CODEFLOW
an advanced IDE for Lua

LUA WORKSHOP 2015
October 16, 2015

Jean-Luc Jumpertz  founder celedev  @JLJump  @celedev
CodeFlow
Interactive Development of iOS / OS X Apps in Lua

- Instant feedback on real devices
- Full SDK scope
- Powerful dev tools

⇒ Faster development cycle
⇒ Easier experimentation, higher creativity
⇒ Better App UX
CodeFlow
Interactive Development of iOS / OS X Apps in Lua
CodeFlow
Interactive Development of iOS / OS X Apps in Lua
CodeFlow
Interactive Development of iOS / OS X Apps in Lua
Lua code editor

• Code editor common features: code coloring, indentation, completion…

• **Standard editors** manage code purely as text

  • 1 file / language
  • regex, list of keywords…

  ➤ The editor has no real knowledge of the program structure!

  ➤ Limited features set
Syntax-driven Lua code editor

• The Lua Abstract Syntax Tree represents the syntax of a given Lua source file
• Each AST node stores the associated char range in the source file
• The AST must be kept in sync with the source code (asynchronously)

➡ All syntax-related actions in the editor use the corresponding AST
➡ Much more powerful and accurate than text-based processing
Syntax-driven Lua code editor

• Demo
  ➡ An advanced syntax-aware Lua editor can be a true help for the developer!

• Under the hood: building the AST
  • Many possible strategies
  • In CodeFlow, Lua AST generation is based on the standard Lua parser (liblex.c, lparser.c)
    ➡ very good performance
    ➡ same language understanding as the Lua runtime
  • Source available at bitbucket.org/jean_luc/luasyntaxer
Lua debugger

• For many developers, prints in the console is still the primary debug tool. 😞

• A good debugger makes bug fixing much easier!

• The ideal debugger should be…
  ➡ Easy to use and well-integrated with the rest of the IDE
  ➡ Fast, low overhead
  ➡ Transparent to use
    i.e not forcing you to change your program code to make it debug-compliant
Debugger features

• step-by-step execution and breakpoints
• runtime errors analysis (+ make them recoverable)
• variable inspector / editor
• stack-context-aware command console
• multi-thread debugging
Lua Modules in CodeFlow

- Managing modules is a classic role of an IDE
- A logical choice for Lua modules interaction: use the `require` function
- In CodeFlow, Lua modules are in the IDE space; but `require` is called from the app on the device
  - Custom `require` function in the CodeFlow Runtime
- Behavior changes
  - Enable multiple return values
  - Reload a module if *syntactically changed* in the IDE
Automatic Bindings generation

- In the target apps, Lua code interacts heavily with C / ObjC APIs
- May needs to call the System SDK, but also custom APIs defined by the target app
- CodeFlow automatically generates *project* bindings for custom APIs defined in the app’s Xcode project.

Integration in the IDE
- auto-completion of *project* bindings APIs in the editor
- inspection of *project* bindings types in the debugger
As a conclusion

• The real value of an IDE comes from integration of its components

• A dynamic language like Lua comes with specific challenges for the developer, that a dedicated IDE can help to address.
Questions?

www.celedev.com
github.com/celedev
bitbucket.org/jean_luc/luasyntaxer

Jean-Luc Jumpertz
contact@celedev.com
@JLJump

Want to try? Download CodeFlow at celedev.com/en/support