IBM Software Solutions | Enterprise Content Management Software

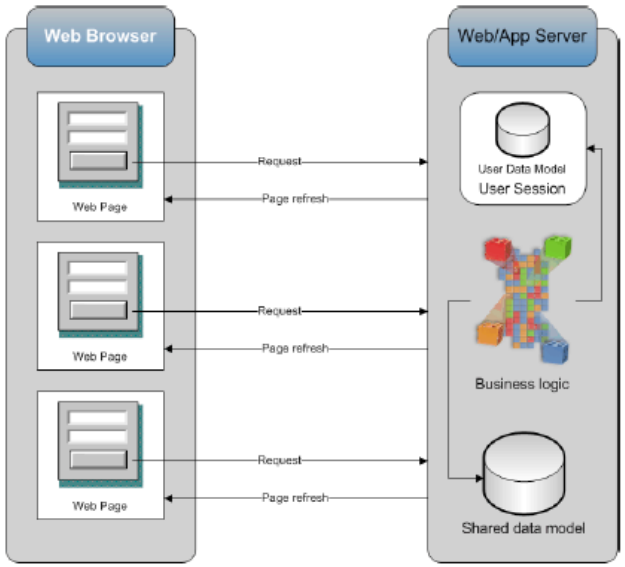
Custom widget development using plugins

IBM Enterprise Content Management Software

Introduction to Dojo Widget Development

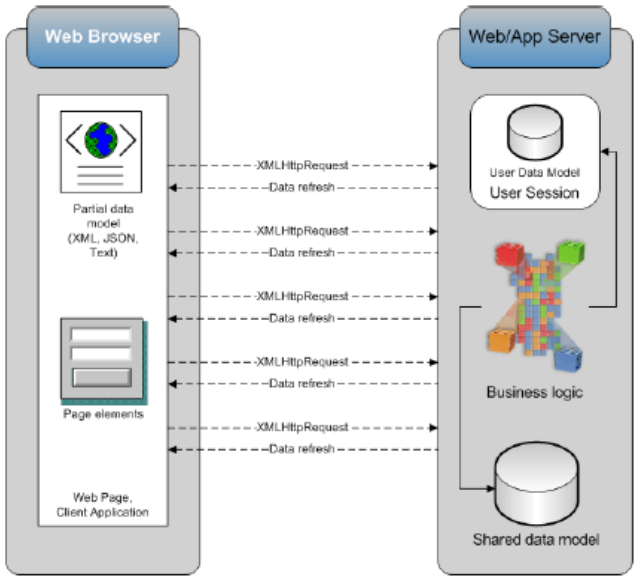
The Web 2.0 user interface evolution

Traditional web interaction



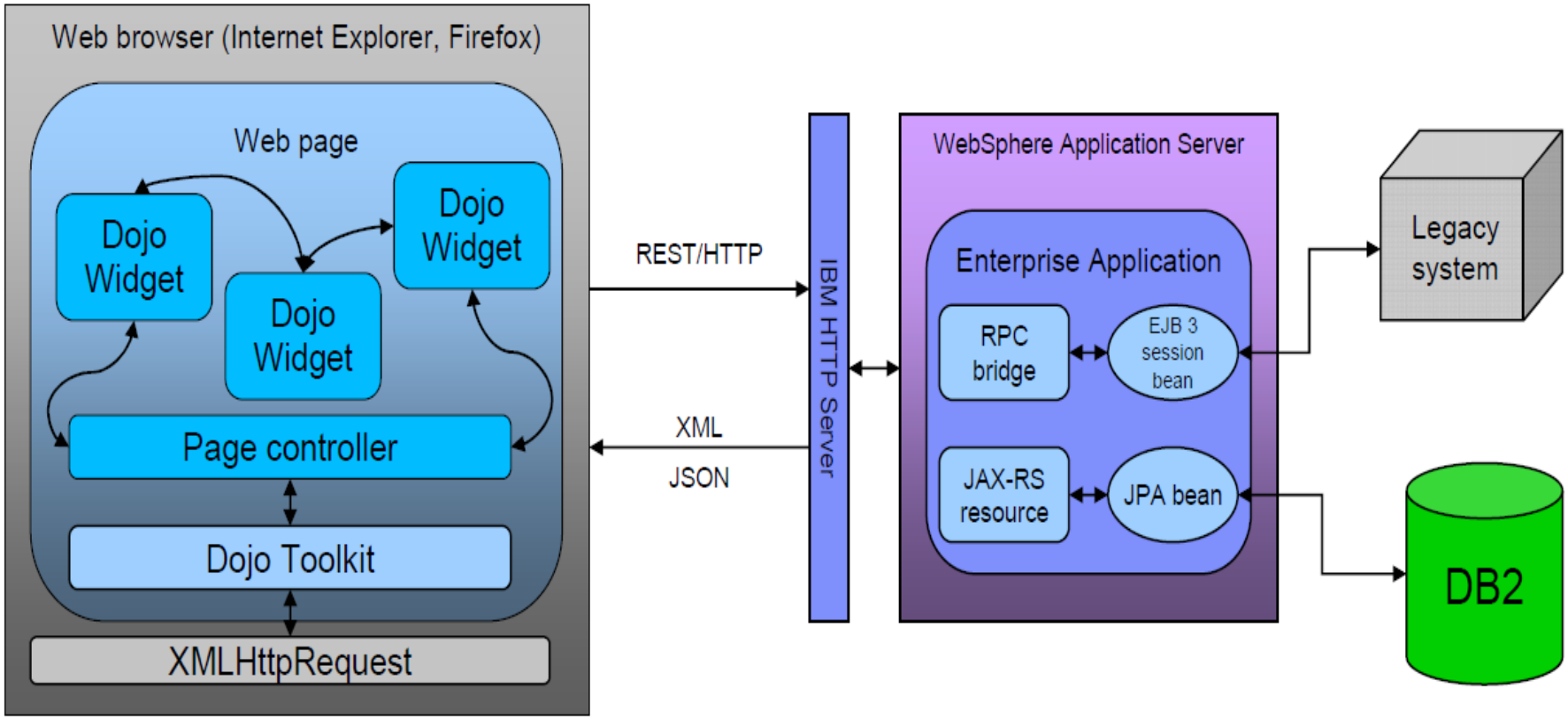
- "Round tripping" provision of the user interface – client/server interaction obvious to the user.
- User interface flow logic managed on the server.
- Application server primarily serves HTML pages.
- Reuse only within the application server infrastructure.

Web 2.0 AJAX interaction



- Smooth user experience – client/server interactions achieved without a full page refresh.
- User interface flow logic managed on the client.
- Application server primarily serves data.
- Reuse through modular AJAX toolkits on the client and RESTful services.

An example of a modern JEE architecture for Web 2.0

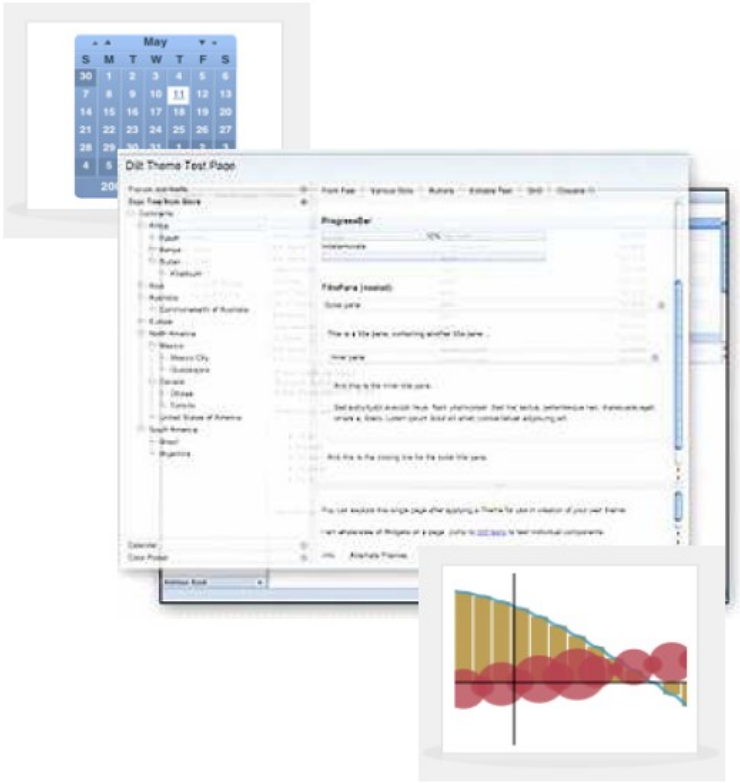


What are AJAX frameworks and why do we need them?

- There are a number of challenges posed by the browser environment including:
  - Differences between browser brands
  - Lack of a consistent component/packaging model
  - Limitations of the standard HTML widgets
  - Extensions to UI behavior always require scripting
  - Lack of a natural separation of concerns between presentation and data
- AJAX frameworks raise the level of abstraction above the base HTML/JavaScript runtime
- There are on the order of hundreds of frameworks currently available, by and large through open source

Introducing the Dojo toolkit

- The Dojo Toolkit is IBM's choice as best of breed AJAX framework
  - Open source, freely available from http://www.dojotoolkit.org
  - Built on open web standards such as HTML, JavaScript, and CSS
  - Seeks to insulate the developer from browser differences and quirks, and to promote modular, open web UIs
- Key features include:
  - Extensible and flexible component model
  - Allows declarative UI extensions
  - Rich user interface components and themes
  - Support for accessibility and internationalization



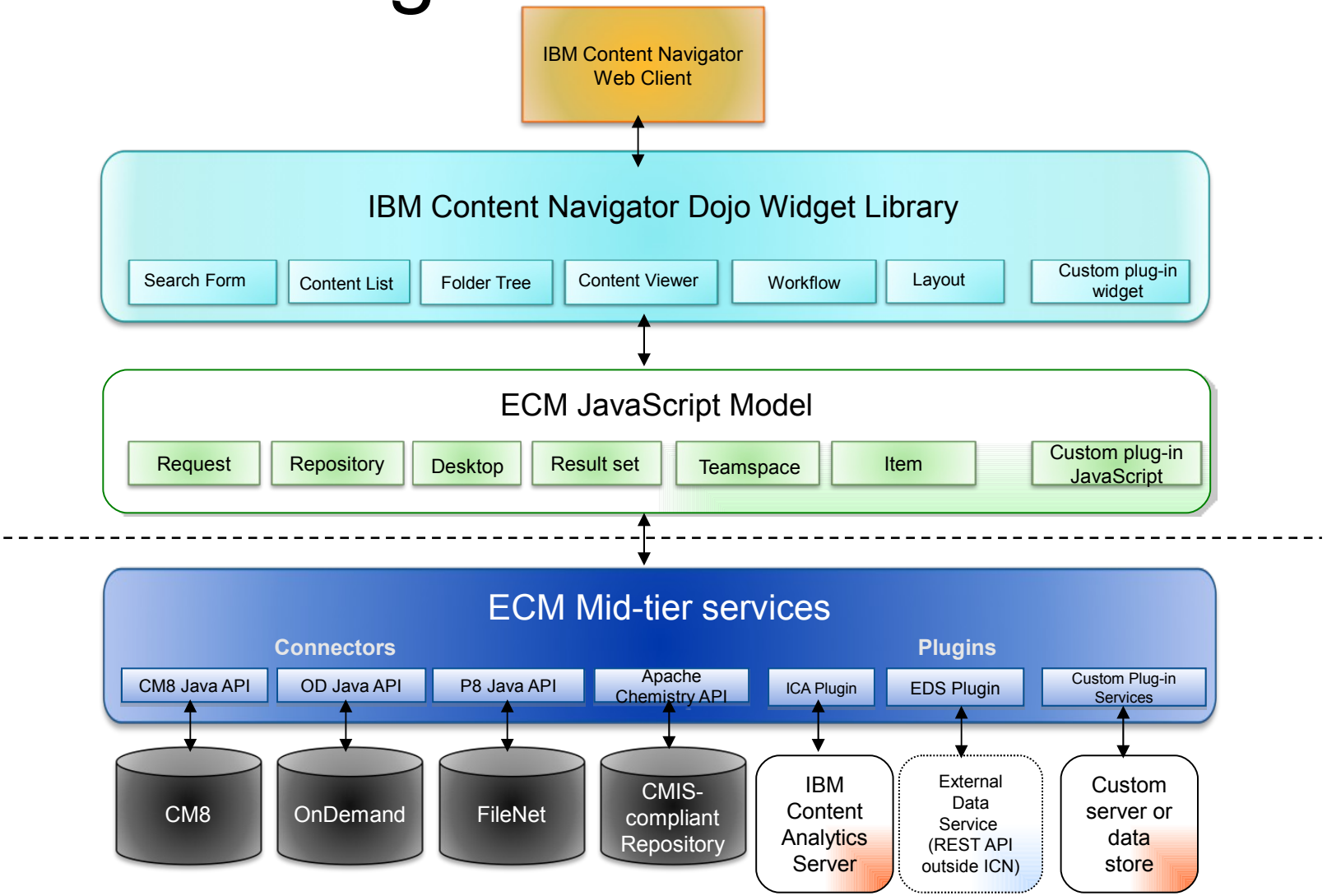
Modules and loading

Introduction to AMD and modules

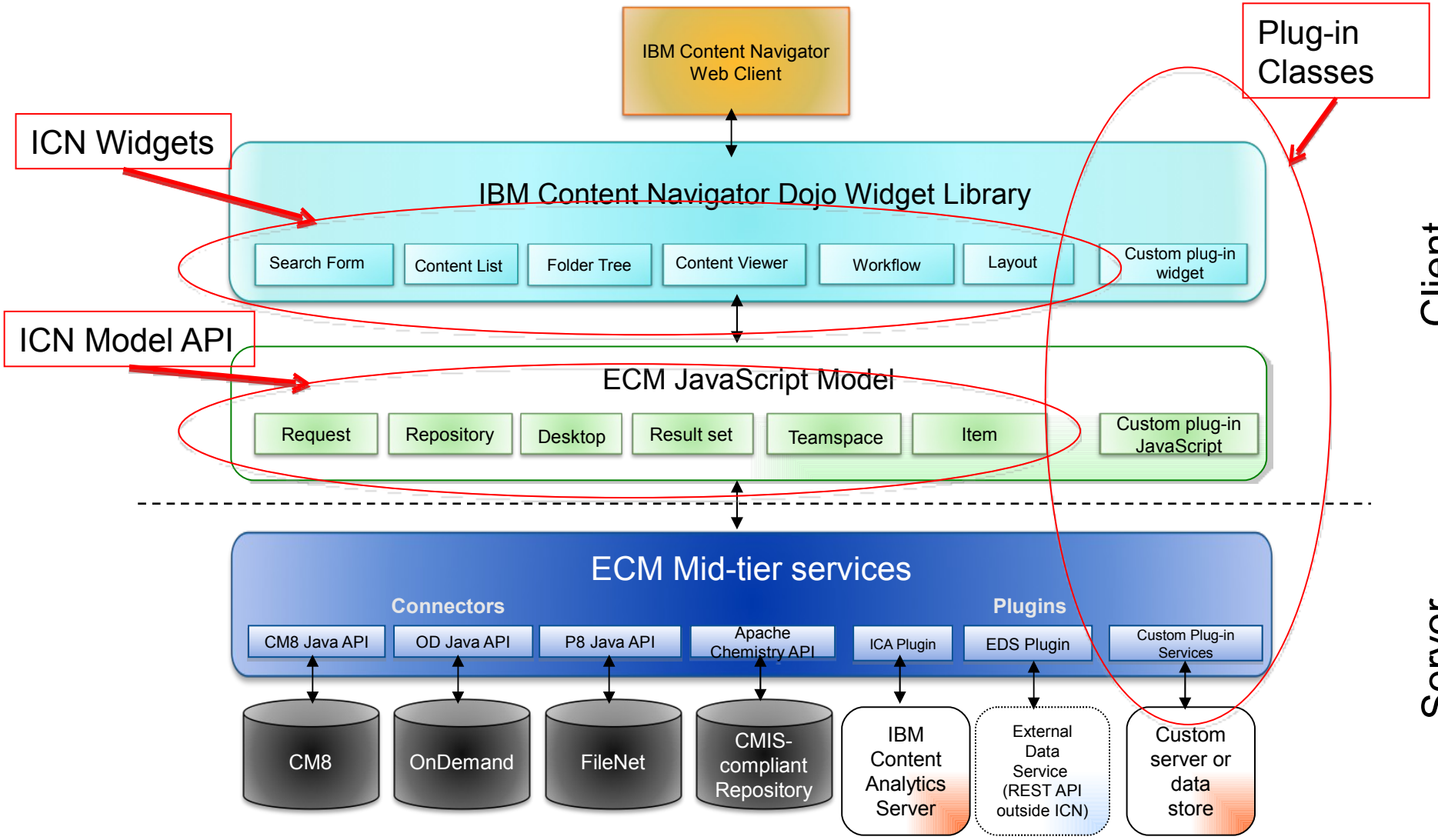
- Starting with Dojo 1.7, module loading is handled by the Asynchronous Module Definition (AMD) system.
  - Avoids polluting the global namespace
  - Better security
  - Loads modules only when and if needed
  - Each module is loaded only once
- Although you'll see classes referred to as dojo.*xxx* or dojo/*xxx,* this is just for specificity. As you'll see, your code will normally end up referencing these modules through local variables.

Overview: developing with the ICN Toolkit

Content Navigator Architecture



Architecture and APIs



Plug-in and Java APIs: Overview

- What is a Content Navigator plugin?

For Content Navigator, a plugin is a package of customizations that can be easily added and configured for a deployment.

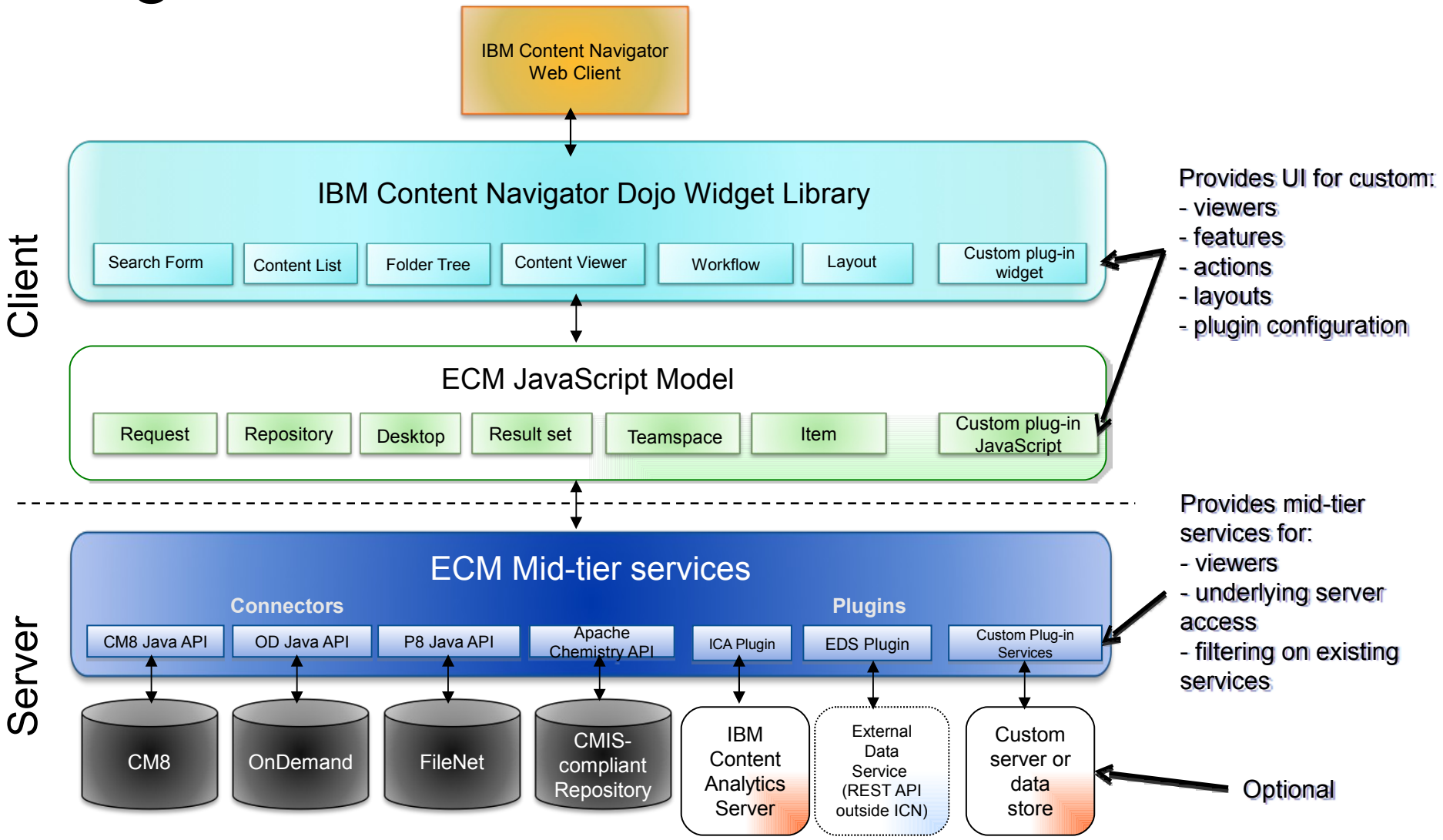
- What can plug-ins be used for?

  - Extending the UI
  - Extending the mid-tier
  - Overriding existing features
  - Extending existing features
  - Creating new features

Plug-ins: Extension points

- Actions: A function on a toolbar or menu.
- Menus: A menu or a toolbar
- Features: A major functional area, selectable as an icon on the left side of the desktop.
- Viewers: A viewer/editor for a document type. Appears in the tabbed viewer window, or optionally in its own browser window.
- Services: A mid-tier function, can invoke other APIs of the content server or other servers.
- Filters: Used to modify the requests and responses to the built-in services.
- Layout: A completely different layout (the entire UI).
- Configuration: A plug-in can provide its own interface for configuration. This is displayed in administration, on the Plug-in tab.

Plug-in and Java APIs



Plug-in structure

- The basic plug-in structure:

```
MyPlugin.jar
     com
         mycompany
             myplugin
                 MyPlugin.class
                 other .class files
                 WebContent
                      .js, .html, and image files
     META-INF
         MANIFEST.MF
```

- Inside the MANIFEST.MF:

```
Plugin-Class: com.mycompany.myplugin.MyPlugin
```

Java API

- What is the Java API? Provides two capabilities:
  1. classes to extend and define plug-ins
  2. classes to access and modify configuration information
- What are the uses of the Java API?

  The Java API is primarily meant for use within the plugin. However, the configuration classes can be used stand-alone to create utilities that manipulate the configuration database.

- Available packages:
  - `com.ibm.ecm.extension`
  - `com.ibm.ecm.configuration`

JavaScript APIs: Overview

- What are the JavaScript APIs?

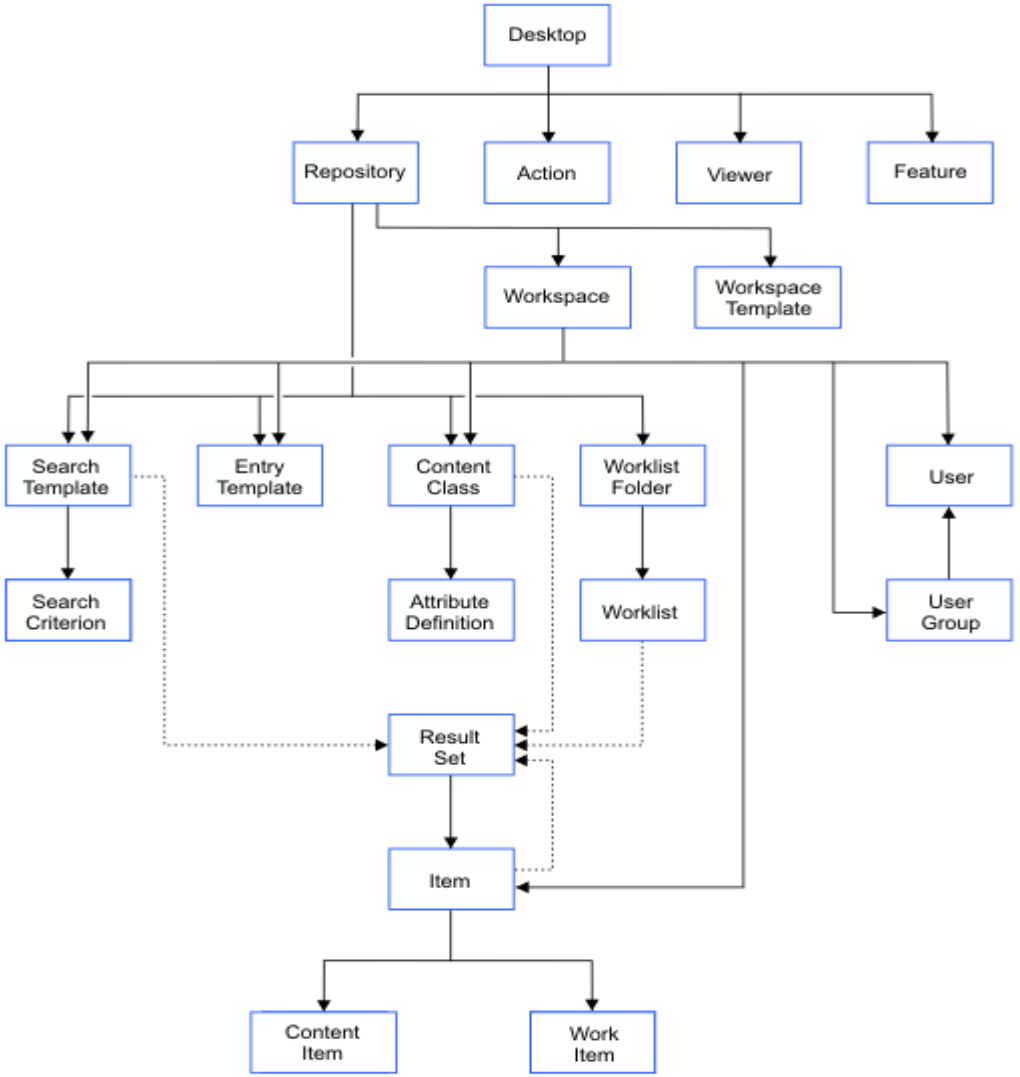
  The JavaScript APIs are a set of both visual and non-visual Dojo classes.

- How are they used?

  Two ways:

  1 – developing custom clients that embed portions of Content Navigator.
  2 – developing Plug-ins to Content Navigator

JavaScript APIs: Model Classes



Desktop	Root object for all information displayed on the desktop.
Repository	A single content repository defined in the desktop.
Action, Viewer, Feature	Describe menu actions, viewers and features as configured for the desktop.
Workspace, WorkspaceTemplate	Represents a teamspace and teamspace template.
SearchTemplate, SearchCriterion	Represents a (saved) search.
EntryTemplate	Represents an entry template.
ContentClass, AttributeDefinition	Represents classes and item types.
WorklistFolder, Worklist	Represents worklists and their various collections
User, UserGroup	Represents users and groups of users
ResultSet	A set of results from searching, folder contents, worklist contents
Item, ContentItem, WorkItem	Various content and process items, including documents, folders, and work items

- *For the comprehensive list, go to the IBM Content Navigator Information Center: http://pic.dhe.ibm.com/infocenter/cmgmt/v8r4m0/index.jsp?topic=/com.ibm.developingeuc.doc/doc/JavaScriptdoc/index.html*

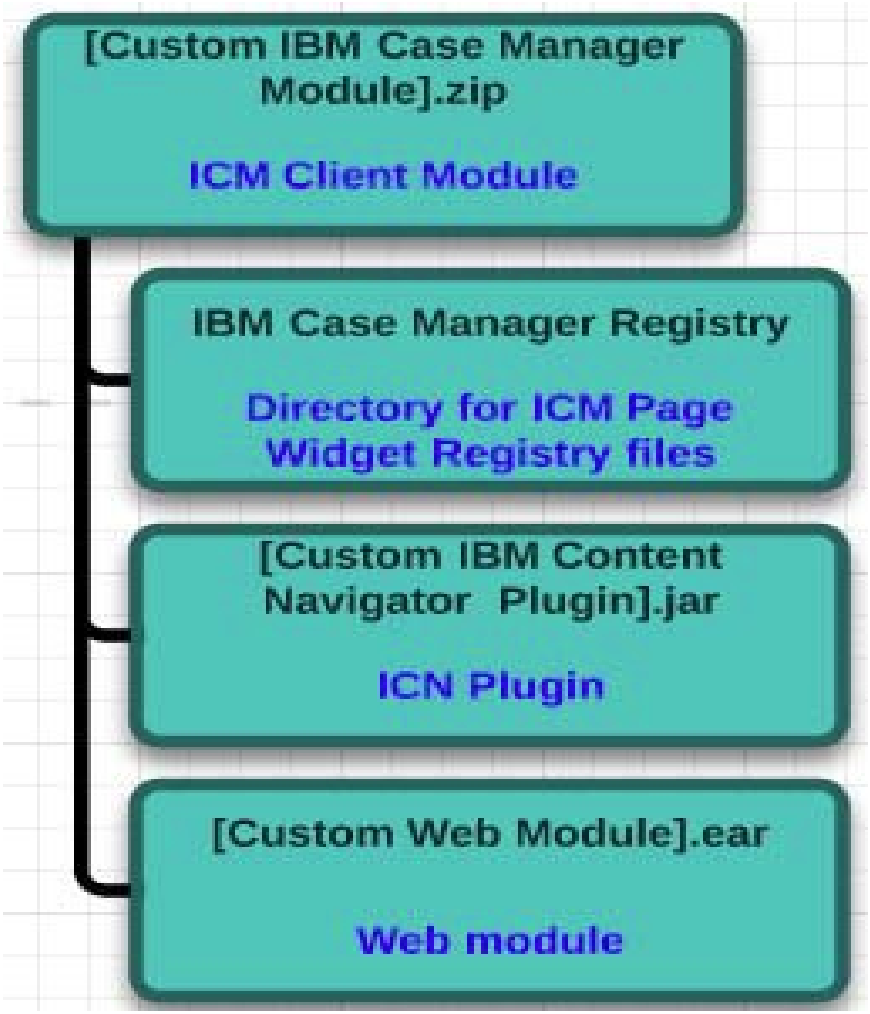
Widget Structure

Objectives

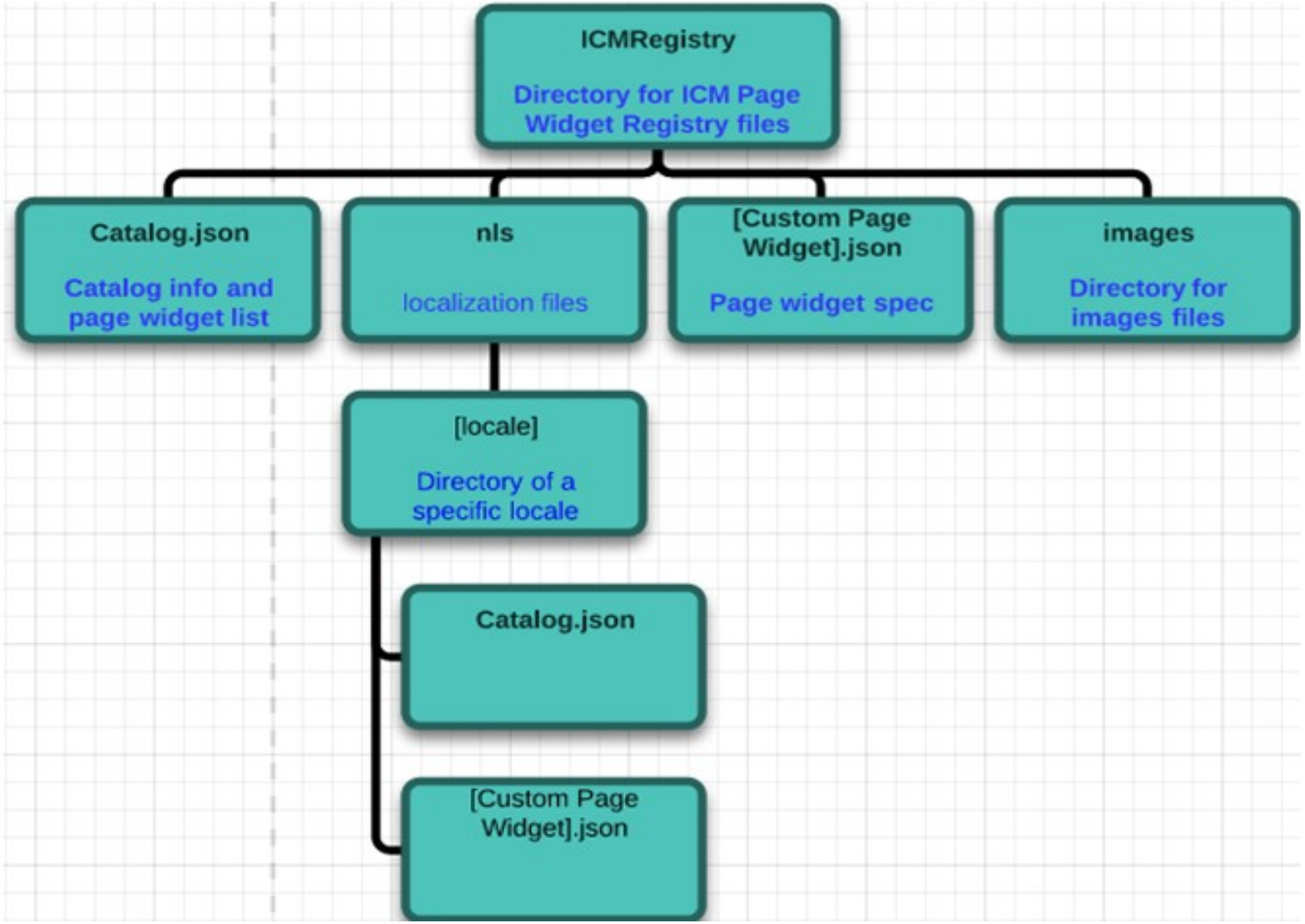
Widgets in ICM 5.2 are called Page Widgets. There are two important structural aspects of widgets that we will cover in this lesson:

- A package of one or more widgets is delivered in a Zip file with a particular structure.
  - Your build environment setup will create this structure automatically.
- The JavaScript code comprising each widget should also follow a particular structure.

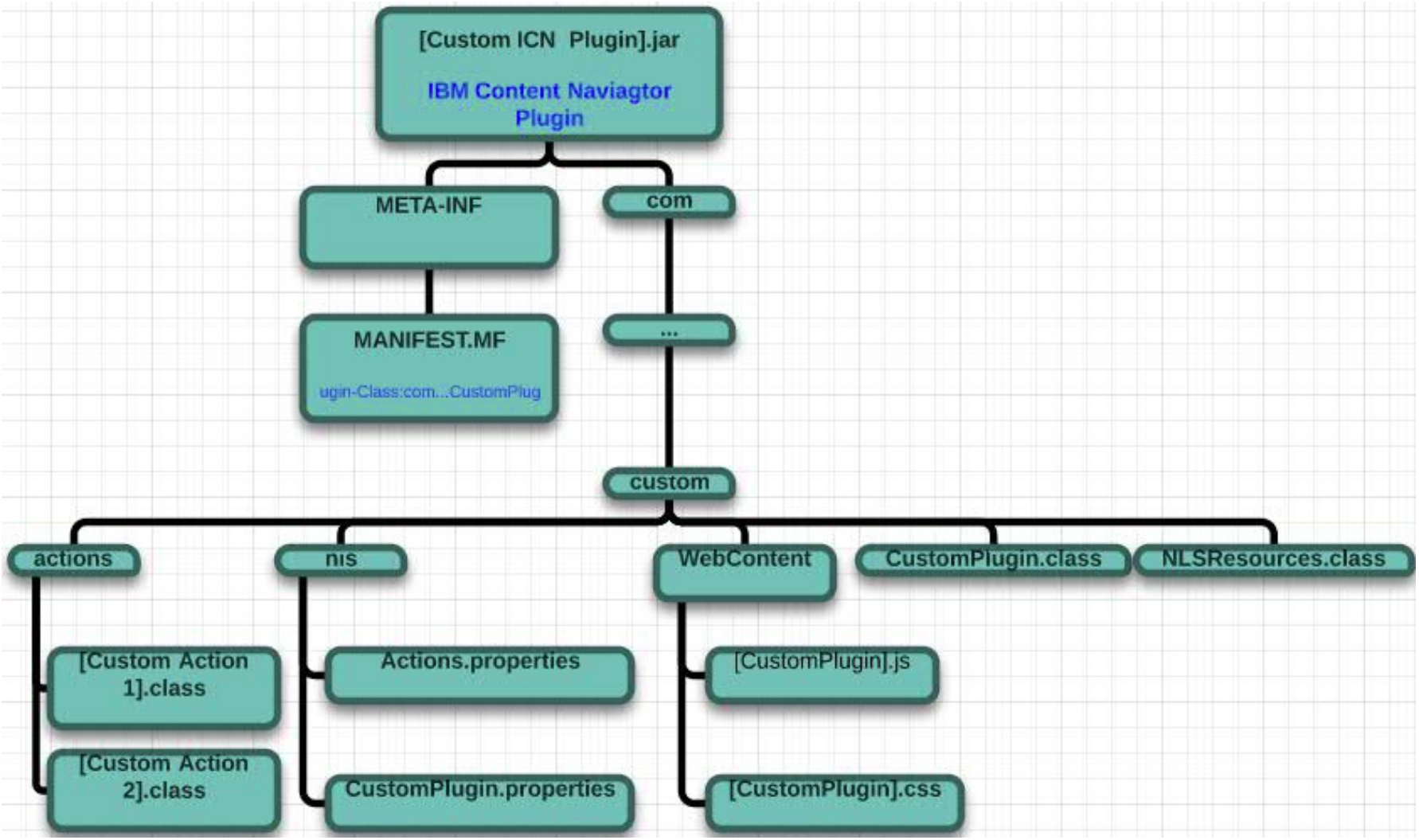
Basic structure of a Page Widget File



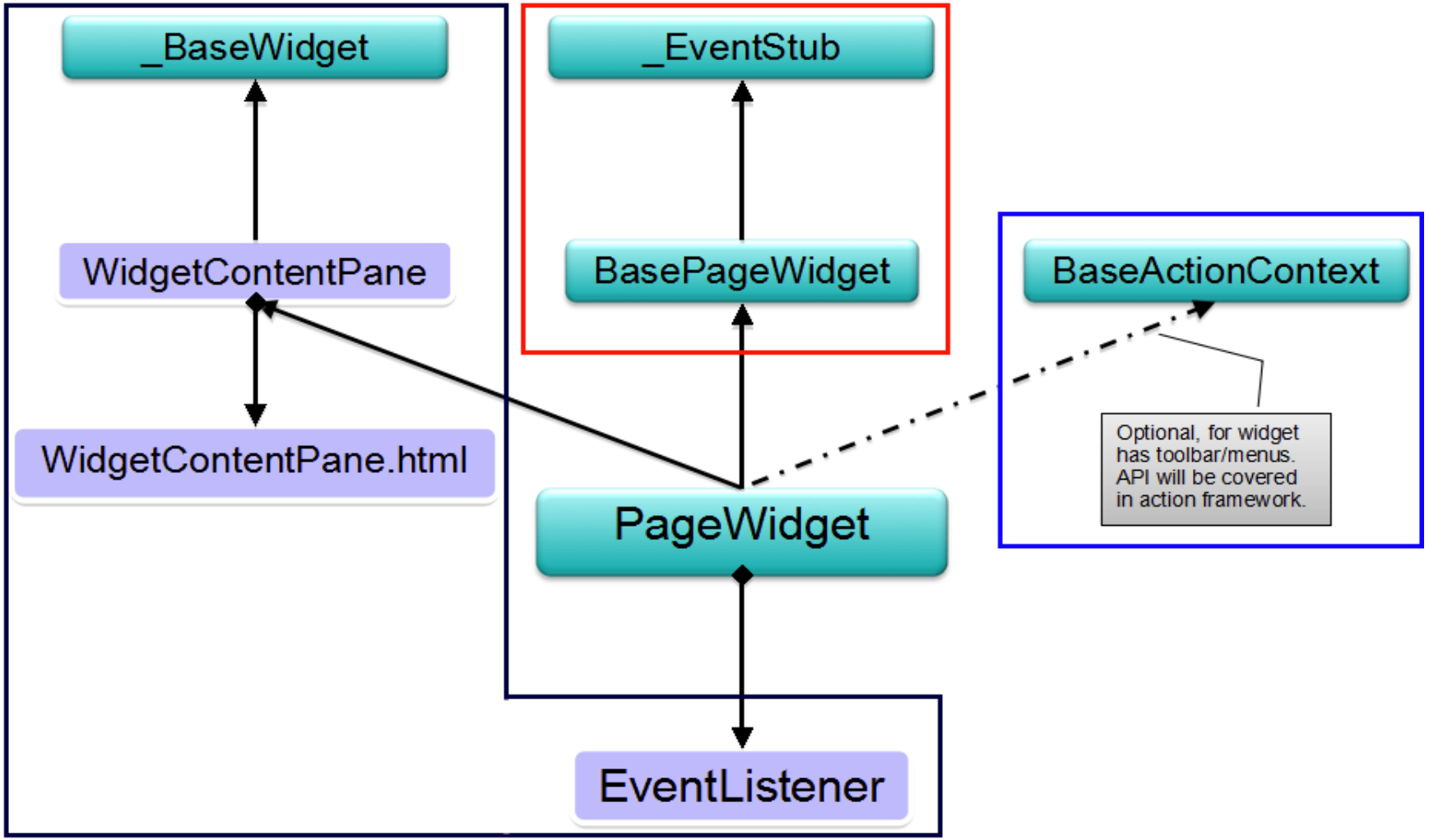
Structure of the ICMRegistry folder



Structure of the ICN plugin jar



Recommended structure for page widget code



Code structure description

- Classes in the black box implement the user interface for the page widget and handle the user's interaction with the interface.
- Classes in the red box are base IBM Case Manager classes from which the custom page widget inherits functionality. The icm.base._BaseWidget class, which provides functions that display the widget description and show or hide the content pane.
- The icm.base._EventStub class provides functions for publishing and broadcasting methods.
- The class in the blue box is the base class for any page widget that hosts a toolbar or a pop-up menu.

Completed Objectives

This lesson covered:

- The structure of the Zip file that will contain one or more widgets.
- The structure that your JavaScript code should follow.

Creating catalog and description files

Objectives

This lesson is designed to introduce you to the catalog and description JSON files:

- Review the 5.1.1 catalog XML file
- Basic format of JSON files in 5.2
- Converting from XML to JSON

.

Catalog and description files

- In ICM 5.2, there are two files that fulfill the same purposes, in JSON format

Catalog JSON file fields

In ICM 5.2, the Catalog file has the following attributes:

- Name – Unique name of the custom widget package
- Description – Description of the custom widget package
- Locale – Optional locale for the catalog, or the default of empty string
- Version – Optional version of the custom widget package
- Categories – List of objects containing ID and category attributes, if you do not want to reuse the OOTB “CaseWidgets” and “GenericWidgets” categories.
- Widgets – List of objects containing the attributes detailed on the following slide

Widget object fields

Each widget in the list will be an object with the following attributes:

- id – unique name of this particular widget
- title – longer name for the widget
- category – one of the IDs listed in the Categories list or the OOTB choices
- description – description of what this widget does
- definition – name of the widget's definition JSON file
- preview – preview image path
- icon – icon image path
- runtimeClassName – full name of the class that implements this widget
- previewThumbnail – preview thumbnail's image path

Constructing the widget definition JSON file

Although there was a file called definition.xml in the custom widget ear file in ICM 5.1.x, its contents is not relevant to creating the ICM 5.2 widget definition file (*widgetname*.json), which will contain the following information:

- All of the same attributes and values from the Widget element in the catalog JSON file for this widget, including a self reference attribute "definition".
- In addition, there are two lists, defined in the following slides:
  - properties
  - events

Properties objects

The properties list is zero or more properties for the widget. Each property is an object with the following attributes:

- "propertyType":"property" // see next slide
- "type":"string" // or "integer", "float", "boolean", etc.
- "id":"*unique ID for this property*"
- "defaultValue":"" // or a value for the property
- "required":true // or false
- "visibility":true // or false
- "title":"*title for this property*"

The property values for each instance of this widget can be set in Case Builder based on this information.

Example properties list

```
"properties":[
        {
            "propertyType":"property",
            "type":"integer",
            "id":"pageSize",
            "defaultValue":15,
            "required":false,
            "visibility":true,
            "title":"Page Size"
        },
        {
            "propertyType":"property",
            "type":"role",
            "id":"roleName",
            "defaultValue":"",
            "required":false,
            "visibility":true,
            "title":"Select Role"
        },
]
```

The propertyType value “group”

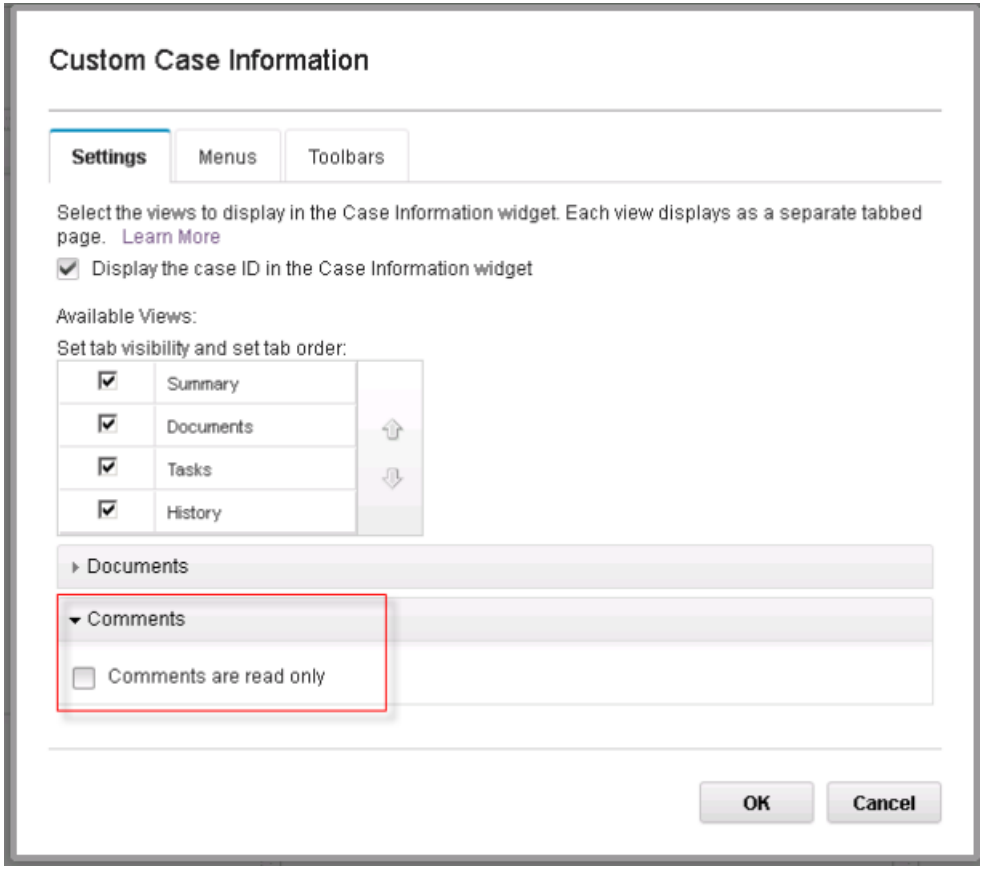
If a widget contains a toolbar, its definition JSON file will contain a property with propertyType set to `group` and otherwise `property`. This defines the Toolbar or Menu tab. This type can also be used to group a collection of settings together in Case Builder.

The property members under this group represent the toolbar(s) or menu(s) for this widget, with context, or properties in this group (“propertyMembers”).

For groups, the “type” attribute can be one of “tab”, “section”, “dropdown”, or “propertyPanel”.

See example on the following slides.

propertyType “group” example



Example of propertyType “group”

```
{
    "propertyType": "group",
    "type": "section",
    "id": "commentsGeneralSettings",
    "title": "Comments",
    "propertyMembers": [
        {
            "propertyType": "property",
            "type": "boolean",
            "id": "commentsAreReadOnly",
            "defaultValue": false,
            "required": true,
            "visibility": true,
            "title": "Comments are read only"
        }
    ]
},
```

Event objects

The events list is zero or more objects that lists the events this widget publishes or handles. Entries have the following attributes:

- direction – “subscribed” or “published”
- type – “wiring” or “broadcast” (if outgoing/published)
- id – Unique ID of the event. Use “icm.CustomEvent” to handle any type of event.
- title – Title of the event or event handler
- functionName – If incoming/subscribed, then the name of the function that will handle the event

Example events list

```
"events":[
        {
            "id":"icm.RoleChanged",
            "title":"Role selected",
            "functionName":"handleReceiveRole",
            "direction":"subscribed",
            "description":"Update the In-baskets widget to
display the in-baskets that are associated with the
specified role."
        },
        {
            "id":"icm.SelectRow",
            "title":"Row selected",
            "direction":"published",
            "type":"wiring",
            "description":"The user clicked a row or
pressed enter in the in-basket to select the work item."
        },
]
```

Completed Objectives

This lesson was designed to:

- Basic format of JSON files in 5.2
- Converting from XML to JSON