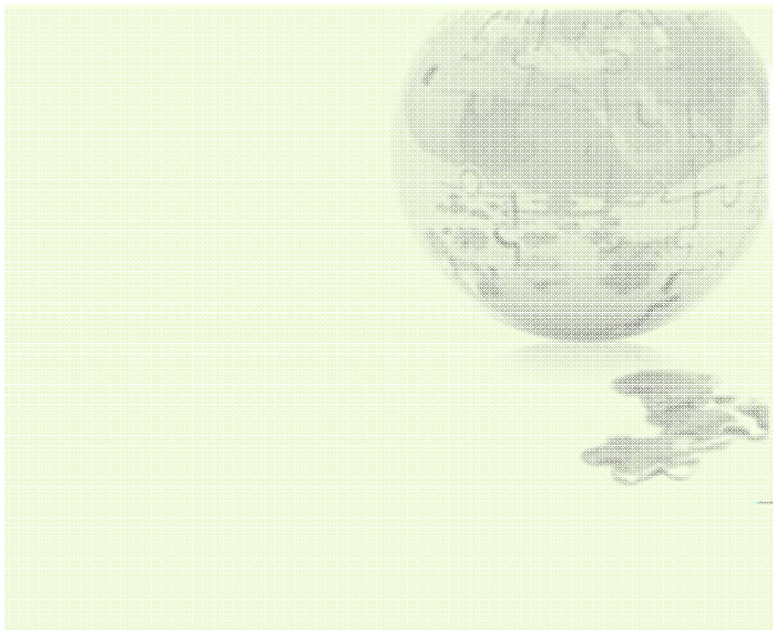
IBM Software Solutions | Enterprise Content Management Software

Case Manager Model and REST API

IBM Enterprise Content Management Software



Rest and Model APIs

Objectives

This lesson is designed to introduce you to the Model APIs and highlight the differences from the older REST APIs, including:

- Review of the basics of REST APIs
- Intro to the Model Layer API paradigm
- REST call to Model Layer API call mapping.

REST API calls

- Existing widgets
  - May make REST API calls to the CMIS, PE, or Case REST services.
  - They may also call custom REST services.
- New Content Navigator approach:
  - Defines a Model API layer that abstracts the underlying content and process servers.
  - ICM 5.2 has extended this concept, and in some cases extended the actual Navigator classes, to support Case operations.

REST call basics

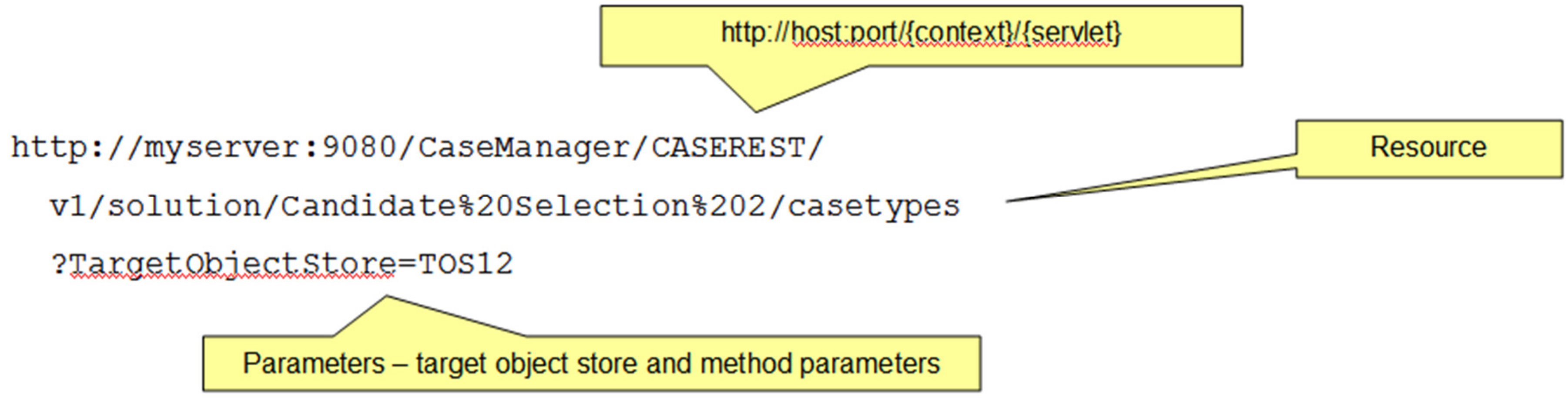
- A REST call makes one of the four HTTP functional calls (GET, PUT, POST, or DELETE) to a given URL.
- Part of the URL may contain effective arguments
- Arguments may also occur encoded at the end of the URL, after a question mark (?), separated by ampersands (&).
- JavaScript most often performs REST calls asynchronously---blocking would leave the UI non-responsive. In JavaScript, this is called AJAX---Asynchronous JavaScript And XML, although the response may be any format, JSON is often used.
- Calls are usually performed using a JavaScript library, such as dojo/request.get() or (older) dojo.xhrGet().

Example of Case REST

- URI format:

```
http://host:port/{context}/CASEREST/v1/{resourceName}[?TargetObjectStore={obj
  ectStoreName}][&{resourceParameters}]
```

- JSON returned in response
- Example - get list of case types in a solution



```
http://myserver:9080/CaseManager/CASEREST/
  v1/solution/Candidate%20Selection%202/casetypes
  ?TargetObjectStore=TOS12
```

http://host:port/{context}/{servlet}
Resource
Parameters – target object store and method parameters

Model API basics

- In the Model APIs, the user makes what appears to be a normal function call.
- One of the arguments will be a callback function that is invoked when the response is received from the server.
- Other arguments will be specific to the function being invoked.
- With few exceptions, the call will operate asynchronously, when it completes invoking a callback function that the caller specifies.
- As with REST calls, asynchronous operation is the norm since it doesn't block operation of the browser/UI while waiting for the response.

What is the “Model”?

1. The Model Layer abstracts the underlying service
   - In principal, the same API could be used regardless of what content repository, workflow system, etc., underlies the service.
   - The user of the API does not need to construct URLs or worry about possible changes in the URL structure from version to version.
   - The search API is therefore more restricted than free-form CE queries which would be repository-specific---more on this below

What is the “Model”?

2. The Model objects follow a Model View Controller (MVC) paradigm.
   - Widget code may subscribe to Model objects to be notified when they change, so the widget can update its display automatically.
   - This feature can reduce the need for handling wired events.

Model objects

- The Model Layer defines not only the API calls, but the types of the response objects that are returned.
- You normally do not construct these response objects---you get them from making a Model API call. Some factory methods are provided that convert from JSON.
- You will need Model objects for invoking other Model and Case API calls.
- For these reasons, you will most likely want to convert REST calls to Model calls, so that you have the Model response objects available for further API calls.

Search

- In the IBM Case Mananger, searching leverages the Navigator search API
- The Case search API is in the icm.pgwidget.casesearch.CaseSearch class.
- The primary advantage of this API is that the search interface is independent of the underlying repository
- Although Navigator's search interface doesn't allow CE query strings to be used, since it's repository-agnostic, ICM extends the syntax to allow it.

Example of performing a search

```
var solution = this.solution; // solution from the widget.
var params = {};
params.ObjectStore = solution.targetObjectStore.id;
params.ceQuery = "SELECT t.[FolderName], …
 FROM [CmAcmCaseFolder] t
 WHERE t.[CmAcmCaseIdentifier] LIKE '%%'
AND t.[CmAcmCaseState] > 1
 ORDER BY t.[CmAcmCaseIdentifier]";
params.CaseType = "";
params.solution = solution;

var searchPayload = new icm.util.SearchPayload();
searchPayload.setModel(params);

var payload = searchPayload.getSearchPayload();

this.context.onBroadcastEvent("icm.SearchCases", payload);
```

Converting REST to Model calls

- While the custom REST services and their usage will remain the same when converting to 5.2, CMIS, PE, and Case REST calls should usually be converted to equivalent Model API calls.
- The tradeoffs are:
  - All ICM 5.2 out of the box widgets use the Model APIs. This is also recommended for custom widgets.
  - Model API calls will return Model objects, ready to be used in other calls.
  - But, any custom code using the returned objects will need to change to expect the Model object that is returned, such as by calling its methods, instead of a bare JSON object.

Example---getting or adding Case Comments via the REST API

- To get or add comments on a case, using the REST API, the URL will be
  - /CASEREST/v1/case/*caseFolderId*/comments
- Optional arguments (after “?” in the URL) are:
  - Target object store id
  - WorkItem id
  - Comment type (task, case, document, workitem)
  - Workflow number
- Use the GET operation to retrieve comments, and the PUT operation to add a comment

Example---getting or adding Case Comments via the Model API

- In the Model API, the paradigm is instead object-oriented.
- You start with an object representing the item whose comments you want to retrieve or add to. For example, an item of type icm.model.Case would represent a case.
- You then invoke a member function reflecting the operation you want to perform. To add a comment, for instance, you would use
  - addCaseComment(*commentContext, commentText, callbackFunction*)
- Once the operation completes, then *callbackFunction* will be invoked.

Model API “scratchpad” objects

- In the ICM Model API, objects returned from making API calls should be treated as immutable (an object whose state cannot be modified after it is created)
- To update an object, you
  - Obtain a “scratchpad” equivalent, whose types contain the word “Editable”
  - Make changes to that object
  - Call save() to persist the changes
- For example, given an object of type icm.model.Case, call its createEditable() method to obtain an icm.model.CaseEditable object. Change it, then call its save() method.
- icm.model.CaseEditable objects have a caseObject field that can be used to get the original object if needed.

Completed Objectives

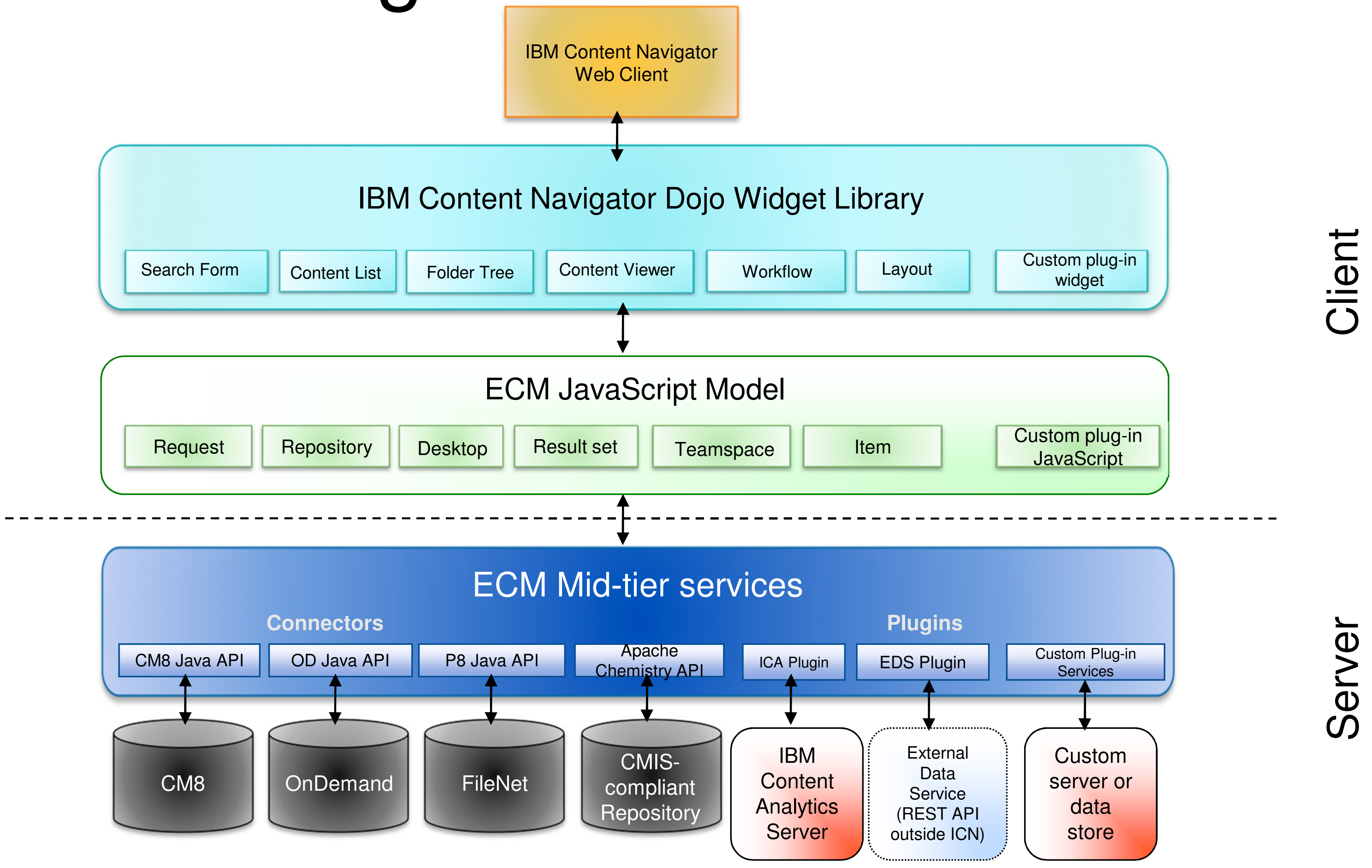
This lesson was designed to enable you to:

- Review of the basics of REST APIs
- Learn the Model Layer API paradigm
- Use Model objects and subscribe to their events
- Map REST calls to Model Layer API calls

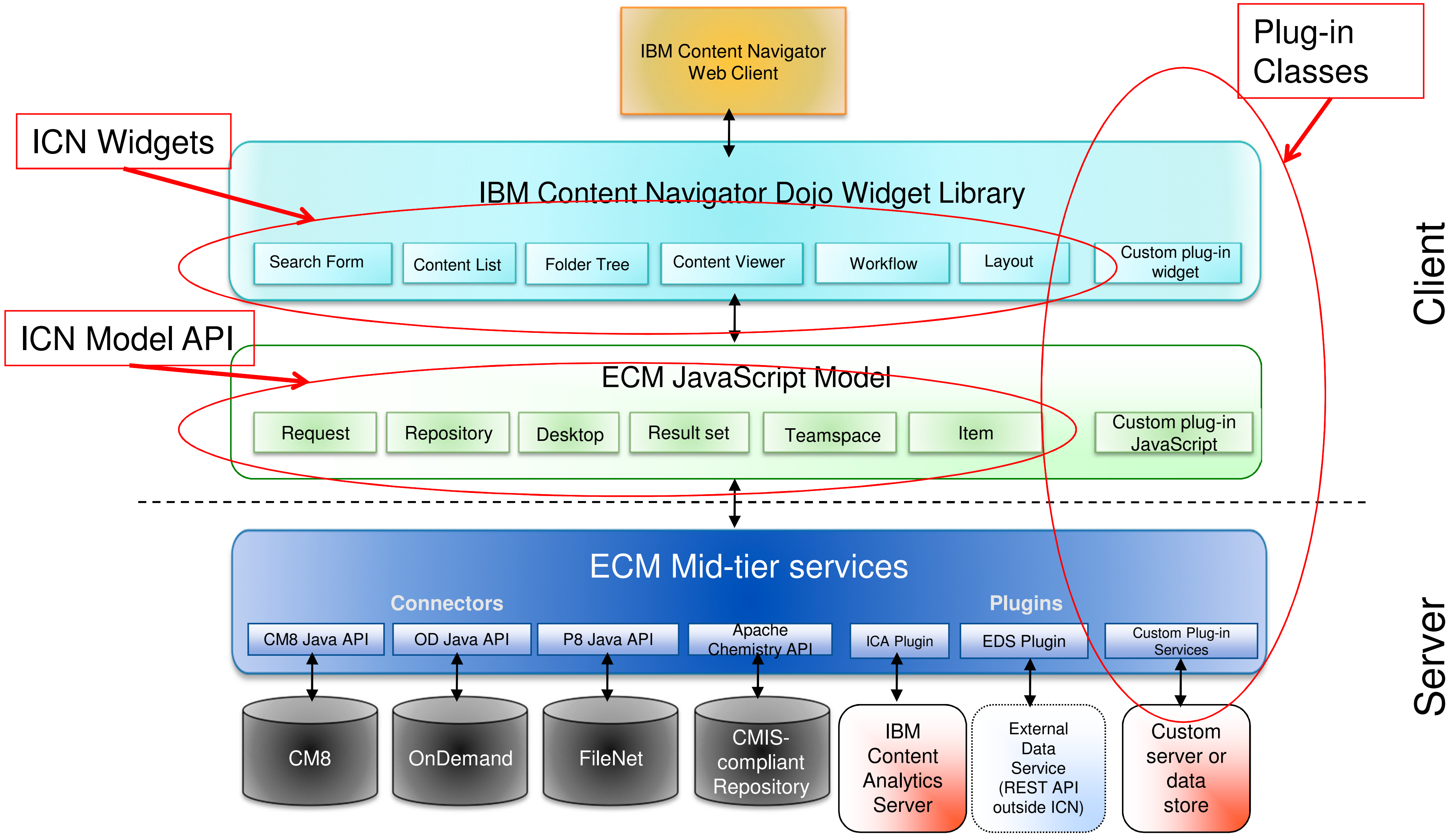
Overview: developing with the ICN Toolkit

IBM Confidential

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.

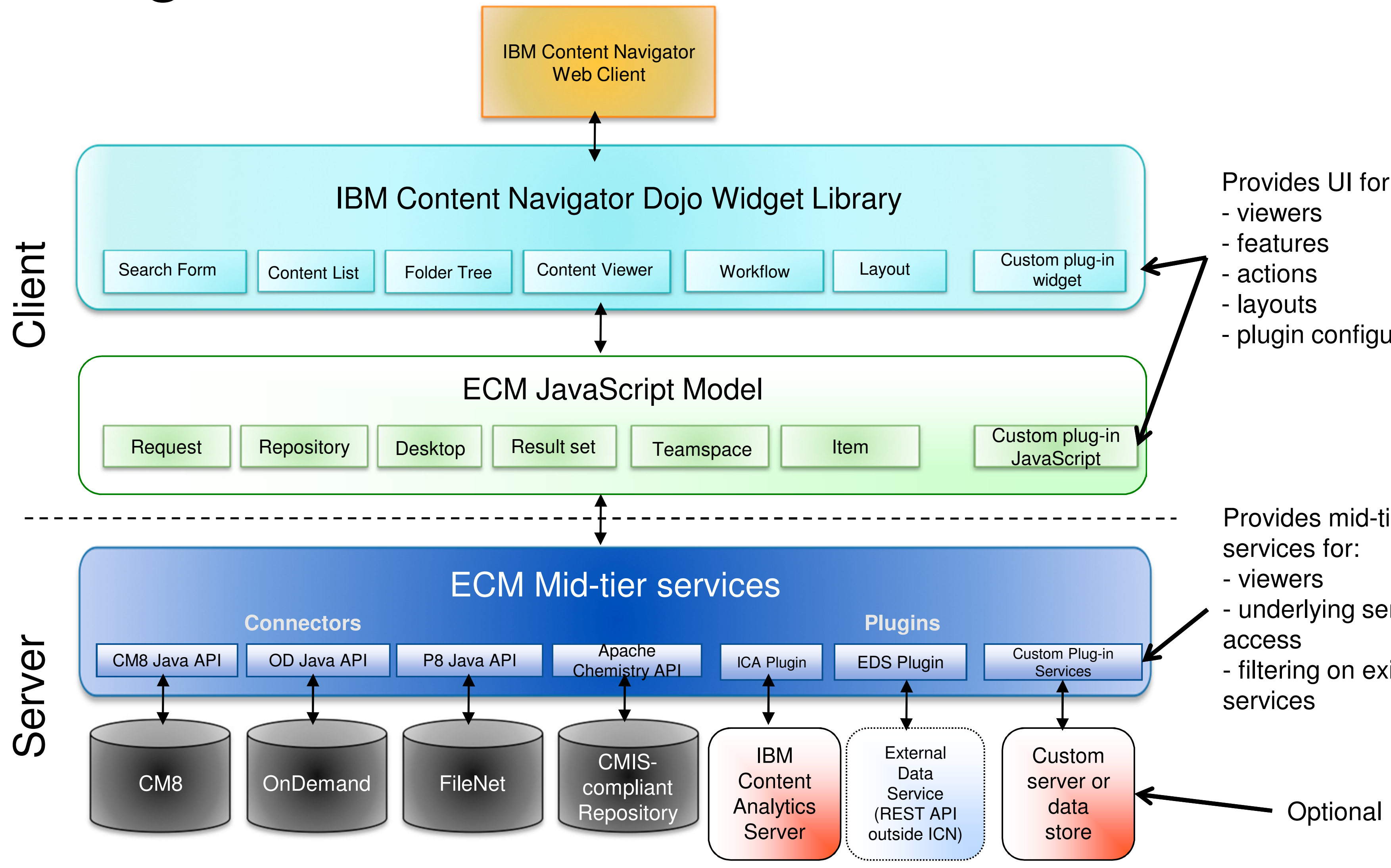
- What can plug-ins be used for?

  - 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



Provides UI for custom:
- viewers
- features
- actions
- layouts
- plugin configuration

Provides mid-tier services for:
- viewers
- underlying server access
- filtering on existing services

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?

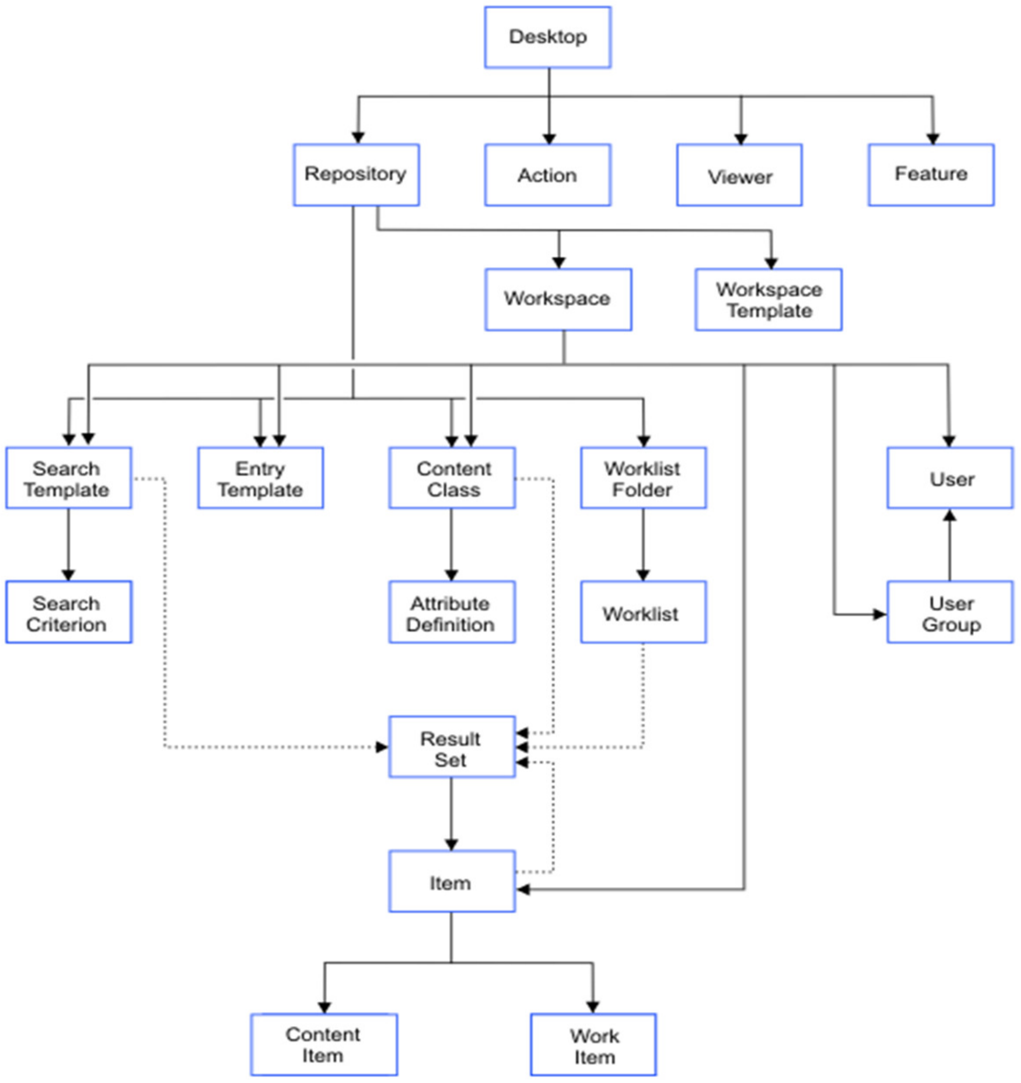
  The JavaScript APIs are a set of both visual and non-visual Dojo classes.

- 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

IBM Confidential

JavaScript APIs: Model Classes

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