PassQueen
a.k.a dumpsqueen.com

# Salesforce Certified OmniStudio Developer

## Salesforce OmniStudio-Developer

**Version Demo**

**Total Demo Questions: 25**

**Total Premium Questions: 259**

**Buy Premium PDF**

https://passqueen.com

support@passqueen.com

# Topic Break Down

| Topic | No. of Questions |
|---|---|
| Topic 1, FlexCards | 46 |
| Topic 2, OmniScripts | 70 |
| Topic 3, Data Tools | 63 |
| Topic 4, Integration Procedures | 76 |
| Topic 5, Mix Questions | 4 |
| Total | 259 |

QUESTION NO: 1

A developer is configuring a DataRaptor Load to save an Opportunity and associate it with an existing Account. The OmniScript provides the Account external ID rather than the Salesforce Account ID.

Which two configuration actions should the developer take?

Choose 2 answers

**A.** Check Is Lookup on the Opportunity AccountId mapping.

**B.** Select Account as the Lookup Object, the Account external ID as the Lookup Field, and Id as the Lookup Requested Field.

**C.** Set AccountId as the Upsert Key on the Opportunity object mapping.

**D.** Use a linked mapping from Opportunity to Account.

**ANSWER: A B**

**Explanation:**

Check **Is Lookup** on the Opportunity AccountId mapping. This indicates that the mapped target field is populated by resolving a related Salesforce record rather than by directly writing the incoming value as an ID.

The developer must also configure the lookup to use the Account object, the external ID field as the lookup field, and Id as the lookup requested field. The DataRaptor Load finds the Account whose external ID matches the OmniScript input and uses the returned Salesforce ID to populate Opportunity.AccountId.

This approach avoids hard-coding Salesforce record IDs and supports integrations that use stable external identifiers. See Trailhead: Create a DataRaptor Load.

**QUESTION NO: 2**

A developer is configuring an Integration Procedure that calls a payment service. If the payment service returns an error, the procedure must handle the failure and return a controlled response to the calling OmniScript.

Which two Integration Procedure elements should the developer use?

Choose 2 answers

**A.** Try Catch Block

**B.** Response Action

**C.** Loop Block

**D.** Calculation Matrix

**ANSWER: A B**

**Explanation:**

**Try Catch Block** is used to handle errors generated by actions within the block. The payment HTTP Action can be placed inside the Try portion of the block, while the Catch portion can set error values, perform alternate processing, or prepare a user-friendly error result.

**Response Action** defines the JSON returned by the Integration Procedure. The developer can configure it to return either the successful payment result or the controlled error response prepared by the error-handling logic. This ensures that the OmniScript receives a predictable response structure.

For Integration Procedure design guidance, see Salesforce Trailhead: OmniStudio Integration Procedures.

## QUESTION NO: 3

A developer is configuring a DataRaptor Load to Savecontract data. The developer needs to set the record type of the contact using DeveloperName.

Which two configuration actions should the developer take to set this up in the DataRaptor Load?

**A.** Check is Lookup property when mapping the fields.

**B.** Add Link to RecordType object in the Contact Object with the id field of RecordType object.

**C.** Select RecordType in the Lookup object list, DeveloperName in the Lookup Field list, and ID in the Lookup Requested Field list.

**D.** Select RecordType in the Lookup Object list. ID in the Lookup Field list, and Development in the Lookup requested Field list.

## ANSWER: A C

**Explanation:**

To populate a Contact's record type from its DeveloperName rather than from a Salesforce record ID supplied by the client, the DataRaptor Load mapping must perform a lookup. Checking *Is Lookup* on the mapping tells the DataRaptor to resolve a value from a related Salesforce object before writing the target field. The lookup configuration then uses the RecordType object as the source of that resolution. *DeveloperName* is the field used to find the intended RecordType record, because it is the API-facing developer identifier for the record type. The requested result is the RecordType record's *Id*, which can be assigned to the Contact's RecordTypeId field during the load. This lets an integration send a stable developer name while the DataRaptor retrieves the org-specific Salesforce ID needed for the save operation. Salesforce documents that record types have a DeveloperName field and that the record type ID is used to associate a record with its record type. See Salesforce RecordType object reference and Trailhead: Get Started with DataRaptors.

## QUESTION NO: 4

The OmniScript must retrieve device details stored in the Asset object and then call an external system to send troubleshooting commands via REST API to the device.

Which two OmniScript element should the developer use to configure this functionality?

**A.** DataRaptor Extract Action

**B.** REST API Action

**C.** Navigation Action

**D.** SOQL Action

**E.** HTTP Action

## ANSWER: A E

**Explanation:**

**DataRaptor Extract Action** is appropriate for retrieving the device details because a DataRaptor Extract is designed to read Salesforce data, including records from the Asset object, and map the returned fields into the OmniScript data JSON. The OmniScript can use that JSON to identify the target device and supply the values required for the subsequent integration request.

**HTTP Action** is the OmniScript element used to invoke an external HTTP or REST endpoint. It can be configured with the endpoint, HTTP method, headers, request body, and response mappings needed to send troubleshooting commands to the external device-management system. The action can also use values already present in the OmniScript data JSON, including the Asset details returned by the DataRaptor Extract Action, to dynamically construct the REST request.

This combination separates Salesforce record retrieval from the outbound service invocation, which aligns with OmniStudio's declarative data and integration capabilities. See Salesforce Trailhead guidance on OmniStudio DataRaptors and OmniScripts.

## QUESTION NO: 5

An integration procedure contains a Remote Action element that calls a method of an APEX class. The method requires two fields are input: Account id and Product Id. The integration Procedure data JSON contains the following nodes:

How should the Remote Action element be configured to pass the data correctly to the method?

**A.** Check the Send Only Additional Input checkbox, and add the Account Id and Product Id key/value pairs to Additional Input.

**B.** Set Return Only Additional Output to true, and add the following Key/Value pairs to additional input.

**C.** Check the DataRaptor Transform checkbox, and add the following Key/Value pairs to Output JSON Path:

**D.** Add the following to Send JSON Path: accountId: %Accountd% ProductId% Details Products%

## ANSWER: A

**Explanation:**

**Check the Send Only Additional Input checkbox, and add the Account Id and Product Id key/value pairs to Additional Input.** This configuration is appropriate when the Apex method needs a specific input payload containing only the two required values. In an Integration Procedure Remote Action, Additional Input maps named keys expected by the Apex action to values from the Integration Procedure's current data JSON. The values should use OmniStudio merge syntax to reference the applicable JSON paths, while the keys should match the names that the Apex method's input map is designed to read—for example, `AccountId` and `ProductId`. Enabling Send Only Additional Input ensures that the Remote Action sends this deliberately constructed input map rather than the full Integration Procedure data JSON. This makes the Apex contract explicit, limits the payload to the required fields, and avoids coupling the Apex method to unrelated nodes that may exist in the procedure's data structure. Remote Actions are intended to invoke custom Apex logic from an Integration Procedure, with input and output controlled through the element's configuration. Salesforce's OmniStudio learning materials describe Integration Procedures and their actions at Salesforce Trailhead: OmniStudio Integration Procedures. Additional OmniStudio developer guidance is available at Salesforce Industries Developer Documentation.

## QUESTION NO: 6

An OmniStudio Developer needs to retrieve data and perform complex filtering, sorting, and aggregation before displaying it in an OmniScript.

Which OmniStudio tool is the recommended best practice for preparing this data?

**A.** Data Mapper Extract Action

**B.** Integration Procedure

**C.** HTTP Action

**D.** Set Values Action

## ANSWER: B

**Explanation:**

Integration Procedure is the recommended tool because it runs server-side and is designed to orchestrate and prepare data before an OmniScript consumes it. An Integration Procedure can combine multiple actions, including Data Mapper actions, Apex remote actions, HTTP calls, conditional logic, response transformations, and calculations, into a single reusable

service. This lets the developer retrieve data and apply the required filtering, sorting, aggregation, and response shaping without placing that processing burden in the OmniScript browser session.

Using an Integration Procedure also supports a clean separation between the user-interface flow and the data-access or orchestration layer. The OmniScript can call the procedure, receive a purpose-built response JSON, and focus on presenting the result and collecting user input. This pattern improves maintainability, promotes reuse across OmniScripts and FlexCards, and can reduce client-server round trips. Salesforce identifies Integration Procedures as server-side, declarative processes for executing multiple actions and integrating data from Salesforce and external systems. See Salesforce Help: Integrate an OmniScript and Trailhead: OmniStudio Integration Procedures.

## QUESTION NO: 7

A developer needs to retrieve data from an external system that stores policy data. The external system supports REST APIs to access and update the policies. Due to the volume of the policy data and peak hours of hours of business, calls to the REST APIs sometimes take longer than expected to response.

The developer creates an Integration Procedure to retrieve the policy data for use in an OmniScript.

Given the external system's knows performance issues, which configuration should be used to implement the call to the external system?

Choose 2 answers

**A.** Set the Timeout property on the HTTP Action in the Integration Procedure

**B.** Configure a Remote action with timeout settings of 120000

**C.** Check the Chainable checkbox on the integration procedure Action in the OmniScript

**D.** Check the Chain on Step Check on the HTTP Action in the Integration Procedure

## ANSWER: A D

**Explanation:**

**Set the Timeout property on the HTTP Action in the Integration Procedure** is appropriate because the HTTP Action is the Integration Procedure element that makes the REST call to the external policy system. Its timeout configuration determines how long OmniStudio waits for the remote service to return a response. Increasing this value appropriately accommodates the known periods in which the policy API responds more slowly, while still placing an intentional upper bound on the call duration.

**Check the Chain on Step Check on the HTTP Action in the Integration Procedure** is also appropriate for a potentially long-running HTTP operation. Chaining enables the Integration Procedure to continue processing in a separate server transaction when needed, helping avoid transaction-duration constraints for operations that take longer than a typical synchronous interaction. This is particularly useful when an external service has variable response times because of high data volume or peak business activity. Together, the HTTP timeout and chaining configuration make the outbound REST integration more resilient while preserving the Integration Procedure as the service layer consumed by the OmniScript. See Salesforce's Integration Procedures documentation and Integration Procedures Trailhead module.

## QUESTION NO: 8

An OmniStudio Developer at Universal Containers needs to build and modify OmniScripts and FlexCards. The developer is unable to access the OmniStudio designers. The administrator has already confirmed the user has a Salesforce license. Which items should the administrator assign to the user to grant the required access?

**A.** The OmniStudio User permission set license (PSL) and the OmniStudio Admin permission set

**B.** The OmniStudio Admin permission set license (PSL) and the OmniStudio Admin permission set

**C.** The OmniStudio Admin permission set license (PSL) and the OmniStudio User permission set

**D.** The OmniStudio User permission set license (PSL) and the OmniStudio Developer permission set

**ANSWER: A**

**Explanation:**

The OmniStudio User permission set license (PSL) and the OmniStudio Admin permission set provide the required entitlement and authoring permissions for a user who must access the OmniStudio designers and create or modify OmniScripts and FlexCards. A permission set license makes the OmniStudio-related permissions available for assignment to the user; it does not, by itself, grant the functional permissions needed to design and administer OmniStudio assets. The OmniStudio Admin permission set supplies the elevated access used by developers and administrators to work with OmniStudio design-time tools and their related metadata.

In practice, an administrator assigns the OmniStudio User PSL to the user and then assigns the OmniStudio Admin permission set. This pairing allows the developer to open the OmniScript and FlexCard designers and manage the artifacts needed during implementation. The user's underlying Salesforce license remains important because it establishes the user account's base access, but the OmniStudio PSL and permission set are what enable the OmniStudio-specific capability described in the question. Salesforce's OmniStudio documentation covers the setup and access model for OmniStudio tools, including permission assignment considerations: Set Up OmniStudio and Permission Sets Overview.

**QUESTION NO: 9**

When launching an OmniScript from an action on a FlexCard, the OmniScript displays, but no Salesforce data is populated:

Which two errors could cause this behavior?

Choose 2 answers

Choose 2 answers

**A.** The Id Field for Actions in the FlexCard is blank.

**B.** There is no active version of the Data Raptor Extract.

**C.** There is no active version of the OmniScript

**D.** In the DataRaptor Extract Action, the Input Parameters Filter Value is misspelled.

**ANSWER: A D**

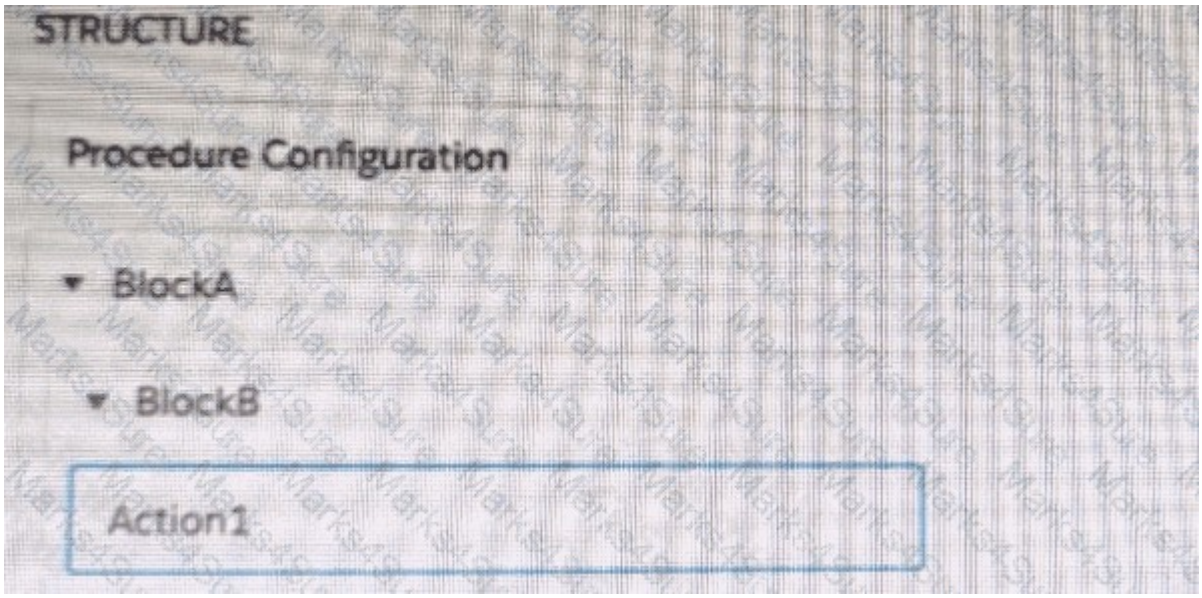
**Explanation:**

The Id Field for Actions in the FlexCard is blank can cause this behavior because the FlexCard action must pass the appropriate record identifier into the launched OmniScript. The OmniScript can open successfully even when that identifier is absent, but it has no record context to use when its DataRaptor Extract retrieves Salesforce data. Configuring the action's Id Field ensures that the relevant ID is supplied as part of the OmniScript launch context.

In the DataRaptor Extract Action, the Input Parameters Filter Value is misspelled can also produce an OmniScript that renders normally but contains no retrieved Salesforce values. A DataRaptor Extract uses its input parameters to resolve filter criteria, commonly by referencing a value passed from the FlexCard or available in the OmniScript data JSON. If that filter-value reference is misspelled, the expected input is not resolved and the extract cannot retrieve the intended record data. The launch itself remains valid, which explains why the OmniScript is displayed while its data is empty. Salesforce's OmniStudio learning material describes how FlexCards invoke actions and pass contextual data, and how DataRaptors use input values to query data: OmniStudio FlexCards and OmniStudio DataRaptors.

**QUESTION NO: 10**

Refer to theexhibit below. In this integration production structure, what Send JSON Path would be used to send the Output of the Action1 element to a Remote Action?




**A.** Action1. BlockB. Block A

**B.** Action1: BlockB. Block A

**C.** BlockA: BlockB. Action 1

**D.** BlockB:BlockB. Action1

**ANSWER: B**

**Explanation:**

**Action1: BlockB. Block A** is the appropriate Send JSON Path because Integration Procedure elements write their results into the procedure's JSON data structure according to their location in the element hierarchy. The colon identifies the action element whose output is being selected, while the portion after the colon identifies the nested block path containing that output. In the illustrated structure, *Action1* is the element that produces the data, and it is located in the nested *BlockB. Block A* context. Supplying this path to the Remote Action limits its input to the JSON produced by Action1 at that location, rather than passing the entire Integration Procedure data JSON. This is useful when a Remote Action should receive only a focused payload from a preceding element. OmniStudio Integration Procedures use JSON paths to control what data elements receive and return, allowing actions to be composed into reusable, structured server-side processes. Salesforce's Integration Procedure learning material describes the use of elements and their JSON data as the basis for orchestrating integrations, while the OmniStudio documentation provides the product context for these configuration patterns. See Salesforce Trailhead: OmniStudio Integration Procedures and Salesforce Help: OmniStudio Overview.

**QUESTION NO: 11**

An OmniStudio Developer needs an Integration Procedure that gets data from Salesforce, uses an Apex class to process the data, and then sends data to the entity that called the Integration Procedure. Which three elements provide this functionality?

**A.** Data Mapper Post Action

**B.** Response Action

**C.** Data Mapper Extract Action

**D.** Remote Action

**E.** HTTP Action

**ANSWER: B C D**

**Explanation:**

Data Mapper Extract Action, Remote Action, and Response Action together implement the required Integration Procedure flow. A Data Mapper Extract Action retrieves data from Salesforce and places the selected record and field values into the Integration Procedure's data JSON. This supplies the Salesforce data that requires further processing. A Remote Action invokes server-side Apex from the Integration Procedure, allowing a custom Apex class to receive and process the data according to the organization's business logic. Finally, a Response Action defines the data returned by the Integration Procedure to its invoking client, such as an OmniScript, FlexCard, or external caller. The Response Action can return a chosen node or transformed structure from the procedure's data JSON, ensuring the consumer receives only the intended result. This sequence separates retrieval, custom server-side processing, and response construction while keeping the orchestration within one Integration Procedure. Salesforce describes Integration Procedures as declarative server-side processes that can combine actions and return data, and documents Remote Actions as the mechanism for invoking Apex from an Integration Procedure. See Salesforce Integration Procedures documentation and Salesforce Remote Action documentation.

## QUESTION NO: 12

A developer configures a FlexCard with a Data Mapper data source that uses the params.id as an input. When the developer clicks " View Data " on the FlexCard, valid data displays. However, when the developer previews the layout, the FlexCard does not display.

What could cause this error?

Choose 2 answers

**A.** The RecordId in the Test Data Source Settings is for the wrong record type.

**B.** There is no Salesforce record for the FlexCard based on the RecordId in the layout ' s Test Data Source Settings.

**C.** The Data Node field for the FlexCard is empty.

**D.** The Attributes haven ' t been configured to pass the data to the fields.

## ANSWER: B C

**Explanation:**

There is no Salesforce record for the FlexCard based on the RecordId in the layout ' s Test Data Source Settings because layout preview obtains its context from the configured test data source. The Data Mapper receives `params.id` only when the preview supplies a usable record identifier. If the configured record does not exist or cannot be retrieved, the data source has no record context from which to produce the data needed by the card, even though manually using View Data can return a valid response.

The Data Node field for the FlexCard is empty because the Data Node identifies the node in the data-source response that the FlexCard should use as its data context. The card elements resolve merge fields and other bindings against that selected JSON structure. Configuring the appropriate node ensures that preview rendering uses the portion of the Data Mapper response containing the expected fields and records. This is particularly important where the Data Mapper output wraps the useful payload in a parent node rather than returning fields at the response root.

Salesforce Trailhead describes how FlexCards use data sources and JSON data to render card elements in Build FlexCards with OmniStudio. For additional platform guidance on configuring OmniStudio tools, see Salesforce OmniStudio documentation.

## QUESTION NO: 13

An OmniStudio Developer at Coral Cloud Resorts has built a complex FlexCard that displays reservation details. They need to validate that the FlexCard correctly processes various JSON inputs before deploying it.

Which built-in tool should the developer use to test the component with different data scenarios directly within the FlexCard Designer?

**A.** The Test Console in the OmniScript Designer

**B.** The Troubleshooting tab in IDX Workbench

**C.** The Test Data and Parameters panel in the FlexCard Designer

**D.** The Data Mapper Turbo Extract preview tab

**ANSWER: C**

**Explanation:**

The Test Data and Parameters panel in the FlexCard Designer is the appropriate built-in tool because it lets a developer supply representative input data and parameter values while working directly on the FlexCard. This supports validation of how the card interprets JSON paths, populates displayed fields, evaluates conditions, and renders states for the data scenarios expected in production. For a reservation-details card, the developer can use the panel to test payloads representing, for example, confirmed reservations, missing guest details, or alternate reservation types, then preview the card's resulting behavior before activation or deployment. Testing at the FlexCard level is important because the card can contain data mappings, conditional display logic, actions, and nested elements whose behavior depends on the shape and content of the input JSON. Salesforce's FlexCard documentation describes configuring test parameters and using the card preview while authoring, allowing developers to validate the design with appropriate data. See Salesforce's FlexCard Designer documentation and the FlexCards overview for details on building and testing FlexCards in OmniStudio.

## QUESTION NO: 14

An OmniStudio Developer is configuring an Integration Procedure Action in an OmniScript. The OmniScript needs a JSON response from the Integration Procedure but does not need to wait for the response for the user to proceed. Which feature should the developer enable?

**A.** Use Future

**B.** Invoke Mode Non-Blocking

**C.** Invoke Mode Fire and Forget

**D.** Toast Completion

**ANSWER: B**

**Explanation:**

Invoke Mode Non-Blocking is the appropriate setting because it lets the OmniScript start the Integration Procedure without holding up the user's progression through the script. The action runs asynchronously from the user interaction, so the OmniScript can continue rendering subsequent steps and collecting input while the Integration Procedure completes. Unlike a request that is simply sent without any result handling, non-blocking invocation is intended for cases where the Integration Procedure's JSON output is still needed by the OmniScript after processing finishes. The returned data can then be used by later elements or logic when it becomes available, while the user is not forced to wait at the point where the action was invoked. This mode is especially useful for longer-running integrations, such as retrieving data from an external system or executing multiple server-side operations, where preserving a responsive guided experience is important. OmniStudio Integration Procedures are designed to orchestrate server-side actions and return structured data for use by OmniScripts and other OmniStudio components. See Salesforce Trailhead's overview of Integration Procedures and the Salesforce OmniStudio Developer Guide for OmniStudio implementation guidance.

## QUESTION NO: 15

Refer to the exhibit below. What JSON code correctly represents the step in the OmniScript Structure panel shown?


A)

```
"Step1": {
        "Block1": {
                "Text1": "Text",
                "Telephone1": "1234567890"
        },
        "Block2": {
                "Checkbox1": false,
                "Block3": {
                        "Multi-select1": "Value A;Value B"
                }
        }
}
```

B)

```
"Step1": {
        "Block1": {
                "Text1": "Text",
                "Telephone1": "1234567890",
                "Block2": {
                        "Checkbox1": false
                }
        },
        "Block3": {
                "Multi-select1": "Value A;Value B"
        }
}
```

C)

```
"Step1":  {
        "Block1": {
                "Text1": "Text "
        },
        "Block2": {
                "Telephone1": "1234567890 ",
                "Checkbox1": false,
                "Block3": {
                        "Multi - select1": "Value A;Value B "
```

**A.** Option

**B.** Option

**C.** Option

**ANSWER: B**

**Explanation:**

The JSON represented by the second displayed *Option* correctly models the OmniScript element shown in the Structure panel. OmniScript uses a hierarchical JSON definition: each element is represented with its configured type and properties, while child elements are nested under the parent in the appropriate element collection. Therefore, the correct representation must preserve the structure-panel hierarchy as well as the configured attributes of the step, rather than merely listing element properties without the required parent-child arrangement. This hierarchical metadata is what OmniScript uses to render the guided interaction and to manage the data JSON as a user progresses through the script. Salesforce's OmniScript developer documentation describes OmniScript as a metadata-driven guided process and explains how its elements and data are structured. See Salesforce Developer Documentation: OmniScript Overview and Salesforce Help: OmniScript.

## QUESTION NO: 16

A developer writes an OmniScript that includes a DataRaptor that updates the Account statue based on information provided from the OmniScript. The information must be updated only if the Account record already exists. Otherwise, a new account must be created.

How should the developer accomplish this task?

**A.** Populate the Lookup object and Lookup fields

**B.** Select the Upsert Key and Is Required for Upsert checkboxes on the Account Id field

**C.** Check the Upsert key checkbox on the Account Status field

**D.** Check Overwrite Target for all Null input checkbox on the Account id field

**ANSWER: B**

**Explanation:**

Selecting the Upsert Key and Is Required for Upsert checkboxes on the Account Id field configures the DataRaptor Load to perform an upsert operation using the record identifier. When the OmniScript supplies an Account Id that matches an existing Salesforce Account, the DataRaptor updates that record with the mapped input values, such as the Account Status value. When an Account Id is not supplied, the DataRaptor can create a new Account instead of attempting to update an unknown record. The Upsert Key identifies the field used to find an existing target record, while Is Required for Upsert ensures that the field is treated as the required matching value when the DataRaptor performs the update path. This is the appropriate pattern when an OmniScript can receive either an existing record context or data for a new record. The field mappings must also include the Account fields that should be written during the load operation. Salesforce describes DataRaptor Load as the OmniStudio mechanism for inserting, updating, and upserting Salesforce records; its mapping configuration supports defining an upsert key for this purpose. See Salesforce Trailhead: Get Started with DataRaptors and Salesforce Help: DataRaptor Load.

## QUESTION NO: 17

Ursa Major Solar needs to embed an OmniScript for external users on a third-party website, not hosted on Salesforce. To achieve this, the OmniStudio Developer must use a method that allows prefilling the OmniScript with data passed in the URL.

Which configuration should the developer choose?

**A.** Generate a URL for the OmniScript and append parameters using the c__ namespace prefix.

**B.** Configure a CORS policy and use a standard Lightning Out application.

**C.** Use a Vlocity Lightning web component configured for off-platform display.

**D.** Embed the OmniScript within an iFrame that points to an Experience Cloud page.

**ANSWER: C**

**Explanation:**

Use a Vlocity Lightning web component configured for off-platform display. This refers to OmniOut, the OmniStudio capability designed to render an OmniScript outside Salesforce, such as on a company's independently hosted website. OmniOut packages the OmniScript as a deployable web component and provides the required runtime resources so that the external site can host the script without being a Salesforce page or Experience Cloud site.

For this use case, OmniOut also supports supplying initial context through URL query parameters. The external site can construct the URL used to load the component and include values needed to prepopulate the script. OmniScript recognizes custom URL parameters that use the `c__` prefix, enabling the script to receive caller-provided values as it initializes. This approach directly satisfies both requirements: true off-platform hosting and URL-driven prefill, while retaining the OmniScript's declarative behavior and integration capabilities.

See Salesforce's OmniOut documentation and the Salesforce Trailhead OmniOut resources for implementation guidance.

## QUESTION NO: 18

A developer needs to configure conditional behavior for an OmniScript input element. The element must be hidden when a customer selects Paperless Billing, and it must become required when the customer does not select Paperless Billing.

Which two Conditional View behaviors should the developer configure?

Choose 2 answers

**A.** Hide element if true

**B.** Set element to require if true

**C.** Disable read only if true

**D.** Set element to optional if false

**ANSWER: A B**

**Explanation:**

**Hide element if true** can be configured with a condition that evaluates whether Paperless Billing is selected. When the condition is true, OmniScript hides the input element from the user.

**Set element to require if true** can be configured using the inverse condition, so that the element is required only when Paperless Billing is not selected. This applies validation based on the user choice while allowing the element to remain optional when the paperless option is selected.

Conditional View controls visibility and validation behavior declaratively from values in the OmniScript data JSON. See Trailhead: OmniScripts.

## QUESTION NO: 19

A developer must send order data from an OmniScript to an external REST service. The service requires a POST request and returns a response that must be mapped into the OmniScript data JSON.

Which DataRaptor type should the developer use?

**A.** DataRaptor Extract

**B.** DataRaptor Load

**C.** DataRaptor Post

**D.** DataRaptor Turbo Extract

**ANSWER: C**

**Explanation:**

**DataRaptor Post** is designed to send data to an external REST service by using an HTTP POST request. The developer can map OmniScript data into the outbound request and configure response mappings so the service response is available in the OmniScript data JSON.

DataRaptor Extract retrieves Salesforce data, DataRaptor Load writes Salesforce records, and DataRaptor Transform reshapes data. A DataRaptor Post is the appropriate declarative tool when the requirement is to post a mapped payload to an external endpoint. See Salesforce Trailhead: OmniStudio DataRaptors for an overview of DataRaptor types.

## QUESTION NO: 20

In an Integration Procedure, a developer needs to perform a multi-step calculation on every element of an array.

Based on best practices, what two methods are recommended?

Choose 2 answers

**A.** Use a Set Values Element inside a Loop Block.

**B.** Use a Decision Matrix Action to call a Decision Matrix.

**C.** Use a List Action to merge the array elements together.

**D.** Use an Expression Set Action to call an Expression Set.

**ANSWER: A D**

**Explanation:**

Use a Set Values Element inside a Loop Block is recommended when the Integration Procedure must process each array item individually and apply calculations in sequence. A Loop Block iterates over the array, establishes the current item context, and allows Set Values to create or update values for that item. This approach keeps the calculation logic declarative, visible in the Integration Procedure, and appropriate for transformations that depend on values from each element or on values calculated in earlier steps of the loop.

Use an Expression Set Action to call an Expression Set is also recommended for more complex or reusable calculation logic. Expression Sets are designed to model ordered expressions and calculations declaratively. Calling one from an Integration Procedure separates calculation rules from orchestration logic, improves maintainability, and supports reuse when the same calculations are needed by other OmniStudio components. This is especially useful when the calculation has several dependent formula steps that would otherwise make the Integration Procedure difficult to maintain. Salesforce describes Integration Procedures as declarative server-side processes and Expression Sets as tools for defining calculation logic: Salesforce Integration Procedures documentation and Salesforce Expression Sets documentation.

## QUESTION NO: 21

A developer configures a Flexcard with a DataRaptor data source that uses the params.id as an. When the developer clicks Views Data on the FlexCard, valid data displays. However, when the developer previews the layout, the FlexCard does not display. What could cause this error?

Choose 2 answers

**A.** The Data Node field for the FlexCard is empty.

**B.** The Record Id in the Test Data Source settings ins for the wrong record type.

**C.** The attribute hasn't been configured to pass the data to the fields.

**D.** There is not Salesforce record for the FlexCard based on the Record Id in the layout's Test Data Source Settings.

**ANSWER: A D**

**Explanation:**

The Data Node field for the FlexCard is empty and the absence of a Salesforce record matching the Record Id in the layout's Test Data Source Settings can both cause this behavior. A FlexCard data source can return a valid response in the View Data action because that action tests the configured DataRaptor request and its input parameters directly. Layout preview, however, uses the FlexCard's test context and must identify the node in the returned JSON that contains the data the card should bind to. The Data Node setting establishes that binding context. If it is blank when the returned payload is structured beneath a node, the previewed card may have no data context from which to render its fields.

Preview also depends on the Record Id configured under the layout's Test Data Source Settings. In a DataRaptor configuration that receives `params.id`, this test record ID supplies the value used during preview. The ID must resolve to an existing Salesforce record that the running user can access; otherwise, the DataRaptor can return no usable record data and the FlexCard has nothing to display. Salesforce's FlexCard training explains data-source configuration and testing behavior in Prepare Data for FlexCards and preview configuration in Build a FlexCard.

## QUESTION NO: 22

A developer creates a FlexCard with five state elements. For of the stateshave a condition. To test the FlexCard, the developer previews it using sample data that causes two of the states to have true conditions.

In this scenario, how will the developer know which state will display?

**A.** The first state with true conditions sequence closest to the top of the FlexCard canvas will display.

**B.** The first state with true nested condition, regardless of sequence in the FlexCard canvas, will display

**C.** The state sequenced first in the FlexCard canvas will display.

**D.** The first state with a true AND condition, regardless of sequence in the FlexCard canvas, will display.

**ANSWER: A**

**Explanation:**

The first state with true conditions sequence closest to the top of the FlexCard canvas will display. FlexCard states are evaluated in the order in which they appear in the FlexCard canvas, starting at the top. When the runtime evaluates a state whose condition resolves to true for the current data, it uses that state and does not continue searching for another matching state. Therefore, when the sample data makes two state conditions true, the developer can determine the displayed state by locating which of those two matching states is positioned first in the state sequence. This ordering makes it possible to define more specific states before broader or fallback states, ensuring that the intended presentation is selected when multiple conditions could match the same record or data payload. State conditions can use values from the FlexCard data JSON, so previewing with representative sample data is an effective way to confirm both the condition results and the configured state precedence. Salesforce's FlexCard guidance describes states as conditional views and emphasizes their configured order in determining which view is rendered. See Salesforce Help: FlexCard States and Trailhead: OmniStudio FlexCards.

## QUESTION NO: 23

An OmniScript displays data from an API using Integration Procedure, but some of the data is missing.

Which two configuration errors could cause this? Choose 2 answers

**A.** The element name for the missing data does not match the JSON node key in the Integration Procedure Response.

**B.** The Integration Procedure Preview Input Parameters do not match the JSON sent from the OmniScript.

**C.** The JSOW sent from the Integration Procedure Action does not match any of the Original Input for the Integration Procedure

**D.** The missing data is trimmed in the Integration Procedure Action Response JSON Path.

**ANSWER: A D**

**Explanation:**

The element name for the missing data does not match the JSON node key in the Integration Procedure Response is a valid cause because OmniScript data binding relies on the response JSON structure and its node names. An element configured to read a particular node cannot populate when the Integration Procedure returns that value under a different key or location. The response must therefore expose the expected JSON node name where the OmniScript element is configured to find it.

The missing data is trimmed in the Integration Procedure Action Response JSON Path is also a valid cause. The Response JSON Path controls the portion of an Integration Procedure's output that is made available to its caller. If that path selects a nested subset that excludes a required node, the Integration Procedure can successfully run while the OmniScript receives only the reduced payload. Configuring the response path to include the needed portion of the output, along with aligning the returned node names to the OmniScript's element configuration, ensures the data is available for display. Salesforce describes Integration Procedures as server-side processes that retrieve, manipulate, and return data for OmniStudio experiences in its Integration Procedures Trailhead module. See also the Salesforce Integration Procedures documentation.

**QUESTION NO: 24**

A developer is creating a Flex Card for a new Community page. The FlexCard will display case information along with action to close the case and update the case. And it will be styled using the Community's theme.

What must be developer do to configure the FlexCard for deployment in a community?

**A.** Add the FlexCard's API name to FlexCard Player component

**B.** Set the Target property in publishing Options to the Community page''

**C.** Configure the Component visibility in the custom Component.

**D.** Set the Developer property in Card Configuration to ''Community''

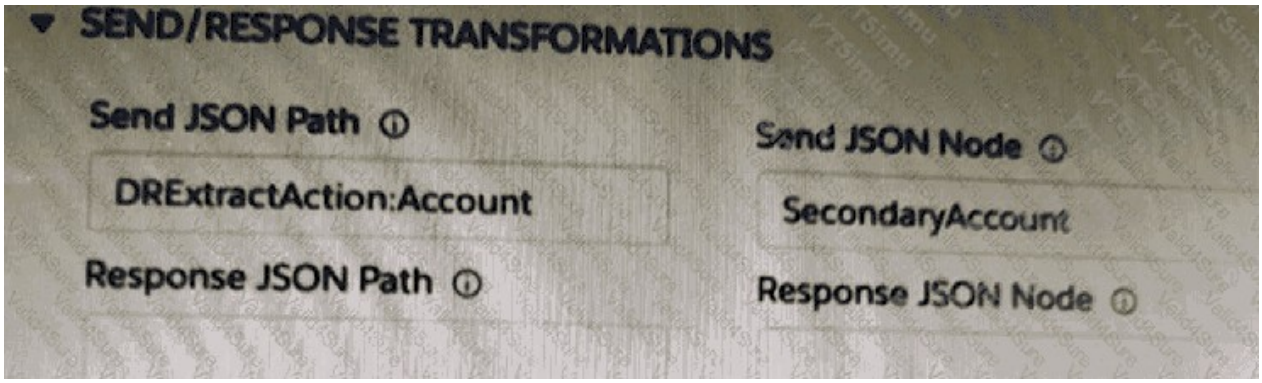
**ANSWER: B**

**Explanation:**

Set the Target property in publishing Options to the Community page'' is correct because a FlexCard's publishing configuration determines the Salesforce experience in which the generated component is exposed for placement. Selecting the Community/Experience Cloud target publishes the FlexCard for use in the community's page-building environment, where an administrator can add it to the appropriate page. Once placed there, the FlexCard renders in the context of that community and inherits the applicable Experience Cloud theme and styling. This setting is specifically about making the FlexCard available to the intended host experience; the FlexCard can then use its configured data source to show case information and its configured actions to support case updates or closure. Publishing is therefore the required deployment-oriented configuration step before the component can be selected and added to the community page. Salesforce's FlexCards learning material describes publishing FlexCards for use as Lightning components, while Experience Cloud guidance describes adding supported components through the site builder. See Salesforce Trailhead: Build Flexible User Interfaces with FlexCards and Salesforce Help: Experience Cloud.

**QUESTION NO: 25**

â€¦â€¦ configure Additional input to send exactly the same data? Assume that the develop checkedSend Only Additional input.




A)


B)


C)


D)


**A.** Option A

**B.** Option B

**C.** Option C

**D.** Option D

**ANSWER: D**

**Explanation:**

Option D is correct because, when *Send Only Additional Input* is selected, the action sends the data defined in the Additional Input section rather than automatically including the action's regular input JSON. Therefore, Additional Input must explicitly reconstruct the required payload. To send exactly the same values that are present in Input Data, define the same keys in Additional Input and map each value to its corresponding Input Data value, such as `{{InputData.key}}`. This preserves both the expected field names and their runtime values while ensuring the request contains only the Additional Input payload.

This configuration is useful when an Integration Procedure, DataRaptor, or other OmniStudio action must receive a controlled payload, but that payload is intended to mirror data already available in the element's Input Data. The merge expressions are evaluated at runtime, so each Additional Input entry resolves from the source data rather than being a hard-coded value. Salesforce's OmniStudio documentation describes how actions use input mappings and merge fields to

construct runtime request data. See Salesforce OmniStudio documentation and Trailhead: OmniStudio Integration Procedures.