TSL Language
Aos Engine Family
SQLite TSL Integration
Year-2

David Simmons, Smallscript Corp
Tcl 2017 Conference
• “af” procedures
  • af: normal
  • afd: deterministic
  • afEach: normal tables
  • afEach: deterministic tables

• Deterministic means
  • cacheable idempotent

• SQLite Api Integration
  • build flags
  • a few other APIs used as hooks
**SQLite: UUID - Universally unique identifier**

- **UUID**

```
123e4567-e89b-12d3-a456-426655440000
xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx
```

```python
uuid-create-sequential()
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Length (Bytes)</th>
<th>Length (Hex Digits)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_low</td>
<td>4</td>
<td>8</td>
<td>integer giving the low 32 bits of the time</td>
</tr>
<tr>
<td>time_mid</td>
<td>2</td>
<td>4</td>
<td>integer giving the middle 16 bits of the time</td>
</tr>
<tr>
<td>time_hi_and_version</td>
<td>2</td>
<td>4</td>
<td>4-bit &quot;version&quot; in the most significant bits, followed by the high 12 bits of the time</td>
</tr>
<tr>
<td>clock_seq_hi_and_res</td>
<td>2</td>
<td>4</td>
<td>1-3 bit &quot;variant&quot; in the most significant bits, followed by the 13-15 bit clock sequence</td>
</tr>
<tr>
<td>clock_seq_low</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>node</td>
<td>6</td>
<td>12</td>
<td>the 48-bit node id</td>
</tr>
</tbody>
</table>
• **Time Conversion**
  - %J: Julian days as float
  - The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar).

• **time-fmt**: tsl-command
  - custom time parsing and formatting library
  - parses most text forms
  - parses numbers as either epoch seconds in epoch, uuid or FileTime, time 10^-7 64-bit time

<table>
<thead>
<tr>
<th>Function</th>
<th>Equivalent strftime()</th>
</tr>
</thead>
<tbody>
<tr>
<td>date(...)</td>
<td>strftime(&quot;%Y-%m-%d&quot;, ...)</td>
</tr>
<tr>
<td>time(...)</td>
<td>strftime(&quot;%H:%M:%S&quot;, ...)</td>
</tr>
<tr>
<td>datetime(...)</td>
<td>strftime(&quot;%Y-%m-%d %H:%M:%S&quot;, ...)</td>
</tr>
<tr>
<td>julianday(...)</td>
<td>strftime(&quot;%J&quot;, ...)</td>
</tr>
</tbody>
</table>

• time-fmt({now +1 day -2 hours -7 min, {\%J}});
• time-fmt(connection.time-created, {\%Y/%m/%d \%H:%M:%S.%f UTC});
• time-fmt(uuid-create-sequential(), {\%Y/%m/%d \%H:%M:%S.%f UTC});
SQLite: JSON Interchange & Ad-Hoc Records

- JSON1 Library
  - Json data in records
    - jFieldName
  - Json as tables in sql expression:
- CRUD triggers on columns and indexes
- JSON1 (flaws)
  - Empty and Nullable
  - Caching

There are two aggregate SQL functions:

14. `json_group_array(value)`
15. `json_group_object(name, value)`

The two table-valued functions are:

16. `json_each(json)`
17. `json_tree(json)`
**Sqlite: sql triggers**

```sql
1 CREATE TRIGGER aAccount_DELETE 
   AFTER DELETE 
   ON aAccount 
   WHEN old.aAccountId NOT NULL 
BEGIN 
   DELETE FROM aPhoneNumber 
   WHERE aAccountId = old.aAccountId; 
   UPDATE aLxNumber 
   SET aAccountId = NULL 
   WHERE aAccountId = old.aAccountId; 
END; 

2 CREATE TRIGGER aAccount_INSERT 
   AFTER INSERT 
   ON aAccount 
BEGIN 
   aAccount 
   UPDATE aLxNumber 
   SET lxNumber = json_extract(new.jAccount, '$.lxNumber'), 
   ppdNumber = json_extract(new.jAccount, '$.ppdNumber'), 
   lpdNumber = json_extract(new.jAccount, '$.lpdNumber'), 
   email = json_extract(new.jAccount, '$.email'), 
   imeiNumber = json_extract(new.jAccount, '$.imeiNumber'), 
   iccidNumber = json_extract(new.jAccount, '$.iccidNumber'), 
   jAccount = json_set(new.jAccount, '$.aAccountRid', new.aAccountRid) 
WHERE rowid = new.rowid /* AND new.jAccount != old.jAccount */ AND 
json_valid(jAccount); 
```
SQLite: sql query using .af command

```javascript
this.query() {
  SELECT rowid AS [~]
  FROM acl.user AS a
  WHERE afd("acl-afxDbAccountMatch", a.uid, a.login, a.info, a.cap, a.pw)
} { break; }
```
SQLite: Integrating TSL

- sqlite "lib"
  - built w/ JSON1
    - convinced a number of the popular open source tools to add JSON1 to their nightlies
- afm built as a single "lib"
  - built from a single "cpp" file
  - Linked as "afm.exe" console subsystem coffbox
    - Renaming "afm.exe" a .SO or .DLL
    - Exporting standard DLL API interfaces
- AOS/S# COFF binary ffi-thunk bBuilder
SqLite: Packages “DBs as vfs Pkg”

- SqLite Header
  - SiteId, AppId
- Common DB extension forms
  - .afx, .afr, .dpk, .zdb [FsPath VFS model]
sqlite_master
  • VersionSchema table concepts
    • VersionSchemaTrigger table

db-resident scripts
  • StoredProcedures
    • Where to place stored procedure code depends whether a vfs-repo-fs table model is available

Undefined sql API hook (NFH)
  • Allows lazy NFH lookup into existing environment, or lazy load from db stored procedure tables
End of Slides
TSL Language
Aos Engine Family

Devops and the CoffBox Model
Year-2

David Simmons, Smallscript Corp
Tcl 2017 Conference
• Executable Components
  • afm.exe
  • afm-symlinked-name.exe
  • executable-db-vfs-pkg.afx
    • usually a “.afr” that’s been symlinked
    • shebang-#!path or pathext-ext registered.
  • script-name.afts

• Registering script types
  • Windows
    • PATHEXT
    • ftype, assoc
    • registry twiddling for overloading
  • nix*
    • shebang - #!path header
    • chmod
CoffBox: Private Cloud

- **My home office environment**  
  (by way of example)
  - Networks
    - 10GB Fiber
    - Mesh WiFi
    - ZWave, Ethernet-over-power
  - over
    - 70TB NAS storage
    - 20 computers
    - 200 devices
  - OS
    - Linux
    - OSX
    - Windows
    - Others

---

Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016

*Consumer Applications to Represent 63 Percent of Total IoT Applications in 2017*

Gartner, Inc. forecasts that 8.4 billion connected things will be in use worldwide in 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020. Total spending on endpoints and services will reach almost $2 trillion in 2017.
CoffBox: Explore Desktop

- Show Files and Locations
- Demo App
• .af directory patterns
  • .conf customization model
  • search paths
• SymLinks
  • client system relative
    • symlinks, reparse-points
  • host system relative
    • junctions
• absolute and relative paths
• repo-treatment
  • ignored, as-file
  • xattr archive-flag
• Virtual Drives
  • .dmg, .vhdx, linux sparse file images

• Packaging Model
  • .af directory pattern
  • .afws
  • .afr, .afx
  • .afts scripts
CoffBox: Build a script

- script w/command line arg processing
Build an afx w/.conf checked into it.
Showcase demo webserver
Customizing a System

- ChildProcess
  - pipes
    - http(s), message-queues
  - sockets
    - http(s)
- EventLog mechanisms
  - telemetry reporting

Reflecting on a System

- ::system::info
  - elevated, admin, etc
- reg- commands
CoffBox: DevOps Concepts

- Binary build model
  - cpp, hxx, hpp, h
- Single Binary for Exec & DLL (.so)
  - Coff Format
  - Subsystems
  - Sections
    - Resources
    - Read-only Memory Repos

- TSL scripts exported as native COFF library binaries (.so, .dll, .exe)
  - your-code-thunk.dll - thunk-bridges
End of Slides
TSL Language
Aos Engine Family
afm
Year-2

David Simmons, Smallscript Corp
Tcl 2017 Conference
iOT: Purpose Built Constrained Environments

Constraints
• Maybe single app
• Security
• Storage types
• Hw permission and firmware controls
• Memory
• Power Management Lifecycle
  • CPU Cycle Cost
• RTOS Constraints and Models
  • Embedded toolset contraints

Typical metrics
• Memory
• Cpu
• Storage
• Power Management
• Os Services
iOT: engine requirements for iOT deployment

- **Os Platforms**
  - Windows, OSX, iOS, Android, Linux, Linux Embedded, FreeRTOS, nuttx, nucleus rtos
  - see https://www.osrtos.com/

- **Cpu Targets**
  - Arm (32, 64), Intel x86/x64, MIPs
iOT: Device Use Case

• Networking
  • Wifi, Bluetooth, Wired-Ethernet, ZWave, Zigbee

• Web Server/Service
  • Telemetry
  • Updates
  • Device Interop (peer, slave, master)

• UX
  • custom displays
  • custom input/sensors

• VFS
  • Packaging, Versioning
  • Resources, executable components

• Power Management Lifecycle
  • Device On/Off

• Device Firmware Update Models
  • Rollback, Upgrade
  • Build, Package, Sign requirements
  • Flashing Lifecycle Constraints
    • OTA, Wired
iOT: Afm’s iOT ready architecture

- **Coff binary build model**
  - cpp, hxx, hpp, h
  - embedded r/o “.afr”
    - compression

- **Codecs**
  - compression

- **Engine Design**
  - Booting Model
  - Thread Model
    - engine-affinity, fiber-co-operative, rpc message streams
      - pipes, tcp, ssl, http
      - json, html, ...

- **Pal: Host Abstraction**
  - Memory - cpu/kernel direct
    - TLS - CPU direct (mirror OS model)
  - Interrupts, Signals, Exception
  - Debugger Aware Channels

- **Pal: Cpu Abstraction**
  - FPU, Vector
  - MP-Sync Instructions
  - Bit Operations
  - Special Forms

- **Pal: FsPath Pluggable Abstraction**
  - Native File System, (tags, versioning) Fs built on SQLite blobs, Registry, Fuse/Dokan, HTTP(S), BuiltIns

- **Communications**
  - Transports
    - Ethernet, Bluetooth, Serial
  - Sockets
  - DNS, DHCP, HTTP (1.0, 1.1, 2.0 ALPN)
  - SSL, Certs (pal and host integrated)
End of Slides
TSL Language
Aos Engine Family
Language Concepts and Usage
Year-2

David Simmons, Smallscript Corp
Tcl 2017 Conference
Statements: JS & TCL 12 Rules
Disambiguation
**Engine: Model**

**Procs and Threads**
- Process Model
  - multiple engines allowed per process
- Engines
  - engines have thread affinity
  - co-operative threading within an engine
- Namespaces
  - ::super
  - ::system
    - ::afm
  - ::shell
  - ::app
    - ::script

**Code Locations**
- FsPath Concepts
  - SymLinks
    - Archive XAttr-Flag
- Process Loading Model
- Script Binding Model
- Web Site and Page Model
Statements: Statement Tokenization and Substitution

- Phase 0: Text Command Statement
  - Statement Delimiting
    - `else` rules [command must defer-level1]
    - `; eos`
    - `{ } eos`
    - `# eos`
    - `ws` eos [tcl-mode only]
- Phase 1: Command
  - Phase 1a: Expr Args
  - Phase 1b: semicolon args
- Phase 2: Binary Operators
- Phase 3: TCL Parameters

- Comment Forms
  - `// EOL`
  - `/* nestable */`
  - `# - special command, NFH form`
    - Allows #command if NO whitespace
- Word Grouping Rules
  - `( )` first following command
    - allows whitespace
  - `( ) {} ... no other tokens`
    - partitions into discrete words whether whitespace or not
- Deferral Levels 1,2,3
  - `, ?, ... :`
  - `for() ...`
  - `do while {}`
  - `return`
Commands: Forms

• Types
  • proc
    • ::super namespace
    • protected from re-definition
  • func (fn)
    • context bound
  • method
    • this, prototype bound

• Mint-Paths
  • All commands are namespace types and get minted

• Modes
  • default
  • ^ uplevel & native proc^
  • @ modules

• Prototypes
  • Tson Declaration/Merge &:{}

• Declaration Features
  • parameter binding
    • binding (*a, &b, ?z, {x {}})

• Invoke named parameters
  • ( key: value, )

• Observation and NFH
  • Loader Hooks, NFH
Eval: Expressions

• Operators
  • new ...
  • func ...
  • = (op-assign)
• numeric ops
  • .qualifiers for typing
  • ==
    • ===, ===~, !=, !==, !=~
• Short Circuit Boolean Ops
  • ||, &&

• Contexts where it is implicit eval
  • pathx invokes ()
  • pathx indices []
  • tcs-line
    • text command statement
Path Expressions: Operators

- PathX
  - $$,$?,$!,$:
  - &: , &:?
  - delete &
  - && upvar
  - (expr), {}, [expr]
- “”, {}, [] and tcp
  - token/macro rules

- PathX Operators (*incl reserved*)
  - . operator
    - *(() invoke
    - .. cascade
  - :: and ! binding operators
Path-Expressions: VarRefs

- Variable References
- & Operator
  - Stack-Context References
    - &(#)
  - &pathx - deferred binding
  - [expr] => {scalar-key}
  - && Upvar model

- Name Partitioning
  - ...
  - this
  - super
  - @key, key@qualifier
  - {literal-key-closing}
    - escape rules

- Valid Name Patterns
Namespaces:

Foundation
- Namespaces
  - ::super
  - ::system
    - ::afm
  - ::shell
  - ::app
    - ::script
  - ::global
    - modules

Concepts
- Minting
  - namespace command
  - mint-name concepts
  - minting rules
- Where Procs Live
- Where System information is
- User owned
- Names
  - CamelCase
  - Hyphenated
Commands: Categories - patterns

- **Naming Patterns**
  - Case Usage
    - Uppercase - factory *(new)*
    - Lowercase - variable
  - CamelCase - methods
  - Hyphenated - functions
  - Special commentary
    - @names
    - this-... names
    - super

- **Locations**
  - Modules
    - ::super proc space
    - ::app, ::lib
    - reserved
      - ::globals - for local composers
      - ::afm, ::system, ::shell
      - ::app, ::script

- **Module Naming pattern**
  - domain - like package model
  - uuids - anon modularization
Commands: Categories - groupings

- file-, dir-, fs-
- string-
- mint-
- sql-
- time-
- ?put*
- @*
Commands: include, require, fs-find

Key variables
- `::system::path`
- `::app::path`
- `::shell::path`
- `::shell::ext`
- `::shell::jails`

Extensions
- `-ncf`
- `-aei`
End of Slides
What we are going to see this morning

Aos Engine Family
25th Anniversary

Rich Web Servers
- Open Ssl, Certs, Json, Markdown, HTML, TCP, Sockets

DevOps Toolset
- CLI, Ansi, EventLog, Services, Pipes, Exec, Codecs

Repo VFS CMS
- acl-security, filesystem, registry, env, time-codec, uuids

Integrated Sqlite
- 1st class Sql support JSON, Sql Procs

Single Binary
- Config Free
- SymLinks

TSL
Text Scripting Language

AFM
Design and demos

11/17/2017

TSL Talk by David Simmons, Smallscript Corp (c) 2017
TSL Language
Aos Engine Family

David Simmons, Smallscript Corp
Tcl 2017 Conference
Refresher, demos, discussion

For those I did not meet last year, my name is David Simmons

• A year has gone by...
  • This morning we are going to see how TSL/AFM has evolved.
• The beginning of this talk will be a refresher
• Then we will go into a series of demos and discussions
Thoughts to share…

Over the last year, the afm system has been actively serving 10,000+/engine PBX cloud telephony users:

- Single-binary, single db pkg-file
  - coff-box symlink model, single db-file package system (vfs)
  - drop install on any machine.

- Setting up new servers is only a few minute operation
  - fail-over, scaling, and versioned upgrades are mostly automated by simple remote check-in/out.

- Full server and devops set of stabilized features
  - More than is possible to cover in a talk.
On the subject of **Names**

- **What is the acronym “st”?**
  - Site, Smalltalk, Smallscript
- **What is the acronym meaning of “AOS®”?**
  - Agents Object System, Agile Object System, Actor Object System
- **What is the acronym root “af” of “afm” stand for?**
  - collates top of the alphabet 😊
  - Aos-Family, Aos File Manager, Aos Fossil Manager
- **But in all seriousness:**
  - Carefully thought through names and naming conventions are critical to a language design and best practices for code libraries.
My language implementation timeline

1978-1997

- NBS Net/ArpaNet HW w/Basic (6512)
  - Forth, APL, Mix of Shell Languages, many others of the era
- Fortran, Lisp, PostScript, ...
- Message-C, SIAL, Informix SQL’s Full Text engine for International Publishing on BRS/Dialog with TeX
- QKS Smalltalk, Prolog, Scheme,...
  - Embedded in QuickTime
- Newton, AppleScript, Kaleida/SK8, Taligent

1997-2017 (Owner/MSFT Architect)

- Smallscript, S# Language
  - Native engine, .NET Runtime
  - DLR 2000 PDC
- VBA, Visual Basic.NET
- JavaScript
- .NET Mobile Runtime and XNA
- PowerShell
- AFM/TSL
Flashback to the good old days

1997
History: Smalltalk’s failure to be relevant

1986-1996 (gave life to the era of Java)

- Awesomely productive and rich toolset capable of talking to almost anything
- Immense integrated frameworks
- Pioneered many aspects of modern software development from patterns, agile programming, unit testing
- Disaster for working with text and file-based code management and practices
- Monolithic image system unable to integrate within other systems, execution engine scaling issues
- Decompositional Model challenge to schema version, package, and be small and bootstrap from nothing

2007-2017 (Rise of the mobile iOT Device Net)

- Everything connected, people message, and watch/create content on their schedule
- Compute is cheap $8/full watch cpu competitive with 2004 PC
- Embedded, real-time, devices proliferate and with it immense scaling challenges
- HTTP, JSON, HTML-UI dominate compute systems
- Text processing and file-based assets dominate compute design with massive stack-based libraries
- devops challenges and complexity single largest impediment to evolution
Back to the present

2017

Why Text matters more than ever
FsPath system: virtual versioned file system

- Let’s begin by talking about the FsPath system
  - pluggable and defaults to supporting:
    - Disk
      - fully symlink, xattr and stream aware
    - Versioned repo (scm)
      - System core files baked into the afm coff binary as a r/o repo
    - Web based files
  - has a specific path syntax for disambiguating content
    - server-type
      - file-system (default, current directory)
        - streams/xattr support
      - repo vfs
        - branches, tags, versions support
      - af web-server generalized
include command: .tsl; .afts; .md; ...

/~/alias/path/file.ext path aliases are located in ::app::path ^ ::system::path maps

File folders on disk

Branch folders in local repo (versioned)

Branch folders in r/o coff repo

Branch folders in https repo (versioned)

/path/file.ext
N:/path/file.ext
N:/path/dir:xattr-stream.ext

/branch:repo/file.ext ~:/path/file.ext

.afr; .afx; .afws

afm.exe
afx.exe
afts.exe

https://af.st/:branch:repo/

./af/site/

./af/sites/domain
End of Slides
AFM executable

Self-installing single host executable that uses symbolic-link file-naming to determine its execution behavior

The executable’s File Name is used by kernel.tsl code to determine the execution path as it examines the exec command line; this allows efficient creation of single shared binary tools using symbolic-link naming patterns.

Rich Web CMS

Afx Web Servers

TSL Language 1.2

AFM executable

Fossil-SCM

Sqlite

Open Ssl

Repo db(s)

.time-fmt, csv/cst, child-process, certs/ssl, sql-procs, many-codecs, regex, debug-io/events, registry/env

11/17/2017
Variable Path Expressions

JSON Expressions and TCL quirks mode support

Examples:

```
$!{string "Hello"}  \^identifier-exp
$!(17 + 5 % 3)  $receiver-path-exp::namespace-exp
```

Cases of Note:

```
&:{ ... JSON pojo ... }  JSON pojo supporting TSL eval expr extensions and auto-conversion.
&:[ ... JSON poja ... ]  JSON poja supporting TSL eval expr extensions.
```

```
var identifier-path = &&var-ref-path;  Binds two variables, as opposed to assignment.
```
Startup sequence: she`bang-script binding

shebang
ms-pathext

Process Exec

shebang

#! /local/bin/afts
ftype.exe && assoc.exe

registry mapping

main
primary thread

Engine Init
TSL Core Registered

Link-name executed as command, if available

Not-found Hook Invoked to load command

Command Executed as Program

1. script relative .af/
2. exe relative .af/
3. coff system pmsp .af/

afm
Fall-through to fossil-default

①

②
Command re-examined as Message

```
cputl $msg;
# A TCL command with macro-param

cputl(msg)
# A TSL function with expr-param

this.cputl(msg);
# A TSL method with an expr-param
```
TSL Language
Aos Engine Family

David Simmons, Smallscript Corp
Tcl/Tk 2016 Conference
What we are going to briefly tour this afternoon..

- **TSL**: Text Scripting Language
- **AFM**: Its COFF binary executable host

Aos Engine Family
DEMO

afts

Command Line Script
TSL

As a language TSL is founded on the **macro and command patterns used in TCL**

As C++ can be described in relation to C, TSL can be described in relation to TCL

It would be incorrect to describe TSL as a dialect of TCL

---

**TCL**

Macro & Command Patterns

**JavaScript blended syntax & intrinsic JSON**

**Smalltalk, Lisp, Self messaging behavior and**

**Intrinsic C++ object integration**

---

**Executable Libraries:** Source, Op-Codes, and precompiled Machine Code

**Object Model:** prototype, mixins

**Path Binding Engine:** predicate binder

**Execution Invocation:** Interp-JIT, FFI

**Host Integration & Lifecycle Model**

---

**AOS Execution Engine and Container System**

DSL: Domain Specific Language

Frameworks, IDE, Image-Snapshots, Intrinsic-Repos

Perspective Types: Versioning, Modularization, Security

Dynamic Composition Aspect-weaving and introspection
AFM executable

Self-installing single host executable that uses symbolic-link file-naming to determine its execution behavior

The executable’s **File Name** is used by **afconfig** code to determine the execution path as it examines the **exec command line**; this allows efficient creation of single shared binary tools using symbolic-link naming patterns.

1. **afconfig.tsl**
2. **afm**

**TSL Language**

*C/C++ implementation*

**AOS Execution Engine**

System and Libraries

**C/C++ Libraries and Services**

**Files and Directories**

BuiltIn FileSystem

Intrinsic Resources

**C/C++ Libraries and Services**

**Sqlite**

**Fossil-SCM**

**Repo db(s)**

**Intel, ARM, ...**

**COFF (binary bundled) Executable**

*nix, ms-windows
Hello, my name is David Simmons

• For 25 years I’ve had a particular specialization in high-performance hybrid dynamic-static language runtime systems.

• By education, I am an Electrical Engineer and Astrophysist specialized in VLSI design. However, most of my career has been as a Software Engineer and Architect
  • Working on Operating Systems, Hardware Devices, Real Time Systems and Programming Languages and their Execution Machinery and Infrastructure
    • I began my career some 40 years ago, in the summer of 1976 at The National Bureau of Standards which is known today as NIST (The National Institute of Science and Technology) working on Fortran Runtime Real Time Libraries, Perkin Elmer Operating System Development, and ArpaNet NBS Net Packet Switch Hardware and Analyzers.

• Employed working with or for:
  • NBS/NIST, U.S. Congress, Air Force, Faculty at UofMd NSF Systems Research Center, Tokyo Gas, Apple, Suse Linux, Microsoft and worked for or owned a number of small to mid-size companies of up to 250 employees.

• Worked on, collaborated on or was lead architect on the following language systems:
  • Apple’s Kaleida, Apple’s AppleScript, Apple’s Quicktime-Track-Scripting, Script on Newtons
  • Basic, Forth, Message-C, SIAL (1990 ODBC equivalent with full-text search engine), Smalltalk and Prolog, S#
  • Microsoft Visual Basic, Javascript/JScript, .NET Runtime (desktop, mobile), XNA, Powershell
Startup sequence

Script binding:

shebang  #!/local/bin/afts
ms-pathext  ftype.exe & assoc.exe

1. current directory
2. ~/ user's home directory
3. ee's home directory
4. coff-built-in directory

Process Exec → main primary thread → Engine Init
TSL Core Registered

afconfig.tsl config located and run → Link-name executed as command, if available
afm Fall-through to fossil-default

Not-found Hook Invoked to load command
Command Executed as Program

11/17/2017
Language
Statements, Expressions and other Concepts

TCL
• Statement
  • Composed of Words
    • First Word is Command
    • Rest is Command Macro Params
• Expressions
• Things
  • Scalars
  • Key-Value Containers

TSL
• Statement
  • Composed of First Word and Rest
  • First Word is Message
    • Command, Function or Method
  • Rest is Message DSL Params
    • Literal, Macro, Eval
• Things
  • Scalars
  • Mixin Prototype Dynamic Objects
# Operators and Keywords

Syntax, Semantics and DSL patterns

<table>
<thead>
<tr>
<th>Operator</th>
<th>Language Type</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{…}</code></td>
<td>tcl</td>
<td>Literal text patterns.</td>
</tr>
<tr>
<td>&quot;…&quot; '…' <code>…</code></td>
<td>tcl+</td>
<td>Macro prefix-names. $(...) forms. (...) invocation. <strong>Quirks</strong> for &quot;[...]&quot;.</td>
</tr>
<tr>
<td><code>[…]</code></td>
<td>tcl↑</td>
<td><strong>Command</strong> patterns. proc, func, method patterns. <strong>Unary and binary symbol method</strong></td>
</tr>
<tr>
<td><code>(...)</code></td>
<td>tsl</td>
<td><strong>Expr (eval)</strong> implicit eval pattern.</td>
</tr>
<tr>
<td><code>$</code></td>
<td>tcl+</td>
<td><strong>MX path Operator</strong> macro-expand operator for variables and other tokens.</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>tsl</td>
<td>Ref path Operator reference operator for variables and path expressions.</td>
</tr>
<tr>
<td><code>\</code></td>
<td>tcl+</td>
<td><strong>Escape Operator</strong> enables UTF8, ascii and inline-operator escapes. <strong>TCL quirks mode for newline and curly-brace EOS.</strong></td>
</tr>
<tr>
<td><code>;</code></td>
<td>tcl↑</td>
<td><strong>End-of-statement</strong> Tcl quirks mode for newline and curly-brace EOS. Expr eval param delimiter.</td>
</tr>
<tr>
<td><code>,</code></td>
<td>tsl</td>
<td><strong>End-of-expression</strong></td>
</tr>
<tr>
<td><code># /* */ //</code></td>
<td>tcl+↑</td>
<td><strong>Comments</strong> JavaScript/C++ compatible comments. <strong>Quirks for TCL #.</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>super, this↑</td>
<td>inheritance operator and contextual variable for accessing inheritance</td>
</tr>
</tbody>
</table>

---

**TSL Talk by David Simmons, Smallscript Corp (c) 2017**

Page 66
Command **re-examined as** Message

| statement | word-tokens: literal, macro-forms | $ "..." […] 
{…} |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
<td><strong>Params (aka words)</strong></td>
<td></td>
</tr>
</tbody>
</table>

```tcl
putl $msg;
# A TCL command with macro-param

putl(msg)
# A TSL function with expr-param

stream.putl(msg);
# A TSL method with an expr-param

stream.putl() $msg;
# A TSL method with a macro-param
```
Variables re-examined

**Declaration**

```tcl
set varName valueParamWord;          # TCL assignment form
set varName;                          # TCL get command

var varRef = eval-expr;              # TSL assignment form
var varRef;                           # TSL decl command form
```

Variable Reference is a **path**

**Path roots are:**
Global, or in Stack Frame Context down-level of global-root-frame.
Variable Path Expressions

JSON Expressions and TCL quirks mode support

Examples:

\$\{putl "Hello"\}\n\$\{putl "Hello"\}^\n\$(17 + 5 \% 3)^\n
$::identifier-expr
$^\text{identifier-expr}
$identifier-expr
$receiver-path-expr::namespace-expr(\?invoke-expr-param?, ...)

Cases of Note:

&{ ... JSON pojo ... }
JSON pojo supporting TSL eval expr extensions and auto-conversion.

&[ ... JSON poja ... ]
JSON poja supporting TSL eval expr extensions.

\text{var} identifier-path = \&\& var-ref-path; \text{Binds two variables, as opposed to assignment.}
DEMO

var sharing

Script Example
Object Model

Logically, objects reference other objects using key-value associations which can be shared. An association can also be called a variable.

In practice, the internals are more highly optimized and only used associations as an object proxy indirection when sharing requires it.

TSL variables and commands play a key role in how perspective types are used for versioning and dynamic security.

ObjRef’s are tagged pointers. Certain scalar values, like IntPs objects, actually have their value encoded directly in the pointer.

Expando Type ➔ proto-super mixin behavior
Object Space

One per engine created within a process.

There can be an unlimited number of engines within a process. Each engine takes microseconds to create or release.

While crucial concepts, as we will see next, the DSL for intrinsic command declaration constructor forms mean that upvar and uplevel are rarely needed.

The variable predicate path-binder plays a key role in mixin and path resolution.

By default, proc declarations are placed here.

By default, func declarations are placed here.
Given a uniform internal object model, all objects can be converted to or from JSON via serialization.

The original technology name for this in AOS 1992 was PIPOs. **Platform Independent Portable Objects**, which contained TOCs, DBs, versioning, schema-migration features and supported cross-machine migration of threads and UX components with automatic re-wiring.

When TSL networking protocols for HTTP/HTTPS with JSON are combined with **SQL TRIGGERS** in **SQLite** using **JSON1**, powerful robust systems can be easily built.
Commands are objects

- Modules are commands with additional metadata.
- Source has *provenance*, which plays a role in perspective-type binding of *versions* and access *security*.
- Since *objects* are *pathable* and *commands* are global or contextual *objects*, commands are commonly used as *namespaces*.

**Scripts are lambda commands**

```javascript
// Discrete thisFrame context
proc myCmd1(param, ...) { }
// ^ Implicit uplevel thisFrame context
proc^ myCmd2(param, ...) { }
// lambda
var lambda = func (...) { }
// Declaring a subtype
func localCommand(...) &{
  // super-type [commands are objects - they have supers too],
  super: lambda,
  // inherited instance methods
  prototype: {
    foo: func () { },
  },
  // @factory initializer
  @body: {
    putl "Hello";
  }
  // Creating an instance
  var inst = new localCommand();
}
```
QUESTIONS

More demos offline and BOF