ICT Project Guidance

Design:   
Technical – BREAD-based Interface Organisation

Version:

0.3

Author:

Sky Sigal, Solution Architect

## Purpose

The purpose of this document is to improve the Usability -- specifically Learnability and Operability – of custom developed systems by describing the well known UX “BREAD” pattern for an optimal, repeatable, organisation of views and related flows common to managing entities in a system.

## Synopsis

Presented here is a means of using current web browser technologies to implement Browse/List, Read, Edit, Add, Delete (BREAD) views in a nestable and repeatable manner, increasing Learnability and Operability and decreasing user support costs.

## Contents

[Description 1](#_Toc145232973)

[Synopsis 1](#_Toc145232974)

[Contents 2](#_Toc145232975)

[Synopsis 3](#_Toc145232976)

[Introduction 3](#_Toc145232977)

[Heading Level 3 3](#_Toc145232978)

[Heading Level 4 3](#_Toc145232979)

[Appendices 4](#_Toc145232980)

[Appendix A - Document Information 4](#_Toc145232981)

[Images 4](#_Toc145232982)

[Tables 4](#_Toc145232983)

[References 4](#_Toc145232984)

[Review Distribution 4](#_Toc145232985)

[Audience 4](#_Toc145232986)

[Structure 4](#_Toc145232987)

[Diagrams 4](#_Toc145232988)

[Terms 5](#_Toc145232989)

## Introduction

Information Systems are developed to collect and render information to end users. interfaces are developed as a nested series of Views[[1]](#footnote-2) through which end users use action choices to navigate their way through.

The more intuitive, repetitive, and learnable the user interface is, the more efficient that end users become, and the less support they need.

## Issue

Developers are used to using the Create, Read, Update, Delete (CRUD) data management pattern, and reuse it out of context to develop user interfaces, producing Views similar to the following:

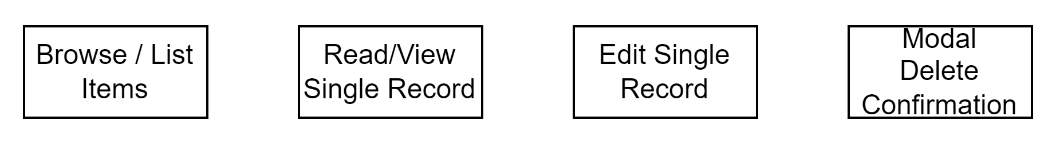


Figure : CRUD Views

By using a pattern intended for a different purpose, they have issues with addressing related aspects of user interface development that come up later, such as the following:

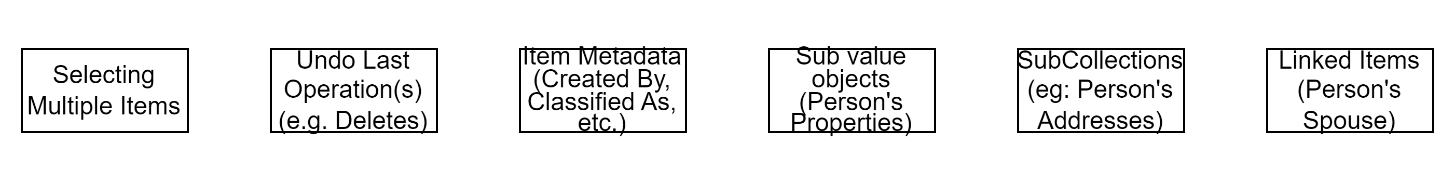


Figure : Common User Interface issues

At which point novel solutions for the above get proposed, which are implemented slightly differently each time.

These slight differences from view group to view group increase development costs and delivery risk, while decreasing the systems learnability, hence reducing end users’ efficiency, while increasing support costs.

Note:  
While CRUD[[2]](#footnote-3) is memorable, it has always had a couple of specific design flaws. For one, it’s R remains ambiguous as to whether it refers to Read Single or Read List. Secondly, it promotes the term “Delete”, which most developers think means a physical delete, rather than understanding that it should be a logical delete using an indexed flag.

## Resolution

Instead of using a data management pattern in the wrong context, use instead guidance specific to user interface development.

Use the Browse, Read, Edit, Add, Delete (BREAD) pattern to develop Views and organise flows between them.

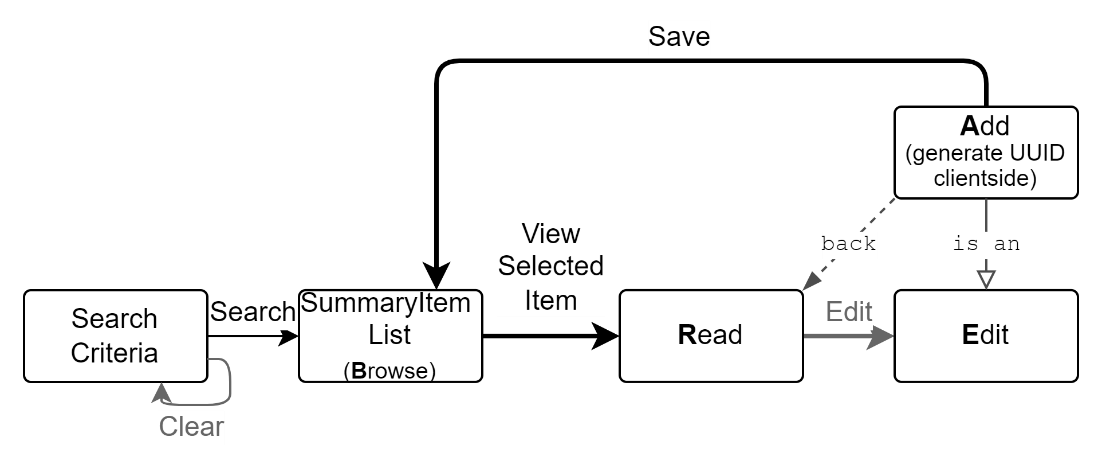


Figure : BREAD Flow Pattern (HL)

BREAD addresses CRUD’s key limitations, adding the means to browse search results, and keeping this action distinct from the Reading of a single resource, while also adding important high value outcomes.

When correctly designed to work with the service agent’s history stack, it produces a better overall user experience.

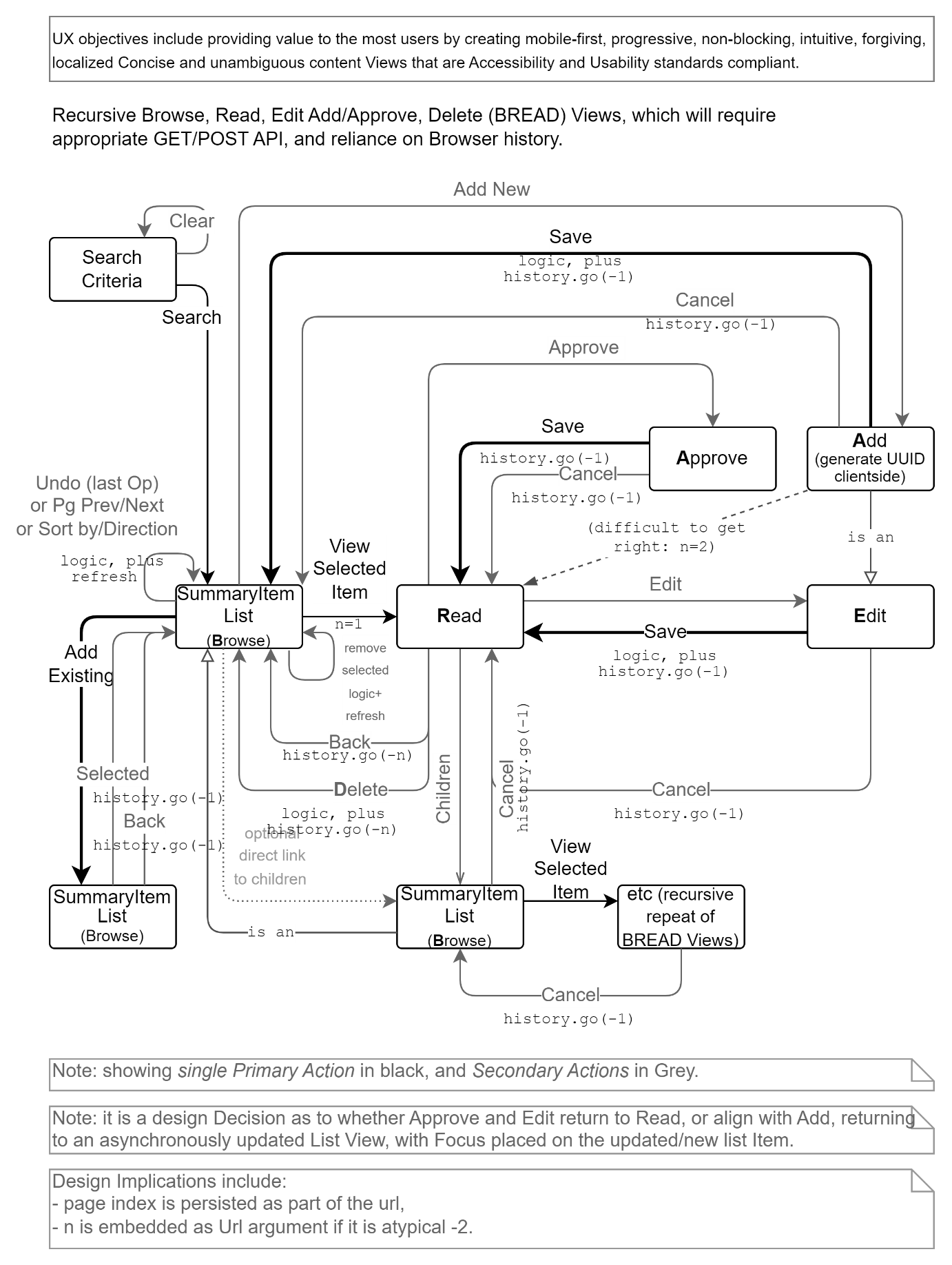


Figure : BREAD Recursive View Flows

The individual views are described in more detail below.

### Browse / List View

Intuitive and forgiving Search capabilities have largely replaced Menu driven navigation around features.

Most user interactions start with a Search based on Search Criteria, which can include one or more of Filtering, Ordering, and Presentation instructions.

The results are returned and rendered as search summary list items.



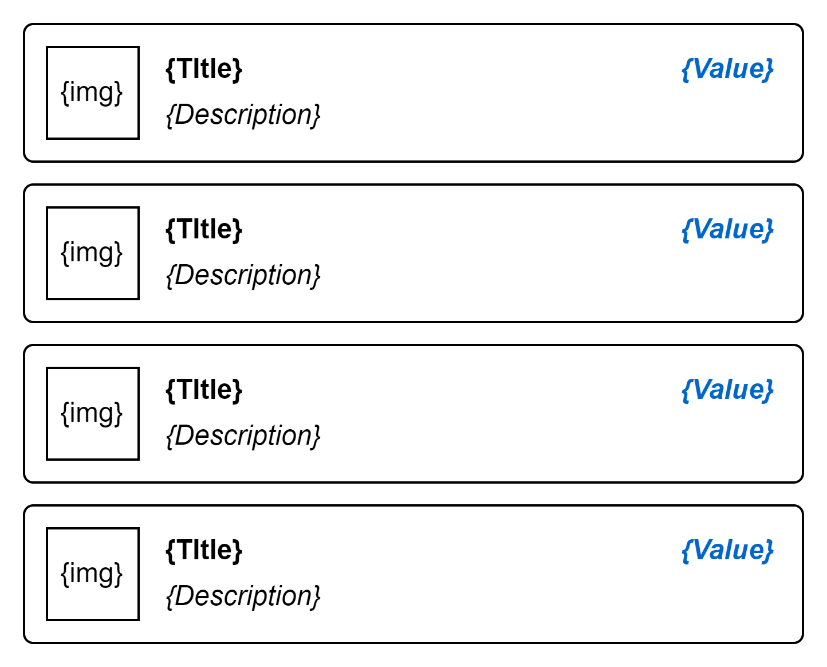


Figure : List of SummaryItems

Note:  
In the desktop era, items used to be rendered in a table. In an era of multiple screen sizes and formats (e.g., mobile to wall displays) current best practice is to develop list item summaries, and render them in a single column, dispensing with the constraints of a table.

The items that are rendered are not the record themselves, but *summaries* of the item, using a common format. A key benefit is that searches can span one or more source catalogues, as *SearchSummaryItems* shares the same format (e.g., Image, Title, Description, Value).

Note:   
This is similar to how Google search results: the item rendered is not the found remote page, but a short summary item, that has a title in bold used for the found page’s title, followed by a Description in italics, dynamically filled with key aspects of the page that relate to your search. Although Google does not, some search solutions may show an Image beside the information. The summary item should have a limited set (one or two ) of key values[[3]](#footnote-4) that may or may not be rendered (e.g., Page Rank, Search Relevance, Task done, etc.).

Selecting a single item triggers the Primary Action – displaying the listed Resource in a Read View, described next.

Alternatively, one or more items can be selected to then perform a small set of secondary then Actions on them (e.g., Delete, Approve, other). If these secondary actions are done, the list is refreshed to show the updated and/or truncated table, with the items unselected again, indicating the action has been completed.

### Read View

If a user selected a single *SearchSummaryItem* the primary action is triggered, and the user can view a presentation model of the resource.

Once again, as was the case for the Browse View, the model presented to the end user is *not* the resource itself, but a presentation model developed to optimise the display of information developed from the resource itself, but also may be made from several other elements or sources of information as well.

The Primary Action of the View is to Return to the source of this view – the previous Browse view[[4]](#footnote-5).

The View offers Secondary actions, one to Edit and the other Delete the Resource.

The Edit Action displays a new View of the model, described next.

Note:  
If developed correctly, the system will not physically delete the record, only doing so logically, via state – which is therefore undoable if need be. Being undoable, the action does not require a modal dialog to confirm the action with the end user.

A delete action brings the user back to the Browse table[[5]](#footnote-6), triggering a refresh of the same query to re-render it minus the item that was removed. Alternatively, if speed is of the essence, the record can be Deleted by client-side code, without re-querying the server -- but this approach causes the side effect of shortening the page (e.g., from 20 displayed records to 19).

### Edit View

If the Edit secondary action is chosen, the end user is presented with an editable form displaying a presentation model of the record.

Note:  
the use of labels or ghost labels is chosen by dynamic adaption and depends on the width of the device’s screen.

Again, the model shown is not the resource record itself, but a presentation model developed for the purpose.

The model may have sub parts. For example, a person may have a collection of 0 or more addresses. These collection items should be presented in a read only mode, on a separate section of the user interface. Maybe a child tab.

Note:  
the child records are read only, because they in turn deserve their own READ flow views (the list is acting as a fixed B of the BREAD).

The primary action of the Edit View is to Save changes and return to the previous View. Which is almost always the previous Read View, that requires re-fetching to update its appearance.

A secondary action is to abandon changes and return to the previous View (the Read View).

### Sublists

As alluded to earlier, a resource may have a fixed or dynamic collection of subitems (e.g., a Person may have zero or more Addresses or Contact Channels). For several reasons (performance, understandability, simplicity, etc.) these sub items should be displayed as read only items.

In fact, the list is almost identical to a nested Browse View described first, with the same Primary and Secondary actions (Read, Add, Delete) – the only difference is how they are rendered.

One can reuse *SearchSummaryItem* models and rendering templates – but in many cases, users will prefer a case-specific template (possibly one line, less high dimensionally, no graphics, etc.).

Note:  
In rare cases, if the child list is long, the nested Browse View may even show Search and filtering terms.

This makes the flow highly repetitive, BREAD within a parent BREADs within possibly a grandparent BREAD, etc.

Another key difference is that it simply becomes a system setting as to whether child BREAD save actions (after editing child resources) are triggered immediately or deferred until the parent record’s save action (effectively an all or nothing approach).

## Actions

As the Views offered are restrained, and components reused where possible, developing a repetitive flow pattern, the actions that are offered on the various Views are also restrained and reused where possible.

### Clicks

The physical button click actions are:

* Primary:
  + Search
  + View
  + Edit
  + Save
  + Cancel
* Secondary:
  + Delete (selected)

Note:  
Having a restrained pallet of actions makes it easier for end users to learn the system, as well as reducing the number of system interface resources that need to be translated.

Other

In addition to the above click events there are other non-button action event to plan for (e.g.: Drag Drop).

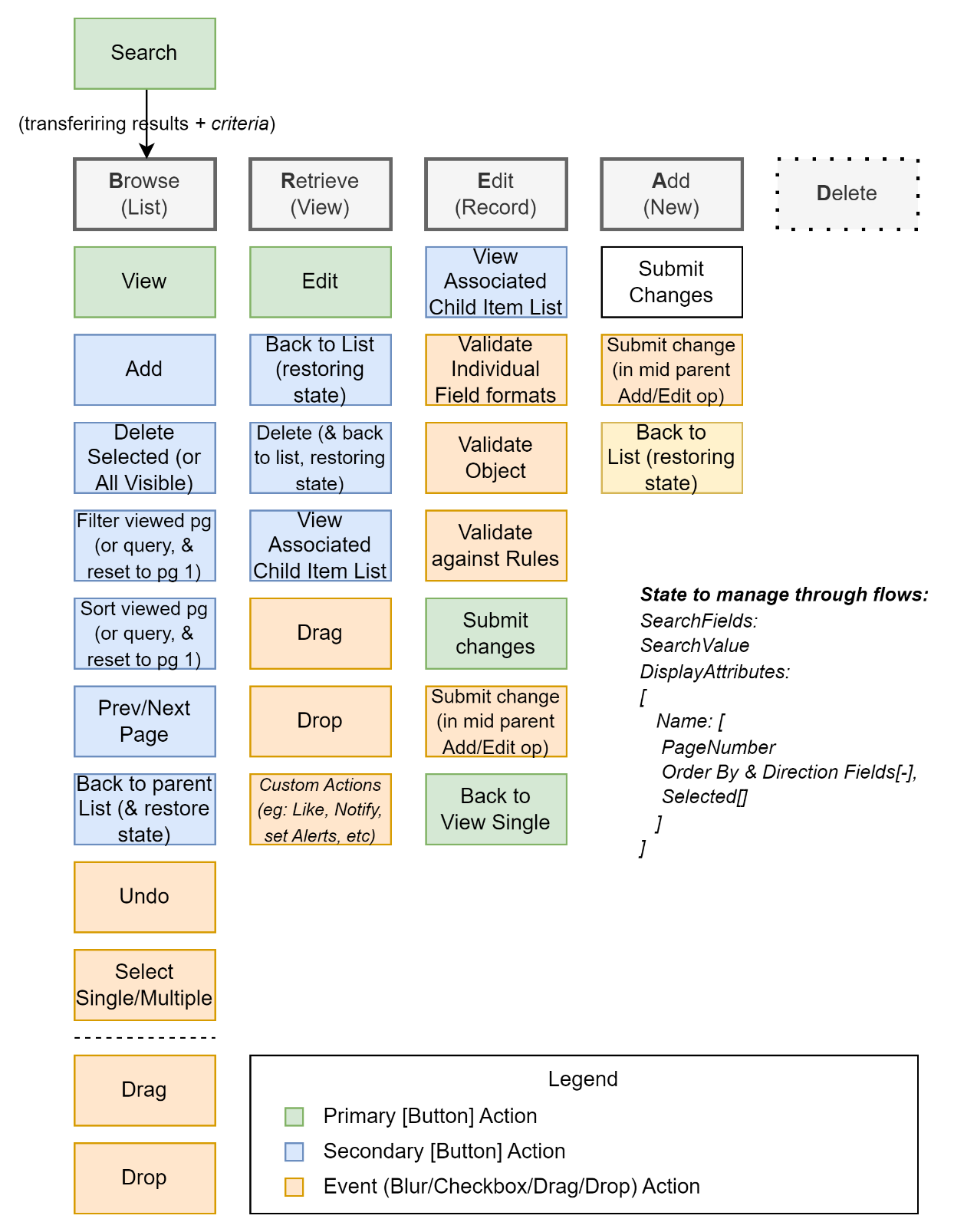


Figure : BREAD View Events to consider

## Conclusion

Using a more correct user interface Pattern to develop interfaces makes end user experiences more learnable, enjoyable, performant.

Appendices

Appendix A - Document Information

### Versions

* 1. Initial Draft
  2. Additions
  3. Added Events

### Images

[Figure 1: CRUD Views 3](#_Toc154144198)

[Figure 2: Common User Interface issues 3](#_Toc154144199)

[Figure 3: BREAD Flow Pattern (HL) 4](#_Toc154144200)

[Figure 4: BREAD Recursive View Flows 5](#_Toc154144201)

[Figure 5: List of SummaryItems 6](#_Toc154144202)

[Figure 6: BREAD View Events to consider 10](#_Toc154144203)

### Tables

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
| Sky Sigal, Solution Architect |  |
| Amy Orr, Data Domain Architect |  |
| Sandy Britain, Enterprise Architect |  |
| Russell Campbell, Project Manager |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### BREAD

: acronym for Browse, Read Edit, Add, Delete (BREAD).

##### CRUD

: acronym for the Data management pattern of Create, Read, Update, Delete.

History

: in the context of Service Agents, it’s the history of pages viewed. A developer can trigger a return to the previous page (e.g.: returning to the Browse View from Read View) by invoking the unwinding of the history stack (e.g. issuing an history(-1)).

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

##### Service Agent

: a web service agent, in the context of web servers, is an end user web browser.

##### View

: a viewable component within an interface. Views can only be one of 3 types: Input Views, Outputs, or Containers of other Containers, Input or Output Views. An Output View is never wired to be also an Input View, and vice versa.

1. Best practice is that they are developed as a series of nested Composite Views, which can in turn contain Input or Output Views. [↑](#footnote-ref-2)
2. We would have much preferred the concept of Create, Read, Update, State (CRUS) becoming well known – but it is less memorable. [↑](#footnote-ref-3)
3. See *IHasValue*. [↑](#footnote-ref-4)
4. by rolling the user agent’s History back one. [↑](#footnote-ref-5)
5. Again, by rolling the user agent’s history back by one. [↑](#footnote-ref-6)