VortexLua Specifications v0.9.0

Preface

The scripting engine backend used in this version of VortexLua is Lua 5.1.0 with some limitations:

1. Some functions from the base library (dofile, load, loadfile and loadstring) are disabled.
2. The I/O and OS standard library are completely disabled.
3. Loading of other C and C++ shared library (using the loadlib standard library) is disabled.

Support Classes

CLuaManager, a script manager class. This class is responsible for:

1. Compiling raw text source codes to CLuaExecutable instances.
2. Registering all instances of CLuaExecutable that will be executed.
3. Registering all instances of CLuaExecutionSpace that will execute the executables.
4. Restoring all saved states of any CLuaExecutionSpace instance when it is being registered.
5. Saving all states of any CLuaExecutionSpace instance when it is being unregistered.

Only registered CLuaExecutable instances that can be executed and only registered CLuaExecutionSpace instances that can execute the executables.

CLuaExecutable, a buffer class used for storing the Lua executable byte codes.

CLuaExecutionSpace, a class which will execute the executable byte codes.

CLuaActionList, a class used for managing pairs of action names and callback functions, this class is to be used by CLuaExecutionSpace to register actions.

CActionCallbackParams, a class used for storing and returning values when an action callback function is called.

Node and Access Control List (ACL)

A node is a registered CLuaExecutionSpace instance. Each node is associated with two names:

1. A **node name**, used as identifier when storing and restoring the states of the node and for direct access to a specific node (by specifying the name of the node to be accessed in script). Each node must have a unique name, but if states saving and direct access are not needed, node name can be set to NULL.
2. A **group name**, used for ACL management.

Node name and group name must be set directly using the CLuaExecutionSpace class member functions, VortexLua has no functionality (and it mustn't!) for setting those names.

ACL is divided into two levels:

1. Per node ACL.
2. Per variable ACL.

Both ACL can be configured to control:

1. Read access.
2. Write access.
3. Penetration access (adding and deleting variables).

The “per node ACL” takes precedence over the “per variable ACL”, this means that if the “per node ACL” denies one of the read, write or penetration access (or combination of them), even if the “per variable ACL” allows the specific access, it will be ignored.

Default rules for newly created nodes and variables:

ACL Type	Allow Read	Deny Read	Allow Write	Deny Write	Allow Penetration	Deny Penetration
Per node ACL	[ALL]	-	[ALL]	-	-	[ALL]
Per variable ACL	[ALL]	-	-	[ALL]	-	[ALL]

Note that all operations with [SELF] will always succeed regardless the per node (and per variable) ACL(s) set(s) in SELF (and in the SELF's variable). This means that adding [SELF] to any of the ACL rules is not meaningful. In simpler term, [SELF] can do anything with itself without needing any permission from anyone/anything.

The VortexGE Extension to Lua

Helper Functions

» vge.errorString(errorCode)

Functions:

Obtaining the human-readable text for the given error code.

Parameters:

Names	Types	Descriptions
errorCode	number	The code number to be translated to string.

Returns:

One string containing the human-readable text.

Examples:

```
print(vge.errorString(vge.ok))

print(vge.errorString(vge.errorNodeNotFound))
```

» vge.getNodeName(specialNodeName)

Functions:

Obtaining the real node name of [SELF] or [EXE].

Parameters:

Names	Types	Descriptions
specialNodeName	string	Must be [SELF] or [EXE], if not, the function will always return nil.

Returns:

One string containing the real node name (can be empty string if unnamed) or or nil if the special node name is not valid/unusable at the current context.

» vge.getGroupName(specialNodeName)

Functions:

Obtaining the real node name of [SELF] or [EXE].

Parameters:

Names	Types	Descriptions
specialNodeName	string	Must be [SELF] or [EXE], if not, the function will always return nil.

Returns:

One string containing the real group name (can be empty string if unnamed) or nil if the special node name is not valid/unusable at the current context.

Version Checking

» vge.extensionVersion()

Functions:

Obtaining the VortexLua version numbers (not the Lua interpreter version number).

Parameters:

N/A

Returns:

Three numbers representing the major, minor and revision version.

Examples:

```
mj, mn, rv = vge.extensionVersion()
```

» vge.extensionVersionMinimum(major, minor, revision)

Functions:

Checking if the running scripting engine supports the minimum requested version.

Parameters:

Names	Types	Descriptions
major	number	The minimum major version requested.
minor	number	The minimum minor version requested.
revision	number	The minimum revision version requested.

Returns:

One boolean, true if the requested minimum version is supported, or false otherwise.

Examples:

```
assert(vge.extensionVersionMinimum(1, 0, 0), “not supported”)
```

Per Node Query & ACL Control

» vge.aclNodeReadable(nodeName)

Functions:

Checking if the given node allows its variables to be read by the node which is currently executing the script.

Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node to be checked.

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	The node allows its variables to be read.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow read access.

Examples:

```
if vge.aclNodeReadable(“npc-1 stat”) then
  -- do something
end
```

» vge.aclNodeWriteable(nodeName)

Functions:

Checking if the given node allows its variables to be written by the node which is currently executing the script.

Parameters:

(as in vge.aclNodeReadable)

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	The node allows its variables to be written.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow write access.

» vge.aclNodePenetrateable(nodeName)

Functions:

Checking if the given node allows its existing variables to be deleted and new variables to be added by the node which is currently executing the script.

Parameters:

(as in vge.aclNodeReadable)

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	The node allows existing variables to be deleted. The node allows new variables to be added.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow penetration access.

» vge.aclNodeAllowRead(allowedGroupName)

Functions:

Modifying the per node ACL rules of the node which is currently executing the script. Note that it will not check for the existence of the given group name.

Parameters:

Names	Types	Descriptions
allowedGroupName	string	The group name to be added to the node's ACL allowance rule. A special string “[ALL]” can be used to refer to all groups.

Returns:

N/A

Examples:

```
vge.aclNodeAllowRead(“[ALL]”)

vge.aclNodeAllowRead(“party member”)
```

» vge.aclNodeDenyRead(deniedGroupName)

Functions:

(as in vge.aclNodeAllowRead)

Parameters:

Names	Types	Descriptions
deniedGroupName	string	The group name to be added to the node's ACL denial rule. A special string “[ALL]” can be used to refer to all groups.

Returns:

N/A

» vge.aclNodeAllowWrite(allowedGroupName)
Functions:
(as in vge.aclNodeAllowRead)
Parameters:
(as in vge.aclNodeAllowRead)
Returns:
N/A

» vge.aclNodeDenyWrite(deniedGroupName)
Functions:
(as in vge.aclNodeDenyRead)
Parameters:
(as in vge.aclNodeDenyRead)
Returns:
N/A

» vge.aclNodeAllowPenetrate(allowedGroupName)
Functions:
(as in vge.aclNodeAllowRead)
Parameters:
(as in vge.aclNodeAllowRead)
Returns:
N/A

» vge.aclNodeDenyPenetrate(deniedGroupName)
Functions:
(as in vge.aclNodeDenyRead)
Parameters:
(as in vge.aclNodeDenyRead)
Returns:
N/A

» vge.queryNodeInGroup(groupName)
Functions:
Querying node names that belongs to the groupName. Only nodes that allow at least one of read/write/penetration access per their node ACL will be included in the query result.
Parameters:

Names	Types	Descriptions
groupName	string	The group name to be queried. A special string “[ALL]” can be used to query for all registered nodes.

Returns:
A table indexed with number (starting from 0) containing strings representing the name of the nodes which are belong to the groupName, **or nil if the group name or the node is not found**.
Examples:

```
t = vge.queryNodeInGroup(“treasure box”)
if t ~= nil then
  i = 0
  repeat
    print(i, t[i])
    i = i + 1
  until t[i] == nil
end
```

Variable Creation and Deletion

» vge.addVar(nodeName, varName, varType)

Functions:

Adding (creating) a new variable in the given node.

Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node in where the new variable will be created. Using the name of an unregistered node will return error. A special string “[SELF]” can be used to refer to the node which is currently executing the script.
varName	string	The name of the variable to be created.
varType	number	The type of the variable to be created, can be one of: vge.typeBoolean vge.typeNumber vge.typeString

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	Success.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow adding variables.
vge.errorAlreadyExist	The variable varName already exist.
vge.errorInvalidParameter	The varType contains an invalid code.

Examples:

```
r = vge.addVar(“[SELF]”, “happy”, vge.typeBoolean)
if r ~= vge.ok then
  -- handle the error
end

r = vge.addVar(“party”, “acquired skill points”, vge.typeNumber)
if r ~= vge.ok then
  -- handle the error
end
```

» vge.delVar(nodeName, varName)

Functions:

Deleting a variable in the given node.

Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node owning the variable. Using the name of an unregistered node will return error. A special string “[SELF]” can be used to refer to the node which is currently executing the script.
varName	string	The name of the variable to be deleted.

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	Success.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow deleting variables.
vge.errorNotExist	The variable varName is not exist.

Examples:

```
r = vge.delVar(“party”, “already met the wizard”)
if r ~= vge.ok then
  -- handle the error
end
```

» vge.hasVar(nodeName, varName[, varType])

Functions:

Checking if a specific variable is owned by the given node.

Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node to be checked.
varName	string	The name of the variable to be checked.
varType	number	Optional parameter which state the type of the variable to be read, can be one of: vge.typeAny (the default value if omitted) vge.typeBoolean vge.typeNumber vge.typeString

Returns:

One boolean, true if the node and the variable are exist and the type is match, false otherwise. Note that it will not check the ACL.

Examples:

```
if vge.hasVar(“party”, “gold”) then
  -- do something with the variable
end
```

Per Variable ACL Control

» vge.aclVarAllowRead(varName, allowedGroupName)

Functions:

Modifying the per variable ACL rules of the node which is currently executing the script. Note that it will not check for the existence of the given node.

Parameters:

Names	Types	Descriptions
varName	string	The name of the variable to be controlled.
allowedGroupName	string	The group name to be added to the variable's ACL allowance rule. A special string “[ALL]” can be used to refer to all groups.

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	Success.
vge.errorNotExist	The variable varName is not exist.

Examples:

```
vge.aclVarAllowRead(“gold”, “[ALL]”)

vge.aclNodeAllowRead(“gold”, “party member”)
```

» vge.aclVarDenyRead(varName, deniedGroupName)
Functions:
(as in vge.aclVarAllowRead)
Parameters:

Names	Types	Descriptions
varName	string	The name of the variable to be controlled.
deniedGroupName	string	The group name to be added to the variable's ACL denial rule. A special string “[ALL]” can be used to refer to all groups.

Returns:
(as in vge.aclVarAllowRead)

» vge.aclVarAllowWrite(varName, allowedGroupName)
Functions:
(as in vge.aclVarAllowRead)
Parameters:
(as in vge.aclVarAllowRead)
Returns:
(as in vge.aclVarAllowRead)

» vge.aclVarDenyWrite(varName, deniedGroupName)
Functions:
(as in vge.aclVarDenyRead)
Parameters:
(as in vge.aclVarDenyRead)
Returns:
(as in vge.aclVarDenyRead)

» vge.aclVarAllowPenetrate(varName, allowedGroupName)
Functions:
(as in vge.aclVarAllowRead)
Parameters:
(as in vge.aclVarAllowRead)
Returns:
(as in vge.aclVarAllowRead)

» vge.aclVarDenyPenetrate(varName, deniedGroupName)
Functions:
(as in vge.aclVarDenyRead)
Parameters:
(as in vge.aclVarDenyRead)
Returns:
(as in vge.aclVarDenyRead)

Variable Manipulations

» vge.getVar(nodeName, varName[, varType])
Functions:
Reading the contents of the given variable.
Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node owning the variable.
varName	string	The name of the variable to be read.
varType	number	Optional parameter which state the type of the variable to be read, can be one of: vge.typeAny (the default value if omitted) vge.typeBoolean vge.typeNumber vge.typeString

Returns:

The variable's value (or nil if error) and a code number which can be one of:

Codes	Descriptions
vge.ok	Success.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow read access.
vge.errorNotExist	The variable varName is not exist.

Examples:

```
r,e = vge.getVar(“[SELF]”, “happy”)
if r == nil then
  -- handle the error based on the value of e
end

r,e = vge.getVar(“party”, “gold”, vge.typeNumber)
if r == nil then
  -- handle the error based on the value of e
end
```

» vge.setVar(nodeName, varName, newValue)

Functions:

Setting the contents of the given variable.

Parameters:

Names	Types	Descriptions
nodeName	string	The name of the node owning the variable.
varName	string	The name of the variable to be set.
newValue	any	New value to be put in the variable. If the type of newValue is not match with the variable type, an error will be occurred.

Returns:

One code number, can be one of:

Codes	Descriptions
vge.ok	Success.
vge.errorNodeNotFound	The node is not found in the registration.
vge.errorAccessDenied	The node does not allow write access.
vge.errorNotExist	The variable varName is not exist.
vge.errorInvalidParameter	The type of newValue is not match with the type of the variable to be set.

Examples:

```
e = vge.setVar(“[SELF]”, “happy”, true)
if e ~= vge.ok then
  -- handle the error
end

e = vge.setVar(“party”, “gold”, 1200)
if e ~= vge.ok then
  -- handle the error based on the value of e
end
```

Invoking Action

» vge.act(actionName[, actionParam[, ...]])

Functions:

Performing an action named actionName. An action is specific to the game developer, currently VortexLua does not define any built-in action.

Parameters:

Names	Types	Descriptions
actionName	string	The name of the action to be invoked. Using the name of an unregistered action will return error.
actionParam	any, action dependent	Parameters to be passed to the action. The number of action parameters depends on the action. If not enough parameters given, error will be occurred. If too much parameters given, the extra parameters will be ignored. However, remember that Lua has limitation on the stack thus limiting the maximum number of parameters that can be passed and/or returned.

Returns:

- **The first return value always a code number**, can be one of:

Codes	Descriptions
vge.ok	Success.
vge.actionNotFound	The action is not found in the registration.
vge.errorNotEnoughParameter	Not enough action parameters given.

- **The second return value always the number of returned parameters.**
- **There can be third, fourth, and so (or even no additional return value), it is all dependent on the invoked action.**

Examples:

```
if vge.act(“change party”) ~= vge.ok then
  -- handle the error
end

e,n = vge.act(“ask player name”, “Ryu”) -- Ryu = default name
if e == vge.ok then
  -- do something with n
end

vge.act(“message box”, “Nurse”, “How are you today?”)
```

On the VortexGE library side:

1. A CLuaExecutionSpace class uses an instance of CLuaActionList to store pairs of action names and their callback functions.
2. If the pairs list in the CLuaExecutionSpace (node) which is currently executing the script is empty, the pairs list in the managing CLuaManager instance will be used.
3. If the pairs list in the CLuaManager is also empty, VortexGE will raise exception to indicate that no action can be performed.
4. If at least one of the pairs list (in point 2 & 3 above) is not empty and the requested action name is not found, vge.act() will returns error code.

The prototype of the callback function must be defined like:

```
CLuaActionList::EActionCallbackResult myCallbackFunction(
          CLuaExecutionSpace       &executingNode,
    const CLuaActionCallbackParams &actionInParams,
          CLuaActionCallbackParams &actionOutParams
);
```

Parameters	Descriptions
executingNode	The node which is currently executing the script.
actionInParams	Parameters which were passed when calling “vge.act”.
actionOutParams	Values that will be returned to VortexLua.

The callback function must return one of the values defined in the CLuaActionList::EActionCallbackResult enumeration.

More on Special Group and/or Node Names

[NONE]

Can be used in all functions which modify ACL rules like :

- vge.aclNodeAllowWrite()
- vge.aclNodeDenyPenetrate()
- vge.aclVarDenyRead()
- vge.aclVarAllowWrite()
- etc.

to specify that there is no node is allowed/denied to read/write/penetrate.

[ALL]

Can be used in all functions which modify ACL rules like :

- vge.aclNodeAllowWrite()
- vge.aclNodeDenyPenetrate()
- vge.aclVarDenyRead()
- vge.aclVarAllowWrite()
- etc.

to specify that all nodes are allowed/denied to read/write/penetrate.

Can be used in :

- vge.queryNodeInGroup()

to specify that all nodes are to be queried (ignoring their group name).

[SELF]

Can be used in :

- vge.queryNodeInGroup()

to query all nodes which are in the same group as [SELF], if [SELF] group name is not defined then only the node name of [SELF] will be returned.

Can be used in :

- vge.getNodeName()
- vge.getGroupName()

to query the real node/group name of [SELF].

Can be used in :

- vge.addVar()
- vge.hasVar()
- vge.setVar()
- etc.

to specify that the variable to be accessed is owned by the [SELF] node.

Can be used in all functions which query for ACL like :

- vge.aclNodeReadable()
- vge.aclNodeWriteable()
- vge.aclNodePenetrateable()

to query for ACL in the [SELF] node, which (of course) will always return vge.ok (access granted).

Note that [SELF] is a node which is currently executing the script.

[EXE]

Can be used in :

- vge.queryNodeInGroup()

to query all nodes which are in the same group as [EXE], if [EXE] group name is not defined then only the node name of [EXE] will be returned.

Can be used in :

- vge.getNodeName()
- vge.getGroupName()

to query the real node/group name of [EXE].

Can be used in :

- vge.addVar()
- vge.hasVar()
- vge.setVar()
- etc.

to specify that the variable to be accessed is owned by the [EXE] node.

Can be used in all functions which query for ACL like :

- vge.aclNodeReadable()
- vge.aclNodeWriteable()
- vge.aclNodePenetrateable()

to query for ACL in the [EXE] node.

Note that [EXE] is a node (a CLuaExecutionSpace instance) which is paired with the CLuaExecutable instance which is currently being executed by [SELF]. This special node will only make sense if the script execution is being done in the context of a game entity class instance.

ooo OOO ooo