FlexCards

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

A CARD NEEDS THE RIGHT CONTEXT

Context Record ID → **Source** Data JSON → **Binding** Fields and states → **Action** Guided task

01 FlexCards

Data sources and record context

- **FlexCards** present contextual data and actions. Use a DataRaptor for Salesforce data mapping, an Integration Procedure for orchestration, or another supported data source that fits the retrieval requirement.
- **View Data** reveals the returned JSON. Field bindings must match its keys and nesting; a Salesforce field API name is not automatically the output JSON name.
- **Input maps** pass context such as recordId to the data source. Test with a real, accessible record of the expected object and compare preview parameters with runtime values.
- **Data Node** selects the relevant returned node. Keep record collections as arrays; selecting the first record changes a collection into a single-record context.

States, blocks and tables

- **States** provide alternative card presentations. State order matters: the first matching conditional state takes precedence. Put more specific conditions before broad matches and provide a suitable default.
- **Collapsible blocks** let users expand optional detail while keeping summary information visible. Use conditional rendering when visibility depends on data rather than user expansion.
- **Repeat Records** creates a card instance for each record. For one Datatable containing the whole collection, avoid repeating the outer card and supply the table with the collection.

Child cards and flyouts

- **Child FlexCards** encapsulate reusable presentation. Define the data node and attributes deliberately so the child receives the intended current record or full records collection.
- **Parent data**, passed attributes and URL parameters are different contexts. Match the child binding to how the value was supplied; test it in the parent as well as in isolation.
- **Flyouts** reveal detail or a guided task without crowding the card. They can host a child FlexCard or OmniScript; backend data tools supply data rather than act as visual flyout content.

FlexCards / OmniScripts

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

GUIDED INTERACTION

Input User values → **JSON** Named nodes → **Action** Service call → **Response** UI update

Actions and channel delivery

- **OmniScript actions** launch guided processes from a card. Pass the correct record identifier so the script can retrieve Salesforce data for the selected customer or case.
- **Reusable OmniStudio Actions** centralize a shared launch configuration used by multiple cards. Keep context mappings consistent across callers.
- **Activation and publishing** prepare the generated component for use on supported Lightning or Experience Cloud pages. Configure the appropriate target and verify the page under its actual theme and user permissions.

02 OmniScripts

Steps and the data JSON

- **OmniScripts** guide users through a business process. Steps organize screens; input elements collect values; action elements retrieve, transform or save data.
- **Element names** form JSON keys. Blocks create nesting and repeatable blocks create arrays of similarly shaped objects. Match response structure to element names and group hierarchy.
- **ContextId** carries the launching record context. Use the exact key supplied by the launcher and remove hardcoded testing IDs before deployment.

Merge fields and formulas

- **Merge fields** use percent delimiters: %Customer:Name% reads a nested value. Colons traverse objects; the full path must match the actual data JSON.
- **Array selection** uses a pipe and one-based index: %Orders|1:Id% selects the first order. Within a repeatable block, |n can refer to the current repeated item.
- **Set Values** assigns, copies or derives JSON values at an execution point. A **Formula element** is appropriate for an immediate calculation based on user input in the current step, including supported aggregates such as COUNTIF.
- **Formula support** depends on the property and tool. CONCAT joins text; DATEDIFF calculates an interval. Check argument types and date units, and place expressions in formula-capable value fields.

Lookup, Type Ahead and editing

- **Lookup** supports selecting a Salesforce record and returning configured values. Use a **DataRaptor Extract Action** when retrieving a broader set of fields or a shaped payload.
- **Type Ahead Blocks** combine search suggestions with fields populated from the selected result. Verify the search key, selection mapping, data settings and nested element names when suggestions work but detail fields stay blank.

OmniScripts

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

GUIDED INTERACTION

Input User values → **JSON** Named nodes → **Action** Service call → **Response** UI update

- **Repeat Blocks** capture repeated groups. **Edit Blocks** provide an add, edit and delete interface for a collection; configure supported persistence actions when changes must reach Salesforce.
- **Text Blocks** display formatted, read-only content with merge fields. Text Area and Rich Text Area inputs collect user-entered text.

Conditions, reuse and navigation

- **Conditional View** can control visibility, requiredness or read-only behavior. Making an element visible and making it required are separate decisions; test both true and false branches.
- **Child OmniScripts** reuse a flow within a parent. Verify the parent-to-child data contract, including passDataJSON where applicable, and avoid conflicting element names in the combined structure.
- **Cancel Options** configure cancellation for the script as a whole. Define the destination and test from every step; a per-step navigation action adds separate configuration to maintain.
- **Activation** selects the callable script version. Preview both the child alone and the complete parent flow because inherited data and execution order differ.

Actions, documents and responses

- **Integration Procedure Actions** invoke server-side orchestration across Salesforce and external systems. DataRaptor actions handle Salesforce reads or writes; HTTP and Remote Actions address external services or custom Apex behavior.
- **Default invoke mode** blocks the UI while an action runs. **Non-Blocking** allows interaction to continue and applies the later response. Ensure downstream elements do not consume the value before it arrives.
- **Fire and Forget** runs without applying a callback response to the data JSON. **Use Future** also suits work that needs no returned payload; neither is a substitute for a later UI data response.
- **Document prefill** maps data into template fields, commonly through a DataRaptor Transform. A **DocuSign Envelope Action** prepares and sends an envelope to recipients; embedded signature capture is a separate interaction.

Integration Procedures

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

SHAPE EACH SERVICE BOUNDARY

Send Select and wrap → **Execute** Data or service → **Response** Trim and reparent → **Return** Caller contract

03 Integration Procedures

Server-side orchestration

- **Integration Procedures** combine data retrieval, transformation and service calls on the server. Return the fields needed by the caller rather than exposing the entire working payload.
- **HTTP Actions** call REST services; **Remote Actions** call configured Apex classes and methods. DataRaptor actions extract, transform or load data, and subprocedures reuse orchestration.
- **Execution conditions** control whether actions run. A **Loop Block** repeats operations over an array; a **List Action** filters, sorts or merges collections rather than replacing calculation logic.
- **Set Values** prepares values before dependent actions execute. An element can succeed in isolation yet fail in a procedure if its input keys, nesting or execution order differ.

Send paths and additional input

- **Send JSON Path** selects a subtree to send. **Send JSON Node** wraps that selected data under a new root key. Paths use parent-to-child order, such as Group:Fetch:Result.
- **Additional Input** maps explicit keys to literal values, merge fields or supported expressions. **Send Only Additional Input** restricts the request to this map instead of also including other context.
- **Merge example:** map CustomerId to %LookupCustomer:Id% to pass a value from an earlier action. In an HTTP URL, %SetValues:Date% resolves the Date field under SetValues.
- **Equivalent payloads** must preserve keys, values and nesting. Selecting FetchAccount:Account and wrapping it as Customer can be expressed by mapping Customer to %FetchAccount:Account%.

Response shaping and debug nodes

- **Response Actions** define what the caller receives. Successful extraction in the execution log does not ensure that the selected response includes the extracted data.
- **Response JSON Path** trims returned data to a subtree. **Response JSON Node** reparents it for the caller. Trimming too deeply or using the wrong destination name can leave UI fields blank.
- **Additional Output** provides explicit returned keys and supports formulas in applicable properties. Restrict the final response to the intended contract.
- **Preview** shows each element result and its debug details. Inspect an action's named Debug node for request configuration and input, then inspect its result node and the final response separately.

Integration Procedures / DataRaptors

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

CHOOSE A DATA OPERATION

- **Extract** — Read Salesforce
- **Turbo** — Simple read
- **Transform** — Reshape data
- **Load** — Write Salesforce

Latency, limits and failure handling

- **HTTP timeouts** must suit the service and supported callout limits. A longer timeout cannot fix missing input, bad authentication or excessive server CPU work.
- **Chainable procedures** split work across transactions when configured appropriately. Caller chainability and step boundaries such as Chain On Step must align; changing invoke mode alone does not split server work.
- **Caching** helps frequently read, slowly changing data. Include the relevant input context in cache design and test stale-data behavior after updates.
- **Error handling** should return useful failure information. Plan transaction boundaries around callouts and writes; test partial failures, retry safety and rollback requirements.

04 DataRaptors and Data Mappers

Choose the mapping tool

- **DataRaptors**, also called **Data Mappers**, separate data access and mapping from presentation. **Extract** reads Salesforce data, **Load** writes it, and **Transform** reshapes supplied data without querying Salesforce.
- **Turbo Extract** provides a simpler, faster path for reading one base object and supported related fields when complex output remapping or formulas are unnecessary.
- **Standard Extract** supports richer extraction steps, filters, formulas and output mappings. Retrieve only needed fields and use relationship field traversal when it avoids unnecessary extraction steps.
- **Transform** can flatten nested data, rename keys, create arrays and prepare supported document or XML output. Confirm both the output structure and the configured output format.

Extract paths, filters and types

- **Extract JSON Path** organizes extracted records internally; **Output JSON Path** defines the caller-facing structure. Match the consumer's expected object and array shapes rather than mirroring every source object.
- **Filters** consume correctly named input parameters. Apply a **LIMIT** in the extraction definition when only one result is needed, with suitable ordering when the chosen record matters.
- **Output Data Type** controls serialization. An integer and a quoted numeric string are different API values; configure the required numeric type rather than relying on the receiver to convert it.
- **Preview inputs** should reproduce the calling action's actual payload. An Extract that works with a manually supplied ID can still fail when the runtime filter receives a missing or incorrectly named value.

DataRaptors / Calculations

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

CHOOSE A DATA OPERATION

- **Extract** — Read Salesforce
- **Turbo** — Simple read
- **Transform** — Reshape data
- **Load** — Write Salesforce

Load, upsert and relationships

- **Load mappings** connect Input JSON Paths to writable Salesforce fields. Required fields must be present at the mapped paths; a similarly named value elsewhere in the payload does not satisfy the mapping.
- **Upsert keys** identify an existing record for update or allow insertion when no match exists. Use stable, appropriately unique identifiers. **Is Required for Upsert** skips a record when a field marked with that setting has no data.
- **Linked mappings** connect records handled within the same Load, using a created record's ID in a related object. Object sequence matters when later writes depend on earlier IDs.
- **Lookup mappings** resolve an existing related record. To set RecordTypeId from DeveloperName, look up RecordType by DeveloperName, request Id, and constrain the object type to avoid ambiguity.

Formulas and shape validation

- **Formula results** must be mapped into the desired output. CONCAT, date functions and arithmetic derive values; an Output JSON Path names a location rather than executing arbitrary formula text.
- **List output** must be a JSON array even when there is one item. Use supported list construction or a List output type, and verify empty, single-item and multiple-item payloads.
- **Expected input and output** are a contract. Copy representative runtime JSON into the designer instead of manually reconstructing a different hierarchy.
- **Partial writes** often indicate missing mappings, read-only fields, access restrictions or validation errors. Inspect the Load result for each object and field rather than assuming success from one updated field.

05 Calculation and decision tools

Procedures and expression sets

- **Calculation Procedures** execute ordered calculation and matrix-lookup steps. **Expression sets** also model reusable decision and calculation logic using their own supported elements and functions.
- **Conditional steps** choose a calculation branch from input values. Keep conditions mutually exclusive where appropriate, use Boolean-compatible expressions and test each branch.
- **Per-item calculation** derives a value for each input row. **Aggregation** summarizes across rows, such as SUM of BasePrice; a row-level formula and a collection total serve different scopes.
- **Output variables** expose results to later stages and callers. Make a calculated value available in the procedure or expression set output when a downstream aggregation depends on that output.

Calculations / Troubleshooting

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

FROM INPUTS TO DECISIONS

Inputs Typed values → **Matrix** Lookup rules → **Calculation** Per item → **Aggregate** Collection total

Matrices, versions and effective dates

- **Calculation or decision matrices** map combinations of input criteria to outputs. A procedure or expression set can use a matrix lookup instead of embedding a large set of repeated conditions.
- **Input and output types** affect comparisons and arithmetic. Test exact values, ranges, boundaries, multiple potential matches and missing matches using the matrix's configured matching behavior.
- **Active versions and effective dates** govern which calculation configuration is eligible. Include the evaluation date and time in tests; the highest visible version number alone is insufficient.
- **Row-versioned matrices** track row changes through versions. Confirm the selected version and its applicable rows before editing or deleting; verify historical and active behavior separately.

06 Troubleshooting and deployment

Trace the data contract

- **Debug in layers:** test the data tool, then the Integration Procedure, then the OmniScript or FlexCard, and finally the deployed page. Use the same representative record and input payload at each layer.
- **All data missing** suggests an absent context ID, unmatched input key, service failure or empty selected response. Compare the real request with the tool's successful preview input.
- **Some data missing** suggests a null source value, incorrect field binding, trimmed response or hierarchy mismatch. Inspect the precise returned node before changing the display element.
- **View Data versus layout preview** tests different context. If the source returns records but the card disappears, inspect Data Node, record context, state conditions and parent-to-child mappings.

Troubleshooting

Salesforce Certified OmniStudio Developer

KNOWLEDGE CHEATSHEET

TRACE BEFORE CHANGING CONFIGURATION

Input Actual payload → **Action** Debug result → **Output** Exact JSON path → **Runtime** User and target

Runtime access and limits

- **Action Debugger and procedure Preview** expose request, response, timing and error details. Trace the earliest failing action and compare its inputs with the immediately preceding output.
- **Apex CPU limit errors** require inspecting expensive extraction, loops, formulas and repeated actions. Trim payloads, reduce repeated queries and move suitable work across transaction boundaries.
- **Runtime permissions** can differ from an administrator's preview. Check object and field access, record sharing, callable Apex access, licenses and permission sets for the actual portal or internal user.
- **External connectivity** depends on endpoint configuration, authentication and supported Salesforce callout authorization, such as Named Credentials or Remote Site Settings. Diagnose the status and error payload before changing mappings.

Version, clone and release

- **Version** a component when evolving the same reusable definition for its existing callers. **Clone** when creating an independently maintained component with its own purpose and identity.
- **Dependency deployment** includes referenced cards, scripts, Integration Procedures, DataRaptors and supporting metadata. Activate or publish the required components and test launch paths in the target org.
- **Release checks** remove test-only Set Values and record IDs, confirm environment-specific endpoints, and retest child components, cancellation, validation and permission-sensitive operations.
- **Runtime compatibility** matters for managed-package and standard-runtime implementations. Use supported deployment tooling for the org and validate the generated components on their actual target pages.