Force Simulation API Reference

Overview

There are three generations of force simulation APIs in this codebase:

1. **OLD (V1)**: `PSD3.Capabilities.Simulation` + `PSD3.Internal.Simulation.*`
2. **TRANSITIONAL (V2)**: `PSD3v2.Capabilities.Simulation`
3. **NEW (V3)**: `Component.CodeExplorerV2.SimulationManager` + `SceneConfigs`

API Comparison

V1: PSD3.Capabilities.Simulation (DEPRECATED)

Location: `src/lib/PSD3/Capabilities/Simulation.purs`

Type Classes:

- `SimulationM selection m` - Base class for static simulations
- `SimulationM2 selection m` - Extended class for dynamic updates

Key Functions:

```
-- SimulationM
init :: SimulationConfig selection a id linkRow key -> m { nodes, links }
start :: m Unit
stop :: m Unit

-- SimulationM2
update :: SimulationUpdate a id linkRow key -> m { nodes, links }
addTickFunction :: Label -> Step selection d -> m Unit
removeTickFunction :: Label -> m Unit
```

State Type: `D3SimulationState_ d` (mutable, contains force handles)

Force Definition (from `PSD3.Internal.Simulation.Types`):

```
data Force d = Force {
    "type"     :: ForceType
  , name       :: Label            -- String-based identification
  , status     :: ForceStatus      -- ForceActive | ForceDisabled
  , filter     :: Maybe (ForceFilter d)
  , attributes :: Array (ChainableF d)
  , force_     :: D3ForceHandle_   -- Mutable D3 handle!
}
```

Problems:

- Force handles are mutable and persist across scene transitions
- String-based force identification
- Parameter values reset unexpectedly
- Complex state management with lenses

V2: PSD3v2.Capabilities.Simulation (TRANSITIONAL)

Location: `src/lib/PSD3v2/Capabilities/Simulation.purs`

Type Classes: Same as V1 but uses PSD3v2's Attribute system

Key Functions:

```
-- SimulationM
init :: SimulationConfig a id linkRow key sel -> m { nodes, links }
addTickFunction :: Label -> Step sel d -> m Unit
removeTickFunction :: Label -> m Unit
start :: m Unit
stop :: m Unit

-- SimulationM2
update :: SimulationUpdate a id linkRow key -> m { nodes, links }
reheat :: Number -> m Unit
setForces :: Array (Force (SimulationNode a)) -> m Unit
```

Step Type (PSD3v2 version):

```
data Step sel datum = Step (sel datum) (Array (Attribute datum))
```

Improvements over V1:

- Uses PSD3v2's type-safe Attribute system
- Added `setForces` for scene transitions
- Better documentation

Still has: Mutable force handles, string-based names

V3: SimulationManager + SceneConfigs (RECOMMENDED)

Location: `src/website/Component/CodeExplorerV2/SimulationManager.purs`

Architecture: Bypasses D3's simulation wrapper entirely

Key Types:

```
-- Opaque handle to D3 force
foreign import data ForceHandle :: Type

-- Simulation state (pure PureScript)
type SimState = {
    nodes :: Array SpagoSimNode
  , links :: Array { source :: Int, target :: Int, linktype :: String }
  , forces :: Map String ForceHandle
  , alpha :: Number
  , alphaMin :: Number
  , alphaDecay :: Number
  , alphaTarget :: Number
  , velocityDecay :: Number
  , running :: Boolean
  , tickCallback :: Effect Unit
}
```

Key Functions:

```
-- Lifecycle
createSimulation :: Effect (Ref SimState)
setNodes :: Array SpagoSimNode -> Ref SimState -> Effect Unit
start :: Ref SimState -> Effect Unit
stop :: Ref SimState -> Effect Unit
reheat :: Ref SimState -> Effect Unit
setTickCallback :: Effect Unit -> Ref SimState -> Effect Unit

-- Force Creation (type-safe, no strings!)
createCollision :: { padding :: Number, strength :: Number, iterations :: Number } -> ForceHandle
createCharge :: { strength :: Number, theta :: Number, distanceMin :: Number, distanceMax :: Number } -> ForceHandle
createChargeFiltered :: {...} -> (SpagoSimNode -> Boolean) -> ForceHandle
createCenter :: { x :: Number, y :: Number, strength :: Number } -> ForceHandle
createLink :: { distance :: Number, strength :: Number, iterations :: Number } -> ForceHandle
createClusterX :: Number -> (SpagoSimNode -> Boolean) -> ForceHandle
createClusterY :: Number -> (SpagoSimNode -> Boolean) -> ForceHandle
createRadial :: { radius :: Number, x :: Number, y :: Number, strength :: Number } -> (SpagoSimNode -> Boolean) -> ForceHandle

-- Force Management
addForce :: String -> ForceHandle -> Ref SimState -> Effect Unit
clearForces :: Ref SimState -> Effect Unit
initializeForce :: ForceHandle -> Array SpagoSimNode -> Effect ForceHandle
initializeLinkForce :: ForceHandle -> Array SpagoSimNode -> Array links -> Effect ForceHandle
```

Scene Configuration (from `SceneConfigs.purs`):

```
type SceneConfig = {
    name :: String
  , forces :: Array { name :: String, create :: Effect ForceHandle }
  , alpha :: Number
  , velocityDecay :: Number
}

-- Apply a scene (clears old forces, adds new ones)
applyScene :: SceneConfig -> Ref SimState -> Effect Unit
```

Advantages:

- Fresh force handles created on every scene transition
- No parameter persistence issues
- Type-safe force creation (record parameters, not strings)
- Clear separation: immutable config vs mutable runtime
- Our own animation loop (full control)
- Forces stored in window registry for control panel access

Migration Path

From V1 to V3:

1. Replace `SimulationM`/`SimulationM2` with direct `SimulationManager` calls
2. Replace `Force d` definitions with `SceneConfig` records
3. Replace string-based force names with typed force creation functions
4. Replace tick functions with `setTickCallback`

Example Migration:

V1 Style:

```
-- Force library with mutable handles
forceLibrary :: Array (Force SpagoSimNode)
forceLibrary = [
  chargeForce "charge" allNodes (strength -100.0)
, collideForce "collision" allNodes (radius nodeRadius)
]

-- In draw function
init { forces: forceLibrary, activeForces: Set.fromFoldable ["charge",
"collision"], ... }
```

V3 Style:

```
-- Scene config (pure data)
orbitScene :: SceneConfig
orbitScene = {
    name: "Orbit"
  , forces: [
      { name: "collision", create: pure $ createCollision { padding: 2.0,
strength: 1.0, iterations: 1.0 } }
    , { name: "clusterX", create: pure $ createClusterX 0.2 isModule }
    ]
  , alpha: 0.3
  , velocityDecay: 0.4
}

-- In draw function
simRef <- createSimulation
applyScene orbitScene simRef
start simRef
```

Files to Delete (after full migration)

Old Internal Simulation:

- `src/lib/PSD3/Internal/Simulation/Types.purs` - D3SimulationState_, Force, ForceStatus, etc.
- `src/lib/PSD3/Internal/Simulation/Forces.purs` - createForce, createLinkForce
- `src/lib/PSD3/Internal/Simulation/Functions.purs` - FFI wrappers
- `src/lib/PSD3/Internal/Simulation/Config.purs` - Force config helpers

Old Capabilities:

- `src/lib/PSD3/Capabilities/Simulation.purs` - SimulationM, SimulationM2 (V1)

Old Component State:

- `src/lib/PSD3/Component/SimulationState.purs` - Old state helpers

Deprecated Config (if not used):

- `src/lib/PSD3/Config/` - May be superseded by SceneConfigs pattern