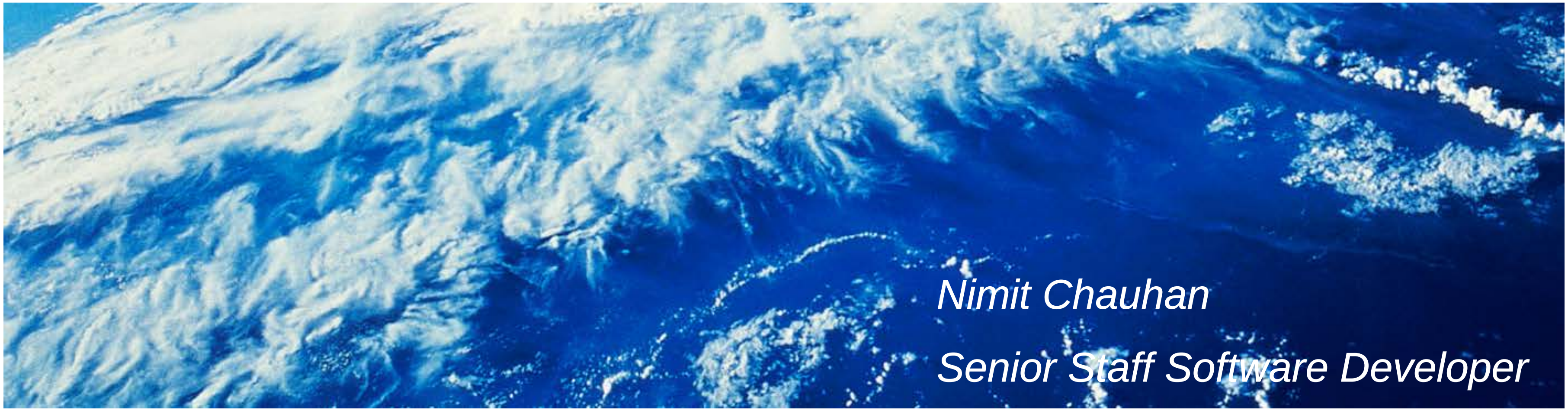
Extensibility in IBM Sterling Store v9.4



Nimit Chauhan

Senior Staff Software Developer

Agenda

- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- Use cases – demo
- Debugging Tips

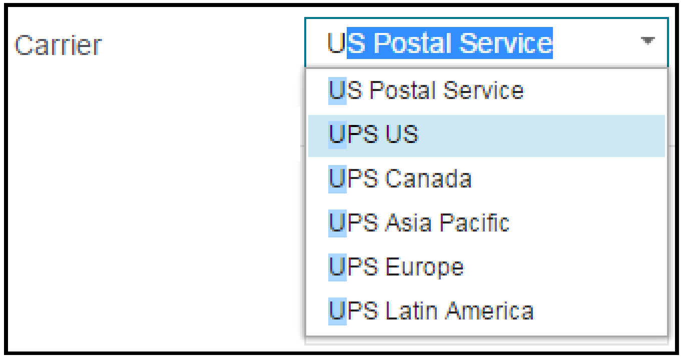
- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- Use cases – demo
- Debugging Tips

Terms

- Widget
- uId – unique identifier
- Screen (View)
- Model
- Binding
- Namespace
- Events and Subscribers
- Mashup
- Mashup Reference (MashupRef)
- Controllers
- Wizard
- Editor

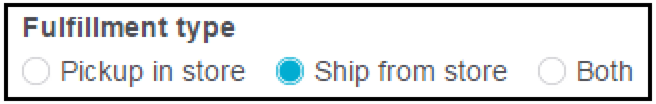
Widget, uId

- **Widget**
  - GUI element that allows the user to interact with application by directly reading or editing information.



Carrier: US Postal Service

- US Postal Service
- UPS US
- UPS Canada
- UPS Asia Pacific
- UPS Europe
- UPS Latin America



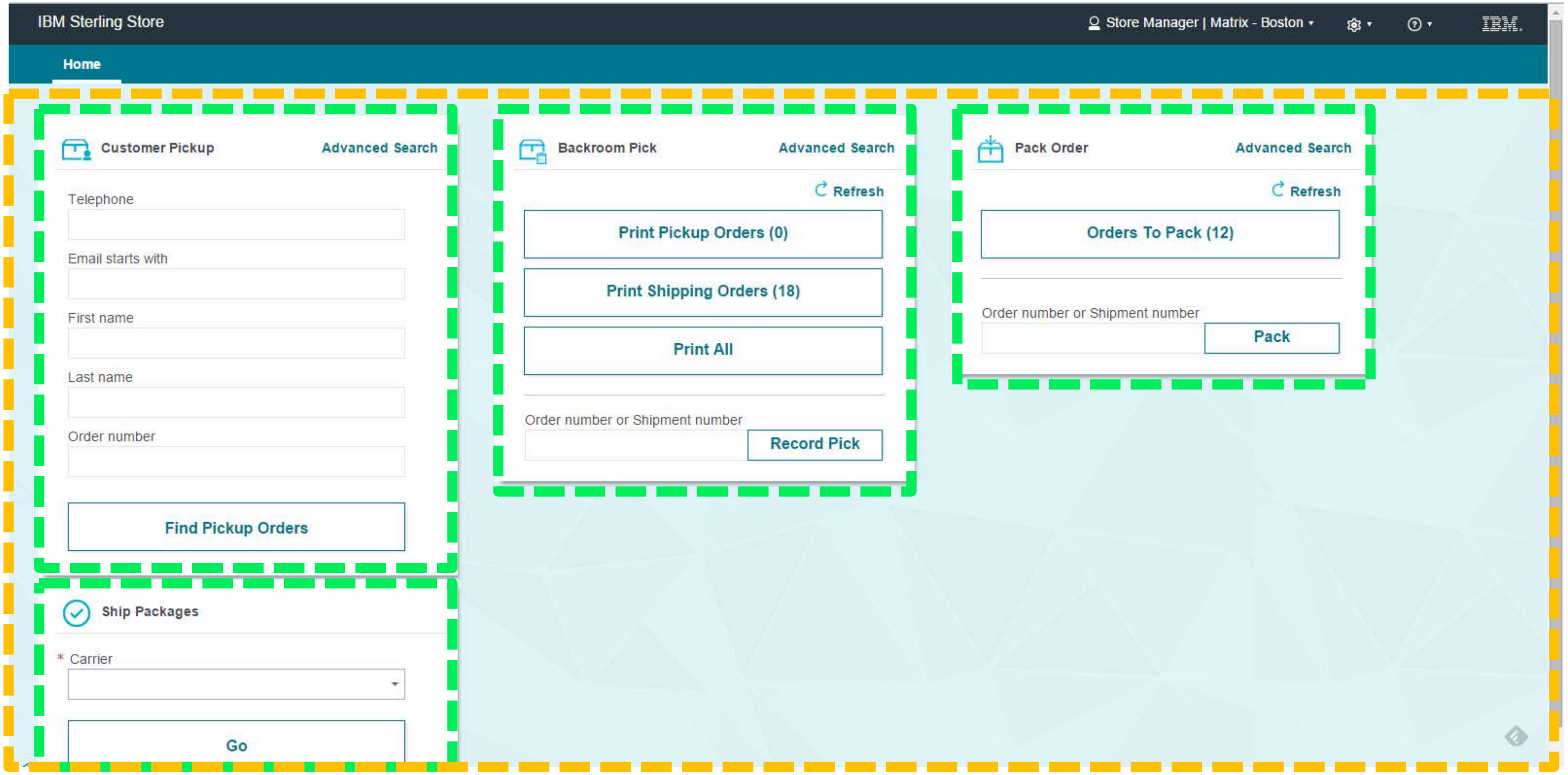
Fulfillment type

○ Pickup in store ● Ship from store ○ Both

  - Modeled as an object: attributes/properties (maintain state) and behaviors.
  - Dojo and IDX (IBM Dojo Extensions) widgets used.

- **uId**
  - Unique identifier for widget.
  - Unique in a screen vs. DOM id that must be unique in a browser tab.
  - Used mainly for extensibility.

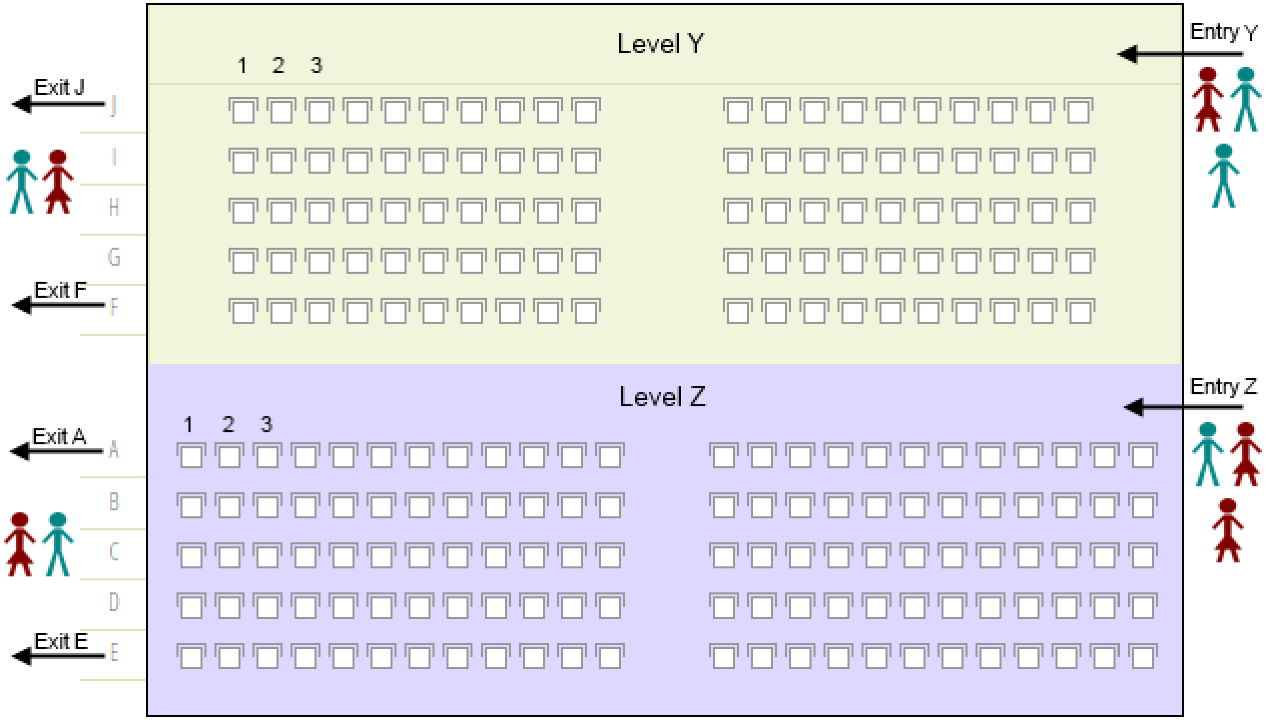
Screen, Model, Binding, Namespace



Theater

screen,
seats

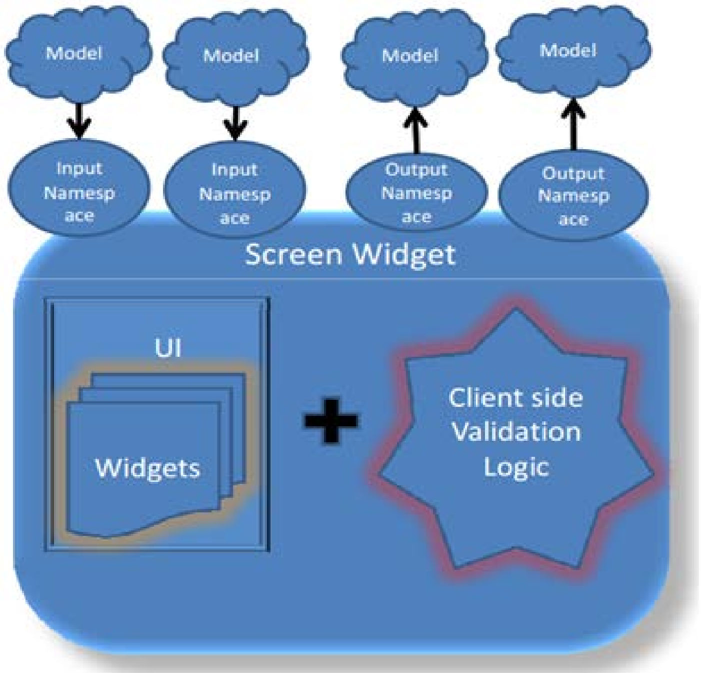
playMovie(),
letPeopleIn(),
letPeopleOut()



Theater	***Screen***
Seats	***Widgets***
People	***Model***
Entry doors	***Source namespaces***
Exit doors	***Target namespaces***
Tickets	***Binding***
letPeopleIn()	***setModel()***
letPeopleOut()	***getTargetModel()***

- *Ticket*: Associated with a Seat and equivalent of binding. *E.g.: YJ1-J1, ZA2-A2.*
  - Source (input) namespace: entry door. *E.g.: Y, Z.*
  - Target (output) namespace: exit door. *E.g.: J, A.*
  - Path: seat number. *E.g.: entry path-J1, exit path-1; entry path-A2, exit path-2.*

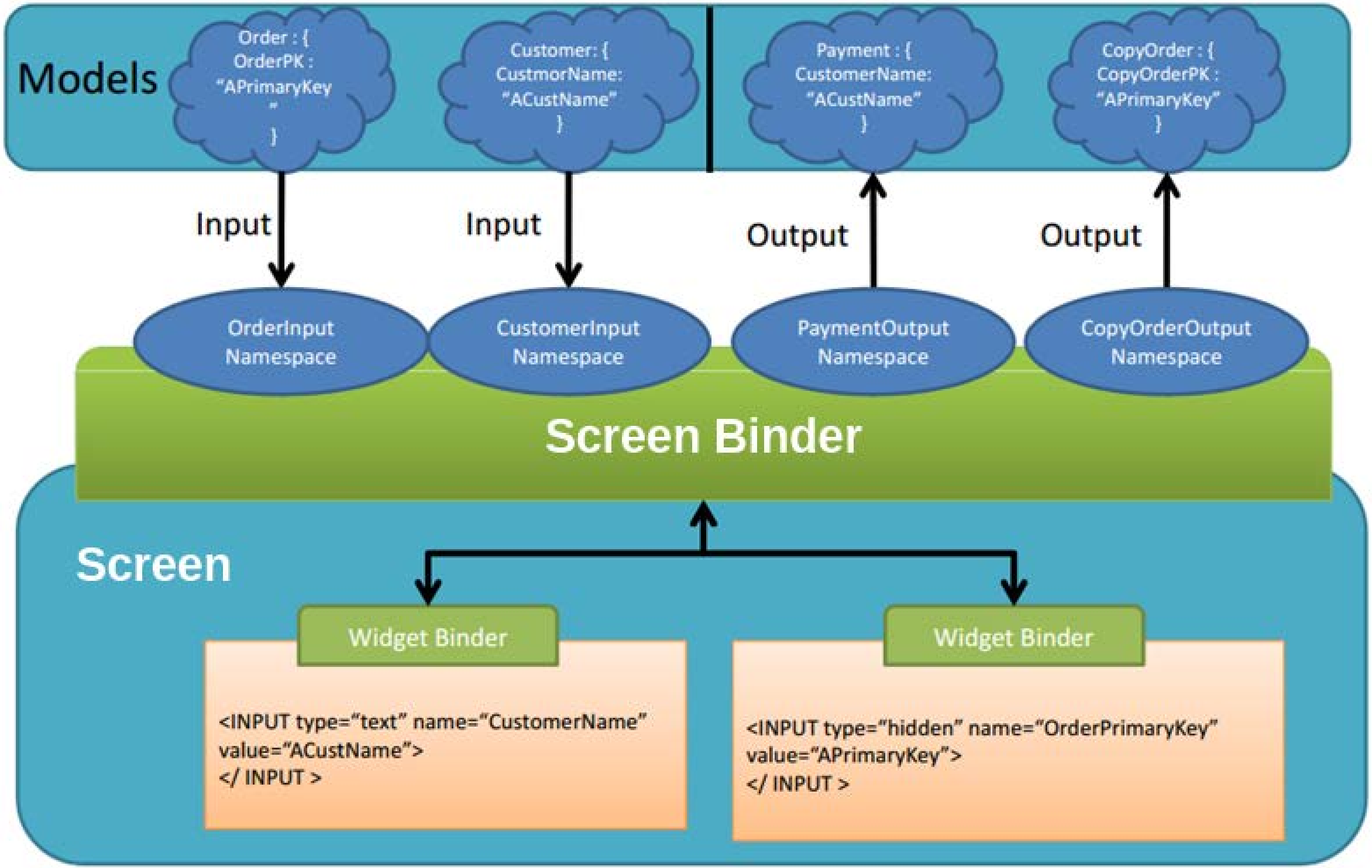
- **Screen**



- Re-usable composite widget that provides data binding and validation capability on its constituents. It is created dynamically from a template.
- **Template HTML**: provides the UI to the screen and contains all constituent widgets as they will be visible in UI.
- **JavaScript files**: Contain user defined behavior and events required for screen.
- **Screen Reference**: Special widget used to refer/use a screen inside another.
- **setModel() and getTargetModel()**: methods to show and get data from screen.

- **Model**
  - The data. Will be in JSON format on UI and XML at the backend.
- **Binding**
  - As a concept it means associating data with a widget/control. It has namespace and path components.
  - Framework supports one-way binding.
  - Main types:
    - *Source*: Input. Used to show data on UI.
    - *Target*: Output. Capture data from UI for server calls.
- **Namespace**
  - The input and output routes for data on screen or any widget.

- **Binding in action**

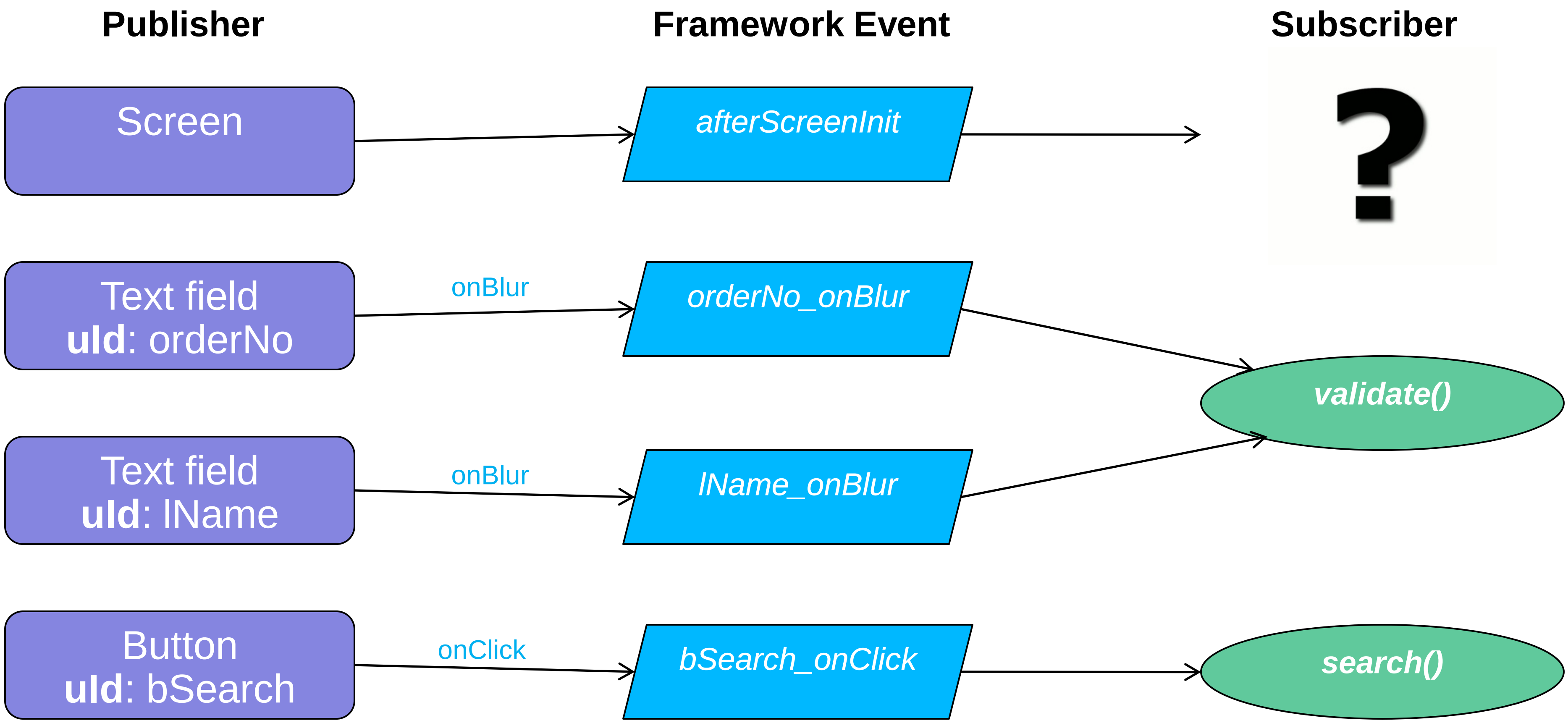


```
<INPUT type="text" name="CustomerName"
value="ACustName">
</ INPUT >
```

```
<INPUT type="hidden" name="OrderPrimaryKey"
value="APrimaryKey">
</ INPUT >
```

Events and Subscribers

- The event framework follows the pub-sub model.
- Screens and their constituent widgets generate/publish events.



- Events may be categorized as:

  1. ***UI events***: Published by widgets to notify any UI action.
     - **Examples**: onClick, onBlur, onChange etc.
     - Naming convention: ***<uId>_<UI event>***. E.g.: *btnSearch_onClick*.
     - Varies from widget to widget but list is fixed.
  2. ***Screen events***: Published by every screen to indicate completion of certain screen level activity.
     - No variation between screens: like UI events, these are fixed.

     i. Initialization events: Signify that screen is loaded and ready. E.g.: *afterScreenInit, afterScreenLoad.*

     ii. General events: Available on all screens. E.g.: *onExtnMashupCompletion, beforeBehaviorSetModel.*

3. ***Business events***: Custom events triggered programmatically by the application based on business logic.
   - Generally, not associated with any UI widget.
   - **Examples**: *backroompickComplete, containerCapacityExceeded, readyForCustomerPick* etc.
   - Events will vary between screens.
   - Usually used to communicate between screens.

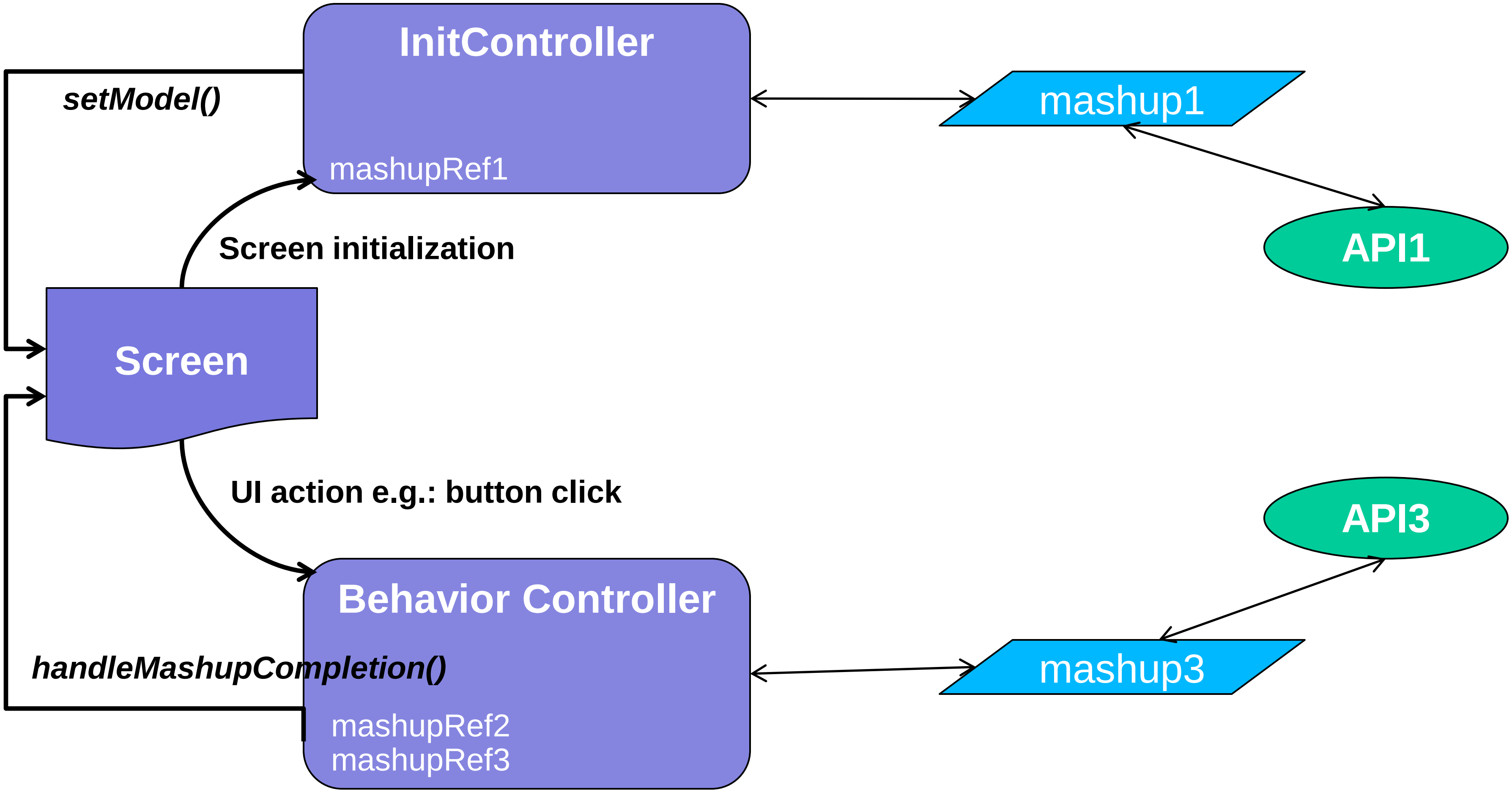
- **Subscribers**: Associated with an event.
  - Notified whenever relevant event is published. Has logic to be performed on an event.
  - No subscriber => no action performed.
  - ***Local subscribers***: called within context of a screen i.e. when an event is triggered, subscribers within that screen get notified.
  - ***Global subscribers***: called across screens i.e. when event is triggered, subscribers between present on all active screens get notified.

Mashup, Mashup Reference, Controllers

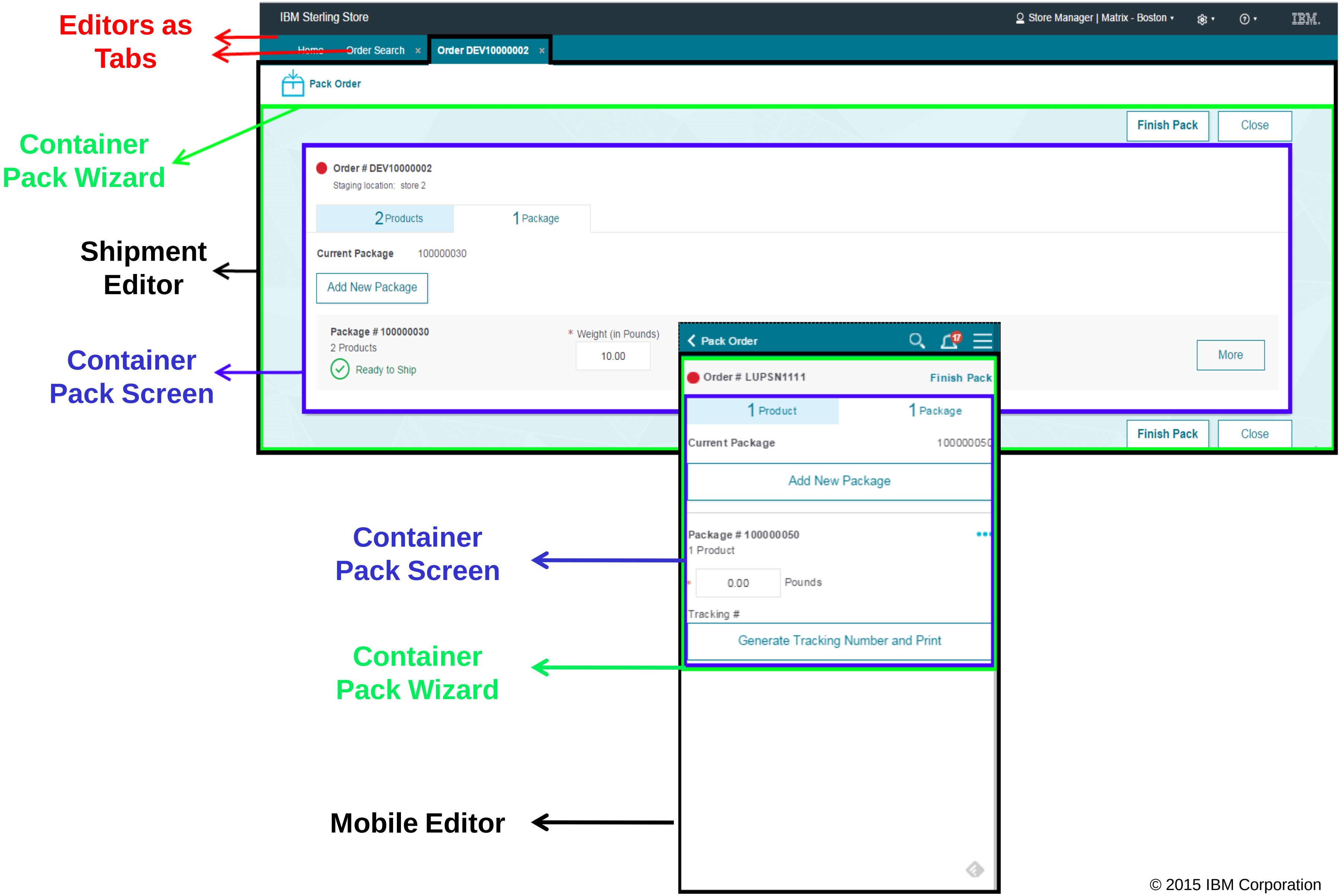
- **Mashup**
  - Backend component used to invoke APIs.
- **Mashup Reference**
  - Association between a screen and mashup(s).
  - Two types: init and behavior.

Mashup	Mashup Reference (MashupRef)
Contains information about API to call, input and output to the API.	Essentially, screen level a reference to the mashup being invoked. Contains mashup id and namespace.
Handles authorization, transforming data and invoking API/business logic.	Used by the controller to invoke the required mashup.
Context of a mashup is the entire application.	Context limited to a screen.
Identified by **id** and must be unique for the application.	Identified by **mashupRefId** and must be unique for the screen.

- **Controllers**
  - Act as mapping between client and the backend mashup layer.
  - Manage the flow of data from the UI to the server.
  - Init Controller:
    - Contains Init Mashup Refs that are invoked during screen initialization.
  - Behavior Controller:
    - Contains Behavior Mashup Refs that are generally invoked on UI action like: button click on the screen.


InitController
mashupRef1
setModel()
mashup1
API1
Screen initialization
Screen
UI action e.g.: button click
API3
Behavior Controller
handleMashupCompletion()
mashup3
mashupRef2
mashupRef3

Editor, Wizard



Editors as Tabs

Container Pack Wizard

Shipment Editor

Container Pack Screen

Container Pack Screen

Container Pack Wizard

Mobile Editor

IBM Sterling Store

Store Manager | Matrix - Boston ▾

Home | Order Search × | Order DEV10000002 ×

Pack Order

Finish Pack | Close

Order # DEV10000002

Staging location: store 2

2 Products | 1 Package

Current Package 100000030

Add New Package

Package # 100000030

2 Products

✓ Ready to Ship

* Weight (in Pounds) 10.00

More

Finish Pack | Close

Pack Order

Order # LUPSN1111 Finish Pack

1 Product | 1 Package

Current Package 100000050

Add New Package

Package # 100000050

1 Product

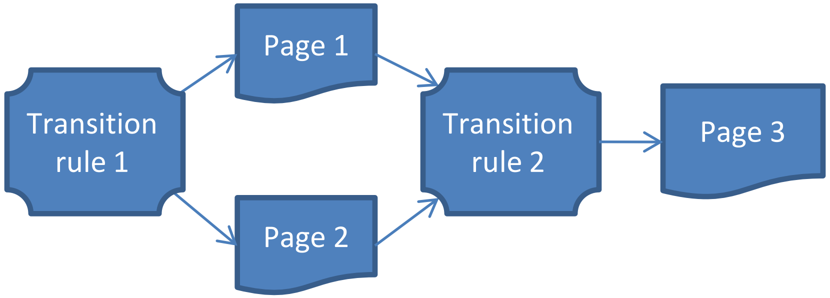
* 0.00 Pounds

Tracking #

Generate Tracking Number and Print

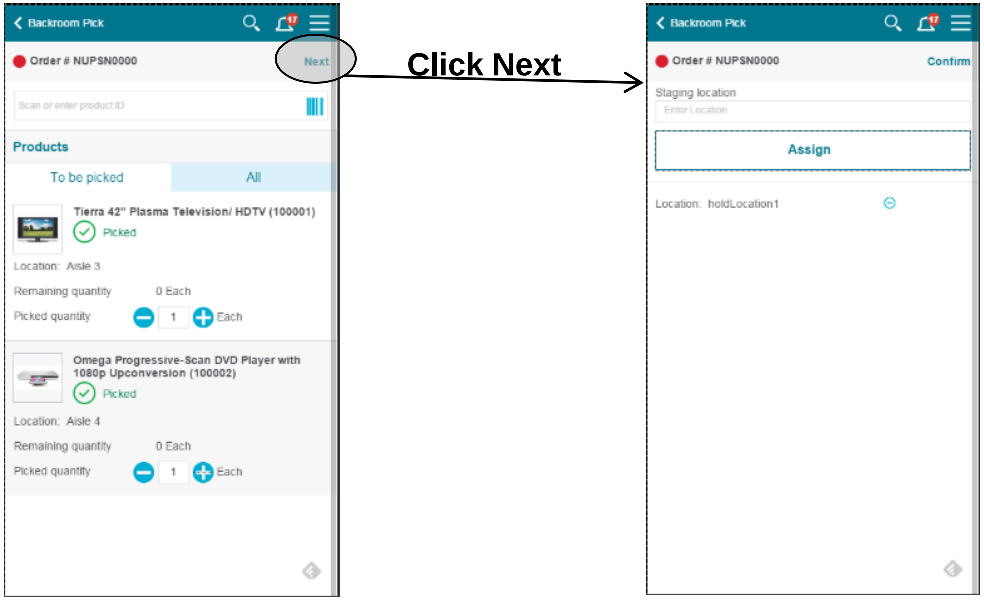
- **Editor:**
  - An editor usually corresponds to an entity. In Web Store we have editors like: Shipment, Product, Home etc.
  - An editor will contain placeholders to display screens and wizards.
- **Wizard:**
  - A set of screens loaded sequentially, commonly used to complete a series of operations that require more than one input.
  - Wizards are used to complete a flow in the application. E.g.: Backroom pick.
  - Transition between screens/pages is governed by rules.
  - Wizards components:
    - *Frontend component*: js and HTML files that render wizard on the browser.
    - *Backend component*: wizard flow definition xml and java implementation classes.

- Wizard definition xml (*Knowledge Center link in slide notes*):



Entities	*They can be pages or rules*
Transitions	*How to move between pages or rules*

- Example: Backroom pick wizard is a simple 2 page wizard.


Backroom Pick

Order # NUPSN0000 Next

Scan or enter product ID

Products

To be picked | All

Tierra 42" Plasma Television/ HDTV (100001)
Picked
Location: Aisle 3
Remaining quantity 0 Each
Picked quantity 1 Each

Omega Progressive-Scan DVD Player with 1080p Upconversion (100002)
Picked
Location: Aisle 4
Remaining quantity 0 Each
Picked quantity 1 Each

Click Next

Backroom Pick

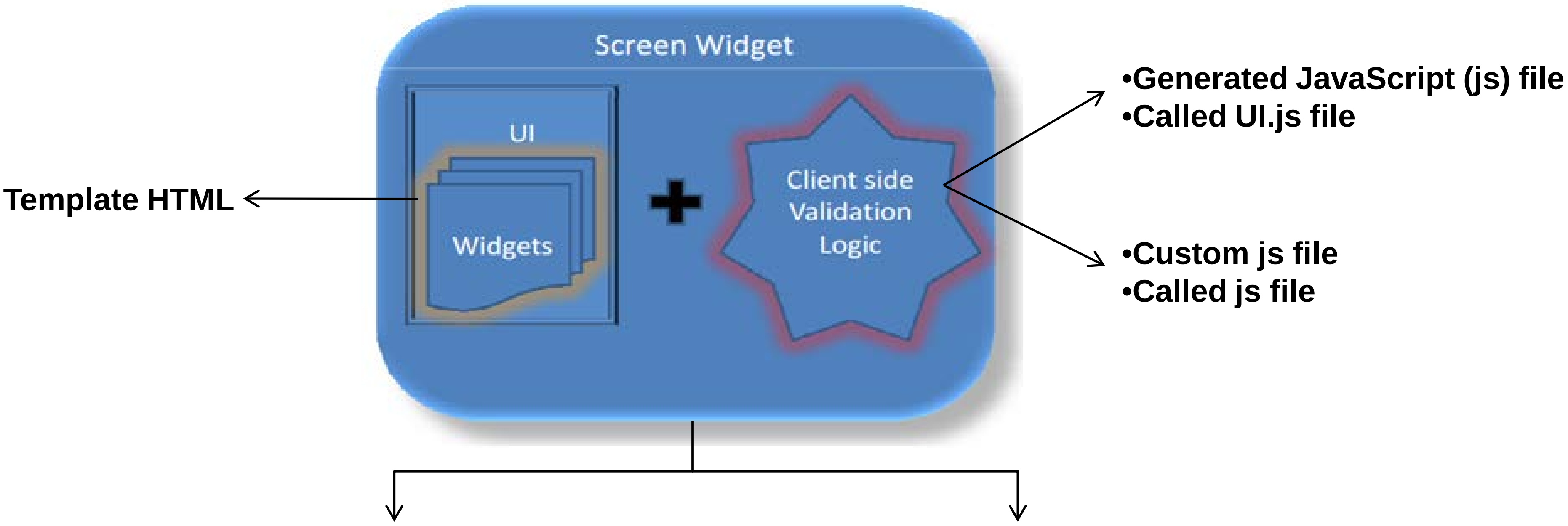
Order # NUPSN0000 Confirm

Staging location
Enter Location

Assign

Location: holdLocation1

Screen and its associated files



Template HTML

- •Generated JavaScript (js) file
- •Called UI.js file

- •Custom js file
- •Called js file

Init Controller js file

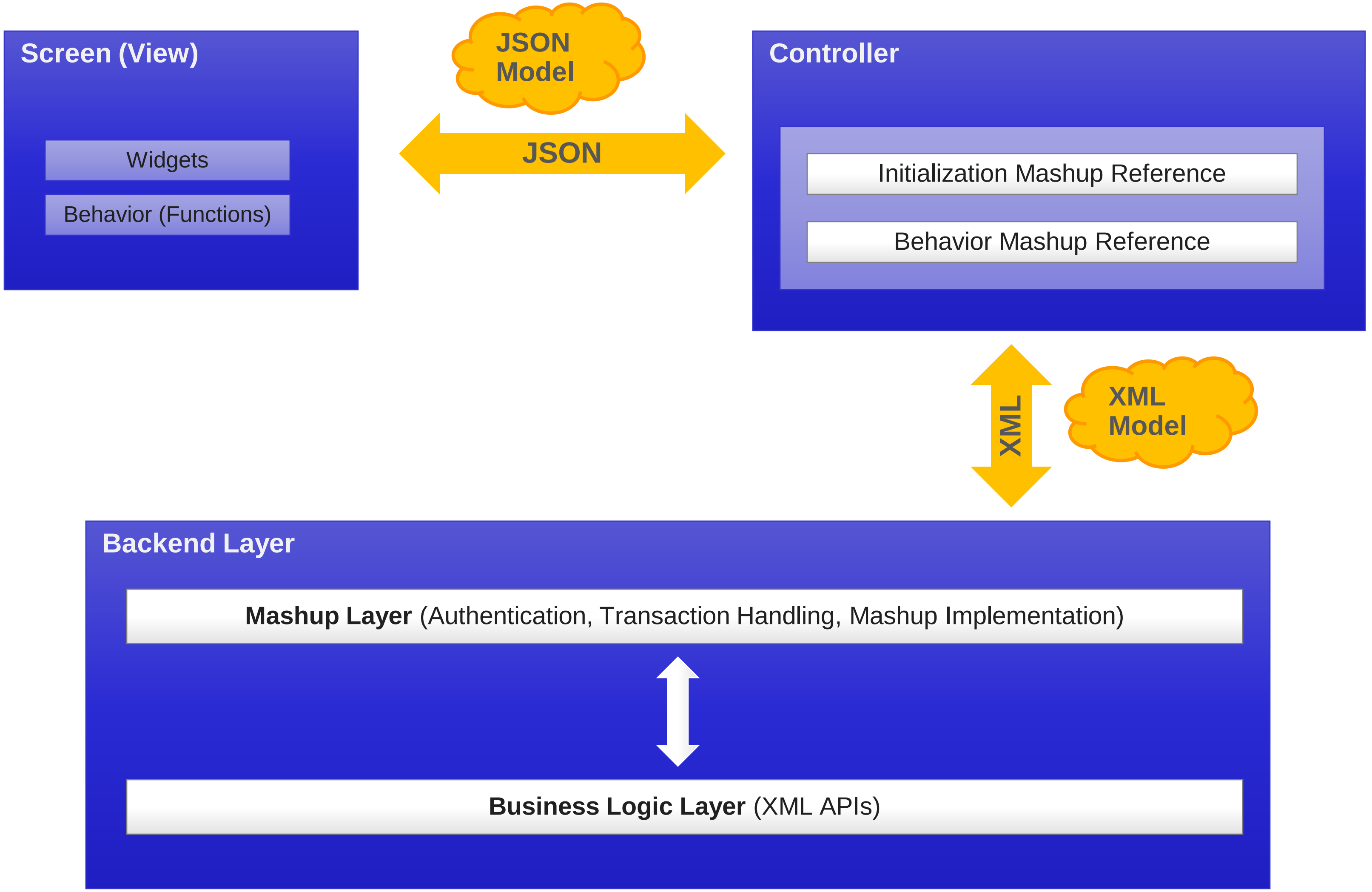
Behavior Controller js file

Files required at server side

Init Controller xml

Behavior Controller xml

How it all works together


Screen (View)
Widgets
Behavior (Functions)
JSON Model
JSON
Controller
Initialization Mashup Reference
Behavior Mashup Reference
XML
XML Model
Backend Layer
Mashup Layer (Authentication, Transaction Handling, Mashup Implementation)
Business Logic Layer (XML APIs)

- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- Use cases – demo
- Debugging Tips

Introduction

- o **Designed for future**: A design principle where the implementation considers future growth or modification.
- o **Runtime extensibility**: A system whose behavior is modifiable at runtime without recompiling or changing the original source code. UI extensibility in our framework follows this paradigm.

Need for extensibility is driven by:

- Integration with other applications.
- Changed use cases.
- Need to capture/show more relevant data to user.
- Need to display data in an easy to understand way.
- Other unforeseen requirements.

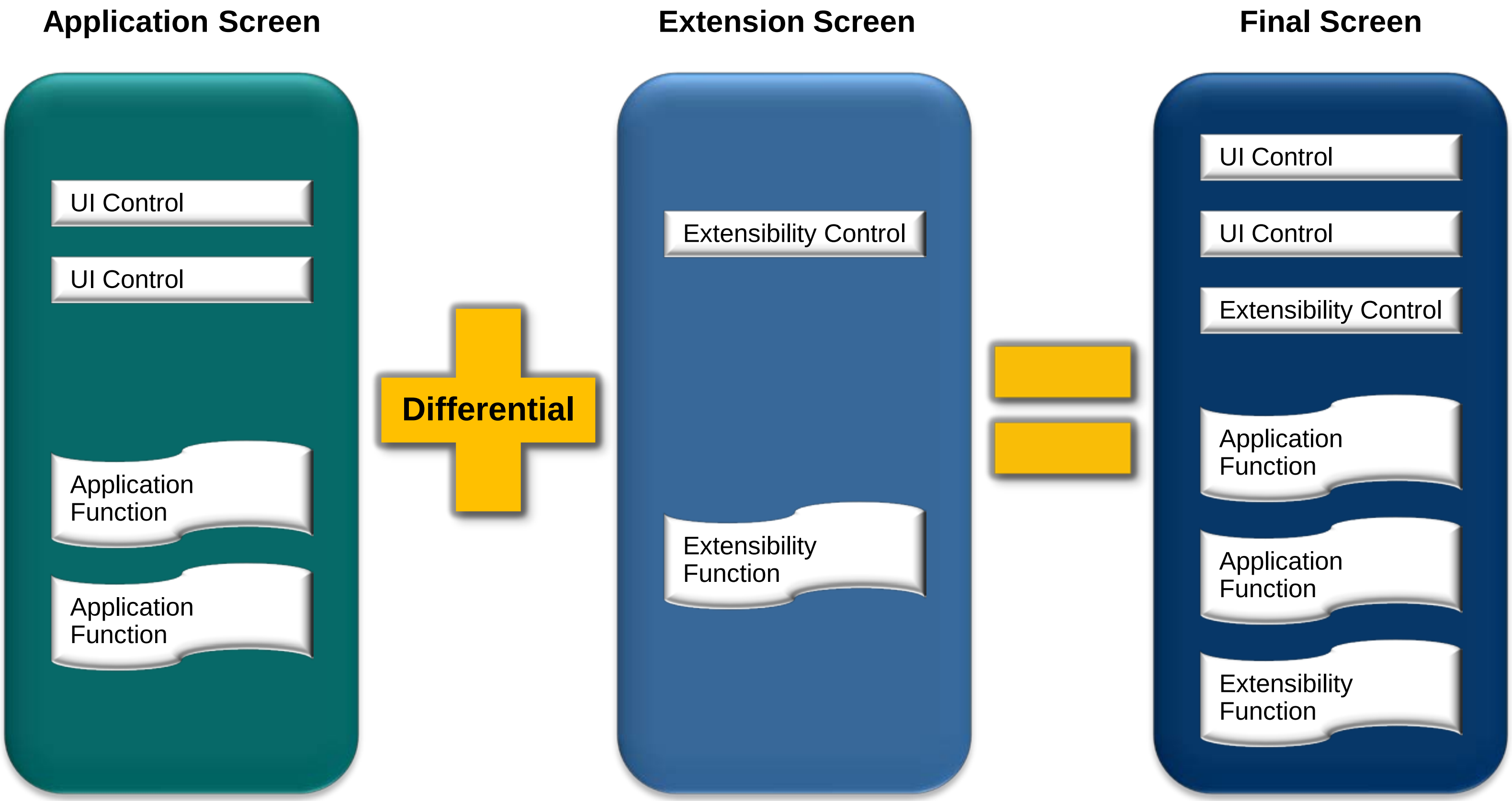
Extensibility should not be mistaken with extends (key word in java) or inheritance.

Types

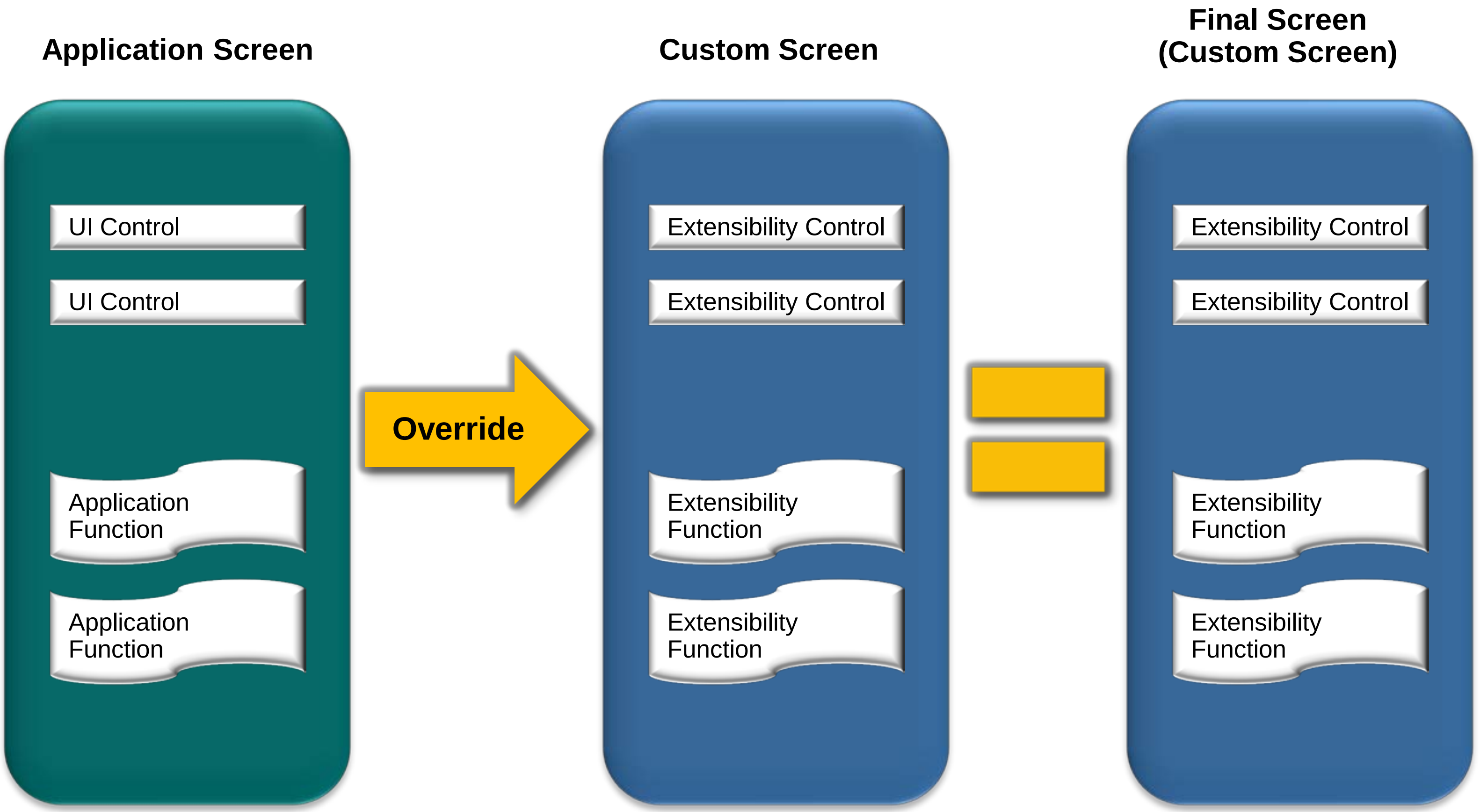
- **Types of Extensions**: There are 2 types of extensions supported.
  - Differential: Add or modify the existing entity. Typically stored in separate file.
  - Override: Replace the entity being extended.

Override	Differential
Pros: • Easy to change behavior, like a new screen.	**Pros:** • Less work and source code management. • Framework provides tools: Extensibility workbench.
Cons: • More work, as need to re-write existing functionality. • Upgrade issues.	**Cons:** • Work in a framework defined way.

- **Differential UI Extensibility**: Involves merging of behavior and UI.



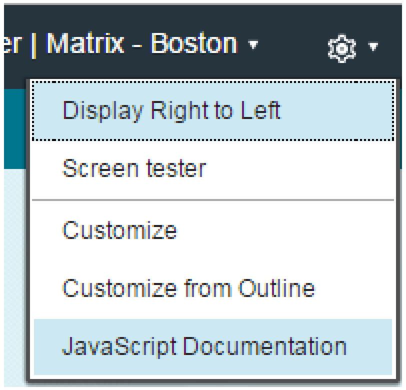
o Override UI Extensibility



- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- Use cases – demo
- Debugging Tips

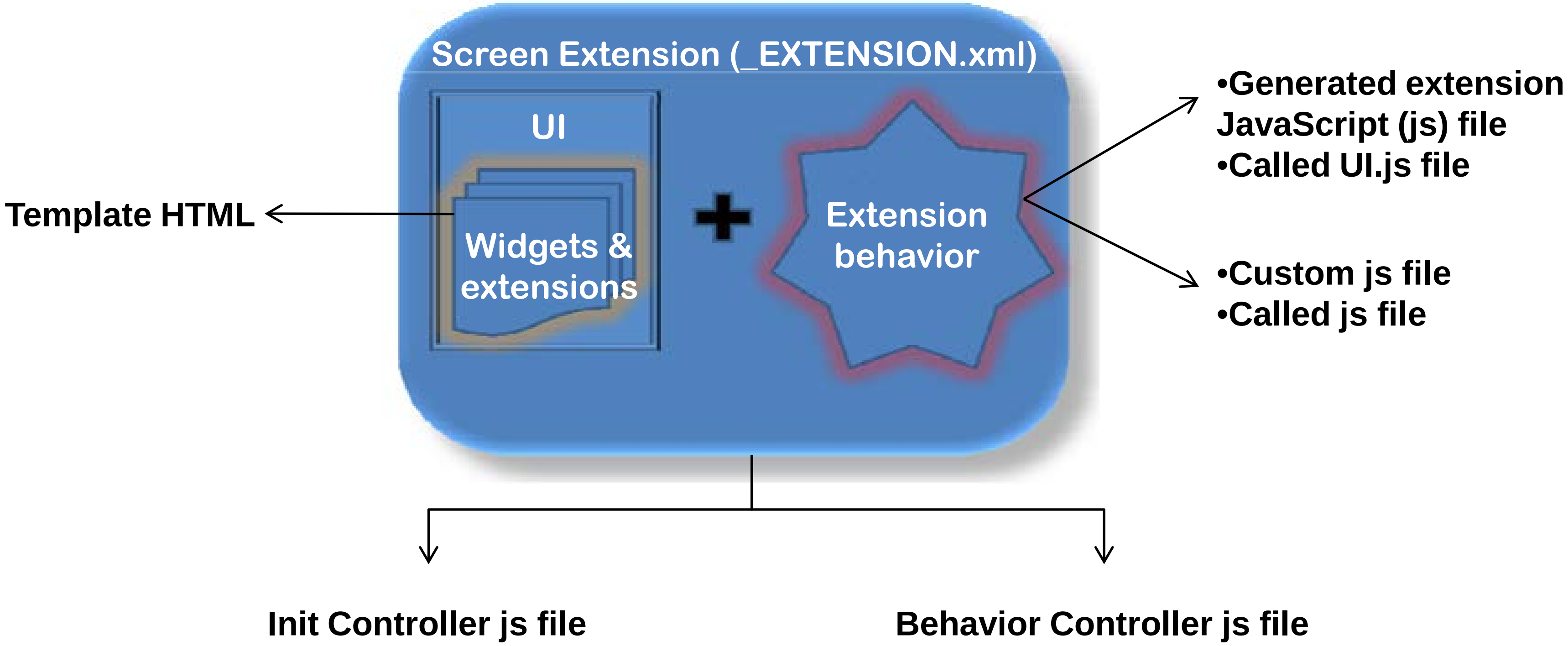
About the Tool

- Extensibility workbench is a WYSIWYG (what you see is what you get) tool.
- It can be used for:
  1. Differentially extending the screen: add/move/hide/change widgets.
  2. Adding new events and subscribers: subscriber method implementation required in custom js.
  3. Define new namespaces on screen.
  4. Extending mashups: differential/override/new.
  5. Viewing utility method documentation.
  6. Understanding the flow with the browser debugger.



- Tool’s WYSIWYG capability is not available for creating new screens or extending wizards and editors. There are well documented steps available for the same on the Knowledge Center: (*refer notes section on the slide*)

Screen extension and its associated files



Template HTML

- Generated extension JavaScript (js) file
- Called UI.js file

- Custom js file
- Called js file

Init Controller js file

Behavior Controller js file

Files required at server side

Init Controller xml

Behavior Controller xml

Files required by tool

Mashup xmls – if extended

Screenbundle xmls

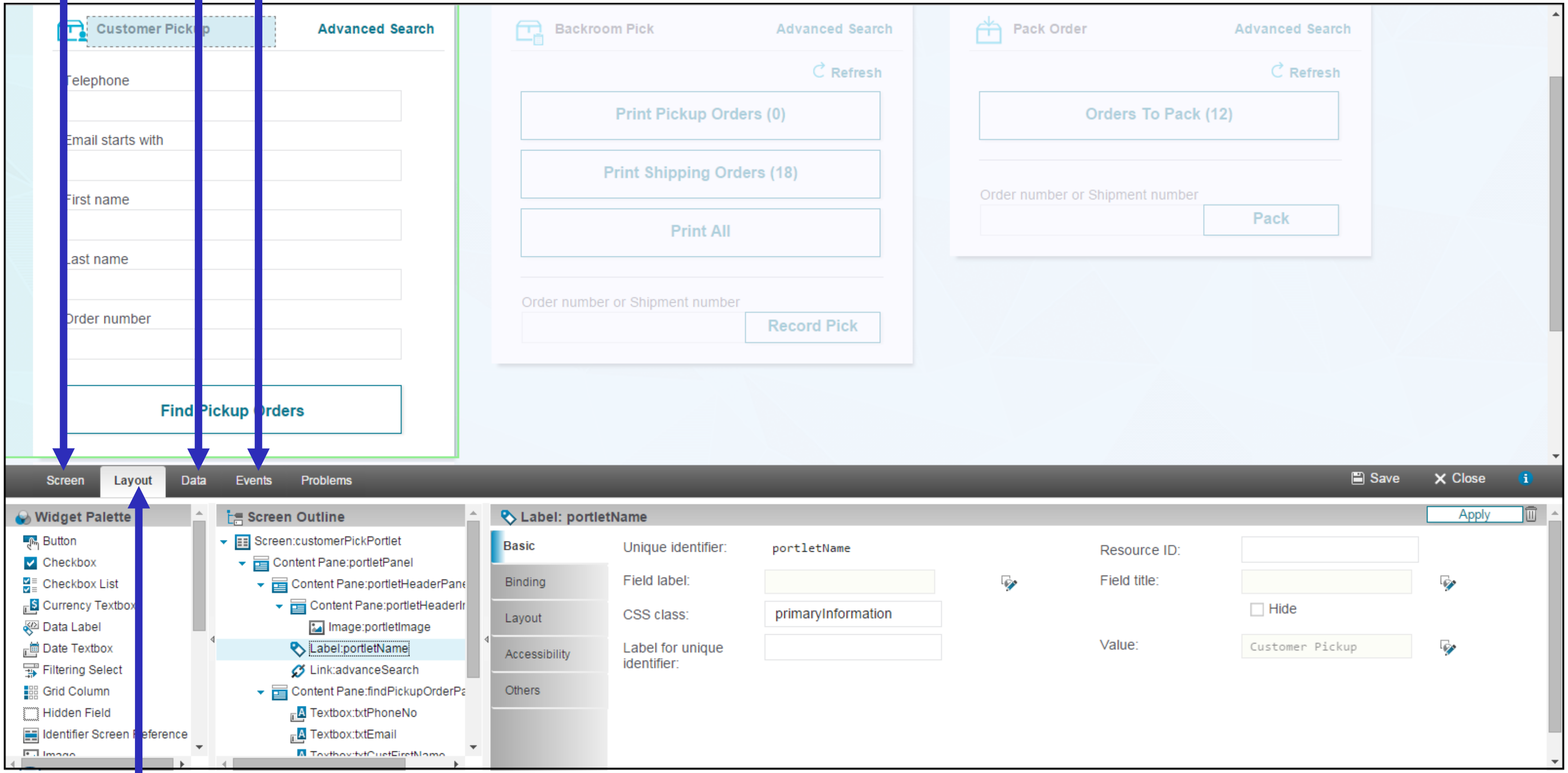
ExtnMetaData.xml

- **Before you begin.. a few things to know**
  - Installation or extension directory should be accessible:
    - *write access*: to save and modify extensions.
    - *read access*: to load/view extensions.
  - Deploy exploded development Web Archive (wscdev.war).
  - Ensure that the value of the ***uiExtensibilityMode*** system argument for the application server is set to true.
  - ***Further reference***: *http://www-01.ibm.com/support/knowledgecenter/SS6PEW_9.4.0/com.ibm.help.ws.custom.doc/c_understand_extensibility_tool.html*

Information about screen like: complete screen name, extension name, path etc.

Namespaces, Mashups and Mashup References

Events and subscribers



Unique identifier: portletName

Field label:

CSS class: primaryInformation

Label for unique identifier:

Resource ID:

Field title:

□ Hide

Value: Customer Pickup

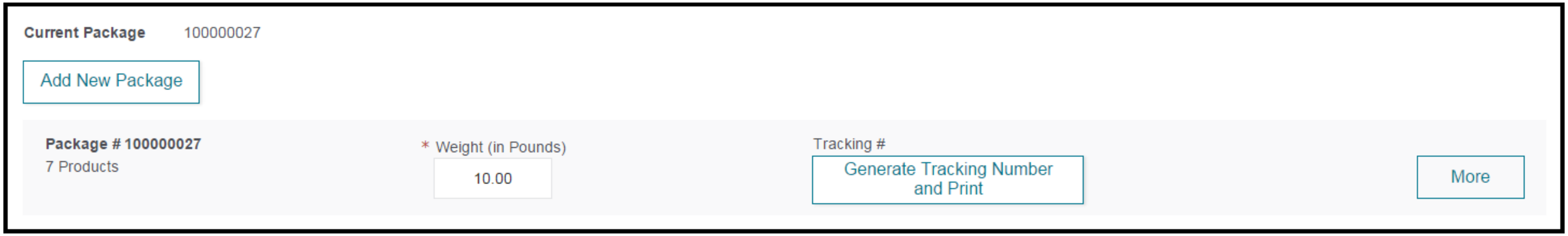
Widget Palette: Widgets that can be added to screen.
Screen Outline: Hierarchical representation of widgets as they appear on screen.
Properties: Editable properties for a widget.

- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- **Use cases – demo**
- Debugging Tips

Use Case 1: Ship from Store

- **Problem Statement:**
  - Customize Web Store to enable a store associate to manually capture tracking information on an Order.

- **Current UI:**



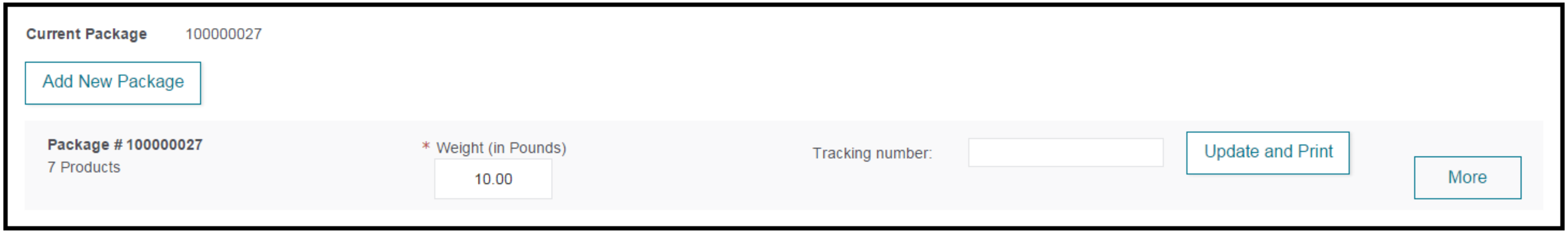
Current Package 100000027

Package # 100000027
7 Products

* Weight (in Pounds) 10.00

Tracking #

- **Extended UI:**



Current Package 100000027

Package # 100000027
7 Products

* Weight (in Pounds) 10.00

Tracking number:

- **Action required: Differentially Extend the screen(s)**

Steps	Action	Tool to use
1	Add new widgets	Extensibility workbench
2	Add event subscribers	Extensibility workbench
3	Add behavior mashup	Extensibility workbench
4	Provide implementation for all subscribers	Any JavaScript editor

- **Further reference:** *refer notes section on this slide*
- **Refer next slide for detailed steps**
- **Short hand notations used in next slide:**

1. ContainerPackContainerList (**List Screen**) and ContainerPackContainerView (**View Screen**). **View Screen** is the parent screen of **List Screen**.
2. Extensibility workbench: **EWB**
3. Any JavaScript editor: **JSE**

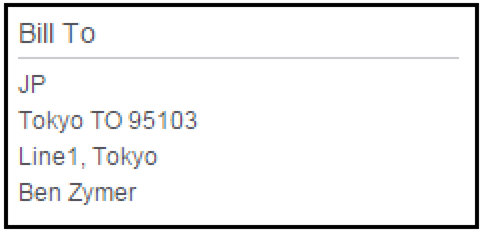
#	Action	Tool
1	**List Screen:** Add required widgets	**EWB**
2	**List Screen:** Add event subscribers to button click	**EWB**
3	**List Screen:** Provide target binding for the textfield	**EWB**
4	**List Screen:** Provide implementation for *subscriber* associated in **#2** – a) get tracking number entered b) fire an event to parent	**JSE**
5	**View Screen:** Create new mashup ref and mashup	**EWB**
6	**View Screen:** Add subscribers for: a) event published by child in **#4b** b) screen event: ***onExtnMashupCompletion***	**EWB**
7	**View Screen:** Provide implementation for *subscriber* associated in **#6a** – invoke mashup defined in **#5**	**JSE**
8	**View Screen:** Provide implementation for *subscriber* associated in **#6b** – a) Open print dialog to print tracking information b) fire event to child	**JSE**
9	**List Screen:** Add event subscriber for event published in **#8b**	**EWB**
10	**List Screen:** Provide implementation for *subscriber* associated in #9 - update the data on screen	**JSE**

Use Case 2: Extend Address panels

- **Problem Statement:**
  - We need to move the Customer Name field from its location at the end to the 1st position in the address panel for a Japanese address.

- **Current UI:**

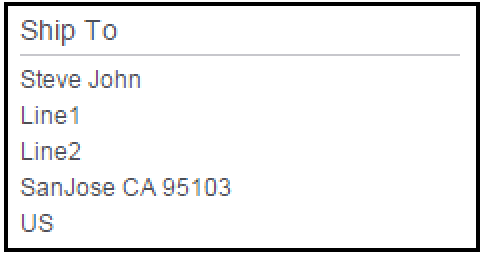
JP



Bill To

JP
Tokyo TO 95103
Line1, Tokyo
Ben Zymer

US

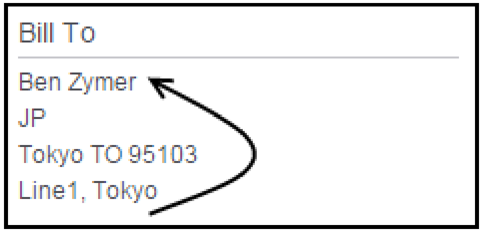


Ship To

Steve John
Line1
Line2
SanJose CA 95103
US

- **Extended UI:**

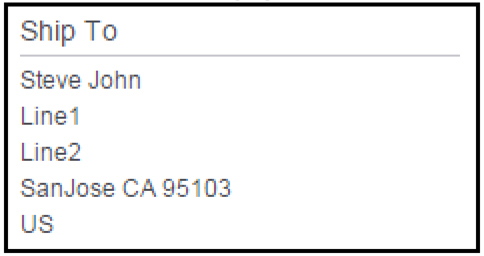
JP



Bill To

Ben Zymer
JP
Tokyo TO 95103
Line1, Tokyo

US



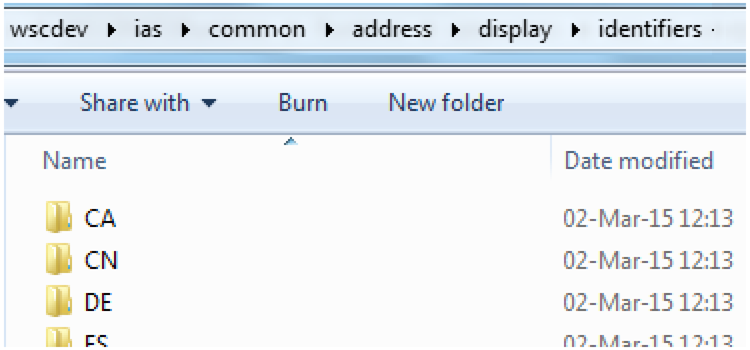
Ship To

Steve John
Line1
Line2
SanJose CA 95103
US

- Address Panel is a special screen – Identifier Screen.
- **What is Identifier Screen?**
  - Special screen that has multiple UI flavors, each of which maps to a unique identifier. Thus, the name.
  - Without any input, the screen does not have any UI.
  - The template HTML is computed in memory at runtime based on some data/condition that returns the UI identifier to be loaded.
- **Why Identifier Screen?**
  - Typically used when the UI of a screen varies significantly based on data but underlying behavior and mashups invoked remain same.
  - Each country has different address format: Thus, UI needs to vary accordingly.
  - Country code is the distinguishing parameter here and can be used as identifier.

Identifier Screen: components

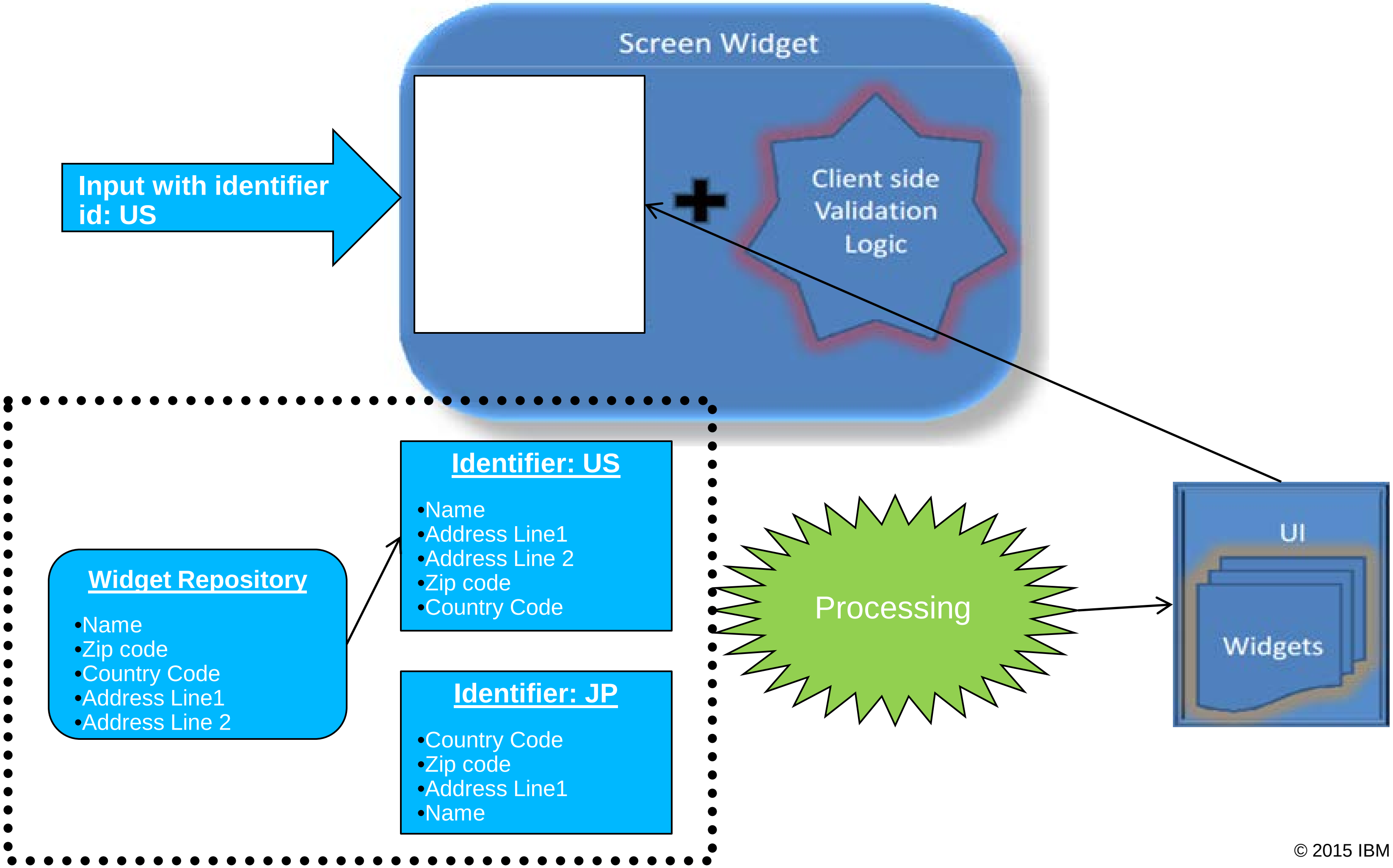
- New component: Identifier HTML file.
- Widget repository: normal HTML file.
- Identifier HMTL file is essentially a wireframe/blueprint that contain information about widgets arrangement on the UI.
- Widgets are **referred** here. Widgets substituted in these placeholders from repository at runtime.


wscdev ▸ ias ▸ common ▸ address ▸ display ▸ identifiers

Name	Date modified
CA	02-Mar-15 12:13
CN	02-Mar-15 12:13
DE	02-Mar-15 12:13
ES	02-Mar-15 12:13

Widget Repository	Identifier
Contains all widgets used across all identifiers.	Each identifier contains widgets required by it.
Flat list – no arrangement.	Widgets arranged as they should appear in UI.
Complete widget definition.	Any properties to be changed defined.

Identifier Screen: working



Input with identifier id: US

Screen Widget

Client side Validation Logic

Widget Repository

- Name
- Zip code
- Country Code
- Address Line1
- Address Line 2

Identifier: US

- Name
- Address Line1
- Address Line 2
- Zip code
- Country Code

Identifier: JP

- Country Code
- Zip code
- Address Line1
- Name

Processing

UI

Widgets

- **Action required: Differentially Extend the screen for Japanese identifier only**

Steps	Action	Tool to use
1	Move the required widgets	Extensibility workbench

- **Further reference:** *http://www-01.ibm.com/support/knowledgecenter/SS6PEW_9.4.0/com.ibm.help.ws.custom.doc/t_ws_movingwidgetsinanidentifierscreen.html*

Use Case 3: Adding Related task

- **Problem Statement:**
  - We need to add a new Related Task with a link for Advanced Search on the Home page in Store.

- **Action required: Create a custom screen for Related Task**

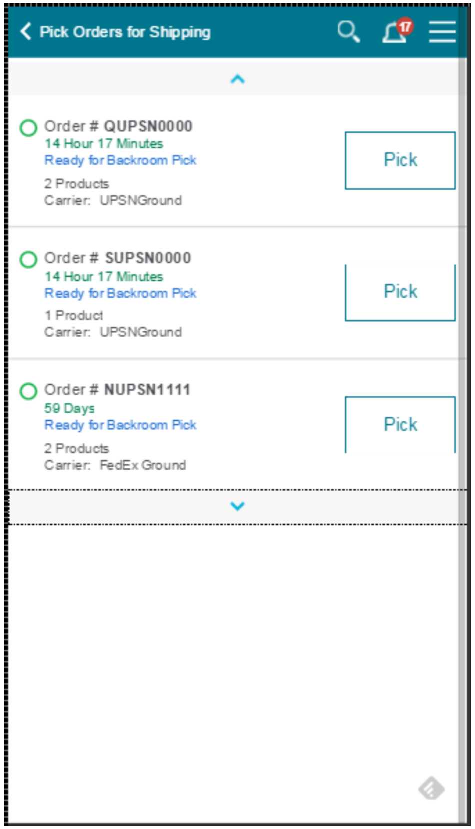
Steps	Action	Tool to use
1	Create a new Screen with an Advanced Search link	Any JavaScript/HTML editor
2	Link new screen to the Home page using Screen Reference widget	Extensibility workbench
3	Add business logic for Advanced Search link	Any JavaScript editor

- **Further reference:** *http://www-01.ibm.com/support/knowledgecenter/SS6PEW_9.4.0/com.ibm.help.ws.custom.doc/c_create_custom_screens.html*

Use Case 4: Customizing page size

- **Problem Statement:**
  - Change the default records to fetch while searching for Orders on the mobile and desktop screens to 10.

- **Current UI:**



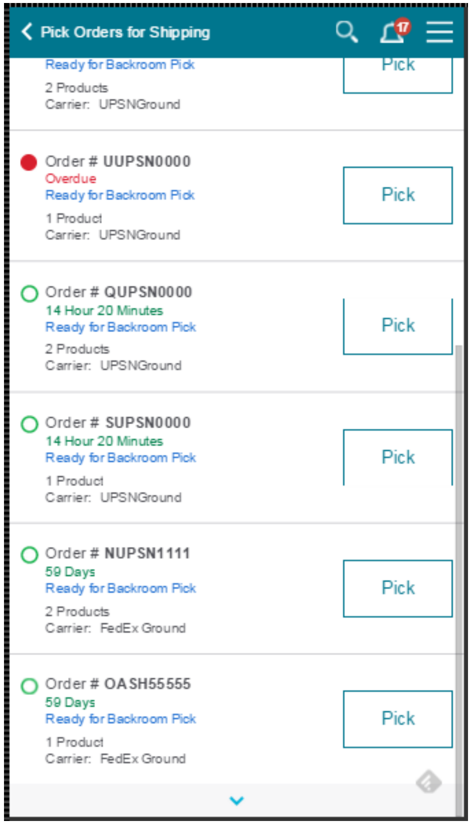
Pick Orders for Shipping

○ Order # QUPSN0000
14 Hour 17 Minutes
Ready for Backroom Pick
2 Products
Carrier: UPSNGround
Pick

○ Order # SUPSN0000
14 Hour 17 Minutes
Ready for Backroom Pick
1 Product
Carrier: UPSNGround
Pick

○ Order # NUPSN1111
59 Days
Ready for Backroom Pick
2 Products
Carrier: FedEx Ground
Pick

Extended UI:



Pick Orders for Shipping

Ready for Backroom Pick
2 Products
Carrier: UPSNGround
Pick

● Order # UUPSN0000
Overdue
Ready for Backroom Pick
1 Product
Carrier: UPSNGround
Pick

○ Order # QUPSN0000
14 Hour 20 Minutes
Ready for Backroom Pick
2 Products
Carrier: UPSNGround
Pick

○ Order # SUPSN0000
14 Hour 20 Minutes
Ready for Backroom Pick
1 Product
Carrier: UPSNGround
Pick

○ Order # NUPSN1111
59 Days
Ready for Backroom Pick
2 Products
Carrier: FedEx Ground
Pick

○ Order # OASH55555
59 Days
Ready for Backroom Pick
1 Product
Carrier: FedEx Ground
Pick

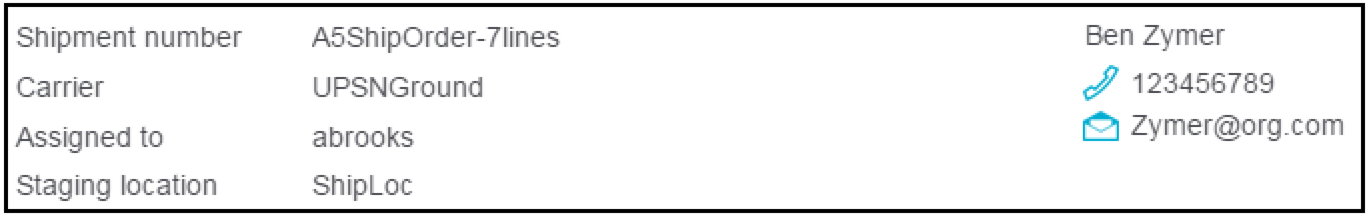
- **Action required: Create page size definition JSON file**

Steps	Action	Tool to use
1	Create the page size definition JSON file in extension directory. Re-build the WAR/EAR and re-start server.	Any JavaScript editor

- **Further reference:** *http://www-01.ibm.com/support/knowledgecenter/SS6PEW_9.4.0/com.ibm.help.ws.custom.doc/t_pagination_extensibility.html*

Use Case 5: Phone number formatting

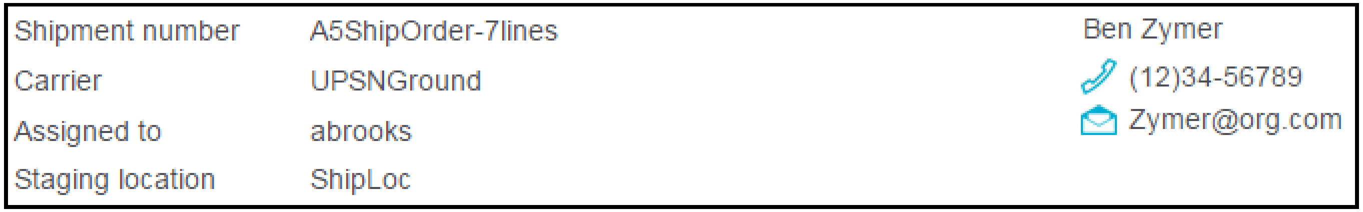
- **Problem Statement:**
  - Format the phone number on the UI.
- **Current UI:**



Shipment number	A5ShipOrder-7lines
Carrier	UPSNGround
Assigned to	abrooks
Staging location	ShipLoc

Ben Zymer
☎ 123456789
✉ Zymer@org.com

- **Extended UI:**



Shipment number	A5ShipOrder-7lines
Carrier	UPSNGround
Assigned to	abrooks
Staging location	ShipLoc

Ben Zymer
☎ (12)34-56789
✉ Zymer@org.com

- **Action required: Create new formatter and associate it with widget**

Steps	Action	Tool to use
1	Create a phone number formatter	Any JavaScript editor
2	Register the formatter	Any JavaScript editor
3	Provide the registered formatter for phone number field	Extensibility workbench

- **Further reference:** *http://www-01.ibm.com/support/knowledgecenter/SS6PEW_9.4.0/com.ibm.help.ws.custom.doc/t_ws_enablingformattingforphonenumberfields.html*

- Framework Terminology
- Extensibility Concepts
- Using Extensibility Workbench
- Use cases – demo
- **Debugging Tips**

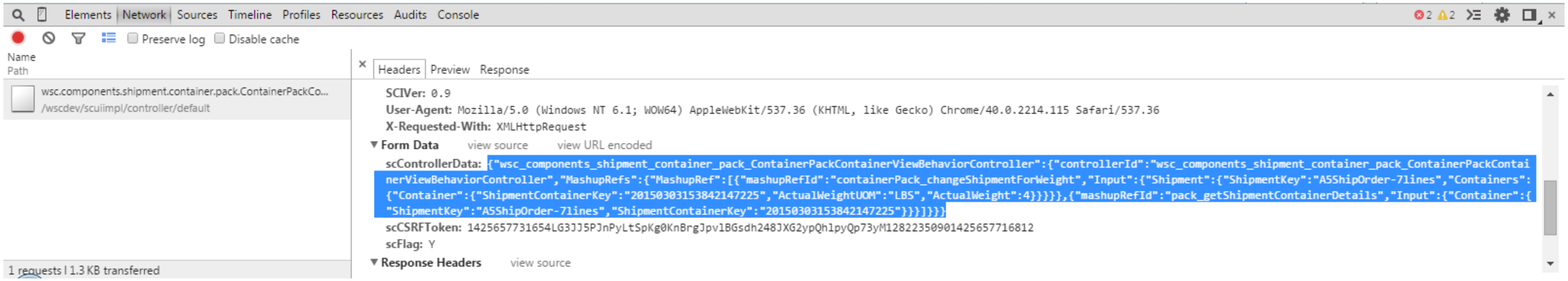
Using the tool and browser debugger

- Debugging the action occurring on click of Update button on the ContainerPackContainerList screen (updates container weight).

Requirement	How to do?
To get name of screen being extended: ***ContainerPackContainerList***	**EWB** – Screen tab
To get the method invoked on button click: ***saveContainerWeight()***	**EWB** – select button and refer Events in Properties view (Layout tab)
To view the source code loaded for screen for debugging	**Debugger** – search for the selected screen name.
To follow the flow of control once button is clicked	**Debugger** – •locate for the method and put a breakpoint. •Exit EWB and click on the button

o Identifying data sent and received by screen: Select the Network tab on browser and clear all old requests and click on Update button in UI.

1. Select the selected request on left to view the request headers. Posted data is under **scControllerData** (highlighted in blue for effect).



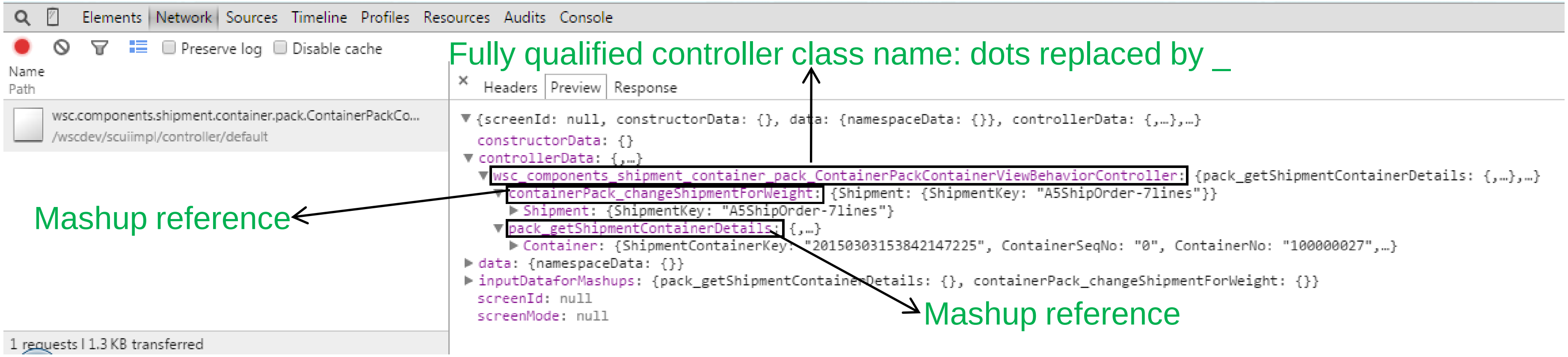
wsc.components.shipment.container.pack.ContainerPackCo...
/wscdev/scuiimpl/controller/default

```
SCIVer: 0.9
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115 Safari/537.36
X-Requested-With: XMLHttpRequest
▼ Form Data      view source      view URL encoded
scControllerData: {"wsc_components_shipment_container_pack_ContainerPackContainerViewBehaviorController":{"controllerId":"wsc_components_shipment_container_pack_ContainerPackContai
nerViewBehaviorController","MashupRefs":{"MashupRef":[{"mashupRefId":"containerPack_changeShipmentForWeight","Input":{"Shipment":{"ShipmentKey":"A5ShipOrder-7lines","Containers":
{"Container":{"ShipmentContainerKey":"20150303153842147225","ActualWeightUOM":"LBS","ActualWeight":4}}}}},{"mashupRefId":"pack_getShipmentContainerDetails","Input":{"Container":{
"ShipmentKey":"A5ShipOrder-7lines","ShipmentContainerKey":"20150303153842147225"}}}]}}}
scCSRFToken: 1425657731654LG3JJ5PJnPyLtSpKg0KnBrgJpvlBGsdh248JXG2ypQhlpyQp73yM12822350901425657716812
scFlag: Y
▼ Response Headers      view source
```

2. **Tip**: Copy the highlighted text and switch to Console tab.

    - Type ***dojo.fromJson(***<put copied text here as a valid string>***)*** – dojo method to convert a json string to a json object.
    - This creates a json object which is easy to understand.

3. Select Preview to view response as json at the expanded path under **controllerData**:



wsc.components.shipment.container.pack.ContainerPackCo...
/wscdev/scuiimpl/controller/default

1 requests | 1.3 KB transferred

```
▼{screenId: null, constructorData: {}, data: {namespaceData: {}}, controllerData: {,…},…}
  constructorData: {}
▼ controllerData: {,…}
  ▼ wsc_components_shipment_container_pack_ContainerPackContainerViewBehaviorController: {pack_getShipmentContainerDetails: {,…},…}
    ▼ containerPack_changeShipmentForWeight: {Shipment: {ShipmentKey: "A5ShipOrder-7lines"}}
      ▶ Shipment: {ShipmentKey: "A5ShipOrder-7lines"}
    ▼ pack_getShipmentContainerDetails: {,…}
      ▶ Container: {ShipmentContainerKey: "20150303153842147225", ContainerSeqNo: "0", ContainerNo: "100000027",…}
▶ data: {namespaceData: {}}
▶ inputDataforMashups: {pack_getShipmentContainerDetails: {}, containerPack_changeShipmentForWeight: {}}
  screenId: null
  screenMode: null
```

Fully qualified controller class name: dots replaced by _

Mashup reference

Mashup reference

4. Identifying if error occurred on server side:

- Response might contain some error code or error string.
- Data might be unexpected/invalid.
- In such scenarios, error may be at Mashup layer level or API level.
- No additional tooling/debugging capability is provided.
- Checking server logs and enabling API trace and verbose logs may be required.

Questions?

Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, Coremetrics, DB2, PowerVM, Rational, WebSphere, and z/VM are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.