TSConfig & Lua Config

Better together

C++ vs. Lua

- Use the TSConfig C++ API (roughly) as an interface to Lua configuration data.
- TSConfig interface takes a path or buffer, along with a list of global symbols.
- The content is parsed / executed by Lua.
- TSConfig copies the data rooted at the specific symbols to internal data
- A C++ tree / value interface lets this data be examined by the caller.

Notes

- Requiring every component to build Lua support is not so good.
- I prefer a generic pull model (C++ logic does the equivalent of ‘dump’) vs. a specialized Lua API for each component.
- Have done some experimentation, still working on the precise mechanism for doing the dump to C++.

Continuation Tracking

We know where you came from

Tracking Continuation Sources

- Each continuation has a link back to a plugin registration.
- Core maintains a “plugin context” in a thread local variable that tracks the currently active plugin.
  - Push on plugin call, pop on return.
  - TSContCreate uses this to set the plugin field of the new continuation.
- Prototype implementation as part of the plugin priority work.
- Remap plugins are also tracked in this way.
  - Base reference stored during remap.config processing.
  - Super class PluginInfo to create registration data structure.

Goals and Features

- Debugging
  - While running can examine which plugin created the continuation.
  - Error / warning messages can describe the responsible plugin
    - “Use of deleted continuation”
- Continuation counts
  - Track the total # of outstanding continuations per plugin.
  - Leak detection with plugin localization.
- Plugin reload
  - Create new registration data.
  - Mark old registration as “dead” then skip events on “dead” plugins.

Overridable Configuration

Leif has it exactly backwards

Goal: Plugin API to override configuration per transaction

- Problem when overridable data is in a subsystem.
  - Duplicate in HttpSM and pass in locally.
  - Pass in HttpSM configuration structure to subsystem
    - Circular dependency! Yay!

My approach

- Each subsystem defines a struct that is the per transaction ("local") configuration values.
- HttpSM override struct inherits the subsystem struct.
- At run time the HttpSM configuration struct is static_cast to the subsystem struct and passed down.
- No more circular dependencies. Yay!
- Easier maintenance.