Tachyons

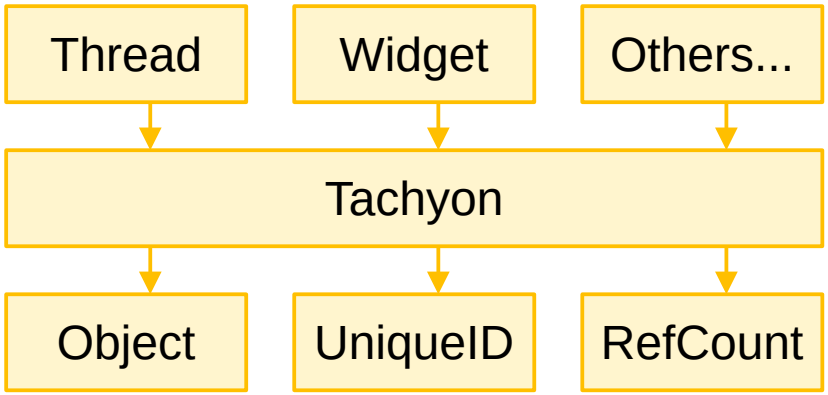
December 2025

Tachyons: Overview

- Heavy Objects
  - Ref/ID
- Scheduled on a Thread
  - setup() to initialize
  - tick() to do normal work
  - teardown() to shutdown
  - snap() exports to frame
- Serialization
- Send/Receive Posts
  - Sub/Pub (events)
  - Slots registered & called before setup/tick/teardown in cycle
- Tree – Parent/Child
- Interfaces/Proxies
- Resources/Delegates

Tachyon: Class Inheritance

- Base Heavy Object
  - All others derive from this
  - Has meta information
- Threading – safety matters!
  - Thread owns the pointer
  - Use frame to get data for another tachyon
  - Tachyons use posts for comms
- UniqueID is used as the TachyonID.



Tachyons – Various Types

- Camera – projection matrix
- Collision – (TODO)
- Desktop – GLFW/X11/Win32
- Gamepad
- Joystick
- Keyboard
- Light
- Manager
- Model – Sim (TODO)
- Mouse
- Physics – (TODO)
- Rendered – Vulkan/GPU
- Scene – Cameras, Lights, Models, Rendereds, etc
- Spatial – view & world matrix
- Thread – Runs things
- Viewer – Visual binder (window, widget & vulkan)
- Widget – Contents
- Window – Desktop's incarnation

Note: this is not a complete listing

Tachyons: Interface/Proxy

- Interfaces (I)
  - Common Shared API
  - Positions, sizes, colors, etc
- Proxies (P)
  - Buffer API
  - Communicate to Tachyons via commands
- Aspects (A)
  - OPTIONAL convenience to implement interfaces for tachyons.

Tachyon: Frame

- Frame
  - Ref-counted
  - Snapshot in time
  - Contains snaps & data
  - Different per thread!
  - Accessible via Frame::current() during setup()/tick()/teardown()
- Snap is Tachyon's state data/properties
  - Name, parent, children, etc
  - Captured whenever marked "dirty"
- Data is Tachyon's active data
  - Posts (In/Out)
  - Always captured

Tachyon Scheduling

- Cycled on Thread
  - inbox posts processed
  - setup/tick/teardown
    - return result governs proceed/abort
  - outbox posts sent
  - snap/data captured
- Frequency … variability is TODO, so currently every thread tick.

- Must be scheduled
  - Should use Tachyon::create<T>() to have this done automatically
- Can be moved between threads
  - Thread::owner(PUSH, thread) does this
  - Unless different thread is enabled, children will be pushed into same thread as parent

Tachyon Life Cycle

- Tachyon’s setup() called every Thread tick, until…
  - {} is returned, proceeds
  - Abort is returned, skips straight to teardown
  - All children are successful in their startup
  - Returning WAIT delays!
- Tachyon’s tick() called every thread tick until… abort/teardown
- Tachyon’s teardown() is called every thread tick until success (WAIT delays)
- Prepare for multiple invocations (so check)
- Call base class methods too!

Tachyon Assets

- Delegates
  - Flexible extensibility points
  - Declare in init_meta()
- Resources
  - Meshes, Images, Textures, etc
  - Declare in init_meta()

Tachyon: static void init_meta()

- Description/Abstract
  - Destructor needs to be public (unfortunate)
- Properties (Object)
- Aspects
  - Include the appropriate writer
  - AAspect::init_meta(w)
- Slots
  - w.slot(&T::m_post_slot)
- Interfaces (if not aspect)
  - w.interface<Iinterface>();
- Resources
  - w.resource(“name”, &T::m_resource);
  - Or via getter/setter
- Delegates
  - w.delegate(“name”, &T::m_delegate)

Tachyon Macros

- Inside class definition
  - If new meta… YQ_TACHYON_META(TMeta)
  - If new snap… YQ_TACHYON_SNAP(TSnap)
  - If new data… YQ_TACHYON_DATA(TData)
  - Always … YQ_TACHYON_DECLARE(T, Base)
    - Listed LAST (because it’ll detect what’s defined in the previous three)
- Source… global namespace
  - Always … YQ_TACHYON_IMPLEMENT(anynamespace::T)

Tachyon Posts

- PRIMARY means of communication between tachyons
  - Send command to order
  - Publish events when things happen
    - Use subscribe() to get these (same thread) or use a subscribe command
  - Requests to get something (or it’s okay to object to)
  - Replies to acknowledge/reject a request
  - Registered handler via slots (by either reference or ref pointer is fine)
  - Manifestos go out as messages