Development Guide

For PIRS Developers & Solution Partners

November 2024 | V3.0

Index

Welcome

to Samsung VXT CMS PIRS Application Development

1.1 What is Samsung VXT CMS?

Samsung VXT CMS (Content Management System) is a powerful Smart Signage platform that offers unparalleled versatility and potential. It sets itself apart from other Platforms on the Market by providing users and developers with access to a wide range of features and functionalities. With Samsung VXT CMS, you can expect to experience a new level of creativity and flexibility in managing your content. Whether you are a User or a Developer, Samsung VXT CMS revolutionizes the way you think about content management as a whole.

VXT, short for Visual eXperience Transformation, represents a significant milestone in Samsung's journey of delivering cutting-edge B2B solutions. This innovative Platform transcends traditional Digital Signage by offering unparalleled versatility and adaptability to meet evolving business needs. With VXT, Samsung continues to lead the industry in providing transformative experiences. First, Digital Signage in the late 2000s, thereafter, Smart Signage through the 2010s, and now ushering in a new era of Interactive Signage in the 2020s. Get ready to embrace the future with VXT and witness the endless possibilities it brings to your business.

Samsung VXT CMS supports the streamlined creation and integration of 3rd Party Apps that bring about a whole new level of vivid and engaging experiences for both Businesses and Consumers regardless of size, vertical, or level of expertise.

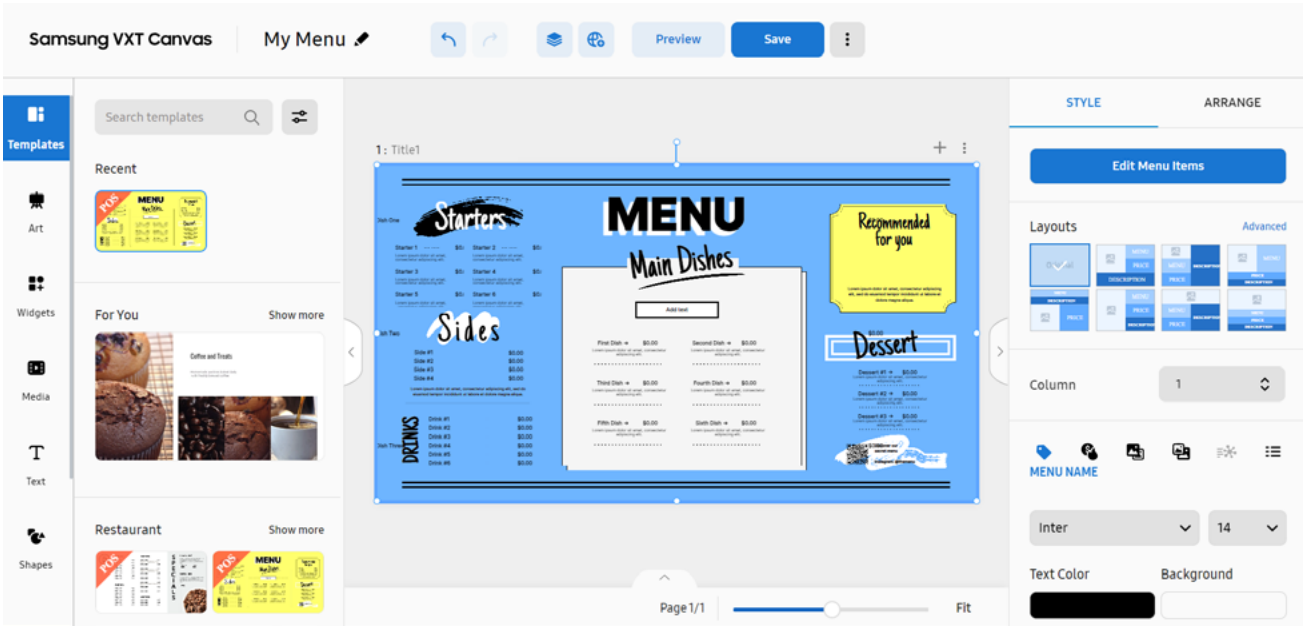
What's more, you can use Samsung VXT CMS not just on Samsung LFD (Large Format Displays), but also across a wide variety of 3rd Party Platforms all thanks to the platform agnostic nature of VXT Player (detailed further below).

Continue reading this Guide to learn more about how you can take the existing awesomeness of your on-going business operations to new level that you have yet to witness coupled together in partnership with Samsung, and Samsung VXT CMS. The future is...now!

1.2 What is Samsung VXT Canvas?

A major Component of Samsung VXT CMS lays in the VXT Canvas. As with any modern CMS Platform, a User can design their own Content based on their specific requirements. This process is performed directly within VXT Canvas using the in-built features made available through its WYSIWYG User Interface. Once a new Content design has been fully created, it can directly be viewed in VXT Canvas (itself), or saved in the form of a VXT Canvas Project, and later published to a Screen running VXT Player. All these are then available directly within VXT CMS within the Content Menu.

Figure1 VXT Canvas Overview.



1.3 What is Samsung VXT Player?

VXT Player is the Component of VXT that is responsible for playback of Content once it has been published and/or designed and published within Samsung VXT CMS or VXT Canvas. This Content can be of various Types and Formats such as Images, Videos, or Web Content. Due to the fact that Samsung VXT CMS also supports integration of 3rd Party Apps known as PIRS (Post Integrated Repeatable Solutions), or simply VXT Apps, VXT Player also supports rendering a special type of Web content leveraging the WiNE API Framework as detailed further in this Guide.

At the time of writing VXT Player can be used across the following Platforms: Web Browsers (Chrome, Edge, Firefox etc.) – known as a Virtual Screen, Samsung Tizen (4.0 and above), Android (10.x and above), and Windows (10 and above). Given the versatile nature of VXT Player, you can deploy almost any Device with a Screen, and use it directly with Samsung VXT CMS.

Figure 2

VXT Player.

2. Becoming a B2B VXT Partner

The Steps to Featuring your PIRS in Samsung VXT

2.1 Who is a B2B VXT Partner?

In general, a B2B VXT Partner is either a System Integrator with development capabilities, or a Software House who wishes to embrace on a future with Samsung VXT CMS, and specifically shares interest in developing PIRS in the form of VXT Apps. This is the theory or otherwise baseline, however in practice it is also encompasses several other aspects, which you will want to become familiar with before deciding to embark on such relationship with Samsung. Continue reading to learn more about how to qualify as a B2B VXT Partner in the paragraphs that follow.

2.2 How to Qualify as a B2B VXT Partner?

As of now becoming a B2B VXT Partner is primarily about your desire to develop PIRS throughout the tenure of your Contract. This means that you will devote your professional time also towards coming up and creating new VXT Apps that extend the native functionality offered within Samsung VXT CMS. It is also assumed that you have prior experience in writing Web Apps and/or you already are an existing Samsung B2B SSSP Partner.

As a B2B VXT Partner you are responsible for maintaining and updating your PIRS once it is released on Samsung VXT CMS. This involves keeping it up to date with changes in Web standards, and tweaking your codebase to ensure continued operability. In other cases, it means you will need to opt in to fixing errors within you PIRS once they are identified by either Samsung, or End Users who purchase your PIRS.

Additionally, as a B2B VXT Partner, you must agree to take ownership of your PIRS throughout its lifetime, or for however long it is featured on Samsung VXT CMS, and until licenses purchased by Users remain in active use. In practice, this signifies that you must participate in the role of ensuring Technical Support by addressing discovered issues in a timely manner such that they do not interrupt the daily usage of your PIRS by Users. These could involve handling standard technical problems, or fixing bugs as outlined above.

To succeed as a B2B VXT partner, your company and staff must also adhere to and adapt to changes within the WiNE Framework and WiNE API.

These components will continue to evolve, incorporating new features and addressing bugs. Consequently, you may need to modify your existing or upcoming PIRS to align with the evolving VXT Ecosystem, ensuring optimal performance and compatibility.

Samsung invites you to join forces and take your business operations to new heights by becoming a B2B VXT partner. As an existing Samsung Smart Signage Platform (SSSP) B2B Partner, you have the opportunity to embrace the next exciting chapter of your business and elevate VXT to unprecedented levels. Together let us shape the future of Digital Signage and create remarkable experiences for businesses and consumers worldwide.

2.3 How to Enroll as a B2B VXT Partner?

Signing up as a B2B VXT Partner is aimed to be an easy process with minimum formalities. Please take a moment to appreciate these requirements in order to enjoy collaborating with Samsung and VXT.

To become a B2B VXT Partner you should have expertise in Web Development technologies and practices, preferably a portfolio of Web Applications that you have the means to showcase to us, or at least introduce as concepts, which you are capable of delivering in practise. If you are already a B2B SSSP Partner, then you will have already ticked this checkbox, and can proceed to also becoming a B2B VXT Partner (see NDA comment below).

To ensure the protection of proprietary information related to the WiNE Framework/API, VXT Developer Portal, and VXT Player, Samsung requires that you sign a Non-Disclosure Agreement (NDA). This agreement is necessary for Partners who are already part of the B2B SSSP Program, or those who plan to become one. We will verify the existing NDA and request an updated version if needed. Please note that signing an NDA is essential to maintain the confidentiality of sensitive information shared during our collaboration.

2.4 What Information to Share with Samsung?

When applying to become a B2B VXT Partner, and unless we already know you, i.e. you are already a B2B SSSP Partner, we would like to receive from you the following information:

- Your Company (Business) Name,
- Your Website,
- Your Country of residence,
- Countries you already have a business presence in,
- Market Verticals you already serve,
- Your Portfolio of Web Applications,
- Key Contact Information.

Please share this required information with your local Samsung Pre-Sales representative when applying to become a B2B VXT Partner.

2.5 What Happens Next After a Successful Enrollment?

Once your Company embarks on an exciting and prosperous journey with Samsung in the form of a B2B VXT Partner, the time is ripe to discuss future plans on how we can together make Samsung VXT CMS best-in-class, and ignite our joint collaboration to a long series of success stories.

During these Meetings and exchanges of further information, we will want to discuss your ideas for PIRS, how you would seek their implementation, what Verticals these could cater for, and what Countries would these PIRS be targeted towards. Use this time to share your plans and enthusiasm, and let us know on any constraints on your end in order to make these visions a reality.

3. Understanding Key VXT Concepts

Understanding the various Elements and how PIRS interact with them

3.1 VXT Canvas

Once your PIRS is approved and integrated into the Samsung VXT CMS, it will be automatically accessible to all CMS users based on the regional support specified for the PIRS. To utilize PIRS, users need to log into VXT CMS using their Samsung Account. After logging in, they will see the VXT CMS dashboard as the initial view. From then on, users can navigate to VXT Canvas using one of the following paths:

3.1.1 Accessing VXT Canvas

Table 1

Accessing VXT Canvas (dashboard).

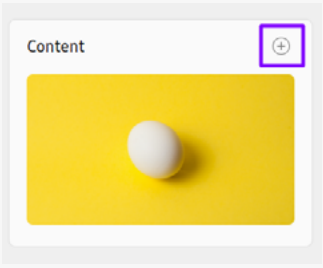
Option	Option 1 – from VXT Dashboard
Description	In VXT CMS dashboard, select the "+" icon by "Content"
Reference Image	

Table 2

Accessing VXT Canvas (content menu).

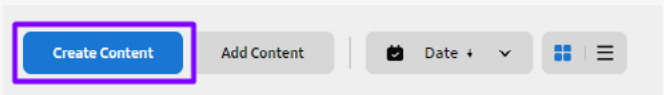
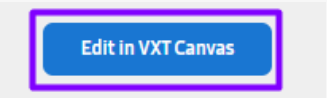
Option	Option 2 – from Content Menu
Description	From VXT CMS dashboard navigate to Content Menu, then click the "Create Content" button
Reference Image	

Table 3

Accessing VXT Canvas (content item).

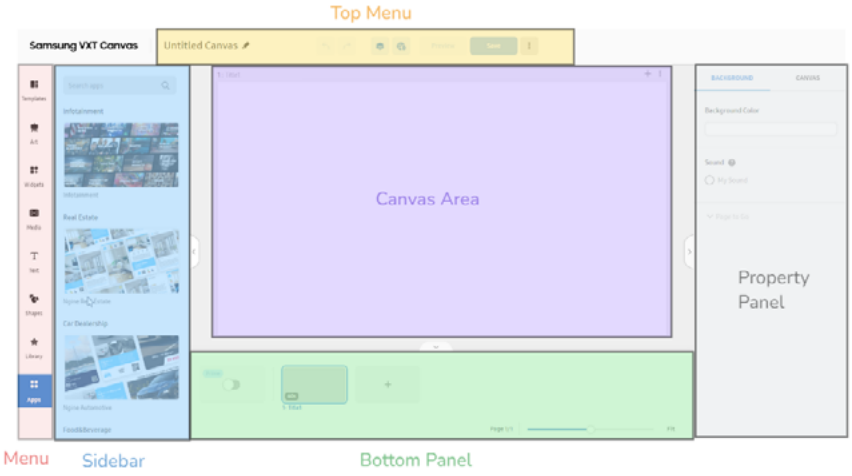
Option	Option 3 – from Content Item
Description	From VXT CMS dashboard navigate to Content Menu, then select Canvas Content Item (marked with a "Canvas" tag), then click "Edit in VXT Canvas" button.
Reference Image	

3.1.2 Canvas Features

- Opens in a new browser tab.
- Various content creation tools, including:
  - **Menu:** A comprehensive list of options.
  - **Sidebar:** Displays relevant content based on menu selections.
  - **Canvas Area:** Supports adding multiple pages with custom titles, page movement, duration adjustment, and transitions.
  - **Top Menu:** Provides VXT Canvas Project management options (saving, resetting, etc.), as well as additional content creation tools (such as ruler, guides, etc.) and access to manual and more information about VXT Canvas.
  - **Property Panel:** Canvas customization options, as well as element-specific properties displayed upon selection, allowing for extra customization of these elements.
  - **Bottom Panel:** Page management, including moving pages and adding a prime page (content shown on all subsequent pages as a bottom layer), zoom functionalities.

Figure 3

VXT Canvas Components.



3.2 VXT Menu

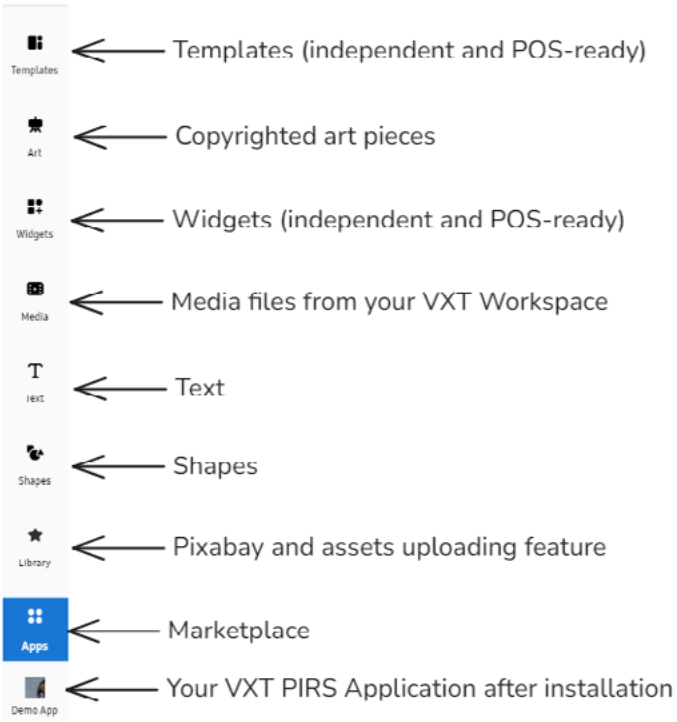
The VXT Menu, located on the vertical leftmost area of the VXT Canvas User Interface, is a key component for navigating through VXT Canvas.

3.2.1 Exploring the VXT Menu

The following sections are available:

Figure 4

VXT CMenu.



When User has selected and installed a PIRS Application from Apps, its icon will be displayed in the Menu bar, just under the Apps icon. It is possible to have multiple Applications installed - their icons will be displayed one after the other in a vertical direction.

3.3 VXT Sidebar

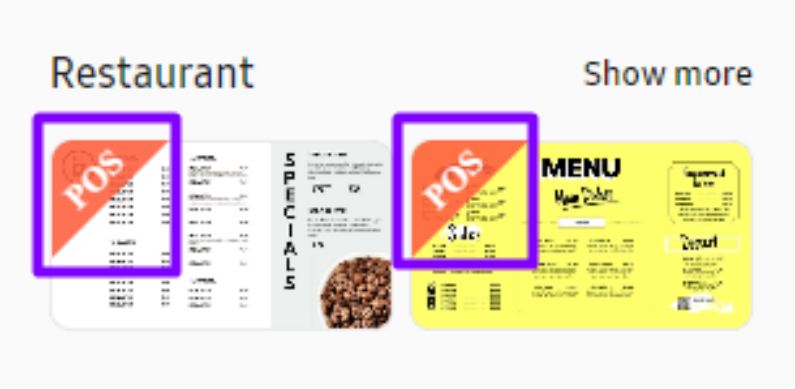
This is the panel that unfolds after User clicks on any icon in the Menu. It shows the details of each section, allowing for selecting elements and dragging and dropping them into the canvas area.

3.3.1 Templates

The Templates section offers a wide range of predesigned templates, enabling users without design expertise to create content efficiently. For a PIRS developer, the most important feature when it comes to Templates is the “POS” label. It indicates, that this particular Template can be populated with data provided by a PIRS Application (to be more specific, a PIRS Application of Data Driven type).

Figure 5

VXT POS-ready Templates.



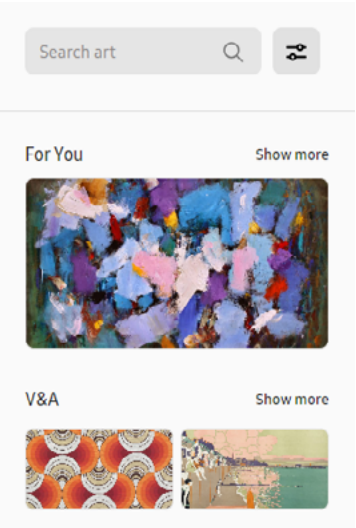
More on PIRS Application Types can be found in “Developing PIRS” Section – Chapter “Types of PIRS”.

3.3.2 Art

The Art section offers a diverse range of paintings, photos, and other visual content. Allows for selecting images from various categories and customizing their properties using the element’s Property Panel. Because Art is copyrighted content, modification options are limited.

Figure 6

VXT Art.

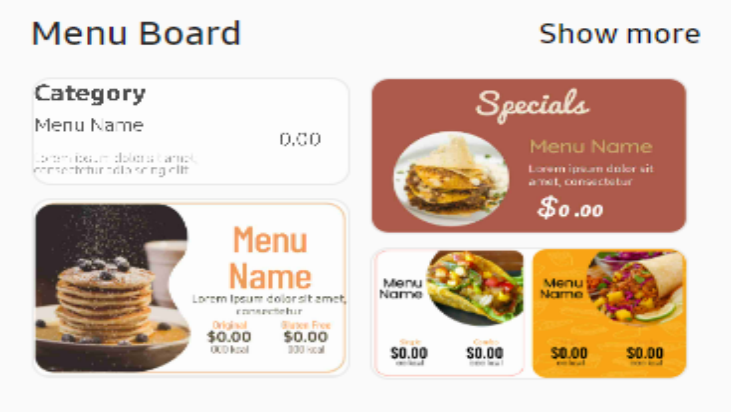


3.3.3 Widgets

Widgets are movable, resizable elements with various sources and effects, including menu boards and clocks. For a PIRS developer, the most important feature when it comes to Widgets is the “Menu Board” category. These Widgets can be populated with data provided by your PIRS Application (to be more specific, a PIRS Application of Data Driven type).

Figure 7

VXT Widgets.



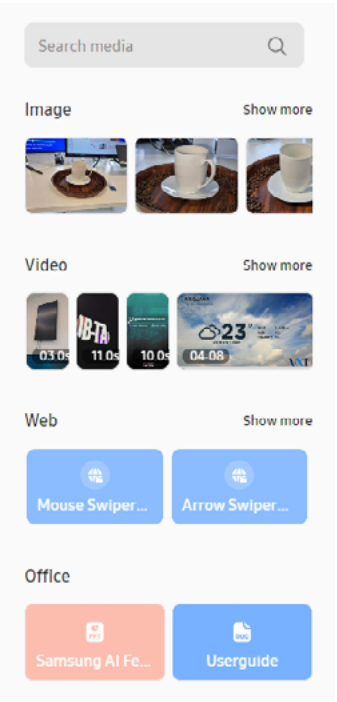
More on PIRS Application Types can be found in “Developing PIRS” Section – Chapter “Types of PIRS”.

3.3.4 Media

This section allows for loading media content such as images, videos, web content and documents from your VXT Workspace.

Figure 8

VXT Media.

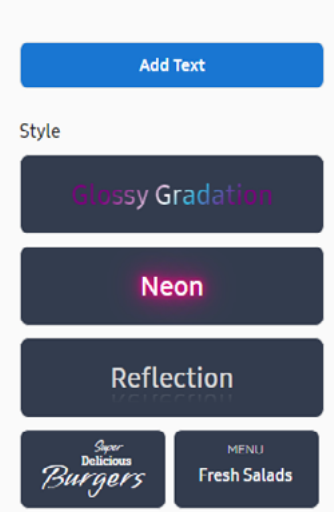


3.3.5 Text

Allows for adding and customizing text using the element's Property Panel.

Figure 9

VXT Text.

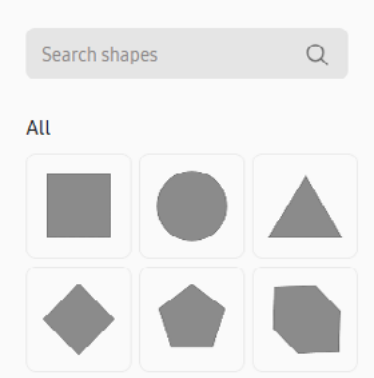


3.3.6 Shapes

In this section, it is possible to add shapes and adjust their properties using the element's Property Panel.

Figure 10

VXT Shapes.

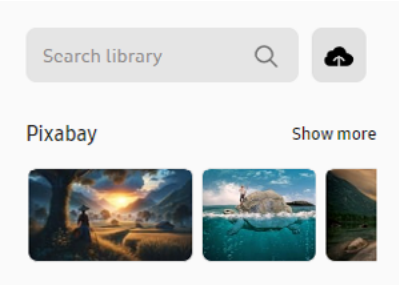


3.3.7 Library

The Library allows for uploading files from your device and searching and loading royalty-free images from Pixabay.

Figure 11

VXT Library.

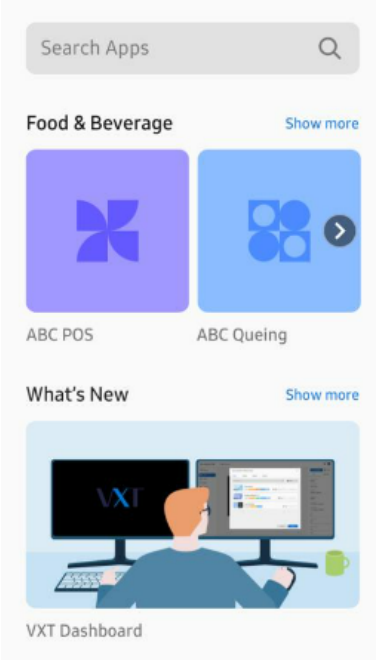


3.3.8 Apps

The Apps section serves as a VXT Marketplace, where PIRS Application are grouped into categories and are waiting to be found and installed. This is where your PIRS Application will be located after its release, under a category selected by you.

Figure 12

VXT Apps.

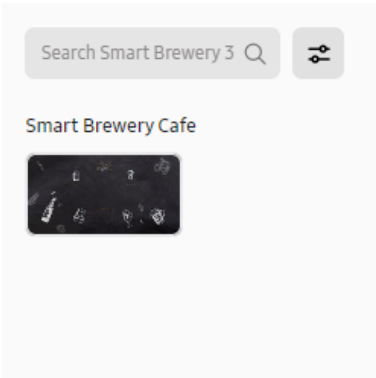


After Your Application has been selected (but not installed yet), it will display App Details Page in the Sidebar.

Once your Application has been installed and opened, the VXT Sidebar is where the Application Settings (if your App features any) and Extension Output/Metadata is displayed. We will discuss these Components in detail in the upcoming Chapters.

Figure 13

VXT Sidebar – Extension Output.



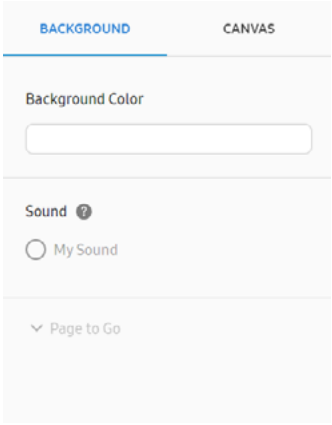
3.4 VXT Property Panel

VXT Canvas features a generic Property Panel, which is visible:

1. When canvas area is empty, or
2. When we click on the empty space of canvas area.

Figure 14

VXT Generic Property Panel.

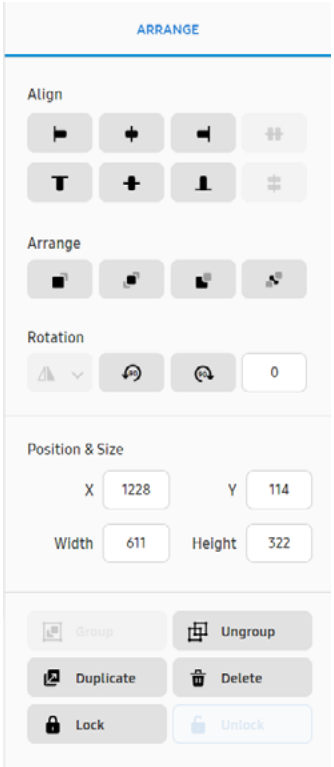


Every element (like templates, text, shapes, etc.) has its own Property Panel, which allows for further customization of it. It is visible once an element has been selected.

Below you can see an example of a Property Panel of a Text element.

Figure 15

VXT Text Element's Property Panel.

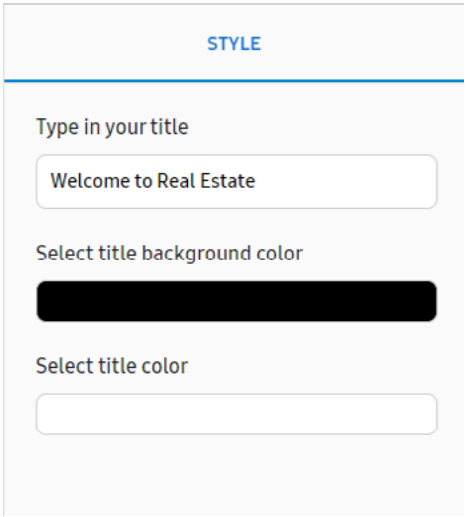


The same is true for your PIRS Application. Though it is not mandatory, it is highly recommended that your Application features a Property Panel to provide user-controllable options due to enhanced versatility it offers.

In the example picture below you can see a custom Property Panel defined by a PIRS developer using available Configuration Objects.

Figure 16

VXT PIRS Property Panel.



More on how to define custom Property Panel can be found in the “Developing PIRS” Section – Chapter “Creating your first VXT Widget Manifest (JSON) File (content-driven only)”.

3.5 VXT PIRS Application

PIRS = Pre-Integrated Repeatable Solution.

- **Diverse Solutions:** Serving multiple markets.
- **Development:** Created and uploaded by Partners or by Samsung as in-house VXT Applications.
- **Integration:** Seamlessly integrated within VXT, allowing users to install and use them without additional developments.
- **Marketplace:** VXT offers a marketplace for various PIRS applications to be easily accessed and installed.

PIRS can be developed in any one of two ways, which differ depending on the role our Application is to service within VXT Canvas and VXT Player. These can be broken down into:

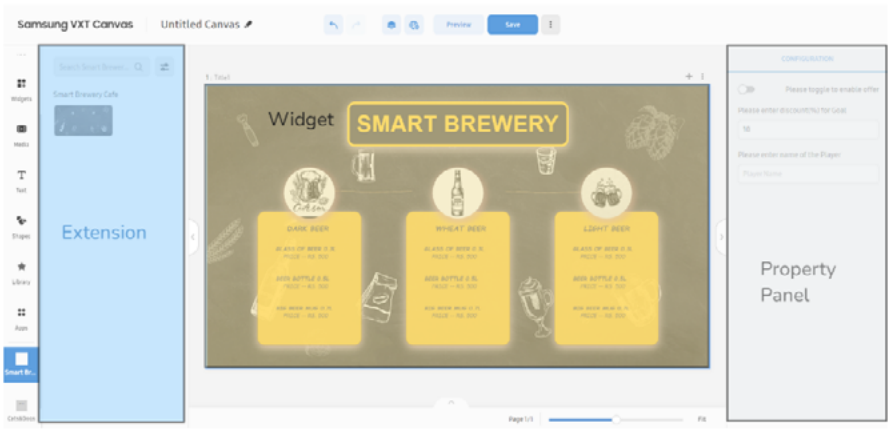
- Data Driven Applications
- Content Driven Applications

In order to integrate an App into VXT Canvas, you need to observe rules related to file structure and learn about the Framework, within which the Apps operate and communicate with VXT (the WiNE Framework).

Core component of every PIRS Application is Extension. Some Applications (of Content Driven Type) feature also the Widget Component. A high-level overview of both Components has been presented in the Chapters below.

Figure 17

VXT PIRS Overview.



All of the above will be discussed in great detail in the upcoming Chapters.

3.6 VXT Extension

3.6.1 Purpose and Functionality

From the moment a PIRS is installed and opened in VXT Canvas, the first Component that is loaded is Extension. It is displayed within the left VXT Sidebar area and can be thought as a place to present all the available Widget's options / designs, which may also be grouped into Categories (for Content-Driven applications) or a place to display meaningful information on how to use the App (for Data-Driven applications).

This can be preceded by optional Settings, which is a place where User may configure our Application to customize their user experience and tailor it to their needs (for example, by selecting a language). When User makes their selection, then is presented with appropriate set of Widgets (for Content-Driven applications), or proper Metadata (for Data-Driven applications) that corresponds with their choices made in Settings (for example, the Widgets that display content only in selected language or Metadata that displays information in selected language).

3.6.2 UI Overview

For Content-Driven Application, the UI of Extension Component can look as follows:

Figure 18

VXT PIRS Settings

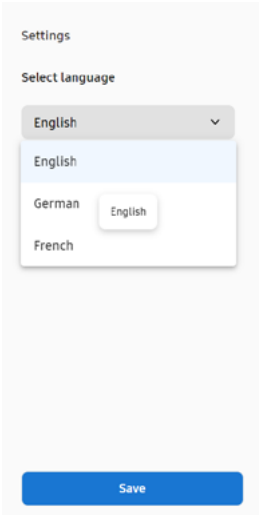
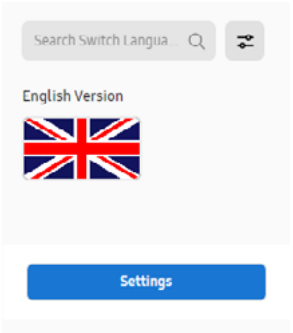


Figure 19

VXT PIRS Extension Output.



For Data-Driven Application, the UI of Extension Component can look as follows:

Figure 20

VXT PIRS Settings.

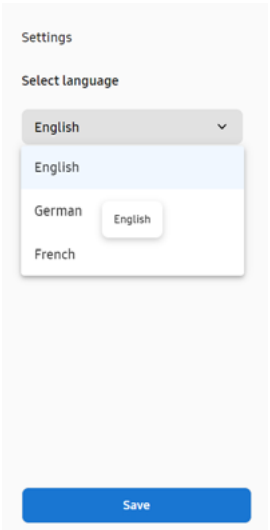


Figure 21

VXT PIRS Metadata.

3.6.3 UI Deep Dive

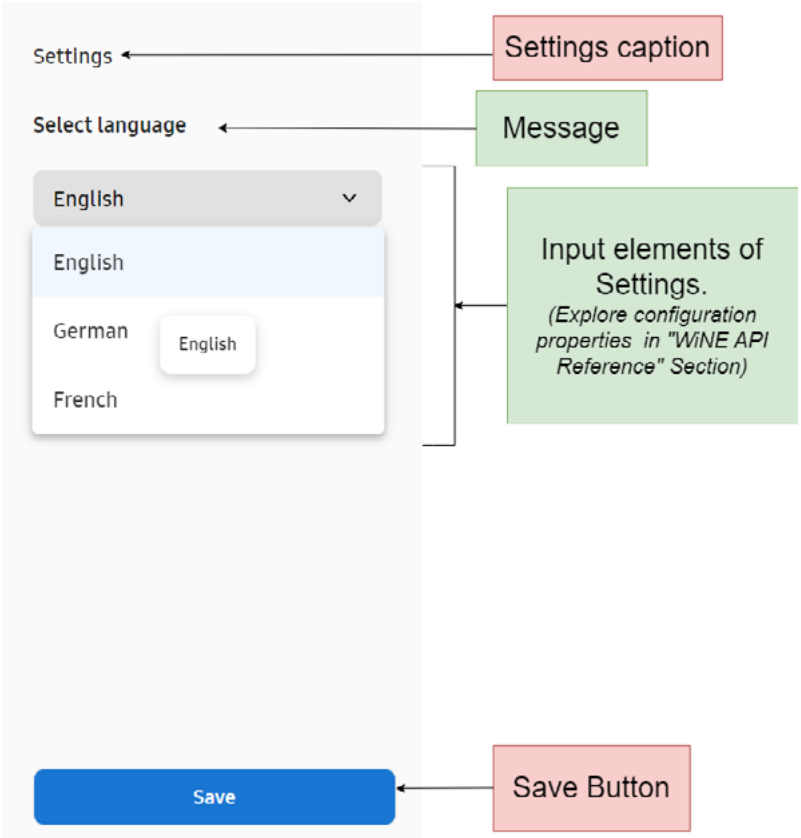
In order to fully understand Extension as a PIRS Component, its role and possibilities, we must know what parts it's made of. In order to prevent misunderstandings, we will divide the Elements into 3 sections:

- **Red section:** Represents VXT Canvas elements that we (as PIRS developers) cannot impact in any ways (but we can make our App respond to User's interactions with some of them).
- **Purple section:** Represents PIRS elements that must appear within Extension (are obligatory).
- **Green section:** Represents PIRS elements that we can pick and choose, mix and match (more on that later).

Since the elements of Settings will be the same for both Content-Driven and Data-Driven Apps, we will discuss them only once, in a wider context.

Figure 22

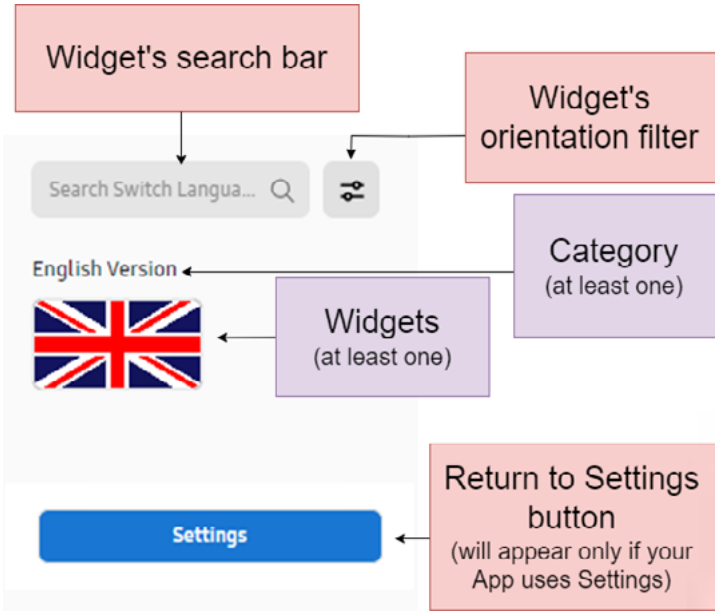
Elements of Settings.



For Content-Driven PIRS, the Extension Output consists of the following elements:

Figure 23

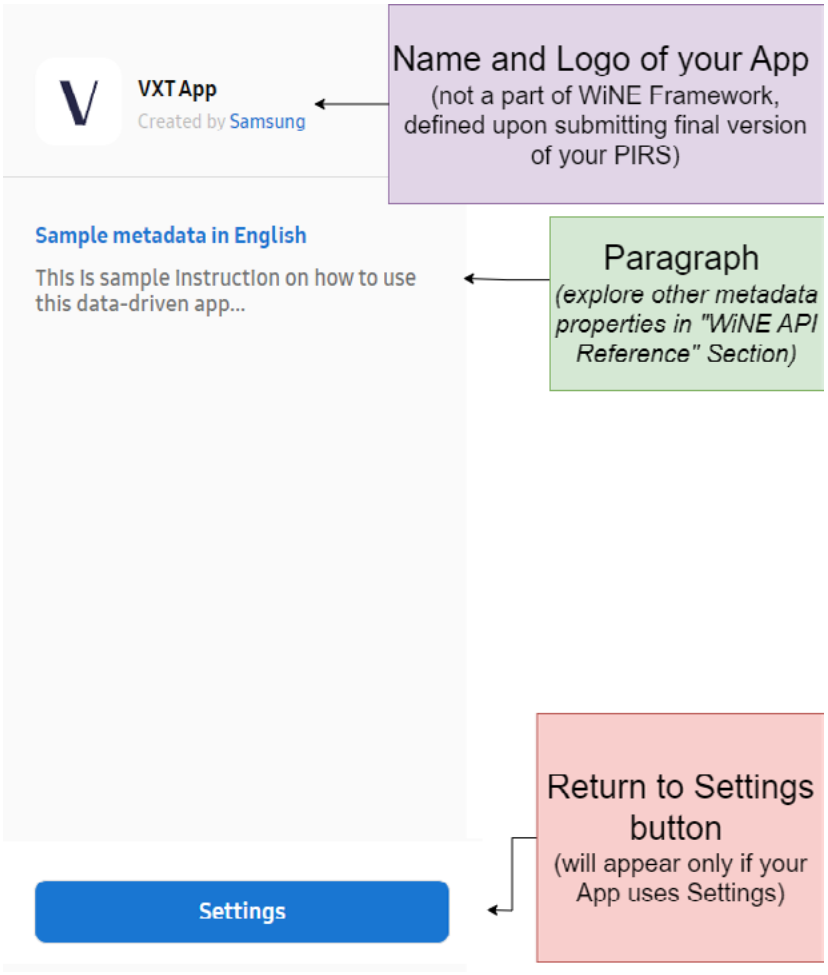
Elements of Extension Output.



For Data-Driven PIRS, we are working with Metadata, which is made of elements described below:

Figure 24

Elements of Metadata.



3.7 VXT Widget

Widget is the content (or, to be more precise, a web application) that is displayed first on canvas area in VXT Canvas, and then on the screen in VXT Player. It is launched (loaded onto canvas area) when User clicks on its graphical representation (thumbnail) in Extension (on the left sidebar). If an App features a Property Panel, then the Widget is subject to various modifications through elements of Property Panel made by the end User, until they are happy with the look and feel of the content. Then, the project is saved in VXT CMS and can be published to the screen running VXT Player. Of course, the project can be later modified and/or deleted. We can think of the Widget as an essence of VXT project created with the use of PIRS in VXT Canvas.

3.8 VXT Developer Portal

3.8.1 What is VXT Developer Portal

It is a place, where you can upload your Application, preview it in VXT Canvas to get an idea of how it looks and how the components work together (and if there is something to improve, you can edit it), initialize the Review process by sending your App to Review, as well as track its progress towards an official release.

3.8.2 Accessing the VXT Developer Portal

Once you've gained access to your VXT Workspace as a partner, head to the Samsung VXT Developer Portal (https://developer.samsungvx.com). If you haven't enrolled yet, kindly follow the steps described in VXT Developer Portal Enrollment Guide.

3.8.3 User Roles

The Samsung VXT Developer Portal assigns various distinct roles. Let's briefly discuss the ones that are the most important when it comes to PIRS development, testing and releasing:

- **DEVELOPER/REPRESENTATIVE DEVELOPER (hereinafter referred to as USER):** A third-party company looking to distribute their assets through VXT. As a user, you can upload, preview, edit, submit for review and request termination of your PIRS application (delete). User is essentially the App's owner.
- **OPERATOR:** A Samsung employee managing the release process of the aforementioned assets.

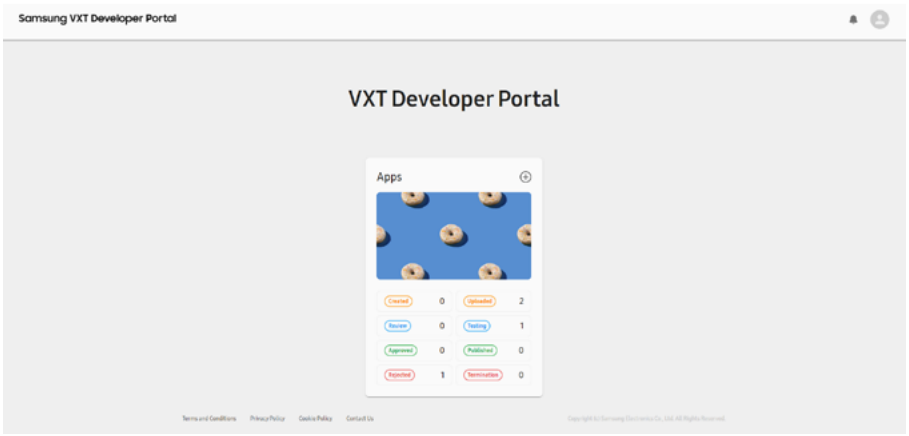
3.8.4 The Dashboard

Upon successful login to the VXT Developer Portal, you'll be greeted by the dashboard.

> NOTE: As VXT Developer Portal is continuously evolving, below pictures may slightly differ from the actual Portal.

Figure 25

VXT Developer Portal Dashboard.



In the center of the platform, you'll find a division showing the number of apps currently in various statuses. There are 8 states an application can be in:

- **CREATED:** The app has been created locally in the VXT Developer Portal but hasn't been uploaded to the server. Changes might be lost at this stage.
- **UPLOADED:** The app's source code has been uploaded to the VXT Developer Portal's server.
- **REVIEW:** The app's owner has submitted the App to Samsung Team to begin the process of checking its readiness for a release. Review is the initial stage of checking the App. User cannot modify it anymore, unless requested by Samsung Team.
- **TESTING:** The operator has confirmed that the app meets all necessary requirements and forwarded it to a tester for comprehensive quality verification.
- **APPROVED:** The tester has provided feedback indicating the Application is of sufficient quality to be published in VXT.

- **REJECTED:** The operator has identified issues that prevent the App from being included in VXT.
- **PUBLISHED:** The App is now available in VXT Marketplace for installation and purchase.
- **TERMINATION:** The App's owner has decided it no longer meets their needs and wants it removed from VXT. In order for it to happen, they click the "delete" button and thus a termination request is sent.
- **TERMINATED:** Operator terminates the App, which the termination request is related to. The App is no longer available neither in VXT Canvas, nor in VXT Developer Portal.

By clicking on each section, users are automatically redirected to Applications in the selected status.

Figure 26

VXT Apps Status (dashboard).

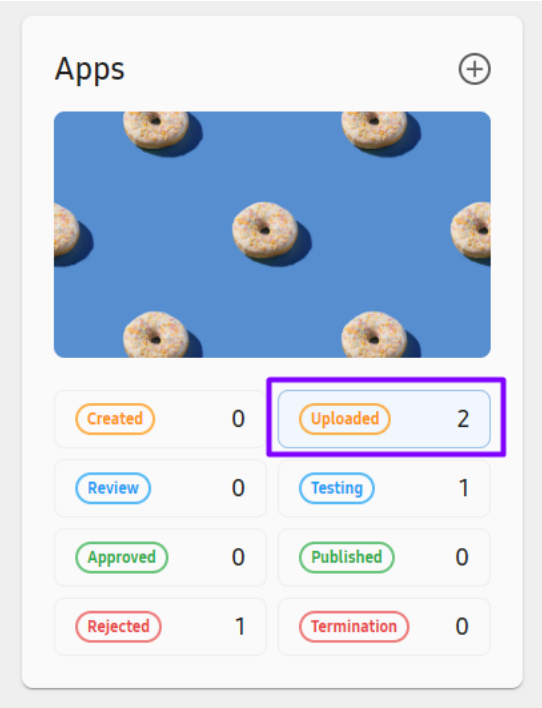
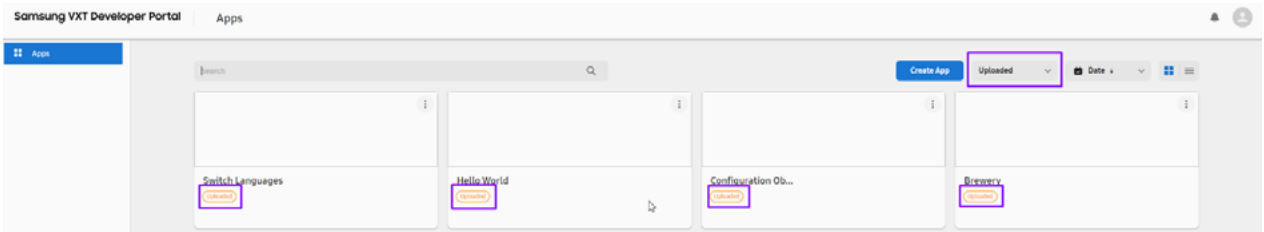
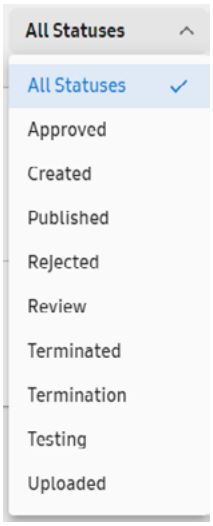


Figure 27 VXT Apps Status (apps menu).



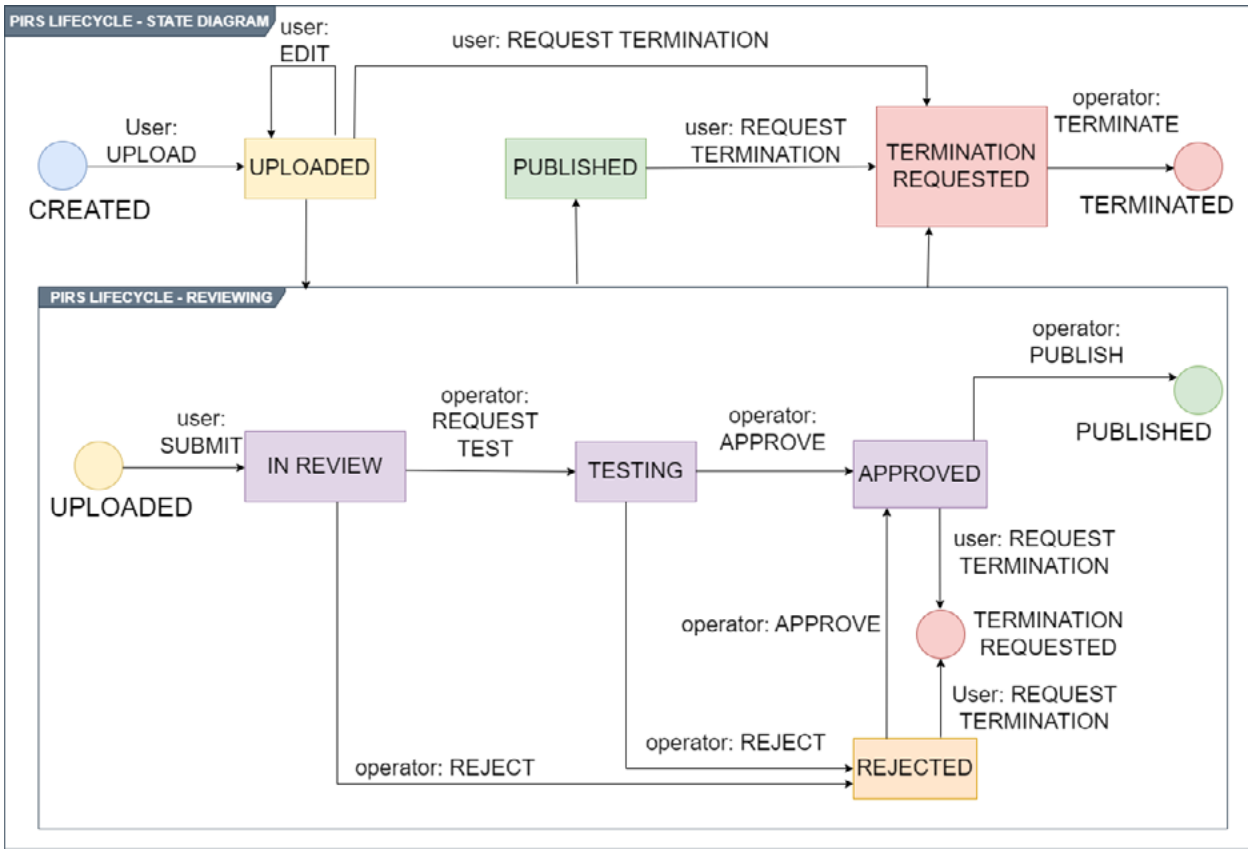
Users can then utilize the 'All Statuses' dropdown list in the top-left corner to change the status view.

Figure 28 Apps Statuses.



3.8.5 Application Lifecycle in the Portal

Figure 29 Application Lifecycle in VXT Developer Portal.



This diagram illustrates the lifecycle of an application within the Samsung VXT Developer Portal, divided into two main sections:

- **Upper section (PIRS LIFECYCLE - STATE DIAGRAM)**
  - Applications start in the **CREATED** state.
  - By completing the App upload process, Users can make their App move from **CREATED** to **UPLOADED** state.
  - Users can freely edit UPLOADED Applications.

- Users can request termination for **UPLOADED** applications by clicking the delete button.
- Users can request termination for **PUBLISHED** applications by clicking the delete button.
- Operators can move App entities from **TERMINATION REQUESTED** to **TERMINATED** state.

- **Lower section (PIRS LIFECYCLE - REVIEWING)**
  - Users submit **UPLOADED** App entities for review.
  - **IN REVIEW** applications can either be:
    - sent for testing by operators, or
    - ejected by operators.
  - Applications in **TESTING** state can either be:
    - Approved by operators, or
    - Rejected by operators.
  - For **APPROVED** applications:
    - Operators can publish them.
    - Users can request termination.

3.8.6 Key Points

- User Actions: Uploading, editing, submitting for review, and requesting termination.
- Users can also view their Apps in VXT Canvas.
- Operator Actions: Requesting tests, approving, rejecting, publishing, and terminating App entities.

3.9 VXT Player

3.9.1 Overview

Having understood the individual components that constitute a PIRS and the environment, which it is processed in (VXT Developer Portal) and the environment that it lives within (VXT Canvas), it is crucial to comprehend the role of VXT Player – the environment that the App's core component is displayed in, as a final result.

Understanding the role of VXT Player is no less important that comprehending the other concepts, as this is the place that your App will be visible not only to the people who have purchased and installed your App in VXT Marketplace, but also to the public – potentially thousands of people. Having fully embraced this idea, we can now see the importance of ensuring good integration of your App with VXT Player.

3.9.2 Functionality and Purpose

If you are a Samsung B2B SSSP/TEP Partner, you know the procedure of making your SSSP/TEP Applications run on a screen and that they would run directly within the Chromium-based Web Engine common to all Samsung Tizen 4.0 and above LFDs (with differing Web Engine versions, however). On the other hand, PIRS, once they are published to selected screens, they operate through the VXT Player, so another layer comes into play (Chromium Web Engine – VXT Player – your PIRS Application).

Leveraging VXT Player, compatibility across different operating systems, including non-Samsung screens is achieved. VXT Player supports the following operating systems to ensure a seamless user experience:

- **Tizen:** Version 4.0 and above
- **Android:** Version 10.0 and above
- **Windows:** Versions 10, 11, IoT (64-bit), with the following specifications:
  - Processor: Intel i5 and above
  - Memory: 8 GB or more
  - Storage: 256 GB SSD or more
  - Display: 4K (FHD, UHD) support

... and we are committed to expanding this list in the future based on business demands.

3.9.3 Virtual Screen

VXT also introduces a concept of Virtual Screen, which runs in your web browser. It is a good, temporary alternative for showcasing purposes, or as a testing tool. It is a great solution if you don't have a physical screen – with Virtual Screen you can display your content anyway!

4. PIRS Design Guidelines

Making sure your App adheres to VXT Design Standards

4.1 Overview

Every PIRS Application should adhere to VXT Design Standards for seamless integration into the VXT environment. Apart from source code (Extension and Widget), you need to provide the following Components:

- App Name
- App Icons (2)
- App Cover Logo
- App Promotion Image (1-5)
- App Description Title
- App Description
- Content Provider Name

These PIRS Components should be considered in a wider context of VXT Canvas environment – namely, the following sections:

- VXT Menu Bar
- VXT Apps Menu (Marketplace)
- VXT App Details Page

Let's go over these sections one by one and understand the guidelines for each PIRS Component.

4.2 VXT Menu Bar

Figure 30

VXT Menu - icons, name.

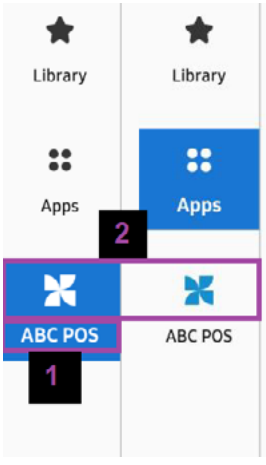


Table 4

VXT Menu - App Name Specifications.

1. App Name	
Description	VXT Menu Bar will display Application Name with a font size of 12px.
Guidelines	Up to 6 characters are allowed (based on English capital 'W').
Extra Information	If Application Name exceeds the aforementioned limit, ellipsis ('...') will be added to the end of the title.

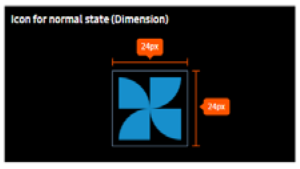
Table 5

VXT Menu - App Icons Specifications.

2. App Icons	
Description	These icons will be displayed in VXT Menu Bar after your App is installed.
Guidelines	It is required that you create two icons – one for normal state, and one for "selected" state.
Extra Information	Simplify icons for better clarity and legibility. Make icons graphic and bold.

File Name	Icon_AppName_Normal.png
Ratio	1:1
Dimension	24 x 24px
Safe Zone	20 x 20px
Background Transparency	100%
File Type	.png
File Size	Max 500 KB

File Name	Icon_AppName_Selected.png
Ratio	1:1
Dimension	24 x 24px
Safe Zone	20 x 20px
Background Transparency	100%
File Type	.png
File Size	Max 500 KB



4.3 VXT Apps Menu (Marketplace)

Figure 31

VXT Apps Menu - logo, promotion image.

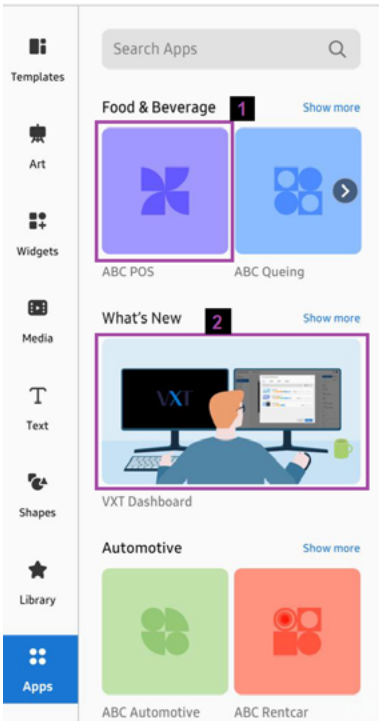


Table 6

VXT Apps Menu – App Logo Specifications.

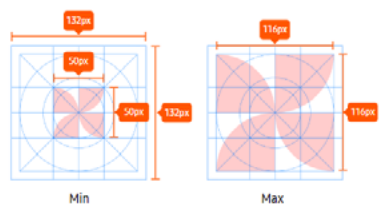
1. Application Logo	
Description	VXT Apps Menu displays various Apps with their logos, grouped into categories to provide easy browsing experience.
Guidelines	It is required that you create two icons – one for normal state, and one for "selected" state. **File Name:** Logo_AppName.png (or .jpg) **Ratio:** 1:1 **Dimension:** 132 x 132px **Safe Zone:** Min : 50 x 50px, Max : 116 x 116px **Background Transparency:** 0% **Border:** None **File Type:** .png or .jpg **File Size:** Max 500 KB 
Extra Information	Once original logo has been provided, VXT will displays in a round square after masking. 

Table 7

VXT Apps Menu - Promotion Image Specifications.

2. Promotion Image	
Description	VXT Apps Menu displays a promotion image instead of App Logo to promote special Apps.
Guidelines	Please refer to PROMOTION IMAGES table in VXT App Details Page below.
Extra Information	Please refer to PROMOTION IMAGES table in VXT App Details Page below.

4.4 VXT App Details Page

Figure 32

VXT App Details Page - logo, name, content provider, promotion images, description title, description.

Table 8

VXT App Details Page - App Logo Specifications.

1. App Logo	
Description	VXT App Details Page will display App logo with its size automatically reduced to 56 x 56px.
Guidelines	You don't have to create another size of App logo.
Extra Information	-

Table 9

VXT App Details Page - App Name Specifications.

2. App Name	
Description	VXT App Details Page will display App Name with a font size of 14px.
Guidelines	Up to 16 characters are allowed (based on English capital 'W').
Extra Information	If App Name exceeds the aforementioned limit, ellipsis ('...') will be added to the end of the name.

Table 10

VXT App Details Page - Content Provider Specifications.

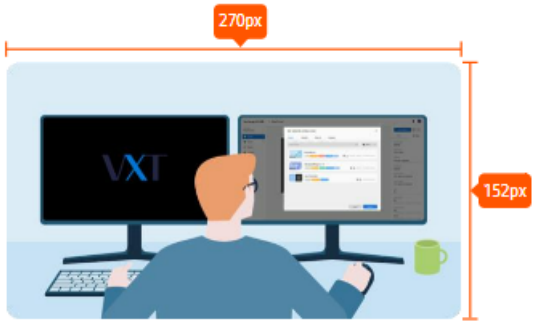
3. Content Provider Name	
Description	VXT App Details Page will display Content Provider name with a font size of 12px.
Guidelines	Up to 13 characters are allowed (based on English capital 'W').
Extra Information	If Content Provider Name exceeds the aforementioned limit, ellipsis ('...') will be added to the end of the name.

Table 11

VXT App Details Page - Promotion Images Specifications.

4. Promotion Image	
Description	VXT App Details Page can display multiple Promotion Images in form of carousel.
Guidelines	

File Name	Promotion01_AppName.png (or .jpg)
Ratio	16:9
Dimension	270 x 152px
Background Transparency	0%
Border	None
File Type	.png or .jpg
File Size	Max 500 KB
Minimum Font Size	10px



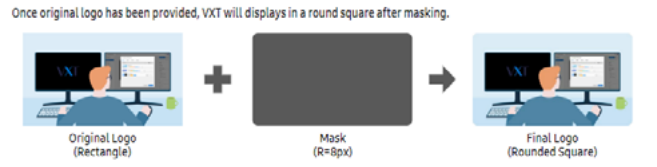
File name must be numbered as below:
- Promotion01_AppName.png (or .jpg)
- Promotion02_AppName.png (or .jpg)
- Promotion03_AppName.png (or .jpg)
- ...

You can upload up to 5 Promotion Images.

Extra Information

The first Promotion Image will appear in Apps Menu to promote the App.

Once original Promotion Image has been provided, VXT will displays in a round square after masking.



NOTE: Make sure to provide copyright-free images only.

Table 12

VXT App Details Page - Description Title Specifications.

5. Description Title	
Description	VXT App Details Page will display Description Title with a font size of 20px.
Guidelines	Up to 30 characters in two lines are allowed (based on English capital 'W').
Extra Information	Description Title is optional. If there's no Description Title provided, the Description body will take its place. If Description Title exceeds the aforementioned limit, ellipsis ('...') will be added to the end of the title.

Table 13

VXT App Details Page - Description Specifications.

6. Description	
Description	VXT App Details Page will display Application Description with a font size of 14px.
Guidelines	If Description Title is provided: Up to 300 characters are allowed (based on English capital 'W'). If no Description Title is provided: Up to 360 characters are allowed (based on English capital 'W').
Extra Information	-

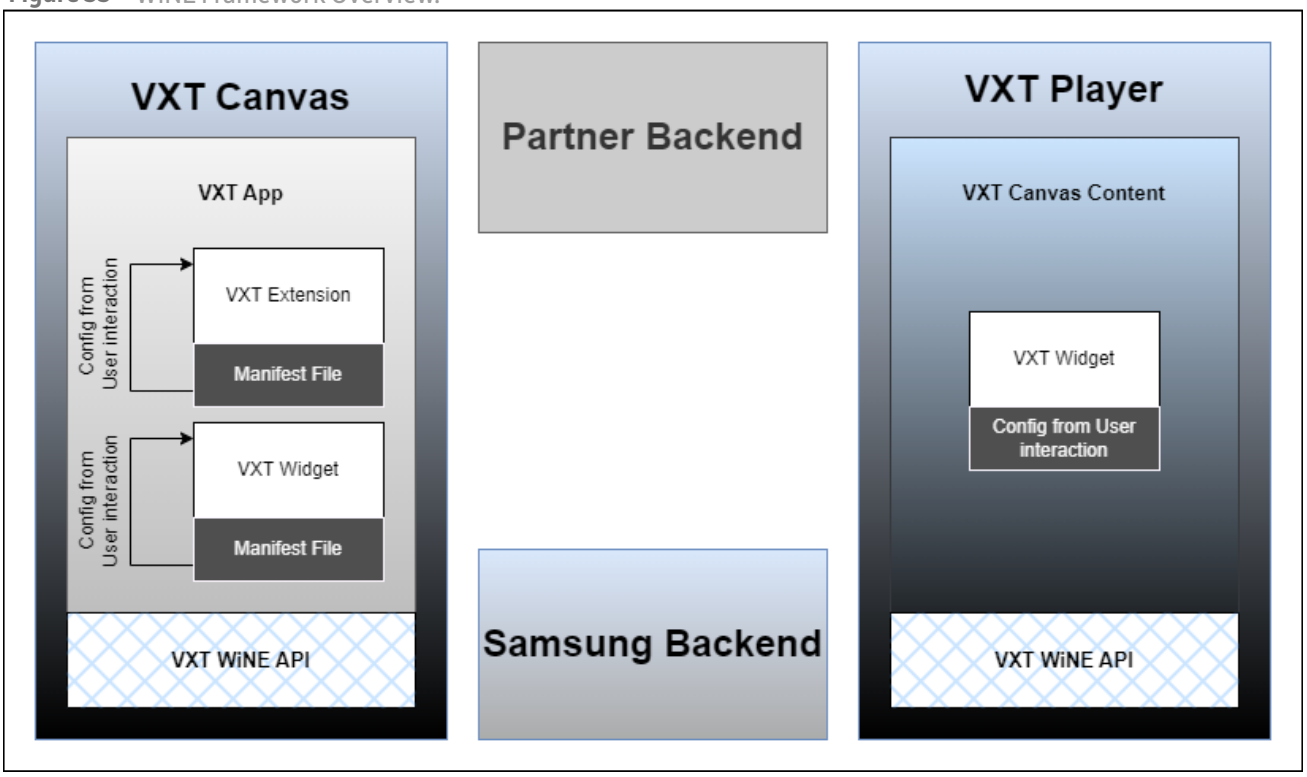
5. Understanding the WiNE Framework & WiNE API

Understanding the various Elements and how PIRS interacts with them

5.1 Components That Make Up the WiNE Framework

Samsung VXT CMS introduces innovative ways to establish an Environment where PIRS Applications operate and communicate. At the heart of this development lies the WiNE Framework (Widget aNd Extension), and this is where all its concepts and components can be mapped. To gain a comprehensive understanding of the WiNE Framework and PIRS components, refer to the diagram below.

Figure 33 WiNE Framework Overview.



The WiNE Framework is composed of four main sections: VXT Canvas, VXT Player, Samsung VXT Backend and (optionally) Partner Backend. To have a better understanding, it is recommended to view the above in blocks viewed from left to right.

- VXT Canvas is where User can configure their VXT Canvas Content (and begin working with any VXT Application), therefore can decide, what exactly they want to display on Screen.

- Both PIRS, as well as the VXT Canvas Content itself, are hosted in Samsung VXT Backend.
- Any PIRS might require its own Backend, too (it's up to each PIRS Developer).
- Once User has completed configuration of their VXT Canvas Content, it's time to publish it to a Screen - this is where VXT Player takes control.

The WiNE API remains a common denominator, both before and after a PIRS is published a Screen running VXT Player, for it controls flow (communication) of data between the individual Application Components and VXT (namely VXT Canvas and VXT Player).

You might have noticed that VXT Extension (alongside with its own Manifest) is not exported over to VXT Player. This PIRS Component lives only within the environment of VXT Canvas. What is sent to VXT Player, upon publishing the Content to a Screen, is the Widget itself. Its "config from user interaction" part allows for editing published Content in VXT Canvas and seeing results on the Screen nearly immediately after saving the changes.

> NOTE: A PIRS displayed on a Screen via VXT Player will always feature a Widget, regardless of whether it's of Data-Driven or Content-Driven type. A Data-Driven PIRS will feature VXT built-in Widget (namely, a POS-ready Template or POS-ready Widget), whereas a Content-Driven PIRS will feature its own, custom-made Widget.

All individual VXT App Components (Extension, Widget, Manifests) will be discussed in detail in the following Chapters.

5.2 What is WiNE API?

As we have explained in the previous Chapter, WiNE API is what makes the communication between VXT and PIRS possible, as all communication between your PIRS Application's Components (Settings and Extension, Extension and Widget, Property Panel and Widget) must go through VXT Canvas. This is made possible and is performed using a Channel.

5.2.1 Opening a Channel

Before any data may be received or sent from your Application, the Channel itself must be opened – both in Extension's index.js and Widget's index.js file. Opening a Channel is akin to creating an instance of a Channel object – we call a function named createChannel() and pass a unique Channel Id. It is useful to save the outcome inside a variable, giving it a descriptive name, as presented in an example below:

```
channel = $vxt.createChannel($vxtSubChannelId);
```

5.2.2 Publishing and Subscribing

Once a Channel is opened and a handle to it is available, the Channel object has two methods: subscribe (receive data from VXT Canvas) and publish (send data to VXT Canvas). In the examples below, we have wrapped "event" and "response" inside angle brackets to indicate that those are just mock variables for now. We will dive deeper into Events and Responses later.

```
channel.subscribe( " <event> ", ( <response> ) => { //do something });
channel.publish( " <event> ", { <response> });
```

Table 14 Publish&Subscribe Overview.

Publish & Subscribe	
How does subscribing work?	VXT Canvas is responsible for listening for changes and giving you information about them. Your Application is responsible for reacting to it. Every time User makes a change, you will get an Event with some payload. It is possible only if you're expecting this particular information (subscribing to it). It's then up to the Application to decide, what to do with it.
How does publishing work?	You can publish something to make it display in VXT Canvas. More information about what exactly can be published and how to do it can be found later in this Guide.

The most important concept to grasp when it comes to WiNE API is that very little happens here automatically. The communication is mostly based on information exchange. VXT Canvas gives us some information (namely about User's selections in Settings, User's changes of Orientation, User's Search Queries and changes in Property Panel) and it's up to us to decide later what to do with given information. We can, for example: manipulate the look and feel our Widget, save some information in our Database, or make something appear in VXT Canvas' UI using the .publish method, for example: display a list of available Widgets, display Metadata, post a toast message, or change our Property Panel. What we can publish is strictly defined and this is the same for what information we can receive from VXT via .subscribe method,

and this is what we're going to discuss in great detail in Chapter "Developing PIRS". For now, it is important to mention that some Events can be published both by Extension and Widget (), and some can be published only by Widget (). The same applies to subscribing – Extension can subscribe to a "get" event only, whereas Widget can subscribe to such Events as , and . A complete list of Events can be found in "WiNE API Reference" Section at the end of the Guide.

5.2.3 Closing a Channel

When you don't need Channel anymore (for example, when User discards the project), it should be closed to prevent memory leaks.

```
channel.close()
```

5.3 What is a Manifest (JSON) File?

5.3.1 Overview

So far, we've only discussed the JavaScript logic that is possible to implement through WiNE API, but there is another WiNE API element to discuss – the Manifest File.

Manifest Files define what is going to be displayed as PIRS Settings (used for customizing User Experience within the Application), as well as PIRS Property Panel (used for customizing the look and feel of selected Widget). As you already learned from the "Understanding Key VXT Concepts", PIRS Settings are located within VXT Sidebar to the left, whereas Property Panel is located on the right side of VXT Canvas. Both of these can play crucial role in defining the end result, which will be displayed on Screen, however they are not mandatory – an App can serve its purpose even without Settings or Property Panel. Having the above in mind, while PIRS Settings and PIRS custom Property Panel are not obligatory, the Manifest Files are, so make sure to include them inside your App packages (you can find more information about App packages in "Developing PIRS" Section).

A Manifest for PIRS Settings is called "Extension's Manifest", whereas a Manifest for PIRS Property Panel is called "Widget's Manifest". Please note, that for data-driven type of Application, only Extension's Manifest can exist, as it doesn't feature its own Widget, but uses VXT built-in Template/Widget with a built-in Property Panel designed specifically for Data-Driven PIRS. From now on, please bear in mind that "Extension's Manifest" will be discussed in the same way for both Data-Driven and Content-Criven PIRS, and the information about "Widget's Manifest" will be applicable to content-driven Applications only.

The elements of both Settings and Property Panel can be defined using configuration objects. Each object features a "type" property, which specifies, what kind of element will be drawn in the UI (for example, a text input box, a dropdown, or a toggle). A complete list of configuration objects (with examples) can be found in "WiNE API Reference" Section.

One last thing that applies to both Extension's Manifest and Widget's Manifest is that they are both JSON files, so a JSON format is the only format required and understood by VXT Canvas.

> NOTE: Configuration objects missing any mandatory properties will not be rendered in the UI, and an error will be thrown to the console.

5.3.2 Extension's Manifest

As we have already mentioned, Extension's Manifest File describes what will be displayed as PIRS Settings. These are the first thing that User sees, having installed and opened your PIRS, and this is where you can give them an option to customize their User experience of your App – for example, select their preferred language. You will be able to detect their selection later in your code and adjust your Application accordingly (more on how to do it in "Developing PIRS" Section).

An important thing to note is that PIRS Settings can be defined only once and cannot be dynamically changed in any way.

An Extension's Manifest File is made of an array of configuration objects:

```
[
  {
    //VXT configuration object 1
  },
  {
    //VXT configuration object 2
  },
  {
    //VXT configuration object n
  }
]
```

A complete list of configuration objects (with examples) can be found in "WiNE API Reference" Section.

5.3.3 Widget's Manifest

Widget's Manifest File describes what will be displayed as PIRS custom Property Panel. These are used for configuring the Widget and deciding, what the end result will look like on a Screen. For example, you can give your User a possibility to change the text color or select their company logo. You will be able to detect their selection later in your code and adjust your Widget's look accordingly (more on how to do it in "Developing PIRS" Section).

An important thing to note is that PIRS Property Panel (unlike PIRS Settings) can be dynamically updated in your code. Another information to grasp is that Widget's Manifest is not the only way of defining Property Panel – it is also possible (and sometimes advised) to do it from within Extension – more information on that can be found in "Developing PIRS" Section.

A Widget's Manifest File is an object, which takes Property Panel's Tab Names as properties. Each of these properties has an array of configuration objects inside (similar to Settings):

```
{
  " <Property Panel's Tab Name 1> ": [
    {
      //VXT configuration object 1
    },
    {
      //VXT configuration object 2
    },
    {
      //VXT configuration object n
    }
  ],
  " <Property Panel's Tab Name 2> ": [
    {
      //VXT configuration object 1
    },
    {
      //VXT configuration object 2
    },
    {
      //VXT configuration object n
    }
  ],
  " <Property Panel's Tab Name n> ": [
    {
      //VXT configuration object 1
```

```
    },
    {
      //VXT configuration object 2
    },
    {
      //VXT configuration object n
    }
  ]
}
```

An important thing to note is that tab names are displayed in the UI in alphabetical order.

5.4 WiNE Data Endpoints

The last feature of WiNE API is the concept of WiNE Data Endpoints (REST API). As the name may suggest, these are utilized solely by Data-Driven PIRS to make the data (stored and managed by various data services) available to VXT – so that it can later populate built-in, POS-ready Templates and Widgets with retrieved data. The correct use of WiNE Data Endpoints is the most crucial part of any data-driven App. VXT Canvas and VXT Player will call these Endpoints on demand, depending on the User's action.

From the above information we can conclude that Partner's Data-Driven PIRS becomes a Data Provider component, which gathers, manipulates and provides data to VXT and relies heavily on its backend service.

The schema looks as below...

```
<BaseURL>/<Endpoint>
```

...where "Base URL" stands for Data Provider's server, which implements specified REST API Endpoints for VXT to call. It is, essentially, a backend service host URL of Data Provider.

A complete WiNE REST API Data Enpoint list (with examples) can be found in "WiNE API Reference" Section.

A simplified flow of calling the endpoints can be described in the following way:

Figure 34 Data-Driven App's Communication Flow - Metadata.

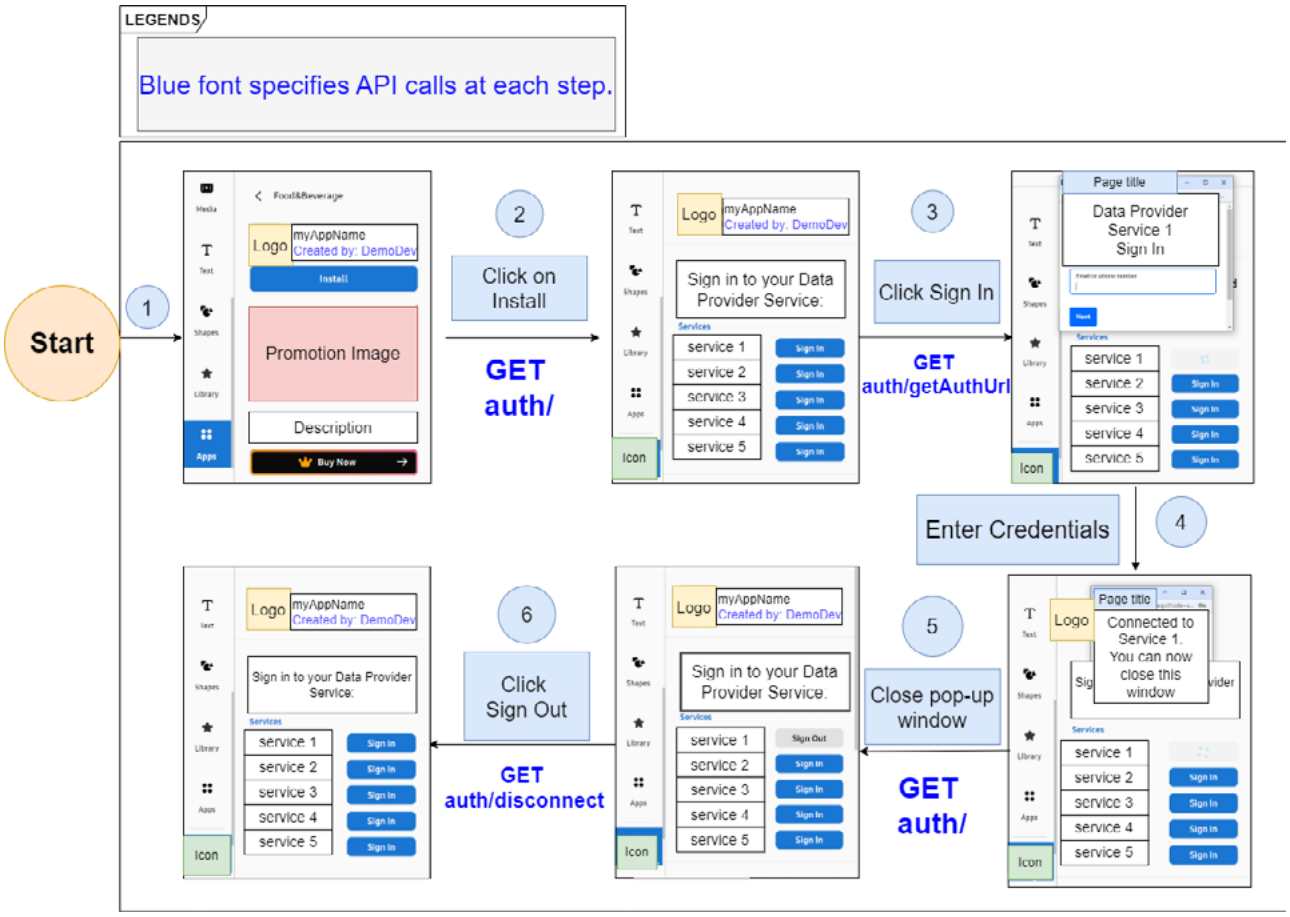


Table 15

Data-Driven App's Communication Flow - Metadata.

Step	API & Response	Additional Information

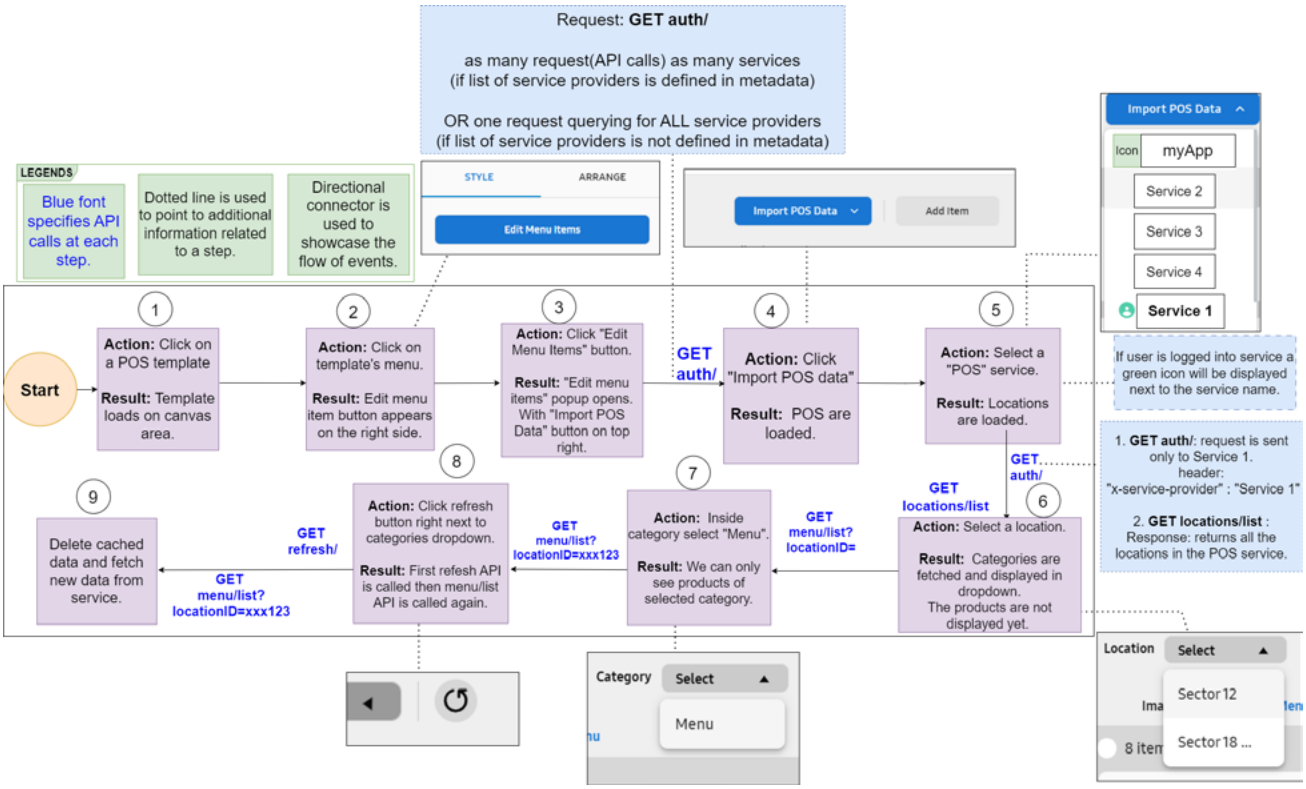
2	1. Request: GET auth
	
	2. Response:
	{
	"posType": "Service 1",
	"status": "Not Connected",
	"message": "Need to sign into Service 1"
	}
		Get: auth/
		
		1. as many requests (API calls) as many services (if list of service providers is defined in metadata)
		
		2. OR one request querying for ALL service providers (if list of service providers is not defined in metadata)
		
		In auth/ request default is "ALL" and all services are expected to be returned.
		
		HEADERS:
		
		1.'x-api-key' (Required) : API key. Generated from VXT Canvas.
		
		2. 'x-service-provider' (Required): Specific POS service name.

3	1. Request: GET auth/getAuthUrl
	
	2. Response:
	{
	"posType": "Service 1",
	"targetUrl": "https://connect.service1/authorize?client_id=xyz"
	}
		UI Result: A popup window opens with specified URL.

5	1. Request: GET auth/
	
	2. Response:
	{
	"posType": "Service 1",
	"status": "Connected",
	"message": "Already connected to Service 1"
	}
		UI Result: Button changes from "Sign In" to "Sign Out".
		
		After closing the pop-up, auth/ is sent again but only once (with this specific service provider included in request headers)

6	1. Request: GET auth/disconnect
	
	2. Response:
	{
	"posType": "Service 1",
	"status": "Disconnected",
	"message": "Successfully signed out"
	}
		UI Result: Button changes from "Sign Out" to "Sign In".

Figure 35 Data-Driven Apps Communication Flow - Edit Menu Items.



6. Developing PIRS

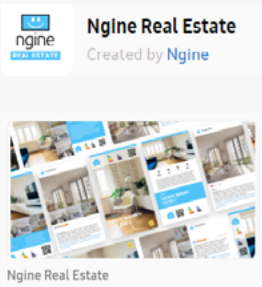
Developing your very first VXT Application

6.1 Types of PIRS

Throughout this Guide we have been mentioning two types of PIRS: Data-Driven and Content-Driven. Before we start developing PIRS, it is important that we summarize and remember the key differences and features of both of these types.

Table 16

Types of PIRS.

Types of PIRS				
Type	Description	Example	Keywords	Type-Specific Chapters
data -driven	This kind of application interacts with VXT built-in templates by loading data from external sources into them. It utilizes Metadata to guide User and display instructions how to use it. Uses a predefined list of REST API Data Endpoints, which will be called by VXT Canvas to obtain data from data sources. Currently supports only menu boards.	Link My POS by Wisar Digital 	Extension Settings Sidebar Metadata Data connector Data Endpoints REST API	WiNE Data Endpoints (data-driven only)
content -driven	This kind of application provides visual content through its own, custom-made Widget, with optional configuration options via Property Panel. It provides a list of Widgets grouped into categories.	Ngine Real Estate by Ngine  Ngine Real Estate	Extension Settings Sidebar Categories Content Widget Property Panel	Writing your first VXT Widget (content-driven only) Creating your first VXT Widget Manifest (JSON) File (content-driven only) Using SSSP or TEP API's in your PIRS (content-driven only)

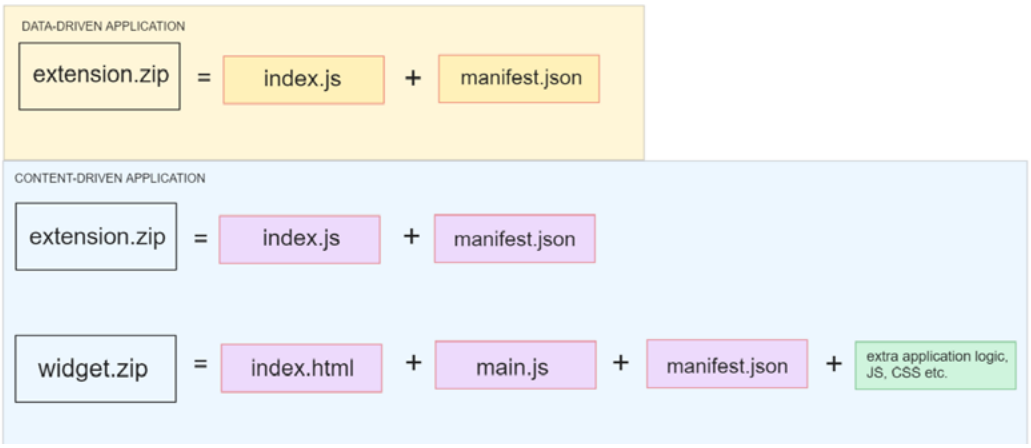
Application development process will differ immensely depending on what type of Application you decide to develop.

6.2 What Makes Up a PIRS

When it comes to uploading our PIRS to VXT Canvas through VXT Developer Portal, you are expected to provide the source files packed into .zip archives. The amount of .zip files will differ according to the Application type (Data-Driven or Content-Driven), which is presented on the schema below:

Figure 36

Zip Archives Composure.



There are three important things to bear in mind when creating the aforementioned .zip archives:

- All above files (besides extra application logic, JS, CSS, etc.) are mandatory.
- Make sure there is nothing between .zip file and source files (they should be placed in the root of the archive – meaning, that the files themselves are zipped, not their parent folder, if there's any).
- The file naming schema should be strictly observed:
  - For Extension: index.js and manifest.json.
  - For Widget: index.html and manifest.json.

> NOTE: Avoid naming any file inside widget.zip "index.js", as unexpected behavior may occur.

You may already know that a PIRS is not only its source code (which is very important from a technical point of view), but also other assets (which are

very important from the marketing point of view). To sum it up, a complete Application package includes:

- **Source code:** extension.zip, widget.zip.
- **Images:** cover logo, icon for selection, icon for normal state, promotion image(s).
- **Text:** name, description title, description, content-provider.

6.3 Writing Your First VXT Extension (data & content-driven)

6.3.1 Overview

Since we already know the communication basics of WiNE API and are familiar with PIRS Types, we can start diving deeper into creating our first PIRS. We'll start from Extension (the leftmost component) and move our way gradually to the right, towards Property Panel.

Extension is a whole new concept when it comes to app development, unlike the Widget Component, which is a web application with extra spice added by WiNE API. Extension, on the other hand, is something that's mostly build with WiNE API. This Component is inherently and tightly connected with VXT Canvas and it is the only place it can live. Since Extension's composure is strictly defined by WiNE API, it is crucial to take your time to fully understand this Component. We already know from the previous Chapter, that this is where we select our Widget that we would like to see on canvas area (and later on a screen). But how do we make the Widgets appear in the sidebar in the first place? How do we define their categories? How can we customize User experience and display only the Widgets that are tailored to their needs, while hiding the ones that are not useful at the moment? Or – in data-driven applications – how do we display Metadata in the language selected by the User?

These are the questions that we'll provide answers for in this Chapter, but we'll not limit ourselves to that – prepare for a deep dive into Extension!

As we have discussed, WiNE API requires that all communication between VXT and your Application goes through a Channel. Once a Channel is opened, it implements two methods: one for sending information to VXT (.publish), and one for receiving information (.subscribe).

From a birdseye perspective, an Extension is built in the following way:

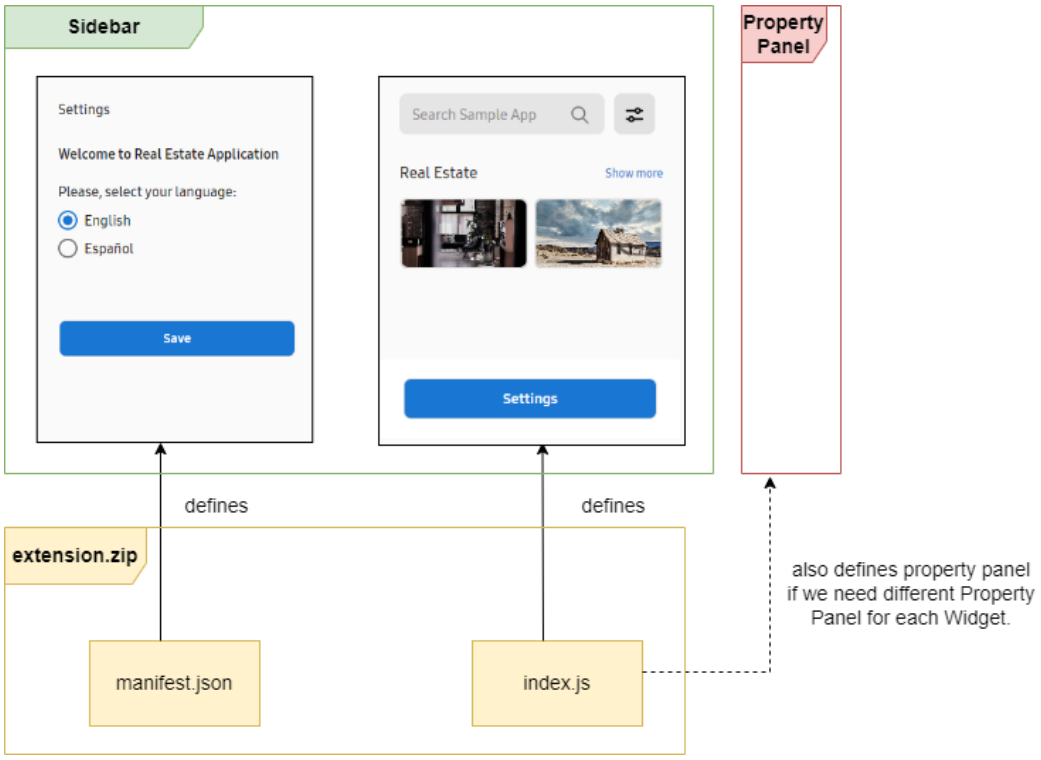
- Open a channel
- Subscribe (listen for):
  - In data-driven and content-driven Application: User interactions in Settings
  - In content-driven Application: Widget orientation changes and keywords searches
- Publish:
  - In data-driven Application: metadata
  - In content-driven Application: list of Widgets and their categories (array of content objects and array of categories)

But before we start our development, we need to understand which file controls which element of the UI.

In content-driven application, the schema can be presented as follows:

Figure 37

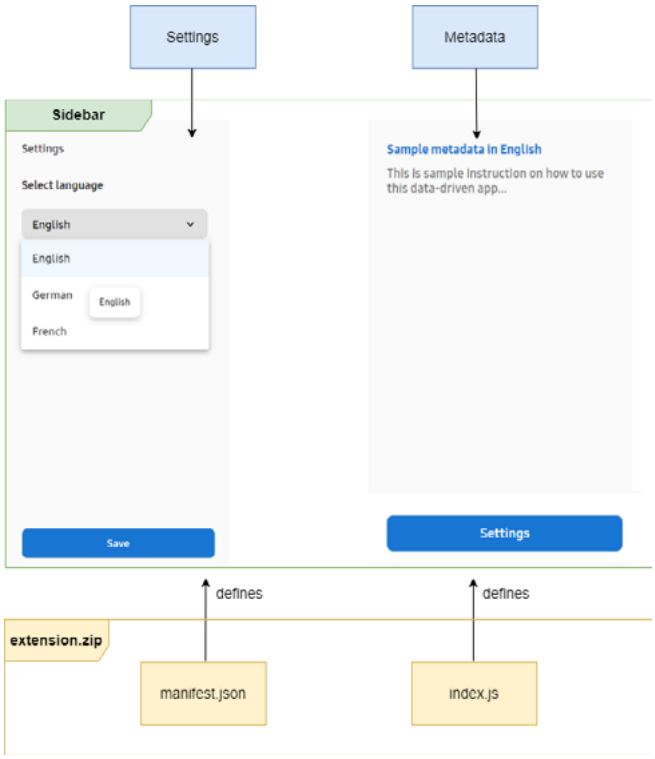
Extension Files (content-driven).



For data-driven application the schema can be presented as follows:

Figure 38

Extension Files (data-driven).



In this Chapter, we'll focus on the content of index.js file.

6.3.2 Events

Let's investigate deeper, how to communicate with VXT using WiNE API in our Extension, and what information we can receive:

Table 17

Basics of Subscribe (extension).

Subscribe - Basics		
Event	Description	How to subscribe?

get	This Event fires when User clicks "Save" in Settings, or (if an Application doesn't use them), when the App is opened.
	
	It can be thought of as an initial Event.
	
	It also fires whenever User types in something in the search bar at the top, or switches orientation using the Widget orientation filter on the left side of the search bar.
		`channel.subscribe("get", (response) => {`
		`//do something with received response`
		`});`

Table 18

Subscribe - Deep Dive (extension).

Subscribe - Deep Dive			
Event	Response types	Description of response	Full response - example

get	contents
	
	metadata
		"type": //VXT Canvas requests "contents" (in case of content-driven Apps) or "metadata" (in case of data-driven Apps)
		
		"payload":
		{
		"searchQuery": //whatever has been typed into the search bar at the top of the Sidebar
		
		"orientation": //widget orientation selected by User (default is "landscape", can be switched to "portrait")
		
		"id": //unique identifier tied to User's VXT Workspace
		
		// Settings selections – written in "id": "value" format
		
		"subscription": information about VXT subscription
			<pre>{
			"type":"contents",
			"payload":
			{
			"searchQuery":"",
			"orientation":"landscape",
			"id":"vxt-...",
			"settingsLang":"English",
			"subscription":
			{
			"requester":"canvas",
			"status":"trial"
			}
			}
			}</pre>

Table 19

Basics of Publish (extension).

Publish - Basics			
Event	How to publish?	Response type	Response payload

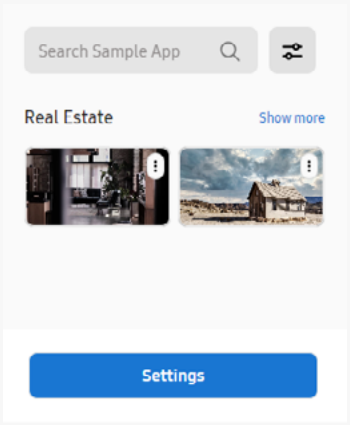
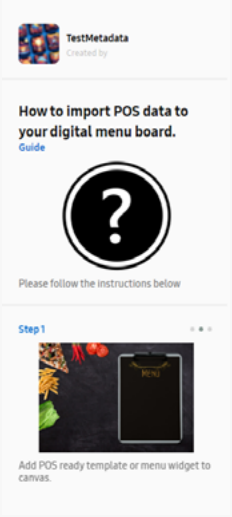
get	<pre>channel.publish("data",{
	type: " <response type> ",
	payload: {
	//payload required for specific response type
	}
	});</pre>
		contents	An array of content objects and an array of categories (strings/objects).

		metadata	An array of metadata objects.

Table 20

Publish - Deep Dive (extension).

Publish - Deep Dive		
Event	How to publish?	Response type

get	channel.publish("data", {
	type: "contents",
	payload: {
	categories: myCategories,
	content: myContent
	}
	});
	
	channel.publish("data", {
	type: "metadata",
	payload: myMetadata
	});
		
		Real Estate" is the category name, defined in myCategories variable.
		
		The rectangles below are graphical representations of content-objects (Widgets) and, in this example, are defined in myContent variable.
		 

More information on how to define content (myContent), categories (myCategories) can be found in the sections below – "Content and Categories (content-driven only)".

A complete metadata (myMetadata) properties reference can be found in Chapter "WiNE API Reference".

6.3.3 Content and Categories (content-driven only)

Take your time to familiarise yourself with the following informations when it comes to Content (Widgets) as it is a key Component of a Content-Driven PIRS. Content objects are nothing less than graphical representation of Widgets available in your Application. We can think of Content as a part of code, whereas Widgets would be a part of UI. You may remember from previous Chapters that there is only one index.html, but there are multiple customization options thereof, and these are passed to VXT as Content.

Content

Content is a configuration object that consists of key – value pairs. Every single object is then later rendered as a single Widget (or, to be more specific – its thumbnail). More information on how to define Content objects is presented in the Tables below:

Table 21

Content Object - Description.

Content Object			
Key	Description	Supported Values	Example
resourceType	Type of resource – information to VXT whether it should display a content-object as a Widget, or as a Folder.	string Options: • "content" • "folder"	resourceType: "CONTENT" resourceType: "FOLDER"
resourceId	An identifier of a particular Widget.	string	resourceId: "myWidget-1"
resourceName	Text shown on mouse hover over the Widget.	string	resourceName: "Demo"
resourceSize	Optional parameter for future WiNE API development.	string	resourceSize: ""
thumbnailPath	A path to resource (picture) that you wish to use as Widget Thumbnail - note that it itself doesn't determine what is going to be displayed as Widget inside Canvas Area.	string Should be an absolute path to resource hosted on an external server.	thumbnailPath: "https://example.com/picture.jpg"
isLandscape	Used for filtering Content by Orientation.	boolean	isLandscape: true

Content Object			
category	This field defines, in which Category the Content object will appear in UI.	string Should be an existing Category.	`category: "Real Estate"`
orientation	Used for filtering Content by Orientation.	string Options: • “landscape” • “portrait”	`orientation: "landscape"`
viewEdit	Used for filtering Content by Orientation.	Object which implements a “message” property, which takes string as a value. This value will be displayed as an instruction for User once the interactivity is enabled.	`viewEdit: { message: "Click the button to display extra information" }`
allowOutOfCanvas	Defines, whether Widget should be allowed to be moved out of canvas area.	boolean	`allowOutOfCanvas: false`
keywords	Used for filtering Content by Keywords through Search Bar.	Used for filtering Content by Keywords through Search Bar. string	`keywords: "demo, english, inside"`
config	Object with an array of configuration objects.	Object with an array of configuration objects.	`config: { Style: [ { type: "message", id: "myMessage", value: "Style your Widget" }, ... ] }`
children	Property handled only when resourceType is “folder”. Defines, what’s inside the folder.	An array of content objects or folder objects.	`Children: [ { resourceType: "CONTENT", resourceId: "myMessage", ... }, ... ]`

An Example of entire content-object can look as follows.

Table 22

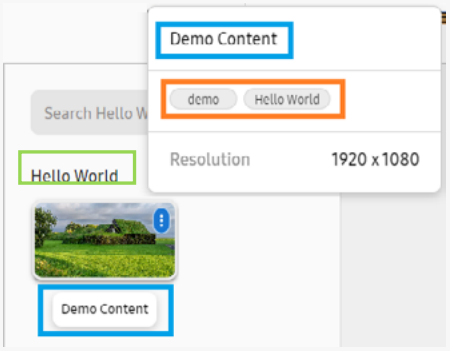
Content Object – Example (resourceType: “CONTENT”).

resourceType: “CONTENT”

Code

```
{
  resourceType: "CONTENT",
  resourceId: "content-1",
  resourceName: "Demo Content",
  resoureSize: "",
  thumbnailPath:
"https://cdn.pixabay.com/house.jpg",
  isLandscape: true,
  category: "Hello World",
  orientation: "landscape",
  allowOutOfCanvas: false,
  keywords: "demo, Hello World",
  config: {
   Style: [
    {
     type: "message",
     id: "myMessage",
     value: "Style your Widget"
    }
   ]
  }
}
```

UI



“Category” makes the Widget appear under specified category. It doesn’t make the category name (“Hello World”) display.

Table 23

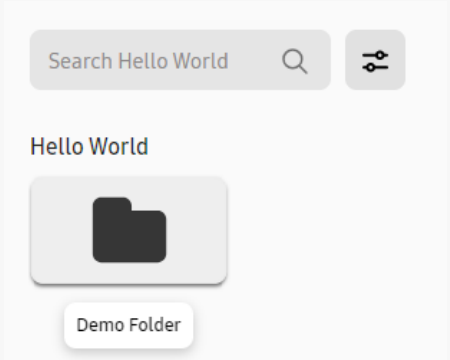
Content Object - Example (resourceType: "FOLDER").

resourceType: “FOLDER”

Code

```
{
  resourceType: "FOLDER",
  resourceId: "folder-1",
  resourceName: "Demo Folder",
  resoureSize: "",
  thumbnailPath: "",
  isLandscape: true,
  category: "Hello World",
  orientation: "landscape",
  allowOutOfCanvas: false,
  keywords: "demo, Hello World",
  children: [
   {
    resourceType: "CONTENT",
    resourceId: "folder-content-1",
    resourceName: "Child 1",
```

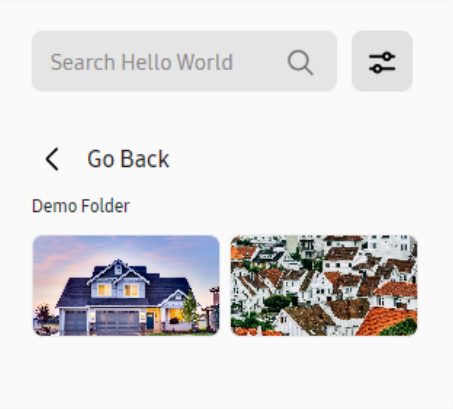
UI



resourceType: “FOLDER”

```
resoureSize: "",
    thumbnailPath: "https://cdn.pixabay.com/house2.
jpg",
    isLandscape: true,
    category: "Hello World",
    orientation: "landscape",
    allowOutOfCanvas: false,
    keywords: "demo, Hello World"
   },
   {
    resourceType: "CONTENT",
    resourceId: "folder-content-2",
    resourceName: "Child 2",
    resoureSize: "",
    thumbnailPath:
"https://cdn.pixabay.com/house3.jpg",
    isLandscape: true,
    category: "Hello World",
    orientation: "landscape",
    viewEdit: {
     message: "Click the button to display extra
information"
    },
    allowOutOfCanvas: false,
    keywords: "demo, Hello World",
    config: {
      Style: [
       {
        type: "message",
        id: "myMessage",
        value: "Style your Widget"
       }
      ]
   }
  ]
 }
```

After clicking on the folder icon, we can see its children (Widgets).



Categories

Now that we have Content covered, let's move to key information when it comes to understanding Categories.

First, you need to think about how you would like to divide your Widgets. You can display all of your Widgets under one category, or divide them into multiple categories. You can even define multiple categories but publish only some of them, based on User's selections in your PIRS Settings – we'll discuss conditional publishing in the next section of this Chapter. When you come up with an idea about how to group your Widgets, you should begin actually defining the Categories, with respect to WiNE API standards.

Note that when defining Content (using content objects), in parameter named "category" we can only specify within which already existing Category we would like given Widget to land in the UI. It is time to create the places we have referred to – we do this by creating an array of objects, where each of them represents a Category and has got two keys:

- **name:** the name of the Category
- **displayMode:** its value determined, whether Widgets within a given Category should be displayed as a Grid or as a Carousel.

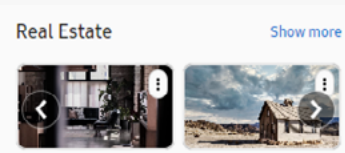
> NOTE: If we keep the number of Widgets as 2 and set displayMode to Carousel, we will see the following message in the browser console:

```
There are no slides to show. So the carousel won't be re-rendered

Option 'items' in your options is equal to the number of slides. So the
navigation got disabled
```

and the carousel will not show up.

Table 24

Categories - Display Mode

Categories - Display Mode		
Display Mode	Example - Code	Example - UI
carousel (default)	`{ name: "Real Estate", displayMode: "carousel" }`	
grid	`{ name: " Propiedad Inmobiliaria ", displayMode: "grid" }`	

NOTE: There is also an option to define category as a string. In such case, provide only the category name - display mode will be by default set to "carousel".

Table 25

Methods of Defining Categories.

Methods of Defining Categories	
Method	Example
Object	`const myCategories = [ { name: "Real Estate", displayMode: "carousel" }, { name: " Propiedad Inmobiliaria", displayMode: "grid" } ]`
String	`const myCategories = ["Real Estate", " Propiedad Inmobiliaria"]`

6.3.4 Conditional publishing

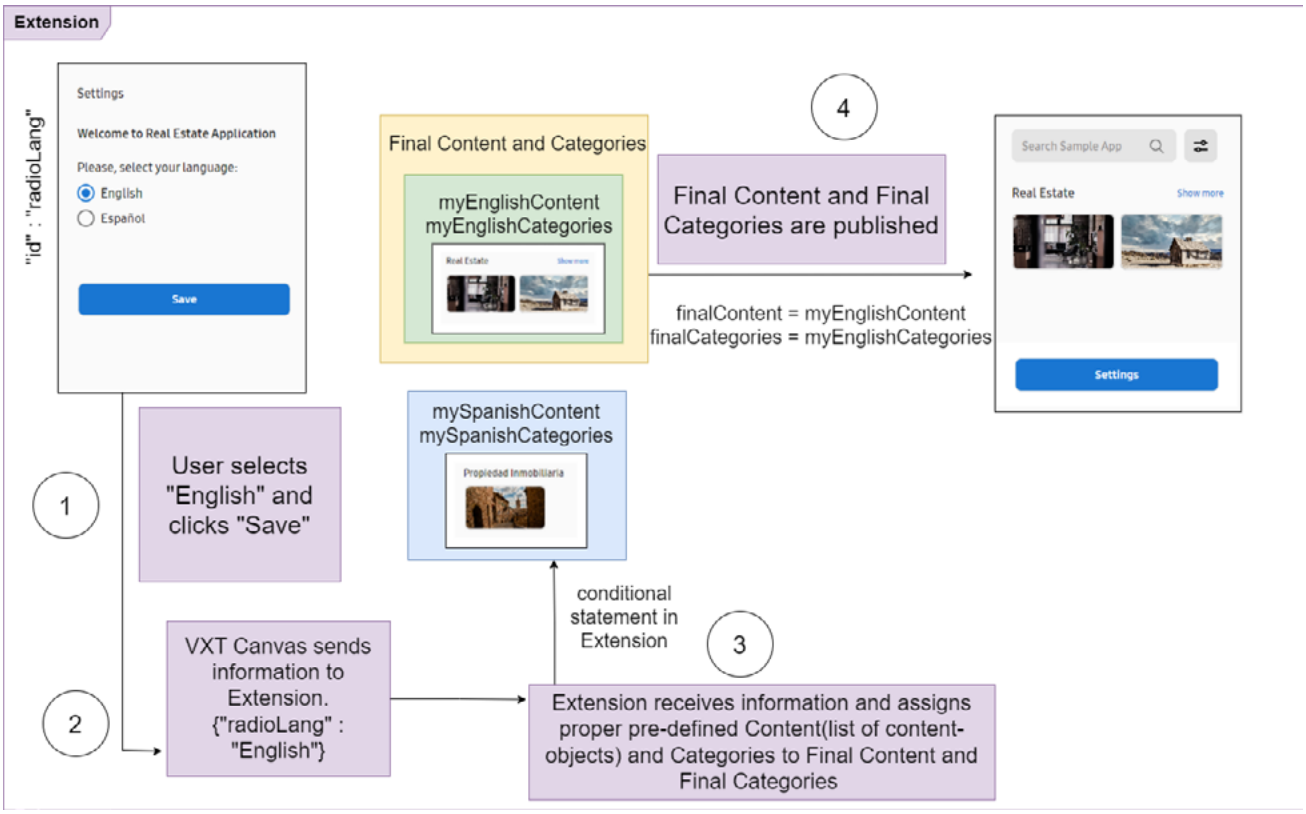
Every time User makes a selections in the Settings panel, changes Widget orientation, or types in something into the search box, Extension will receive a new “get” event with information about current User preferences regarding these three properties (Settings selections – if your PIRS uses Settings, orientation and search query). It is Extension’s responsibility (and yours – as a PIRS Developer) to implement correct handling of Settings and orientation selections, as well as of searching through your Widgets using keywords.

Settings filtering

If your Application uses Settings, you can decide, which Widgets and what Categories (or what Metadata – for Data-Driven App) will be displayed, based on User’s preferences. The simplest way of implementing it (for showcase purposes only, as real-life code is much more complicated than any demo) would be having a separate set of a Widgets list for every Settings option (or at least update content object properties) – or a separate set of Metadata, in case of Data Driven Applications.

A simplified example for Content Driven App can look as follows:

Figure 39 Settings filtering for Content Driven Application.



> NOTE: In this example myEnglishContent has 3 (landscape) Widgets, but since displayMode is set to "Carousel", the third Widget is hidden and will be displayed when User clicks the arrow visible on mouse hover over the Widgets, or when they click "Show more".

> NOTE II: Initial "orientation" value returned in a "get" event is always "landscape".

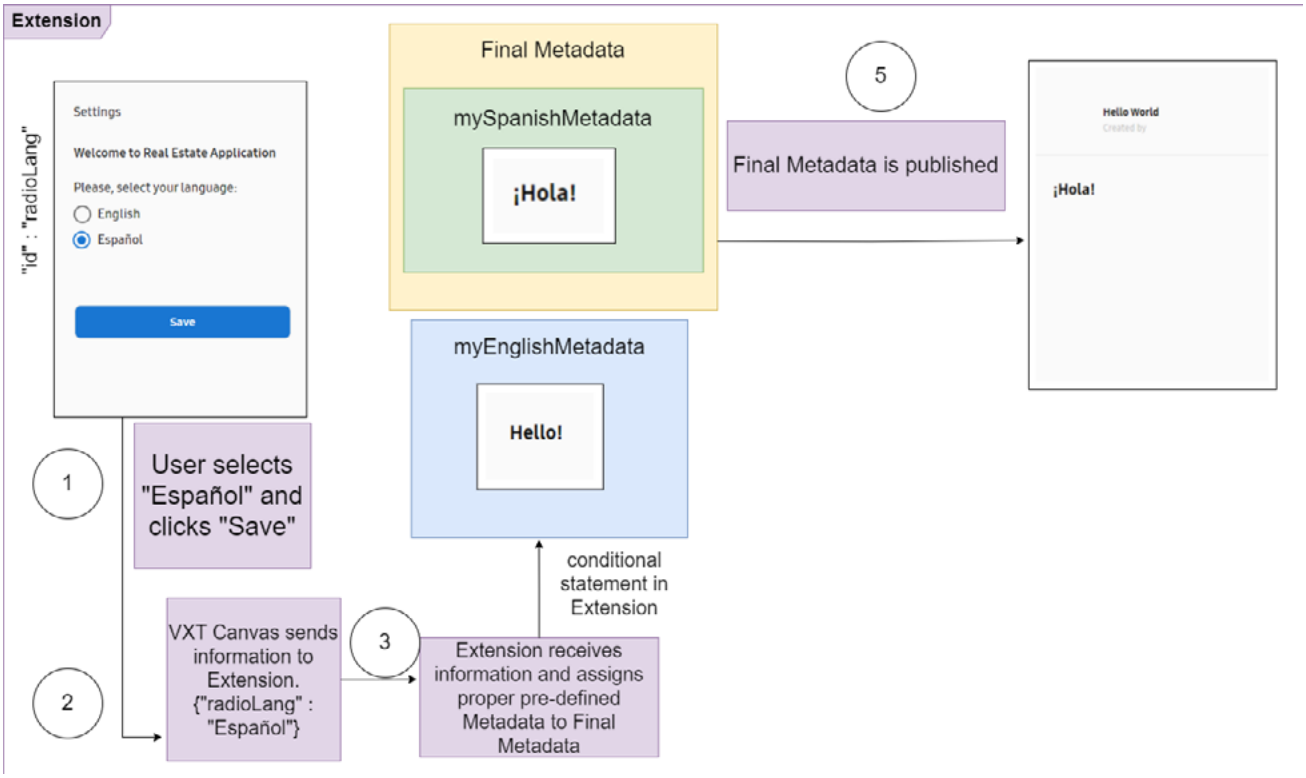
```
const channel = $vxt.createChannel($vxtSubChannelId);

let myEnglishCategories = [
  {
        name: "Real Estate",
        displayMode: "carousel"
  }
]

let mySpanishCategories = [
  {
         name: "Propiedad Inmobiliaria",
         displayMode: "grid"
  }
]

let myEnglishContent = [
       {
        resourceType: "CONTENT",
        resourceId: "1",
        resourceName: "Inside",
        thumbnailPath:
         "https://images.com/photo.jpg",
        isLandscape: true,
        category: "Real Estate",
        orientation: "landscape",
        keywords: "demo, english, inside",
        allowOutOfCanvas: true,
        config: getConfig("English", "1")
```

```
  },
  {
    resourceType: "CONTENT",
    resourceId: "2",
    resourceName: "Outside",
    thumbnailPath:
      "https://cdn.pixabay.com/photo/2022/01/30/19/46/school-6982073_1280.jpg",
    isLandscape: true,
    category: "Real Estate",
    viewEdit: {
      message: "Click the button to display extra information"
    },
    orientation: "landscape",
    keywords: "demo, english, outside",
    allowOutOfCanvas: false,
    config: getConfig("English", "2"),
  },
  {
    resourceType: "CONTENT",
    resourceId: "3",
    resourceName: "Balconies",
    thumbnailPath:
      "https://images.com/photo2.jpg",
    isLandscape: true,
    category: catName,
    orientation: "landscape",
    keywords: "demo, english, outside",
    allowOutOfCanvas: false,
    config: getConfig("English", "3"),
  },
  {
    resourceType: "CONTENT",
    resourceId: "4",
    resourceName: "Balconies",
    thumbnailPath:
      "https://images.com/photo2.jpg",
    isLandscape: false,
    category: "Real Estate",
    orientation: "portrait",
    keywords: "demo, english, outside",
    allowOutOfCanvas: false,
    config: getConfig("English", "4"),
  }

]
```

```
let mySpanishContent = [{
            resourceType: "CONTENT",
            resourceId: "4",
            resourceName: "Demo",
            thumbnailPath:
             "https://images.com/photo3.jpg ",
            isLandscape: true,
            category: "Propiedad Inmobiliaria",,
            orientation: "landscape",
            keywords: "demo, spanish",
            config: getConfig("Español", "5"),
          }]

  channel.subscribe("get", (response) => {
            let finalContent;
            let finalCategories;

// SETTINGS FILTERING
      switch (response.payload.radioLang){
        case "English":
                          finalContent = myEnglishContent;
                          finalCategories = myEnglishCategories;
         break;
        case "Español":
                          finalContent = mySpanishContent;
                          finalCategories = mySpanishCategories;
         break;
           default:
                          finalContent = myEnglishContent;
                          finalCategories = myEnglishCategories;
           }
             channel.publish("data", {
      type: "contents",
      payload: {

       categories: finalCategories;
       content: finalContent;
      }
     });
   });
```

A simplified example for Data Driven App can look as follows:

Figure 40 Settings Filtering for Data Driven Application.




```
const channel = $vxt.createChannel($vxtSubChannelId);

const myEnglishMetadata = [
{
type:"header",
title:"Hello!"
}
]

const mySpanishMetadata = [
{
type:"header",
title:"¡Hola!"
}
]

channel.subscribe("get", (response) => {
let finalMetadata;
switch (response.payload.radioLang){
      case "English":
                        finalMetadata = myEnglishMetadata;
                        break;
            case "Español":
```

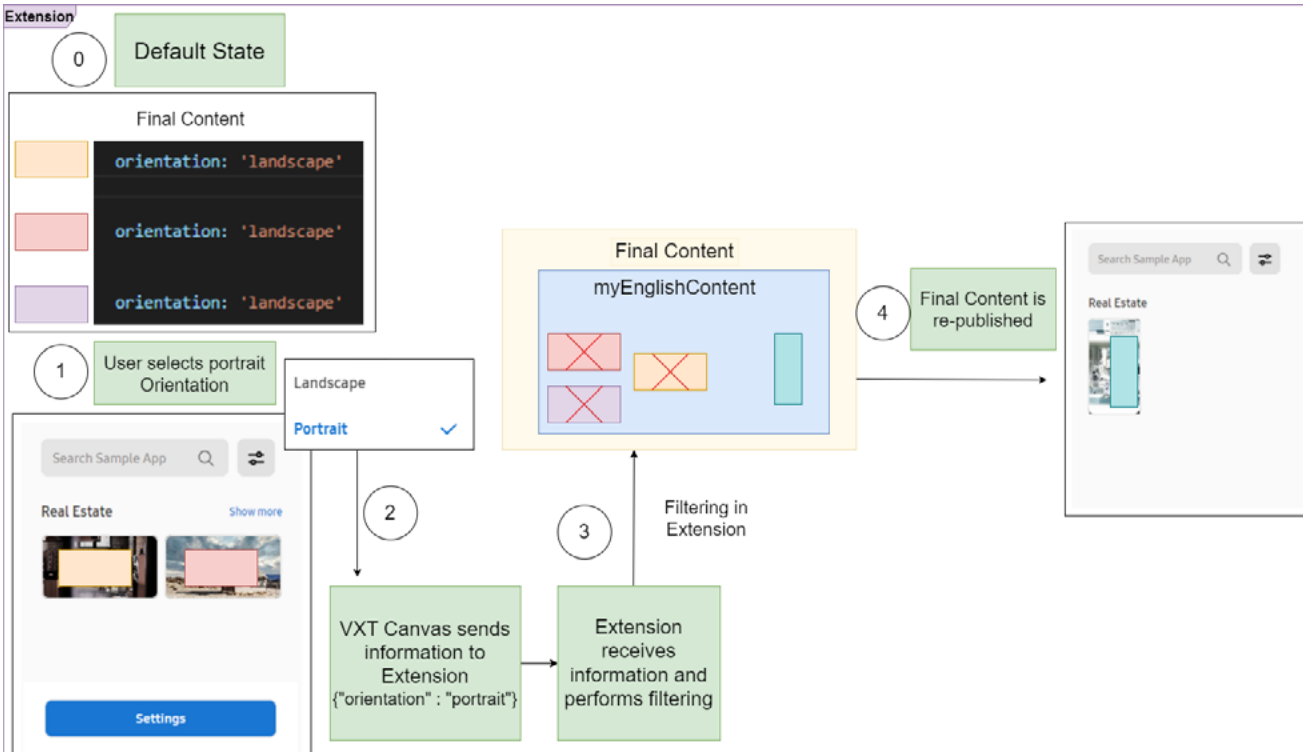
```
                    finalMetadata = mySpanishMetadata;
                    break;
                    default:
                    finalMetadata = myEnglishMetadata;

channel.publish("data", {
  type: "metadata",
  payload: finalMetadata
 });
});
```

Orientation filtering (content-driven only)

If your Application supports both Portrait and Landscape Orientations, it is crucial that it is communicated to Users. As we know, Users can filter Widgets based on that condition using the "filter" button on the right side of search box. Default orientation is always "landscape" and when User selects "portrait" Orientation, Content must be re-published.

If we add orientation filtering to our previous example (regarding Content Driven Apps), it will look as follows:

Figure 41 Orientation Filtering.



> NOTE: In this example myEnglishContent has 3 (landscape) Widgets, but since displayMode is set to "Carousel", the third Widget is hidden and will be displayed when User clicks the arrow visible on mouse hover over the Widgets, or when they click "Show more".

> NOTE II: Initial "orientation" value returned in a "get" event is always "landscape".

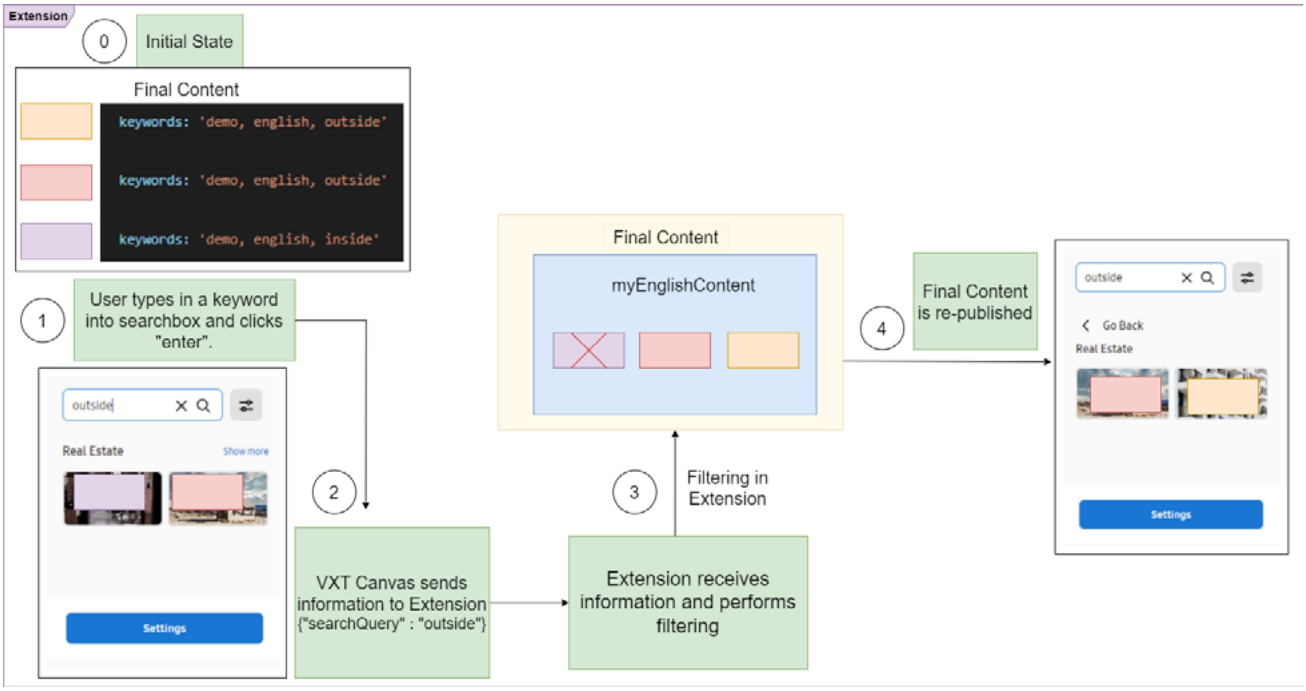
```
channel.subscribe("get", (response) => {
        let finalContent;
let finalCategories;
// SETTINGS FILTERING
    switch (response.payload.radioLang){
      case "English":
                    finalContent = myEnglishContent;
                    finalCategories = myEnglishCategories;
       break;
      case "Español":
                    finalContent = mySpanishContent;
                    finalCategories = mySpanishCategories;
       break;
      default:
                    finalContent = myEnglishContent;
                    finalCategories = myEnglishCategories;
        }

//ORIENTATION FILTERING
   finalContent = finalContent.filter(widget => widget.orientation ===  response.payload.orientation)

        channel.publish("data", {
    type: "contents",
    payload: {
          categories: finalCategories,
     content: finalContent
    }
  });
 });
```

Search Query filtering (content-driven only)

Apart from Settings and Orientation filtering, VXT Canvas also gives Users the possibility to filter Widgets by Keywords (for example, if they only want to see Widgets for marketing info display). Initial Search Query is always empty (""), and when User types something in the Search Box above Widgets (and presses "Enter"), Content must be re-published.

If we add search query filtering, the outcome will look like presented below:

Figure 42 Search Query Filtering.



> NOTE: In this example myEnglishContent has 3 (landscape) Widgets, but since displayMode is set to "Carousel", the third Widget is hidden and will be displayed when User clicks the arrow visible on mouse hover over the Widgets, or when they click "Show more".

> NOTE II: Initial "orientation" value returned in a "get" event is always "landscape".

```
channel.subscribe("get", (response) => {
    let finalContent;
    let finalCategories;
```

```
        // SETTINGS FILTERING
            switch (response.payload.radioLang){
              case "English":
                        finalContent = myEnglishContent;
                        finalCategories = myEnglishCategories;
               break;
              case "Español":
                        finalContent = mySpanishContent;
                        finalCategories = mySpanishCategories;
               break;
                        default:
                        finalContent = myEnglishContent;
                        finalCategories = myEnglishCategories;
          }

          //ORIENTATION FILTERING
    finalContent = finalContent.filter(widget => widget.orientation ===   response.payload.orientation)

          //SEARCH QUERY FILTERING
     if (response.payload.searchQuery !== ""){
      finalContent = finalContent.filter(widget => widget.keywords.indexOf(response.payload.searchQuery) !==
-1)
     }

            channel.publish("data", {
      type: "contents",
      payload: {
       categories: finalCategories;
       content: finalContent;
      }
     });
    });
```

6.4 Writing Your First VXT Widget (content-driven only)

6.4.1 Overview

Widget is the content that is displayed first on canvas area in VXT Canvas, and then on the screen in VXT Player, as a Canvas content. It is launched (loaded onto canvas area) when User clicks on its graphical representation (thumbnail) in Extension, in the left sidebar. If an App features a Property Panel, then the

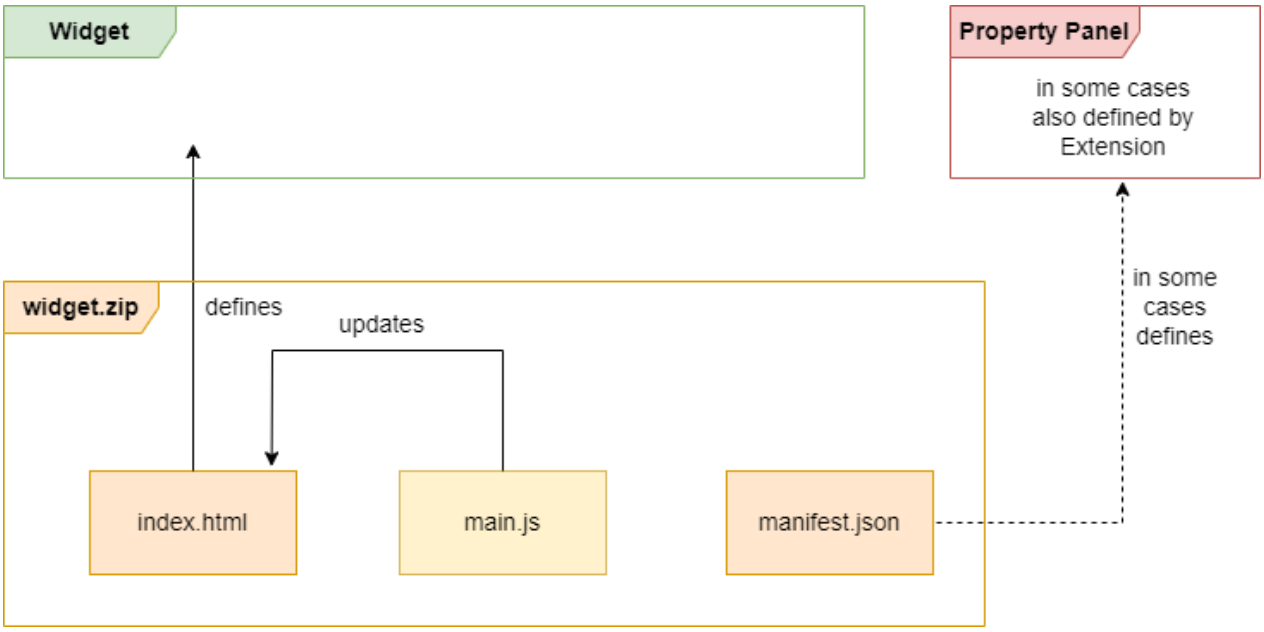
Widget is subject to various modifications through elements of Property Panel made by the end User. Then, the project is saved in VXT CMS and can be published to the screen. Of course, the project can be later modified and/or deleted. We can think of the Widget as the essence of your Application.

Since Widget is a web application at its core, its creation is far less complicated than the creation of Extension. It still, however, requires the use of WiNE API. From the birdseye perspective Widget is built in the following way:

- Open a channel.
- Subscribe (listen for):
  - Configuration selections coming from Property Panel (and other useful information),
  - Playstates,
  - Vxtstates.
- Make your Widget/Application respond to received information (configuration selections, playstates, vxtstates) – for example, change color of the text, pause video content or save some data in your database – the implementation of this part is completely up to you, no WiNE API is involved in it.
- Publish:
  - Toast message in VXT Canvas UI,
  - New Manifest – update the elements of Property Panel.

But before we start our development, we need to understand which file controls which element of the UI:

Figure 43 Widget Files.



In this Chapter, we'll focus on the content of main.js file.

6.4.2 Events

Table 26

Basics of Subscribe (widget).

Subscribe - Basics		
Event	Description	How to subscribe?
config	Information about configuration properties' states (state of Property Panel and the hidden property), and Widget's instance ID.	`channel.subscribe("config", (response) => { // do something with received response });`
playstate	Information about the state of playing the Widget (both in VXT Canvas and VXT Player).	`channel.subscribe("playstate", (response) => { // do something with received response });`
vxtstate	Information about VXT Canvas Project.	`channel.subscribe("vxtstate", (response) => { // do something with received response });`

Table 27

Subscribe - Deep Dive (widget).

Subscribe - Deep Dive			
Event	Response Types	Description of Response	Full Response - Example

config	Type not used	{
		"theme": // for future VXT development
		"data": // Property Panel configuration denoted as:
		{
		"<Tab Name>": [
		{ "<id>": "<value>" },
		...
		],
		...
		}
		"subscription": // info about VXT subscription
		"id": // unique identifier tied to User's workspace
		"widgetId": // unique identifier for Widget's instance on canvas area
		}
			`{"theme": {}, "data": { "Configuration": [{ "colorPalette":"***", "limitedOffers":"#e66465" }] }, "subscription": {"requester": "canvas", "status":"trial"}, "id":"vxt-...", "widgetId":"vxt-..." }`

Subscribe - Deep Dive			
playstate	ready	This type of playstate fires when Widget is loaded ; upon receiving this playstate message the Widget prepares content to be played in next lifecycle Event.	{"type": "ready", "payload": {}}
	play	This type of playstate fires when Widget is loaded and played; upon receiving this playstate message the Widget starts displaying its content.	{"type": "play", "payload": {}}
	hold	This type of playstate fires when VXT Player exit confirmation pop-up is displayed (when someone clicks “info” on the remote, then selects “exit” on the right panel); upon receiving this playstate message the Widget pauses displaying the content and renders the default UI of the Widget.	{"type": "hold", "payload": {}}
	resume	This type of playstate fires when user clicks “cancel” in the aforementioned confirmation popup; upon receiving this playstate message the Widget resumes displaying the content and hides the default UI.	{"type": "resume", "payload": {}}
	stop	This type of playstate fires when User discards project / deletes Widget / the playing is stopped - upon receiving this playstate message the Widget should finish playing content, close communication, disconnect from external sources, stop timers and close platform APIs.	{"type": "stop", "payload": {}}
vxtstate	new	This type of vxtstate fires when User clicks “New” in VXT Canvas (either after saving the project, or discarding the project completely without saving).	{"type": "new", "payload": {}}
	save	This type of vxtstate fires when User clicks “Save” or “Save as” in VXT Canvas. It indicates that the project might be later displayed on Screen.	{"type": "save", "payload": {}}
	preview	This type of vxtstate fires when User clicks “Preview” in VXT Canvas.	{"type": "preview", "payload": {}}

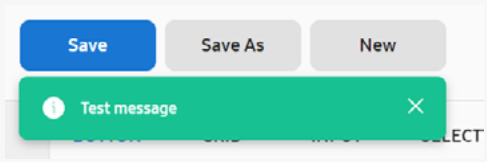
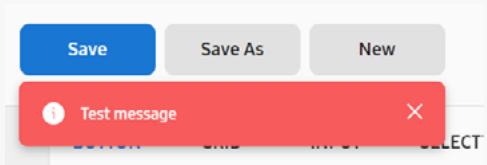
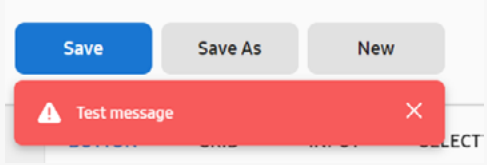
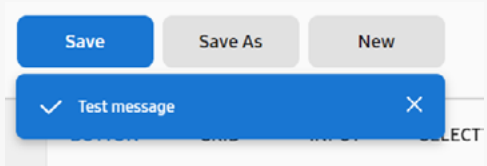
Table 28

Basics of Publish (widget).

Publish - Basics			
Event	How to publish?	Response Type	Response Payload
toastMessage	`channel.publish("toastMessage",{ type:"<toast message type>", payload:{ message:"<message to display>" } })`	info warning error success	Message to display inside the toast pop-up.
data	`channel.publish("data",{ type:"widgetManifest", payload:{ "<Tab Name>":[ { // configuration object }, ... ] } })`	widgetManifest	New Property Panel definition. If specified tab name doesn't exist, it will be added, together with the elements specified in its array. If it exists, VXT will check if the elements specified in the array are to be updated, or added anew. To update existing elements, use the id and the properties that should be changed. To add new elements, write the full configuration object and give it unique id.

Table 29

Publish - Deep Dive (widget).

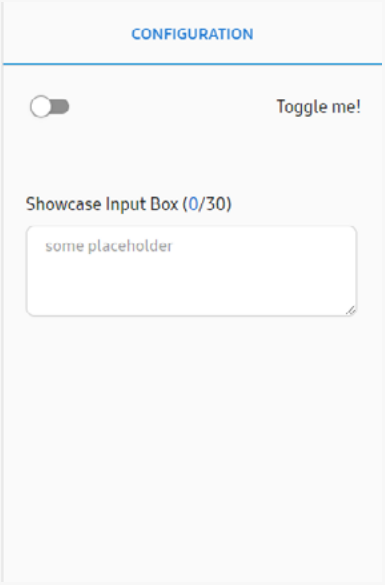
Publish - Deep Dive		
Event	Example - Code	Example - UI
toastMessage	`channel.publish("toastMessage",{ type: "info", payload: { message: "Test message" } });`	
	`channel.publish("toastMessage",{ type: "warning", payload: { message: "Test message" } });`	
	`channel.publish("toastMessage",{ type: "error", payload: { message: "Test message" } });`	
	`channel.publish("toastMessage",{ type: "success", payload: { message: "Test message" } });`	

Publish - Deep Dive

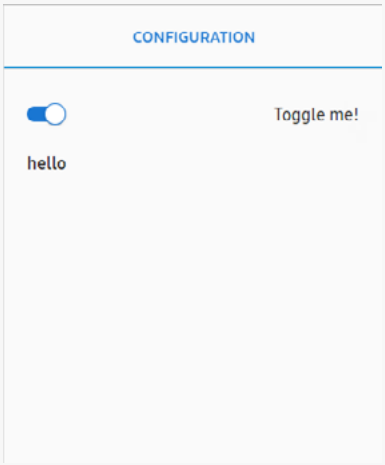
data

```
channel.publish("data" {
  type: "widgetManifest",
  payload: {
   "Configuration": [
    {
     "id": "myToggle",
     "value": true
    },
    {
     "id": "myInput",
     "viewState": "hide"
    },
    {
     "type": "message",
     "id": "newMessage",
     "value": "hello"
    }
   ]
  }
})
```

Initial State:



State after publishing data (widgetManifest):



We have successfully updated two elements (changed value of toggle to true and hid input box) and added one new element (message).

NOTE: Usually this kind of operation would be wrapped in a conditional statement.

6.4.3 Making your Widget interactive in VXT Canvas

By default, in VXT Canvas, Widget dragged and dropped onto canvas area is not interactive, due to the context it's being put in. It means that if your Widget contains any buttons or input boxes, they won't be accessible to the User in VXT Canvas (except for the Preview, where the context changes and Widget is displayed as any other web content, as a top layer – the same is applicable to VXT Player).

However, there is a way to make your Widget interactive in canvas area.

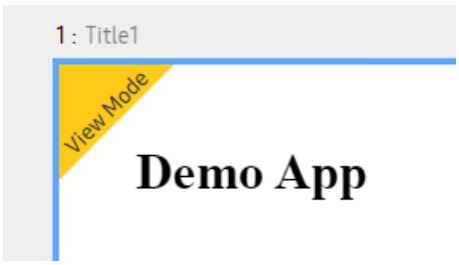
As you might remember, when defining the list of Widgets (content) in Extension, you can use a property called "viewEdit".

```
{
  resourceType: "CONTENT",
  resourceId: "1",
  resourceName: "Demo",
  resourceSize: "",
  thumbnailPath: "",
  isLandscape: true,
  category: "Demo",
  orientation: "landscape",
  viewEdit: { //this property enables Widget's interactivity
    message: "some extra message (guidance for user) to display at the top"
  },
  keywords: "demo"
}
```

If it is present and has a valid value, when the Widget that's carrying such property (once dragged and dropped onto canvas area) will have a yellow, clickable corner:

Figure 44

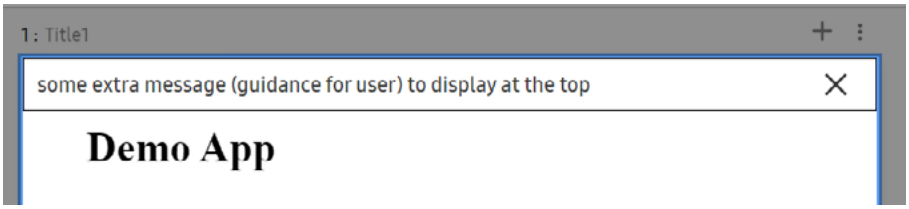
View Mode Flag.



When the corner is clicked, the message (additional guide for User) will be displayed...

Figure 45

Edit Mode Message.



... and if you have any interactive elements in your Widget, they will become clickable.

This functionality is often used with publishing Manifest – based on User interactions with the Widget, we can update Property Panel, for example, to enable or disable some elements, hide them or show.

6.4.4 Distinguishing between Widgets

There are two concepts to join together when it comes to Extension and Widget. We already know that we can have multiple Widgets in our Extension (for example, to give our Users the opportunity to choose from various templates).

How can it be achieved if there is only one index.html file?

In order to answer this question and fully grasp these two concepts, we need to understand, what a Widget in Extension is.

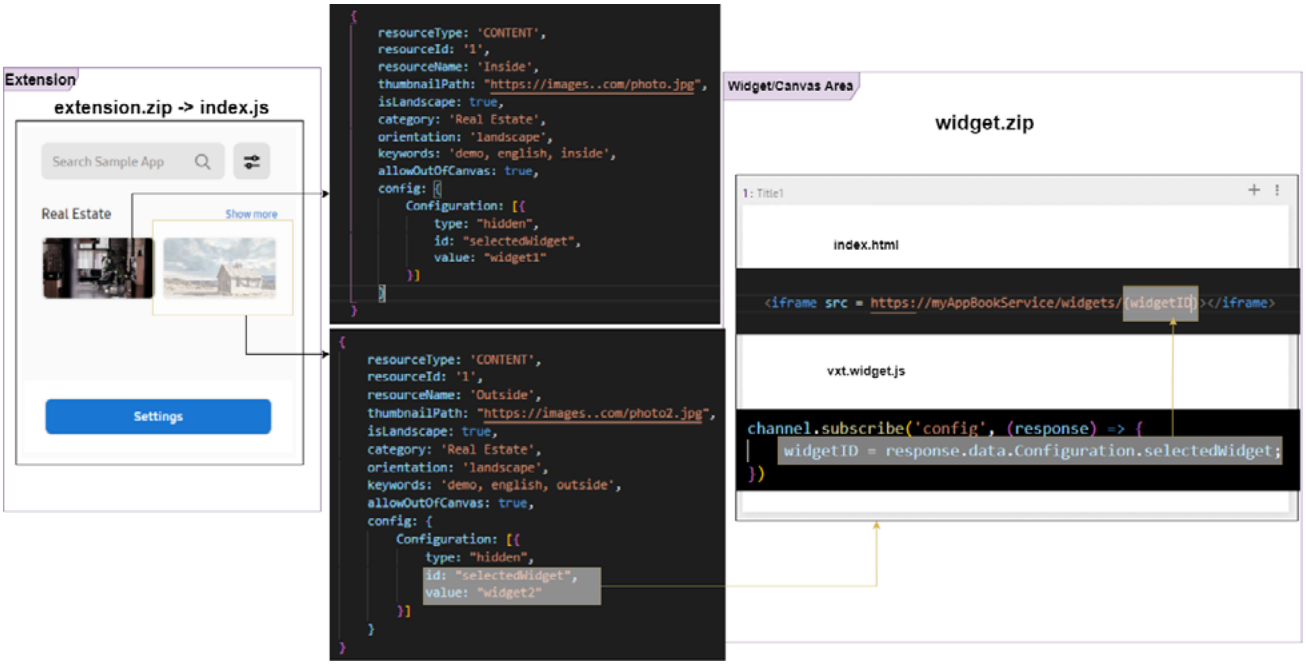
These thumbnails are not references to separate index.html files – these are custom configuration objects. Their configuration, if declared correctly, will be communicated to your Widget's JS file, as a response of a "config" event. This way, you will be able to decide, which content should be displayed on canvas area (for example by embedding a correct iframe source, or otherwise rendering the content).

This is where the hidden property comes in handy. It should be placed inside the "config" property when defining content object, together with other Property Panel elements.

To learn more about configuration objects and the hidden property, please refer to "Creating Your First VXT Widget Manifest (JSON) File (content-driven only) Chapter" and "WiNE API Reference" Section.

A simplified schema of the hidden property concept can be presented in the following way:

Figure 46 Hidden Property Concept.



6.5 Creating Your First VXT Extension Manifest (JSON) File (data & content-driven)

As we already know, a Manifest file is made of an array of configuration objects.

A complete list of configuration objects (with examples) is presented in "WiNE API Reference" Section.

When it comes to Extension's Manifest, it defines what's displayed in PIRS Settings.

An important thing to bear in mind is that we cannot use all the configuration objects in Settings (type: img and type: gridImage can be used only in Property Panel).

In our example (language switcher), the manifest JSON looks like this:

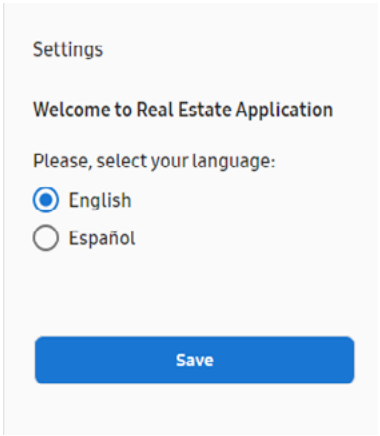
```
[{
  "type": "message",
  "id": "welcome",
  "value": "Welcome to Real Estate Application"
},
{
  "type": "radio",
```

```
    "id": "settingsLang",
    "caption": "Please, select your language:",
    "value": "English",
    "list": ["English", "Español"]
  }]
```

And the outcome in the UI is:

Figure 47

Example Settings.



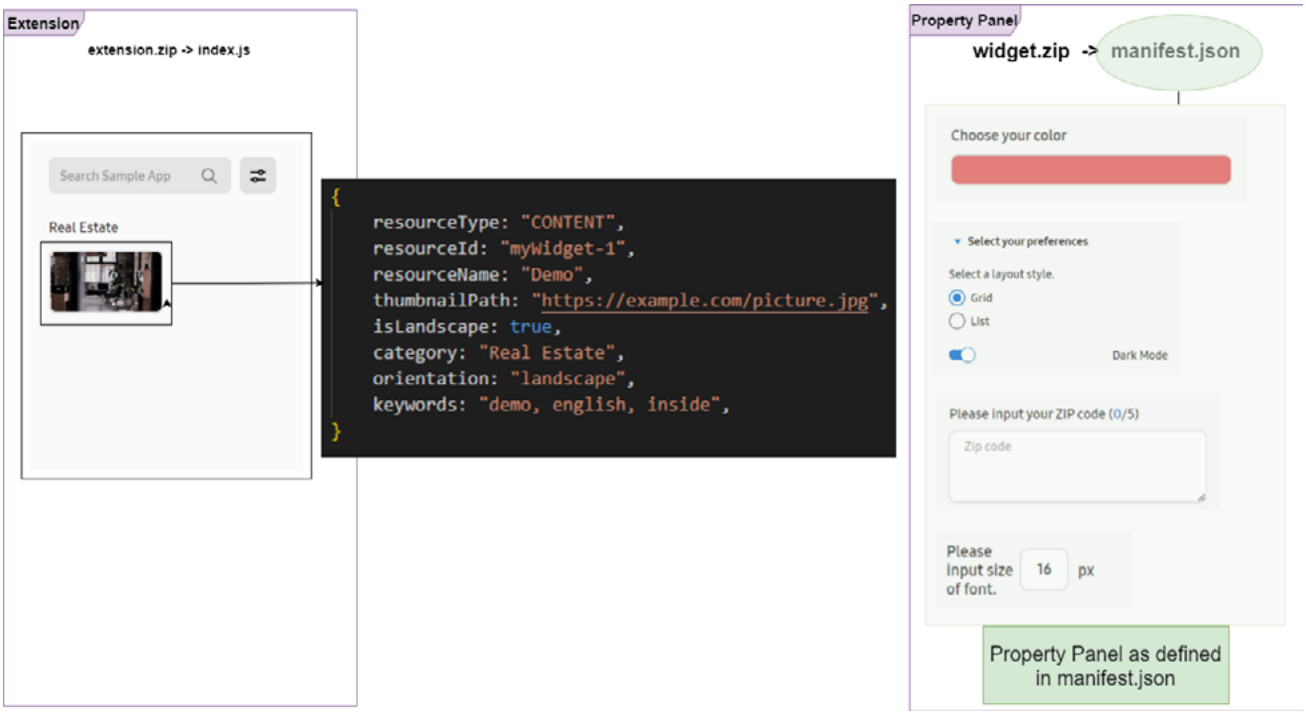
Note that even though we made our App support two languages, the Settings part will always be displayed in English (or in whatever language we write configuration objects' displayable elements). This manifest cannot be dynamically updated.

6.6 Creating Your First VXT Widget Manifest (JSON) File (content-driven only)

In order to understand the role of Widget's Manifest (JSON) File, we must be familiar with the role of Extension and content objects. Now we know that in Content Driven App's Extension we have a list of Widgets (published as a value of "content" property in payload). Every Widget can have its own customization options, so its own version of Property Panel. We can decide, however, that all Widgets should have the same Property Panel. Finally, we can conclude, that the best option for our Application would be if all of our Widgets had some common elements of Property Panel, and some of their own. This is where the concepts of Manifest and "config" property of individual content object come together.

Let's examine all three situations, starting with the simplest scenario – when we use only one Widget, so our App will have only one version of Property Panel. In this case, we don't need to use the "config" property when defining our content object and we can rely solely on Manifest.

Figure 48 Property Panel (manifest).



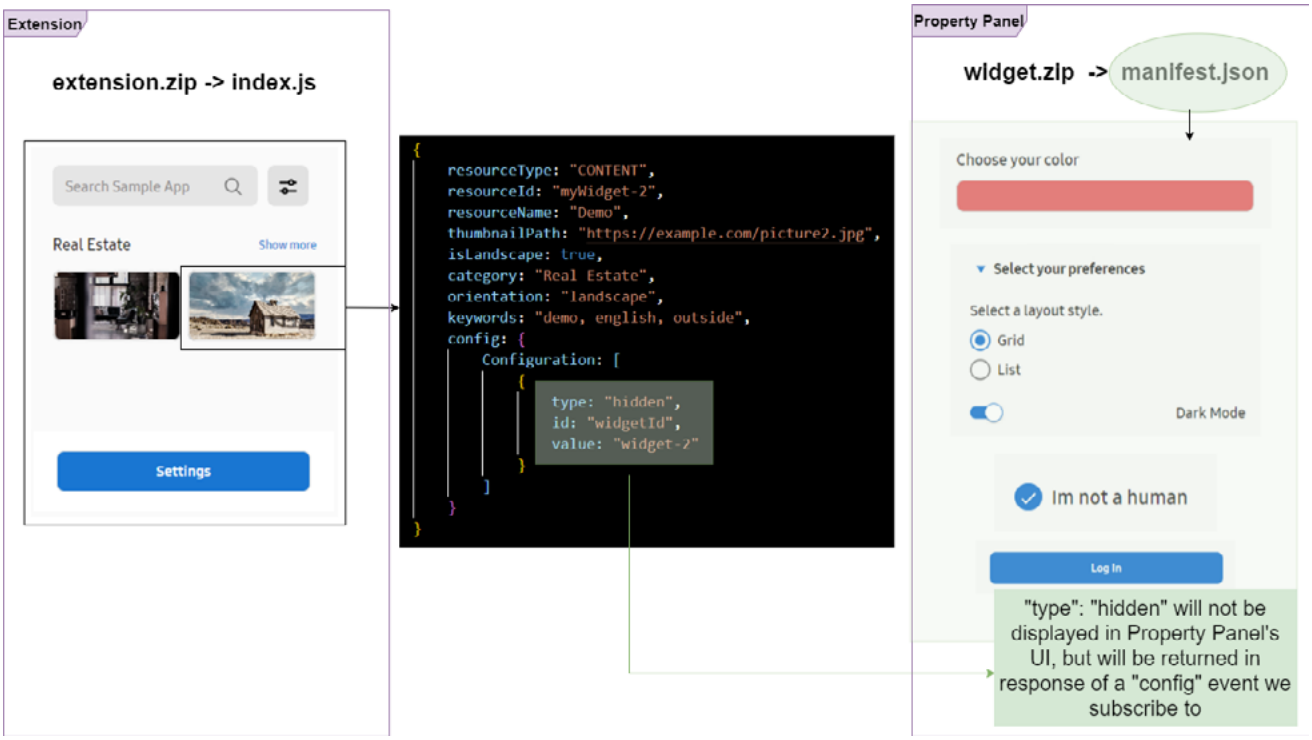
In this example, the Manifest looks like this:

```
{
  "Configuration":[
    {
      "type":"color",
      "id":"colorPalette",
      "caption":"Choose your color",
      "value":"#e66465"
    },
    {
      "type":"accordion",
      "id":"acc",
      "caption":"Select your preferences",
      "list":[
        {
        "type":"radio",
        "id":"radioSelectStyle",
        "caption":"Select a layout style.",
        "list":[
          "Grid",
```

```
            "List"
          ],
          "value":"Grid"
          },
          {
          "type":"toggle",
          "id":"toggleTheme",
          "caption":"Dark Mode",
          "value":true
          }
        ]
      },
      {
        "type":"text",
        "id":"inputZipCode",
        "caption":"Please input your ZIP code",
        "value":"",
        "multiline": true,
        "charactersLimit": 5,
        "label":"Zip code"
      },
      {
        "type":"number",
        "id":"inputFontSize",
        "caption":"Please input size of font.",
        "value":16,
        "label":"Font",
        "min":5,
        "max":25,
        "numericType":"float",
        "units":"px"
      }
    ]
  }
```

Second situation, similar to the one above is when we want to have multiple Widgets, but all of them should have exactly the same Property Panel. In this case, we need to use the “config” property in our Extension to pass an ID of individual Widget, so that we can detect in our Widget’s JS code, which one of them had been selected by the User (for example, if our Widgets act as templates, with differing layouts).

Figure 49 Property Panel (manifest and hidden).



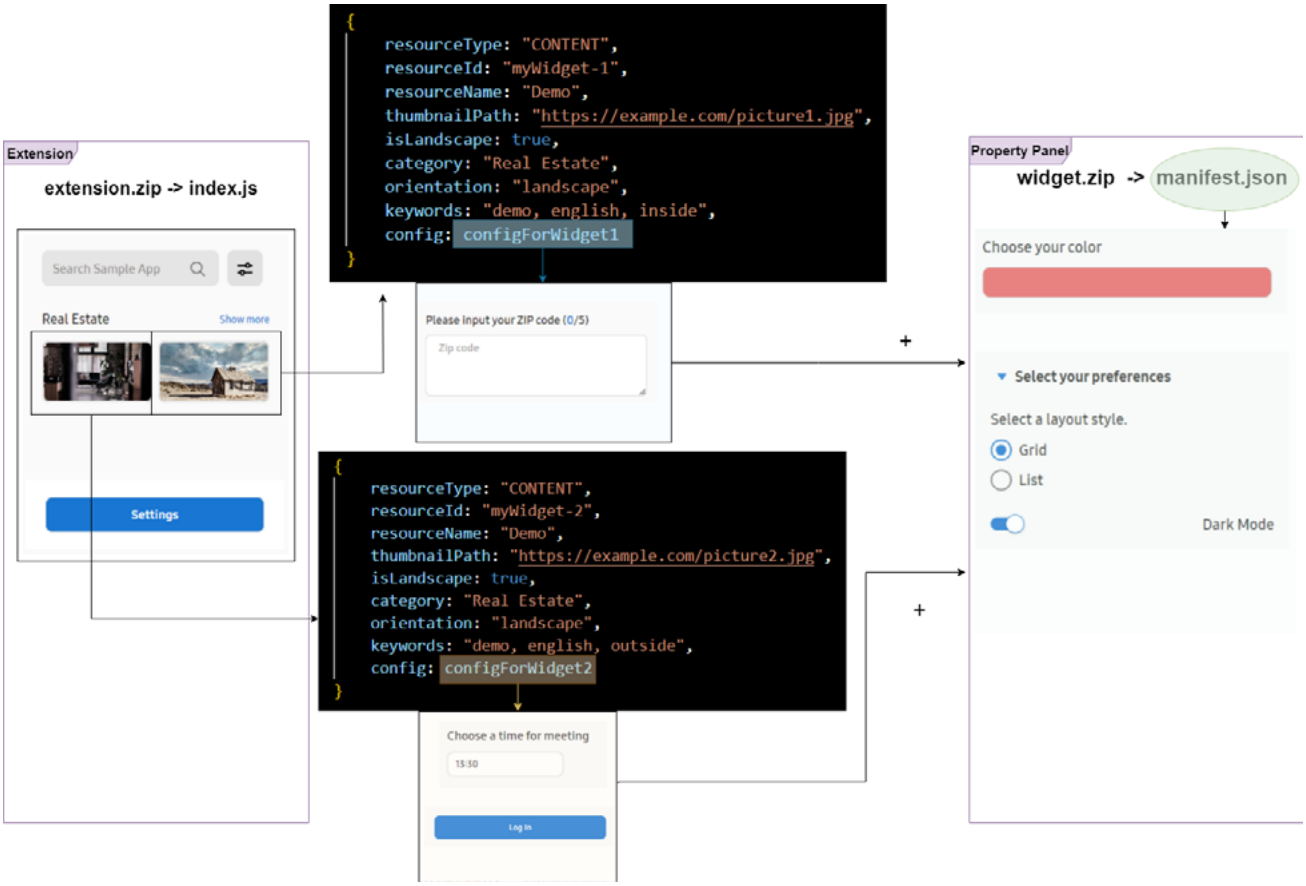
Since we already know the “config” value from studying the schema above, it’s time now to have a look inside the Manifest file:


```
{
  "Configuration":[
    {
      "type":"color",
      "id":"colorPalette",
      "caption":"Choose your color",
      "value":"#e66465"
    },
    {
      "type":"accordion",
      "id":"acc",
      "caption":"Select your preferences",
      "list":[
        {
        "type":"radio",
        "id":"radioSelectStyle",
        "caption":"Select a layout style.",
        "list":[
          "Grid",
          "List"
        ],
        "value":"Grid"
        },
```

```
        {
          "type":"toggle",
          "id":"toggleTheme",
          "caption":"Dark Mode",
          "value":true
        }
      ]
    },
    {
      "type":"checkbox",
      "id":"checkNotHuman",
      "caption":"Im not a human",
      "value":true
    },
    {
      "type":"url",
      "id":"urlVideo",
      "caption":"Log In",
      "value": "https://myApp.com/login",
      "openType":"dialog",
      "dialogPosition": {
        "x": 10,
        "y": 10
      },
      "dialogSize":{
        "width":560,
        "height":315
      },
      "dialogAuthValidation": "https://myAppBackService.com/getToken"
    }
  ]
}
```

Third situation is when we want our App to support multiple Widgets and some of them should have their own customization options, as well as a common part of Property Panel shared across all Widgets. In this case, we also need to use both “config” property in our Extension, as well as manifest.json. This time, however, the “config” property’s value will be more complex than in the previous example.

Figure 50 Property Panel (common and separate).



In this example, the configForWidget1 and configForWidget2 variables look like this:

```
const configForWidget1 = {
 Configuration: [
  {
   type:"time",
   id:"time24",
   caption:"Choose a time for meeting",
   value:"13:30",
   format:"HH:MM",
   size:"small",
   position:{
     x:0,
     y:15
   }
  },
  {
   type:"url",
   id:"urlVideo",
   caption:"Log In",
```

```
    value: "https://myApp.com/login",
    openType:"dialog",
    dialogPosition: {
      x: 10,
      y: 10
    },
    dialogSize:{
      width:560,
      height:315
     },
    dialogAuthValidation: "https://myAppBackService.com/getToken"
    }
  ]
}

const configForWidget2 = {
  Configuration: [
   {
    type:"text",
    id:"inputZipCode",
    caption:"Please input your ZIP code",
    value:"",
    multiline: true,
    charactersLimit: 5,
    label:"Zip code"
  }
  ]
}
```

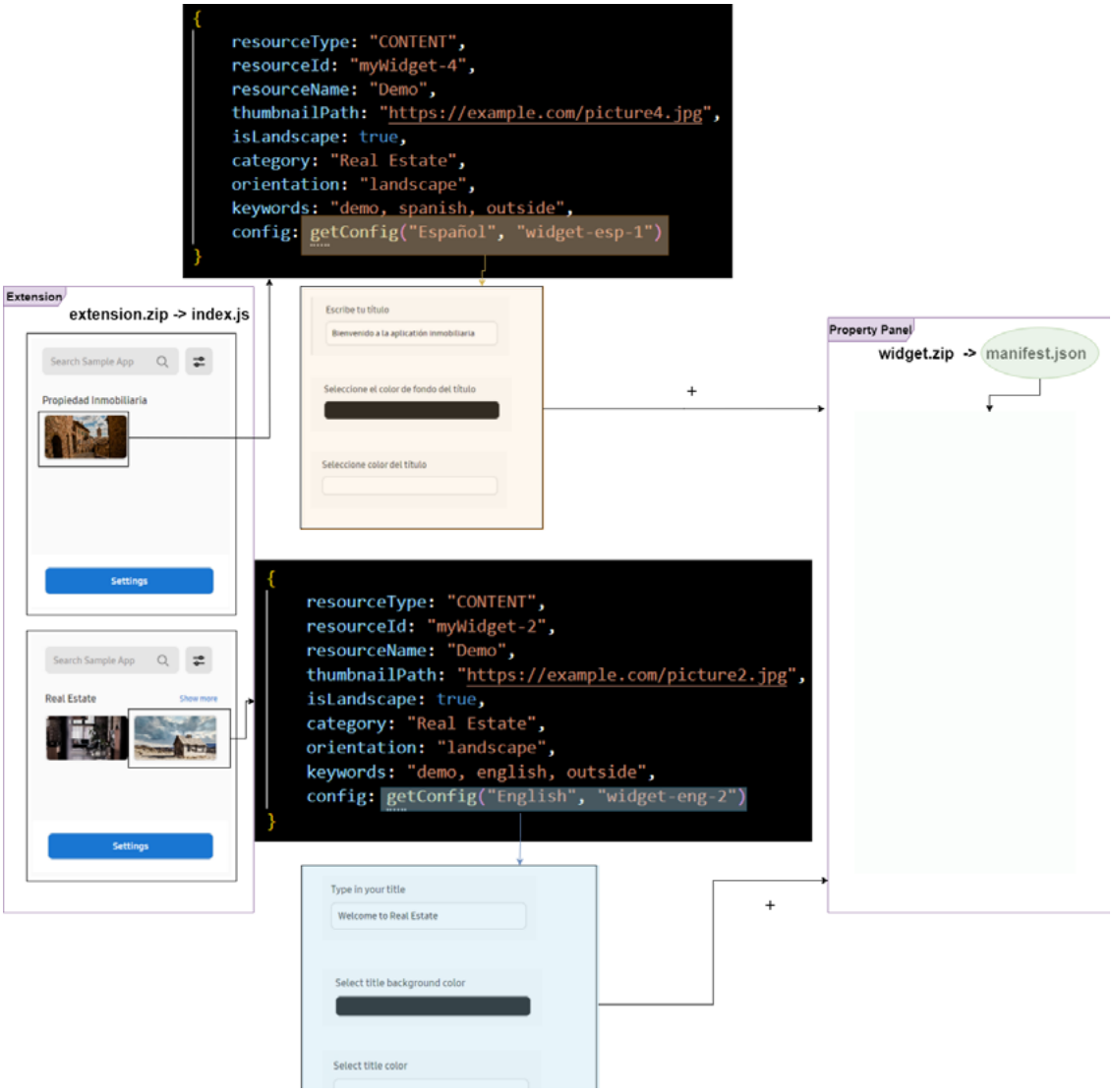
And manifest.json is composed in the following way:

```
{
  "Configuration":[
    {
      "type":"color",
      "id":"colorPalette",
      "caption":"Choose your color",
      "value":"#e66465"
    },
    {
      "type":"accordion",
      "id":"acc",
      "caption":"Select your preferences",
```

```
        "list":[
          {
          "type":"radio",
          "id":"radioSelectStyle",
          "caption":"Select a layout style.",
          "list":[
            "Grid",
            "List"
          ],
          "value":"Grid"
          },
          {
          "type":"toggle",
          "id":"toggleTheme",
          "caption":"Dark Mode",
          "value":true
          }
        ]
      }
    ]
  }
```

Fourth scenario was presented in the Extension's example in this Guide, where we want to support two language versions. Even if we decide that English Widgets should have a part of Property Panel in common, the Property Panel for Spanish version will be completely different (the elements will be written in Spanish). In this case, we cannot have any common Property Panel, therefore our manifest.json will be empty {} and the entire Property Panel will be defined and passed in Extension, inside a "config" property of each content object.

Figure 51 Property Panel (separate only).



In this example, the getConfig() function looks like this:

```
function getConfig(language, id) {
  switch (language) {
   case "English":
    return {
     Style: [
      {
       type: "text",
       id: "titleText",
       caption: "Type in your title",
       value: "Welcome to Real Estate",
       label: "Type in..."
      },
      {
       type: "color",
       id: "bgColor",
```

```
        caption: "Select title background color",
        value: "#000000"
       },
       {
        type: "color",
        id: "titleColor",
        caption: "Select title color",
        value: "#ffffff"
       },
       {
        type: "hidden",
        id: "selectedWidget",
        value: id
       }
      ]
     }
    case "Español":
     return {
      Estilo: [
       {
        type: "text",
        id: "titleText",
        caption: "Escribe tu título",
        value: "Bienvenido a la aplicación inmobiliaria",
        label: "Escribe..."
       },
       {
        type: "color",
        id: "bgColor",
        caption: "Seleccione el color de fondo del título",
        value: "#000000"
       },
       {
        type: "color",
        id: "titleColor",
        caption: "Seleccione color del título",
        value: "#ffffff"
       },
       {
        type: "hidden",
        id: "selectedWidget",
        value: id
       }
      ]
     };
   }
  }
```

And Manifest is empty JSON {}.

A complete list of configuration objects (with examples) is presented in "WiNE API Reference" Section.

6.7 Using SSSP or TEP API's in Your PIRS (content-driven only)

If you are Samsung SSSP/TEP Partner, you are most likely familiar with the concept that is this Chapter's title. But don't leave just yet, as we'll talk about how you can connect your existing SSSP/TEP Solution with PIRS – or rather, how you can turn your SSSP/TEP Application into a PIRS. However, if you are new to this topic, let us briefly explain it to you.

Samsung Smart Signage Platform Applications and Tizen Enterprise Platform Applications are applications that utilize various API's that give access and can communicate with the SoC of Tizen Screens. They are usually created within a tool called Tizen Studio, they must be signed with a proper certificate and compiled to a wgt package, and when it comes to using the API's, appropriate privileges must be added to the configuration file. When running on a Tizen screen, they somewhat imitate native applications behavior.

More information about SSSP/TEP can be found here: https://developer.samsung.com/smarttv/signage.

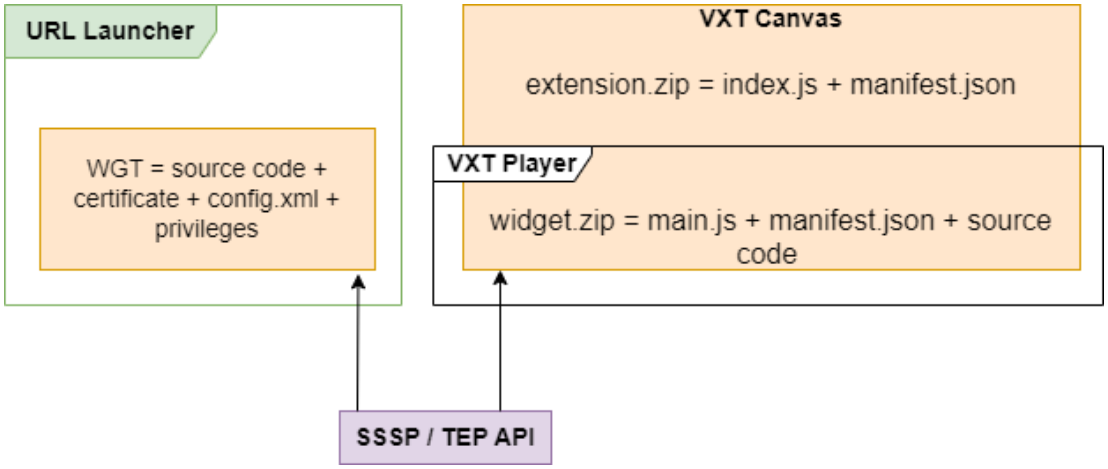
The main idea of using SSSP/TEP application as a PIRS is that you don't have to build the VXT application from scratch, even if you are using multiple SSSP/TEP API's. Usually, all it takes is:

- Testing, if the SSSP/TEP methods you wish to use are supported in VXT Player (which is where your PIRS will ultimately be displayed on LFD);
- Thinking about how you want the App to function within VXT Canvas UI, whether or not to use Settings and Property Panel – because now, your App will not only be visible on LFD, but also in our content creation tool – VXT Canvas, so the look and feel of all its components (settings, extension output, property panel) must be taken into consideration – not only the Widget;
- Adding a couple of WiNE API methods (create a channel, publish and subscribe)
- Remembering to observe the file structure (.zip archives, their composure and appropriate file naming)

It is important to note that even if you wish to use SSSP/TEP API's in your PIRS, you don't need to add any privileges or sign your App, or package it into a .wgt file – this part is already taken care of by VXT Player.

The concept of presenting SSSP/TEP in the context of PIRS Development can be summarized by the schema below:

Figure 52 SSSP/TEP vs PIRS.



6.8 What Happens Next

After you've successfully designed and developed your PIRS, it is time for you to:

- Upload your Application into VXT Developer Portal – you can learn how to do it by studying the "Testing Your Application" Section.
- Test your Application in VXT Canvas and VXT Player - you can learn how to do it and check our recommendations when it comes to best practices by studying the "Testing Your Application" Section.
- Prepare Test Cases and apply for SQA – you can learn how to do it by studying the "Navigating SQA Phases for Seamless PIRS Integration with Samsung VXT CMS" Section.

7. Testing your Application

Ensuring your VXT Application works as planned

7.1 Ways you can test your Application

There are various ways you can test your Application.

- If your App uses an embedded URL (for example, as an iframe), you can test in on a Screen by uploading it as a Web Content.
- On the topic of URL's, we also advise to test them in a web browser running the same web engine version as the target screen.
- You can also test your Application in VXT Canvas to validate its look and feel as a part of content creation system.
- In VXT Canvas, you can also Preview your Application, to obtain a rough idea of how your Application might look on a screen.
- You can also test your Application both on a virtual and a real screen.
- First and foremost, we'll discuss two methods of testing your PIRS Application as a whole:
- Through VXT Developer Portal.
- Through VXT Mock App using WiNE Development Server.

7.2 VXT Developer Portal

This guide outlines how to upload your Application into VXT Developer Portal to be able to test its Components in VXT Canvas and send the Application to Review to initialize the commercialization process for your PIRS.

> NOTE: This method will not allow you to test your Application in VXT Player.

7.2.1 Prerequisites

- Ensure Extension and Widget source files are zipped into two archives: extension.zip and widget.zip.

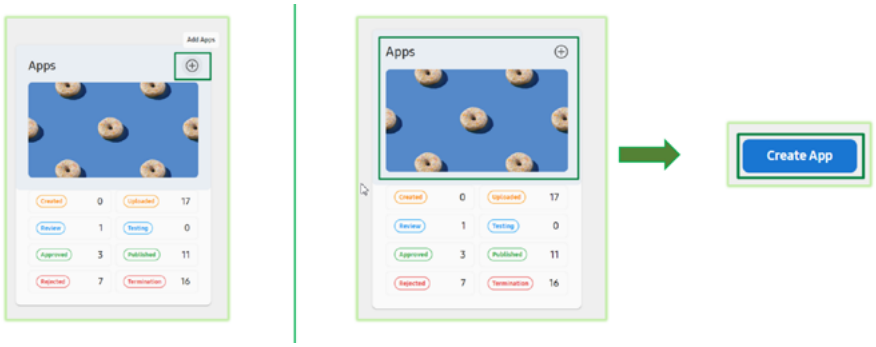
- Zip only the source files (for extension.zip: manifest.json and index.js; for widget.zip: manifest.json, index.html and other source files), not the 'extension' and 'widget' folders.

7.2.2 Step One – Uploading your Application to VXT Developer Portal

- Navigate to VXT Developer Portal (https://developer.samsungvx.com).
- Use the "+" button or click "Apps" and then click the "Create App" button:

Figure 53

Create App.



- Fill in all the required fields and click "Next":

Figure 54

App Info 1.

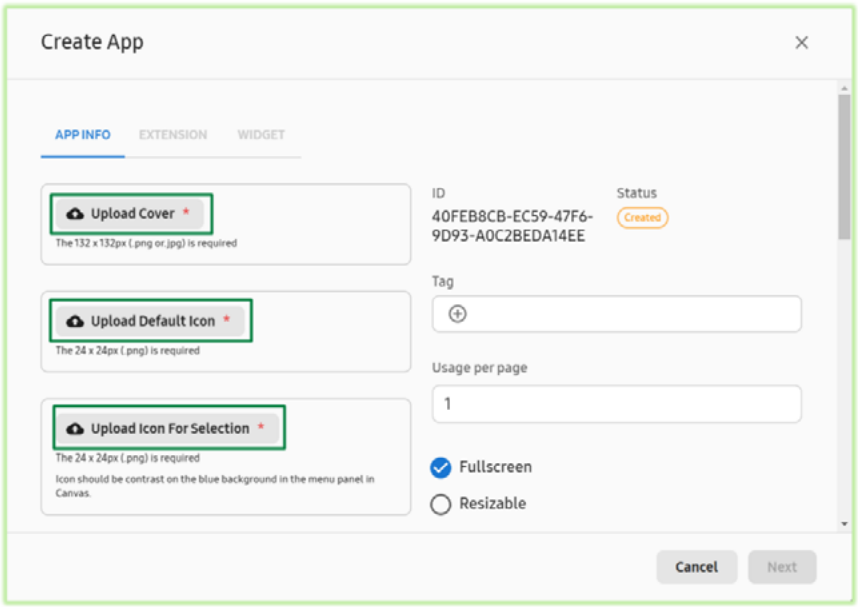


Figure 55

App Info 2.

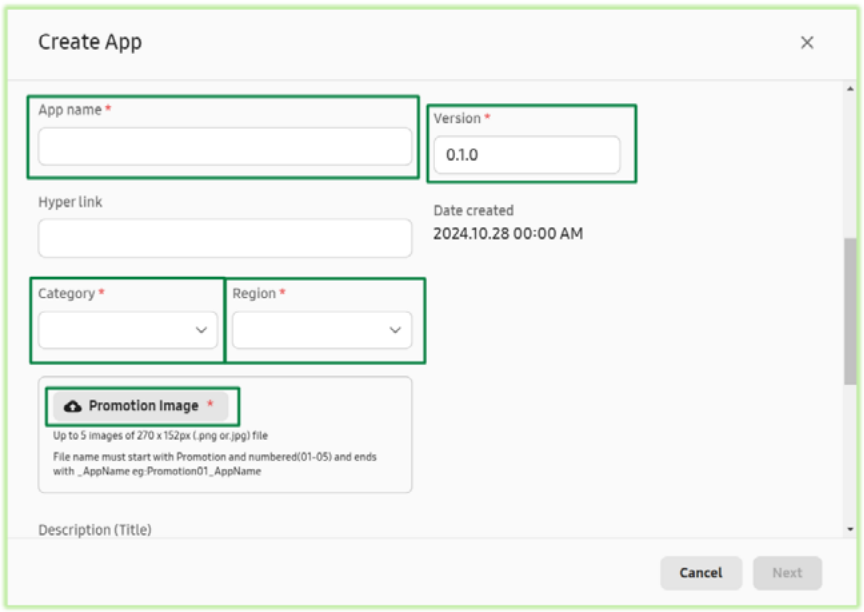
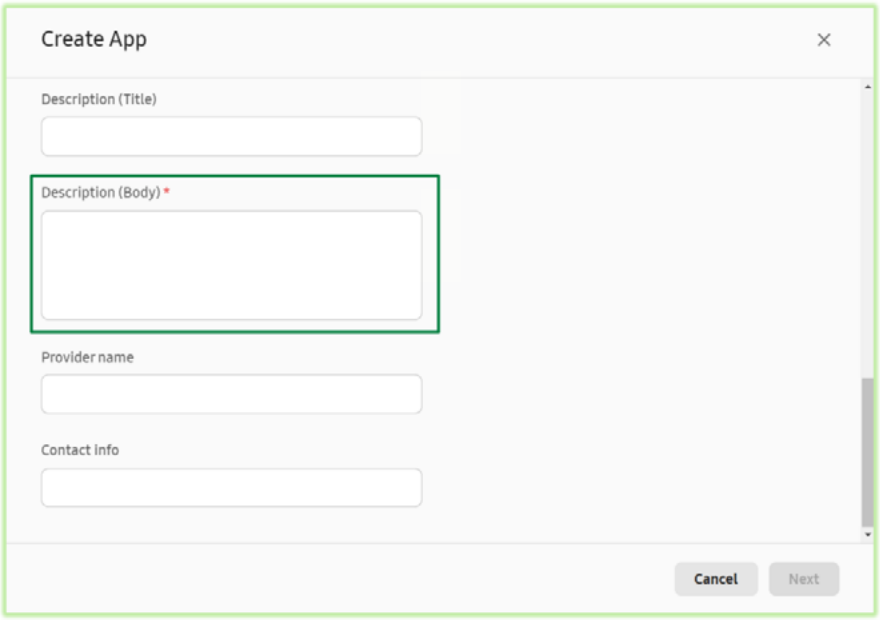


Figure 56

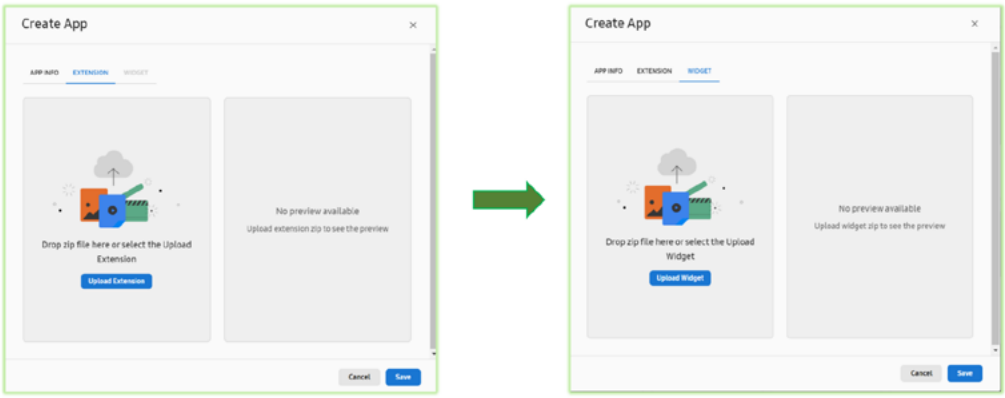
App Info3.



- Upload Extension (extension.zip) and Widget (widget.zip) and click “Save”:

Figure 57

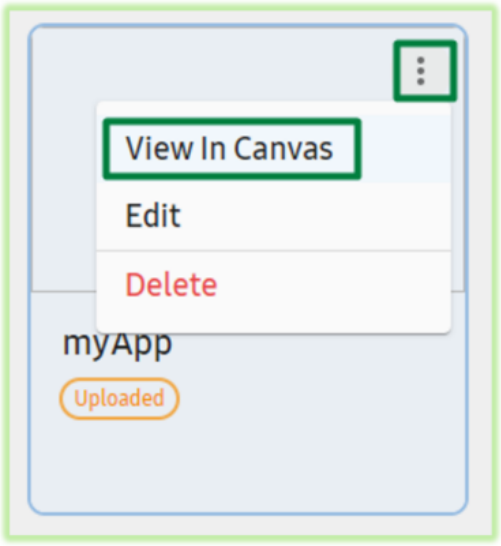
Upload extension.zip and widget.zip



- Locate your Application entry and find the “View In Canvas” option under “three vertical dots” button – this will allow you to see your Application in VXT Canvas (VXT Canvas will open in a separate browser window, allowing you to install your Application straight away):

Figure 58

View In Canvas.



> NOTE: Your Application will be installed and available only in the newly opened VXT Canvas window - it will not be accessible to the public, or in a separate VXT Canvas window.

7.3 WiNE Development Server for PIRS Applications

This guide outlines the key components and concepts for creating a Node.js server using Express to connect your Extension and Widget to VXT Mock Application, which will act as a “shell”, hosting your PIRS Application.

This method will allow you to test your Application in VXT Player in early stages of App development.

> NOTE: Watermark will be applied to the Widget (both in VXT Canvas and VXT Player).

7.3.1 Prerequisites

- Node.js is installed on your system.
- Chrome browser is installed on your system.

7.3.2 Step One - Project Setup

- Create a new project directory.
- Initialize a new Node.js project.
- Install required dependencies: Express and CORS.

7.3.3 Step One - Project Structure

Organize your project with the following structure:

```
app-dev-server/
├── public/
|   ├── extension/
|   |   ├── index.js
|   |   └── manifest.json
|   |   └── ...
|   └── widget/
|       ├── index.html
|       └── manifest.json
|       └── ...
├── config.json
├── package.json
└── server.js
```

7.3.4 Step One - Configuration Files

package.json

Ensure your **package.json** includes:

- Correct project name and description.
- Express and CORS as dependencies.
- Main entry point set to **server.js**.

```
{
  "name": "app-dev-server",
  "version": "1.0.0",
  "description": "Server for PIRS development",
  "main": "server.js",
  "scripts": {},
  "author": "SRPOL",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "path": "^0.12.7"
  }
}
```

config.json

Create a configuration file with the following settings:

- Port number.
- HTTPS flag.
- Allowed domains for CORS.

```
{
  "port": 3001,
  "https": false,
  "allowedDomains": ["*"]
}
```

7.3.5 Step One - Server Implementation (server.js)

Implement the following features in your server:

- **Express and Middleware Setup**
  - Initialize Express.
  - Set up CORS middleware with configuration from config.json.

- Implement a logging middleware for request tracking.

- **File System Operations**
  - Use the fs module to check for the existence of extension and widget folders.

- **Route Handlers**
  - Root endpoint (/): Return information about available Extension and Widget.
  - Extension endpoint (/extension): Return information about extension files.
  - Widget endpoint (/widget): Return information about widget files.

- **Static File Serving**
  - Serve static files from the public directory.

- **Server Creation and Startup**
  - Create an HTTP server.
  - Start the server on the specified port.

Example HTTP Server can look like presented below.

```
const express = require('express'),
  app = express (),
  cors = require('cors'),
  fs = require('fs'),
  path = require('path'),
  config = require('./config.json'),
  publicFolderPath = path.join(__dirname, 'public'),
  extensionFolderPath = path.join(publicFolderPath, 'extension'),
  widgetFolderPath = path.join(publicFolderPath, 'widget');

const corsConfig = {
  origin: config.allowedDomains,
};

app.use(cors (corsConfig));

app.use(function (req, res, next) {
  console.log(`${req.method} ${req.url}`);
  next();
});

app.get('/', (req, res) => {
```

```
  const appStructure = {};

  if (fs.existsSync(extensionFolderPath)) {
    appStructure.extension = true;
  }
  if (fs.existsSync(widgetFolderPath)) {
    appStructure.widget = true;
  }

  res.json(appStructure);
});

app.get('/widget', (req, res) => {
  const widgetSrcPath = path.join(widgetFolderPath, 'index.html');
  const widgetManifestPath = path.join(widgetFolderPath, 'manifest.json');

  if (fs.existsSync(widgetFolderPath) && fs.existsSync(widgetSrcPath)) {
    const config = { path: 'widget/index.html' };
    if (fs.existsSync(widgetManifestPath)) {
      config.manifest = require(widgetManifestPath);
    }
    res.json(config);
  } else {
    res.status(404).end();
  }
});

app.get('/extension', (req, res) => {
  const extensionSrcPath = path.join(extensionFolderPath, 'index.js');
  const extensionManifestPath = path.join(
    extensionFolderPath,
    'manifest.json',
  );

  if (fs.existsSync(extensionFolderPath) && fs.existsSync(extensionSrcPath)) {
    const config = { path: 'extension/index.js' };
    if (fs.existsSync(extensionManifestPath)) {
      config.manifest = require(extensionManifestPath);
    }
    res.json(config);
  } else {
    res.status(404).end();
  }
});
```

```
app.use(express.static(path.join(__dirname, 'public'), { index: false }));

const server = require('http').createServer(app);
const port = config.port;

server.listen(port, () => {
  console.log(
    `WiNE application development server listening on port ${port}`,
  );
});
```

7.3.6 Step Two - Opening Chrome with Disabled Security

In order for VXT Mock Application to be able to fetch served Extension and Widget files, Chrome browser needs to be launched with disabled security.

Create a temporary folder for Chrome (for example “ChromeDev”) and then run the following command:

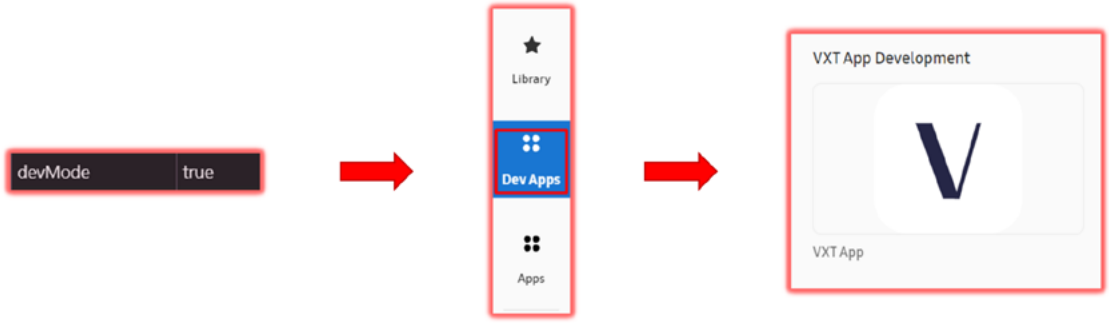
- **For Windows:**
  “C:\Program Files\Google\Chrome\Application\chrome.exe” --disable-web-security --disable-site-isolation-trials --allow-running-insecure-content --user-data-dir="%LOCALAPPDATA%\ChromeDev"
- **For MacOS:**
  open -n -a "Google Chrome" --args --disable-web-security --disable-site-isolation-trials --allow-running-insecure-content --user-data-dir="/tmp/ChromeDev“
- **For Linux:**
  google-chrome --disable-web-security --disable-site-isolation-trials --allow-running-insecure-content --user-data-dir="/tmp/ChromeDev"

Once you’ve successfully opened Chrome with disabled security flags, proceed to VXT CMS (https://samsungvx.com).

7.3.7 Step Three – Using VXT Mock App

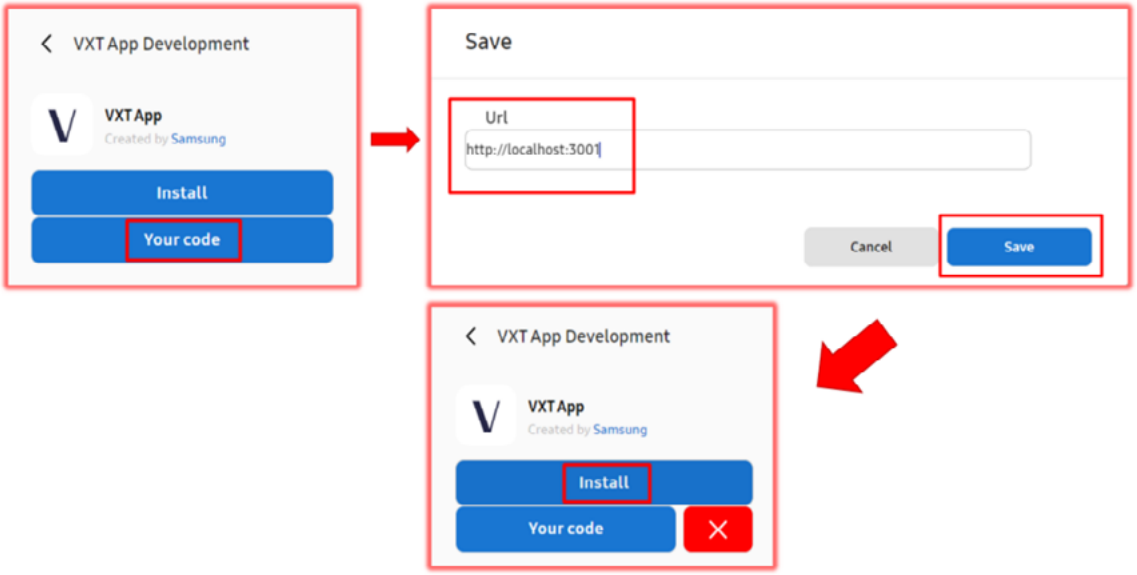
- **Enable Developer Mode:**
  - Open VXT Canvas.
  - Press Ctrl+Shift+F or open developer tools and add the following key-value pair in application local storage (“devMode”: true) and refresh page.
  - In Dev Apps, you should see VXT App:

Figure 59 Find VXT Mock App.



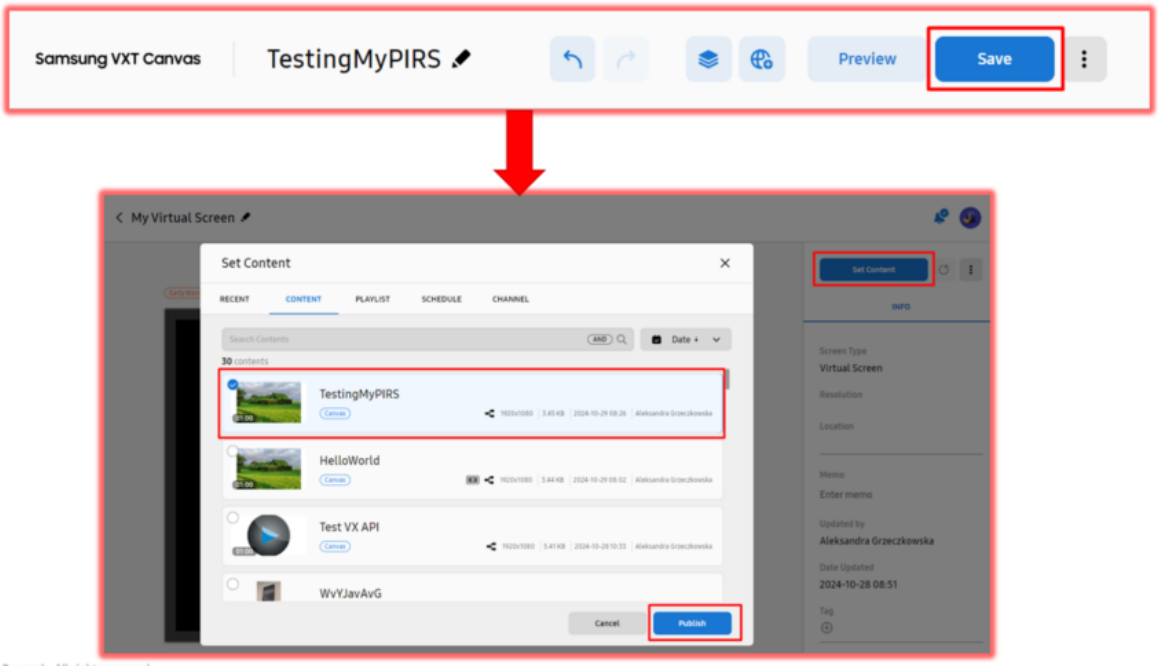
- **Connect your code to VXT App:**
  - Enter VXT App and click “Your code”.
  - Provide base URL to your local server and click “Save”.
  - VXT Canvas will reload.
  - Find VXT App again and click “Install”:

Figure 60 Insert Your Code into VXT Mock App Shell.



- **Test your App on a screen:**
  - Your target App, hosted on provided address will be installed.
  - Create content using your App and save it.
  - Publish created content to your screen:

Figure 61 Test Your Application on a Screen.



7.3.8 Key Concepts for WiNE Development Server and VXT Mock App Usage

- **File serving**
  - Endpoints should serve both the path to the main files (index.js for extension, index.html for widget) and their corresponding manifest files.

- **Flexibility**
  - Server should work regardless of whether both extension and widget are present, or just extension.

- **Compatibility**
  - Tizen/Android/Windows Players use HTTP.
  - o Virtual Screen opened in Chrome launched with disabled security can use HTTP.
  - o For using Virtual Screen opened in Chrome launched without disabled security, HTTPS might be necessary.

- **Network**
  - Ensure both your server PC and the screen are in the same subnet.
  - For testing on a physical device, make sure you provide the IP address of the server, not “localhost”.

- **Watermark**
  - A watermark will be applied to your Widget, both in VXT Canvas and VXT Player:

Figure 62

"Samsung Demo" Watermark.

7.4 Testing your Application directly in VXT Canvas

In VXT Canvas, you can preview all PIRS features, making it a vital part of the testing process. In order to make Users want to purchase a license for your Application, you must pay close attention to how your Application (as a whole – not only the Widget) looks and functions in VXT Canvas, because this is where the journey starts.

Luckily, testing your Application in VXT Canvas is very easy and can be done both via uploading your App to VXT Developer Portal, as well as via WiNE Development Server and VXT Mock Application.

Table 30

Advantages and Disadvantages of Using VXT Developer Portal for Testing Your App in VXT Canvas.

Using VXT Developer Portal For Testing Your App In VXT Canvas	
Advantages	Disadvantages
No watermark added to the Widget.	If bugs are found and code is updated, you must re-zip the archives and edit the Application entry in VXT Developer Portal, which can be time-consuming.
Possibility to test, how other Components are displayed (cover logo, icons, App name, App description, etc.).	

Table 31

Advantages and Disadvantages of Using WINE Development Server and VXT Mock App for Testing Your App in VXT Canvas.

Using WINE Development Server and VXT Mock Application for Testing Your App In VXT Canvas	
Advantages	Disadvantages
If bugs are found and code is updated, the changes can be easily implemented and seen in VXT, without the need to zip and re-upload the source code files (though a page refresh/server restart might be necessary).	Watermark is added to the Widget.
	You cannot test other Components (cover logo, icons, App name, App description, etc.), just the code.

Here are some key points to consider:

User Experience Testing:

- Ensure you test the user experience thoroughly. We advise you walk through the App with the mindset of an inexperienced User who sees your App for the very first time. Check if your entire PIRS App looks appealing – not only the Widget, but also Settings and Property Panel.
- Make sure your Extension reacts correctly to User's selections in Settings.
- Check responsiveness of your Widget and make sure it scales correctly to the size of canvas area.
- Check your backend communication with the VXT Widget to ensure seamless data flow and interaction.
- Make sure your Widget reacts correctly and seamlessly to User's selections in Property Panel.
- Check how your Application looks in Preview mode.

7.5 Testing your Application on a VXT Virtual Screen

Before testing your Application on a real device, it is highly recommended to test it on a VXT Virtual Screen. This way, you can pre-determine how your Application behaves in VXT Player, leveraging the use of Developer Tools. Testing your Application on a Virtual Screen allows you to debug and resolve any issues before deploying the App to physical screens.

Table 32

Advantages and Disadvantages of Using Virtual Screen for Testing Your App in VXT Player.

Using VXT Virtual Screen For Testing Your App in VXT Player	
Advantages	Disadvantages
Access to Developer Tools.	It cannot be assumed that the results achieved on the virtual screen will be identical to those achieved on a target device (for example, due to differences in Chromium version, which can be older on real devices).
You don't need a physical device to test your App in VXT Player – Virtual Screen is launched in a browser.	

7.5.1 Steps for Testing

- **Add a Virtual Screen to your VXT environment:**

Figure 63

Add Screen.

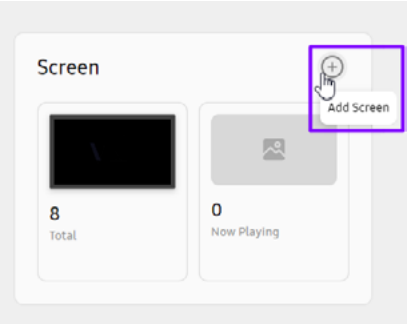


Figure 64

Open VXT Player in Your Browser.

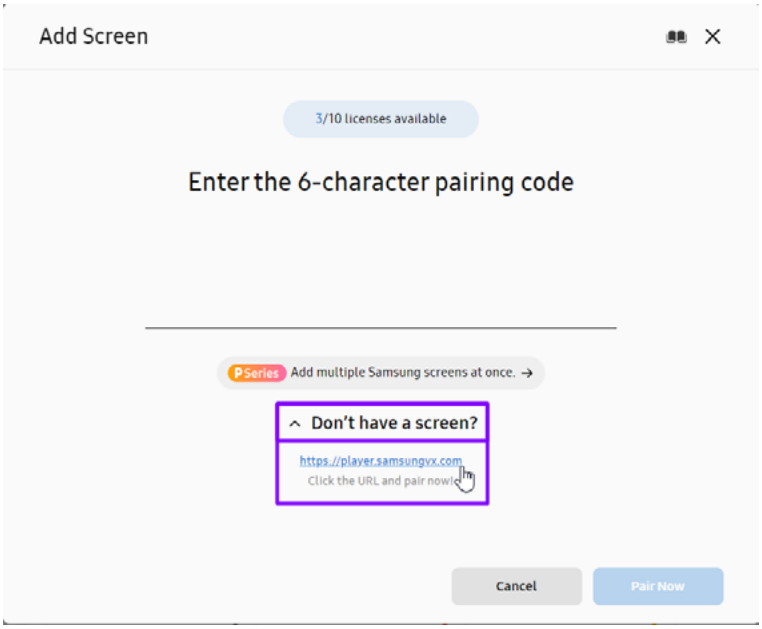
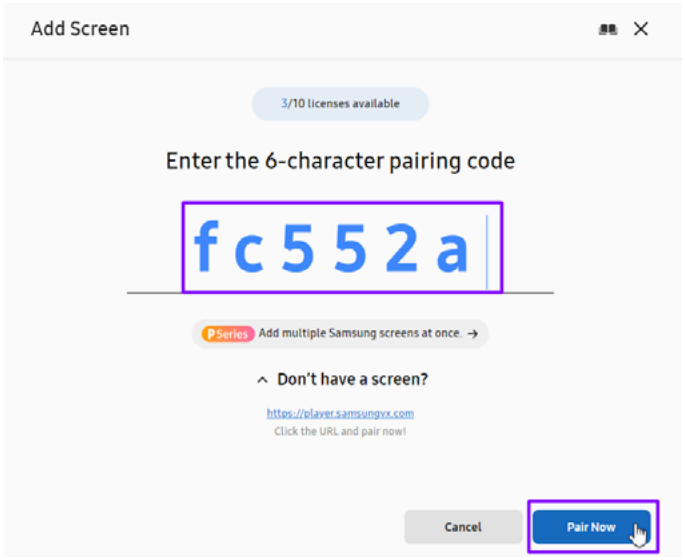


Figure 65

Get Pairing Code.

Figure 66

Insert Pairing Code and Pair Your Virtual Screen.



- **Deploy Your Application:**
  - If your App uses an embedded URL (for example, as an iframe), you can test in on a Screen by uploading it as a Web Content:

Figure 67

Enter Content Menu.

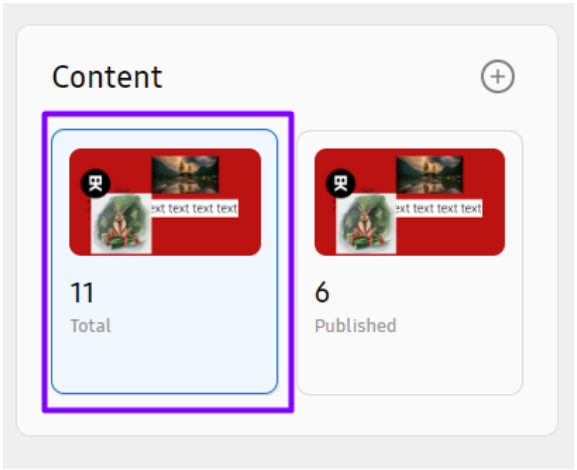


Figure 68

Add Content.

Figure 69

Add Your Web Content (URL).

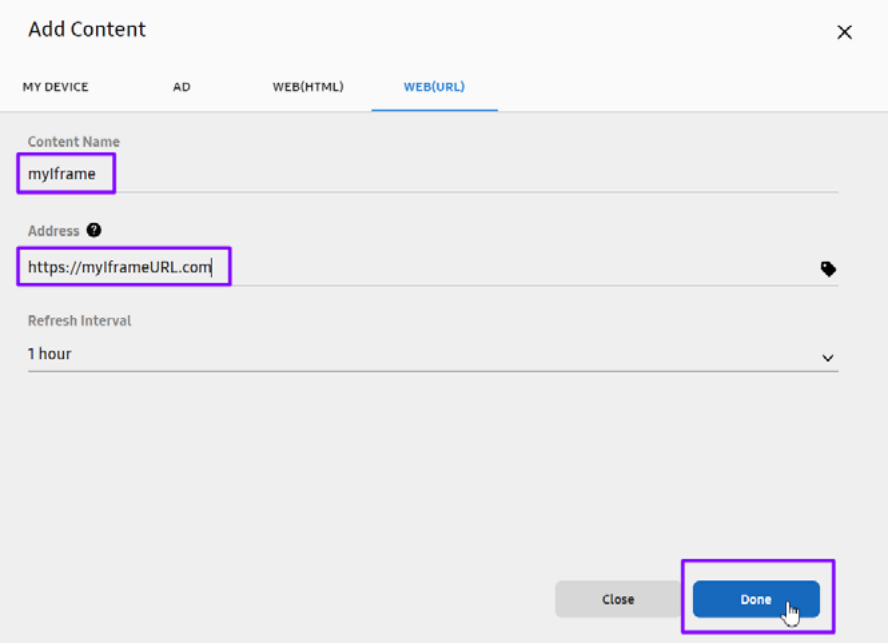
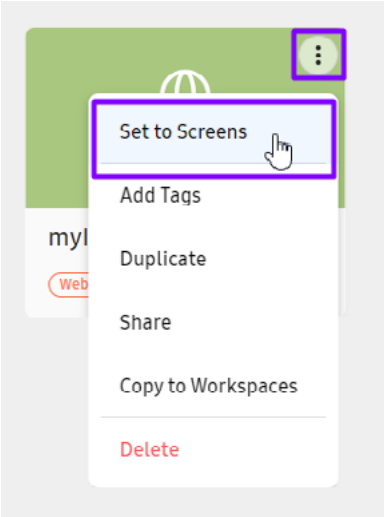


Figure 70

Set the Web Content to Screen.



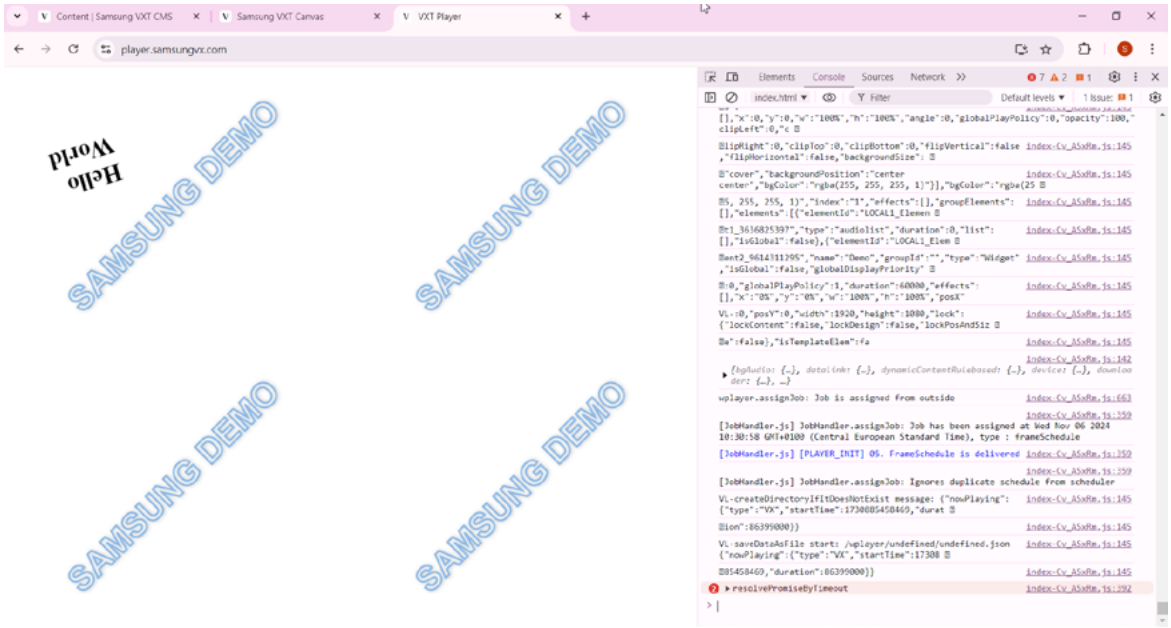
> NOTE: Using this method you can test only a part of your Application. To achieve more reliable results, testing your PIRS as a whole is necessary - it can be done by following the method described below.

- For comprehensive PIRS testing, use WiNE Development Server and VXT Mock Application:

Follow the steps provided in “WiNE Development Server for PIRS Applications” Chapter.

- **Access DevTools in Virtual Player:**
  - Open DevTools in your browser where Virtual Screen is running. This can be done by right-clicking on the virtual screen in your browser and selecting “Inspect” or by pressing Ctrl+Shift+I (Windows) or Cmd+Option+I (Mac):

Figure 71 Virtual Screen and Developer Tools.



- **Test Application Functionality:**
  - Navigate through your application on the virtual screen as if you were using a physical device. Interact with all features, and ensure everything behaves as expected. This step is vital for verifying the functional aspects of your application.
  - Debug issues using Developer Tools:
    - Elements Panel: Inspect and modify the DOM and CSS. This is useful for checking the structure and styling of your application.
    - Network Panel: Monitor network requests and responses. This is crucial for ensuring that your application is making and receiving the correct network calls.
    - Sources Panel: View and debug JavaScript. You can set breakpoints, step through code, and watch variables to understand the flow of your application.
    - Application Panel: Inspect storage, cookies, and other application-related data.

- **Optimize Performance:**
  - Use the Performance Panel to record and analyze your application's performance. This tool provides insights into how your application executes over time, highlighting areas that may cause slowdowns or inefficiencies.
  - You can start a performance recording by clicking the "Record" button and then interacting with your application. Stop the recording to view a detailed breakdown of where time is being spent, such as in rendering, scripting, or network requests. This helps you identify performance bottlenecks.

- **Confirm Resolution:**
  - After making changes based on your testing and debugging, re-test your application to ensure all issues have been resolved. Verify that the application functions optimally under various conditions. This step involves thorough testing to confirm that your fixes and optimizations have been successful.

7.6 Testing your Application on your Physical Device

After testing your Application on a Virtual Screen, it is now time to validate the results on real devices. Running your Application on a target device is the only way to determine how your Application will behave when it's live.

Ensuring the functionality of your VXT PIRS on a physical device is an indispensable phase of the development lifecycle due to the variations in web engine versions across different devices, which can significantly impact the behavior of your application. These discrepancies in web engine versions affect rendering, performance, and even feature availability, which can result in inconsistent user experiences across devices. Ensuring your application is compatible with these diverse environments is vital to deliver a seamless and reliable user experience.

7.6.1 Steps for Testing

Table 33

Advantages and Disadvantages of Using Real Device for Testing Your App in VXT Player.

Testing your Application on a real Device	
Advantages	Disadvantages
Reliable results.	No access to Developer Tools.

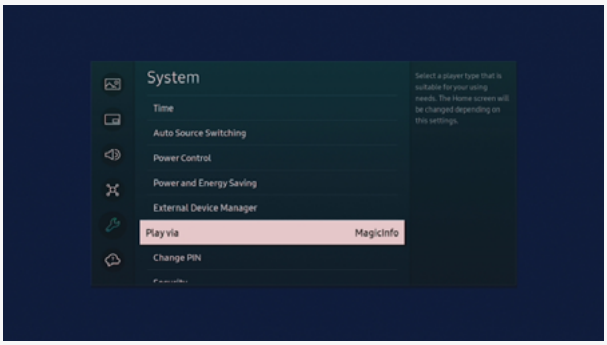
- **Add your Screen to VXT environment:**
  - Tizen 4.0 ~ 6.0

Table 34

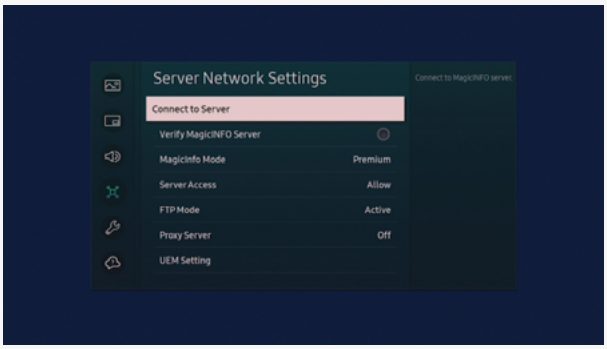
Adding Tizen 4.0-6.0 Screen to VXT.

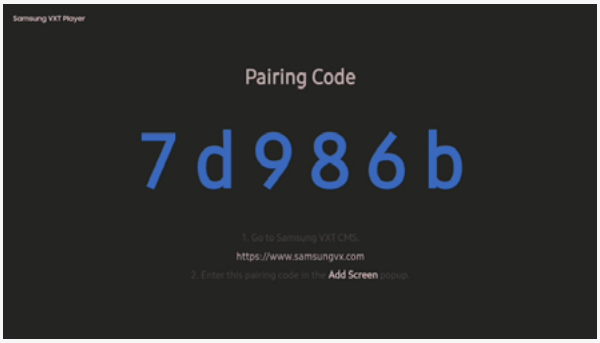
	Tizen 4.0~6.0
Preconditions	Ensure your Tizen 4.0 screen is connected to a network without IPv6.

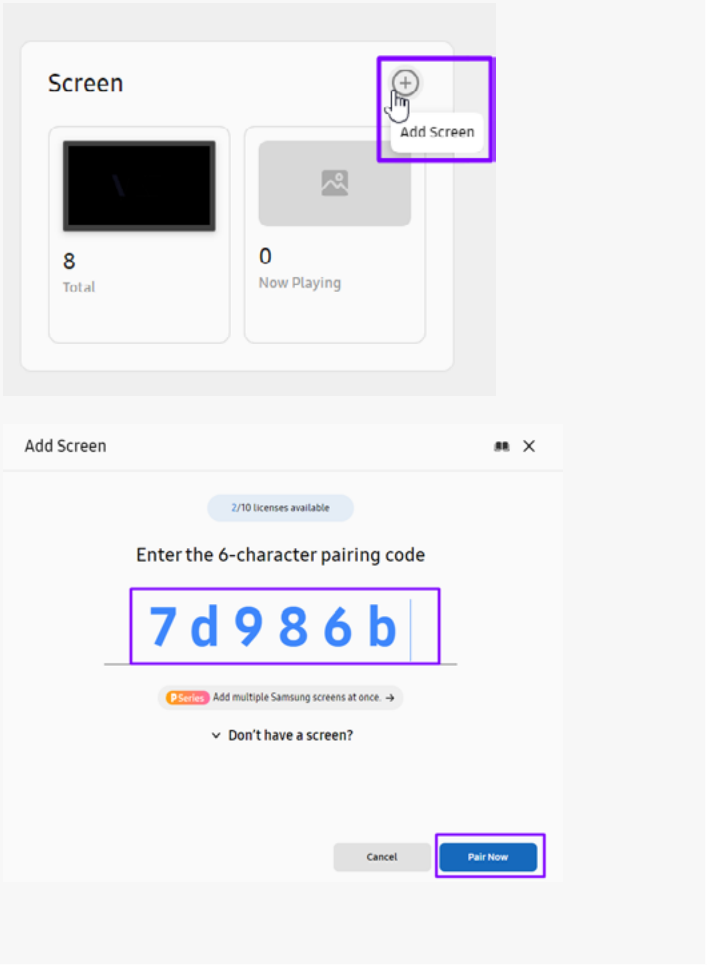
Navigate to [MENU] Menu / System and configure as follows:
- Play via: MagicINFO



Navigate to [MENU] Menu / Network / Server Network Settings and configure as follows:
- MagicInfo Mode: Premium
- Access to Server: Allow



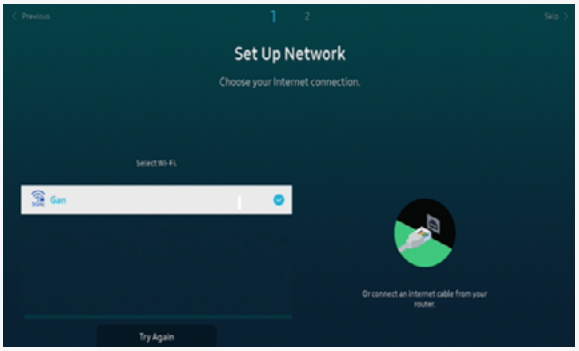
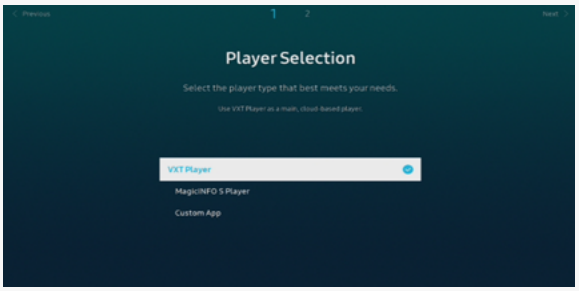
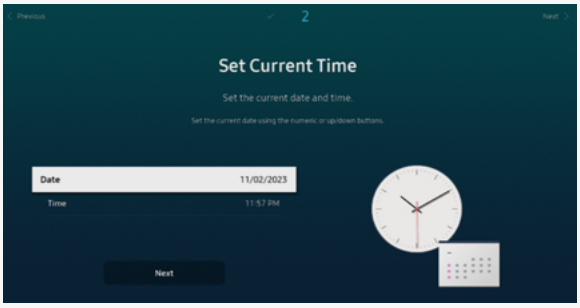
Tizen 4.0~6.0	
Installation Manual	**Navigate to [MENU] Menu / Network / Server Network Settings / Connect to Server and configure as follows:** • Address: rm.samsungvx.com • TLS: use • Port: 443  **Navigate to [HOME] Home / Magic INFO Player / Network Channel.** Wait for the VXT Player to initialize. A Pairing Code should be displayed. 

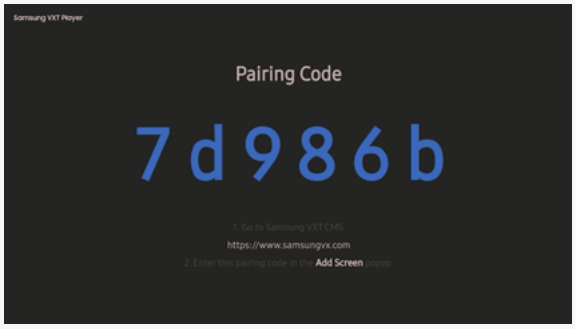
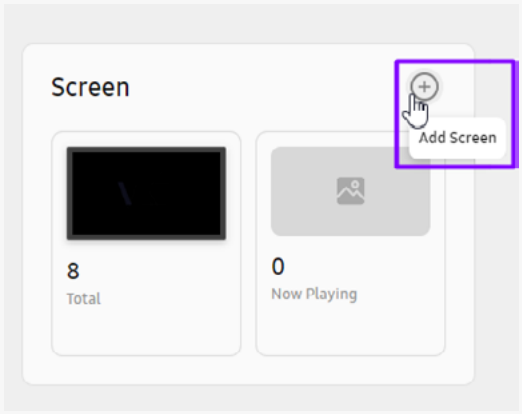
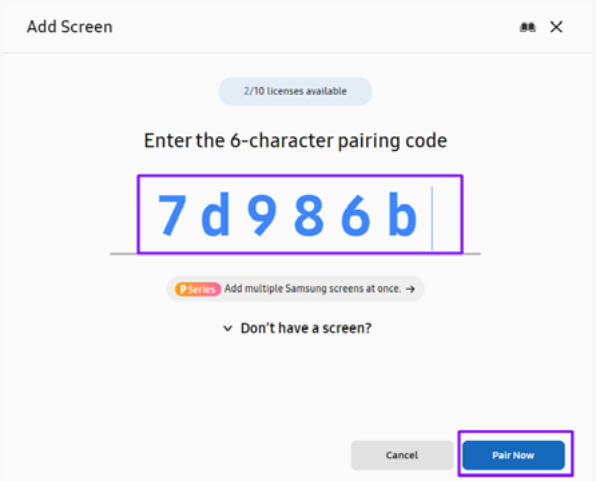
Tizen 4.0~6.0	
Screen Pairing	

- Tizen 6.5+

Table 35

Adding Tizen 6.5+ Screen to VXT.

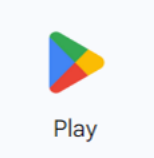
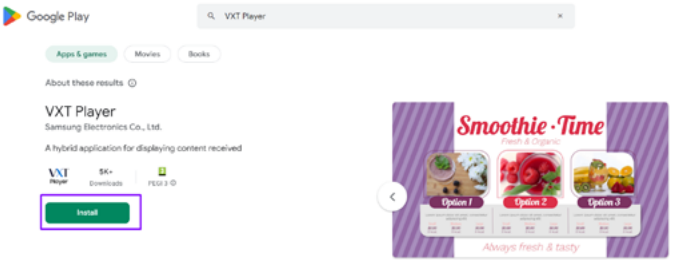
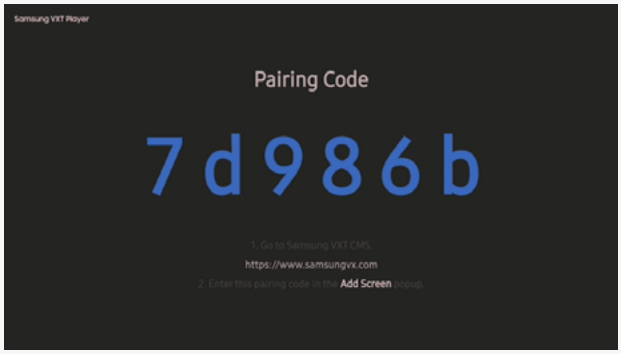
Tizen 6.5+	
Preconditions	Complete Initial Setup: • Make sure your device is connected to network  • Accept T&C  • Select VXT Player  • Make sure your time settings are correct 

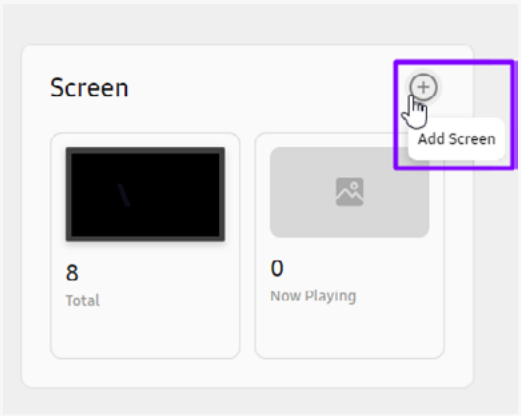
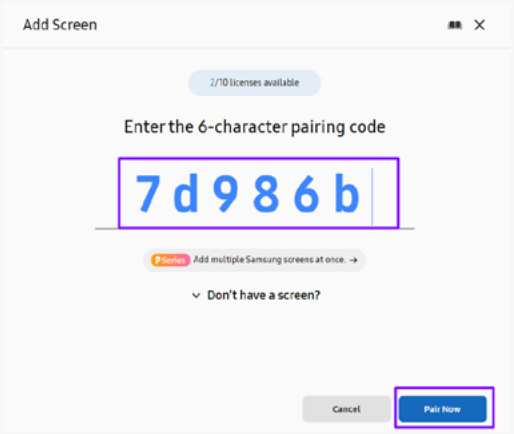
Tizen 6.5+	
Installation Manual	Launch the VXT Player app (if it hasn't launched automatically) by clicking the [HOME] button on your remote and selecting "VXT Player". A Pairing Code should be displayed.  NOTE: If during Initial Setup "Custom App" was chosen, you can download VXT Player App from Apps Marketplace on your device. NOTE II: If during Initial Setup you cannot see the "VXT Player" option, device firmware update is required.
Screen Pairing	 

- Android Player

Table 36

Adding Android Screen to VXT.

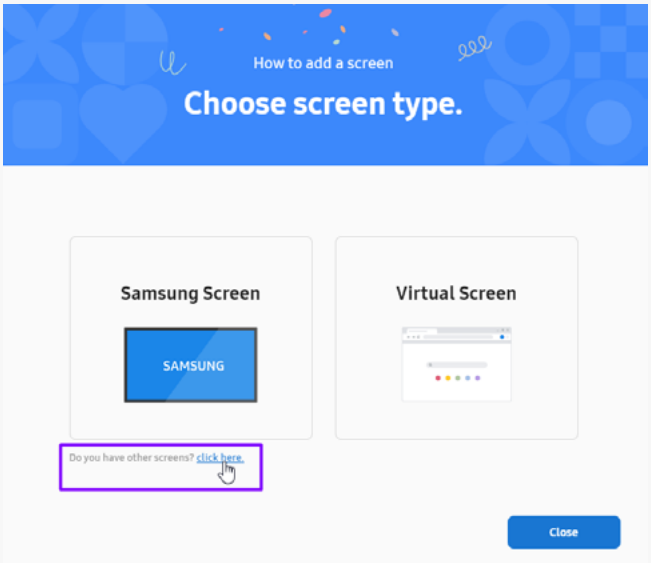
Android Player	
Preconditions	Ensure your Android version meets the specifications (version 10.0+).
Installation Manual	Open Google Play Store on your Android device.  Search for "VXT Player" and download the app.  Launch the VXT Player app. A Pairing Code should be displayed. 

Android Player	
Screen Pairing	 

- Windows Player

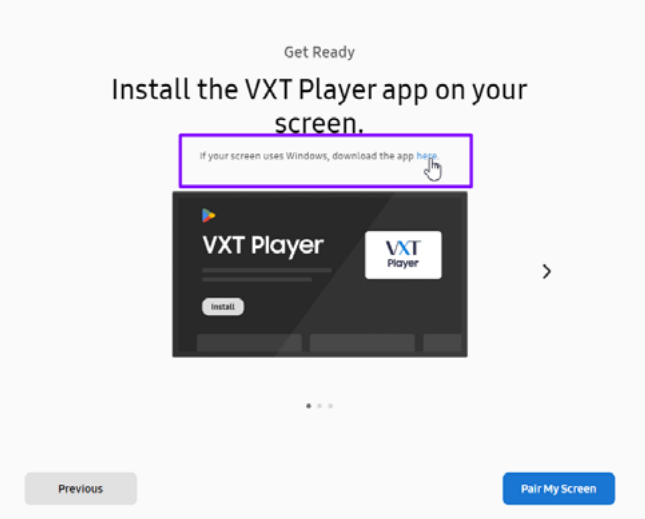
Table 37

Adding Windows Screen to VXT.

Windows Player	
Preconditions	Ensure your Windows device and version meets the specifications (version 10+).
Instalation Manual	Open Chrome browser on your Windows device and go to VXT CMS (https://samsungvx.com) Navigate to Add Screen / Tutorial  Click the "click here" hyperlink. 

Windows Player

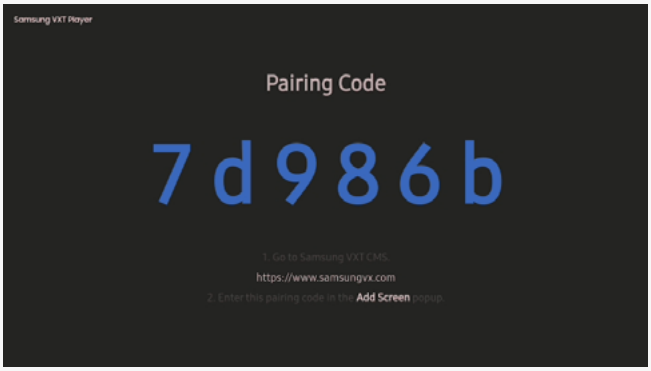
Download VXT Player for Windows installer by clicking the “here” hyperlink.

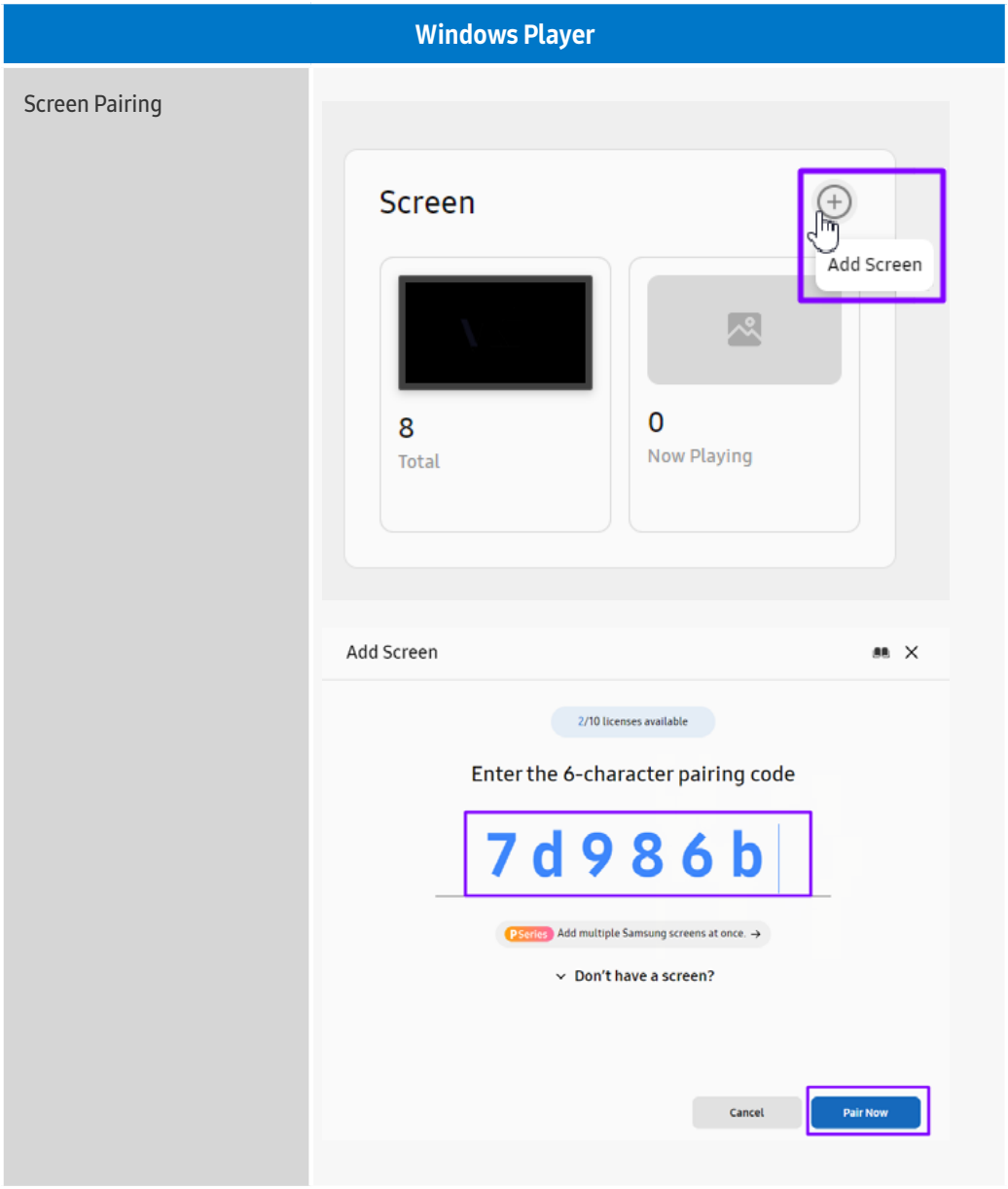


Install VXT Player and grant appropriate permissions when requested.

Launch the VXT Player app.

A Pairing Code should be displayed.





- **Deploy Your Application:**
  - If your App uses an embedded URL (for example, as an iframe), you can test in on a Screen by uploading it as a Web Content:

Figure 72

Enter Content Menu.

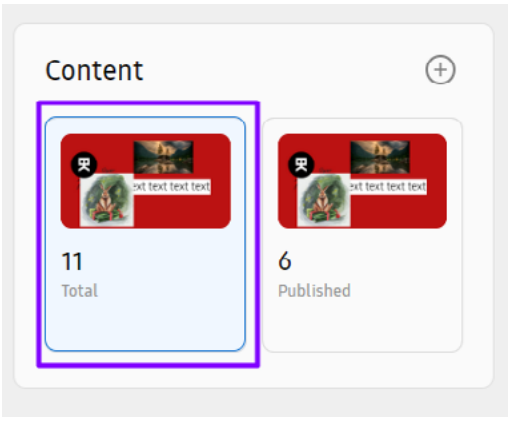


Figure 73

Add Content.

Figure 74

Add Your Web Content (URL).

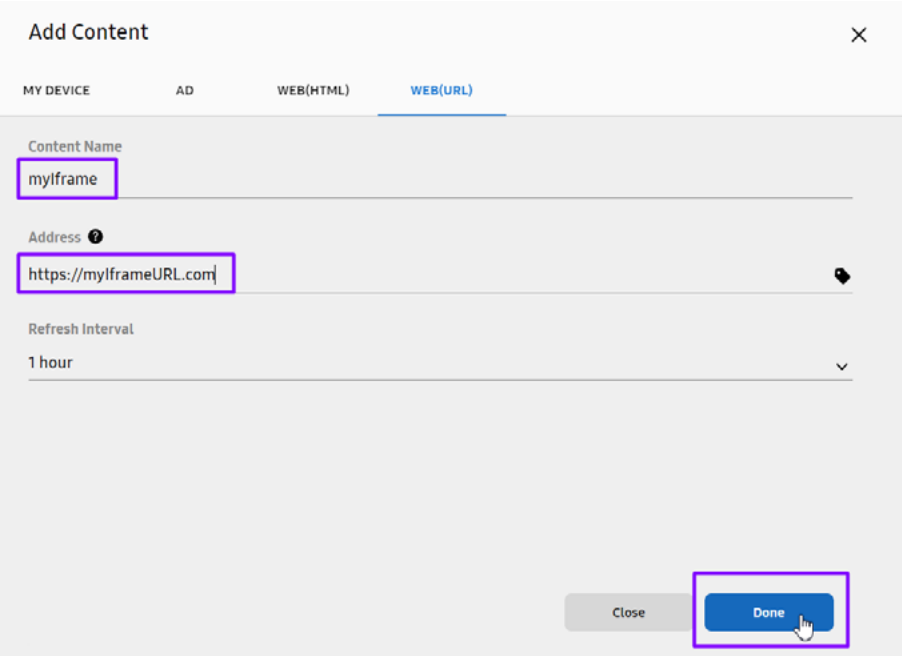
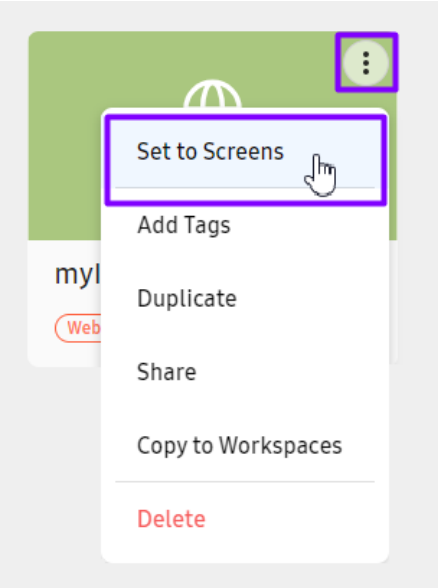


Figure 75

Set the Web Content to Screen.



> NOTE: Using this method you can test only a part of your Application. To achieve more reliable results, testing your PIRS as a whole is necessary - it can be done by following the method described below.

- For comprehensive PIRS testing, use WiNE Development Server and VXT Mock Application:

> NOTE: Make sure your development PC and the physical device are on the same subnet to allow seamless communication.

Follow the steps provided in "WiNE Development Server for PIRS Applications" Chapter.

- **Test Application Functionality:**
  - Interact with your application on the physical device, just as an end-user would. This includes navigating through menus, using interactive features, and performing tasks that your application supports.
  - Pay attention to how the application renders and performs on the actual hardware.
  - Record the screen to document issues. Use these recordings to analyze and identify any problems, as well as for future reference.
- **Optimize Performance:**
  - Make necessary adjustments to improve rendering speed and responsiveness.
  - Use screen recordings to compare results before and after tweaking Application's performance.
- **Verify Consistency Across Devices:**
  - If you have access to multiple physical devices with different web engine versions, repeat the testing process on each device.
  - Ensure your application behaves consistently across all target environments.

NOTE: When it comes to Samsung Screens, paying close attention to web engine version is crucial, as some modern JavaScript and CSS functionalities might not work on all the Screens, and therefore - the end result might be far from perfect. For more information, please study the table below:

Table 38
Web Engine Versions.

Tizen Version	Web engine Version
Tizen 7.0	Chromium M94
Tizen 6.5	Chromium M85
Tizen 6.0	Chromium M76
Tizen 5.5	Chromium M69
Tizen 5.0	Chromium M63
Tizen 4.0	Chromium M56

7.7 Best Practices

Ensuring a high-quality user experience through meticulous testing is essential for developing reliable and efficient Applications. This Chapter provides an overview of the testing requirements necessary to align with Samsung's standards, ensuring compatibility and optimal performance across Samsung devices. Through comprehensive testing strategies and environments, developers can address potential issues early, resulting in a smoother and more satisfactory user experience. Moreover, submitting a well-tested PIRS Application is more likely to get through the SQA process smoothly.

Below you can find a list of best practices when it comes to PIRS development and testing:

- **Process**
  - Understand that thorough testing on your end can significantly shorten and streamline the SQA process.
  - Begin testing early in the development lifecycle.
  - Maintain consistent testing throughout the development process.

- Continuously refine the application based on feedback.
- Keep detailed records of issues and their resolutions.

- **Testing in VXT Canvas**
  - Pay attention to Settings and Property Panel elements' placing, ensuring a user-friendly and clean UI.
  - Document use cases for each user scenario, covering all potential paths a User might take. This documentation helps in identifying and resolving issues quickly.
  - Don't rely too much on the Preview in VXT Canvas – it can give you a rough idea of how your App might look on a screen, but it will never be a substitute of testing your App on a real device.
  - Leverage DevTools (both in VXT Canvas and in Virtual Screen) for debugging and optimization.
  - In your Extension's code, apply settings, orientation and search query filtering, ensuring that only filtered content and categories are published.

- **Testing on a Screen**
  - Test on actual hardware for comprehensive validation.
  - Test on various devices with different web engine versions.
  - Ensure uniform functionality across different devices.
  - Optimize application for improved speed and responsiveness.
  - In your Widget's code, implement correct handling of all playstates.

8. Navigating SQA Phases for Seamless PIRS Integration with Samsung VXT CMS

Proceeding through the SQA Process

8.1 Minimum Application Requirements to Qualify for SQA

Development of a new PIRS is both a streamlined but also grand undertaking that requires multiple efforts from your Developers and Designers alike. On one side, WiNE API aims to simplify the process and get you developing with minimum delay. On the other hand, development for Samsung VXT CMS is intertwined with select hardships, and limitations as in all professional software engineering projects. Ultimately, the performance of your PIRS (both visual and actual) will be a direct result of these combined efforts.

Samsung strives to make End User satisfaction from using Samsung VXT a best-in-class experience that dwarfs that of any other Smart Signage Platform on the Market. As a result, it is key that all PIRS meet certain requirements for operability and performance prior to being accepted, and officially launched directly within Samsung VXT CMS.

The SQA Process explained further below is intended as a gateway to ensuring PIRS fulfill a level of quality that aids satisfying User expectations, both when using and configuring them within VXT Canvas, as well as later when deploying them to VXT Players (on multiple Screen types).

Before you submit your PIRS for processing through Samsung SQA, you need to be aware of what Samsung and its Testers will seek and scrutinize through various forms of testing. In doing so they will act to identify key parts and behavior of your PIRS, which may lessen an otherwise great End User experience.

Prior to submitting your PIRS to Samsung for SQA, you will need to ensure adequate testing has been performed on your side, and that the version you are submitting is mature in nature, and does not break the natural User flow in any meaningful way through its normal use. In particular, you should focus your efforts on validating all areas that the User is likely to encounter through general use, either in separate tests, or as a series of tests, whereby the next test relies on the result

of the one prior to it. This is especially true if your VXT App has multiple configuration options, and a User is more than likely to utilise them in combination with each other. Samsung recommends you emphasise your efforts on the following key areas of your PIRS:

- How fast your App (Extension and Widget) loads in VXT Canvas
- Whether your Settings Pane, Property Panel is rendered correctly in VXT Canvas
- If your Widget is displayed correctly in Canvas, and zooms in/out responsively
- How your Widget loads in VXT Player on multiple Screen Platforms
- How your Widget displays across various Tizen versions
- Whether there are substantial delays when transitioning between Pages in a VXT Canvas Project containing multiple Widgets
- Ensuring a satisfactory UX/UI experience

The list above is not comprehensive, and merely aids to highlight key points where you should concentrate your attempts. Samsung understands that it is not possible to identify all potential pitfalls in your Application in the first batch of testing and evaluation. Samsung SQA Teams will also assist you in locating potential issues through the SQA Process, after which you will have the opportunity to rectify them by your Team. The more mature a version of your App prior to submitting it for Samsung SQA, the fewer issues will be communicated back to you.

8.2 The SQA Process explained

The Samsung SQA Process is a series of phases designed to identify key issues in how your PIRS behaves both directly within VXT Canvas, as well as on VXT Player. Each Phase is individual, meaning it is fully isolated from each other. Altogether, there are three Phases, and each subsequent Phase aims to improve your App's performance to a greater degree. Ultimately, once all three are completed, the expected outcome is a version that qualifies for official release on Samsung VXT CMS.

All Samsung SQA Phases are performed by Samsung Testers within the same region where your Company is registered, thus where your App is submitted for qualification. In some cases, you may already be

familiar with them through having exchanged prior communication. If this is your first PIRS SQA submission, you will discover their names in due course, for throughout the SQA Process you will be in constant communication with them as they identify issues in your App, and you provide feedback from your attempts to fix them. Samsung aims to make the SQA Process personal, meaning you will not be sending and receiving emails to/from generic mailboxes. If a select issue(s) are grand enough to require substantial clarification, a Samsung Tester or Engineer may elect to convey this information to you over a Conference Call. This is intended to assist you in overcoming major concerns and obstacles in your PIRS. In no circumstances should you feel left to face problems on your own.

Each Phase is designed to deliver key outcomes in accordance to its weighting within the SQA Process. Take a moment to study what each Phase involves, and what it strives to achieve.

- **Phase 1**

  This marks the first time you have submitted your new PIRS for Samsung SQA. This version is expected to be mature, and after already having undergone a series of in-house tests by your Team. When submitting your PIRS for Phase 1, you will also need to supply a document listing your Test Cases and Results from your own testing. Samsung Testers will repeat these same Tests, and add additional ones designed to address areas that your Test Cases overlooked, or did not advance enough for their results to appear reliable.

  Phase 1 will last circa Two Weeks, after which Samsung will email you a new list of Test Cases & Results along with feedback on outstanding issues. You may expect Samsung to identify a variety of issues from this Period. If your PIRS contains many Widgets and/or configuration options (within Settings, or the Property Panel), the list of encountered issues is expected to extend beyond normal.

  Once Samsung communicates this feedback back to you, your Team will also have a few weeks to go over them, and fix them based on description.

- **Phase 2**

At this point, your App has already been identified to contain a number of issues that needed addressing. It also commences once you are confident to have fixed them, and communicated this information back to the Samsung SQA representative.

Phase 2 will also last about Two Weeks, and Test Cases that proved cumbersome to overcome will once again be re-validated. Additional Test Cases will equally be added based on context. For example, Phase 1 may have concluded that your Widget(s) behave(s) abnormally on a specific Tizen version. In such case additional emphasis will be placed on such context. Once Phase 2 is completed, once again Samsung will communicate outstanding issues back to you for remedy by your Team.

Normally it is expected that your Team solves these issues in about Two Weeks, however Samsung understands that at this stage some bugs maybe harder to solve than others. In such case, this time period maybe extended to some degree to cater for additional development efforts. Additionally, for you to fully understand the root causes of select bugs, you may need further communication with Samsung Testers, either via email or over a Call. Feel free to reach out should this be the case.

- **Phase 3**

Good going! At this stage, your Development Team has learnt about a host of pressing issues concerning your App, and the chances are most of them have already been fixed. Thanks to efforts on both sides, your PIRS is truly shaping up!

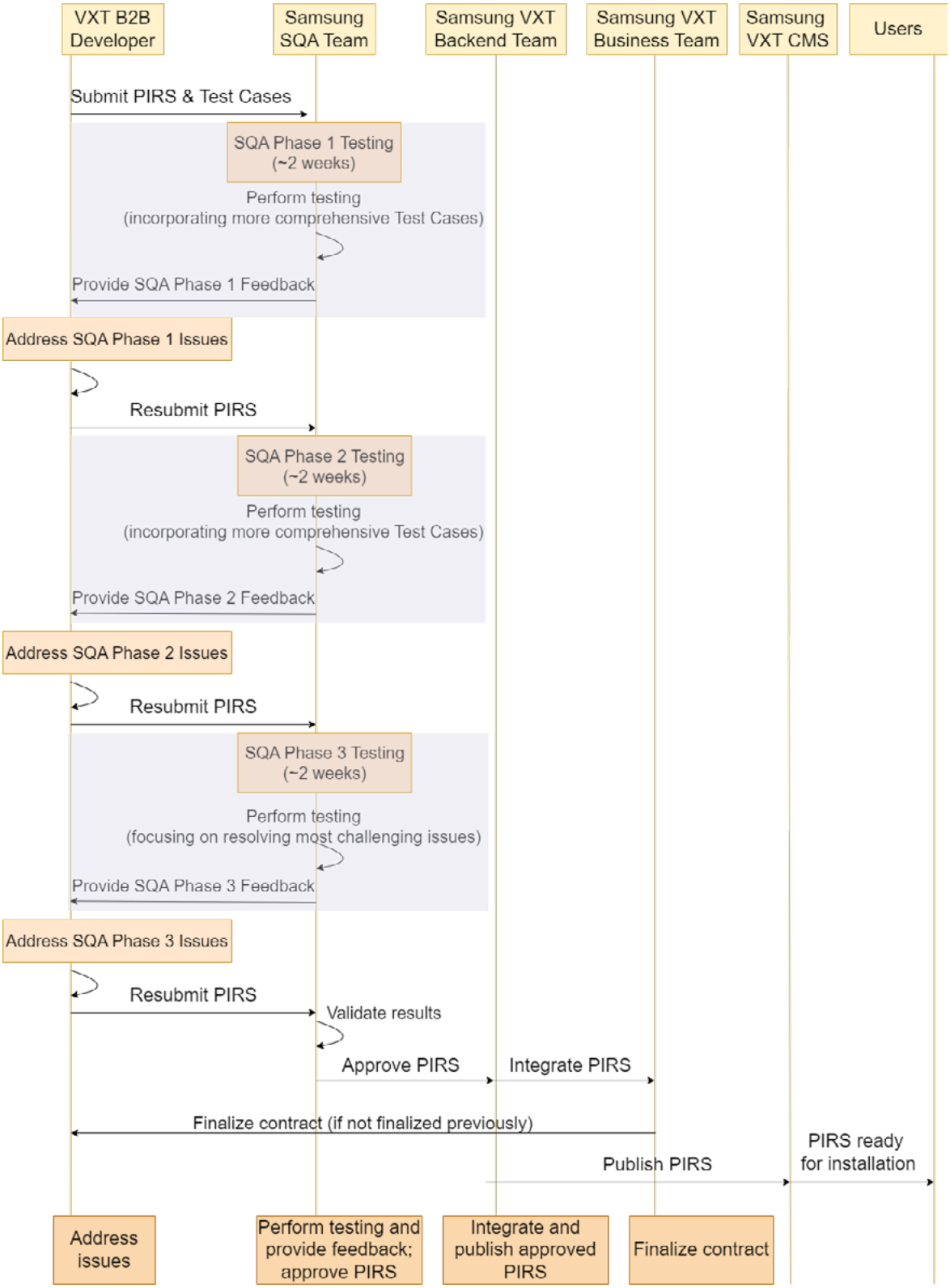
Phase 3 is designed to iron out the worst of the worst issues, which have proven most difficult to solve. It is likely that several attempts at fixing them have already been attempted until this point, but have failed due to one or more reasons. Debugging will likely entail further communication efforts, and seeking in-depth understanding. Phase 3 is also expected to last Two Weeks, though may extend beyond this timeframe due to the nature of remaining issues.

During Phase 3 all problematic Test Cases will be once again be re-validated, and based on outcome further assistance will be provided to your Team by Samsung Engineers in order for them to be overcome. In select cases, the root cause of these severe bugs may lay within VXT Canvas and/or VXT Player. In such case, they will be escalated within Samsung for appropriate handling. In either instance, it is expected

that all identified issues throughout the previous SQA Process Phases have been suitably addressed, and are no longer an ongoing concern.

8.3 Next Steps after SQA is Completed

Figure 76 SQA Process.



Congratulations! Your PIRS is now not only mature but also dedicated to delivering a quality orientated User experience, both within VXT Canvas as well a VXT Player.

Successfully completing the SQA Process is a formal qualifier to officially launching it on Samsung VXT CMS, where Users may choose to purchase it, and find its feature set beneficial to their business.

Once your PIRS has passed the SQA Process, and the results have been formally conveyed back to you, it is now only a matter of time when it becomes available to all who login to Samsung VXT CMS at https://samsungvx.com. The exact date when this happens is dependent on two things. Firstly, it is a case when the Samsung VXT Backend Team may integrate your App into Samsung VXT CMS. Secondly, it is down to finalising your Contract & Agreement with the Samsung VXT Business Team.

From the moment your PIRS is officially launched, it will be available for everyone from within the VXT Apps Menu option directly within VXT Canvas under the Category specified by you and/or Samsung. If you chose to limit it to a specific Region, it will only be available for Users logging in from such location(s).

Do not allow a single PIRS to determine your prosperous future with Samsung VXT CMS. If you have more ideas for applications in the pipeline, then share them with Samsung for mutual review. Samsung is always interested in superior ideas that satisfy User requirements across various Market Verticals. Do not hesitate to get in contact, as your Competitor(s) may follow suit before you do. If in doubt, contact your SQA Process Engineer for further guidance on how to proceed, and become a Prime B2B VXT Partner delivering many PIRS!

9. Partnership in Business as a Successful Samsung VXT CMS Partner

What happens once your VXT Application is Live on Samsung VXT CMS

9.1 Ownerships and Responsibilities

Becoming a B2B VXT Partner and developing PIRS for Samsung VXT CMS can be a rewarding experience for your Company, especially as it allows you to scale to new Markets and Continents, and reap far encompassing benefits of having a vast and varied audience readily available. At the same time, it is also a period of mutual B2B participation in satisfying the needs and wants of this Market.

This section outlines such requirements and goals with the intent to increasing awareness among tasks that await before, during, and after you have successfully developed and launched your VXT App for availability on Samsung VXT CMS.

9.2 Maintaining & Updating Your PIRS

From the moment your App is officially launched on Samsung VXT CMS, it becomes a hallmark of your Company and the talents of both your Developers, Designers, and Testers alike! We join in your satisfaction, and wish you a long deserved success.

Throughout the lifespan of your PIRS you remain its Owner and sole Developer. All successful software projects strive on the essential premise of continued operability. The same is true for PIRS, for they must deliver the same level of performance across VXT Platforms, and continue to do so over a period of years. A User choosing to use your PIRS must feel confident that it not only adheres to their expectations, but also ensures reliability in the months and years ahead.

Samsung is fully committed to maintaining the Samsung VXT CMS eco-system to remain compliant with modern web development standards, and deliver a rich and rewarding End User experience. As a B2B VXT Partner, Samsung expects the same commitment by your Company in the following areas, not least limited to upholding the operability of your Apps itself:

- PIRS and all its Components (Source Code, JSON)

- Your B2B VXT Partner Backend (if used by your PIRS)
- 3rd Party API's (if used by your PIRS)
- VXT Templates (if used by your PIRS)

Should your existing PIRS version require ample changes to meet new Web Standards ensuring the same user experience, it will need to undergo renewed Testing in the form of commencing a new and full SQA Process. If other cases, if the latest version of your PIRS only incorporates minor changes (i.e. fixing JS bugs, CSS related glitches etc.), and providing this can be efficiently proven as solved through minimum testing, then it can be re-deployed to Samsung VXT with minimum delay, and additional work.

In either case, you are required to implement regular reviews of your App to ensure it continues to deliver the same level of performance across different Screens running VXT Player. This is particular true in instances when VXT Player can be run on various Browsers and versions.

Samsung expects most PIRS will utilize a Backend as well as 3rd Party API's. In both these cases, you will need to ensure not only that your Backend is online, but also that it can satisfies ongoing service requirements and performance expectations instilled by Users of Samsung VXT CMS. If a User must wait too long for your App to fetch Content (as an example) from an external source, it may hinder their desire to proceed with its use over an extended time period. 3rd Party API's are also prone to evolve over time. Should this happen, compatibility in how your App interacts with them has to be maintained throughout their lifecycle. In cases when a 3rd Party API version becomes deprecated, you must closely monitor when such event is to unfold, and modify your App prior to this taking place.

9.3 Providing Technical Support for PIRS

With Samsung VXT, Samsung introduces a new model for providing Technical Support for Users of your PIRS.

Should a User experience an issue with your PIRS, he/she will in the first instance contact Samsung Technical Support, who will investigate and act to ascertain the root cause of the problem experienced. If following analysis it is deemed to be related to Samsung VXT CMS by itself (including the VXT Backend), then it will be dealt with internally by any number of Samsung Technical Support and/or Development Teams and staff necessary to solve it. More so, these Teams of Samsung engineers will work as a collective force on a Global scale to both limit the impact these issues may have on Users of your App, but also strive to remedy these issues with minimum delay.

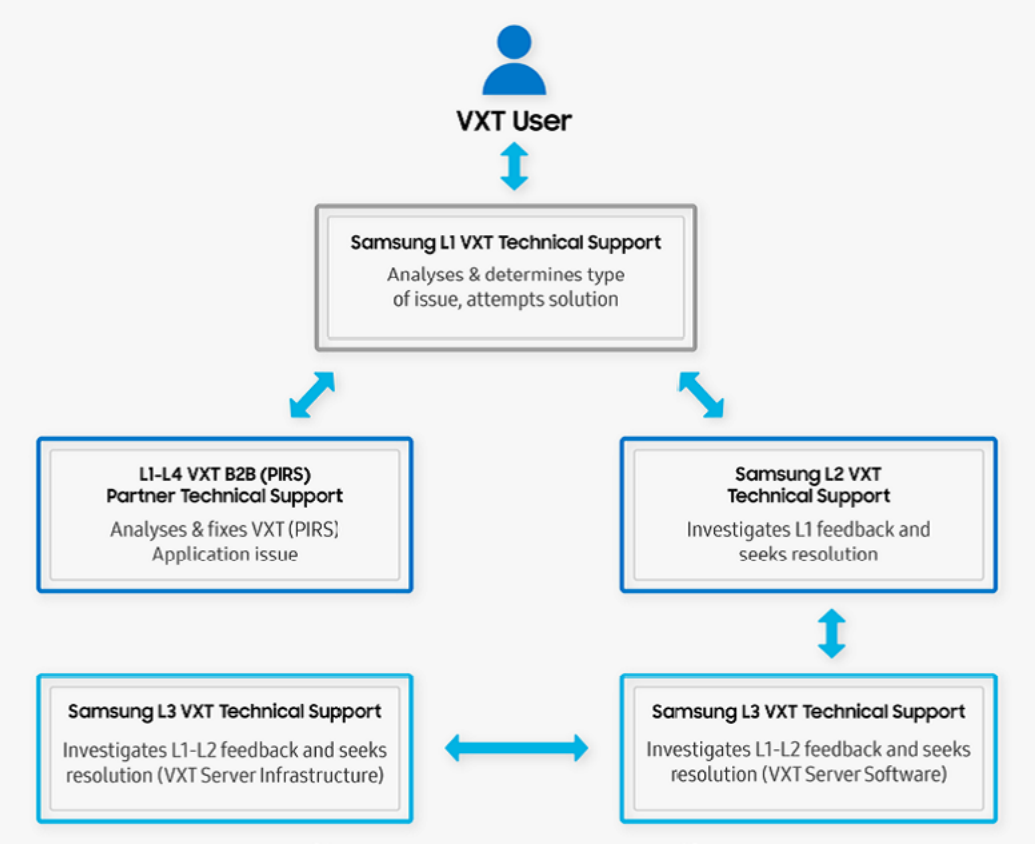
Should analysis however suggest that the root cause lies within your App, then the User's support request will be forwarded to your Company for continued handling and remedy thereafter. For this reason, you will need to allocate adequate resources to act as Technical Support Engineers for supporting your PIRS, as well as take into account having staff on hand that speak the local language in which your Application is offered for purchase, as not all Users may prefer to communicate in English. This is particular true in select Countries and/or Regions where English proficiency is low.

Should Samsung Technical Support need to escalate a support request to you as a B2B VXT Partner, this information will be relayed to you in the form of a Support Ticket via a technical support ticketing Platform. This approach is designed to streamline communication between Samsung and B2B VXT Partners. Samsung recommends you communicate information on your current Ticketing System when you are preparing to launch your App. Such practice will, either assist in ensuring integration of your Ticketing System into that used by Samsung, or provide Samsung with the opportunity to work out a solution that is mutually compatible.

Providing Technical Support is a necessity, however you can strive to reduce the likelihood of a User needing to submit support requests via implementing a string of best practices:

Figure 77

Technical Support Schema.



- Fixing all outstanding issues identified during internal testing
- Fixing all outstanding issues identified through Samsung SQA*
- Making your App more user friendly
- Adding context information into your App

(*) Pre-requisite to launching your PIRS on Samsung VXT CMS.

This can be achieved in a number of ways, but not least to using clear descriptions within the VXT Extension/Widget and/or labels in the VXT Property Panel, or providing animated storyboards or videos that may be displayed (activated by the User) showcasing steps to using your Application. A key aspect to remember is that if a User does not know how to successfully use your PIRS, and cannot seek assistance without submitting a support request, they are more likely to lose interest, and seek fulfillment with a PIRS from one of your competitors. Samsung recommends to always focus on making your App's UX best-in-class.

10. WiNE API Reference

Table 39 $vxt Object Reference.

$vxt	
Description	
$vxt object is added to the window instance by the WiNE API, allowing for communication with other components of PIRS.	
Methods	
createChannel(id: VXTId): Channel	closeChannel(id: VXTId): void
Description	Description
Extension/Widget calls this API to open communication channel with VXT Canvas/ VXT Player.	Extension/Widget calls this API to close communication channel with VXT Canvas/ VXT Player.
Parameters	Parameters
id: extension ID/ widget ID. $vxtSubChannelId variable should be used to create channels of communication.	id: extension ID/ widget ID. $vxtSubChannelId variable should be used to create channels of communication.
Example	Example
`let channel = $vxt.createChannel($vxtSubChannelId);`	`$vxt.closeChannel($vxtSubChannelId);`

Table 40 Channel Reference.

Channel	
Description	
Communication channel is required for Extension/Widget to communicate with the VXT Canvas/Player and publish/subscribe events in the WiNE framework.	
Methods	
subscribe(event: string, callback: (data: Object) => void): void	publish(event: string, payload: Object): void

Channel	
Description	Description
Subscribe method enables extension/widget to receive events from the WiNE framework and to modify application's behaviour based on data received.	Publish method enables widget/extension to send events to the WiNE framework and impact the VXT UI.
Parameters	Parameters
Event: String that describes the kind of event subscribed by the Extension/Widget Callback: Handler function called upon receiving event from the WiNE API. This function handles information obtained in the event. Received information varies based on the Event.	Event: String that describes the type of event send by the Extension/Widget Response: Object containing additional data that are send by the Extension/Widget. Object should contain two keys: type: String – specifies Event Type payload: Object – actual data that are being sent while publishing an Event. Structure of this object varies depending on Event.
Example	Example
`channel.subscribe("<event>", (<response>) => { //do something });`	`channel.publish("<event>", {<response>});`

Table 41 Subscribe Events Reference.

Subscribe Events			
"get"	"config"	"playstate"	"vxtstate"
Description	Description	Description	Description
This Event fires when User clicks "Save" in Settings, or (if an Application doesn't use them), when the App is launched. Later on, it fires on orientation change or search query input. It can be thought of as an initial Event. This Event should be followed by publishing "data" Event to display either list of Widgets ("contents") or metadata ("metadata").	Information about configuration properties' states (state of Property Panel and the hidden property), and Widget's instance ID. Upon receiving this Event, Widget needs to update based on the received data.	Information about the state of playing the Widget (both in VXT Canvas and VXT Player). Upon receiving this Event, Widget needs to update based on the received data.	Information about VXT Canvas Project.
App Component	App Component	App Component	App Component
Extension	Widget	Widget	Widget
Data Structure	Data Structure	Data Structure	Data Structure

Subscribe Events

For type: “content”:

```
{
 "type": "content",

 "payload": {
  "searchQuery": query: Str,

  "language": language: Str,

  "orientation":
orientation: Str,

  "id": workspaceId: VXTWork-
spaceStr,

  settingsKey: Str settingsVal-
ue: any
 }

}
```

For type: “metadata”:

```
{
 "type": "metadata",
 "payload": {
  "language": language: Str,
  settingsKey: Str settingsVal-
ue: any
 }
}
```

```
// IN VXT CANVAS

{
 "data": {
  propPanelTabName: Str : {
   propPanelKey: Str : propPan-
elValue: any,
        ...
      },
      ...
 },

 "subscription": {},
 "theme": {},
 "vxtId": workspaceId: VXTWork-
spaceString,
 "widgetId": widgetId: VXTWid-
getString
}

// IN VXT PLAYER

{
 "data": {
  propPanelTabName: Str: [{
  propPanelKey: Str : propPan-
elValue: any,
   ...
  }],
  ...
 },
 "subscription": {},
 "theme": {},
 "vxtId": workspaceId: VXTWork-
spaceString,
 "widgetId": widgetId: VXTWid-
getString,
 "playerData": {
  "tags": [{
   "key": tagKey: Str,
   "name": tagName: Str,
   "value": tagValue: Str
  }]: Array <{key: Str, name: Str,
value: Str}>,
  "name": screenName: Str,
  "orientation": orientation: Str,
  "absolute": {
   "x": xValue: Number,
   "y": yValue: Number,
   "width": widthValue: Number,
   "height":  heightValue:
Number,
  }
 }
}
```

```
{
 "type": playstateType: Str,
 "payload": {}
}
```

```
{
 "type": vxtstateType: Str,
 "payload": {}
}
```

Subscribe Events			
Types	Types	Types (playstateType)	Types (playstateType)
"content" – in Content Driven Applications; fires when Extension should publish content.	Type not used	"ready" - fires when Widget is loaded; upon receiving this playstate message the Widget prepares content to be played in next lifecycle Event.	"new" - fires when User clicks "New" in VXT Canvas.
		"play" - fires when Widget is loaded and played; upon receiving this playstate message the Widget starts displaying its content.	"save" - fires when User clicks "Save" or "Save as" in VXT Canvas.
"metadata" – in Data Driven Applications; fires when Extension should publish metadata.		"hold" - fires when VXT Player exit confirmation pop-up is displayed; upon receiving this playstate message the widget pauses displaying the content and renders the default UI of the widget	"preview" - This type of vxtstate fires when User clicks "Preview" in VXT Canvas
		"resume" - fires when user clicks "cancel" in the aforementioned confirmation popup; upon receiving this playstate message the widget resumes displaying the content and hides the default UI.	
		"stop" - fires when User discards project / deletes Widget / the playing is stopped - upon receiving this playstate message the Widget should finish playing content, close communication, disconnect from external sources, stop timers and close platform APIs.	

Table 42

Publish Events Reference.

Publish Events		
"data"		"toastMessage"
Description		**Description**
Extension should fire this event in response to "get" operation returning data with list of Widgets for type "contents" or guidelines for User for type "metadata". Widget should fire this event in order to update its Manifest.		Widget fires this event in order to display a pop-up toast message in VXT Canvas UI.
App Component	**App Component**	**App Component**
Extension	Widget	Widget
Types	**Types**	**Types (toastType)**
"contents" – publish list of Widgets	"widgetManifest" – update Property Panel	"info" – display green pop-up with info icon
		"warning" – display red pop-up with warning icon
		"error" – display red pop-up with error icon
"metadata" – publish guidelines		"success" – display blue pop-up with success icon
Data Structure	**Data Structure**	**Data Structure**

For type: "contents":

```
{
 "type": "contents",
 "payload": {
 "categories": [
   Category
   ...
 ]: Array <String | {name: Str, displayMode: Str}>,
 "content": [
   Content
 ]: Array <Object>
}
}
```

For type: "metadata":

```
{
 "type": "metadata",
 "payload": [
   metadataObject: Object,
   ...
 ]: Array <Object>
}
```

```
{
 "type": "widgetManifest",
 "payload": {
  propPanelTabName: Str: [
   {
    "id": propPanelKey: Str,
    propToUpdate: Str : updatedValue: any
   },
   ...
 ]: Array <Object>
 }
}
```

```
{
 "type": toastType : Str,
 "payload": {
  "message": Str
 }
}
```

Table 43

Content Object Reference.

Content Object

```
interface Content {
  "resourceType": resourceType: String,
  "resourceId": resourceId: String,
  "resourceName": resourceName: String,
  "resourceSize": resourceSize: String,
  "thumbnailPath": thumbnailPath: String,
  "isLandscape": isLandscape: Boolean,
  "category": categoryName: String,
  "orientation": orientation: String,
  "keywords": keywords: String,
  "viewEdit?": viewEdit: Object,
  "config?": configObject: Object,
  "children?": contentArr: Array,
}
```

```
enum ResourceType {
 CONTENT = 'CONTENT',
 FOLDER = 'FOLDER'
}

enum Orientation {
 ALL = 'all',
 LANDSCAPE = 'landscape',
 PORTRAIT = 'portrait'
}
```

Table 44

Category Reference.

Category

```
interface Category {
 name: string;
 displayMode: CategoryDisplayMode;
}
```

```
enum CategoryDisplayMode {
 GRID = 'grid',
 CAROUSEL = 'carousel'
}
```

> NOTE: These interfaces and enums have been re-named solely for the purpose of this Guide.

Table 45

Configuration Properties Reference.

Configuration Properties

Type	Other Properties	Property Description	Supported Values
text	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User's input. Value will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String, Special characters Length: 0~1000 characters

Configuration Properties			
text	Multiline (optional)	When this property is present and set to true, input box will turn into a text area.	Boolean Default: false
	charactersLimit (optional)	Displays how many characters can be used as input value. Works only with “multiline” property set to true.	Number
	label (optional)	A placeholder.	String
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
number	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display on the left side of the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User’s input. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	Number
	min	The minimum value permitted for entry.	Number

Configuration Properties			
number	max	The maximum value permitted for entry.	Number
	numericType (optional)	The numeric data sub-type that is permitted for entry.	String Possible values: decimal/float Default: float
	units (optional)	The unit of the entered value displayed on the right side of the element.	String
	label (optional)	A placeholder.	String
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
date	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	String the value must match the format: YYYY-MM-DD / MM-DD-YYYY / DD-MM-YYYY

Configuration Properties			
date	format	Date format.	String Possible values: YYYY-MM-DD / MM-DD-YYYY / DD-MM-YYYY Default: YYYY-MM-DD
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
time	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	String The value must match the format: HH:MM
	format	Time format.	String Possible values: HH:MM / 12h(HH:MM)

Configuration Properties			
time	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
color	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	String Must be a valid color format
	parameters (optional)	Array of elements of type “range” to display extra custom color sub-options. NOTE: None of these sub-options are applied automatically – this element works just like any other of type: “range” – it will simply inform us of its value.	An array of configuration objects of type: “range”.
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show

Configuration Properties			
color	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
media	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	mode	Determines, if User can select multiple media items, or just one at a time.	String Possible values: single/multi
	mediaType	Type of media possible for selection.	String Possible values: image
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value (a path to media file) overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	String Path to media file.
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show

Configuration Properties			
media	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
dropdown	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display on mouse hover over the dropdown element.	Number, String, Special characters Length: 1~100 characters
	list	Array of selectable values.	Array of strings/objects with the following properties, that support strings (Text and icon) and booleans (displayText): Text – name of list item Icon – icon to display next to the item displayText – if list item name should be displayed Max list length: 5
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	String/object with the following properties, that support strings (Text and icon) and booleans (displayText): Text – name of list item Icon – icon to display next to the item displayText – if list item name should be displayed
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show

Configuration Properties			
dropdown	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
range	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	min	The minimum value permitted for entry.	Number
	max	The maximum value permitted for entry.	Number
	step	A unit of movement.	Number
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	Number
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium

Configuration Properties			
range	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
checkbox	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display on the right side of the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User’s selection. Value will be returned as a response of a “config” event that we subscribe to in Widget’s JS file.	Boolean
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
toggle	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters

Configuration Properties			
toggle	caption	Title to display next to the element.	Number, String, Special characters Length: 1~100 characters
	value	Custom default value overwritten later by User's selection. Value will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Boolean
	rightside (optional)	Toggle is by default sticked to the left side of parent container. With the "rightside" property present and set to true, we can stick the toggle to the right side of parent container.	Boolean
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: "x" and "y" Both "x" and "y" support numeric values.
radio	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	list	Array of selectable values.	String array Max list length: 5

Configuration Properties			
radio	value	Custom default value overwritten later by User's selection. Value will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	String
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: "x" and "y" Both "x" and "y" support numeric values.
url	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display over (on) the clickable button	Number, String, Special characters Length: 1~100 characters
	value	Target url to be opened either inside a pop-up window or in a new tab.	string 0-1000 characters URL
	openType	Determines, whether the URL should be displayed inside a pop-up modal (dialog), or in a new tab (window).	String Possible values: dialog/window

Configuration Properties			
url	dialogPosition	Define, where the dialog modal should be positioned. Works only with “openType” property set to “dialog”.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values. Default: { “x”: 0, “y”: 0 }
	dialogSize	Set custom size of dialog modal. Works only with “openType” property set to “dialog”.	Object with two properties: width and height Both “width” and “height” support numeric values. Default: { “width”: 400, “height”: 500 }
	dialogAuthValidation (optional)	Backend API to get authentication token. If this property is provided, VXT Canvas will send validation request after dialog is closed. Auth validation response payload should be in JSON format, which will be validated by VXT Canvas. If the format is correct, the response will be returned as a value of the “value” property. It is then Widget’s responsibility to handle positive/negative auth validation. Works only with “openType” property set to “dialog”.	String API
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element (button).	String Possible values: small/medium/large Default: medium

Configuration Properties			
url	position (optional)	Sets custom position of element (button).	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
img NOTE: Type not supported in Settings.	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	value	A path to image file.	String Path to image file.
	parameters (optional)	An option to display an extra side panel attached to the left side of the Property Panel. We can display various configuration elements inside Parameters side panel, except for “gridImage” – if we want to display images in a grid inside Parameters panel, we should use the “children” property instead.	Array of any configuration objects, except for “gridImage”.
	children (optional)	Grid of images, which should be displayed inside Parameters side panel. It's actually a property of "parameters" property.	Array of configuration objects of type: “img”.
	ratio (optional)	Ratio of displayed image button.	String Possible values: 16:9/9:16/1:1 Default: 16:9
	viewSate (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show

Configuration Properties			
img NOTE: Type not supported in Settings.	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
gridImage NOTE: Type not supported in Settings.	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	list	Array of image elements of to display in a grid.	Array of configuration objects of type: “img”.
	value	Path to image file, selected from a grid.	String Path to image file.
	caption (optional)	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.

Configuration Properties			
gridRange	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element.	Number, String, Special characters Length: 1~100 characters
	link	A checkbox, which enables linking all the sliders together. If the sliders are linked, when any of them gets updated, it affects all of the rest.	Boolean Default: true
	list	Array of slider elements of to display in a grid.	Array of configuration objects of type: “range”.
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
accordion	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	caption	Title to display above the element (next to the fold/unfold button).	Number, String, Special characters Length: 1~100 characters
	list	Array of configuration elements to be displayed in accordion.	Array of configuration objects
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium

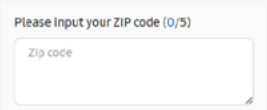
Configuration Properties			
message	id	Unique ID.	Number, String Length: 1~100 characters
	value	Body of a message.	String
	caption (optional)	Title to display above the message body.	Number, String, Special characters Length: 1~100 characters
	viewState (optional)	Determines if element should be active, disabled or hidden.	String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
guide	id	Unique ID.	Number, String Length: 1~100 characters
	value	Text to display on mouse hover over the tooltip.	String String Possible values: show/hide/disable Default: show

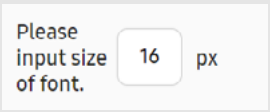
Configuration Properties			
guide	viewState (optional)	Determines if element should be active, disabled or hidden.	String String Possible values: show/hide/disable Default: show
	size (optional)	Sets custom size of element.	String Possible values: small/medium/large Default: medium
	position (optional)	Sets custom position of element.	Object with two properties: “x” and “y” Both “x” and “y” support numeric values.
divider	id	Unique ID.	Number, String Length: 1~100 characters
hidden	id	Unique ID that will be returned as a response of a "config" event that we subscribe to in Widget's JS file.	Number, String Length: 1~100 characters
	value	Any value that should be passed to the Widget.	Number, String

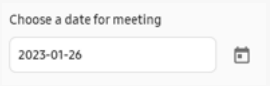
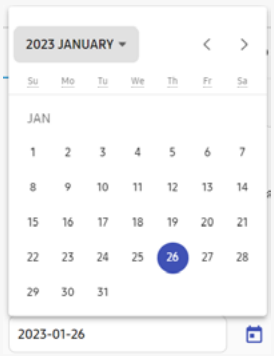
Table 46

Configuration Objects Examples.

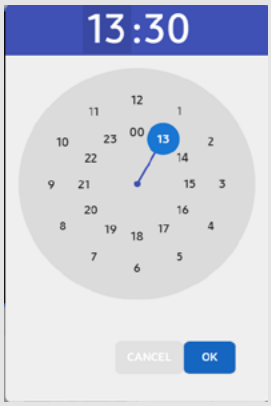
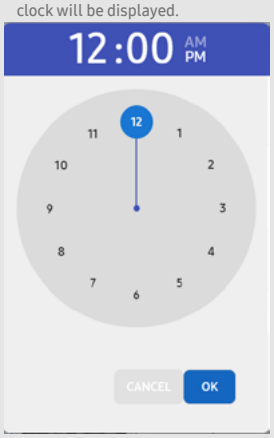
Configuration Objects - Examples			
Type	Example - Code	Example - UI	Expected Output To Widget Config

text	{
	"type":"text",
	"id":"inputZipCode",
	"caption":"Please input your ZIP code",
	"value":"",
	"multiline": true,
	"charactersLimit":5,
	"label":"Zip code"
	}
			inputZipCode: “”

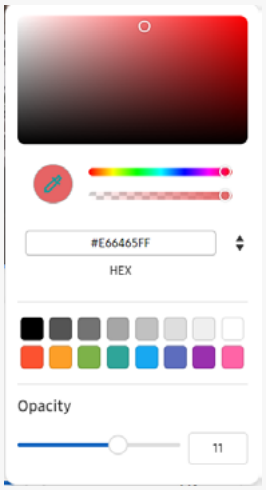
number	{
	"type":"number",
	"id":"inputFontSize",
	"caption":"Please input size of font.",
	"value":16,
	"label":"Font",
	"min":5,
	"max":25,
	"numericType":"float",
	"units":"px"
	}
		
		When we remove the value, we’ll be able to see the label.
		 
			inputFontSize: 16

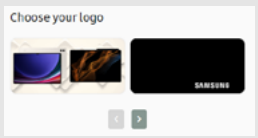
date	{
	"type":"date",
	"id":"dateCalendar",
	"caption":"Choose a date for meeting",
	"value":"2023-01-26",
	"format":"YYYY-MM-DD",
	"position":{
	"x":0,
	"y":15
	}
	}
		
		After clicking on the calendar icon, the calendar will be displayed.
		 
			dateCalendar: “2023-01-26”

Configuration Objects - Examples			

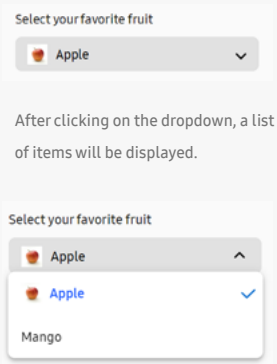
time	24 hour format:
	
	{
	"type":"time",
	"id":"time24",
	"caption":"Choose a time for meeting",
	"value":"13:30",
	"format":"HH:MM",
	"size":"small",
	"position":{
	"x":0,
	"y":15
	}
	}
	
	12 hour format:
	
	{
	"type":"time",
	"id":"time12",
	"caption":"Choose a time for meeting",
	"value":"12:00",
	"format":"12h(HH:MM)",
	"size":"small",
	"position":{
	"x": 10,
	"y": 25
	}
	}
		Choose a time for meeting
		13:30
		
		After clicking on the input box, a clock will be displayed.
		
		 
		
		12 hour format:
		
		Choose a time for meeting
		12:00 PM
		
		After clicking on the input box, a clock will be displayed.
		
		 
		
			24 hour format:
			time24: “13:30”
			
			12 hour format:
			time12: “12:00”

Configuration Objects - Examples			

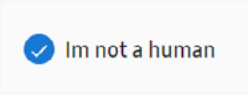
color	{
	"type":"color",
	"id":"palette",
	"caption":"Choose your color",
	"value":"#e66465",
	"parameters": [
	{
	"type":"range",
	"id":"opacity",
	"caption":"Opacity",
	"min":-20,
	"max":30,
	"step":0.5,
	"value":11
	}
	]
	}
		
		After clicking on the color, a palette with extra parameters will be displayed.
		 
			palette: "#e66465"

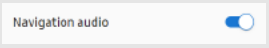
media	{
	"type":"media",
	"id":"picker",
	"mode":"multi",
	"mediaType":"image",
	"caption":"Choose your logo",
	"value":[
	"images/favicon.svg"
	]
	}
		
		After clicking on the image, a pop-up modal will be displayed.
		 
		If we set the “mode” property to “multi”, selected images will be displayed in Property Panel as a carousel.
		 
			picker: ["images/favicon.svg"]
			
			// when other images are selected, their source paths will be present in the array

Configuration Objects - Examples			

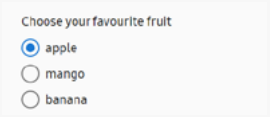
dropdown	{
	"type":"dropdown",
	"id":"fruits",
	"caption":"Fruits",
	"list":[
	{
	"text":"Apple",
	"icon":"https://cdn.pixabay.com/apple.jpg",
	"displayText": true
	},
	"Mango"
	],
	"value":"Apple"
	}
			fruits: "Apple"

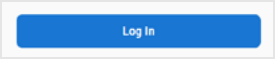
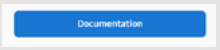
range	{
	"type":"range",
	"id":"rangeSlider",
	"caption":"Desired temperature",
	"min":-20,
	"max":30,
	"step":0.5,
	"value":11
	}
			rangeSlider: 11

checkbox	{
	"type":"checkbox",
	"id":"check",
	"caption":"Im not a human",
	"value":true
	}
			check: true

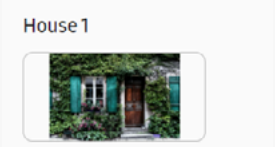
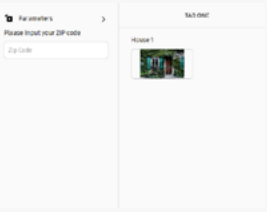
toggle	{
	"type":"toggle",
	"id":"nav",
	"caption":"Navigation audio",
	"value":true,
	"rightside": true
	}
			nav: true

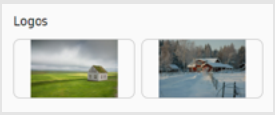
Configuration Objects - Examples			

radio	{
	"type":"radio",
	"id":"radioChoose",
	"caption":"Choose your favourite fruit",
	"value":"apple",
	"list":["apple","mango","banana"]
	}
			radioChoose: “apple”

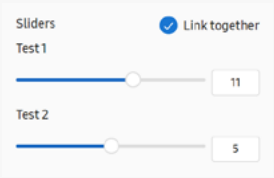
url	dialog:
	{
	"type":"url",
	"id":"login",
	"caption":"Log In",
	"value": "https://sample.com/login",
	"openType":"dialog",
	"dialogPosition": {
	"x": 10,
	"y": 10
	},
	"dialogSize":{
	"width":560,
	"height":315
	},
	"dialogAuthValidation": "https://myAppBackService.com/getToken"
	}
	window:
	{
	"type":"url",
	"id":"doc",
	"caption":"Documentation",
	"value":"https://sampleApp.com/documentation",
	"openType":"window"
	}
		
		After clicking on the button, a page specified in “value” property will be displayed in a pop-up modal.
		 
		window:
		 
		After clicking on the button, the URL specified in “value” property will be opened in a new browser tab.
			dialog:
			login: { //data returned from dialogAuthValidation API }
			
			window:
			doc: "https://sampleApp.com/documentation"

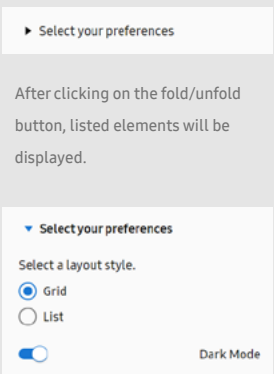
Configuration Objects - Examples			

img	{
	"type":"img",
	"id":"house1",
	"caption":"House 1",
	"value": "https://cdn.pixabay.com/photo.jpg",
	"parameters":[
	{
	"type":"text",
	"id":"inputZipCode",
	"caption":"Please input your ZIP code",
	"value":"",
	"label":"Zip Code"
	}
	]
	}
		
		After clicking on the image, a Parameters tab will be displayed on the left side of Property Panel.
		 
			house1: "https://cdn.pixabay.com/photo.jpg"
			
			If the image was clicked and Parameters tab was displayed and some value (for example, "123") was provided into the input box, the output will look as follows:
			
			"house1":{"value": "https://cdn.pixabay.com/photo.jpg" "parameters": {"inputZipCode": "123"}}}

gridImage	{
	"type":"gridImage",
	"id":"gridLogo",
	"caption":"Logos",
	"value":"",
	"list":[
	{
	"type":"img",
	"id":"house2",
	"caption":"House 2",
	"value": "https://cdn.pixabay.com/photo1.jpg"
	},
	{
	"type":"img",
	"id":"house3",
	"caption":"House 3",
	"value": "https://cdn.pixabay.com/photo2.jpg"
	}
	]
	}
			gridLogo: ""
			
			After clicking on any of listed images, its value (path to the image) will be returned as a value of "gridImage" – for example, clicking on the first image will result in the following output:
			
			gridLogo: "https://cdn.pixabay.com/photo1.jpg"

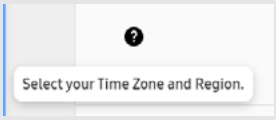
Configuration Objects - Examples			

gridRange	{
	"type":"gridRange",
	"id":"ranges",
	"caption":"Sliders",
	"link":"true",
	"list":[
	{
	"type":"range",
	"id":"rangeSlider1",
	"caption":"Test 1",
	"min":-20,
	"max":30,
	"step":0.5,
	"value":11
	},
	{
	"type":"range",
	"id":"rangeSlider2",
	"caption":"Test 2",
	"min":0,
	"max":10,
	"step":0.1,
	"value":5
	}
	]
	}
			ranges: { rangeSlider1: 7.5, "rangeSlider2": 7.5 }

accordion	{
	"type":"accordion",
	"id":"acc",
	"caption":"Select your preferences",
	"list":[
	{
	"type":"radio",
	"id":"radSelStyle",
	"caption":"Select a layout style.",
	"list":[
	"Grid",
	"List"
	],
	"value":"Grid"
	},
	{
	"type":"toggle",
	"id":"toggleTheme",
	"caption":"Dark Mode",
	"value":true
	}
	]
	}
			acc: { radSelStyle: "Grid", toggleTheme: true }

Configuration Objects - Examples			

message	{
	"type":"message",
	"id":"msg",
	"value":"This Application has been developed for demo purposes only",
	"caption": "Welcome to My Sample Application!",
	"size":"small",
	"position":{
	"x":0,
	"y":20
	}
	}
			N/A

guide	{
	"type":"guide",
	"id":"stepByStepGuide",
	"value":"Select your Time Zone and Region.",
	"position":{
	"x":15,
	"y":0
	}
	}
		
		After hovering the mouse over the element, text specified in “value” property will be displayed.
		 
			N/A

divider	{
	"type":"divider",
	"id":"divider"
	}
			N/A

hidden	{
	"type": "hidden",
	"id": "wgtId",
	"value": "Wgt-Id-55"
	}
		N/A	wgtId: “Wgt-Id-55”

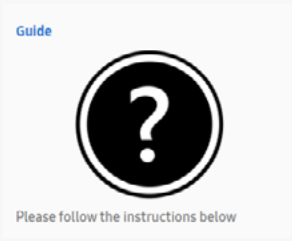
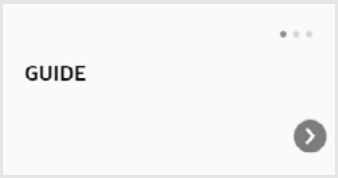
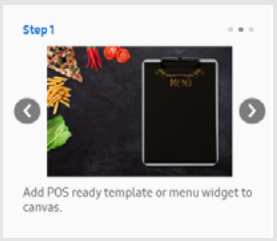
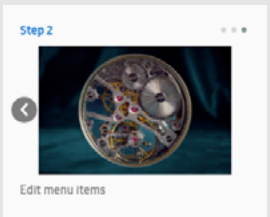
Table 47

Metadata Properties Reference.

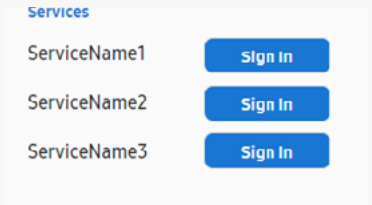
Metadata Properties		
Type	Other Properties	Descriptions
header	title	String to be displayed as a header.
paragraph	title	String to be displayed as paragraph title.
	description	Body of the paragraph.
	image (optional)	Path to image to be displayed between paragraph title and paragraph body.
carousel	items	Array of metadata elements of type “header” and/or “paragraph” to be displayed in carousel format.
signIn	title	String to be displayed above the service providers set.
	services (optional)	Array of service names to be displayed next to the sign in/sign out buttons. If this property is not specified, VXT Canvas will send a request to the auth/ endpoint with value set to “ALL”, and the service list will be generated based on the response.
	displayType (optional)	String which defines, how service providers should be displayed. Possible values: grid/dropdown
divider	-	Display a horizontal line.

Table 48

Metadata Objects Examples.

Metadata Objects		
Type	Example - Code	Example - UI
header	`{ type: 'header', title: 'How to import POS data to your digital menu board.' }`	
paragraph	`{ type: 'paragraph', title: 'Guide', description: 'Please follow the instructions below', image: 'https://cdn.pixabay.com/photo.png' }`	
carousel	`{ type: 'carousel', items: [{ type: 'header', title: 'GUIDE' }, { type: 'paragraph', title: 'Step 1', description: 'Add POS ready template or menu widget to canvas.', image: 'https://cdn.pixabay.com/photo1.jpg' }, { type: 'paragraph', title: 'Step 2', description: 'Edit menu items', image: 'https://cdn.pixabay.com/photo2.jpg' }] }`	First slide (header):  Second slide (paragraph):  Third slide (paragraph): 

Metadata Objects		

signIn	<pre>{
	type: 'signIn',
	title: 'Services',
	services: [
	'ServiceName1', 'ServiceName2', 'ServiceName3'
	]
	}</pre>
		

divider	<pre>{
	type: 'divider'
	}</pre>	

Table 49

WiNE Data Endpoints Glossary Reference.

WiNE Data Endpoints - Glossary	
Term	Description
VXT User	A User of VXT CMS.
POS	Point Of Sale.
POS Owner	A person who owns and runs POS hardware terminal for their business (for example, restaurant owner).
POS Credentials	ID/secret for signing in to POS.
POS Data	Menu data, which POS stores in its database.
Data Provider	Who gathers/manipulates/provides POS data to VXT.
Base URL	Backend service host URL of Data Provider.

Table 50 WiNE Data Endpoints Reference.

WiNE Data Endpoints - Glossary						
Endpoint	Description	Request Type	Request Header	Request Body	Response Code	Response Body

auth/	Response should return the connection status to specified service or to all services.	GET	'x-api-key' (Required)
			
			'x-service-provider' (Required)
					200	1) 'x-service-provider': 'myService'
						{
						"posType": "myService",
						"status": "Connected",
						"production": true,
						"message": "Already connected to myService",
						}
						
						2) 'x-service-provider': 'ALL'
						[
						{
						"posType": "myService",
						"status": "Connected",
						"production": true,
						"message": "Already connected to myService",
						},
						{
						"posType": "Sample Service",
						"status": "Connected",
						"production": true,
						"message": "Already connected to Sample Service",
						},
						{
						"posType": "CSV",
						"status": "Connected",
						"production": true,
						"message": "Already connected to CSV",
						}
						]

WiNE Data Endpoints - Glossary						

					200	1)'x-service-provider': 'myService'
						{
						"posType": "myService",
						"status": "Not Connected",
						"production": true,
						"message": "Need to sign in to myService",
						}
						2)'x-service-provider': 'ALL'
						[
						{
						"posType": "myService",
						"status": "Not Connected",
						"production": true,
						"message": "Need to sign in to myService",
						},
						{
						"posType": "Sample Service",
						"status": "Not Connected",
						"production": true,
						"message": "Need to sign in to Sample Service",
						},
						{
						"posType": "CSV",
						"status": "Not Connected",
						"production": true,
						"message": "Need to sign in to CSV",
						}
						]

					401	{
						"message": "Unauthorized request"
						}

auth/ disconnect	Response should return the connection status to specified service.	GET	'x-api-key' (Required) 'x-service-provider' (Required)		200	{ "posType": "myService", "status": "Disconnected", "message": "Successfully signed out from myService", }

					200	{
						"posType": "myService",
						"status": "Connected",
						"message": "Failed to sign out from myService",
						}

					401	{
						"message": "Unauthorized request"
						}

WiNE Data Endpoints - Glossary						
auth/ getAuthUrl	Response should return a redirection URL to service authen-tication page.	GET			200	{ "posType": "myService", "targetUrl": "REDIRECT_URL_TO_POS_AUTH_ PAGE" }
					503	{ "message": "Service Unavailable" }
locations/list	Response should return a list of locations.	GET	'x-api-key' (Required) 'x-service-provider' (Required)		200	[{ "locationId": "string", "locationName": "string", "description": "string" }]
					503	{ "message": "Service Unavailable" }
locations/ search	Response should return details of specified location.	POST	'x-api-key' (Required) 'x-service-provider' (Required)	{ "location-Id":"string" }	200	{ "locationName":"string", "description":"string", "address1":"string", "address2":"string", "city":"string", "stateCode":"string", "zipCode":"string", "country":"string", "phone":"string", "latitude":0, "longitude":0 }
					503	{ "message": "Service Unavailable" }

WiNE Data Endpoints - Glossary						

menus/list	Response should return list of menus of specified location.	GET	'x-api-key' (Required)
			
			'x-service-provider' (Required)
				{
				"location-Id":"string"
				}
					200	[
						{
						"categoryName":"string",
						"categoryId":"string",
						"menuName":"string",
						"menuId":"string",
						"menuImageUrl":"string",
						"menuBgImageUrl":"string",
						"description":"string",
						"portions":[
						{
						"portion":"string",
						"price":"string",
						"currency":"string",
						"calory":0
						}
						],
						"allergies":[
						"string"
						]
						}
						]

					503	{
						"message": "Service Unavailable"
						}

menus/search	Response should return filtered list of menus of specified location.	POST	'x-api-key' (Required)
			
			'x-service-provider' (Required)
				{
				
				"menuIds":[
				"string"
				]
				}
					200	[
						{
						"categoryName":"string",
						"categoryId":"string",
						"menuName":"string",
						"menuId":"string",
						"menuImageUrl":"string",
						"menuBgImageUrl":"string",
						"description":"string",
						"portions":[
						{
						"portion":"string",
						"price":"string",
						"currency":"string",
						"calory":0
						}
						],
						"allergies":[
						"string"
						]
						}
						]

WiNE Data Endpoints - Glossary						

					503	{
						"message": "Service Unavailable"
						}

inventory/
search
	Response
	should return
	filtered list
	of inventory
	of specified
	menus.
		POST	'x-api-key'
			(Required)
			
			'x-service-
			provider'
			(Required)
				{
				"menuIds":[
				"string"
				]
				}
					200	[
						{
						"menuId":"string",
						"status":"string",
						"quantity":0
						}
						]

					503	{
						"message": "Service Unavailable"
						}

refresh/	VXT Canvas will
	delete cached
	data, then fetch
	new data from
	service.
		GET	'x-api-key'
			(Required)
			
			'x-service-
			provider'
			(Required)
					200	[
						{
						"posType":":"myService",
						"dataRetreived":":"EPOCHTIME"
						}
						]

					503	{
						"message": "Service Unavailable"
						}

11. Frequently Asked Questions

11.1 Extension

Q: I click on my Application, install it, open settings and click save and then... nothing happens. The dots - loading animation is played on loop within the sidebar section.

A: Make sure your Extension JavaScript file is called index.js - otherwise it won't be recognized as an extension file.

Q: In extension (sidebar), there are two thumbnails/Widgets, but I have declared only one in the source code.

A: The source of the problem might lay in the “categories” variable. What causes the issue is setting the displayMode to “carousel” or providing only the name of category, as categories are displayed as carousel by default (when just category name is provided). The solution would be setting displayMode to “grid”.

Q: My Metadata signIn buttons don't get rendered - loading animation is played on loop where there should be a "Sign In" or "Sign Out" button.

A: Make sure the /auth endpoint returns correct format and values, as VXT Canvas sends a request there before displaying the buttons and based on your server's response, it will render either "Sign In" or "Sign Out" button. If it is unable to understand your response or reach your server, it will not render the buttons.

11.2 Widget

Q: Can Widget subscribe to get? As in: channel.subscribe('get', (payload) => ...)?

A: Widget can subscribe to whatever you want it to, but it will never receive data event. API events are specifically separated between Extension and Widget (except for "data" event, where both Widget and Extension can publish it, but the payload is very different).

Q: Why do we get "ready" and "play" playstates when Widget is loaded onto canvas area in VXT Canvas? Shouldn’t we get the “play” playstate only when the Widget is published to screen (VXT Player)?

A: Widget in Canvas follows Player playstates where Widget should first be initialized (on ready) and then actually run (on play).

Q: After typing in a number into the numeric input box (type: "number"), it sometimes changes the value by itself.

A: Make sure the mandatory "min" and "max" properties are set and the values of numeric properties (min, max, value) are numbers, not strings.

Q: My Property Panel suddenly stopped working.

A: Please see if you're utilizing the media picker feature (type: "media") – if so, make sure the value is defined as a string array, not a string. If you want to display the "+" placeholder in the UI, then the default value of this property must not be an empty string "", but rather an empty array []. If you define this property as a string (and let's say you.ve put in there some default picture, for example: "value": "https://www.example.com/myimage.png", then in VXT UI, if you remove the picture, the value will return to empty, and this will cause the entire Property Panel to break. It does not happen if the value is defined as an array in the first place.

Q: How can I implement a button functionality in my Property Panel and detect every click?

A: Use "type": "gridImage" and leave its value as empty string for now. Inside its "list" property, define your button(s) of "type":"img". Each of them will have its own value (path to the image they are displaying). Once any of them is clicked, its value (its path to the image) will fill the "gridImage"'s value (which was initially empty). This way you can detect the click (if the value of "gridImage" is not empty string, it means the button has been clicked). To detect subsequent clicks, make sure to clear the value of "gridImage" using channel.publish("data" { type: "widgetManifest", ...}) after every click.

Q: Can we disable or hide a Property Panel element?

A: Yes – use the "viewState" property of the element and set it to "disable" or "hide".

11.3 WiNE Framework

Q: Is Application Name and Application Logo a part of WiNE Framework?

A: No, they are used when uploading the Application through VXT Developer Portal. WiNE Framework doesn't play any role in defining your Application's Name or Logo.

11.4 Other

Q: Can we handle various business use cases inside a single Application?

A: Yes, for example you can have multiple Widgets under various categories. Moreover, you can let your Users decide, which Widgets they want to use (or which business use cases they want to follow) by implementing PIRS Settings feature.

12. Useful Links

Table 51
Useful Links.

Description	Link
VXT CMS	https://samsungvx.com
VXT Developer Portal	https://developer.samsungvx.com
SSSP/TEP API Documentation	https://developer.samsung.com/smarttv/signage
Web Engine version support verification tool.	https://caniuse.com
Description: VXT Manual	https://docs.samsungvx.com/docs/dashboard.action
Description: VXT Official Webpage	https://vxt.samsung.com/

13. Appendix

12.2 Table of Figures (Tables)

12.2 Table of Figures (Figures)

14.Who to Contact for More Information

Table 52 Contact Info.

Region	Samsung Institute	Contact Info
Europe	Samsung R&D Institute Poland	a.honek@samsung.com a.grzeczkows@samsung.com m.petrynski@samsung.com c.pulawski@samsung.com
United States	Samsung Research Tijuana	m.regalado@samsung.com e.rincon@samsung.com s1.ruiz@samsung.com
Other	Samsung R&D Institute India-Delhi	k.prabodh@samsung.com devesh.ks@samsung.com sumit.gautam@samsung.com

Samsung VXT