KNX API

2016

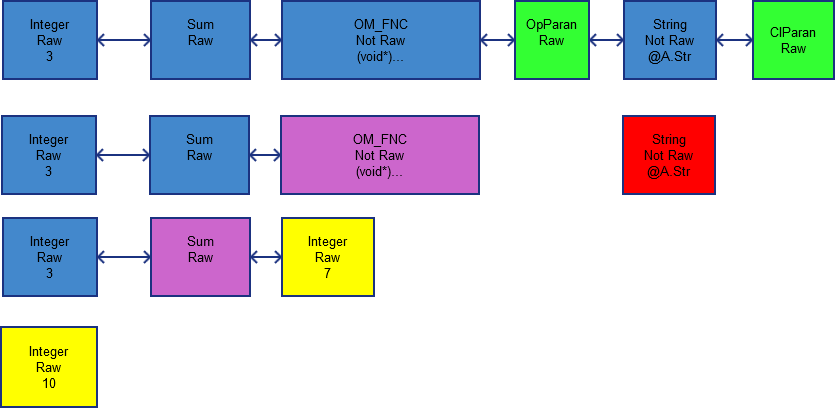
V 1.0.0 A

The KNX language does not, by itself, define many operations. Instead, most functionality is defined externally. These external libraries of functions that come with the language make up the “standard libraries”. This document covers both the internal operations such as keywords along with the external implementations, sorted by library.

A general explanation on the structure of functions begins at the chain resolver. It is helpful to understand how operations are resolved in order to best utilize their operations.

The execution stage of the interpreter, where commands are processed, will scan over a token stream. That is, a list of individual points of data that consist of information and commands. This is called a *chain*. When the chain is being ‘resolved’, or reduced through the process of execution, certain sections of the stream are processed, removed and then replaced with result of the operation.

For instance, consider a token stream of the following:



As each step is resolved, the old tokens are replaced with the results. In this way, results of functions may be used as objects.

**Keywords**

Keywords are built in operations. These allow the user to access the core functions of the engine. In essence, these operate as regular functions. They take parameters and return a result. However, they are always available in any scope and do not require imports to become available.

|  |  |  |
| --- | --- | --- |
| Keyword | Use | Description |
| int | int *name*  int *(name, value)* | Creates an integer. An initial value may be set, but is 0 by default. |
| short | short *name*  short *(name, value)* | Creates a short integer. An initial value may be set, but is 0 by default |
| long | long *name*  long *(name, value)* | Creates a long long integer. An initial value may be set, but is 0 by default |
| float | float name  float *(name, value)* | Creates a double precision float (double). An initial value may be set, but is 0 by default. |
| bool | bool name  bool *(name, value)* | Creates a boolean. An initial value may be set, but is false by default.  All non-zero assignments are cast to 1. |
| char | char *name*  char *(name, value)* | Creates a character. An initial value may be set, but is 0 (NULL) by default. |
| string | string *name*  string (*name, value*)  string *(name, size)* | Creates a character string. An initial value may be set, but is empty by default.  An initial size may also be set |
| class | class *name* | Creates an empty class. |
| list | List *name*  List *(name, size)* | Creates an aggregate list. While an initial size may be set with all empty spaces set to NULL, the list is empty by default. |
| array | array *(name, type)*  array*(name, type, size)*  array*(name, type, element…)* | Creates an array of a particular type. Three overloads are possible. An empty array may be initialized with a name and a type.  A starting size may also be specified.  Otherwise, all additional elements will simply be loaded in at the time of creation. |
| void | void *name*  void *(name, target)* | Creates a void object. If only a name is specified, it will be initialized to NULL and type *none*.  Otherwise, it will conform its identity to the input. |
| auto | auto *(name, value)* | Creates an automatically determined object based on the input. |
| import | import “*library*”  import *library*  import (*library,…)* | Imports a library into memory. While it is recommended to pass the input as a string, it is not necessary. An implicit raw token will also work. Multiple inputs are allowed simultaneously. |
| delete | delete *object*  delete *(object,…)* | Deletes one or more objects. This excludes nodes. |
| terminate | terminate(*node)* | Terminates a node |
| xnode | xnode(*name*)  xnode*(name, accessPolicy)*  xnode(*name, command*)  xnode*(name, accessPolicy, command)* | Creates a child node. An access policy may be set, as well as an initial command |
| return | return *value* | Returns a value or object from the current scope. |
| exit | exit  exit *retCode* | Exits the program. An exit code may be specified (integer) but returns 0 by default. |
| loop | loop {…}  loop *condition {…}* | Will perform a loop of a given command. This will wait until the next real input before moving on if an operation is not specified on the calling line.  A condition to execute may also be set. |
| break | break | jumps to end of loop and ends it. |
| continue | continue | jumps to end of body |
| if | if *(condition)* {…} | Executes given command if the given condition evaluates to true. Set the current execution level to *true* if executed, or *false* otherwise |
| else | else {…} | If current execution level is *false,* executes given command |
|  |  |  |

**Flags**

The flag descriptions listed here are the standard effect of each. However, these are non-binding, and are subject to varying effects depending on any particular implementation.

|  |  |
| --- | --- |
| Flag | Standard Description |
| c | Flags object declaration to create a constant value (set to Raw, immutable) |
| g | Begin at global scope |
| l | Begin at local scope (default) |
| f | Force operation |
|  |  |
|  |  |
|  |  |

**Standard Library**

Standard Libraries:

* IO
* string
* math
* pipe
* network
* structures
* file
* regex
* memory
* cgraphics
* wgraphics
* usb
* standard

Most of these libraries implement additional functionality. Others, however, may add new data types as well.

Make note of libraries that generate additional data structures, however. Unloading the library with the custom type in use may result in potentially undefined behavior and cause memory leaks.

**IO**

|  |  |  |
| --- | --- | --- |
| *void* **display** … | Variable inputs to print to screen in order. Accepts textual and numerical values, including lists and arrays of these elements |  |
| *string* **gets**  *string* **gets** stream | Get string from terminal (stdin stream)  Get string from designated stream | ~f: Command will request temporary stdio control |
| *char* **getc**  *char* **getc** stream | Get character from terminal (stdin stream)  Get character from designated stream | ~f: Command will request temporary stdio control |
| *char* **getkc** | Get single keyboard stroke | ~f: Command will request temporary stdio control |
| *string* **getks** | Get single line from keyboard | ~f: Command will request temporary stdio control |
| *void* **setClr** *int* frgnd, *int* bckgrnd | Set text color to |  |
|  |  |  |
|  |  |  |