

Declarative Internal DSLs in Lua

A Game-Changing Experience

Alexander Gladyshev, ag@logiceditor.com
LogicEditor.com CTO, co-founder

Lua Workshop 2011

Outline

Introduction

Ad-hoc approach

More realistic case

The "proper" solution

Internal Declarative DSL in Lua

```
namespace:method "title"  
{  
  data = "here";  
}
```

...Without sugar

```
_G["namespace"]:method(  
    "title"  
    ) ({  
    ["data"] = "here";  
    })
```

Naïve implementation

```
namespace = { }  
  
namespace.method = function(self, name)  
  return function(data)  
    -- ...do something  
    -- ...with name and data  
  end  
end
```

Hyphotetic UI description language

```
ui:dialog "alert"  
{  
  ui:label "message";  
  ui:button "OK"  
  {  
    on_click = function(self)  
      self:close()  
    end;  
  };  
}
```

UI description language "implementation", I

```
function ui:label(title)
  return function(data)
    return GUI.Label:new(title, data)
  end
end

function ui:button(title)
  return function(data)
    return GUI.Button:new(title, data)
  end
end
```

UI description language "implementation", II

```
function ui:dialog(title)
  return function(data)
    local dialog = GUI.Dialog:new(title)
    for i = 1, #data do
      dialog:add_child(data)
    end
    return dialog
  end
end
```


Ad-hoc approach

- + Easy to code simple stuff

But:

- Easily grows out of control
- Difficult to reuse
- Hard to handle errors
- Hard to add new output targets

Practical example: HTTP handler

```
api:url "/reverse"  
{  
  doc:description [[String reverser]]  
  [[  
    Takes a string and reverses it.  
  ]]  
  api:input { data:string "text" };  
  api:output  
  {  
    data:node "result" { data:string "reversed" };  
  };  
  handler = function(param)  
    return { reversed = param.text:reverse() }  
  end;  
}
```

What do we want to get from that description?

- ▶ **HTTP request handler itself**, with:
 - ▶ Input validation
 - ▶ Multi-format output serialization (JSON, XML, ...)
 - ▶ Handler code static checks (globals, ...)
- ▶ **Documentation**
- ▶ Low-level networking **client code**
- ▶ Smoke **tests**

Request handler: input validation

```
INPUT_LOADERS["/reverse"] = function(checker, param)
  return
  {
    text = check.string(param, "string");
  }
end
```

Request handler: output serialization

```
local build_formatter = function(fmt)
  return fmt:node("nil", "result")
  {
    fmt:attribute("reversed");
  }
end

OUTPUT["/reverse.xml"] = build_formatter(
  make_xml_formatter_builder()
):commit()

OUTPUT["/reverse.json"] = build_formatter(
  make_json_formatter_builder()
):commit()
```

Request handler: the handler itself

- Handler code is checked for access to illegal globals.*
- Legal globals are aliased to locals at the top.*
- Necessary require() calls are added automatically.*

```
local handler = function(param)
  return
  {
    reversed = param.text:reverse();
  }
end
```

```
HANDLERS["/reverse.xml"] = handler;
HANDLERS["/reverse.json"] = handler;
```

Documentation

/reverse.{xml,json}: String reverser

Takes a string and reverses it.

IN

?text=STRING

OUT

XML:

```
<result reversed="STRING" />
```

JSON:

```
{ "result": { "reversed": "STRING" } }
```

Smoke tests

```
test:case "/reverse.xml:smoke.ok" (function()  
  local reply = assert(http.GET(  
    TEST_HOST .. "/reverse.xml?text=Foo")  
  ))  
  assert(type(reply.result) == "table")  
  assert(type(reply.result.reversed) == "string")  
end)
```


Too complicated for ad-hoc solution!

TODO: Spaghetti cat image goes here

The "proper" solution?

- ▶ Should be easy to add a new target.
- ▶ Should have nicer error reporting.
- ▶ Should be reusable.

The flow

- ▶ Load data
- ▶ Validate correctness
- ▶ Generate output

Let's recap how our data looks like

```
api:url "/reverse"  
{  
  doc:description [[String reverser]]  
  [[  
    Takes a string and reverses it.  
  ]]  
  api:input { data:string "text" };  
  api:output  
  {  
    data:node "result" { data:string "reversed" };  
  };  
  handler = function(param)  
    return { reversed = param.text:reverse() }  
  end;  
}
```

Surprise! It's a tree!

```
{ id = "api:url", name = "/reverse";  
  { id = "doc:description", name = "String reverser";  
    text = "Takes a string and reverses it.";  
  };  
  { id = "api:input";  
    { id = "data:string", name = "text" };  
  };  
  { id = "api:output";  
    { id = "data:node", name = "result";  
      { id = "data:string", name = "reversed" };  
    };  
    handler = function(param)  
      return { reversed = param.text:reverse() }  
    end;  
  };  
}
```

We need a loader, that does this: (I)

```
namespace:method "title"  
{  
  data = "here";  
}  
  
⇒  
  
{  
  id = "namespace:method";  
  title = "title";  
  data = "here";  
}
```

We need a loader, that does this: (II)

```
namespace:method "title"    ⇒    {  
                                id = "namespace:method";  
                                title = "title";  
                                }
```

We need a loader, that does this: (III)

```
namespace:method  
{  
  data = "here";  
}
```

```
⇒ {  
  id = "namespace:method";  
  data = "here";  
}
```


We need a loader, that does this: (IV)

```
namespace:method "title"  
[[  
  text  
]]
```

⇒

```
{  
  id = "namespace:method";  
  title = "title";  
  text = [[  
    text  
  ]];  
}
```

We need a loader, that does this: (V)

```
namespace:method "title"(function()  
  -- do something  
end)
```

⇒

```
{  
  id = "namespace:method";  
  title = "title";  
  handler = function()  
    -- do something  
  end;  
}
```

...And adds some debugging info for nice error messages:

```
-- my_dsl.lua:
42: namespace:method "title" ⇒
43: {
44:   data = "here";
45: }

{
  id = "namespace:method";
  title = "title";
  data = "here";
  file_ = "my_dsl.lua";
  line_ = 42;
}
```

Nested nodes just... nest:

```
namespace:method "title"  
{  
  data = "here";  
  foo:bar "baz_1";  
  foo:bar "baz_2";  
}
```

⇒

```
{  
  id = "namespace:method";  
  title = "title";  
  data = "here";  
  
  { id = "foo:bar", name = "baz_1" };  
  { id = "foo:bar", name = "baz_2" };  
}
```