BMC Software Inc.

Technical Disclosure Publication Document

Author
Chandrajit G. Joshi

Method and System for Migrating Fields from a Standard View to a Progressive View

Posted: December 11, 2024

Overview

This document describes an invention for migrating fields from a standard Mid-Tier View to a Progressive View to enable applications for Progressive Web Applications. The invention described herein will result on average in about 70% to 80% completion of a migration. This invention is not expected to do 100% of the migration because there will always be cases where app developers will:

- Make updates to panel structure and panel hierarchy in the Progressive View;
- Make updates to how migrated fields are grouped in the Progressive View; and
- Move fields from one panel to another, etc.

However, considering the complexity involved in migrating a standard Mid-Tier View to a Progressive View, completion of 70 to 80% of the migration is significant.

Background

This invention facilitates migration of the user interface from an old and dated Mid-Tier based technology to a technology that is more modern, user friendly and is compatible with modern handheld devices.

A progressive web application (PWA) provides a modern, mobile-friendly interface that is like Smart IT but is also customizable and extensible like Mid-Tier. A PWA represents a novel approach to presenting IT service management (ITSM) applications because it is developed using modern user interface (UI) technology, but at the same time, is feature-rich like a Mid-Tier View.

Many customers continue to develop custom applications on platforms. Observing the growing acceptance of Progressive Views and the modern user interface, along with its multi-device compatibility, there is a rising demand for tools and utilities that facilitate the conversion or migration of existing standard Mid-Tier Views into Progressive Views. Leveraging PWAs offers the opportunity to attain the modern UI and superior user experience that they provide.

This invention disclosure describes a method of migrating fields from a standard Mid-Tier View to a Progressive View.

Solution

This invention offers a mixed-mode approach to migrating fields from a standard view (SV) to a progressive view (PV), where "mixed-mode" refers to a layout based on both rows and columns. Depending on the field structure in the SV, the invention dynamically arranges the target PV into rows, columns, or a combination of both, adjusting per container field as follows:

- If fields with a parent ID of 0 (directly placed on the canvas) are best suited to a row-based layout, they are rendered in row-oriented panels in the PV.
- If a container field (such as a panel, panel holder, or trim box) contains child fields that fit a column-based layout, they are arranged in column-oriented panels in the PV.
- If a container field in the SV has a mix of rows and columns, the PV is structured accordingly in both row- and column-oriented panels.
- The field structure for the target PV is dynamically identified and selected at runtime by analyzing the existing structure in the SV.

The structure of fields in the SV are assessed at runtime to determine the optimal layout for the target PV.

Initialization Steps

- Retrieve Fields: Obtain the list of fields for the specified SV from the form.
- Create Data Structures:
  - Fields-by-Parent ID Map: A mapping with each parent ID as the key and its corresponding list of child fields as the value.
  - Ordered Parent ID List: A sequence of parent IDs starting with parent ID 0, followed by its child container fields' IDs, and then the child fields for each subsequent child container, recursively.
  - Fields-by-ID Map: A map where each field ID is the key, and the associated field object is the value, serving as a SV fields' cache.
- Process Each Parent in Order:
  - For each parent ID in the ordered list, retrieve its child fields from the Fields-by-Parent ID map.
  - Sort these child fields by their Y-axis coordinates, then use their X and Y coordinates to determine the appropriate number of columns, creating a list of Column objects.
  - Assess if a column-based structure alone is sufficient or if a row-based or a combination of row and column-based structures is needed to arrange all fields effectively (details on each option are provided in subsequent sections).

Creating a Column-Based Panel Structure

The column-oriented approach organizes fields into columns based on their layout. Figures 1 and 2 illustrate examples: in Figure 1, fields are easily divided into three columns, while in Figure 2, the structure is similar except for an additional container field (F5).

Key aspects of the column-based approach include:

- Default Structure Assumption: This approach assumes fields can be arranged into columns. Under this assumption, the algorithm generates Column objects to support panel creation for a column layout.
- Logic Flow:
  - Identify the fields with minimum and maximum coordinates (X1, Y1, X2, Y2) to determine the canvas dimensions.
  - Sort fields along the Y-axis; e.g., fields from Figure 1 are sorted as F1, F2, …, F5, F6, …, F9, F10, …, F13.
  - Use this sorted list to determine the number of columns, creating a Column object for each.
  - Divide the canvas width by the number of columns to set each column's width, using a uniform height for all columns.
- Field Sub-grouping Within Columns:
  - Columns are analyzed for container fields.
  - Non-container columns are grouped into a single bucket containing all fields.
  - Columns with container fields are subdivided:
    - Each container field forms its own bucket.
    - Non-container fields (data fields, buttons, view fields, etc.) are grouped separately. For example, in Figure 2, fields F1 to F4 form one bucket, while F5, a panel, forms a separate bucket.

Ultimately, this column-based structure guides panel creation for the target PV. Figure 3 and Figure 4 illustrate the resulting PV layout for the structures in Figures 1 and 2.

Creating a Row-based panel structure

In this approach, fields are divided into rows, and a row-oriented panel structure is created in the PV to match this layout, as illustrated in Figure 5.

Key design elements of the row-oriented approach include:

- Initial Column Evaluation: By default, a column-based structure is assumed as was mentioned earlier. With this assumption, Column objects are created. The container fields within each column are assessed to ensure they fit within the column boundaries.

- Switch to Row-Based Division: If a container field spans multiple columns, the Column objects are discarded, and fields are organized by rows:
    - Fields are first sorted along the X-axis (e.g., for Fig. 5, F1, F6, F9, F2, …, F13).
    - Each field is added to a row until a container field is encountered; this container field starts a new row.
    - Container fields are isolated into individual buckets within their rows.
- Adjusting Row Contents: After initial placement, fields are further adjusted based on proximity to container fields:
    - Non-container fields within the height range of a container field are moved to such container field's row but remain in separate buckets.
- Column Subdivision within Rows: Rows are then processed to sort fields along the Y-axis and subdivide them into multiple buckets, using the column-based division method on the sorted lists.

The final panel structure in the PV is based on this row-oriented field organization. Figure 6 demonstrates the PV layout for the SV in Figure 5.

Creating a Row-and-Column Panel Structure

The fields in this approach are organized in a row-and-column layout and then the panel structure is created in PV to match this configuration, as demonstrated in Figure 7.

Key design aspects of the row-and-column-oriented approach include:

- By default, a column-based structure is assumed. Once Column data structures are created, they are evaluated to ensure that each container field fits within its assigned column boundaries.
- If a container field overlaps multiple columns but not all of them on the canvas, the following steps are taken to adjust the structure:
    - For each container field in a column data structure, identify the number of columns it spans.
    - Merge these column structures to form a larger column if needed.
    - Apply the row-based division method within each merged column to create Row data structures.
    - For columns that are not merged, use the column division method to form individual buckets within those columns.
- Using this combination of row and column structures, the final panels are created in the target PV.

Figure 8 illustrates how the PV panel layout appears based on the SV shown in Figure 7.

Special Handling for Box-Type Trim Fields

Trim fields are horizontal and vertical lines, boxes, and text that enable the user to modify the appearance of a form. A transparent box-type trim field (also called a trim box) is used to group the fields visually in a SV. The box provides a border for the fields, which creates a visual group of the fields inside the border. But unlike container fields like panels and panel holders, the trim boxes are not the parent of the fields inside its borders.

A trim-box and a trim-text field is shown in Figure 9, which shows a portion of the canvas from a SV.

The heading for the trim box is usually provided using a trim-text field, as in Figure 9. For example, “System Information” is a trim-text field, and the trim-box field is the gray-colored box inside in which the fields are structured in two columns. The trim-box field provides visual grouping for the fields inside it.

When the fields are migrated to a PV, the fields should continue to be grouped together. To achieve this, the invention treats a trim-box field as a parent of the fields inside its borders – just like a panel is the parent of the fields inside it. This method is used to apply several criteria that are based on the X, Y coordinates of the trim box and the fields inside it to determine whether a field is inside the borders of a trim box. Though not necessary for this invention, with each criterion, a tolerance can be provided.

Drawings

Figure 1.

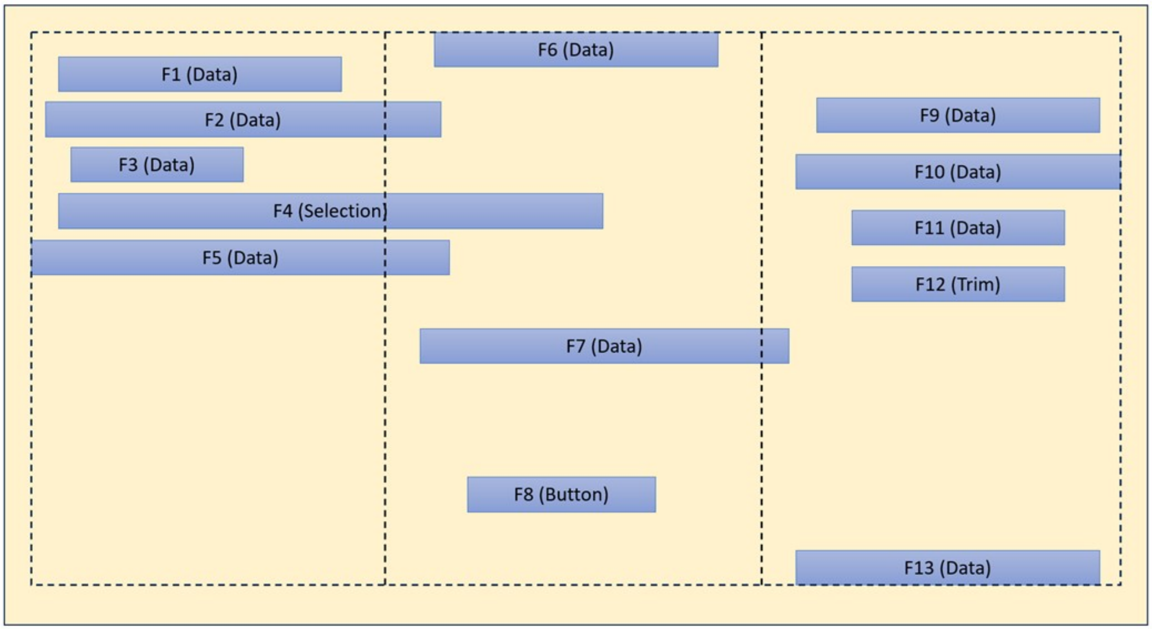


Figure 2.

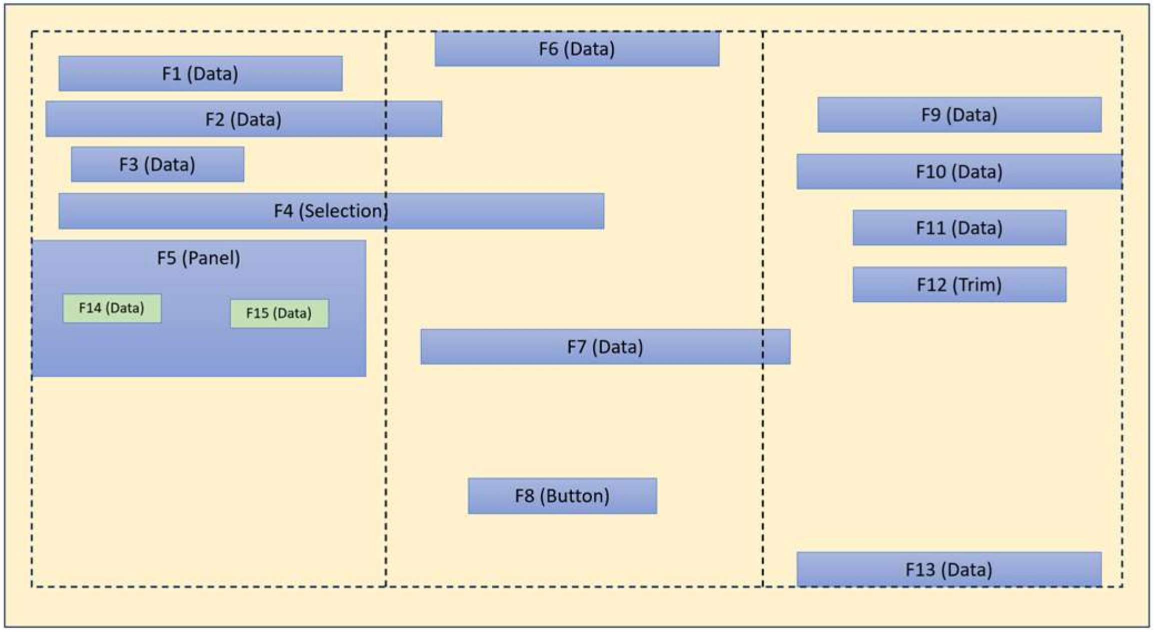


Figure 3.

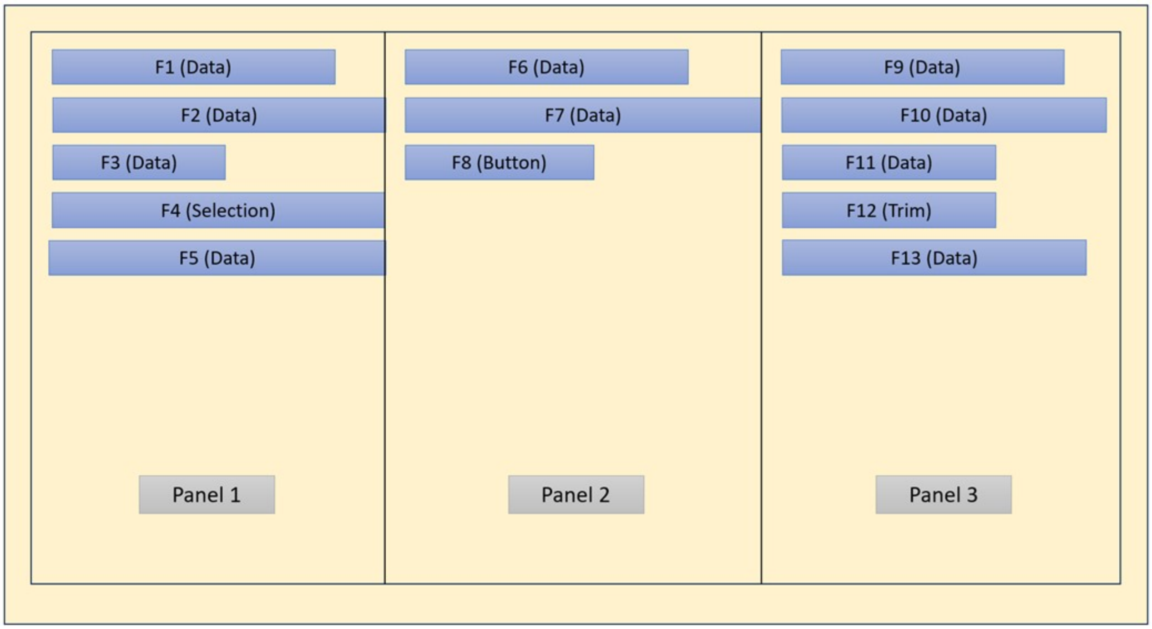


Figure 4.

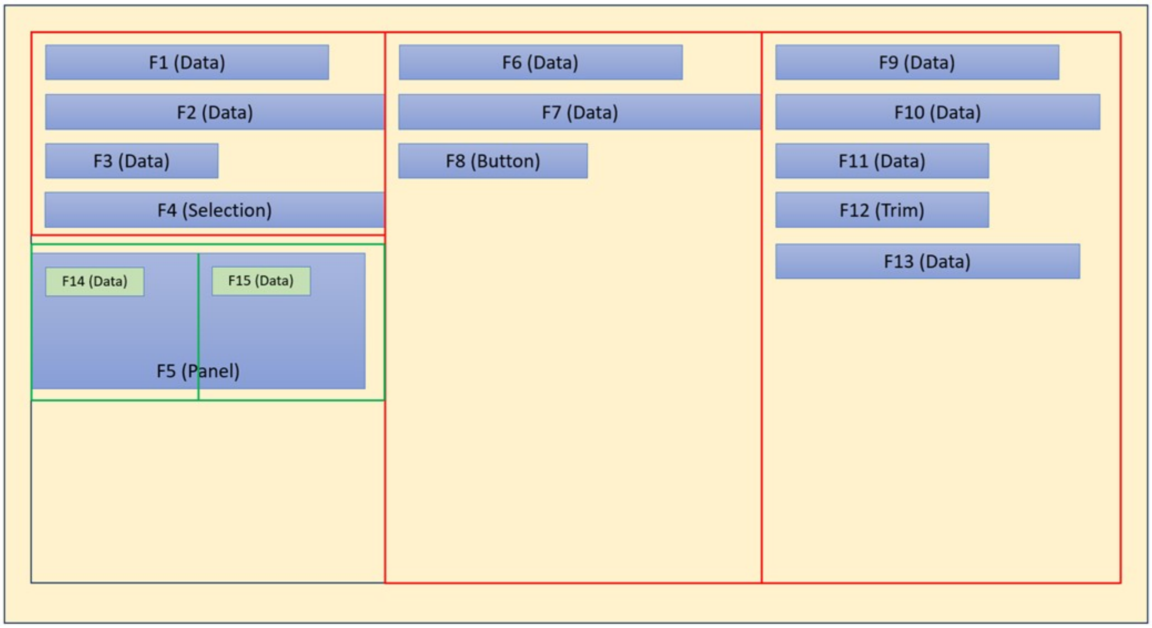


Figure 5.

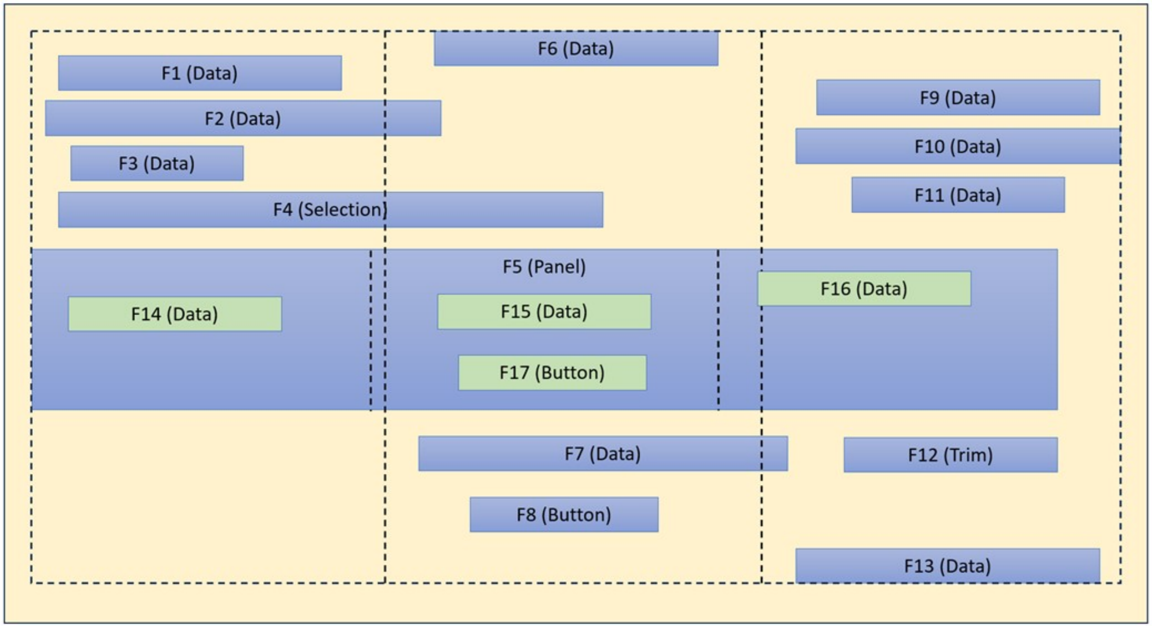


Figure 6.

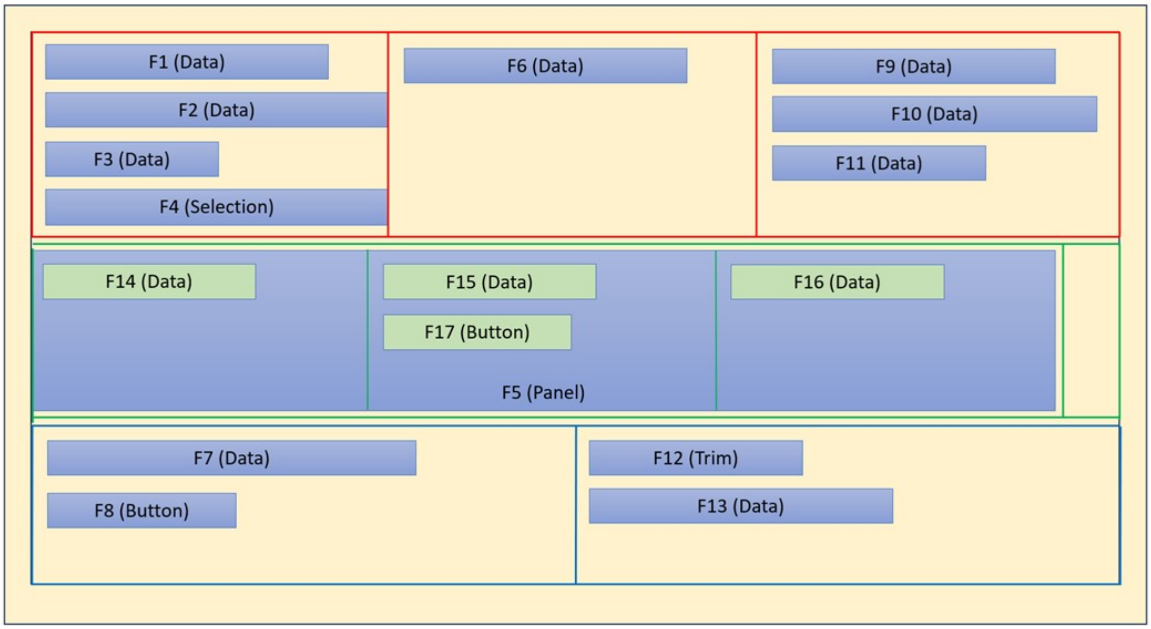


Figure 7.

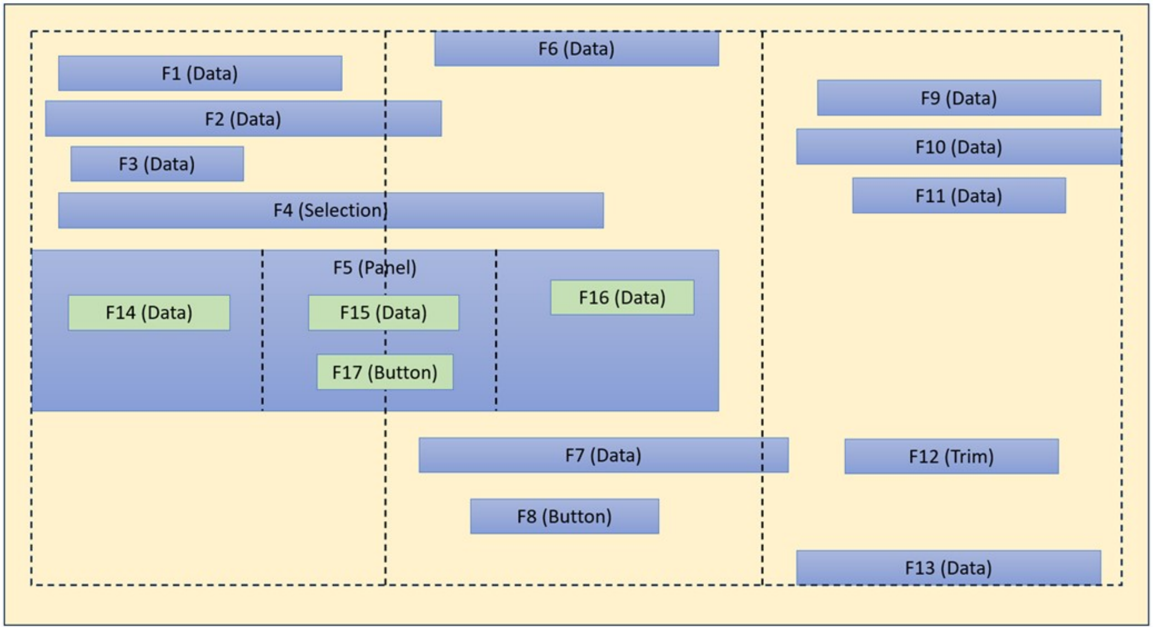


Figure 8.

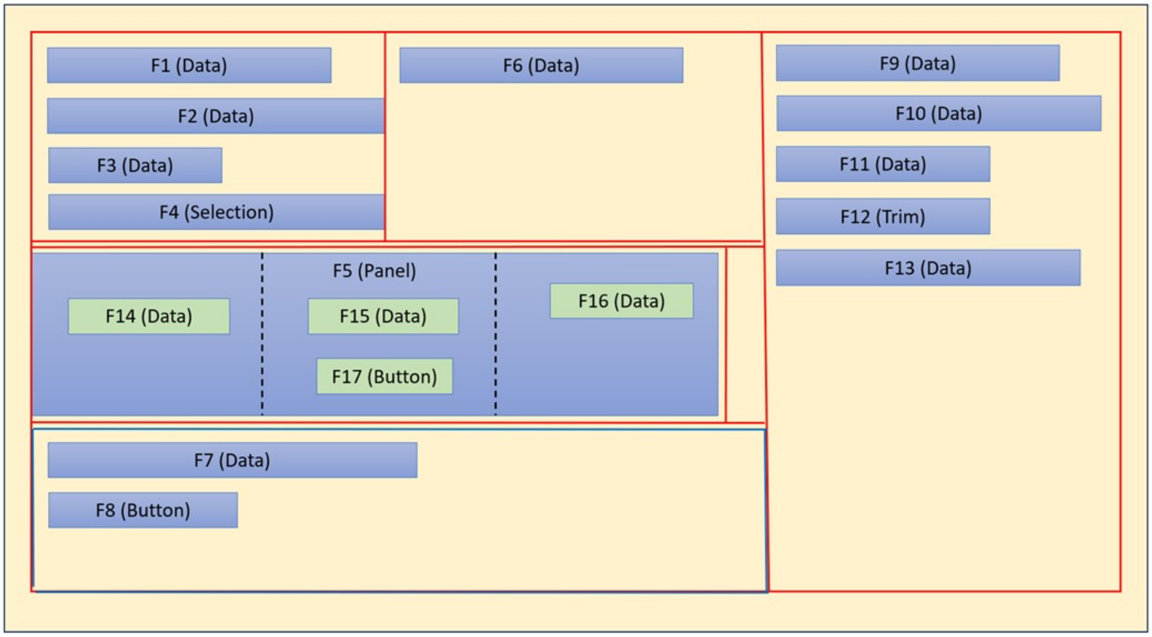
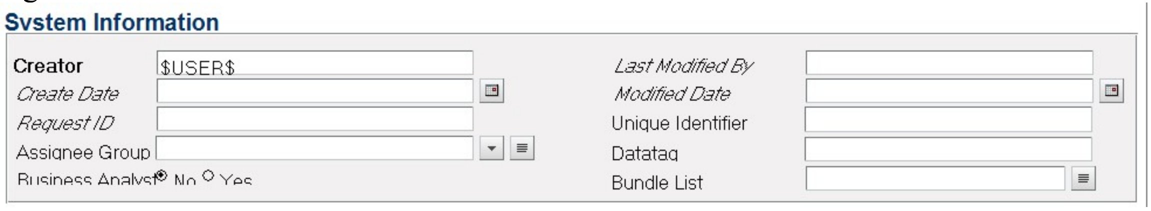


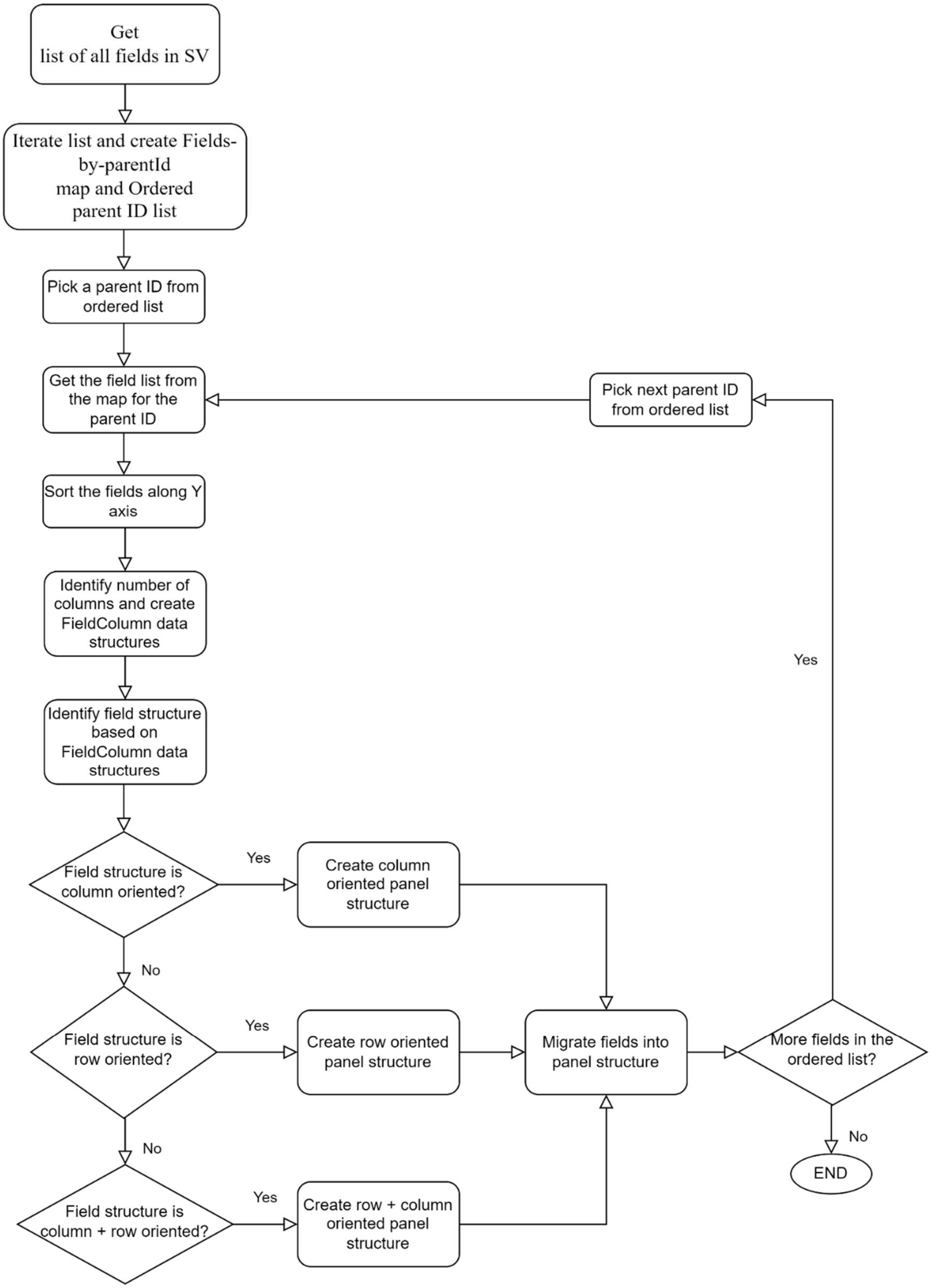
Figure 9.



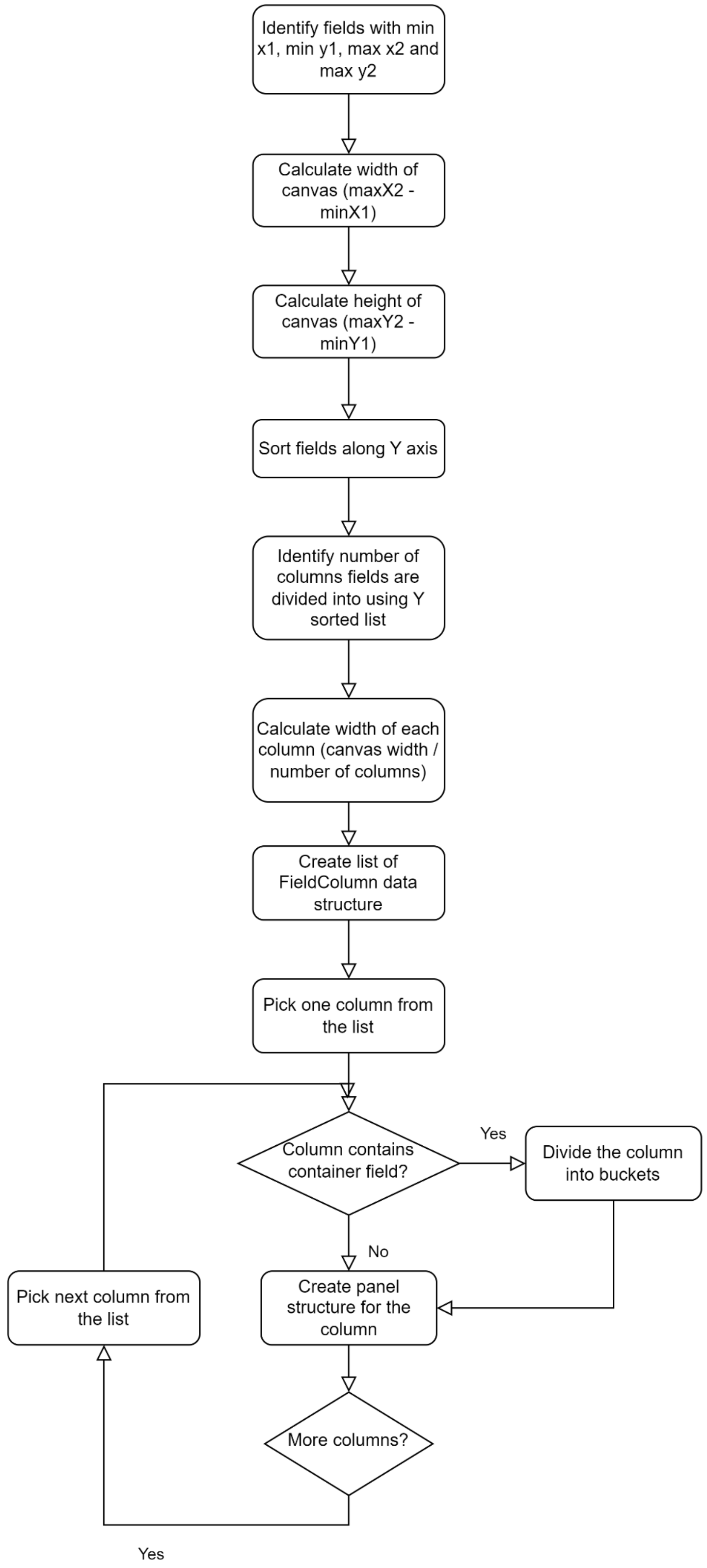
System Information

Creator	$USER$		*Last Modified By*	
Create Date			*Modified Date*	
Request ID			Unique Identifier	
Assignee Group			Datatag	
Business Analyst	● No ○ Yes		Bundle List	

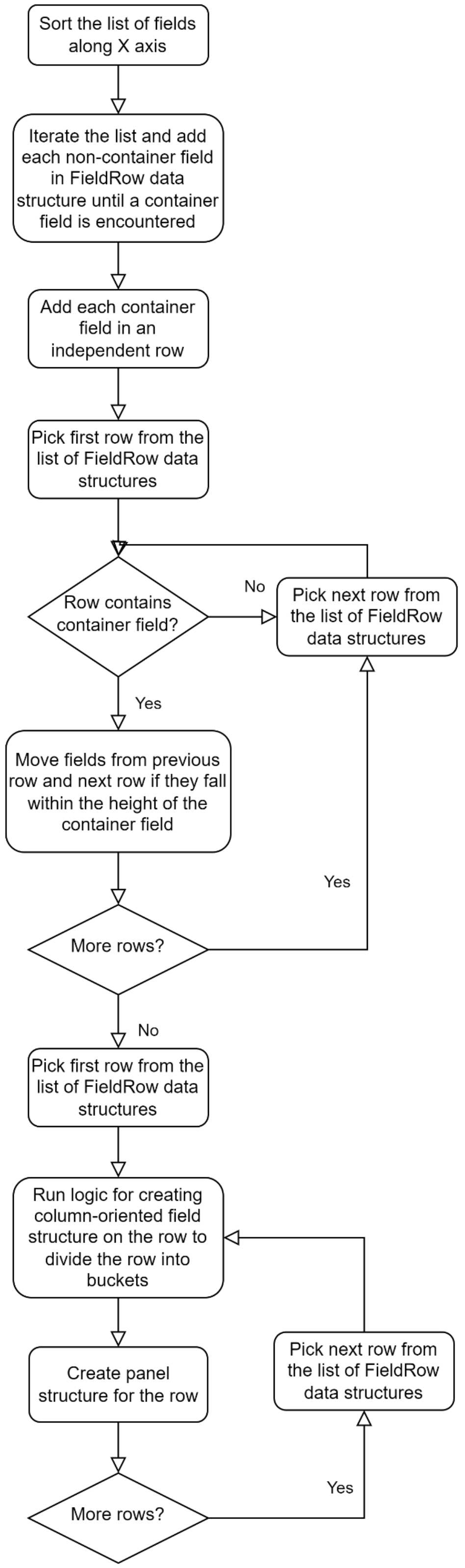
High-Level Flowchart:


Get list of all fields in SV
Iterate list and create Fields-by-parentId map and Ordered parent ID list
Pick a parent ID from ordered list
Get the field list from the map for the parent ID
Pick next parent ID from ordered list
Sort the fields along Y axis
Identify number of columns and create FieldColumn data structures
Identify field structure based on FieldColumn data structures
Yes
Field structure is column oriented?
Yes
Create column oriented panel structure
No
Field structure is row oriented?
Yes
Create row oriented panel structure
Migrate fields into panel structure
More fields in the ordered list?
No
No
END
Field structure is column + row oriented?
Yes
Create row + column oriented panel structure

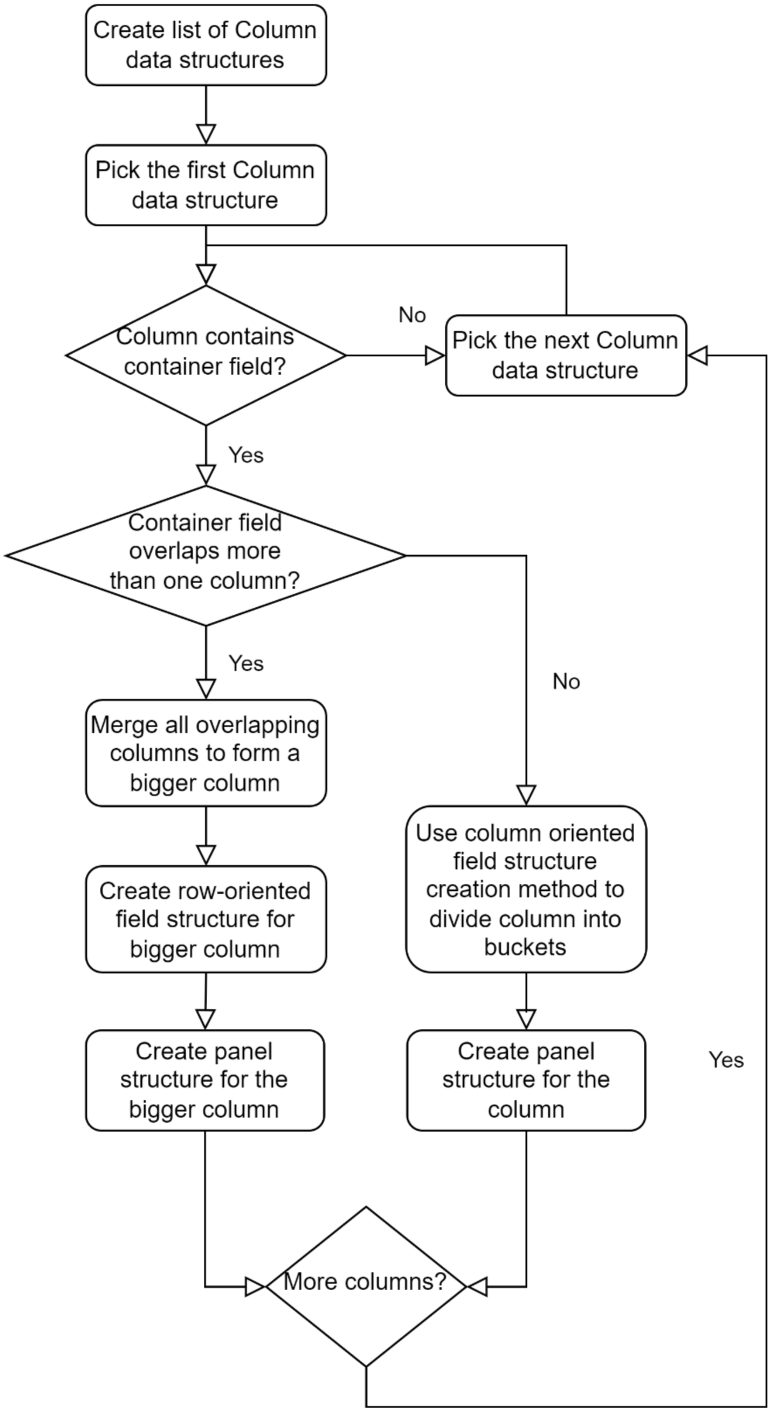
Flowchart – Column-Based Method:


Identify fields with min x1, min y1, max x2 and max y2
Calculate width of canvas (maxX2 - minX1)
Calculate height of canvas (maxY2 - minY1)
Sort fields along Y axis
Identify number of columns fields are divided into using Y sorted list
Calculate width of each column (canvas width / number of columns)
Create list of FieldColumn data structure
Pick one column from the list
Column contains container field?
Yes
Divide the column into buckets
No
Create panel structure for the column
More columns?
Yes
Pick next column from the list

Flowchart – Row-Based Method:


Sort the list of fields along X axis
Iterate the list and add each non-container field in FieldRow data structure until a container field is encountered
Add each container field in an independent row
Pick first row from the list of FieldRow data structures
Row contains container field?
No
Pick next row from the list of FieldRow data structures
Yes
Move fields from previous row and next row if they fall within the height of the container field
More rows?
Yes
No
Pick first row from the list of FieldRow data structures
Run logic for creating column-oriented field structure on the row to divide the row into buckets
Create panel structure for the row
Pick next row from the list of FieldRow data structures
More rows?
Yes

Flowchart – Row-And-Column-Based Method:


Create list of Column data structures
Pick the first Column data structure
Column contains container field?
No
Pick the next Column data structure
Yes
Container field overlaps more than one column?
Yes
No
Merge all overlapping columns to form a bigger column
Use column oriented field structure creation method to divide column into buckets
Create row-oriented field structure for bigger column
Create panel structure for the bigger column
Create panel structure for the column
Yes
More columns?