List library

All functions are morphic.

**element\_in\_list (*list* list, *string* element -> *int* value)**

**Returns 1 if “element” is a member of “list”, 0 otherwise.**

**invert\_list (*list* list -> *list* list)**

**Reverses the order of “list” and returns it.**

**length\_of\_list (*list* list -> *int* value)**

**Returns the number of elements in “list”.**

**(legacy name: “count\_list”)**

**push (*str* head, *list* tail -> *list* list)**

**Adds head onto the front of tail and outputs the result.**

**read\_in\_assoc\_array (*str* file, [*str* location, *str* locbase] -> *key-value-list* list)**

**As read\_in\_list, except that “file” should be either a 2+-column 2da, a 3+-column 2da where the 2nd column is all “=>”, or a 1+-column 2da where the first column consists of key=>value pairs, and the output is read into a key=>value list in the obvious way.**

**read\_in\_list (*str* file, [*str* location, *str* locbase] -> *list* list)**

**Reads in the file “file”, located in the directory specified by “location” and “locbase” (see get\_directory), which should be a 2da. Output its first column as a space-separated list.**

**return\_first\_entry (*list* list, [*str* separator] -> *list* list, *str* entry)**

**Returns the first element of “list” as “entry” and the remainder as “list”. Optionally, if “separator” is set to any one-character string, “list” will be treated as a list of strings separated by that string rather than by space. Fails if “list” is empty.**

**(Legacy name: action\_return\_first\_entry)**

**return\_first\_pair (*key-value-list* list -> *str* key, *str* value, *arrow-list* list)**

**Returns the first key-value pair of “list” as “key” and “value”; returns the remainder of the list as “list”. Fails if “list” is empty.**

**(Legacy name: action\_return\_first\_pair)**

**split\_from\_left (*list* string, *str* splitter -> *list* left, *list* right)**

**Finds the first occurrence of “splitter” in “string”; breaks the list in two at that point (discarding that occurrence itself) and returns the two parts.**

**split\_from\_right (*list* string, *str* splitter -> *list* left, *list* right)**

**Finds the last occurrence of “splitter” in “string”; breaks the list in two at that point (discarding that occurrence itself) and returns the two parts.**

**write\_list (*str* file, *list* list, [*str* location]->null)**

Takes the contents of list, and append them, one per row, to “file”, located in “location”, or in %workspace% if location is not set. (NB: “location” is absolute, not relative to your mod directory or to component\_loc.) Creates the file if it does not already exist.

Tools library

Action functions

**consolidate\_table (*str* in, *str* out, [*str* location] -> *null*)**

**Read in the table “in” from “location” (which is absolute, not relative to your mod base location), or from %workspace% if location is not set. “in” should be a 2-column array. For each distinct element in the first column, make a space-separated list of all the elements in the second column. Output to “out”, in the same base location, a file where the first column is the distinct first-column elements of “in” and the remainder is a list of the 2nd-column elements.**

**(So**

**animal cat**

**animal dog**

**monster yeti**

**would become**

**animal cat dog**

**monster yeti)**

**include (*list* files, [*str* location, *str* locbase]-> *null*)**

Include each file in the list as tp2 code, looking in whatever location is defined by “location” and “locbase” (see **get\_directory**). Prior to inclusion, MAKE\_PATCH is replaced by ACTION\_CLEAR\_ARRAY patch\_data ACTION\_DEFINE\_ASSOCIATIVE\_ARRAY patch\_data BEGIN, and PUSH foo bar is replaced by SPRINT foo “%foo%”^bar. Fail if any of them cannot be found.

“file” is an acceptable synonym for “files”. Note that because this inclusion occurs within a function, it cannot be used to permanently set or modify variables. (Use plain WEIDU INCLUDE to do that.)

**process\_table (*str* table, *func* function, [*str* inline, *str* location, *str* locbase] -> *null*)**

**Read in the file “table”, located at the location specified by “location” and “locbase” (see get\_directory), or located inline at …/stratagems\_inline if inline is set to “yes”. “table” should be a 2da where the first row gives the names of the entries in the remaining rows.**

**For each row, treat the elements of the first row as variable names, set those variables to the values in the row, and feed the result as STR\_VAR inputs to “function”.**

**run (*list* files, [*str* location, *str* locbase, *str* version]-> *null*)**

For each element “func” of “files”, look for “func.tpa” in whatever location is defined by “location” and “locbase” (see **get directory**). Include it, and then attempt to run the ACTION\_FUNCTION “func”, with “version” (if set) as an argument. (So “func.tpa” should consist of a function definition for func along with any auxiliary functions required.) Prior to inclusion, carry out the same substitutions as for **include.** Fail if any file is missing or fails to contain the required function.

“file” is an acceptable synonym for “files”. Note that because this inclusion occurs within a function, it cannot be used to permanently set or modify variables. (But you shouldn’t even be trying to do that with **run**!)

Patch functions

**get\_field\_value(*int* loc, *int* length -> *int* value)**

**If “length”, is 1, 2, or 4, issue the appropriate READ\_BYTE/ SHORT/ LONG command to read that many bytes at“loc”, and return it. Otherwise, fail.**

**read\_rest\_of\_line (*int* row, *int* col, *int* colmin -> *list* list)**

**In the current file (which should be a 2da), look at all rows with >= colmin columns. In the “row”th such row, read in everything from column “col” onwards as a list. (This is useful for “2da” files with varying-length rows.)**

**set\_field\_value(*int* loc, *int* length, *int* arguments -> *null*)**

**If “length”, is 1, 2, or 4, issue the appropriate WRITE\_BYTE/ SHORT/ LONG command to write “arguments” to that many bytes at“loc”, and return it. Otherwise, fail. (“value” is a legal legacy alternative to “arguments”.)**

Morphic functions

**check\_ini (*str* ini -> *int* value)**

**Returns the value of the “ini” entry ini file (read in at the start of SFO installation), or zero if that entry is not set.**

**(Legacy names: action\_check\_ini, patch\_check\_ini)**

**check\_label (*str* label -> *int* value)**

**Returns 1 if a file called [label].mrk exists in the override (or in %workspace%, if put\_labels\_in\_workspace is set to 1 in the ini file); returns 0 otherwise. Used for component detection along with make\_label (consider using Weidu LABEL instead if your mod isn’t too complicated).**

**color (*int* red, *int* green, *int* blue -> *int* color)**

**“red”, “green” and “blue” should be in the range 0-255, and are combined into a 4-byte LONG integer according to IE game conventions.**

**count\_hundreds (*int* counter, *int* hundreds, [*str* task] -> *int* counter, *int* hundreds)**

**Adds one to counter; if the result is 100, sets counter to zero, adds 1 to hundreds, echoes “%hundreds%00 task”. The default value of “task” is “files analysed”: the intended use of this is to give an end user feedback on how much of a large task has been done.**

**edit\_sanity\_check(*str* editstring, *str* edits, [*str* task] -> *null*)**

**Fails if either (i) “editstring” is neither empty nor contains “=>” (meaning that it’s not a key=>value list), or (ii) “edits” contains “=>” (meaning that it isn’t the name of an array, and plausibly *is* a key=>value list). If “task” is set, quotes it in the failure message.**

**ensure\_hex (*int* in, [*int* pad\_length] -> *str* out)**

**Convert “in” into the string that is its hex representation. If pad\_length is set, string is padded out to be at least that many digits (so if pad\_length=8, 32 would be converted to 0x00000020).**

**get\_directory([*str* location, *str* locbase]-> *str* directory)**

**Returns a directory, which is either [mod base directory]/component\_loc/local\_loc/location if locbase is empty, or [mod base directory]/locbase/location otherwise. Empty values of any of component\_loc, local\_loc or location are dealt with correctly (i.e. you won’t get // occurring).**

**(Legacy name: action\_get\_directory)**

**key\_value\_sanity\_check(*int* is\_key, *str* arguments -> *null*)**

**Fails, with a vocal warning, if either (i) “arguments” is a key=>value list and is\_key=0, or (ii) “arguments” is *not* a key=>value list and is\_key=1.**

**log\_this (*str* input*, str* file, [*str* loc, *str* repeat=yes] -> *null* )**

**Append “input” to the file “file” located at “loc”, or in %workspace% if loc is not set; if the file does not exist, create it.**

**If “repeat” is set to “no” (or “NO”) then the string will only be appended if it is not already there (as a full line).**

**make\_label (*str* label -> *null*)**

**Create an empty file called “[label].mrk” in the override (or in %workspace% if you set “put\_labels\_in\_workspace” to 1 in the ini file.) Used for component detection along with check\_label (consider using Weidu LABEL instead if your mod isn’t too complicated).**

**reinclude\_this (*str* input -> null)**

**Does variable substitution on “input” and then REINCLUDEs it (via PATCH\_REINCLUDE or REINCLUDE according to context) as tp2 input.**

**NB: since this is a function, it can’t be used to set or modify variables. Use the “reinclude\_this” macro if you want to do that.**

**(Legacy names: action\_reinclude\_this, patch\_reinclude\_this)**

**return\_first\_letter (*str* input -> *str* firstletter, *str* rest)**

**Returns the first character, and the remainder, of “input”.**

**return\_function\_and\_argument (*str* input -> *str* function, *str* argument)**

**If “input” has the form “function(argument)”, returns function and argument; otherwise, fails.**

**reverse\_long (*int* input -> *int* output)**

**Treat the input as a 4-byte LONG, and reverse the first and last SHORT. (If you’re not sure why you want to do this, you don’t.)**

**strip\_quotes\_from\_string (*str* string -> *str* string)**

**Removes any leading and trailing ‘~’ and ‘”’ from the string.**

**strip\_spaces\_from\_string (*str* string -> *str* string)**

**Removes any leading and trailing spaces from the string.**

**warning (*str* arguments -> *null*)**

Logs the text in “arguments” (unless blank or equal to “null”) to %workspace%/sfo\_warnings.txt, and echoes it to the screen. “warning” is an acceptable synonym for “arguments”.

(Legacy names: action\_display\_warning, patch\_display\_warning)