, int * destSize
                      GroupleSymbolClass * sourceSymbol, int sourceSize,
                      GroupleSymbolClass symbolChar, int numBytes        )


int Symbol_rchr( GroupleSymbolClass * destSymbol, int * destSize
                 GroupleSymbolClass * sourceSymbol, int sourceSize,
                 GroupleSymbolClass symbolChar, int numBytes        )


int Symbol_rchr_nByte( GroupleSymbolClass * destSymbol, int * destSize
                       GroupleSymbolClass * sourceSymbol, int sourceSize,
                       GroupleSymbolClass symbolChar, int numBytes        )


int Symbol_pbrk( GroupleSymbolClass ** destSymbol, int * destSize,
                 GroupleSymbolClass * symbol1, int size1,
                 GroupleSymbolClass * symbol2, int size2          )
```

The following one function returns the same as strspn():

```
int Symbol_spn( GroupleSymbolClass * symbol, int size,
                GroupleSymbolClass * tokenSymbol, int tokenSize )
```

The following one function returns the same as strcspn():

```
int Symbol_cspn( GroupleSymbolClass * symbol, int size,
                 GroupleSymbolClass * tokenSymbol, int tokenSize )


int Symbol_str( GroupleSymbolClass ** destSymbol, int * destSize,
                GroupleSymbolClass * symbol1, int size1,
                GroupleSymbolClass * symbol2, int size2          )


int Symbol_str( GroupleSymbolClass ** destSymbol, int * destSize,
            GroupleSymbolClass * symbol, int size,
            GroupleSymbolClass * tokenSymbol, int tokenSize  )
```

---

2 The Grouple Term Module

This section contains the following routines:

1. `NewTerm()`
2. `DeleteTerm()`
3. `CopyTerm()`
4. `EvaluateTerm()`
5. `SelectTerm()`
6. `SubstituteTerm()`
7. `SubstitutePrefixTerm()`
8. `SubstituteSuffixTerm()`

2.1 NewTerm

```
GroupleTermClass * NewTerm( void )
```

- Parameters: none
- Attempts to allocate a `GroupleTermClass` structure
- Assigns all fields to zero, if allocation successful
- Returns: pointer to allocated term

2.2 DeleteTerm

`int DeleteTerm( GroupleTermClass * oldTerm )`

- Parameter: Old term to be destroyed
- If the term contains a symbol, deallocate the symbol
- If the term contains a sublist, deallocate that grouple via `DeleteGrouple()`
- Deallocate the term
- Note: *a free-list may be maintained, but such internal features are not required for compatibility*
- Passing a null or empty term will be considered successful
- Returns success status

These routines follow from the Grouple and Symbol routines.

2.3 CopyTerm

2.4 EvaluateTerm

2.5 SelectTerm

2.6 SubstituteTerm

2.7 SubstitutePrefixTerm

2.8 SubstituteSuffixTerm

3 The Groupple Module

The following functions are described below:

1. `NewGroupple()`
2. `DeleteGroupple()`
3. `CopyGroupple()`
4. `EvaluateGroupple()`
5. `SelectGroupple()`
6. `SubstituteGroupple()`

3.1 NewGroupple

`GroupleHeadClass * NewGrouple( void )`

Synopsis: Attempt to allocate memory for a new grouple
Parameters: none
Tasks:

- Attempt to allocate memory for a new grouple
- If unsuccessful, release all memory allocated with this task
- If grouple is not null, assign all fields to null values
- Insert new grouple into the DataSpace, without regard to location

Return:

1. If successful, return an allocated grouple reference with all data fields assigned zero
2. If unsuccessful (memory full or unavailable), return null

3.2 DeleteGrouple

```
int DeleteGrouple( GroupleHeadClass * oldGrouple )
```

Synopsis: Remove one link to this grouple
Parameter: grouple to be destroyed
Tasks:

- Decrement the link count of this grouple
- If the number of readers or number of links is non-zero, do nothing
- Else, remove specified grouple from the DataSpace
- Deallocate the grouple header (`GroupleHeadClass` structure) before destroying any terms within the specified grouple
- Deallocate each term via `DeleteTerm()` only after the header has been destroyed

  Note: *a free-list may be maintained, but such internal features are not required for compatibility*
- Passing a null grouple shall always be successful

Return success status.

3.3 CopyGrouple

```
int CopyGrouple( GroupleHeadClass * destGrouple,
                 GroupleHeadClass * sourceGrouple )
```

Synopsis: Duplicate entire contents and structure of the source grouple, deleting the contents of the destination grouple if non-null
Parameters:

1. destination grouple to be assigned

2. source grouple to be duplicated

Tasks:

- Assign destination to the duplicate

  Note: *this is a complete copy of the source, not just a second link*

- Delete original destination grouple contents, if not originally null.

- Passing a null destination and a non-null source shall never be successful

Return success status

3.4 EvaluateGrouple

```
int EvaluateGrouple( GroupleHeadClass * expressionGrouple,
                     GroupleHeadClass * spaceSearchOrder   )
```

Synopsis: perform match and substitute; execute matched function pointers
Parameters:

1. expression grouple (contains Lisp expression or function pointers)
2. space search order grouple (contains list of spaces to search in order)

Tasks:

- If first term of expression is a function pointer, call that function
- Call grouple function pointers with entire expression grouple as the parameter; function will call selection and evaluation routines as needed
- Non-function pointers are evaluated by calling `EvaluateSymbol()`
- Run task (function) to completion, even if multiprocessing is available

  Note: *no examination of the function is made, so discrimination is advised*
- The results of the called routine will be substituted into the expression grouple when the function returns
- Function pointer calls with no return cause an empty grouple substitution
- If the expression head is a symbol, parse the head via `SelectSymbol()`
- While the expression is not an atomic symbol or function pointer (i.e., is a sublist), *call this routine recursively* with that sublist
- Passing an empty expression grouple is considered a successful evaluation: evaluates to itself
- Passing a null expression grouple will fail since a substitution cannot be made back into the expression

- Passing a null space search order grouple will search the entire DataSpace
- Passing an empty space search order grouple will always fail.

Returns success status

Notes:

1. *This routine may be* called recursively.
2. *On a shared memory system, any modifications to the expression grouple will be lost at the time of substitution, no matter who made the modifications, with or without security implemented.*
3. *Grouple functions should return the address of its parameter if the function modifies the expression grouple, as noted elsewhere.*
4. *The substitution may be uncoupled from this routine, thus may be done at anytime, so comparing expression references before using the substitution is advised.*

3.5 SelectGrouple

```
int SelectGrouple( GroupleHeadClass * matchingGrouple,
                   GroupleSymbolClass * matchingAddress,
                   int * addressLength,
                   GroupleHeadClass * patternGrouple,
                   GroupleHeadClass * searchSpace      )
```

Synopsis: Attempt to find one match of the pattern in the space
Parameters:

1. grouple containing matching to continue from and/or pre-allocated grouple to be linked with matching
2. referenced symbol character or string describing actual address of the grouple sought and/or actual address of the matching
3. referenced length of symbol describing actual address
4. grouple containing pattern to be matched;

   Note: *This is a grouple and not just a symbol, but the grouple should be kept simple.*
5. grouple containing reference to the DataSpace or a subspace or a flat grouple

Tasks:

- Link matchingGrouple to the first match found, if any
- if no match is found, the matching grouple will be unmodified and the task will be a failure
- Calls `SelectSymbol()`
- Passing a non-empty matching is assumed to be the previous matching, thus searching will begin following the given matching
- Passing a non-null actual address tells this routine that its job is done if the addressed grouple is a match for the pattern, otherwise it is an error

- Passing both a non-empty matching and a non-null actual address starts searching for the next match, assuming the given matching is at the given actual address
- Passing a null matching address and/or null address length prevents assigning both parameters
- Passing a null or empty pattern matches everything
- Passing a null space will force selection from entire DataSpace
- Passing an empty space can never be matched, thus will fail
- Passing a space containing function pointers will cause comparisons to be made as follows:

  Tasks:

  - a copy of the pattern grouple with the function pointer prefixed will be made
  - functions to be compared will be evaluated via `EvaluateGrouple()` with parameter being the modified pattern copy

Returns:

1. In general cases, return success status
2. If the actual address is passed and is incorrect, return the `MOSES_Select_Stale_Address` error code

3.6 SubstituteGrouple

```
int SubstituteGrouple( GroupleHeadClass * oldGrouple,
                       GroupleHeadClass * sourceGrouple )
```

Synopsis: Copy reference of the old grouple into a temporary address

Parameters:

1. old grouple to be replaced
2. source grouple to substitute with

Tasks:

- Link the old grouple references to the source grouple

  Note: *a duplication is not made, a second link is created*
- Delete original destination grouple contents, if not originally null.
- Passing a null old grouple and a non-null source is never successful

Return success status

The following two routines described here are a composition of the *delete* and *substitute* grouple operations. The new routines are formed to handle tasks which could not otherwise be done. That is, these are not just calling two routines consecutively but are operations of the actual original grouples stored in the DataSpace.

The following functions are described below:

1. `DeleteSelectGrouple()`
2. `SubstituteSelectGrouple()`

3.7 DeleteSelectGrouple

```
int DeleteSelectGrouple( GroupleHeadClass * matchingGrouple,
                         GroupleSymbolClass * matchingAddress,
                         int addressLength,
                         GroupleHeadClass * patternGrouple,
                         GroupleHeadClass * searchSpace      )
```

- Parameters:
  1. grouple containing a matching to continue from
  2. referenced symbol character or string describing actual address of the grouple sought
  3. length of symbol describing actual address
  4. grouple containing pattern to be matched;

     Note: *This is a grouple and not just a symbol, but the grouple should be kept simple.*
  5. grouple containing reference to the DataSpace or a subspace or a flat grouple
- Attempt to find one match of the pattern in the space and delete that grouple
- Calls `SelectGrouple()` followed by `DeleteGrouple()` to delete the grouple matched, not a copy of the match or a secondary reference to it
- Passing a non-empty matching is assumed to be the previous matching, thus searching will begin following the given matching
- Passing a null matching does not cause the routine to fail, as it does with `SelectGrouple()`
- Passing a non-null actual address tells this routine that its job is done if the addressed grouple is a match for the pattern, otherwise it is an error

- Passing both a non-empty matching and a non-null actual address starts searching for the next match, assuming the given matching is at the given actual address
- Passing a null matching address and/or null address length prevents assigning both parameters
- Passing a null or empty pattern matches everything
- Passing a null space forces a search of the entire DataSpace
- Passing an empty space can never be matched
- Passing a space containing function pointers will cause comparisons to be made as follows:
  - a copy of the pattern grouple with the function pointer prefixed will be made
  - functions to be compared will be evaluated via `EvaluateGrouple()` with parameter being the modified pattern copy
- Returns:
  1. In general cases, return success status
  2. If the actual address is passed and is incorrect, return the `MOSES_Select_Stale_Address` error code

3.8 SubstituteSelectGrouple

```
int SubstituteSelectGrouple( GroupleHeadClass * matchingGrouple,
                             GroupleSymbolClass * matchingAddress,
                             int * addressLength,
                             GroupleHeadClass * patternGrouple,
                             GroupleHeadClass * searchSpace,
                             GroupleHeadClass * sourceGrouple     )
```

- Parameters:

  1. pre-allocated grouple containing a possible initial matching and to be the grouple which gets substituted grouple for the matching
  2. referenced symbol character or string describing actual address of the grouple sought and/or actual address of the newly substituted grouple
  3. referenced length of symbol describing actual address
  4. grouple containing pattern to be matched;

     Note: *This is a grouple and not just a symbol, but the grouple should be kept simple.*
  5. grouple containing reference to the DataSpace or a subspace or a flat grouple
  6. source grouple to replace matching grouple

- Attempt to find one match of the pattern in the space and substitute the source grouple for the matching grouple

- Calls `SelectGrouple()` followed by `SubstituteGrouple()` to replace the matched grouple which is in the DataSpace, not a copy of that grouple or a secondary reference to it

- Note: *The matching address may have to be re-calculated if the DataSpace is ordered by a hashing function or tree structure.*

- Passing a non-empty matching is assumed to be the previous matching, thus searching will begin following the given matching

- Passing a non-null actual address tells this routine that its job is done if the addressed grouple is a match for the pattern, otherwise it is an error
- Passing both a non-empty matching and a non-null actual address starts searching for the next match, assuming the given matching is at the given actual address
- Passing a null matching address and/or null address length prevents assigning both parameters
- Passing a null or empty pattern matches everything
- Passing a null space forces a search of the entire DataSpace
- Passing an empty space can never be matched
- Passing a space containing function pointers will cause comparisons to be made as follows:
  - a copy of the pattern grouple with the function pointer prefixed will be made
  - functions to be compared will be evaluated via `EvaluateGrouple()` with parameter being the modified pattern copy
- Returns:
  1. In general cases, return success status
  2. If the actual address is passed and is incorrect, return the `MOSES_Select_Stale_Address` error code

4 Grouple I/O

The extent to which this object provides functionality determines if MOSES is a stand-alone operating system or a meta operating system, which is an application to another operating system.

All memory which this object might need to use will be managed by the DataSpace Object. That is, all object-global variables shall be allocated and manipulated via the DataSpace Object routines described in this document.

The Interrupt Handler, Device Drivers, and File Descriptor I/O modules will be provided by the native operating system which MOSES will be running on. These modules need only provide enough facility to support the routines outlined in this section. Specifically, the native operating system must support file descriptors (or an equivalent high-level access to devices).

4.1 Message I/O

The following functions are described below:

1. `IO_SelectNextFD()`
2. `IO_GroupleReadFD()`
3. `IO_GroupleWriteFD()`

4.1.1 IO_SelectNextFD

```
int IO_SelectNextFD( int * currentFD,
                     IO_ModeType mode )

enum _IO_ModeType { READ_MODE, WRITE_MODE, EXCEPTION_MODE };
```

- Parameters:
  1. reference to the file descriptor to be assigned
  2. enumeration of function mode: read, write, or exception to select
- Select the next file descriptor to be the current one
- Make selection based upon file descriptors waiting to be read, written, or have an exception pending
- Passing a null reference will always fail
- Passing an invalid mode will always fail
- Returns success status

4.1.2 IO_GroupleReadFD

```
int IO_GroupleReadFD( Grouple_Class * grouple,
                      int fd                   )
```

- Parameters:
  1. reference to an allocated grouple
  2. file descriptor number
- Read data from the device associated with the specified file descriptor
- Form a grouple by parsing the input stream according to the Common Lisp programming language syntax
- Returns success status

4.1.3 IO_GroupleWriteFD

```
int IO_GroupleWriteFD( Grouple_Class * grouple,
                       int fd                   )
```

- Parameters:
  1. reference to an allocated grouple
  2. file descriptor number
- Writes a grouple to the device associated with the file descriptor
- Forms the Common Lisp programming language syntax from the specified grouple
- Returns success status

4.2 Miscellaneous Grouple I/O Routines:

4.2.1 Initialize_IO

```
int Initialize_IO( void )
```

- Parameters: none
- Performs any and all initialization tasks, dependent upon the implementation
- Returns success status

---

5 The Variables Module

All variables are grouples, thus stored in the DataSpace. The purpose of a special module to handle variables is to make client programmer jobs so easy that they will want to use the Variables module for the benefits of run-time debugging without the aid of source-level debuggers and the ability to modify values at run-time.

The following routines are described below:

1. `MakeSymbolVariable()`
2. `MakeTermVariable()`
3. `MakeFnVariable()`
4. `MakeGroupleVariable()`

5.1 MakeSymbolVariable

```
int MakeSymbolVariable( GroupleHeadClass ** grouple,
                        char * name,
                        GroupleSymbolClass * value,
                        int valueLength             )
```

Synopsis: Attempt to allocate a new grouple in the DataSpace
Parameters:

1. referenced grouple pointer for allocated grouple
2. C string containing variable name
3. Referenced symbol character or string containing the value of the variable variable
4. Length of the value symbol

Tasks:

- Attempt to allocate a new grouple in the DataSpace
- Assign new grouple head to a symbol containing the specified name
- Assign new grouple tail to the specified value symbol
- Passing a null grouple pointer reference will fail
- Passing a null name will fail

Returns success status

5.2 MakeTermVariable

```
int MakeTermVariable( GroupleHeadClass ** grouple,
                      char * name,
                      GroupleTermClass * term      )
```

Synopsis: Attempt to allocate a new grouple in the DataSpace
Parameters:

1. referenced grouple pointer for allocated grouple
2. C string containing variable name

3. Referenced term

Tasks:

- Attempt to allocate a new grouple in the DataSpace
- Assign new grouple head to a symbol containing the specified name
- Assign new grouple tail to the specified term
- Passing a null grouple pointer reference will fail
- Passing a null name will fail

Returns success status

5.3 MakeFnVariable

```
int MakeFnVariable( GroupleHeadClass ** grouple,
                    char * name,
                    GroupleFnPtrClass fn         )
```

Synopsis: Attempt to allocate a grouple in the DataSpace
Parameters:

1. C string containing variable name
2. function pointer to be associated with name

Tasks:

- Attempt to allocate a grouple in the DataSpace
- Assign new grouple head to a symbol containing the specified name
- Assign new grouple tail to a term containing the specified function
- Passing a null name will fail

Returns:

1. If successful, returns an allocated grouple reference
   - containing a `DataSpace` variable
   - value should be retrieved via `SelectVariable()`
2. If unsuccessful (memory full or unavailable), returns null

5.4 NewVariableGrouple

```
GroupleHeadClass * NewVariableGrouple( char * name,
                                       GroupleHeadClass * grouple )
```

Synopsis: Attempt to allocate a grouple in the DataSpace
Parameters:

1. C string containing variable name
2. referenced grouple to be variable value

Tasks:

- Attempt to allocate a grouple in the DataSpace
- Assign new grouple head to a symbol containing the specified name
- Assign new grouple tail to the specified grouple reference
- Passing a null name will fail

Returns:

1. If successful, returns a grouple reference
    - containing a DataSpace variable
    - value should be retrieved via `SelectVariable()`
2. If unsuccessful (memory full or unavailable), returns null

5.5 SelectVariable

```
int SelectVariable( char * name,
                    GroupleSymbolClass * address,
                    int * addressLength,
                    GroupleSymbolClass * value,
                    int * valueLength             )
```

Synopsis: Select the first matching grouple and assign the physical address, value, and associated lengths

Parameters:

1. C string containing the name of variable variable
2. referenced symbol character or string expected physical address of the variable
3. referenced length of physical address
4. referenced value symbol to be assigned (typecast if non-symbol)
5. referenced length of value symbol

Tasks:

- Select the first matching grouple and assign the physical address, value, and associated lengths
- Passing a null name will always fail
- Passing a non-null physical address will attempt to select addressed grouple
- Passing an invalid physical address will cause address to be ignored
- Passing a null address or address length will cause both parameters to not be assigned
- Passing a null value or value length will cause both parameters to not be assigned

Return success status
