|  |
| --- |
| macmillan |
| Platform-X Shell |
| Specification of Core Services and Client API |
|  |
| **David Newman, NorthPoint Solutions, LLC** |
| **9/29/2011** |

|  |
| --- |
| This document is the specification of all core services provided by the Platform-X Shell as well as the Platform-X Client API. |

Contents

[Overview 4](#_Toc311537775)

[Basic Architecture 4](#_Toc311537776)

[The Data Store 5](#_Toc311537777)

[The Skeleton 5](#_Toc311537778)

[The Manifest 7](#_Toc311537779)

[Common Services 7](#_Toc311537780)

[Page Layout Engine 7](#_Toc311537781)

[Pages and Components 7](#_Toc311537782)

[Navigation 8](#_Toc311537783)

[Page Layout Hierarchy 9](#_Toc311537784)

[Common Metadata 9](#_Toc311537785)

[Page Items 11](#_Toc311537786)

[Zone Items 11](#_Toc311537787)

[Component Items 13](#_Toc311537788)

[Menus 15](#_Toc311537789)

[Common Metadata 15](#_Toc311537790)

[allow 15](#_Toc311537791)

[bfw\_display\_flags 16](#_Toc311537792)

[bfw\_edit\_flags 16](#_Toc311537793)

[bfw\_menu\_callbacks 16](#_Toc311537794)

[Menu 17](#_Toc311537795)

[MenuItem 17](#_Toc311537796)

[Methods 18](#_Toc311537797)

[MenuEditorCollection 18](#_Toc311537798)

[MenuEditor 19](#_Toc311537799)

[Methods 20](#_Toc311537800)

[Client API 20](#_Toc311537801)

[The PxClientAPI object 22](#_Toc311537802)

[Event System 22](#_Toc311537803)

# Overview

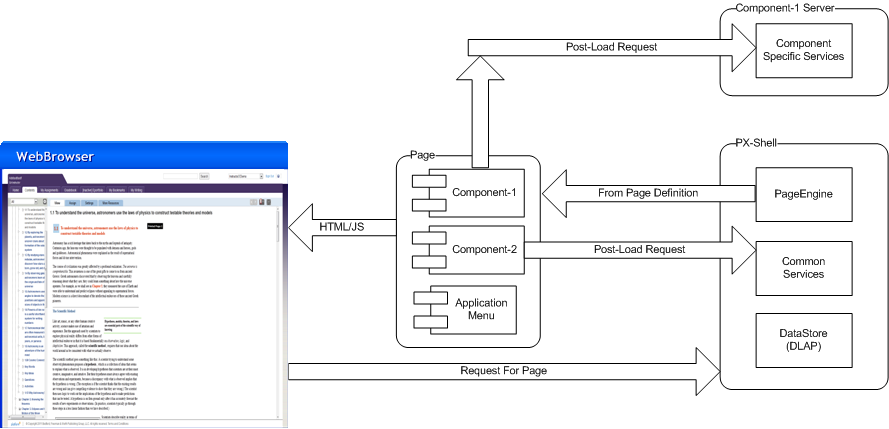
The Platform-X Shell is a web application that acts as a host for individually developed and deployed components. The purpose of the shell is to provide a set of standard APIs through which components can communicate with each other and the shell itself. Components will be responsible for rendering user interfaces that are specific to their requirements and performing whatever data access is necessary to populate them.

In order to speed up development of components the Platform-X Shell provides a set of APIs that components can use for simple data access. The shell also specifies all hard and implicit interfaces and protocols that components must implement in order to be successfully hosted.

## Basic Architecture

The Platform-X Shell is an ASP.NET MVC3 application that hosts a dynamic set of loosely coupled components. Each component is allowed to determine its implementation including technology stack, hosting technology, and hosting location. In order to meet these requirements the Shell has been designed to communicate with components through a set of well-defined APIs.

The following image illustrates the basic architecture of the Shell and how it would host an existing page:



As can be seen in the image above, a logical page that a user is presented with in the browser is decomposed into Components. Each component is ignorant of the others that may be on the page and is only able to communicate through a set of standard APIs. The definition of which Components are on a given page is stored as metadata by the Shell as one of its core functions.

The Shell can be looked at as an eco-system for hosting Components and other special case functionalities such as Menus and authentication. In order to serve all of these needs the Shell has been divided into the following areas of functionality:

* Data Store – this is the location where all metadata, and optionally custom component data, required by the Shell is stored
* Common Services – this is a set of services to provide basic access to the Shell’s features and the Data Store
* Page Layout Engine – this is the heart of the Shell and defines how pages get created and how components must behave in order to be hosted in them
* Client-Side API – this is a javascript library that allows components to communicate with each other in a loosely coupled, event based manner

The following sections describe each of these areas of functionality in detail.

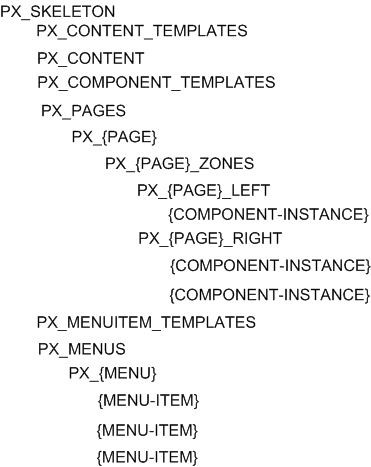
# The Data Store

The Platform-X Shell provides a standard repository in which all content and metadata is stored. The repository relies on a strictly defined, hierarchical structure in order to keep data organized by type. There are two logical structures in the data store: the skeleton and the manifest.

### The Skeleton

The skeleton is a hierarchical structure that is used by the Platform-X Shell to store application level data including the set of available content types, installed components, and page layouts. Essentially, all data required for the shell to keep track of its own internal state and host components is stored in the skeleton.

The following is a representation of the skeleton structure, and a basic description of what data each part of the skeleton stores. Sections of the skeleton will be referred to throughout the rest of the document with more detailed information about what data is contained in that section and how it applies to components or specific APIs.



**PX\_SKELETON** - Acts as the root of the skeleton structure. All elements of the skeleton descend from it.

**PX\_CONTENT\_TEMPLATES** - Acts as a container that stores each content type's template.

**PX\_CONTENT** - Acts as a container for all content created in the system as well as editorially curated content.

**PX\_COMPONENT\_TEMPLATES** - Acts as a container that stores each components template.

**PX\_PAGES** - Acts as a container that stores the definition of every page in the system.

**PX\_{PAGE}** - Represents a logical page as seen by the user. {PAGE} is the name of the page, e.g. Home.

**PX\_{PAGE}\_ZONES** - Acts as a container for all of the Zones defined for a specific page.

**PX\_{PAGE}\_LEFT** - Example of a Zone defined for the page named {PAGE}.

**PX\_MENUITEM\_TEMPLATES** - Acts as a container for all of the types of menu items.

**PX\_MENUS** - Acts as a container for each menu defined in the system.

**PX\_{MENU}** - Represents a logical menu as seen by the user. {MENU} is the name of the menu, e.g. Primary Nav.

### The Manifest

The manifest is a specialized area in the data store that stores content items that have been assigned to students. The topic of what it means for an item to be assigned, completed, graded, etc is outside the scope of this document. However, it is important to identify this separate storage location as it influences other areas of the shell and the APIs it provides.

Unlike the skeleton, the manifest does not have a defined structure. The only exception to this is that every item in the manifest must ultimately descend from the PX\_MANIFEST root item.

# Common Services

The Server API is a set of compiled class libraries that provide access to the underlying data store and other back-end functionality. The Server API can be used by components as a foundation for building their server side functionality. This is most useful for components that load or store custom data, new content types, etc.

As necessary, some Common Services can be exposed publicly as routes in the Shell Application. This will allow them to be accessed from the client-side by Components using standard AJAX techniques.

# Page Layout Engine

The Page Layout Engine is the part of the Platform-X Shell responsible for managing the pages that have been defined for a course, what zones are defined on those pages, and what components exist in the zones. The Page Layout Engine is also responsible for driving the storage of and interaction with Menus.

## Pages and Components

Web applications are built around the concept of pages. Conceptually a page is the sum total of all content and assets that a user sees and provides common facilities like navigation to different pages and other content items. Traditional web applications represent the concept of pages using static assets like HTML files and potentially have some dynamically created pages by making use of web frameworks like ASP.NET, PHP, etc. to construct the HTML on the fly.

The traditional structure of web applications requires them to be deployed as monolithic applications, which makes deploying changes an all or nothing proposition. In order to avoid this, a different approach is required for Platform-X.

Instead of statically creating pages, the Platform-X Shell instead defines a structure that represents a page and all of the information a page requires to render itself. The definition for the page is stored in the skeleton structure defined previously and is retrieved when the page is accessed by a user. The shell is only responsible for loading the page definition and setting up any components that may be on the page. The components themselves decide what further data they require, if any, and generate their own markup to be delivered to the client.

The Platform-X shell divides logical pages into sections called zones. Zones themselves contain components which are responsible for rendering whatever functionality they are designed to provide. Some components are display only, while others allow the user to provide input or enter new content into the system.

Aside from defining the concept of what it means to be a page, the Platform-X Shell provides all of the necessary server and client-side APIs for managing the contents of a page and allowing components to communicate with each other. Since the shell merely determines what components are to be rendered it is possible to have components that are hosted on separate physical servers or implemented in technologies different from that of the shell itself.

## Navigation

The Platform-X Shell provides users with the ability to navigate between logical pages in the application. Similar to pages and components, the shell only provides definitions, storage, and APIs for what constitutes a menu and its items. New types of menu items can be created outside of the shell and deployed separately.

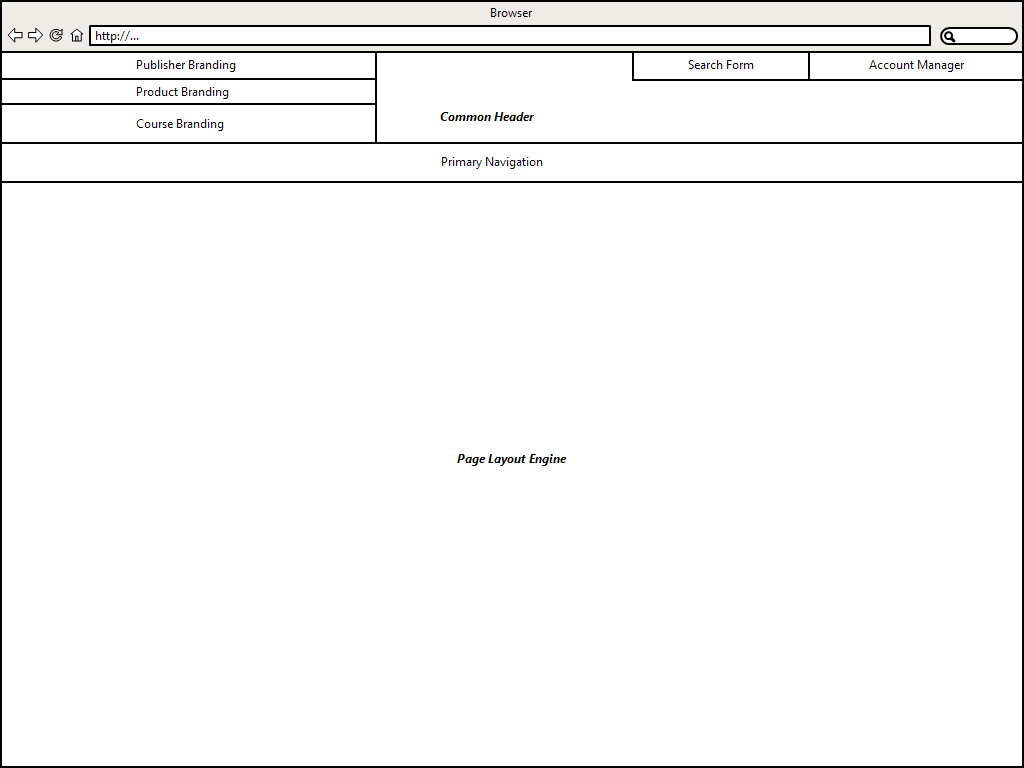
There are two forms of navigation supported by the shell. The first type of navigation is referred to as primary navigation and occurs when the user selects an item from a menu that was defined in the shell. The second type of navigation is inter-content navigation which can be initiated by components through the Client API.

Primary navigation is a familiar concept in web applications and is generally used to navigate between logical pages in the application. For instance, a user on the home page may wish to access the Assignment Center and so will select that menu item. The result in such a case would be for the user to be taken to the Assignment Center page.

Secondary navigation as defined by the Platform-X Shell is less obvious and takes many forms. For example, when a user selects and item from the Content Browser's Table of Contents the user is not taken to a different page. Instead, the Table of Contents component uses the client API to inform the shell that a content item has been selected. At that point, any components listening for that event can load the selected content item for the purpose of display.

A further use of secondary navigation via the Client API is inside of user or editorially created content. For example, authors of e-book content preloaded into a course could use simple HTML anchor tags with client side behaviors to tell the Platform-X Shell that it should navigate to a different page or content item when the link is clicked.

The following image represents the standard page envelope which includes the site header and footer. The center area is where the logical page is rendered including all of its zones and components.



## Page Layout Hierarchy

Logical pages in Platform-X are hierarchical. The root item on any logical page is a Page item. Pages contain one or more Zone items, which in turn contain one or more Component items. Using this structure it is possible to construct an infinite variety of logical pages each with a layout specific to its purpose. The following sections describe each part of a logical page in turn.

### Common Metadata

There are certain metadata that are used across the items used in the Page Layout Engine. This section describes them in a single place for the sake of efficiency.

#### bfw\_type

The bfw\_type element is used by Platform-X to mark content items in a way that represents what the item is logically used for. For example, each content type supported by Platform-X has a unique value for the bfw\_type field. When Platform-X sees this value it then understands how to interact with the item.

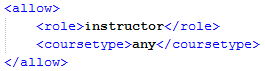
In the case of the Page Layout Engine, the bfw\_type element tells the engine whether the item is a Page, Zone, or Component.

#### bfw\_subtype

The bfw\_subtype element is similar in function to the bfw\_type element except that it provides more details. For example, there is a bfw\_type of Component, but each *type* of component has its own bfw\_subtype. For instance the CustomWidget component has a bfw\_type value of Component and a bfw\_subtype value of CustomWidget.

#### allow

The allow element restricts access to a Page, Zone, or Component. The following listing shows an example allow element.



As can be seen in the above listing, the allow element has two child-elements. Role restricts access to users in the role specified, e.g. instructors will make it so that only instructors have access to the Page, Zone, or Component. The coursetype child-element restricts access to a specific type of course, e.g. product makes it so that access is only granted on a product course.

The final decision of whether or not to allow access to a Page, Zone, or Component is based on the AND of the current user's role and the type of course being accessed. Both the role and the course type must match the value of the allow element in order for access to be granted.

The allow element is not used on its own, but rather as a child element for other metadata. When there are multiple allow elements the effect is an ORing of all of the allow elements. Therefore if the restrictions of any allow element is met, the access will be granted.

#### bfw\_display\_flags

The bfw\_display\_flags element contains child allow element which determine whether or not the Page will be rendered for the user. The allow elements work as previously described and the current user's role as well as the course type being accessed must match the values specified.

#### bfw\_edit\_flags

The bfw\_edit\_flags element contains child allow elements which determine whether or not the Page can be edited by the user. The allow elements work as previously described and the current user's role as well as the course type being accessed must match the values specified.

The Page Layout Engine applies edit flags based on hierarchy. This means that even if a user is allowed to edit a Page they may not be allowed to edit a specific Zone or Component. In order to edit any item, the user must meet the constraints specified by the bfw\_edit\_flags element on the item itself as well as on the items higher in the hierarchy.

### Page Items

Like all data constructs in Platform-X Page items are XML documents that contain metadata describing their behavior. Each Page item has a name that describes its logical purpose in the application. For instance there may be a Page item named "Home" that is used to render the course's home page.

Pages also contain other metadata values that determine whether or not they are editable and if so by whom. The following listing shows a representative XML document of a Page item.



As can be seen in the above listing, many elements in the item's XML document are from the underlying data store. For example, the "type" element indicates the underlying item used to represent a Page in the Page Layout Engine is a Folder primitive. The following sections describe each of the Platform-X specific elements and how they influence the Page Layout Engine. Common metadata have already been defined in a previous section and will be ignored here.

### Zone Items

Zones are items that contain Components. The purpose of a Zone is to act as a container for Components on a Page and determine where each Component is to be rendered. Each Zone may configured with a list of the types of Components it allows as well as restrict display and editability based on user role and course type.

The following listing shows an example XML document representing a Zone item located inside of the PX\_HOME Page.



The following sections provide details on metadata specific to Zones.

#### bfw\_allowed\_components

The bfw\_allowed\_components element specifies which types of Components may be placed inside the Zone. Each Zone allows this type of configuration because depending on the design of a page space for each Zone may be limited and therefore only certain size Components may be used. A typical example of this is on a home page where there are two Zones. One Zone might be significantly larger than the other which means that only certain Components will fit naturally into either.

The "component" element is a child of bfw\_allowed\_components that describes the approved components. Each component element stores the unique id of the component type's template as well as the human readable display name of the component. The human readable name is what the user sees when they are editing the Zone.

### Component Items

Component items are the most diverse type of item in the Page Layout Engine. Component is a general term that covers everything from secondary navigation, content display, and home page widgets. The Client and Server APIs responsible for rendering and interacting with components are written generically so that they apply to all components regardless of their purpose.

Like Page and Zone items, Components make use of many common metadata such as bfw\_type, bfw\_display\_flags, etc. There are also a number of metadata elements specific to Component items that influence how Components are integrated with the shell. The following listing shows an example XML document for a Component. Subsequent sections then describe each of the elements that are specific to Component items and how they are used by the Page Layout Engine.



#### bfw\_component\_callbacks

The bfw\_component\_callbacks element is used to connect the Component to the Platform-X Shell. The Platform-X Shell specifies an implicit interface for Components that must be implemented in order to be successfully hosted in the shell. Each required method of the Component Interface is represented in the bfw\_component\_callbacks element as a "method" child-element.

The method element has several attributes that indicate how the Component's implementation of the Component Interface can be accessed. The exact attributes used by the Page Layout Engine depend on whether the component is being hosted internally or externally.

The "controller" and "action" attributes are used by internally hosted components that have their routes registered in the Platform-X Shell's routing table

## Menus

As noted previously, the Page Layout Engine is also responsible for storing the metadata necessary to support customizable menus in Platform-X. Menus provide a fundamental method of navigation in the application and can be created and edited in real-time by users with the appropriate permissions.

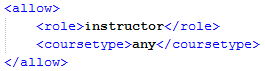
Like all content and settings in Platform-X, the data for the menu system are stored as XML based documents in DLAP. The following sections describe each type of item and how it is used by the menu system.

## Common Metadata

There are certain metadata that are used across the items used in the Menu system. This section describes them in a single place for the sake of efficiency.

### allow

The allow element restricts access to a Menu or MenuItem. The following listing shows an example allow element.



As can be seen in the above listing, the allow element has two child-elements. Role restricts access to users in the role specified, e.g. instructors will make it so that only instructors have access to the Menu or MenuItem. The coursetype child-element restricts access to a specific type of course, e.g. product makes it so that access is only granted on a product course.

The final decision of whether or not to allow access to a Menu or MenuItem is based on the AND of the current user's role and the type of course being accessed. Both the role and the course type must match the value of the allow element in order for access to be granted.

The allow element is not used on its own, but rather as a child element for other metadata. When there are multiple allow elements the effect is an ORing of all of the allow elements. Therefore if the restrictions of any allow element is met, the access will be granted.

### bfw\_display\_flags

The bfw\_display\_flags element contains child allow element which determine whether or not the Menu will be rendered for the user. The allow elements work as previously described and the current user's role as well as the course type being accessed must match the values specified.

### bfw\_edit\_flags

The bfw\_edit\_flags element contains child allow elements which determine whether or not the Menu or MenuItem can be edited by the user. The allow elements work as previously described and the current user's role as well as the course type being accessed must match the values specified.

The Page Layout Engine applies edit flags based on hierarchy. This means that even if a user is allowed to edit a Menu they may not be allowed to edit a specific MenuItem. In order to edit any item, the user must meet the constraints specified by the bfw\_edit\_flags element on the item itself as well as on the items higher in the hierarchy.

### bfw\_menu\_callbacks

The bfw\_menu\_callbacks elements contains "method" child-elements that describe the operations that can be performed by a MenuItem. Every "method" element has a "name" attribute that allows the Page Layout Engine to find the "method" necessary to support a specific action defined by the implicit interface for MenuItems.

Since the Page Layout Engine doesn't know how to invoke the compiled code for the MenuItem it uses several attributes to determine how to invoke the method. The "controller", "action", and "url" attributes are used by the Page Layout Engine to compute a URL to load in order to execute the method. If the destination is within the application then the "controller" and "action" attributes are used to compute the internal URL to the method. For off-site methods the "url" attribute is used. The "type" attribute is used to determine the style of display for the method's result. The value "fne" in the "type" attribute means that a Focused and Engaged pop-up will be used to display the content pointed to by the method. The default value of "navigate" means that the user will be taken to a different page inside or outside of the application.

Method elements can have "parameter" child elements. These elements will be used to construct the query string for the URL computed based on the "controller", "action", and "url" attributes. Some parameter values can be supplied by the Page Layout Engine. For example, when editing the MenuItem the MenuItem's ID is passed automatically.

## Menu

Menus are the core construct in the menu system. A Menu is a Folder that contains MenuItems. Menus provide users with the ability to navigate between pages and content in Platform-X. The following listing shows the XML definition of a typical Menu.



## MenuItem

MenuItem's are XML documents that describe a specific link in the Menu. A MenuItem's primary role is to tell the system where to take the user if that MenuItem is selected. Like Menus, MenuItems make use of the bfw\_display\_flags and bfw\_edit\_flags metadata elements to control access to the item.

The following listing shows an example XML document that represents a MenuItem.



The following sections describe any metadata elements that are specific to the MenuItem document type.

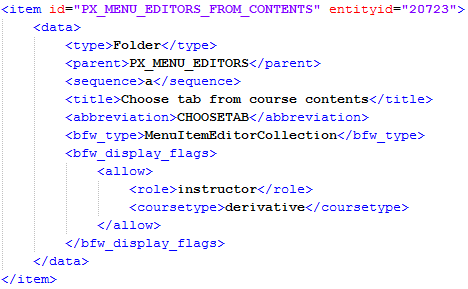
### Methods

#### Navigate

The navigate method is part of the implicit interface defined for MenuItems. The navigate method is responsible for rendering the content the user expects to see when the MenuItem is selected. There are no standard parameters defined for the navigate method.

## MenuEditorCollection

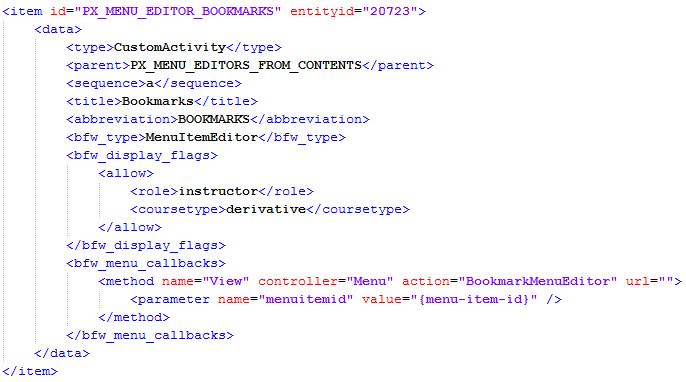
MenuEditorCollections are XML documents used to group MenuEditors. This grouping is used to drive specific sections of the UI when creating or editing MenuItems. In general the MenuEditorCollection is simply a folder that contains MenuEditor items, but it also uses the bfw\_display\_flags elements to control access.



### MenuEditor

MenuEditors are XML documents used to store metadata about a specific user-interface that can be used to create MenuItems. This allows the Page Layout Engine to support a variety of MenuItems, each with a custom UI that is optimized for creating and editing it.

The following listing shows an example MenuEditor XML document.



### Methods

#### View

The view method is invoked when the Page Layout Engine wants to allow the user to create or edit a MenuItem. When the particular type of item the user is interested in is selected, the appropriate editor is displayed. The view method is automatically passed a "menuitemid" parameter when it is editing as opposed to creating a new MenuItem.

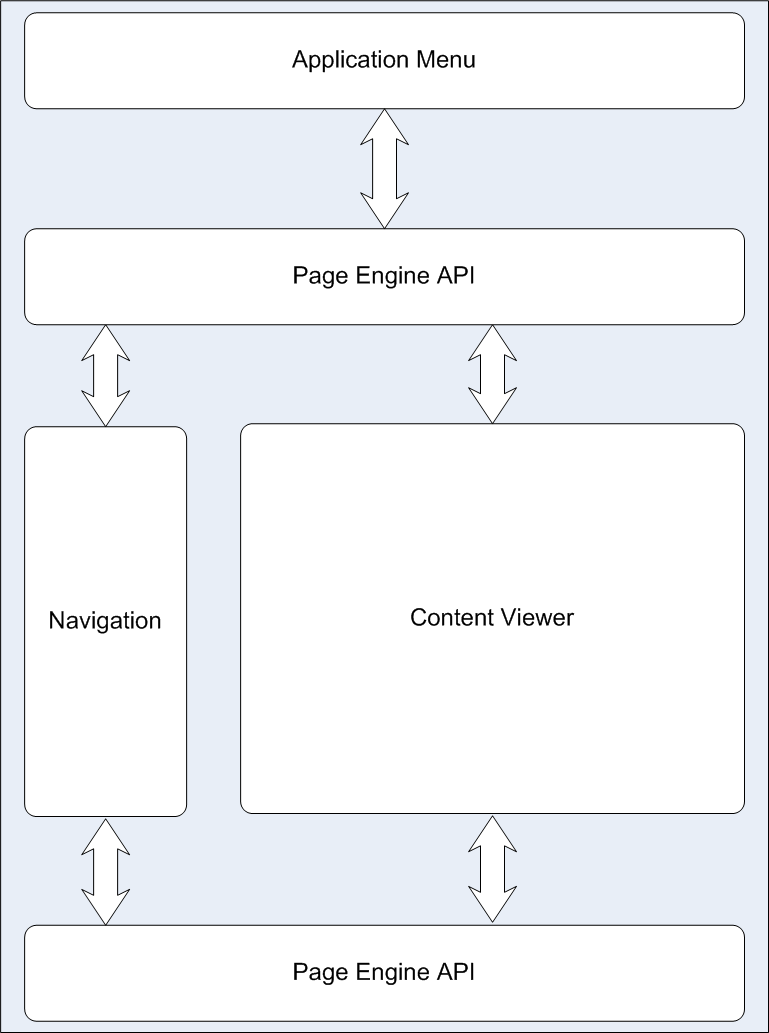
# Client API

The Client API provides facilities so that components can communicate with each other and the Platform-X Shell itself. The general method of communication is through events, but components that are aware of each other may also communicate directly. Using the client API scenarios such as having a dedicated navigation component tell a dedicated player component to render a content item is made possible.

The Client API also exposes user session data to components so that components always know which user is logged in and what they are authorized to do. Having this information available at the client level reduces the burden of development for component authors and ensures that information is accessed in a uniform way.

In order to support the data requirements of client-side driven components the Client API also provides wrappers around common data access functionality such as loading and saving content, assigning items, and general search functionality.

The following image illustrates the role of the Client API on a page hosted in the Platform-X Shell:



As can be seen in the image above, the Components in the hypothetical page do not communicate directly. Any communication between the Components, the Page, or the Shell is done through the Client API.

## The PxClientAPI object

The Client API ensures that a javascript object called PxClientAPI is resident on all pages prior to any component being initialized. The PxClientAPI is the mechanism through which all Client API functionality is accessed and any inter-component communication takes place.

The following sections define some of the core functionality of the PxClientAPI object.

### Event System

Events are a primary communication mechanism between components. The API exposed through the PxClientAPI object is based on the one used by jQuery.

#### Event Binding

When events happen, it is a common use case for some other piece of code to execute. In order for this to work, the code that should execute when the event happens needs to be bound to the event. Users of the PxClientAPI can bind handlers to events in the following way:

PxClientAPI.bind(“event-name”, *event-handler*)

In the above code, *event-handler* is a callback function that will be executed each time the event occurs. The name of the event the handler is being bound is a string that uniquely identifies the event. There is a listing of standard events later in this document.

Sometimes, the link between an event and handler must be broken to prevent the handler from executing. To do this the handler must be unbound from the event as shown below:

PxClient.unbind(“event-name”, [*event-handler]*)

In the above code, *event-handler* is an optional parameter that can be the same function that was passed into the bind function. This allows for either one specific handler to be unbound, or all handlers currently attached to the event.

#### Event Triggering

In order for an event to happen, it must be triggered through the PxClientAPI. The following code demonstrates how this works:

PxClientAPI.trigger(“event-name”, [event-args])

In the above code, the event-name specifies which event is being triggered. Since this can be an arbitrary string, the trigger function is able to support custom component events as well as the standard events provided by the shell. The event-args array will be passed to all handlers bound to the event at the time the event is triggered. Each element of the event-args array is passed to the corresponding handler’s parameter list. For example, if the following handler is bound to an event:

var handler = function(first, second, third) { … }

PxClientAPI.bind(“some-event”, handler)

Then the following call should be used to trigger the event:

PxClientAPI.trigger(“some-event”, [first, second, third])

#### Use of “this” in event handlers

A description of the *this* keyword in javascript is outside the scope of this document, however it is necessary to understand that in all event handlers then *this* keyword is ALWAYS going to be a reference to the PxPageAPI object.

For example, the following code would work as expected:

var handler = function(a, b) {

this.trigger(“some-event”, b);

}

PxClientAPI.bind(“event-one”, handler)