# Wohola X

Release: 1.2.0 internal beta

Author: Pankaj Goel

Date: 1st December 2024

Document version: 1.2 (date: 1st December 2024)

Document version 1.1:

**Added ‘Chapter 5: Object Management’ Please download the latest version from github to get the code examples for newly added chapter.**

Document version 1.2:

**Added ‘Chapter 6: Role based access control’ Please download the latest version from github to get the code examples for newly added chapter.**

# Chapter 1: Getting Started

## Introduction

Wohola X is an enterprise grade solution for developing large scale web based solutions quickly and easily. It can be configured to handle websites of any size, scale or traffic.

Wohola X scales automatically based on server capability. There is no need for external software or any coding for the same. It has a very small footprint, very small code base and fast page processing times so even without any additional configuration it can handle large loads on relatively smaller servers.

Wohola X is the tenth version of Wohola, a long running platform. WX was written from scratch and has been developed with the mindset that one solution should fit all and should take care of potentially ALL problems faced in their web based solutions by developers, managers, testers, server or database administrators, site administrators, users and companies.

The development in WX is done with WXML a programming language designed specifically for Wohola X coding. It is an enhanced version of HTML so HTML/CSS developers can build most of the solution without knowledge of other programming languages, technical platforms, databases or software architecture.

WX manages most of the technical complexity automatically under the covers. WX also handles most database and back-end functions automatically so WXML developers only need HTML/CSS skills in general. Anything that is not already implemented can be implemented using javascript so there is no limitation on what can be done.

WX can handle any software development requirements. Since the underlying platform is normal HTML/CSS and Javascript which means anything that can be done on any technical platform can be done on WX. There are no limitations other that what is possible to be done with any other language or technology.

Since it is designed to be a one solution fits all model for web based solutions, we are targeting Wohola X as an alternative for not only development platforms like React, Angular, Laravel, Symphony, Django, Bootstrap, jQuery and other such platforms, but also as an alternative to CMS systems like WordPress, Joomla, Drupal etc. Going one step further we plan to take Wohola X to a stage where it could be seen as viable alternative to ERP systems like SAP or MS Dynamics.

The UI in Wohola X is WYSIWYG enabled, so it is easy to develop as there is no need to build additional administrative screens and solutions built on Wohola X are easy to administer as the site administrators can simply click on the item they see on the page and change it instead of having to sift through menu items and pages in admin panels.

## Installation

1. Install Mongodb and start it: <https://www.mongodb.com/download-center/community/releases>
2. Download Wohola X
3. Configure PATH in include the wx folder

## Test the installation

To test the installation go to the folder where you installed Wohola X and run the following command from windows command prompt.

wx file=sites/check/check.xml port=8000

Then in your browser go to the URL : <http://localhost:8000>

You should see the welcome message if the installation is successful.

This process must be running in order for that site to be live.

To kill the process use ctrl+c on command prompt.

## Observations

* If your computer has more than one CPU you would see can see that WX starts as many processes as the number of CPUs. This is the best configuration for live environment for maximising performance.
* So WX is automatically scalable. You simply increase server capacity and the WX scales automatically. If your application is very large which requires a server farm then WX can be configured for that too. WX can also handle database scaling for large applications.
* WX is enterprise grade solution that can handle any size application. For high bandwidth enterprise solutions you can contact one of the Wohola Partners for your project requirements and they can help.

## Running in single process mode

While developing and debugging WX applications you may need to run it in single process mode. To do so add –mode=debug to your command like this:

wx file=sites/check/check.xml port=8000 mode=debug

Now WX would run in a single process mode to make it easier during development phase.

## Running the demo site

To run the demo site run the following command:

wx file=sites/demo/demo.xml port=8000 mode=debug

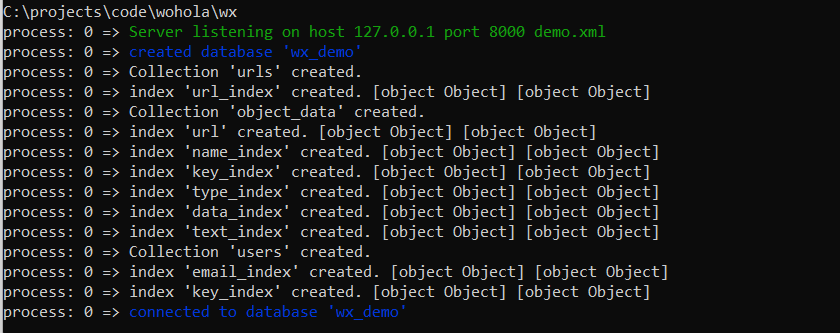
You would see following output on command prompt.

Running with chokidar

During development if any files change in your project folder and you would like the server to restart automatically then you can run it using chokidar like this.

chokidar file=sites/demo/demo.xml port=8000

This would restart the server when any of the project files change. Chokidar only runs the server in dev mode and should not be used in live deployment.



First time WX runs it creates the database, collections and the indexes needed for YOUR application.

WX manages the database internally so you don’t have to. It comes with a default DB which is sufficient for most needs of most applications. In the event you have some unlikely requirement that cannot be handled by default WX db there are mechanisms to customize the database or use a different database altogether. Those topics would be covered in advanced configuration techniques later on.

Now go to localhost:8000 to see your output. You only see some layouts with blank elements or dummy text. This is not a problem because in WX admin can manage the content after logging in. Let us try that next.

## Admin Login

To log in you need to create the first admin user. For that go back to the WX folder and run first-admin.

Enter the database name, in this case database name given in sites/demo/demo.xml file, email, password and user type as admin.



Now your have your first admin user ready.

Go to login page on the demo site and log in with these credentials. You would now see many icons and buttons appear. Hover over these icons to find out about them.

Admin can edit content, upload images and manage various elements through these buttons.

## Editing WX sites

Once logged in you can click on text to edit it. In demo these sections clearly state that you can click to edit those text elements.

On top left of image elements you see an upload icon.

Click on these icons and upload images wherever you see them.

To add items to banner, list or grid layouts on the page click on + icon at top right of these elements and items would be added.

Reduce the browser width to see that these items are mostly responsive. Further responsiveness can be configured easily.

## Observations

While developing the site developers do not need to worry about actual content. They can create the pages, page layouts and fully working functionality and the site admin can add the actual content later on after completion. This makes the development process simpler.

In general WX sites are inherently responsive.

The editing functions are all on screen, so reducing admin complexity for having to navigate through complex admin panel for large sites or for developers to have to spend time building additional screens for administrators.

# Chapter 2: Creating a new website

## Create a new website

Copy the demo folder to any folder outside of core WX system folder.

Change the name of the folder to any name you would like to give to your site. In this document let us call it ‘test-site’.

Go to test-site folder and change the name of the demo.xml file to test-site.xml.

Open the test-site.xml file in a code editor of your choice and change the name of the database to any name you would like to give. Let us give it a name wxdb\_test\_site.

Save this file. Your first website is ready.

You can run it with any of the following command.

wx file=sites/demo/demo.xml port=8000

wx file=sites/demo/demo.xml port=8000 mode=debug

The port number can be of your choice. We would learn how to make the site live later on. During development you can choose a port which is not used by any system services and run it from localhost:port.

The database is created automatically and as you go along you would see most of the database functions are also enabled automatically.

**Notes:**

1. Name of the site can contain dashes (-) and underscores (\_) but rest of the characters must be alphanumeric only. Its best to give the name in lowercase only. Mixed case names can cause issues in browsers when used as URLs.
2. Name of the database can contain underscores (\_). Rest of the name must be alphanumeric.

## Introduction to server file

Each site in WX has a starting point which is the server file. The demo.xml or test-site.xml files we talked about are server files for those websites. These are starting points for WX and are given as --file argument when you start your WX server from command line.

For demo.xml it looks like this:

#include \_system/system.xml

#define max-width   1000px

<db name="wx\_demo" />

#include pages/home.xml

#include pages/blog.xml

#include pages/contact.xml

#include pages/login.xml

You would notice #include and #define directives here which are similar to C/C++ programming languages.

## #include directive

#include is meant to include code written in other files. Technically you could have code for your whole WX site in the server file and you would not need any other files but that is not maintainable. You need to break the code into different files to make it easier to manage. You can then include different parts of code in different files wherever needed using #include directive.

Here we are including \_system/system.xml

This file contains all the system features called XTENS (short for extensions – and X as roman numeral for ten as well as X for Wohola X). As the name suggests xtens allow enhancing the features of Wohola X and WXML. Any feature that does not already exist in WXML can be added by creating a new xten. There is no limit of what type of functionality needs to be implemented using an xten. Anything that can be done with HTML/CSS and javascript can be implemented as an xten.

We are also including various page files in the server file.

## Path resolution

The paths to these files are relative to the folder where the server file is located.

However, there is no \_system/system.xml in your site folder. WX looks for a path in the site folder first if its not found there then it looks for the same file in the WX installation folder.

## Overriding system files

The mechanism for path resolution in WX is a very powerful feature as it allows you to override system features in your site.

**Examples for this:**

1. There are icons and other files being included by default from system folder. If you want to make one or more of the icons look different then you just need to put that icon file in your site folder with same relative path and file name. Then your file would be used.
2. There are xtens which implement default features. If you need one of the xtens to do something different than default processing then you create the file for that xten in your site folder with same name and same relative path. Then for that xten your file with your custom code would be used.

In both cases only the files you override would be used from your site folder and rest would be picked up from system folder. So you can selectively override certain files or features if the defaults do not meet your project requirements. This means that you are not tied to some particular function working in a particular way. You can make any system feature work differently if you need to.

This also allows you to not have to change system files risking the core system being corrupted for your custom project needs. You can only override small parts of features that are not sufficient for your project and let WX use the remaining features as default.

## #define directive

#define is to define any variables. Once defined these variables can be used anywhere in the code like this [#var-name].

Example variable used here is max-width. It is being used like this in page files:

<page url="\_home" max-width="[#max-width]">

So you defined the value for max-width variable in one place and then can use it everywhere. In future if you want to change this value you only need to change it at one place and that change would be affected in all files.

## Adding pages to the website

Create a new xml file in your site pages folder.

Give it a name which would be the URL for that page. For example if you want that page URL to be ‘blog’ then the file name should be blog.xml, if that URL is meant to be ‘search’ then the file name should be ‘search.xml’.

Now add the following code to this file.

<page url="my-url">

    my new page

</page>

You would see #includes at the end of your server file where all the pages are included.

#include pages/home.xml

#include pages/blog.xml

#include pages/contact.xml

#include pages/login.xml

Include the file in there in similar manner.

Now you can restart the WX process and then go to localhost:port/my-url to see your page.

**Notes:**

1. The URL names must all be in lowercase and should not contain any special characters. Underscores and dashes are fine. You may have dots also.
2. The folder and naming conventions are recommendations only. In WX there is no restriction on folder structure, database structure or file names. You can give any names or structures you want which you would learn as you use WX. The recommendations are for purposes of clarity while getting started or in some cases for allowing best use of core WX features.

## Header and footer

First let us look at the code for blog page to understand page code structure.

<page url="blog" max-width="[#max-width]">

    #include includes/header.xml

    <container name="page-body">

        blog page

    </container>

    #include includes/footer.xml

</page>

Here you see max-width variable being used. It would set the maximum page width to the value defined using #define. The page would be centred. If browser width is more than this width then the rest of the space would show background for the body. If its less than the max-width then page would span the whole width of the browser.

Also you see header.xml and footer.xml being included. If you have some code which would be used in more than one place then you should separate out that code into a separate file and include it wherever needed.

You also have container for page body. Let us understand what these page and container tags mean and where they come from.

## WXML and Xten Introduction

In this code example you see page and container tags. These are not standard HTML tags. These are xtens. There is a library of xtens that exists which can make things easier or automate certain tasks and you can develop custom xtens too.

To understand this think of WXML as nothing but standard HTML where all standard HTML tags work as it is. WX has a mechanism through which you can define new tags which can be used in normal HTML just like any other tag, with one difference, HTML tags do not do anything whatsoever, but WX tags take care of functionality including display, responsiveness, database management and anything else that needs to be done.

These new tags which can be defined in WX are called xtens and the new enhanced version of HTML that the result becomes after adding all these new tags is like a brand new programming language called WXML.

WXML is not limited to existing xtens (tags). WX has an architecture where new tags can be defined at will to take care of any features as needed or override existing xtens (tags).

**Notes:**

1. Technically you could have plain simple HTML/CSS and Javascript based application and it would work perfectly fine with WX being used as a server. But in that case you could use any other webserver like apache, IIS or nginx and you do not need WX. So while the capability to run a normal HTML/CSS, Javascript site exists in WX it is not what its meant for.
2. On the other hand since WXML is an enhanced version of HTML, CSS and Javascript (fron end and node.js backend javascript) combined into one language so all the features of HTML, CSS and Javascript are available and can be used within WXML. The xtens use HTML, CSS and Javascript to implement the required features anyway.

## Adding page to navbar

Now that you have a new page it can only be accessed from URL. To add it to navbar open includes/header.xml

You would find navbar code like this:

<navbar transform="uppercase" css-margin="0.5rem"

item-gap="15px" navitem-hover-style="color: blue">

<navitem href="/">Home</navitem>

    <navitem href="blog">Blog</navitem>

    <navitem href="contact">Contact</navitem>

    <navitem>

    <show role="public"><a href="login">Login</a></show>

        <show role="logged\_in"><logout>Logout</logout></show>

    </navitem>

</navbar>

Add your page to this navbar in the similar manner.

**Note**

If you notice while defining the page the home page was defined like this:

<page url="\_home">

While accessing it we have given the href=”/” in the navbar. It is the same concept that is used by all other webservers where they name the home page as index.html, index.asp or index.php. The reason is that you cant have a filename with a blank name or name = ‘/’. So blank or / URL has to have a different file name. Other web servers use index while in WXML we use \_home for the same.

Now you have learnt how to create a new website and how to add a page to this website. Let us look at adding code to this newly created page.

## Important Points to note

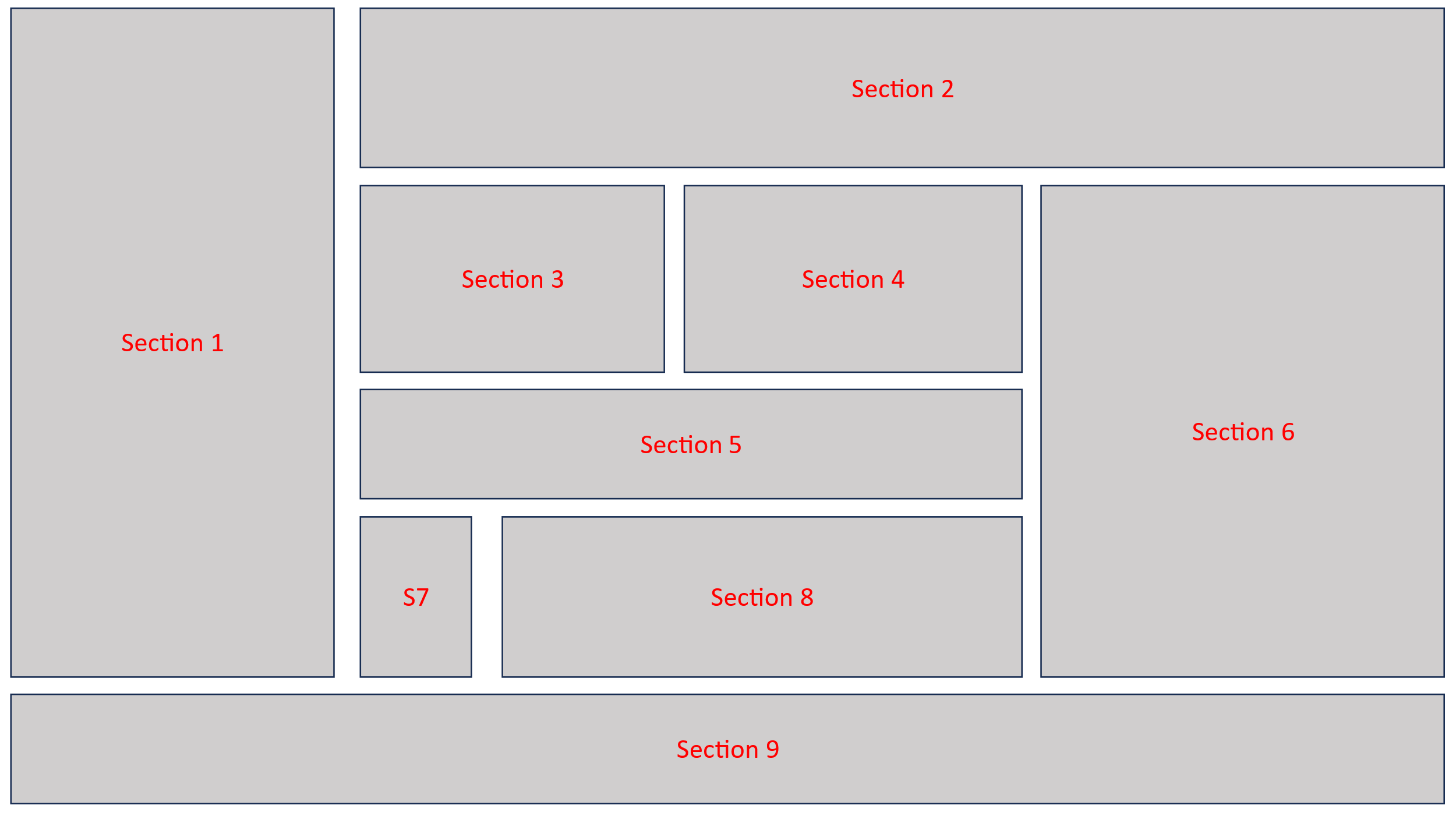
* All tags in WXML must be closed. In normal html there are many tags which are not normally closed e.g. img, input, br etc. In WXML all tags must be closed e.g. <img src=”sad.png” /> or <br/>. The server would shut down and give an error message if that is not done.
* Each node in WXML can have either text or other nodes as children but not both. So if text and children are both needed then wrap the text in another tag like span.

# Chapter 3: Creating page layout

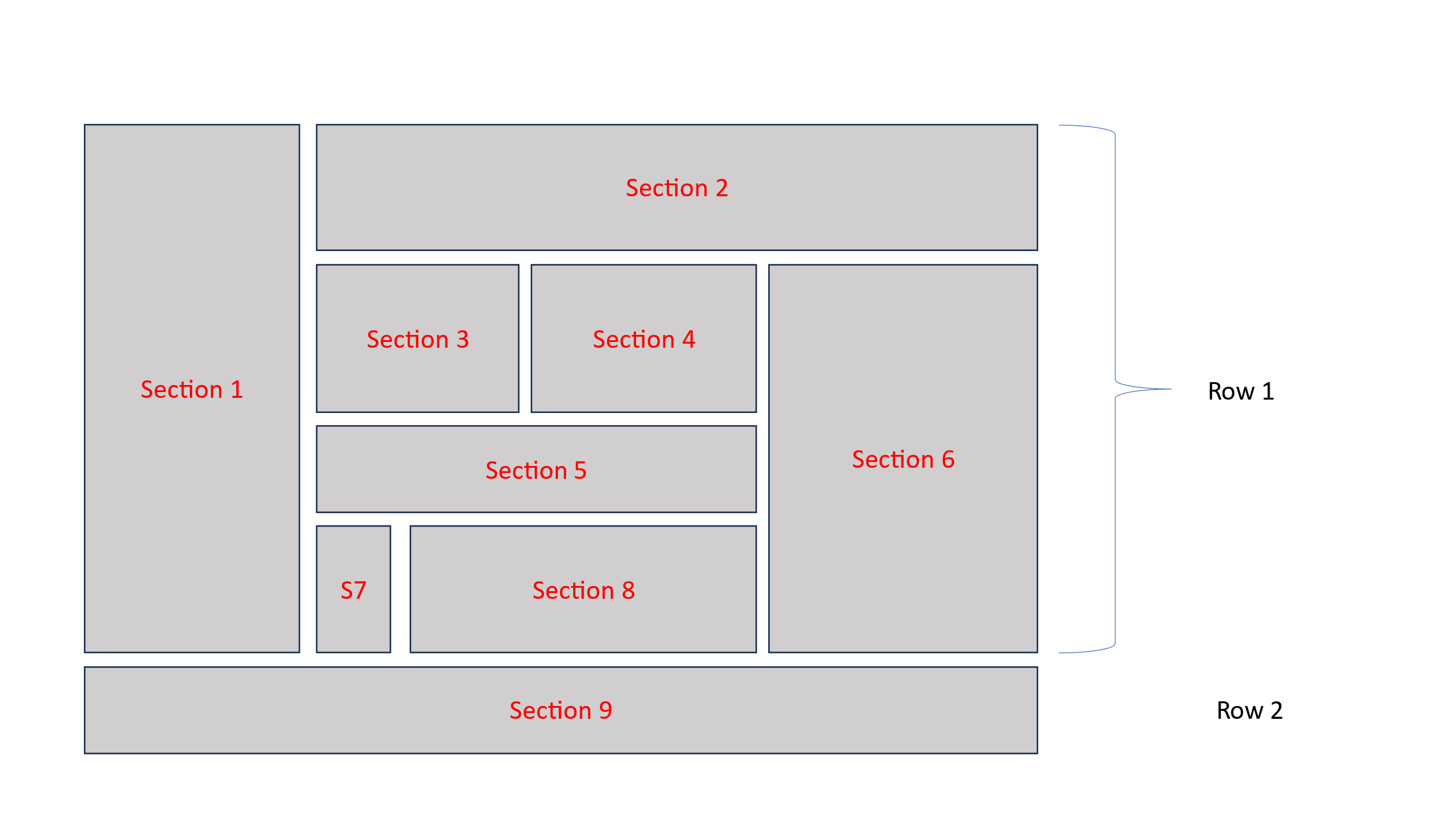
A page layout can be logically constructed by a series of nested rows and columns. For this purpose WXML has row and column xtens.

Let us create a complex page layout. For purpose of visualization and clarity we would give each element some height and a different background color.

This is the layout we would create.



Here we break down the rows and columns one by one



So we write:

<page url="layout-example" max-width="[$max-width]">

    #include includes/header.xml

    <container name="page-body">

<!-- section 1 to 8 -->

        <row>

            To be done

        </row>

<!-- section 9 -->

        <row css-background-color="yellow" css-height="100px">

            <center>

                Section 9

            </center>

        </row>

    </container>

    #include includes/footer.xml

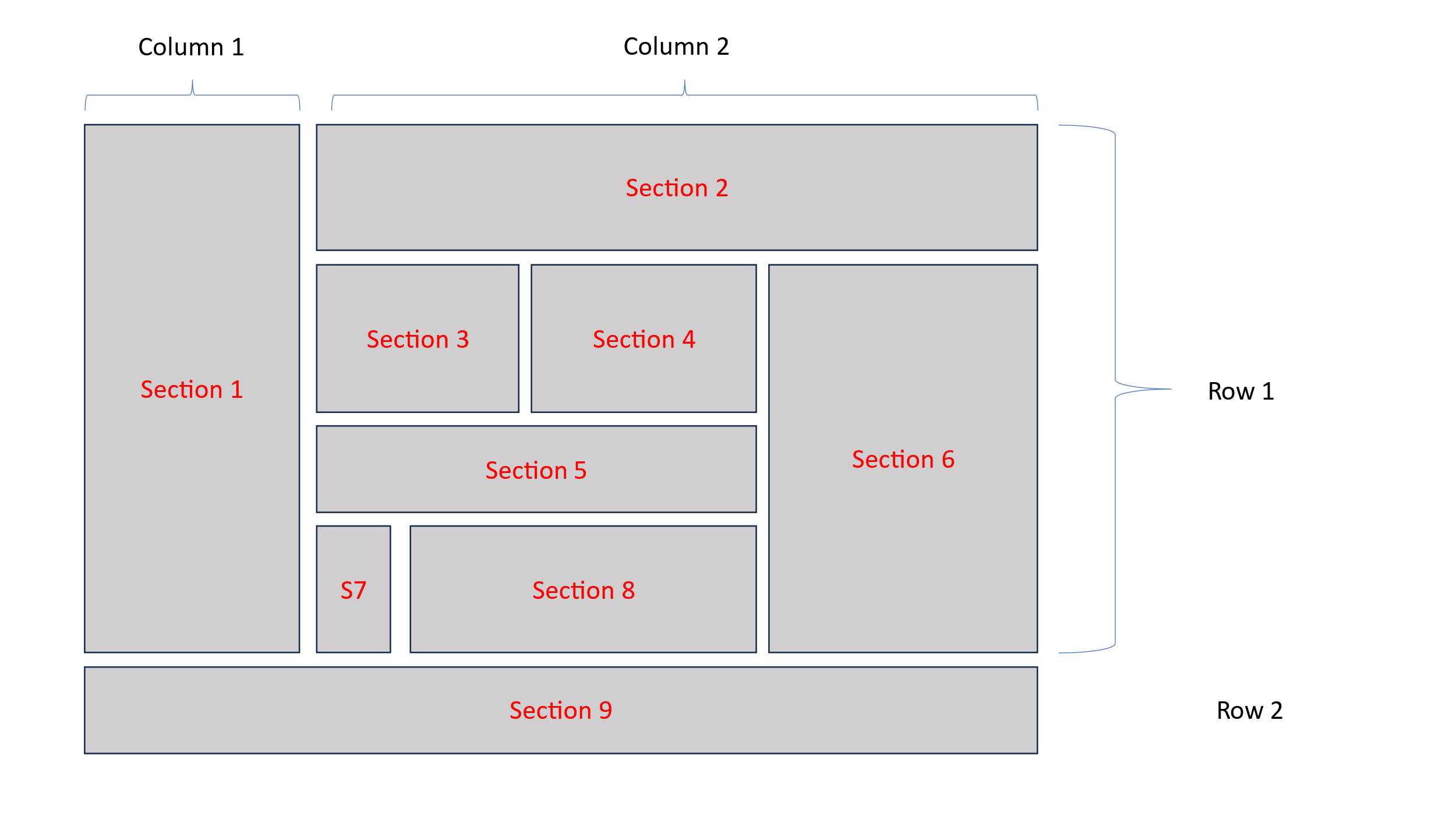
</page>

The color and height would help us see the layout.

**Notes:**

1. We have used center xten which would align the text into the middle of the row vertically and horizontally.
2. We have used css-background-color and css-height attributes to give some height and background color. Any css attributes can be used in this manner by prefixing them with css and giving them as attributes.

We break down this layout further. In Row 1 we have 2 columns:



So we write

<page url="layout-example" max-width="[$max-width]">

    #include includes/header.xml

    <container name="page-body">

        <row css-min-height="300px" css-height="fit-content">

<!-- section 1 to 8 -->

            <column css-background-color="lightgrey">

                <center>

                    Section 1

                </center>

            </column>

            <column ratio="3">

                to be done

            </column>

        </row>

        <row css-background-color="yellow" css-height="100px">

<!-- section 9 -->

            <center>

                Section 9

            </center>

        </row>

    </container>

    #include includes/footer.xml

</page>

We have given a ratio=”3” to second column because visually we can see that there is 1:3 ratio between these columns. If we omit this then all columns end up with same width. By giving ratio we can specify the relative ratio between columns with default being 1.

In this manner we keep breaking this down and end up with the final code like this.

    <container name="page-body">

        <row >

            <column css-background-color="lightgrey">

                <center>

                    Section 1

                </center>

            </column>

            <column ratio="3">

                <row css-background-color="yellow">

                    <center>

                        Section 2

                    </center>

                </row>

                <row>

                    <column>

                        <row>

                            <column css-background-color="brown" color="white" css-height="100px">

                                <center>

                                    Section 3

                                </center>

                            </column>

                            <column>

                                <center css-background-color="green">

                                    Section 4

                                </center>

                            </column>

                        </row>

                        <row css-background-color="red" css-color="white" css-height="200px">

                            <center>

                                Section 5

                            </center>

                        </row>

                        <row>

                            <column css-background-color="brown" css-color="white" css-height="200px">

                                <center>

                                    Section 7

                                </center>

                            </column>

                            <column ratio="5">

                                <center css-background-color="green">

                                    Section 8

                                </center>

                            </column>

                        </row>

                    </column>

                    <column css-background-color="blue">

                        <center>

                            Section 6

                        </center>

                    </column>

                </row>

            </column>

        </row>

        <row css-background-color="yellow" css-height="100px">

            <center>

                Section 9

            </center>

        </row>

    </container>

Notice how we removed other heights and only gave heights to section 3, 5 and 7 and rest of the columns automatically resized to fill the layout. In real project you should not give hard coded heights. With row and column xtens the column heights would automatically be adjusted as per the content height so there is no need to specify hardcoded heights.

Hardcoding is to be avoided in general except in very rare cases.

Also try to reduce the browser width to see that this is a responsive layout. Further responsiveness can be improved which we would look into when delve into more advanced topics.

The code for this layout example can be found in demo/pages/layout-example.xml

The output looks like this



# Chapter 4: Database Management

WX environment has 2 types of developers.

## WXML developers

The analogy for wxml developers in any other system would be the programmers who USE things like React, Angular, Laravel, PHP, Python etc to create solutions.

As far as wxml developers are concerned they simply develop the screen layouts and the database functions are automatically enabled as soon as they complete the screen layout and log in as admin. It feels like magic to them.

## XTEN developers

The analogy for xten developers in any other system would be programmers who create new libraries, products or APIs for systems like React, Angular, Laravel, PHP, Python etc and these libraries are then used by developers who use such systems to help them create solutions. Any xten created by xten developers becomes a new WXML feature which wxml developers can use to build solutions.

Xten developers need a little bit of knowledge of Javascript and databases, but even for them they get and receive the data simply by passing the context of the element. In simpler words they do not need to apply much time thinking about data.

WX figures out where to get the data from and where to update based on the context of the element, so even for xten developers there is no need for extensive coding or extensive knowledge of database etc, in majority of cases (90%+ situations in 90%+ projects globally based on my experience).

## Examples of automatic database management

### Images

In normal HTML developers use <img> tag for images. Instead if they use <image> xten then they just need to give a meaningful name to that image like so:

<image name=”product-image” [other optional attributes if needed] />

They do not even need to specify what image. They simply need to know what that image is meant for and pass that as attribute, e.g. name.

And rest is taken care of by WX internally based on the context of that element. Immediately that image element becomes a living entity where admin or allowed user roles can upload an image directly on the page.

An xten developer creates such tags which wxml developers simply call like normal HTML tags the features related to that tag become available to admin or authorised user roles who can then uploads the image as per their liking directly on page and internally WX figures out where and how to store and retrieve that image based on the context and how to link that whole process seamlessly.

This not only applies to images on static pages but also for complex data structures like products, customers, real estate listings, user profiles or any other objects that your application may need.

### Text data

Similarly, for text, instead of using various types of text html tags the developer simply uses one of the following:

<inline name=”customer-name” [other optional attributes if needed]>

Some random dummy text

</inline>

Or

<textedit name=”house-description” [other optional attributes if needed]>

Some dummy random text

</textedit>

The only difference between the two being one is meant for single line unformatted text while the other is meant for formatted text blocks where admin or allowed user roles can edit that text block, format it and more.

WXML developer writes that tag like above and provides the proper context like a meaningful name and other such things which are easy to figure out for the developer. During development there is no dependency on actual data as they can put any random strings of text hence simplifying the process. Once completed admin or allowed user can log in and put the real data by simply clicking on that text and editing it directly on the page.

## It feels like magic

Since this is not a normal thing for most developers to understand here it is again.

Yes, it feels like magic but it actually happens. As wxml developers you simply write the tag and all functionality including database functionality becomes available automatically. And once you have created the layout of the screen, anyone with right user role can log in and click on the text to edit it directly in place on screen or click on upload button on an image or video and change it. They can click on add button to add items to list or grid or banner.

Again, this capability is not limited to static pages only but for **ALL** business objects being used in your application, whether it be products, real estate listings, dating profiles, customers, purchase orders, invoices or any other type of object your project requires.

There is no need for you to know any technical things on what is the actual data to put in during development and how and where WX stores the data and how it retrieves it after the data is inputted by admin or authorised roles.

You can check out examples of the code written for the demo site and search for inline, textedit, grid, list, banner and image xtens and see how much code exists for each element, and this is how much code wxml developers write for any project. Then you can log in as admin with the admin user you created using first-admin account and then go back and see the result of that code by going to each element.

### Data management for xten developers

Lets take an example code for retrieving correct data for xten developers.

For getting data for your text or other such elements an xten developer may write this code:

    <load>

        const value = await database.inlineValue(node, context, xten);

        if (value) {

            node.text = value;

        }

    </load>

This is an example of the type of code an xten developer would need for writing data retrieval function for something like an inline function.

The developer simply calls a function and passes the node, context and xten. None of these things are defined by that developer and developer can simply copy and paste this.

These variables were defined internally by WX while parsing the page before serving. WX tries to figure out the context and populates these variables internally when the page is served and then when making any calls developer simply passes back the same variables that it received from WX mostly as is.

## An analogy to understand the context

The simplest question that comes to mind is that if the xten developers only need to pass back the same variables they received from WX then why do they even need to do it? Another question is what is the context anyway?

To understand this let us take a real life example. Think of WX server or any other webserver as a call centre. Let us say you call a call centre to open bank account. They give you account number IBAN/Swift/sort/IFSC code and give you a pin. So they now have your context. Compare it with loading the page. So while loading WX parses the page figures out the context of the data items based on hints like name given by wxml developer.

Now after opening the account you call the same call centre for some account enquiry or transaction. By the time you opened the account and called again that call centre has received many calls and when you call again they have no way of knowing who you are, so you need to provide the same data they gave you while opening the account so that they could figure out who you are.

Similarly, in WX or any other web server the page gets served to many people and when user clicks on a button that leads to an update transaction on the server, WX does not know which page or what item and what context that data and transaction refer to. So just like you give back the same account number back to the bank which they had given you in the first place you need to provide the same context back to WX so WX knows which page, and which context this update request came from.

Of course some inputs are required from wxml developer while coding. Rest is figured out by WX.

## Examples of input required from wxml developers.

### name attribute

The most important item wxml developer needs to pass is the name of the element. They need to pass proper names, e.g. if its company name they need to provide something meaningful like so:

name=”company-name”

if its an image xten which deals with logo element then they need to provide something like this:

name=”company-logo”

This is easy to figure out for wxml developers on what type of name to give to their elements. Its based on their requirement. If its customer address they can give a name that is relevant to that textedit block like “customer-address”.

### object-name attribute

This is an optional parameter. By default only name is required and this can be skipped so the data would be populated based on the context of the current page.

Let us take an example of when a wxml developer would need to provide this attribute. Let us say the current page for which the code is being written is “contact” and since “company-address” is a data item related to contact page on “contact” page the object-name attribute is not required. But on the other hand let us say on the “complaints” page also there is an address field which is supposed to be the same address then developer can write

name=”address” object-name=”contact”

this would tell WX that the object being referred to is related to the same object that was on contact page.

### list-child attribute

in the demo example there are examples of nested data structures like list, grid and banner which means there are nested objects involved. For such objects you can notice the relevant xtens use attribute list-child=”true”. This tells WX that the data item being referred to is a child of the parent element which could be a banner or grid or list.

Again, here you need to indicate to WX that it is a child of the parent by saying list-child=”true” and nothing more. Rest is taken care of by WX based on that context hint.

### Defining new object-name(s)

By default if you skip the object-name attribute WX tries to resolve that data item based on the scope of the current page. Let us say you want some particular data item which is applicable to all pages or some data item that applies for all product pages. Then you can simply define new scopes by giving your own name to object-name.

For example if you want a particular data item to be applicable to all pages or objects then you write object-name=”all” or if you want a particular data item to be applicable to all product pages then you write object-name=”products”. You can simply define any scope by giving some name which is meaningful. These can be names given by you based on what objects you are dealing with. For example you could decide that some field should be same on all invoices then for that field you just write object-name=”invoices” and that scope would be defined automatically within the WX context and you can use that scope for any fields which are supposed to be applicable to all invoices.

Similar simple data management strategies apply for complex business objects as well whether it be any object like blog post, news item, product, invoice or any other such object needed within your application. You simply give indications to WX on what your intention with that data item is and WX manages the rest. We would learn more about those once we get to the topic of object management.

In summary, for wxml developers the type of things to figure out are name of the object, whether that item is child of a list type of object and specify list-child=”true” it it is. Another thing for them to figure out is object-name whether it be an existing object that was defined by them previously or an object they decided they want to have on the fly which may not even exist. This is the only database or backend knowledge the wxml developers need.

The code in demo site should give a clear indication on the concepts mentioned above.

# Chapter 5: Object Management

## Object basics

Until now we looked at the page elements for static pages. This is not sufficient for any project needs, so we would look into a quick start introduction to object management in WX.

Normally in any application you have many entities like customers, products, orders, invoices, purchase orders, blog posts, news, or any other such entity. In WX these entities are called ‘object-type’. Normally a separate database table or collection would be created for each such entity and then a large amount of code is written for managing each of these, however in WX only what that object looks like on the screen is defined just like the page elements above and rest is taken care of automatically by WX.

Each of these entities could have many records and each of these records are called an ‘object’ in WX.

In WX all objects are managed in the same manner by default. The theory behind it is that adding, deleting, editing, paginating, searching algorithms for the objects are normally same for most objects in most cases, so there should not be any need for hiring skilled developers who write code for managing each object in each project separately. In general 90% of the situations in all projects around the world the same algorithm is applied repeatedly again and again by developers.

So WX takes these 90% of the situations and manages them through a standard algorithm so there is no code that needs to be written for these 90% of situations. For exceptional situations or remaining 10% where something different is needed WX can be customized. So it can handle all kinds of objects that may exist in your application.

## Blog post example

Let us take a simple ‘object-type’ like ‘blog-post’. Whether the object is simple or complex it is managed in the same manner so this example would be a starting point as a quick start.

Let us assume our object-type=”blog-post” has 3 data elements name=”blog-image”, name=”blog-title” and name=”blog-description”.

While WX knows how to manage the object in the background it does not know what you want. So you need to at least tell WX something. That something is you tell WX how that object looks like on screen. WX does not know how you want the screen to look like and where you want to show what data item, so that is the bare minimum you need to specify. You do that through ‘object-config’ xten.

How to manage that object under the covers is then done by WX.

object-config=”blog-post”

To create an object-config let us create a file called blog-post.xml and put it in includes folder.

There are mainly 3 elements that need to be defined in object-config:

<object-config name="blog-post">

    <page name="default">

    ...

    </page>

    <object-view name="default">

    ...

    </object-view>

    <list-view name="default">

    ...

    </list-view>

</object-config>

For each of these views define what that object looks like on screen.

You can have multiple views, e.g. you may have a product where on main product page you want to show all details but on a summary page you may want to show a subset with a different layout. In that case you can create multiple views with different names. At least one view must be name=”default”.

## object-list in object-config

Ideally you would define this view first because through this view the default feature of adding new objects is enabled.

This view defines the ways that object can be displayed in a list and what each item within a list looks like. You may wish to display that list in any format, be it table, ul or div.

This view has 3 elements

* data-view: This is the main wrapper within which the list-items would be displayed. Example of this could be that you wish to have a header and footer for each list.
* list-item-view: This is the view that defines what each item in that list looks like.
* empty-view: This view defines what should be displayed if there is no data in that list.

Let us take the code example. This example can be found in demo/includes/blog-post.xml file.

    <list-view name="default">

        <data-view name="default">

            <div>Blog Post List</div>

            <div>

                <list-items />

            </div>

            <div>my footer</div>

            <object-list-toolbar />

            <paginator object-type="blog-post" pagesize="[$pagesize]" list-style="padding: 10px;" />

        </data-view>

        <list-item-view name="default">

            <div name="list-item">

                <image name="blog-image" aspect-ratio=".75" object-name="[@object-name]" />

                <gap height="10px"/>

                <textalign>

                    <inline name="blog-title" object-name="[@object-name]">

                        Blog Title

                    </inline>

                </textalign>

                <gap height="10px"/>

                <value name="blog-description" summary="40" readonly="true" element="div" object-name="[@object-name]">

                    Blog Post

                </value>

                <gap height="10px"/>

                <a class="continue-link" href="[@url-default]">

                    <span>Continue Reading...</span>

                </a>

                <object-item-toolbar />

            </div>

        </list-item-view>

        <empty-view name="default">

            <div>list is empty</div>

            <object-list-toolbar />

        </empty-view>

    </list-view>

Here we have defined the 3 views

* empty-view contains a message that the list is empty. You can create any layout in this view. It also has the default object-list-toolbar. This toolbar allows you to add objects.
* data-view defines what to show above and below the list and where to place the list-items.
* The list-item-view defines what each item looks like.

Important thing to notice is how the data items are being managed. The data elements all have object-name=”[@object-name]”. This allows for WX to know this data item is linked to the object in question which would be determined at run time by WX.

There is also [@url-default] which leads to <page name=”default”>. If you had another page view like <page name=”summary-view”> then that page could be accessed with [@url-summary-view].

The reason for indicating the data item belongs to the object is that the same view could also contain data items not linked to the object, so which items are linked to the object are indicated using name=”[@object-name]”.

The resulting items are placed in data-view where <list-items/> placeholder is. This is the position where multiple copies of list-item-view would be substituted with actual data and placed.

Notice how object-list-toolbar and object-item-toolbar are placed. These can be placed anywhere within the object. The list-toolbar contains add object button currently while item-toolbar contains delete button. More buttons can be added for any custom requirements by xten developers.

Notice how paginator has been added.

There are further examples of list-views in demo/includes/blog-post.xml where the lists use the format of table-tr-td in one example and ul-li format in another.

## object-view in object-config

This view defines the what an individual object looks like on screen. This object can be embedded anywhere in any page by <object name=”blog-post-oi8q-ywuw” /> where ‘blog-post-oi8q-ywuw’ is the WX generated name for the object.

To find out the name of a particular object go to the page for that object. The URL is of format <object-type>-<page-view-name>-<system-generated-id> while the object-name is of format <object-type>-<system-generated-id> so removing the <page-view-name> from URL yields object name. Using this name that object can be embedded anywhere in the application if need arises.

## Page View in object-config

The page view is defined like <page name=”some-name”>. This defines what the main page for that object looks like.

Example would be product-details page which shows details of each product. This page may have a header, footer or other items like any other page, and in the middle somewhere there would be details of a particular product.

In this example the product details page would be defined as page view and the section where the object details are placed would be defined as object-view.

Notice how <object name=”[@object-name]” is embedded in the page view.

Using object-list

There are examples of using the 3 different types of list views created for blog-post on demo/pages/blog.xml

<object-list object-type="blog-post" pagesize="6" orientation="row" columns="3" item-gap="15px" />

<object-list object-type="blog-post" list-view="table" page-size="6" />

<object-list object-type="blog-post" list-view="list" page-size="6" />

In first example since default list-view is being used the list-view attribute has been omitted. In remaining 2 lines list-view=”<name>” allows you to use a particular list-view.

# Chapter 6: Role based access control

Wohola has inbuilt role based access control. By default there are following roles which are defined admin, logged\_in and public. More roles can be defined as required by an application using <roles> property in the server XML file.

<roles>designer, hr-manager, employee</roles>

These can be arbitrary roles. When a user is created that user is given one of these roles. For example using first-admin utility you can create a user with matching roles.

Once the roles are defined and users for those roles are created then the access for these roles can be controlled within WXML. There are different ways of doing this.

In Xten you can define <allow> property to set which users have access to editing functions of that Xten. This becomes default of the Xten but this can be overridden at the node level while calling the Xten.

<xten name="my-xten">

    <allow>admin</allow>

    <!-- other Xten properties -->

</xten>

<!-- the code where you call this Xten -->

<my-xten allow="designer, admin" />

In this case the Xten default is set to allow but at node level the WXML programmer overrode the default and allowed editing of the Xten to designer and admin.

You can also show and hide various elements based on role. This can be done by <show> Xten.

<show role="public">

    <a href="login">Site Admin Login</a>

</show>

<show role="logged\_in">

    <logout>Site Admin Logout</logout>

</show>

In this case when the user is logged in user would see the logout link and if he is logged out then he would see the login link.

Using <show> various components in the page can be shown or hidden based on the role.

When the user has access to an editable component during page load a class ‘wx-state-editable’ is added to the element. Your CSS can take advantage of this and show and hide the editing functions based on this class.

You can also call function elment.checkAccess to find out if the given user has access to perform editing on that Xten.

<xten name="my-xten">

    <css>

        .wx-xten-\_list .list-item-toolbar {display: none;}

        .wx-xten-\_list.wx-state-editable .list-item-toolbar {display: flex;}

        .wx-xten-\_list .list-toolbar {display: none;}

        .wx-xten-\_list.wx-state-editable .list-toolbar {display: flex;}

    </css>

    <!-- other Xten properties -->

    <initElement>

    <%

        const allowed = element.checkAccess();

        if (allowed) {

            /\* do what needs to be done if user is allowed access \*/

        }

        else

        {

            /\* do what needs to be done if user is not allowed access \*/

        }

    %>

    </initElement>

</xten>

On server side the access is automatically checked during each transaction before calling your transaction function so you do not need to check access again. Only if the user is authorized to perform the transaction then only your transaction function would be called.

To restrict access to a page based on the role you can simply add allow attribute to your page node.

<page url="public-page">

    <!-- your page code -->

</page>

<page url="my-page-1" allow="logged\_in">

    <!-- your page code -->

</page>

<page url="my-page-2" allow="admin, designer" redirect="redirect-page">

    <!-- your page code -->

</page>

This example shows different ways of restricting the page to a given user. If the user does not have access to this page then the user would be taken to redirect url. This can be any page within the site. If redirect url is not provided user would be taken to login page by default.

# Conclusion

This concludes the quick start guide.

This only covers some basic initial concepts to get you started. There are a lot more advanced use cases and advanced configurations that are possible including writing xtens for any requirements that are not taken care of by the default system xtens.

A reference manual for existing xtens and a detailed developer guide to wxml would be published shortly.

# Thank You.