HOOKS

Expression3's API must be used in a specific order, each api functions must be called during a specific hook. If called outside of this hook an error will be thrown.

Hook:	Expression3.RegisterExtension
Description	Extension should be registered here.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadClasses
Description	This is where all none extended classes must be defined and registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadExtendedClasses
Description	This is where all extended classes must be defined and registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadConstructors
Description	This is where all class constructors must be registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadMethods
Description	This is where all class methods must be registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadAtributes
Description	This is where all class attributes must be registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadLibraries
Description	This is where all libraries must be registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.LoadFunctions
Description	This is where all library functions must be registered.
Returns:	
Input:	
[SHARED]	

Hook:	Expression3.PostRegisterExtensions
Description	Called after all extensions have been registered.
Returns:	

Input:	
[SHARED]	

Hook:	Expression3.PostCompile.System.**<function>**
Description	Replace function with the name of a function on the system library. This hook acts as the compiler instruction for that function on the system library.
Returns:	● **String**- The return types unique id. ● **Int**- The total amount of values returned.
Input:	● **Compiler**- The instance of the compiler. ● **Instruction**- The instruction for the function call operation. ● **Token**- The first token of the operation. ● **(...)**- A list of instructions, each one is an argument to the function.
[SHARED]	

EXPR_LIB

These functions all exist on the global EXPR_LIB namespace This is the core library and no data outside the defined API should be altered in this namespace. Each function on the API should be called during its related hooks ownly.

Functions

Name:	EXPR_LIB.SetServerState
Description	Sets the state of the next registered operator, method or function to server side only.
Returns:	
Input:	
[SHARED]	

Name:	EXPR_LIB.SetClientState
Description	Sets the state of the next registered operator, method or function to clide side only.

Returns:	
Input:	
[SHARED]	

Name:	EXPR_LIB.SetSharedState
Description	Sets the state of the next registered operator, method or function to server side and clientside.
Returns:	
Input:	
[SHARED]	

Name:	EXPR_LIB.SetPrice
Description	Sets the price of the next registered operator, method or function.
Returns:	
Input:	● **Int** - The price in performance points, required to run the next set of operations.
[SHARED]	

Name:	EXPR_LIB.RegisterPermission
Description	Registers a client side permission node, this is used to allow players to select what can be done client side.
Returns:	
Input:	● **String**- The given nice name of the node. ● **String**- The Icon used in gui for this node. ● **String**- The description used in the gui for this node.
[SHARED]	

Name:	EXPR_LIB.RegisterClass
Description	Registers a new class with E3 and returns an api object representing that class.

Returns:	E3Class - A new E3 class object.
Input:	● **String**- The unique id name of the class. Any id longer then 1 character in length with be prefixed with a n underscore. ● **String**- The inline type name of the class. ● **String**- A function that takes an object and returns true if the object is an instance of the e3 class. ● **Function**- A function that takes an instance of this class and returns true if the class is valid.
[SHARED]	

Name:	EXPR_LIB.RegisterExtendedClass
Description	Registers a new class with E3 extending another e3 class and returns an api object representing that class.
Returns:	E3Class - A new E3 class object.
Input:	● **String**- The unique id name of the class. Any id longer then 1 character in length with be prefixed with a n underscore. ● **String**- The inline type name of the class. ● **String**- The unique id name of the class to extend from. ● **Function**- A function that takes an object and returns true if the object is an instance of the e3 class. ● **String**- A function that takes an instance of this class and returns true if the class is valid.
[SHARED]	

Name:	EXPR_LIB.RegisterWiredInport
Description	Registers a wire in port method onto an e3 class.
Returns:	
Input:	● **String**- The unique id name of the class. ● **String**- The wire port type as defined by wiremod. ● **Function** - A function that takes an object of this class returns a wire compatible object
[SHARED]	

Name:	EXPR_LIB.RegisterWiredOutport

Description	Registers a wire out port method onto an e3 class.
Returns:	
Input:	● **String**- The unique id name of the class. ● **String**- The wire port type as defined by wiremod. ● **Function** - A function that takes a wire object and returns an e3 object.
[SHARED]	

Name:	EXPR_LIB.RegisterConstructor
Description	Registers a construction operator for a class.
Returns:	
Input:	● **String**- The unique id name of the class this constructor is for. ● **String**- a list of unique ids of each input to this constructor separated by commas. ● **Function** - The function e3 will call to create this object. This functions takes all the inputs from the call to the constructor. This function must return a new e3 object. ● **Boolean**- If true then the first argument to the above function will always be the global context of the executing gate.
[SHARED]	

Name:	EXPR_LIB.RegisterMethod
Description	Registers a method operator for a class.
Returns:	
Input:	● **String**- The unique id name of the class this method is on. ● **String**- a list of unique ids of each input to this constructor separated by commas. ● **String**- The unique id of the return class or an empty string. ● **Int**- total amount of values returned on the stack or 0. ● **Function** - The function e3 will call to create this object. This functions takes the global context of the executing gate, the object the method was called on, followed by the inputs. ● **Boolean**- If true then the first argument to the above function will be omitted when called.
[SHARED]	

Name:	EXPR_LIB.RegisterAtribute
Description	Registers an attribute on a class. This attribute will be set on the e3 instance of the class.
Returns:	
Input:	● **String**- The unique id name of the class this attributes on. ● **String**-The name of the attribute.. ● **String**-The name id of the attributes class type. ● **String**-The native lua index on the e3 object to use.
[SHARED]	

Name:	EXPR_LIB.RegisterOperator
Description	This will register a new operator with e3.
Returns:	
Input:	● **String**- The type of operation. ● **String**- a list of unique ids of each input to this operator separated by commas. ● **String**- The unique id of the return class or an empty string. ● **Int**- total amount of values returned on the stack or 0. ● **Function** - The function e3 will call for this operator. This functions takes the global context of the executing gate, followed by the inputs. ; If set to null, and the operator type is supported natively in lua, e3 will use the native lua operation directly instead, this is recommended where possible. ● **Boolean**- If true then the first argument to the above function will be omitted when called.
[SHARED]	

Name:	EXPR_LIB.RegisterCastingOperator
Description	Registers a casting operator, used to transform between classes
Returns:	
Input:	● **String**- The unique id of the class this operators casts to. ● **String**-The unique id of the class this operator cats from. ● **String**-The name id of the attributes class type. ● **Function** - The function e3 will call for this operator. This functions

	takes the global context of the executing gate, followed by the object to cast from. ● **Boolean**- If true then the first argument to the above function will be omitted when called.
[SHARED]	

Name:	EXPR_LIB.RegisterLibrary
Description	This will register a new library definition with e3.
Returns:	
Input:	● **String**- The name of the library.
[SHARED]	

Name:	EXPR_LIB.RegisterFunction
Description	Registers a new function to an existing e3 library.
Returns:	
Input:	● **String**- The name of the library to register the function on. ● **String**- The name of the function on the library.. ● **String**- a list of unique ids of each input to this operator separated by commas. ● **String**- The unique id of the return class or an empty string. ● **Int**- total amount of values returned on the stack or 0. ● **Function** - The function e3 will call for this operator. This functions takes the global context of the executing gate, followed by the inputs. ● **Boolean**- If true then the first argument to the above function will be omitted when called.
[SHARED]	

Name:	EXPR_LIB.GetAllClasses
Description	Returns a copy of the e3 class registry.
Returns:	**Table**- Where each index is the unique id of a class, and the value is the registered api class object.
Input:	
[SHARED]	

Name:	EXPR_LIB.GetClass
Description	Returns an e3 class from the e3 class registry registry.
Returns:	E3Class - The E3 class api object.
Input:	
[SHARED]	

Name:	EXPR_LIB.RegisterExtension
Description	Registers a new E3 extension, this is a wrapper for the api.
Returns:	E3 extension- the new new E3 extension
Input:	**String**- Name of the extension.
[SHARED]	

Registering Operators

An operator sets the way E3 handles logical, binary and mathematical operations.

Operator Types

Type	**Inputs**	**example**		
and	A, B	A && B	Logical and	[NATIVE]
or	A, B	A || B	Logical or	[NATIVE]
add	A, B	A + B	Addition	[NATIVE]
sub	A, B	A - B	Subtraction	[NATIVE]
div	A, B	A \ B	Division	[NATIVE]
mul	A, B	A * B	Multiply	[NATIVE]
exp	A, B	A ^ B	Exponent	
mod	A, B	A % B	Modulo	
ten	A, B, C	A ? B : C	Tinenairy	

bxor	A, B	A ^^ B	Binary xor	
bor	A, B	A | B	Binary or	
band	A, B	A & B	Binary and	
neg	A, B	A != B	Not equal	
eg	A, B	A == B	Equal	[NATIVE]
lth	A, B	A < B	Less than	[NATIVE]
leg	A, B	A <= B	Less or Equal	[NATIVE]
gth	A, B	A > B	Greater than	[NATIVE]
geq	A, B	A >= B	Greater or equal	[NATIVE]
bshl	A, B	A >> B	Binary shift left	
bshr	A, B	A << B	Binary shift right	
neg	A	-A	Negative	[NATIVE]
not	A	!A	Not	[NATIVE]
len	A	#A	Length	[NATIVE]
call	A, ...	A(...)	Invoke	[NATIVE]

E3 Extension Object

The E3 extension object is a wrapper for the api, this object is used to register things with the api, at the same time. The wrapper will perform the operations at the appropriate times, as well as provide a user option for this extension to be disabled.
Since this is a wrapper class, all the functions on this object point to their respective functions on the EXPR_LIB name space.

- Extension:SetPrice
- Extension:SetServerState
- Extension:SetSharedState
- Extension:SetClientState
- Extension:RegisterPermission
- Extension:RegisterClass
- Extension:RegisterExtendedClass
- Extension:RegisterWiredInport
- Extension:RegisterWiredOutport
- Extension:RegisterConstructor
- Extension:RegisterMethod

- Extension:RegisterAtribute
- Extension:RegisterOperator
- Extension:RegisterLibrary
- Extension:RegisterFunction
- Extension:CheckRegistration
- Extension:RegisterConstructor

Name:	Extension:EnableExtension
Description	Enables the extion object, it will queue all operations to register at the appropriate times.
Returns:	
Input:	
[SHARED]	

Example Extensions

Extensions must be created using *EXPR_LIB.RegisterExtension* with in the *Expression3.RegisterExtension* hook. Saving your extensions lua file, in the extensions folder will cause it to load at this time.

For this set of examples, focuses on a simple extension to add a math library.

```
local extension = EXPR_LIB.RegisterExtension("math");

extension:RegisterLibrary(“math”);
```

Two functions are registered using the existing native lua math library. Each function takes one int and returns 1 int..

```
extension:RegisterFunction("math", "sin", "n", "n", 1, math.sin, true);

extension:RegisterFunction("math", "cos", "n", "n", 1, math.sin, true);
```

The extension needs to be enabled to, before it can be loaded into the compiler.

```
extension:EnableExtension();
```

For this next set of examples, focuses on a vector extension and class. Again a new extension is defined.

```
local extension = EXPR_LIB.RegisterExtension("vector");
```

When registering the vector object, two functions are required a type checker and a null checker.

```
local function notNil(v)
	return v ~= nil;
End

local function notNil(v)
	return v and v.x and v.y and v.z;
end

extension:RegisterClass("v", "vector", isvector, notNil)
```

A constructor is used to allow the user to define a new vector object taking 3 integers.

```
extension:RegisterConstructor("v", "n,n,n", function(x,y,z) return {x = x, y = y, z = z} end, true);
```

We can allow the user access to the coordinates as attributes.

```
extension:RegisterAtribute("v", "x", “x”);
extension:RegisterAtribute("v", "y", “y”);
extension:RegisterAtribute("v", "z", “z”);
```

A set of basic operators are defined, such as logical and mathematical operators.

```
extension:RegisterOperator("eq", "v,v", "b", 4, function(a, b)
	return (a.x == b.x) and (a.y == b.y) and (a.z == b.z);
end, true);

extension:RegisterOperator("neq", "v,v", "b", 4, function(a, b)
	return (a.x ~= b.x) and (a.y ~= b.y) and (a.z ~= b.z);
end, true);

extension:RegisterOperator("add", "v,v", "v", 1, function(a, b)
	return {x = (a.x + b.x), y = (a.y + b.y), z = (a.z + b.z)};
end, true);
```

```
extension:RegisterOperator("sub", "v,v", "v", 1, function(a, b)
	return {x = (a.x - b.x), y = (a.y - b.y), z = (a.z - b.z)};
end, true);

extension:RegisterOperator("neg", "v", "v", 1, function(a)
	return {x = (-a.x), y = (-a.y), z = (-a.z)};
end, true);
```

Methods can be added to classes, in this example the vector class has a clone method that will return a duplicate object.

```
extension:RegisterMethod("v", "clone", "", "v", 1, function(v)
	return {x = v.x, y = v.y, z = v.z};
end, true);
```

The extension needs to be enabled to, before it can be loaded into the compiler.

```
extension:EnableExtension();
```