BasicData Mojo

MojoPortal host strategy and BasicData function/datashell doctrine for migration, logging, and side-by-side modern Angular turnover.

1. Purpose

This document extends the BasicData BookShelf foundation by adding the mojoPortal-side strategy for how BasicData is actually used in routed web content. It is intended to serve as a standing foundation document for later conversations about migration, logging, host-contract recovery, and side-by-side modernization.

2. Verified foundation

2.1 BasicData is not a plain object

BasicData should be treated as a callable function-object datashell. It carries mutable attached data while also supporting drained command-lane behavior. This means it is both a data vessel and part of the runtime control contract.

2.2 Parental and Parentop are part of the contract

BasicData does not stand alone. Parental acts as the nearest parent exchange anchor and Parentop acts as the top available anchor across frames where accessible. Together they form the function/control shell around the data shell.

2.3 Communication is shared mutable state plus drained command lanes

The runtime pattern is not a simple API fetch model. It is shared mutable state, plus command-lane reads and polling behavior, consumed between page, iframe, parent, and top-frame contexts.

2.4 mojoPortal is the authoritative host

Within mojoPortal-routed content, the host page remains authoritative. The client shell does not own the business contract. It consumes the seeded host contract.

3. MojoPortal strategy for BasicData

3.1 mojoPortal seeds the route world

In routed slices such as ABdash and related application shells, mojoPortal acts as the route host, frame authority, and state seeder. The host page or surrounding frame environment seeds BasicData, exposes parental exchange, and determines how route entry is shaped.

3.2 Client shells are consumers, not sovereign roots

Legacy Angular2js and similar client surfaces should be understood as consumers of seeded state. They bind to BasicData, react to callbacks, and mutate attached fields, but they are not the root of truth.

3.3 Missing static shells can be regenerated by routed host behavior

Where a static file such as index.html is removed or missing, the host route and first route entry may reconstruct the frame and wiring. This means route authority sits above static shell files.

3.4 Route authority outranks client packaging

A route should be understood first from its host entry, seeded data, and callback contract, then from the client-side framework files. The route contract is more authoritative than the packaging style of Angular2js or later Angular builds.

4. ABdash as the canonical example

4.1 ABdash is a routed multi-view application shell

ABdash is not one page. It is a routed shell with multiple states and views, with seeded data supplied from the host and consumed inside the route controllers and templates.

4.2 The real parameter model is host-injected state

The Angular route table uses simple paths, but the effective parameters are mostly delivered through parent BasicData and related host structures rather than URL path parameters.

4.3 BasicData is both field source and behavioral bridge

Controllers and templates read attached data such as User, Tracker, Workplace, Residence, Vehicle, MenuA, MenuB, and Toggler, while also depending on host methods and callbacks for success, cancel, destination changes, and messaging.

5. Verified route families and likely data anchors

5.1 Workplace and Residence

These routes align strongly with place-style address contracts and likely map onto the Place table shape, including address fields such as Address1, Address2, City, Region, and Zip.

5.2 Factory and Assign

These routes align strongly with tracker and device setup flows and likely map onto UserDevice, using fields such as TrackName, DeviceMode, HardwareVersion, IMEI, and SIMCard.

5.3 Install

This route appears to bridge tracker and vehicle semantics, likely combining UserDevice with a vehicle-side wrapper or legacy object.

5.4 Customer and Vehicle

These routes are high semantic-risk areas because they show heavy reuse or overloading of stock and legacy fields.

5.5 Tracker

This is a convergence route. It appears to join user identity, tracker/device identity, vehicle semantics, and residence/workplace address context.

6. Sample usage patterns

6.1 Host-seeded data sample

A typical slice depends on parent BasicData objects such as User, Vehicle, Tracker, Residence, Workplace, MenuA, MenuB, and Toggler. These objects are presented to the client shell already shaped by the host page and surrounding scripts.

6.2 Callback sample

A typical form route uses host callbacks such as OK(ftype, actvt), Cancel(ftype), DashDest(), and Messaging(...). These callbacks should be treated as part of the runtime contract, not as incidental helpers.

6.3 Semantic overlay sample

Some routes reveal that stock or legacy fields may be semantically overloaded. For example, a field named Continent may carry tracker identity, Addtl may carry a VIN-like value, and PasswordAnswer may act as an alternate phone field in specific workflows.

6.4 Datashell plus function shell sample

BasicData carries attached mutable data, while Parental and Parentop provide control exchange and upward communication. Together they create a datashell plus function shell model.

7. Migration doctrine

7.1 Log first

Before replacing routes or modernizing client shells, capture structured logs describing route entry, seeded data, command lanes, callbacks, read fields, written fields, likely table anchors, and semantic warnings.

7.2 Preserve awkward semantics first

The first migration pass should mirror production behavior, even when field names or semantic mappings are inept. Normalization can happen later.

7.3 Build modern Angular side-by-side

A new Angular shell should be introduced beside the legacy Angular2js shell, consuming the same host-seeded contract through a compatibility adapter.

7.4 Migrate route-by-route

Routes should be turned over gradually, beginning with cleaner place and user-device surfaces before moving to convergence routes such as tracker.

8. Compatibility adapter doctrine

8.1 The adapter is the first modern asset

The first true modernization deliverable should be an adapter that wraps BasicData, Parental, Parentop, command lanes, and callback behavior into a stable bridge for modern Angular.

8.2 Mirror first, normalize later

The adapter should preserve the real legacy contract as-is at first, including awkward names and callback shapes, so route parity can be proven.

8.3 Logging belongs in the adapter

The adapter is the right place to emit .zson audit records for route behavior, field reads and writes, callback flow, and turnover status.

9. .zson audit doctrine

9.1 Purpose

.zson should be used as the audit payload format for route analysis, route parity, screenshot pairing, and turnover tracking.

9.2 Shape

Each route record should capture route name, state, entry files, screenshot references, BasicData keys, command lanes, parental usage, reads, writes, callbacks, likely tables, field overlays, server-shaping notes, and migration status.

9.3 Repetition rule

The canonical JSON payload should be repeated sequentially in the .zson according to the audit convention.

10. Screenshot and codebehind triangulation

10.1 Why screenshots matter

Screenshots help reveal whether a field is display-only, editable, grouped, or repurposed, which is often not obvious from code alone.

10.2 Why codebehind matters

The aspx and codebehind layers often reveal how datasets are shaped and how fields are projected into forms, which is essential for understanding the true route contract.

10.3 Three-part evidence packet

The ideal evidence packet for a route is: visual screenshot, structured .zson log, and server-side shaping source.

11. Recommended route order for side-by-side turnover

11.1 First routes

Workplace and Residence are the best early candidates because they align cleanly with Place-style address semantics.

11.2 Second routes

Factory and Assign are strong next candidates because they align clearly with UserDevice and device/tracker setup semantics.

11.3 Third routes

Install is a bridge route and should follow once the cleaner routes are understood.

11.4 Later routes

Customer and Vehicle should wait until semantic overlays are better documented.

11.5 Convergence routes last

Tracker and other convergence or orchestration routes should come later because they join multiple data vessels and semantic overlays at once.

12. Warnings and problem areas

12.1 Route contracts are larger than route paths

The real route contract includes seeded host state, callbacks, parent exchange, and regenerated shell behavior.

12.2 Static file assumptions may be false

A missing index.html does not necessarily mean the route is broken. The host may regenerate the shell and route wiring dynamically.

12.3 Field names may be deceptive

Stock and legacy fields may carry non-obvious production meaning.

12.4 Client code alone is insufficient

Client controllers and templates are only one layer of truth. Host scripts, codebehind, datasets, and screenshots all matter.

13. Next steps

13.1 Produce route-level .zson logs

Start with workplace, residence, factory, and assign.

13.2 Pair screenshots with route logs

Use screenshot-plus-log packets to confirm visible semantics.

13.3 Pair route logs with codebehind shaping

Recover the aspx and codebehind shaping logic for the same routes.

13.4 Build the BasicData compatibility adapter

Use the host contract as the first engineering interface for modern Angular.

13.5 Turn over one route at a time

Only move a route into modern Angular after input, output, callback, and visible behavior parity are proven.