

Rigorous specification

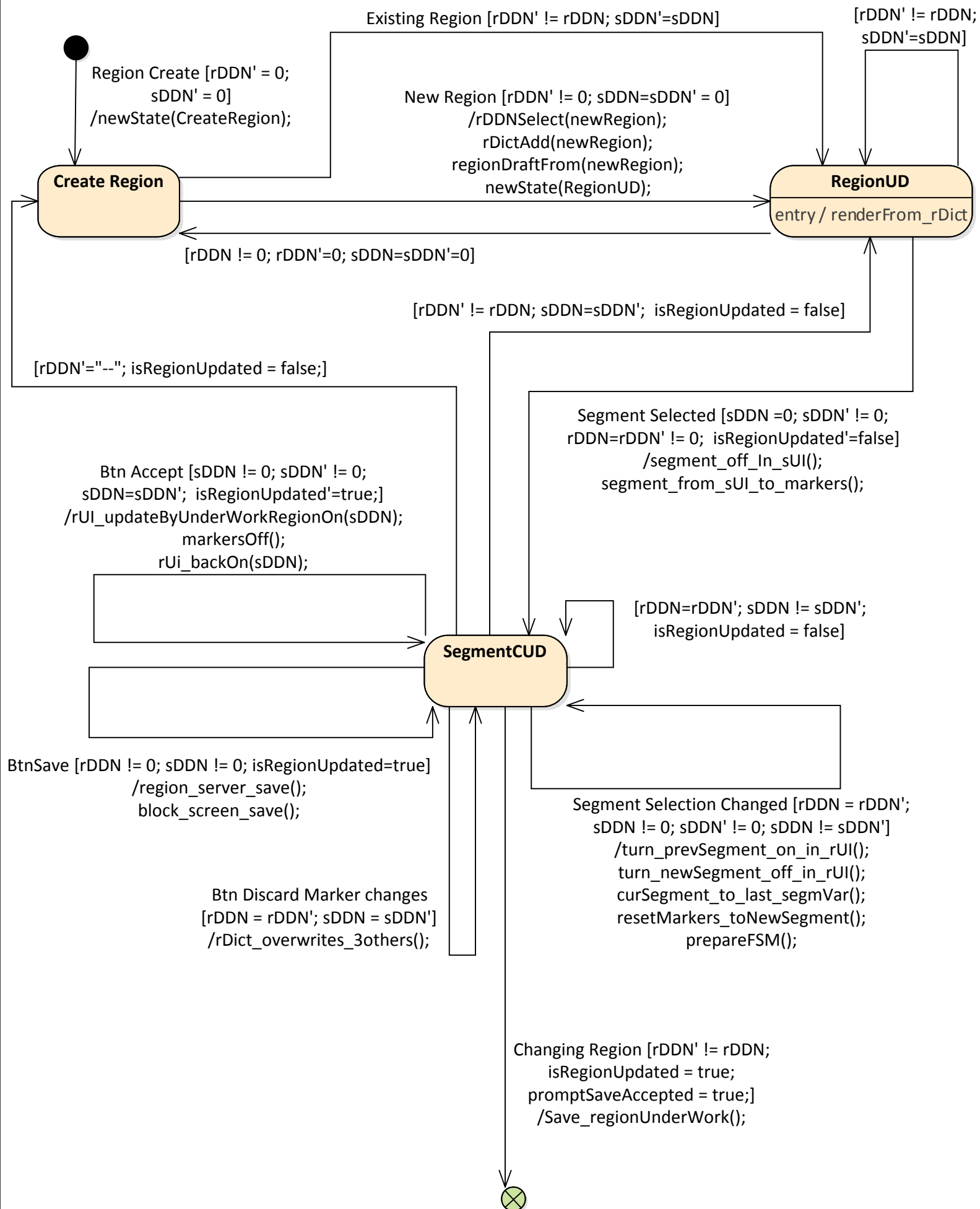
At the time being it is usually accepted that software will evolve fast. This brings doubt to the necessity of starting from a rigorous specification. Development will soon deviate from the specification to the degree that could make it even useless. There are different reasons why this may happen. For this reason, many people support tests and the various methodologies like "Test Driven Development", "Agile" and so on, as a better alternative to starting from a formal specification (here "formal" means mathematically specified).

I am convinced that when starting a project from scratch, efforts invested in developing a rigorous specification at initial steps, if done correctly, will lead to both a better software, and better development experience.

It does not mean that the development is going to stick to that initial specification, as it may very well deviate from. The thing is that aiming to rigorously specify the very early steps, may work for you as a tuner of the pitch of your project. It very well decreases the number of iteration cycles down the the road, as development goes on.

The code here is not yet complete and the updates are yet to be uploaded to Github. The following specification is in raw state and under ongoing change, but it is what the software development started from. Still a long road to have the software and specification complete.

Nevertheless everything is uploaded just in case, with the hope to possibly help someone who may need it now, in its current state.



— **section** SegmentCUD **parents** standard_toolkit
└

!!IMPORTANT:

before accepting any point in the "mouse input file", have the region rect reference,
check that the point clicked
always inside region

/* state: SegmentCUD */

/*
CUD:
Control, Update, Delete
*/

/* regionCDS is "only" toggled to specify the region corners update-state */

— **theorem** NoRegionUpdate
└ $\vdash ? \text{state} = \text{state}' = \text{SegmentCUD} \Rightarrow \text{regionCDS} = \text{regionCDS}' = \text{F}$
└

—
└ ON == T
└

—
└ OFF == F
└

main DS:

- rDict
- sUI
- rUI
- RDraft: region draft, to which any region rect update or segment update is applied, saving will send it to server
- segmentCDS: control variable, holds information that in SegmentCUD which segment gets update
- regionCDS: control variable, holds information that in RegionUD, if region rect got updated
- previousSegment: when sDDN selection changes, holds the previous value, by resaving it every time selection changes, right before using the old previousSegment, making $\text{previousSegment}' = \text{sDDN}'$

functions:

InitSegmentCDS(v?)
OverwriteSUI(Region, segment_name, On/OFF)

```

OverwriteSMarker(Region, segment_name, On/OFF)
OverwriteSDraft(p1?, p2?, segment_name)
let p1 = getSOrigin(rDict(region_name), segment_name)
let p2 = getSTerminus(rDict(region_name), segment_name)

```

```

/* (1) */
/*

```

Init

Descr.:

1. Initialize SegmentCUD,
2. Region is already selected and not "--",
3. New segment selected and not "--",
4. previous Segment = --
5. Render the selected segment OFF, so to let user to manipulate markers, updating the segment in draft, so:
6. turn on markers for the selected segment of selected region from original copy in rDict
7. reset the segment control dictionary all False

note.: the region-draft is already loaded with selected Region
(in RegionUD, only state which may navigate to SegmentCUD)

```

*/
┌ SegmentCUD_Init
  ΔRegionEnv
  v?: VOID
  |
  previousSegment = --
  sDDN = --

  sDDN' ≠ --
  previousSegment' = sDDN'

  rDDN = rDDN'
  rDDN' ≠ --

  segmentCDS' = InitSegmentCDS(v?)

  let region_name = rDDN'(rDDNVal')
  let segment_name = sDDN'(sDDNVal')

  ∃OverwriteSUI(NULL, segment_name, OFF)
  ∃OverwriteSMarker(rDict(region_name), segment_name, ON)
└

```

```
/* (2) */  
/*
```

Markers Updated

Descr.:

Overall:

When markers are updated, two things happen: a) draft's segment is overwritten, b) segment CtrlDS updated

1. selected segment and region has no change
2. draft's segment is updated with the name of the current selected segment, using input points of markers
3. segments' control DS gets updated based on the updated segment

Note:

- When mouse data available, two parties get it the same time: both the rFSM to setup game objects, and the OverwriteSDraft method to overwrite the draft. That's why the input of the method is points, and not region.
- While user updates, markers are managed by rFSM

```
*/
```

```
┌ SegmentCUD_MarkersUpdated
```

```
  ΔRegionEnv
```

```
  p1?, p2?: POINT
```

```
|
```

```
  rDDN = rDDN'
```

```
  sDDN = sDDN'
```

```
  let segment_name = sDDN(sDDNVal)
```

```
  ∃OverwriteSDraft(p1?, p2?, segment_name)
```

```
  segmentCDS' = segmentCDS ⊕ {segment_name ↦ T}
```

```
└
```

```
/* (3) */  
/*
```

Btn Discard

Descr.:

Purpose:

To undo marker changes in markers and turned-off sUI, using original Region copy of rDict

Overall:

When Discard,

1. have segment's name to undo data for, and origin and terminus points of its original copy from rDict
2. using points and segment's name overwrite the draft
3. overwrite markers

```
*/  
└ SegmentCUD_OnBtn_Discard  
  ΔRegionEnv  
  |  
  let segment_name = sDDN'(sDDNVal')  
  segmentCDS' = segmentCDS ⊕ {segment_name ↦ F}  
  
  let region_name = rDDN(rDDNVal)  
  let p1 = getSOrigin(rDict(region_name), segment_name)  
  let p2 = getSTerminus(rDict(region_name), segment_name)  
  ∃OverwriteSDraft(p1, p2, segment_name)  
  ∃OverwriteSMarker(rDict(region_name), segment_name, ON)  
└  
  
/* (4) */  
/*
```

Btn Commit

Descr.:

Purpose:

User being happy with what the segment must be, aims to submit changes for save

Overall:

Markers hold the state of manipulation, draft holds the same data, the sUI is overwritten by the draft, then
sUI turns on and markers turn off. Finally, sDDN'==-- so that we leave the segment.

Note:

When init back to SegmentCUD, if any segment is selected that its name points to True in the segmentCDS,

We turn on markers based on draft, not the rDaict

- 1.
- 2.
- 3.

```

*/
┌ SegmentCUD_OnBtn_Commit
  ΔRegionEnv
  |
  let segment_name = sDDN(sDDNVal)
  segmentCDS' = segmentCDS ⊕ {segment_name ↦ T}

  ∃OverwriteSUI(RDraft, segment_name, ON)
  ∃OverwriteSMarker(NULL, NULL, OFF)

  sDDN' = --
  state = SegmentCUD
  state' = RegionUD
└

```

```

/*
Leave save btn for RegionRD state
it is disabled in SegmentCUD
SegmentCUD state has only two btns:
Discard: undo changes on markers, overwrite sUI and draft back from rDict
Commit: accepts changes of draft (will not overwrite it), overwrites sUI of related
segment using draft
*/

```

```

/* (5) */
/*

```

Btn Save

Descr.:

If there exist any segment updated, screen is blocked and data sent to server, else, no reaction

Note:

RDraft is sent to server to save, result will be back as a newly saved region, which will replace rDict and get selected rDDN and make sDDN'==-- and reset sUI, rDict and its newly saved copy also updates draft, which (this draft-updating) happens in the event handler that responds to the savedEvt from server.

```

*/
┌ PFSegmentCUD_OnBtn_Save
  ∃ RegionEnv
  |
  ∀ s : segmentCDS' • second s = F
  state' = state = SegmentCUD

```

┌

┌ PTSegmentCUD_OnBtn_Save

\exists RegionEnv

|

$\exists s : \text{segmentCDS}' \bullet \text{second } s = T$
 $\text{state}' = \text{state} = \text{SegmentCUD}$

┌

—

 SegmentCUD_OnBtn_Save ==

 (P_TSegmentCUD_OnBtn_Save \wedge SaveServer_BlockScreen) \vee

 P_FSegmentCUD_OnBtn_Save

┌

/ rDDN change in SegmentCUD state */*

*/**

we block scene to save draft-region and on resp of server the updated region is selected and reRendered

so because user Committed prompt-save, we are considered of nothing but make save request and block

**/*

— **theorem** promptSaveAccepted_axiom

$\vdash ? \text{promptSaveAccepted?} : \text{BOOL}$

┌

┌ PTSegmentCUD_DrDDN

$\Delta \text{RegionEnv}$

|

$r\text{DDN} \neq r\text{DDN}'$

$s\text{DDN} = s\text{DDN}' \neq --$

$\exists s : \text{segmentCDS}' \bullet \text{second } s = T$

$\text{state} = \text{SegmentCUD}$

$(\text{state}' = \text{RegionUD} \wedge r\text{DDN}' \neq --) \vee (\text{state}' = \text{CreateRegion} \wedge r\text{DDN}' = --)$

┌

┌ P_FSegmentCUD_DrDDN

$\Delta \text{RegionEnv}$

|

$r\text{DDN} \neq r\text{DDN}'$

$s\text{DDN} = s\text{DDN}' \neq --$

$\exists s : \text{segmentCDS}' \bullet \text{second } s = F$

```
state = SegmentCUD
(state' = RegionUD  $\wedge$  rDDN'  $\neq$  --)  $\vee$  (state' = CreateRegion  $\wedge$  rDDN' = --)
└
—
SegmentCUD_DrDDN ==
  (PTSegmentCUD_DrDDN  $\wedge$  promptSaveAccepted?  $\wedge$  SaveServer_BlockScreen)  $\vee$ 
  PFSegmentCUD_DrDDN
└
└ SaveServer_BlockScreen
  ΔRegionEnv
  |
  ∃ReqServerSaveDraftEvt
  ∃ScreenBlockedWaiting
└
```

/* sDDN change in SegmentCUD state */

/* if segment is updated, it means user committed changes from draft (markers' change are there) to sUI

so when changing sDDN, we have either updated sUI or original one (equal to rDict related region)

therefore we are not concerned on this, and simply turn sUI of leaving state on, save new state as prev.

and turn markers off

*/

D: Delta

If user did not commit changes and tried changing segment, we consider existing changes to be discarded

If the sDDN' = -- then we turn off SMarker, because in that state marker manipulation is not allowed

while rDDN has no change, sDDN changed to either -- or something other than that

if the previous segment was not updated, we overwrite the sUI for it, using the unmanipulated rDict copy and

turning it on, else, with the knowledge that the previous segment was updated, we overwrite sUI with latest

update, which is kept in RDraft, and then turn sUI on

if the new segment that is obtained from sDDN' value (the current value of sDDN) has not been updated then

turn the markers on for it, using rDict original copy of the region and segment, where region name comes from

rDDN=rDDN', if the name of newly selected segment not found in segmentCDS, it could be only in a single case

when the value is "--", and in that case marker manipulation is not allowed, so turn markers off (and disallow saving, by evaluating "segment's name \neq "--"). Otherwise (which means that there is a segment_name with value of updated as true), turn on markers for the newly selected segment, using RDraft.

┌ SegmentCUD_DsDDN

 ΔRegionEnv

|

 rDDN = rDDN'

 sDDN \neq sDDN' \neq -- \vee sDDN \neq sDDN' = --

let region_name = rDDN(rDDNVal)

if \exists s: segmentCDS • first s = previousSegment \wedge second s = F **then**
 \exists OverwriteSUI(rDict(region_name), previousSegment, ON)

else \exists OverwriteSUI(RDraft, previousSegment, ON)

 previousSegment' = sDDN'(sDDNVal')

let segment_name = sDDN'(sDDNVal')

if \exists s: segmentCDS • first s = segment_name \wedge second s = F **then**
 \exists OverwriteSMarker(rDict(region_name), segment_name, ON)

else if \forall s: segmentCDS • first s \neq segment_name \exists OverwriteSMarker(NULL, segment_name, OFF)

else \exists OverwriteSMarker(RDraft, segment_name, ON)

└

— **section** AppData FSM - Home-DDN-Img **parents** standard_toolkit

└

/ Announce for everyone: AppData is renamed to AppDS, being the main Data Structure */*

/ seqURL, url and texture: related to interaction of HomeDDN and HomeImg*/*

/ Given sets */*

— [URL, TEXTURE, EVENTS] └

/ REQ target is the functionality that AppServiceProvider (abbr. AppProvider) offers */*

└

└ REQ ::= ReqLastUrls | ReqTexture

└

/ RES is the response of the server that comes back to App, via server >JS >proxy >provider */*

└

└ RES ::= LastUrlsCB | TextureCB

└

/ Events to App */*

— **theorem** evtIn

└ $\vdash? \{HInitEvt, SetTextureEvt\} \subseteq \text{EVENTS}$

└

/ Events from App */*

— **theorem** evtOut

└ $\vdash? \{UrlsUpdateEvt\} \subseteq \text{EVENTS}$

└

/ State-schema and Init operation */*

└ AppData

└ urlsList: seq URL

└ lastReqUrl: URL

└ texture: TEXTURE

|

└ $\text{lastReqUrl} \neq \emptyset \Leftrightarrow (\text{urlsList} \neq \emptyset \wedge \text{lastReqUrl} \in \text{ran urlsList})$

└

└ AppDataInit

└ AppData

└ req!: REQ

|

└ $\text{urlsList} = \emptyset$

└ $\text{lastReqUrl} = \emptyset$

```
texture = ∅  
req! = ReqLastUrls  
└
```

/ Operations */*

```
└ AppOnHomeInit  
  ∃AppData  
  e!,e?: EVENTS  
  |  
  e? = HInitEvt  
  e! = UrlsUpdateEvt  
  urlsList! = urlsList = ∅  
  lastReqUrl! = lastReqUrl = ∅  
  texture! = texture = ∅  
└
```

```
└ AppOnLastUrlsCB  
  ΔAppData  
  urls?: seq URL  
  rs?: RES  
  e!: EVENTS  
  rq!: REQ  
  rq_url!: URL  
  |  
  rs? = LastUrlsCB
```

```
  urlsList = ∅  
  lastReqUrl = ∅  
  texture = ∅
```

```
  urls? ≠ ∅  
  urlsList' = urls?  
  lastReqUrl' = head urls?  
  texture' = texture
```

```
  rq! = ReqTexture  
  rq_url! = lastReqUrl'  
  e! = UrlsUpdateEvt  
└
```

```
└ AppOnTextureCB  
  ΔAppData  
  rs?: RES  
  texture?: TEXTURE  
  url?: URL  
  e!: EVENTS
```

```

|
rs? = TextureCB
texture? ≠ ∅
url? = lastReqUrl
texture' = texture?
e! = UrlsUpdateEvt
└

```

/ Image upload will trigger the same functionality, after uploaded image url returns to App */*

```

└ AppOnSetTextureEvt
  ΔAppData
  e?: EVENTS
  url?: URL
  rq!: REQ
  url!: URL
|
e? = SetTextureEvt
url? ≠ lastReqUrl
lastReqUrl' = url?
rq! = ReqTexture
url! = url?
└

```

```
— section appregion parents standard_toolkit
└
```

```
/* Region scene */
```

```
/* Holds the Regions related data structures */
— [ URL, TEXTURE, RNAME ] └
```

```
/* Boolean definition and meaning */
```

```
— theorem d_HasIntegerType
```

```
└? d :  $\mathbb{Z}$ 
```

```
└
```

```
—
└ F == (d ∈  $\mathbb{Z}$  ∧ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—
└ T == (d ∈  $\mathbb{Z}$  ∨ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—
└ BOOL ::= T | F
```

```
└
```

```
—
└ RSTATE ::= INTR | RC | RR | RU | RD
```

```
└
```

```
/* So that if rState is in RR, mState can leave initM and be in any of other states, until
that sits in initM */
```

```
—
└ MSTATE ::= INITM | RMC | RMR | RMU | RMD
```

```
└
```

```
—
└ REQ ::= ReqRegionsOfUrl
```

```
└
```

```
—
└ RES ::= RegionsOfUrlCB
```

```
└
```

```
—
└ INTENT ::= RadioRCreateN | DdnRReadN | RadioRUpdateN | BtnRDeleteN
```

```
└
```

```
—
└ UICMD ::= RadioRCreate | RadioRRead | RadioRUpdate | BtnRDelete
```

```
└
```

```
—
└ REPORT ::= WaitingServerResponse
```

└

└ RegEnv

 rState: RSTATE

 mState: MSTATE

 rDDN: seq RNAME

 rDict: RNAME \leftrightarrow Region

 applmg: TEXTURE

 appUrl: URL

 uMagnitude: \mathbb{N}

 SceneLocked: BOOL

 interactable: BTN \rightarrow BOOL

 promptSave: BOOL

|

 ran rDDN = dom rDict

 uMagnitude > 0

 applmg $\neq \emptyset$

 appUrl $\neq \emptyset$

└

└ RegEnvInit

 RegEnv'

 rq!: REQ

 applmg?: TEXTURE

 appUrl?: URL

 uMagnitude?: \mathbb{N}

 rp!: REPORT

|

 rq! = ReqRegionsOfUrl

 rp! = WaitingServerResponse

 SceneLocked' = T

 promptSave' = F

 rState' = INITR

 mState' = INITM

 rDDN' = \emptyset

 rDict' = \emptyset

 applmg' = applmg?

 appUrl' = appUrl?

 uMagnitude' = uMagnitude?

 interactable = {(RadioRCreate, T), (DdnRRead, T | RadioRUpdate | BtnRDelete)}

└

/* when user in Home at chose img, RegionInitEvt to AppRegion, to JS query url-regions */
/* so when user goes to RegionScene, regions related to chosen url must already be in

place */

/* response of server: empty rDDN and rDICT, or, non-empty, each an op-schema */

/* AppRegion only sends regions set of imgUrl, when server response comes back. No init data to be ever sent! */

/*

1

lock region landing page until server response is in
after that, wait for opState change between create or read (select) region
read (select) region can lead to => update-region, delete-region or m-crud
*/

```
┌ RegEnvUnLocked
  ΔRegEnv
  rs?: RES
  rDDN?: seq RNAME
  rDict?: RNAME ⇔ Region
|
  rs? = RegionsOfUrlCB
  rDDN' = rDDN?
  rDict' = rDict?
  SceneLocked' = F
└
```

```
┌ NAME
  DECLS
|
  PREDS
└
```



```
— section appregion2 parents standard_toolkit
└
```

```
/* Region scene */
```

```
/* Holds the Regions related data structures */
```

```
— [ URL, TEXTURE, RNAME, GAMEOBJECT, VOID ] └
```

```
/* Boolean definition and meaning */
```

```
— theorem d_HasIntegerType
```

```
└? d :  $\mathbb{Z}$ 
```

```
└
```

```
—  
└ F == (d ∈  $\mathbb{Z}$  ∧ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—  
└ T == (d ∈  $\mathbb{Z}$  ∨ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—  
└ BOOL ::= T | F
```

```
└
```

```
/* Region-Delete happens by BtnRegionDelete, Region-Update: l-r-click */
```

```
/* there will be a tmpGameObjArr for temporarily saving obj, will be reset on every  
app-state-change */
```

```
/* Region-Update: when starts, resets tmpGameObjArr redefine to how many needed,  
reset mFSM to init */
```

```
—  
└ STATE ::= CreateRegion | RegionUD | SegmentCUD
```

```
└
```

```
/* this is memory-state, to organize mFSM, or memory-FSM */
```

```
—  
└ MSTATE ::= MInit | Origin | Terminus
```

```
└
```

```
—  
└ MEASUREMENT ::= A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2 | --
```

```
└
```

```
—  
└ MSET ::= A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2
```

```
└
```

```
—  
└ PROMPT ::= AwaitServerUpdate | InvalidUpdate_LetUserRegRepeat |  
PleaseProvideRegionName  
| PleaseProvide_UniqueRegionName
```

app_region_2_.zed8

```
┌
├
├ validRegionName == RNAME → BOOL
├
├
├ evaluateState == ( $\mathbb{N} \times \text{MEASUREMENT}$ )  $\leftrightarrow$  VOID
├
├
├ resetTmpGObjArr == seq GAMEOBJECT  $\rightarrow$  seq GAMEOBJECT
├

├ Angles
  E1, E2:  $\mathbb{N}$ 
  |
  | E1  $\geq$  0
  | E2  $\geq$  0
├

/* the image is loaded in the 3rd section of XY coordinate system */
├ POINT
  x,y:  $\mathbb{Z}$ 
  |
  | x < 0
  | y < 0
├

├ SEGMENT
  p1,p2: POINT
  |
  | p1  $\neq$  p2
├

├ Region
  diagonal: SEGMENT
  angles: Angles
  mSet: MSET  $\rightarrow$  SEGMENT
├

/* Region Environment State Schema */
├ RegLabEnv
  mState: MSTATE
  rDDN: seq RNAME
  sDdnVal: MEASUREMENT
  cDdnVal: MEASUREMENT
  rDdnSelectedVal:  $\mathbb{Z}$ 
  rDict: RNAME  $\rightarrow$  Region
```

```

segmentsObjDict: RNAME  $\rightsquigarrow$  seq (seq GAMEOBJECT)
anglesObjDict: RNAME  $\rightsquigarrow$  seq GAMEOBJECT
regionsObjDict: RNAME  $\rightsquigarrow$  seq GAMEOBJECT

```

```

tmpGameObjArr: seq GAMEOBJECT

```

```

applmg: TEXTURE
appUrl: URL
uMagnitude:  $\mathbb{N}$ 
SceneLocked: BOOL

```

```

|
rDDN = dom rDict
dom segmentsObjDict = dom rDict
dom anglesObjDict = dom rDict
dom regionsObjDict = dom rDict
uMagnitude > 0
applmg  $\neq \emptyset$ 
appUrl  $\neq \emptyset$ 
rDdnSelectedVal  $\geq 1$ 
└

```

— **section** Initialization - Lock screen awaiting server update of Regions for current URL └

/ Region Environment Initialization State Schema */*

/ Lock the view totally, await update of Regions data structure of the current url: appUrl? */*

```

┌ Init
  RegLabEnv'
  applmg?: TEXTURE
  appUrl?: URL
  uMagnitude?:  $\mathbb{N}$ 
  p!: PROMPT
|
  mState' = MInit
  rDDN' = {(1, --)}
  rDdnSelectedVal' = 1
  sDdnVal' = cDdnVal' = --
  rDict' =  $\emptyset$ 
  segmentsDict' =  $\emptyset$ 
  anglesDict' =  $\emptyset$ 
  applmg' = applmg?
  appUrl' = appUrl?
  uMagnitude' = uMagnitude?
  tmpGameObjArr' =  $\emptyset$ 
  SceneLocked' = T
  p! = AwaitServerUpdate
└

```

/ WaitingServerUpdate maybe already obtained and cached in the AppRegion */*

— **section** RegionEnv state conditions \perp

/ code: any change to rDDN or sDDN must call a method that will reset state, this is where to start programming! */*

\perp StateIsCreateRegion
 \exists RegLabEnv
 |
 rDdnSelectedVal = 1
 sDdnVal = --
 \perp

\perp StateIsRegionUD
 \exists RegLabEnv
 |
 rDdnSelectedVal > 1
 sDdnVal = --
 \perp

\perp StateIsSegmentCUD
 \exists RegLabEnv
 |
 rDdnSelectedVal > 1
 sDdnVal \neq --
 \perp

\perp StateUpdate
 Δ RegLabEnv
 |
 rDdnSelectedVal \neq rDdnSelectedVal' \vee sDdnVal \neq sDdnVal'
 reEvaluateState(rDdnSelectedVal', sDdnVal')
 \perp

/ There is the update of Regions data structure - a dictionary. Using it, deduce other DSs */*

\perp OnOkServerUpdate
 Δ RegLabEnv
 appUrl?: URL
 rDict?: RNAME \rightarrow Region
 |
 appUrl = appUrl?
 rDDN' = {(1, --)} \cup dom rDict?
 rDict' = rDict?
 SceneLocked' = F

└

/* Rendering function could be called any time an update to rDict is made (from server or by the user) */

/* Using conjunction, include this schema in a robust def. of any schema when reRender is needed */

└ Render

ΔRegLabEnv

|

anglesObjDict' = anglesObjDict ⊕ {r: rDict' • renderRegionAngles(r)}

segmentsObjDict' = segmentsObjDict ⊕ {r: rDict' • renderRegionSegments(r)}

regionsObjDict' = regionsObjDict ⊕ {r: rDict' • renderRegionMarkers(r)}

└

└ OnBadOrNoServerUpdate

≡ RegLabEnv

p!: PROMPT

|

p! = InvalidUpdate_LetUserRegRepeat

└

/* Unsuccessful initialization ends up with a locked screen until a successful update from server */

/* The successful update brings the Regions data structure from server to client */

—

ServerUpdate == (OnOkServerUpdate ∧ Render) ∨ OnBadOrNoServerUpdate

└

— **section** -R- CRUD operations after successful initialization └

└ EnterCreateRegionState

ΔRegLabEnv

≡ StateIsCreateRegion

|

[(rDdnSelectedVal ≠ 1 ∧ rDdnSelectedVal' = 1 ∧ sDdnVal' = sDdnVal = --)

∨ (rDdnSelectedVal' = rDdnSelectedVal = 1 ∧ sDdnVal ≠ -- ∧ sDdnVal' = --)]

tmpGameObjArr' = ∅

mState' = MInit

└

/* tmpGameObjArr:

{txtA, txtB, txtC, LineImg} A and B for origin-terminus, C for region marker, LineImg: segment

*/

└ CreateRegionDiagonalOrigin

```

ΔRegLabEnv
≡ StateIsCreateRegion
p1?: POINT
|
mState = MInit
mState' = Origin
validRegionVertex(p1?)
tmpGameObjArr' = tmpGameObjArr ⊕ 1 ↦ textObjFrom(p1)
└

```

```

└ CreateRegionDiagonalTerminus
  ΔRegLabEnv
  ≡ StateIsCreateRegion
  p2?: POINT
  p!: PROMPT
  |
  mState = Origin
  mState' = Terminus
  validRegionVertex(p2?)
  let p1 = getVector2D(tmpGameObjArr(1))
  tmpGameObjArr' = tmpGameObjArr ⊕ {2 ↦ textObjFrom(p2),
3 ↦ regionMarkerFrom(p1,p2)}
  SceneLocked' = T
  p! = PleaseProvideRegionName
└

```

```

└ CreatedRegionSave
  ≡ RegLabEnv
  ≡ StateIsCreateRegion
  p!: PROMPT
  rName?: RNAME
  |
  SceneLocked = T
  p! = AwaitServerUpdate
└

```

```

└ CreatedRegionSave_FailedOnDuplicatedName
  ≡ RegLabEnv
  ≡ StateIsCreateRegion
  p!: PROMPT
  |
  SceneLocked = T
  p! = PleaseProvide_UniqueRegionName
└

```

```

—
CreatedRegionSave_RepeatOnDuplicatedName == CreatedRegionSave

```

└

/ selectNewCreatedRegion(rDDN) will happen before rendering, it causes rDDN-refresh, which calls Render */*

└ CreatedRegionSuccess

 ΔRegLabEnv

 ≡ StateIsCreateRegion

|

 tmpGameObjArr' = resetTmpGObjArr(tmpGameObjArr)

 updateWithNewRegionDS(rDict)

 selectNewCreatedRegion(rDDN)

└

/ reRender region game obj dict (as rDict is updated, running the already created render schema does the job) */*

/ Render will be realized from selectNewCreatedRegion(rDDN) of CreatedRegionSuccess, as it will cause rDDN refresh */*

—

 R_CreatedRegion == (CreatedRegionSuccess ∧ Render) ∨

 CreatedRegionSave_FailedOnDuplicatedName

└

/ State: SegmentCUD */*

/ crud-measures and reRendering */*

— **section** appregion2text **parents** standard_toolkit

└

/*

Existence effect:

rState: Environment state, being either in creation mode or read-update-delete, read -> measure

mState: either in '--', which is 'read mode of rState', or not '--', where M-CRUD happens for a region

rDDN: the list of region names in DDN

rDict: a function from region name to region data structure

deleted this one: toUpdate: a function from region name to boolean value, to define if the region DS is overwritten and must get updated

applmg: the image that the current region set belongs to

appUrl: the url of the image described above

uMagnitude: the value of the unit vector to apply (todo: apply a new unit value to all regions' segments)

rDdnSelectedVal: value of the current DDN selection, always equal to 1, at init (selected = '--')

SceneLocked: if scene is covered with a dialogue panel to communicate with user
*/

/* By here, we have regions DS either empty or not, meaning rDDN has either 1 dummy element "--" or more

and rDdnSelectedVal = 1, AND rendering phase is over with segments, region-markers and angles, we may:

- create region and select it

- if #rDDN>1 then read any of its entries, from there ready to update/delete, having Measure ≠ "--"

- if rDDN is selected to other than dummy element, then can change "Measure" from "--" to "A1, etc." and do measurement

- if rDDN is selected to other than dummy element, and "Measure" is on "--", then can update region, or delete it

main DSs: segmentsObjDict, anglesObjDict, regionsObjDict

*/

— **section** AppData FSM - Unit **parents** standard_toolkit
└

— [POS, URL, DATE, EVENTS] └

/* Boolean definition and meaning */

— **theorem** d_HasIntegerType

└? $d : \mathbb{Z}$

└

—

└ $F == (d \in \mathbb{Z} \wedge d \notin \mathbb{Z})$

└

—

└ $T == (d \in \mathbb{Z} \vee d \notin \mathbb{Z})$

└

—

└ $BOOL ::= T \mid F$

└

/* REQ target is the functionality that AppServiceProvider (abbr. AppProvider) offers */

—

└ $REQ ::= ReqLastUnit \mid ReqSaveUnit$

└

/* RES is the response of the server that comes back to App, via server >JS >proxy
>provider */

—

└ $RES ::= LastUnitSavedCB \mid LastUnitResCB$

└

/* Events to App */

— **theorem** evtIn

└? $\{UnitEvt, SetUnitEvt\} \subseteq EVENTS$

└

/* Events from App */

— **theorem** evtOut

└? $\{UnitUpdateEvt, UnitAvailableEvt\} \subseteq EVENTS$

└

/* State-schema and Init operation */

┌ AppUnit
 magnitude: \mathbb{N}
 p1, p2: POS
 url: URL
 date: DATE

```

isDirty: BOOL
|
  magnitude = 0  $\Leftrightarrow$  (p1 =  $\emptyset$   $\wedge$  p2 =  $\emptyset$   $\wedge$  url =  $\emptyset$   $\wedge$  date =  $\emptyset$ )
  magnitude  $\neq$  0  $\Leftrightarrow$  (p1  $\neq$   $\emptyset$   $\wedge$  p2  $\neq$   $\emptyset$   $\wedge$  url  $\neq$   $\emptyset$   $\wedge$  date  $\neq$   $\emptyset$ )
└

```

```

┌ AppUnitInit
  AppUnit'
  rq!: REQ
  |
    magnitude' = 0
    p1' =  $\emptyset$ 
    p2' =  $\emptyset$ 
    url' =  $\emptyset$ 
    date' =  $\emptyset$ 
    isDirty' = F

  rq! = ReqLastUnit
└

```

/ Operations */*

/ when user leaves unit scene, if unit created magn>0, appUnit sends srv evt sets dirty=true
 if a CB comes bk srv eql to unit, dirty sets false and user may nav to regions, else region disabled
 /

```

┌ AppOnSetUnitEvt
  ΔAppUnit
  e!, e?:EVENTS
  magnitude!, magnitude?:  $\mathbb{N}$ 
  p1!, p2!, p1?, p2?: POS
  url!, url?: URL
  date!, date?: DATE
  |
    e? = SetUnitEvt
    e! = UnitUpdateEvt

    magnitude? > 0
    magnitude' = magnitude?
    p1! = p1' = p1?
    p2! = p2' = p2?
    url! = url' = url?
    date! = date' = date?
    isDirty' = T
└

```

— **section** new1spec **parents** standard_toolkit

└

This specification describes state of App (scenes and data), home, unit, and region scenes.

Use case: measure a unit, measure images, export CSV file of measurements.

App, means Prj, meaning the Singleton Project Prefab of Zenject

/ Type definitions */*

— [URL, TEXTURE, VPOS, EVENT, UI, CHAR, SERVEROPS] └

/ Application Events */*

/ implementation: package authS and authF to authUpdateEvt*

/ events distributed for s:set meaning the event is between 's' and 'app' either as req. or res.*

or related to a concept like Authentication, that could be between any 's' and 'app'.

**/*

—

AuthEvents ::= authSuccessEvt | authFailEvt | qAuthStateEvt | reqLoginEvt

└

—

HomeEvents ::= hInitEvt | hDdnUpdateEvt

└

—

UnitEvents ::= ulnitEvt | qUnitEvt | unitUpdateEvt

└

—

ImgUrlEvents ::= reqTextureByUrlEvt

└

—

EVENTS == EVENT u AuthEvents u UnitEvents u HomeEvents

└

/ Boolean definition and meaning */*

— **theorem** xBool_HasIntegerType

└? xBool : \mathbb{Z}

└

—

F == (xBool $\in \mathbb{Z} \wedge$ xBool $\notin \mathbb{Z}$)

└

—

T == (xBool $\in \mathbb{Z} \vee$ xBool $\notin \mathbb{Z}$)

└

—

BOOL ::= T | F

```

└
—
SHOW == T
└
—
HIDE == F
└

—
VISIBILITY ::= SHOW | HIDE
└

—
UITYPE ::= inField | btn | ddn
└
—
PANEL == UI ↔ UITYPE
└
—
MNAME == A1 | A2 | B1 | B2 | C1 | C2 | D1 | D2 | UN
└
—
SCENE == HOME | UNIT | REGION
└

—
ENABLE == T
└
—
NAME == F
└
—
INTERACTIBILITY ::= enable | disable
└
—
ISINTRACTABLE == UITYPE ↔ INTERACTIBILITY
└
—
TEXT == seq CHAR
└
—
UITEXT == UITYPE ↔ TEXT
└
—

```

```
ASYNQUEUE ::= getLatestUrls | getTextureOfUrl
└
```

```
/* Schema types */
```

```
└ MVector
  p1, p2: VPOS
  uvMagn: ℕ
  name: MNAME
└
```

```
└ Img
  url: URL
  texture: TEXTURE
└
```

```
/* App State Schemas */
```

```
└ Signal
  FiredEvt: EVENTS → T
└
```

```
└ AppImgUrl
  curlImg: Img
  urls: seq URL
└
```

```
└ AppImgUrlInit
  AppImgUrl'
|
  curlImg' = ∅
  urls' = ∅
└
```

```
└ AppUnit
  unitLatest': MVector
  unitImg': Img
└
```

```
└ AppUnitInit
  AppUnit'
|
  unitLatest' = ∅
  unitImg' = ∅
```

└

└ AppData
 loggedIn: BOOL
 scene: SCENE
 hDDN: seq URL

└

└ AppDataInit
 AppData'
 |
 loggedIn' = FALSE
 scene' = HOME
 hDDN = ∅

└

└ AppProvider
 doLogin: EVENTS → BOOL
 asyncIntentQueue: ASYNQUEUE → SERVEROPS
 lastReqUrl: URL

└

└ AppProviderInit
 AppProvider'
 |
 doLogin' = ∅
 dom asyncIntentQueue' = ∅
 lastReqUrl' = ∅

└

└ Authenticated
 AppData

|

 loggedIn = T

└

└ UnAuthenticated
 AppData

|

 loggedIn = F

└

└ AppInit
 AppImgUrlInit
 AppUnitInit
 AppDataInit
 HomeInit
 AppProviderInit

└

/ Home State Schemas */*

```
┌ Home
  texture: TEXTURE
  chosenImg: URL  $\leftrightarrow$  TEXTURE
  hDDN: seq URL
  hMainPanel, hLoginPanel, hDialogPanel: PANEL
  homePanelState: IPANEL  $\leftrightarrow$  VISIBILITY
  isIntractable: ISINTRACTABLE
|
  hLoginPanel = {(InLogin, inField), (InPwr, inField), (BtnLogin, btn)}
  hMainPanel = {(BtnUploadNewImg, btn), (BtnLoadByRegion, btn), (BtnDelDdnImg, btn),
  (DdnHome, ddn), (BtnLogout, btn), (BtnUnitSize, btn), (BtnRegions, btn), (BtnExportCSV,
  btn)}
  hDDN  $\in$  DdnHome
   $\exists$  url  $\in$  ran hDDN • texture = chosenImg url
  dom homeCurState = {hMainPanel, hLoginPanel}
  homePanelState = {(hMainPanel, SHOW ), (hLoginPanel, HIDE)}  $\vee$  {(hMainPanel, HIDE
  ), (hLoginPanel, SHOW)}
└
```

/ The login panel is visible in case the user is not authenticated, otherwise it is hidden */*

```
┌ hLoginPanelON
  Home
|
  homePanelState = {(hMainPanel, HIDE ), (hLoginPanel, SHOW)}
└
```

```
┌ hLoginPanelOFF
  Home
|
  homePanelState = {(hMainPanel, SHOW ), (hLoginPanel, HIDE)}
└
```

```
┌ hLoginPanelSwitchLaw
  AppData
  Home
|
  (loggedIn  $\wedge$  hLoginPanelOFF)  $\vee$  ( $\neg$ loggedIn  $\wedge$  hLoginPanelON)
└
```

```
┌ hToggleLoginPanel
   $\Delta$ Home
```

```

    evt?: EVENTS
  |
  [(evt? =authSuccessEvt) ∧ hLoginPanelON'] ∨ [(evt? = authFailEvt) ∧ hLoginPanelOFF
  ,']
  └

```

/* At initialization, assume that user is not authenticated, show login panel and query auth state */

/* hInitEvt (home init signal) is equivalent of: qCurlImgAndUrlEvt, qUnitEvt, qImgUrlsEvt, qAuthStateEvt */

/* to initialize Home we need: current working image, ddn-urls, unit-existence-query, auth-state

/* BtnRegions: without a unit, we will not measure */

```

└ HomeInit
  ΔSignal
  Home'
  hLoginPanelON'
  |
  FiredEvt' {hInitEvt}
  texture' = ∅
  chosenImg' = ∅
  hDDN' = ∅
  isIntractible' = {BtnLogin→enable, BtnUploadNewImg→enable, BtnDelDdnImg→enable,
    BtnLoadByRegion→enable,DdnHome→disable, BtnLlogOut→enable,
    BtnUnitSize→enable,
    BtnRegions→enable, BtnExportCSV→enable}
  └

```

/* Login process: the Home and AppProvider are involved in the business */

/* Example why to Z-document software: */

When the login process is to take place, the field of password is not empty
 InPwr ≠ ∅, when in: hLogin

```

└ hUserLogin
  ΔSignal
  Home
  getText: UITEXT
  clickedOn?: btn
  username!, password!: TEXT
  |
  InPwr ≠ ∅
  InLogin ≠ ∅
  clickedOn? = BtnLogin
  username! = getText InLogin

```



```

password! = getText InPwrd
FiredEvt' {reqLoginEvt}
└─

└─ AppLogin
  Signal
  AppProvider
  evt?: EVENTS
  |
  evt? = reqLoginEvt
  [doLogin(evt?) ∧ FiredEvt(authSuccessEvt)] ∨ [¬doLogin(evt?) ∧ FiredEvt(authFailEvt)]
└─

```

/* at the start of Home panel, ddn may or may not have photos, if it has, the first one is auto-selected. If not, waiting for user to upload a photo. Btns of Unit, Region and CSV are disabled */

```

current working image, ddn-urls
/* informally:
when @home: => enable BtnUnitSize */
└─ Home_BtnRegions_Enabling
  Home
  e?: EVENTS
  unitMagnitude?: ℤ
  |
  unitMagnitude? > 0
  e? = unitUpdateEvt
  isIntractible' = isIntractible ⊕ {BtnRegions→enable}
└─

```

/* all possible hDDN events:

1. @Init: new seq of urls, or none:
 - ddnUpdateEvt (has seqUrls, url: equal to first one, texture of the url)
2. ddn selection changed in home, texture update is expected
 - reqTextureByUrlEvt (provide url in request)
 - response: - ddnUpdateEvt (has url set to reqUrl, texture, ddn list is null)
3. user uploaded new img, url @App level not found in seqURLs, so an evt with new seqURL with
 - added new img's url and texture comes back, totally updating ddn, selecting the url in it and
 - updating texture:
 - exactly same as #1
 - overall: if seq ddn is null, from the existing ddn select the evt.url if found, then rest

texture
if url is not in the current-existing seq ddn, raise error

event exchange:

1.

req: hInitEvt

res: hDdnUpdateEvt

abt: res may have or have not url-texture for selection, in app side, lastReqUrl will = head req hDDN

if, before texture of head req hDDN obtained and sent back, usr req another url from list, that one

sits in lastReqUrl and requested the texture of, from server, so the texture which will be sent to

user, will always be the last one that user requested.

2.

req: reqTextureByUrlEvt

res: lastReqUrl will be updated by url? of req evt, provider messaged with url? as param.

3.

event: provider received texture of a url

if app's lastReqUrl is that, it is sent back to user, else, texture is discarded

so:

if lastReqUrl = providerUrl? then appFire(hDdnUpdateEvt) with hDDN, url, texture.

4.

req:

*/

/* @Init: hInitEvt sent to app, this is response of app*/

┌ Home_DdnInit

 ΔHome

 e?: EVENTS

 ddnList?: seq URL

 texture?: TEXTURE

|

 e? = hDdnUpdateEvt

 texture' = texture?

 hDDN' = ddnList?

 url' = head ddnList?

└

/* @Selection REQ url? of user-selected being sent to app*/

┌ HomeDdn_SelectionChanged

 ΔSignal

```

ΔHome
url!: URL
getSelected: UITEXT
e!: EVENTS
|
url! = getSelected hDDN
e! =
  FiredEvt' {e!}
└

```

/*

A response to:

- user selected a url of ddn, or
- uploaded a new image,

*/

/* @Selection RES */

```

└ Home_ImgUpdate
  ΔHome
  e?: EVENTS
  url?: URL
  texture?: TEXTURE
  getSelected: UITEXT
  |
  url? ∈ ran hDDN
  e? = hDdnUpdateEvt
  getSelected' hDDN = url?
  texture' = texture?
└

```

/* error case: raise error, using hDialogPanel of Home schema, deferred!

/* left for reader exercise! */

/* @Err */

```

└ Home_ImgUpdate
  ∃ Home
  e?: EVENTS
  url?: URL
  texture?: TEXTURE
  getSelected: UITEXT
  |
  url? ∉ ran hDDN
  e? = hDdnUpdateEvt
└

```

/* @App */

/* @hInitEvt Signal */

send down ddn, even if null, but req JS by proxy, for latest 10Urls

└ AppHomeInitEvtRes

```

    ∃ Authenticated
    ∃ AppData
    Δ AppProvider
    e!, e?: EVENTS
    hDDN!: seq URL
|
    e? = hInitEvt
    hDDN! = hDDN
    if hDDN=∅ then dom asyncIntentQueue' = dom asyncIntentQueue u {getLatestUrls}
└

```

/* @ */

```

┌ AppGetUrlTexture
    ∃ Authenticated
    ∃ AppData
    Δ AppProvider
    url?: URL
|
    url? ≠ ∅
    wellformed(url?)
    asyncIntentQueue' = dom asyncIntentQueue u {getTextureOfUrl}
└

```

lastReqUrl

reqTextureByUrlEvt

hDdnUpdateEvt
 e?: EVENTS
 ddnList?: seq URL
 texture?: TEXTURE

url! = getSelected hDDN
 e! =

=====

```

┌ AppProvider
    doLogin: EVENTS → BOOL
    asyncIntentQueue: ASYNQUEUE
└

```

```

┌ AppProviderInit
    AppProvider'
|
    doLogin' = ∅
    asyncIntentQueue' = ∅
└

```

┌ AppImgUrl

```

curlImg: Img
urls: seq URL
└─

└─ AppImgUrlInit
  AppImgUrl'
  |
  curlImg' = ∅
  urls' = ∅
└─

└─ AppUnit
  unitLatest': MVector
  unitImg': Img
└─

└─ AppUnitInit
  AppUnit'
  |
  unitLatest' = ∅
  unitImg' = ∅
└─

└─ AppData
  loggedIn: BOOL
  scene: SCENE
  hDDN: seq URL
└─

└─ Authenticated
  AppData
  |
  loggedIn = T
└─

```

e! with url! of ddn goes to appProvider, chng AppImg downloading from inet, returning texture update req-e!

=> AppImg void (bye appProvider) for its internal texture and set img new url

=> provider to download img from inet itself

=> after download update appImg texture and issue event of appTextureUpdate

=> home makes sure coming url in ddn, then select it (if it's not, adds it), and updates texture

in Unit scene:

- if user deleted a unit which was loaded, or opened an img but didn't create any unit yet => isIntractible BtnAccept = disable

if cur img and url not available, the first of ddns will be chosen, if no ddns url, then wait for upload

when img upload, the url returns back to app, if home receives "just uploaded", if ddns is empty, will be placed there, else, add to ddns as last img.

logout btn must work like a master reset, as the next user is not necessarily the who was logged in

logout btn sends a reqNullReset to app, and app sends a reqSceneReset (any scene listens it)

```
/* hDDN changed issuing evt, if applmg.url is the same and applmg.texture not null, that will be sent as response, as "imgTextureUpdate", but if either texture is null or url different, provider will obtain that img, then both applmg updated and event toward cur-img-texture-update fired when a new-url comes in, sets to get its img download when new url-img available, ddns of app inspected, if not contains, an image of curlmgTextureAndDdnsUpdate fired curDdnsSelectionTextureUpdateEvt: will update ddns, select the url, and updates texture all together. */
```

```
/* current working image is function of selection from ddns.
```

ddns on every update will receive a copy of app url-seq, and totally updates to it

on every full ddns update, it issues query event asking cur app img and texture and gets updated to

if ddns has no img or has any, and one uploaded, the url is added to ddns then selected. if ddns has img and one is selected, the working image updates with it Application code guarantees that any img app is working on, belongs to imgUrlsList */


```
— section RegionFSM parents standard_toolkit
└
```

```
/* Specifying the concept of Region FSM */
```

```
— [ URL, TEXTURE, RNAME, GAMEOBJECT, VOID ] └
```

```
/* Boolean definition and meaning */
```

```
— theorem d_HasIntegerType
```

```
└? d :  $\mathbb{Z}$ 
```

```
└
```

```
—
```

```
└ F == (d ∈  $\mathbb{Z}$  ∧ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—
```

```
└ T == (d ∈  $\mathbb{Z}$  ∨ d ∉  $\mathbb{Z}$ )
```

```
└
```

```
—
```

```
└ BOOL ::= T | F
```

```
└
```

```
—
```

```
└ DDNRNAME == -- ∪ RNAME
```

```
└
```

```
—
```

```
└ MSET == {A1 , A2 , B1 , B2 , C1 , C2 , D1 , D2}
```

```
└
```

```
—
```

```
└ MEASUREMENT == MSET ∪ {--}
```

```
└
```

```
—
```

```
└ STATE ::= CreateRegion | RegionUD | SegmentCUD
```

```
└
```

```
└ Angles
```

```
└ E1, E2:  $\mathbb{N}$ 
```

```
|
```

```
└ E1 ≥ 0
```

```
└ E2 ≥ 0
```

```
└
```

```
/* the image is loaded in the 3rd section of XY coordinate system */
```

```
└ POINT
```

```
└ x,y:  $\mathbb{Z}$ 
```

```
|
```

```
└ x < 0
```



```

    y < 0
└─

└─ SEGMENT
    p1,p2: POINT
    |
    p1 ≠ p2
└─

└─ Region
    rname: RNAME
    diagonal: SEGMENT
    angles: Angles
    mSet: MSET ⇔ SEGMENT
└─

—
RenderFrom == Region ⇔ VOID
└─

—
SegmentOffIn_sUI == MSET ⇔ VOID
└─

—
RegionDraftUpdate == POINT × POINT ⇔ VOID
└─

—
BlockAndPromptSaveRegionDraft == RNAME ⇔ VOID
└─

—
InitSegmentUpdatedSet = (MSET × BOOL) → {sName:MSET • sName ⇨ F}
└─

```

```

/*
rDDN: seq RNAME // region dropdown
sDDN: seq MEASUREMENT // segment dropdown
rDict: RNAME ⇔ Region // dictionary of region name to region data structure
state: STATE // environment state, one of creationR | R-update/delete | segmentCRUD
rDDNVal: ℕ // selected value in region dropdown
sDDNVal: ℕ // selected value in segment dropdown
img: TEXTURE // texture chosen in home
url: URL // url of the chosen texture
unit: ℕ // unit created in unit scene
isRegionUpdated: BOOL // know if region-under-work is manipulated (and saving /
discarding as next ops)
previousSegment: MEASUREMENT // so that if any segment of prev. segment is

```

rendered off, render on back,
 when newSegment chosen in sDDN
 */

```

┌ RegionEnv
  rDDN: seq RNAME
  rDDNVal: ℕ

  sDDN: seq MEASUREMENT
  sDDNVal: ℕ

  rDict: RNAME ↔ Region

  regionDraft: Region

  state: STATE

  img: TEXTURE
  url: URL
  unit: ℕ

  isRegionUpdated: BOOL
  isSegmentUpdated: MSET → BOOL
  previousSegment: MEASUREMENT
|
  rDDNVal ∈ dom rDDN
  sDDNVal ∈ dom sDDN
  unit > 0
  img ≠ ∅
  url ≠ ∅
  rand rDDN = dom rDict
  isSegmentUpdated ≠ ∅
└

┌ RegionEnv_Init
  RegionEnv'
  img?: IMG
  url?: URL
  unit?: UNIT
  rseq?: seq REGION
|
  img' = img?
  url' = url?
  unit' = unit?
  rDDN' = 1↦-- ∧ {r:rseq? | (second r).rname ≠ -- • (first r+1) ↦ (second r).rname}
  rDict' = {r:rseq? • r.rname ↦ r}
  sDDN' = {1↦"---", 2↦A1, 3↦A2, 4↦B1, 5↦B2, 6↦C1, 7↦C2, 8↦D1, 9↦D2}

```

```

rDDNVal' = 1  $\wedge$  rDDN'(rDDNVal') = --
sDDNVal' = 1  $\wedge$  sDDN'(sDDNVal') = --

regionDraft =  $\emptyset$ 

isRegionUpdated' = F
isSegmentUpdated = InitSegmentUpdatedSet(isSegmentUpdated)
previousSegment' = --

state' = CreateRegion
└─

/*

```

The process of creating new region down by rFSM is deferred here, as a result of creation, a new region DS is sent to server for save and if saving succeeded the Region comes back, making the next op to take place, changing state from CreateRegion to RegionUD, and getting selected to rDDN, while sDDN remains unselected (it selects first value, i.e. --)

roles:

- rDict: latest region data structure from server, source of update region to original
- regionDraft: contains a copy of region from rDict to hold updates, and maybe saved to server or discarded
- RegionUI: datastructure to render Region UI, copies from rDict
- SegmentUI: data structure to render Segment UI, copies from rDict

Only source of information for Region that updates all others, is rDict, every other related DS is passive, but rDict is active DS.

```
*/
```

```

└─ NewRegion_fromServer
  ΔRegionEnv
  r?: Region
  |
  rDDN' = rDDN  $\cup$  {(#rDDN+1)  $\mapsto$  r?.rname}
  rDDNVal' = #rDDN+1
  regionDraft = r?
  rDict' = rDict  $\cup$  {r?.rname  $\mapsto$  r?}

  sDDNVal' = 1  $\wedge$  sDDN'(sDDNVal') = --

  isRegionUpdated = isRegionUpdated' = F

```

```
isSegmentUpdated = InitSegmentUpdatedSet(isSegmentUpdated)
previousSegment = previousSegment' = --
```

```
state = CreateRegion
state' = RegionUD
```

└

```
└ RegionUD_Entry
  ΔRegionEnv
```

|

```
state' = RegionUD
state ≠ state'
```

```
let region_name = rDDN'(rDDNVal')
  ∃RenderFrom(rDict(region_name))
```

└

```
└ RegionUD_to_CreateRegion
  ΔRegionEnv
```

|

```
rDDNVal' = 1 ∧ rDDN'(rDDNVal') = --
sDDNVal' = 1 ∧ sDDN'(sDDNVal') = --
```

```
isRegionUpdated' = F
previousSegment' = --
```

```
state = RegionUD
state' = CreateRegion
```

└

```
└ RegionUD_to_SegmentCUD
  ΔRegionEnv
```

|

```
isRegionUpdated = isRegionUpdated' = F
```

```
previousSegment = --
previousSegment' = sDDN'(sDDNVal')
sDDN'(sDDNVal') ≠ --
rDDN'(rDDNVal') ≠ --
```

```
let region_name = rDDN'(rDDNVal')
let segment_name = sDDN'(sDDNVal')
  ∃SegmentOffIn_sUI(region_name, segment_name)
  ∃MarkersAndFSM_to(rDict, region_name, segment_name)
```

```
state = RegionUD
```

```

state' = SegmentCUD
└─

└─ SegmentCUD_Segment_SelectionChanged
  ΔRegionEnv
  |
  sDDN'(sDDNVal') ≠ --
  rDDN'(rDDNVal') ≠ --

  previousSegment ≠ --
  let region_name = rDDN(rDDNVal)
  ∃RenderSegment(previousSegment, rDict, "on")

  previousSegment' = sDDN'(sDDNVal')
  ∃RenderSegment(previousSegment', rDict, "off")
  ∃MarkersAndFSM_to(rDict, region_name, previousSegment')

  state = state' = SegmentCUD
└─

└─ SegmentCUD_Region_SelectionChange
  ΔRegionEnv
  p1?, p2?: POINT
  v?: VOID
  |
  isRegionUpdated' = T
  rDDN'(rDDNVal') ≠ --
  ∃RegionDraftUpdate(p1?, p2?)
  let region_name = rDDN(rDDNVal)
  ∃BlockAndPromptSaveRegionDraft(region_name)
└─

└─ SegmentCUD_ChgSaveServer
  ΔRegionEnv
  v?: VOID
  promptSaveAccepted?: BOOL
  p!: REPORT
  |
  isRegionUpdated = T
  rDDN(rDDNVal) ≠ --
  promptSaveAccepted? = T
  let region_name = rDDN(rDDNVal)
  ∃ServerSaveRegionDraft(region_name)
  ∃BlockScene(v?)
  p! = PleaseWaitServerSavingGoinOn
└─

```

```

┌ SegmentCUD_ChgDiscard
  ΔRegionEnv
  v?: VOID
  promptSaveAccepted?: BOOL
  |
  isRegionUpdated = T
  isRegionUpdated' = F
  promptSaveAccepted? = F
  let region_name = rDDN(rDDNVal)
  ∃ResetDraft(region_name, rDict)
  ∃ResetSegmentUI(region_name, rDict)
  ∃UnBlockScene(v?)
└

```

```

┌ SegmentCUD_to_CreateRegion
  ΔRegionEnv
  |
  rDDN'(rDDNVal') = --
  isRegionUpdated = F;

  state = SegmentCUD
  state' = CreateRegion
└

```

```

┌ SegmentCUD_to_RegionUD
  ΔRegionEnv
  |
  rDDN'(rDDNVal') ≠ --
  isRegionUpdated = F;
  sDDN' ≠ sDDN

  state = SegmentCUD
  state' = RegionUD
└

```

```

— section Leaf Measurement parents standard_toolkit
└─

— [ URI, PHOTO, ITEMID, USERNAME, PASSWORD, UNIX_EPOCH, EVENT ] └─
└─
  LOGINEVT == EVENT ↔ {TRUE, FALSE}
  └─
└─
  FIREVT == EVENT ↔ {QueryLoginState, ReqLogin}
  └─
└─
  DATE == UNIX_EPOCH
  └─
└─
  URL == URI
  └─
└─
  REQ == PHOTOS | REGINAMES
  └─
└─
  EVENT ::= IsLoggedIn | IsLoggedOut
  └─
└─
  TRUE == 1
  └─
└─
  FALSE == 0
  └─
└─
  BOOLEAN == {TRUE, FALSE}
  └─
└─
  SAVED == BOOLEAN
  └─
└─
  SEGMENT_NAME == { A1, A2, B1, B2, C1, C2, D1, D2, E1, E2}
  └─
└─
  SEGMENT_SIZE == ℕ
  └─
└─
  POINT == X x Y
  └─
└─
  COORDINATES == POINT x POINT
  └─
└─

```

```

REGIONALS == COORDINATES
└
—
SEGMENT == COORDINATES
└
—
OVERLAP == (REGIONALS x REGIONALS) → BOOLEAN
└
—
INSIDEOF == ( SEGMENT x REGIONALS) → BOOLEAN
└

```

— **section** Main Data Structures **parents** standard_toolkit └

```

└ UnitVectorDS
  magnitude: SEGMENT_SIZE
  url: URL
└
└ RegionDS
  sgValue: SEGMENT_NAME → SEGMENT
  ofMagnitude: SEGMENT_NAME → SEGMENT_SIZE
  region: REGIONALS
  regionId: ITEMID
  url: URL
|
  ∀ s: sgValue • (ran(s) INSIDEOF region) = TRUE
└
—
  ⊢? ∀ r1, r2: RegionDS | r1 ≠ r2 ∧ r1.url = r2.url • r1.region OVERLAP r2.region = TRUE
└
└ DB
  munit: DATE ↔ UnitVectorDS
  regions: ITEMID ↔ RegionDS
  login: (USERNAME x PASSWORD) → BOOLEAN
└

```

— **section** Application Classes **parents** standard_toolkit └

```

└ class ProjectWideData
  └
    PhotoURL: URL
    IsLoggedIn: BOOLEAN
  └
  └ Init
    PhotoURL' = ∅
    IsLoggedIn' = FALSE
  └

```



```

┌ OnQueryLoginState
  e!, e?: EVENT
  |
  e? ∈ dom FIRE EVT
  EVENTFIRE e? = QueryLoginState
  e! ∈ dom LOGINEVT
  LOGINEVT e! = IsLoggedIn
└

┌
└ class CoverPanel
  ↑ ( Visibility , Username, Password )
  ┌
  visibility: BOOLEAN
  username: USERNAME
  password: PASSWORD
  └
  ┌ Init
  visibility = TRUE
  username = ∅
  password = ∅
  └

┌
└ class HomeSceneCtrl
  ↑ ( CoverPanelSet , BtnLogin_OnClick, UploadPhoto_OnClick , QueryByRegion_OnClick,
    RemovePhoto_OnClick, UnitScene_OnClick, RegionScene_OnClick )
  ┌
  CoverPanel
  └
  ┌ Init
  e!: EVENT
  |
  e! ∈ dom EVENTFIRE
  QueryLoginState = EVENTFIRE e!
  CoverPanel'.Init
  └
  ┌ CoverPanelSet
  Δ ( CoverPanelSet )
  e?: EVENT
  |
  e? ∈ dom LOGINEVT
  CoverPanel.visibility' = LOGINEVT e?
  └
  ┌ RequestLogin
  e! ∈ dom FIRE EVT
  EVENTFIRE e! = ReqLogin
  e!.data.username = CoverPanel.username

```

```

    e!.data.password = CoverPanel.password
  }
}

class HomeServiceProvider
  ↑ ( SetLoginState )
  [ OutboundService ]
  ReqLoginService == OutboundService.LoginService
  RequestLogin
  e? ∈ dom FIREEVT
  EVENTFIRE e? = ReqLogin
  ReqLoginService(e!.data.username, e!.data.password)
}

```

```

}
/*

```

1. JS when logged in, must req srv of latest 10 photos by date (assumption: latest photos is what user is interested to work with)
 2. user can make req of nav to next 10 photos from UI ddn related btns, and ddn will get updated
 3. if a photo-url from ddn is chosen, but the photo can't be obtained from srv, del from ddn too
 4. dialogue of Lynda pass to better learn UI
 5. photo full CRUD, before going any other part. Warn user if to del photo, all related regions get del
 6. all needed CRUDs:
 - photo
 - unit
 - region
 - measurements
- ```

*/

```