DUMPSBOSS.

Vlocity Platform Developer Exam (v5.0)

Vlocity Vlocity-Platform-Developer

Version Demo

Total Demo Questions: 9

Total Premium Questions: 97

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, DataRaptors	21
Topic 2, Integration Procedures	25
Topic 3, OmniScripts	21
Topic 4, Vlocity Cards	24
Topic 5, Vlocity CPQ	6
Total	97

QUESTION NO: 1

In which two fields in an Integration Procedure or DataRaptor can you use a function like

CONCAT or DATEDIFF?

Choose 2 answers

A. In a Set Values Action in a Value field.

B. In a Remote Action in an Additional Output value field.

C. In a DataRaptor Action in an Input Parameters value field.

D. In a DataRaptor in an Output Tab Output JSON Path.

ANSWER: A C

Explanation:

Functions such as CONCAT and DATEDIFF are supported in expression-capable value fields, where OmniStudio evaluates the expression at runtime and can combine literal values with data from the procedure's JSON context. In a Set Values Action in a Value field, a function can calculate or compose a value before the Integration Procedure continues. This is a common pattern for deriving values, formatting strings, and constructing values needed by subsequent actions.

In a DataRaptor Action in an Input Parameters value field, functions can likewise be used to calculate the value passed into the invoked DataRaptor. This permits the Integration Procedure to transform input dynamically—for example, assembling a composite string with CONCAT or calculating a date-related value with DATEDIFF—before DataRaptor processing begins. In both cases, the value field is intended to accept OmniStudio merge syntax and functions rather than merely identifying a JSON location.

For further context on configuring Integration Procedure actions and passing values between actions, see the Salesforce Trailhead Integration Procedures module. Salesforce's Integration Procedure documentation also describes how Integration Procedures orchestrate actions and transform runtime data.

QUESTION NO: 2

You are configuring the API URL in an HTTP Action element within an Integration

Procedure. What is the merge code syntax for passing a

Date node from an element named SetValues in the URL?

A. {{SetValues. Date} Calculator

B. [SetValues' [Date] on

C. %SetValues:Date%

D. %Setvalues.Date%

ANSWER: C

Explanation:

`%SetValues:Date%` is the correct merge-code syntax for inserting the value of the `Date` node produced by the `SetValues` element into an HTTP Action URL. Integration Procedures use percent signs to delimit a merge expression, with the element name and the node name separated by a colon. At run time, the Integration Procedure resolves this expression from its working data JSON and substitutes the resulting value directly into the configured URL. This lets an HTTP Action build a dynamic endpoint or query-string parameter from data created earlier in the procedure without requiring custom code. The element name must match the Integration Procedure element's configured name, including its capitalization, and the node reference must match the data key made available by that element. For a date value, ensure that the generated value is in the format expected by the target API and URL-encode it when the API contract requires

encoding. Salesforce Industries describes Integration Procedures as server-side processes that orchestrate actions and pass data between elements; merge fields provide the mechanism for reusing that runtime data in element configuration. See Salesforce Industries Integration Procedures documentation and Trailhead: Get Started with Integration Procedures.

QUESTION NO: 3

An Integration Procedure calls an external service. If the service returns an error, the procedure must execute an alternate error-handling path instead of continuing with the normal actions. Which element should be used?

A. Conditional Block

B. Try Catch Block

C. Response Action

D. Set Values Action

ANSWER: B

Explanation:

A **Try Catch Block** is used to control error handling in an Integration Procedure. Actions that might fail can be placed in the Try portion of the block, and alternate actions can be configured in the Catch portion to process the failure, set error details, or return a controlled response.

This prevents an integration error from being handled as though the normal processing path completed successfully. A Conditional Block evaluates a Boolean condition, while a Response Action defines data returned to the caller. See Salesforce Trailhead: OmniStudio Integration Procedures.

QUESTION NO: 4

If the email address of a Contact is changed in an OmniScript, which of the following should be configured to update the contact's record in

Salesforce?

Multiple Books

A. A DataRaptor Transform that maps the new Email address to the old Email address field.

B. A DataRaptor Extract that includes the RecordId, the upsert key selected, and the new Email address.

C. A DataRaptor Load that includes the RecordId, the upsert key selected, and the new Email address.

D. A DataRaptor Transform that includes the previous email with the upsert key selected and the new Email address.

ANSWER: C

Explanation:

A DataRaptor Load that includes the RecordId, the upsert key selected, and the new Email address is the appropriate configuration because DataRaptor Load is the OmniStudio DataRaptor type designed to write data to Salesforce records. When an OmniScript captures a revised Contact email address, it must pass an identifier that lets Salesforce locate the existing Contact rather than create an unrelated record. Including the Contact record ID or configuring an appropriate upsert key supplies that matching mechanism, while mapping the new email value to the Contact Email field provides the value to save. The DataRaptor Load action can then be invoked from the OmniScript to perform the update as part of the guided interaction. This approach keeps the update declarative and aligns the OmniScript data JSON with the fields required by the target Salesforce object. Salesforce describes DataRaptors as OmniStudio tools for reading, transforming, and writing Salesforce data; the Load type specifically supports creating or updating records. See the OmniStudio DataRaptors Trailhead module and the Salesforce OmniStudio DataRaptors documentation.

QUESTION NO: 5

You are configuring a DataRaptor Load to update an existing Salesforce record. Which two approaches can identify the record that should be updated?

Choose 2 answers

A. Map the Salesforce record ID to the target object.

B. Configure a unique field mapping as an Upsert Key.

C. Set an Output JSON Path for the target object.

D. Configure a Data Node on the consuming FlexCard.

ANSWER: A B

Explanation:

A DataRaptor Load can identify an existing record by receiving the Salesforce record ID, or by using a field configured as an Upsert Key. A record ID directly identifies the target Salesforce record. An upsert key uses an incoming value to locate a matching record before determining whether to update an existing record or create a new one.

The selected approach must be mapped from the incoming JSON to the correct Salesforce field. Output JSON paths and FlexCard data nodes organize response data, but they do not identify the target record for a load operation. See Salesforce Trailhead: OmniStudio DataRaptors.

QUESTION NO: 6

How is data accessed for a Field element in a FlexCard that wants the AccountName?

A. records

B. {records}

C. {AccountName}

D. AccountName

ANSWER: C

Explanation:

`{AccountName}` is the correct way for a FlexCard Field element to access and display the value named `AccountName`. FlexCards use merge-field syntax with curly braces to resolve values from the card's data JSON. When the FlexCard data source returns a node or field called `AccountName`, placing that token in the Field element's value configuration tells OmniStudio to substitute the corresponding runtime value when the card renders.

This syntax is important because a Field element is not limited to literal text: it can bind directly to data supplied by the FlexCard's data source, including DataRaptor, Integration Procedure, or other supported sources. The token must match the relevant key in the response data available to the card. For example, if the runtime data includes an `AccountName` property, the rendered output will show that property's value, such as the account's name. Curly braces identify the content as a data reference rather than static text. Salesforce's FlexCard documentation describes using merge fields to display data returned by a card data source; see Salesforce Trailhead: OmniStudio FlexCards and Salesforce Help: FlexCards.

QUESTION NO: 7

For testing an Omniscript the ContextId is the only key in a Set Values element Before going into production, what are two possible best practices for this ContextId?

Choose 2 answers

A. Delete the Set Values element.

B. Do nothing. It will be ignored at runtime.

C. Deactivate the Set Values element.

D. Add the correct ContextId to the {Data} modal

ANSWER: A D

Explanation:

When an OmniScript is tested in the designer, a Set Values element is often used to supply a temporary ContextId so that DataRaptors, Integration Procedures, and other actions have a record context without relying on a launch mechanism. Before deployment, that hard-coded test setup must not remain as the source of the production context. One sound approach is to delete the Set Values element entirely, allowing the OmniScript to receive its ContextId from its runtime invocation and configured data inputs. This keeps the script reusable across records and prevents a test record identifier from being applied in production.

A second valid approach is to add the appropriate ContextId to the OmniScript's Data JSON through the Data modal. This preserves the expected input structure while ensuring that the value is supplied as script data rather than as a testing-only assignment in the element flow. Both approaches separate temporary designer testing from the production runtime data contract. OmniStudio's OmniScript design concepts, including data JSON and runtime data handling, are covered in Salesforce Trailhead: OmniScript Basics and the Salesforce OmniScript documentation.

QUESTION NO: 8

When an OmniScript is launched from an Action on a FlexCard, the OmniScript displays, but no Salesforce data is populated. What error

could cause this behavior?

Choose 2 answers

A. There is no active version of the OmniScript.

B. There is no active version of the DataRaptor Extract.

C. The Id Field for Actions in the FlexCard is not configured with the correct Recordid.

D. In the DataRaptor Extract Action, the Input Parameters Data Source is misspelled.

ANSWER: C D

Explanation:

"The Id Field for Actions in the FlexCard is not configured with the correct Recordid" can produce this symptom because the FlexCard action is responsible for passing the source record context into the launched OmniScript. If the action's Id Field points to an incorrect JSON node, uses the wrong field, or does not resolve to the intended Salesforce record ID, the OmniScript can still open normally but its data-retrieval logic receives an invalid or empty identifier. As a result, no matching Salesforce data is returned for display.

"In the DataRaptor Extract Action, the Input Parameters Data Source is misspelled" can likewise leave the OmniScript without populated data. A DataRaptor Extract Action obtains values from the OmniScript data JSON according to its configured input parameter data source. That JSON path must exactly match the node containing the record ID or other query value passed from the FlexCard. A misspelled path resolves to no usable input value, so the DataRaptor cannot retrieve the record data expected by the script. Correct end-to-end data mapping—from the FlexCard action into the OmniScript data JSON and then into the DataRaptor Extract input—is therefore required. See Salesforce Trailhead guidance on OmniStudio FlexCards and OmniStudio Data Tools and Integration Procedures.

QUESTION NO: 9 - (SIMULATION)

The card layout below has an Integration Procedure as a data source. The cards use the layout data source. Which JSON data structure below supports this card layout and uses best practices.

A)

"“ContactCellPhone™: "(212) 154-8562",

"ContactName": "Edward Stamos",

“AccountPhone": "2221546450",

"AccountName”: "Acme"

I

{

"ContactCellPhone™: "(650) 156-1102",

"ContactName": "Howard Jones",

"AccountPhone"™: "2221546450",

"AccountName"™ "Acme"

}

]

B)

"Contact": [

{

"CellPhone": "(212) 154-8562",

"Name": "Edward Stamos"

},

t

"CellPhone". "(650) 156-1102",

"Name": "Howard Jones"

}

ANSWER: See the explanation for the answer

Explanation:

Answer: B — use a hierarchical JSON structure with one `Account` object and a `Contact` array. This avoids repeating the same account fields for every contact and lets the card layout bind Account fields once while repeating the Contact card for each item in the array.

Option A is not a best practice because it flattens the data and duplicates `AccountName` and `AccountPhone` for each Contact record. The nested structure in B represents the parent Account and its child Contacts correctly.