**Configuration Standards**

1. **Object Standards**

* Choose short, descriptive names.
* Specify the object (AccountId, PolicyAssetId, ContactId, CaseId, etc.)
* From the Developer’s standpoint, use standard naming conventions for ease of

maintenance, and optimize the script for best performance.

* Use formula fields instead of adding same conditional view for multiple elements.
* Run logic on the Server where possible (Formulas in DataRaptor, Conditional Execution in

Integration Procedures)

1. **Template Standards**

* Avoid hard Coding in Vlocity templates
* Use $timeout instead of setTimeout
* Use $interval instead of setInterval
* Use $window instead of window
* Use $http instead of $.ajax
* Use $location instead of window.location or $window.location
* Use $cookies instead of document.cookie
* Standard templates can be overridden for each OS element, but it is recommended only to

modify the CSS, not recommended to mimic the elements with custom HTML/JS templates. Vlocity upgrade doesn't support the custom HTML elements.

* Put input fields in an order that makes sense for your use case. Use Block elements under

steps only if rAequired for your use case. To reduce the hierarchy reference in Apex and Angular templates.

* Remove spaces from all UI element names – This improves OS load time and also improves

the use of elements in Angular templates.

* While migrating the data packs only items that have been changed or are new should be

migrated rather than all the components picked up as dependency while creation of datapack.

1. **Field Standards**

* Never hard-code a Salesforce Id in Set Values
* ContextId is a reserved key and should be assigned from URL - avoid using it for another

inputs during runtime. Use Set Values to set other important non-context Ids.

* Minimize the amount of data your users must enter by using the contextId to prefill any

available key information (OR) Prefill the data wherever needed instead of fetching all the data one time.

* Provide informative, actionable help text and feedback in the UI to guide the User
* Reduce Conditional Views, Merge Fields, Formulas wherever possible.

1. **DataRaptor Standards**

* Trim JSON requests – Use ‘Send/Response Transformation’ to cut down JSON payload to

external services.

* Use Integration Procedures to reduce server roundtrips – Use for common Remote calls,

multiple DataRaptors, or any time you have more than one server call in between steps. Integration Procedures can also be used as “fire and forget” asynchronous transactions with no callback.

* Trim DataRaptor Extract outputs – Don’t fetch or return more data than you need.
* Reduce the Apex Remotes Actions instead go with OOB DataRaptors wherever possible.
* Trim JSON responses – Use Send/Response transformations to trim API responses, which

speeds up the application of the response across the OmniScript Data JSON

* Never hard-code a Salesforce Id in a DataRaptor mapping
* Look up the Id for Record Type
* Use a Sobject Type reference
* Use a reference to the Developer Name of the Record Type in the DataRaptor
* Reuse the Dataraptors wherever required
* Use one dataraptor Post action for loading related objects into Salesforce.
* Use Limit in Dataraptors
* Extract DataRaptor should not query more than 5 objects
* Not all JavaScript functions are supported by Transform DataRaptor

1. **Custom Label Standards**

* Choose short, descriptive names.
* Use Upper Camel case. No spaces, and avoid special characters.
* Relate the element to the purpose. For example, if an insurance script uses contact data for

three different parties - the insured, claimants, and beneficiaries - assign element names that disambiguate them: InsuredStreet, ClaimantStreet, and BeneficiaryStreet.

* Use VerbObjectDetails, and standardize the verbs you use. For REST actions, use Get and

Post as verbs. For DataRaptor actions, use Create, Read and Update as verbs.

* From the User’s standpoint, strive to maximize the clarity of the UI by choosing the right

elements and providing descriptive, unambiguous text and an understandable logical structure.

1. **Action Element Standards**

* Use the ‘Internal Notes’ property to document the purpose and change history of an

element. This is especially useful for Formula or Action Elements

* Use Remote Options or Extrapayload in Remote Actions to send OS elements instead of

reading entire inputMap

* Decomposition Relationship should have a destination product

1. **OmniScript Standards**

* Avoid multiple remote actions, use VIP instead. This will avoid multiple client-server calls
* Maintain a golden copy for each OmniScript which supports when multiple developers working on same OmniScript. It also avoids issues when multiple teams to handle the same OS handle the same OS.
* Design a skeleton of the entire OmniScript before build. This will help to identify reusable components.
* To ensure DataRaptors/Apex classes remain valid after script creation, do not rename OmniScript elements
* OmniScript can act as a parent (wrapper) that contains one or more embedded reusable OmniScripts. Reusable OmniScripts can be shared across multiple parents.
* Reusable OmniScripts adopt the script configuration of the parent script - The reusable OmniScript JSON will be embedded inside of the parent JSON and at compilation time there is no functional difference between the two.
* A reusable OmniScript cannot contain another reusable OmniScript - Only one level down is allowed (Parent and child)
* We can use some prefix in the reusable OmniScript to avoid same element names in parent and child OmniScript.
* Distribute Actions across OmniScript – If multiple server calls in between steps take too much time cumulatively, distribute the actions (if possible) to other parts of the OmniScript.
* Reduce number of OmniScript elements to make OS size is less than 15MB.
* The name of elements in omniscript should have same name as JSON node from which the value should be fetched
* Use the ‘Internal Notes’ property to document the purpose and change history of an element. This is especially useful for Formula or Action Elements.
* ContextId is a reserved key and should be assigned from URL - avoid using it for another input during runtime. Use Set Values to set other important non-context Ids.
* Standard templates can be overridden for each OS element, but it is recommended only to modify the CSS, not recommended to mimic the elements with custom HTML/JS templates. Vlocity upgrade doesn't support the custom HTML elements.

1. **Element Naming Conventions**

|  |  |
| --- | --- |
| Element Names | Choose short, descriptive names. |
| Capitalization | Use Upper Camel case.  No spaces and avoid special characters. |
| Ids | Specify the object (AccountId, PolicyAssetId, ContactId, CaseId, etc.) |
| Use Standard abbreviations | wk (week), wkly (weekly), mos (months), mnly (monthly), yr (year), yrly (yearly), hrs (hours), hh (household), bene (beneficiary), rel (relationship), |
| Same object, different purpose | Relate the element to the purpose. For example, if an insurance script uses contact data for three different parties - the insured, claimants, and beneficiaries - assign element names that disambiguate them: InsuredStreet, ClaimantStreet, and BeneficiaryStreet. |
| Actions | Use VerbObjectDetails and standardize the verbs you use. For REST actions, use Get and Post as verbs. For DataRaptor actions, use Create, Read and Update as verbs. Other recommended verbs for action element names:  **Remote actions:** Invoke  **DocuSign envelope:** DocuSend  **DocuSign signature:** DocuSign  **Email:** Send  **Calculation:** Calc  **Set Values, Set Errors:** Set  **PDF:** PDFGen |

1. **OmniScript Design Principles**

* From the Developer’s standpoint, use standard naming conventions for ease of maintenance, and optimize the script for best performance.
* From the User’s standpoint, strive to maximize the clarity of the UI by choosing the right elements and providing descriptive, unambiguous text and an understandable logical structure.
* Minimize the amount of data your users must enter by using the ContextId to prefill any available key information (OR) Prefill the data wherever needed instead of fetching all the data one time.
* Put input fields in an order that makes sense for your use case. Use Block elements under steps only if required for your use case. To reduce the hierarchy reference in Apex and Angular templates.
* Provide informative, actionable help text and feedback in the UI to guide the User
* Trim JSON requests – Use ‘Send/Response Transformation’ to cut down JSON payload to external services.

1. **OmniScript Performance Tuning (Server Side)**

* Use Integration Procedures to reduce server roundtrips – Use for common Remote calls, multiple DataRaptors, or any time you have more than one server call in between steps. Integration Procedures can also be used as “fire and forget” asynchronous transactions with no callback.
* Trim DataRaptor Extract outputs –  Don’t fetch or return more data than you need.
* Distribute Actions across OmniScript –  If multiple server calls in between steps take too much time cumulatively, distribute the actions (if possible) to other parts of the OmniScript.
* Reduce the Apex Remotes Actions instead go with OOB DataRaptors wherever possible.

1. **OmniScript Performance Tuning (Client Side)**

* Reduce Conditional Views, Merge Fields, Formulas wherever possible.
* Trim JSON responses – Use Send/Response transformations to trim API responses, which speeds up the application of the response across the OmniScript Data JSON
* Remove spaces from all UI element names –  This improves OS load time and the use of elements in Angular templates.
* Reduce number of OmniScript elements to make OS size is less than 15MB.
* Use formula fields instead of adding same conditional view for multiple elements.
* Run logic on the Server where possible (Formulas in DataRaptor, Conditional Execution in Integration Procedures)
* Use Remote Options or Extrapayload in Remote Actions to send OS elements instead of reading entire inputMap.
* The name of elements in omniscript should have same name as JSON node from which the value should be fetched.
* While migrating the data packs only items that have been changed or are new should be migrated rather than all the components picked up as dependency while creation of datapack.

1. **Exception Handling in OmniScript**

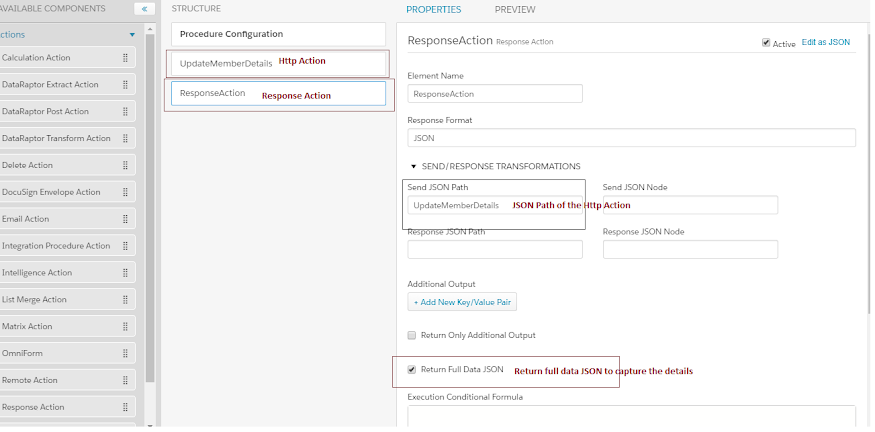
Create an Exception Logger object with following fields

* Description
* Exception Dated
* Error Code
* API Failure

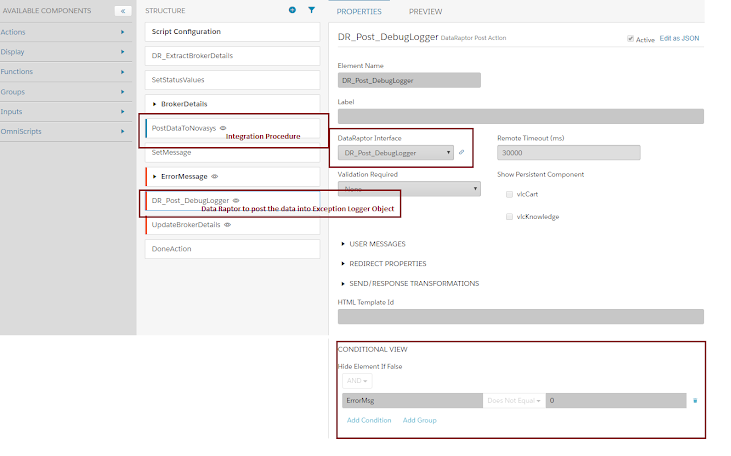
Whenever any exceptions or failure status are received, log that in the Exception Logger object for investigation.

All Integrations are handled in Integration Procedure and sending the response back to Omni Script. From Omni Script, call a data raptor to post the details in Exception Logger Object.

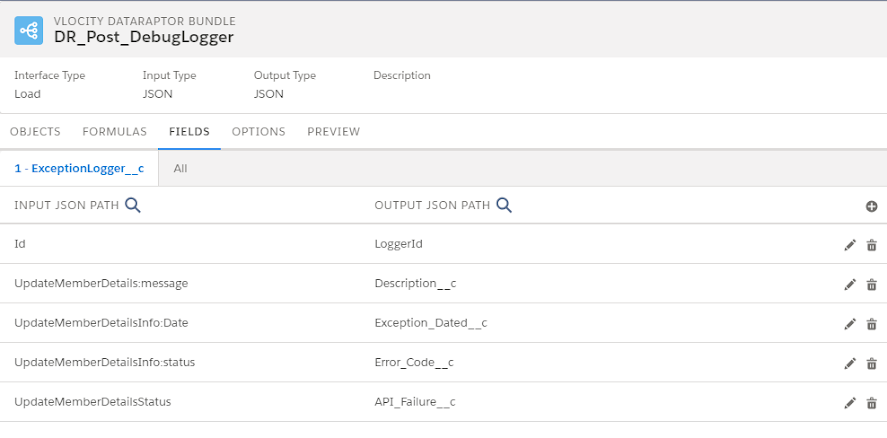
1. **Http Action and Response Action in Integration Procedure**



1. **Calling Data Raptor to Log the exception details**



1. **Data Raptor to Log details in Exception Logger**



1. **Details of the Exception Logger**

