TCPT

Integration Exception and Error Handling Mechanism

July 15, 2015

Document Control Information

Document Information

|  |  |
| --- | --- |
| Document Identification | 2.DV.AA.01884 |
| Document Name | Integration Exception and Error-Handling Mechanism |
| Project Name | TCPT |
| Client | Compassion International |
| Document Author | Bernd Kamasi, Nathan Holst |
| Document Version | 1.0 |
| Document Status | Submitted |
| Date Released | 21-Apr-2014 |
| File Name | TCPT\_Integration\_Exception\_And\_Error-Handling\_Mechanism\_2\_DV\_AA\_01884.docx |

Document Edit History

| Version | Date | Additions/Modifications | Prepared/Revised by |
| --- | --- | --- | --- |
| 1.0 | 17-Apr-2014 | Initial Draft | Bernd Kamasi |
| 1.01 | 13-Jul-2015 | Adding common classification, common error format and other clarifications | Bill Spafford |
|  |  |  |  |

Document Review/Approval History

| Date | Name | Organization/Title | Comments |
| --- | --- | --- | --- |
| < dd-mmm-yyyy > | <Name> | <Organization/Title> | <Comments> |
|  |  |  |  |
|  |  |  |  |

Distribution of Final Document

The following people are designated recipients of the final version of this document:

| Name | Organization/Title |
| --- | --- |
| Les Butler | Compassion Manager, SOA Products |
| Cathy Shipley | TCPT Technology Project Manager |
| James Qua | TCPT Technology Lead |
| Jennifer Rocks | TCPT Project Manager |

Table of Contents

[1 Introduction 5](#_Toc424746960)

[1.1 Background 5](#_Toc424746961)

[1.2 Purpose and Scope 5](#_Toc424746962)

[1.3 Application Landscape 5](#_Toc424746963)

[1.4 Error and Exception Handling Approach 5](#_Toc424746964)

[1.4.1 Error Handling Considerations 7](#_Toc424746965)

[1.4.2 Error and Exception Classification 7](#_Toc424746966)

[1.4.3 Error and Exception Categories 8](#_Toc424746967)

[1.4.4 Error Formatting 9](#_Toc424746968)

[1.4.5 Exception Handling Principles 9](#_Toc424746970)

[1.5 Assumptions 10](#_Toc424746971)

[1.6 Nomenclature 11](#_Toc424746972)

[2 Errors and Exceptions 12](#_Toc424746973)

[2.1 Source System Expected Message Handling 17](#_Toc424746974)

[2.1.1 Source System Errors Driven by Application Logic 17](#_Toc424746976)

[2.1.2 Source System Data Specific Errors 17](#_Toc424746977)

[2.1.3 Other Source System Exceptions and Handling 17](#_Toc424746978)

[2.2 Target System (Destination) Expected Message Handling 17](#_Toc424746979)

[2.2.1 Target System Errors Driven by Application Logic 17](#_Toc424746980)

[2.2.2 Target System Data Specific Errors 17](#_Toc424746981)

[2.2.3 Other Target System Exceptions and Handling 17](#_Toc424746982)

[2.2.4 Other Exceptions 18](#_Toc424746983)

[2.3 Third Party Interaction Error Handling 18](#_Toc424746984)

[2.3.1 Third Party Application Error Handling 18](#_Toc424746985)

[2.3.2 Third Party Data Error 18](#_Toc424746986)

[2.3.3 Other Third Party Exceptions and Handling 18](#_Toc424746987)

[3 Roles and Responsibilities 19](#_Toc424746988)

[3.1 Source and Destination Systems 19](#_Toc424746989)

[3.2 Third-party System 20](#_Toc424746990)

[4 Appendix 21](#_Toc424746991)

# 

# Introduction

## Background

The Transform Core Processes with Technology (TCPT) Strategic Program is comprised of three (3) strategic programs – Beyond COMPASS, USA Beyond COMPASS, and International Partner Beyond COMPASS, each with unique requirements and needs. However, all three operate within TCPT with the following objective: We will improve our global core processes and apply effective technology while replacing ineffective systems to enable the 2020 Vision.

## Purpose and Scope

The Integration Exception and Error Handling Mechanism document provides developers with guidance on exception and error handling mechanisms for system integrations implemented as part of the Compassion International (CI) Solution.

This document addresses integration exception and error handling principles, roles and responsibilities, source / target systems, and third party interaction error handling.

## Application Landscape

Please refer to TCPT Application Landscape document for complete information on the TCPT SharePoint site at: **http://tcpt.ci.org/technology/ap/02041/default.aspx**

## Error and Exception Handling Approach

Error and exception handling is categorized by integration types and enforced by certain parameters.

| Integration Type | | Definition |
| --- | --- | --- |
| Point to Point | | Point-to-point integration (also known as *one-to-one integration*) is used when a sender sends a message to a single receiver (i.e., a 1:1 relationship). |
| SOA / Middleware | Publish / Subscription Broadcast | In publish and subscribe, the client acts as a publisher and a service provider acts as a subscriber in a one-way message transfer. If a response is needed, it occurs with the same pattern, but with the parties switching roles; the service provider becomes the publisher and the client becomes the subscriber. This pattern may be used whenever the client does not need an immediate response. Key scenarios include a) when the client and the service provider may not be online at the same time, b) when more than one subscriber is interested in the request, and c) when the processing of the request is not time-sensitive. In general, with publish and subscribe, the client can publish a request and move on with additional work. A response, when needed, is generally delivered back to the client on a different connection. Message delivery can be guaranteed. |
| Request/Reply | When a Simple Object Access Protocol (SOAP) request is received, the Web service performs an action based on the request and returns a SOAP response. In many implementations, SOAP requests are similar to function calls with SOAP responses returning the results of the function call. |
| API Calls | REST | A REST resource is an abstraction of a piece of information, such as a single data record, a collection of records, or even dynamic real-time information. Each resource in a REST API is identified by a named Uniform Resource Identifier (URI), and is accessed using standard HTTP methods (HEAD, GET, POST, PATCH, DELETE). |
| SOAP | This API is for most enterprise users who are developing client applications for their organization. The enterprise Web Service Definition Language (WSDL) file is a strongly typed representation of your organization’s data. It provides information about your schema, data types, and fields to your development environment, allowing for a tighter integration between it and the Force.com Web service. This WSDL changes if custom fields or custom objects are added to, renamed, or removed from, an organization’s Salesforce configuration. If you are downloading an enterprise WSDL and you have managed packages installed in your organization, you need to take an extra step to select the version of each installed package to include in the generated WSDL.  Note the following when generating the enterprise WSDL:   * If new custom fields or objects are added to, renamed, or removed from your organization’s information, you need to regenerate the WSDL file in order to access them.   The generated WSDL contains the objects and fields in your organization, including those available in the selected versions of each installed package. If a field or object is added in a later package version, you must generate the enterprise WSDL with that package version to work with the object or field in your API integration. |

Table 1 : Integration Types

The table below defines the standard validations and verifications that may be enforced to handle error and exception handling for integrations implemented as part of the CI solution.

| Verification/Validation Method | Definition |
| --- | --- |
| Header Metadata Controls | Header Metadata contains basic information of transmitted data which may include data type, routing information, etc.  Header metadata controls allow applications to determine data routing without scanning the entire data content. |
| Input / Output Controls | The design Input (requirement) is verified by demonstrating that the derived Output (specification) is met. Input Validation is the outer defensive perimeter for Web Services, protecting the core business logic, processing and output generation.  The Output is tested directly against the input, validating that the Output meets requirements. |
| Record Count Controls | Verification that the number of records being sent as a part of the integration message reflects the number of records expected. |
| Message Content Verification | Confirms that the structure of the message being sent is correct (including fields are in the correct order). |
| Message Delivery Verification | Confirms that the message was successfully delivered to recipient. |
| Application Error | Verification or validations performed by the application’s internal logic. Examples include:   * Validation/verification if a service/user is authorized to perform a specific action. * Validation/verification of the presence of required fields. * Validation/verification of data consistencies such as field types, lengths. |
| Transport Protocol | Verification that message delivery adheres to one of the supported protocols (e.g. MQ, JMS). |
| Payload Format | Validation that message content adheres to one of the supported formats (e.g. XML, JSON). |
| Schema Validation | Verification method to check that XML and JSON documents conforms to a known schema. |
| Parameter Validation | Verification method to check that input as URI template parameters, query string parameters conform to a known parameter contract definition. |
| Service Identification | Verification that URI is supported by web services. |
| Methods | Validation of HTTP call (e.g. GET, PUT, POST) to indicate action to be performed on the identified resource is authorized for that call type |
| Security | Verification of message authentication method |

Table 2 : Validation/Verification Methods

### Error Handling Considerations

Integration errors have different error handling mechanisms built over a standard framework. While designing the Error Handling mechanism, key points listed on the table below must be considered prior to applying validation/verification methods.

| Consideration | Definition |
| --- | --- |
| Integration vs. Application error | * Integration error: error caused by application interconnectivity in a solution. * Application error: error resulting from an application’s internal logic. |
| Inbound vs. Outbound | * Inbound: data that originates from an external application, feeding into the system. * Outbound: data that originates from the system, feeding into another application. |
| Integration Technology | * Technology used to enable system integration across various applications (i.e. Neuron Enterprise Service Bus (ESB)). |
| Error Type | * Categorization of the error (system or business). |
| Retry capability | * System capability to retry calls or connections if the previous attempt was unsuccessful. |
| Error Notification Requirement | * List of technical requirement for error requirement. |
| Error screen visibility to users / support team | * Design of error notification to either user or support team (i.e. pop up screen). |

Table 3: Error Handling Considerations

### Error and Exception Classification

Exceptions occurring within an integration process fall into two classes. These are referred as the “class” of the exception throughout the document. The two “classes” of exceptions are listed in the table below and must be considered in parallel with validation / verification methods.

| Exception Classes | Definition |
| --- | --- |
| System Exceptions | Low-level exceptions encountered by the hardware or integration software (i.e. database connection failure, file input / output failure, FTP/SFTP transfer error, invalid username / password for FTP/SFTP connection, integration thread / process out of memory. |
| Business Exceptions | Exceptions associated with a type of validation step, data transformation or business rule, or data value that is violated during the execution of the integration process (i.e. invalid order number, invoice number data types, a price field with more than 2 digits for the cents value, schema violation). |

Table 4: Exception Classification

### Error and Exception Categories

Exceptions occurring within an integration process can be grouped into various categories. It is helpful if all systems follow these categories when returning an error to an integration component. If a downstream component is not able to comply with these categories, the middleware will transform the native exception information into the correct category.

| Exceptions | Definition and Handling |
| --- | --- |
| Authorization Error | **Description**: Supplied credentials are invalid or the user is not authorized to perform the requested action.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**: In a public interface no details as to the particular reason for the failure should be returned to the caller.  **Retry**: Not retryable until corrected |
| Input Validation Error | **Description**: These failures require a detailed message so that the caller may fix the problem before trying again.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**: In a public interface only published information should be included.  **Retry**: Not retryable until corrected |
| Communication Error | **Description**: Timeouts, connection failures, unexpected network errors.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**: In a public interface no details as to the particular systems and reason for the failure should be returned to the caller.  **Retry**: May be retryable if outage was a temporary condition. |
| Application Error | **Description**: This is a failure that occurs in application logic.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**:  **Retry**: May be retryable |
| Integration Error | **Description**: This is a failure that happens in the ESB and related components during message processing, workflow or composition.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**: In a public interface no details as to the particular systems and reason for the failure should be returned to the caller.  **Retry**: May be retryable |
| General System Error | **Description**: This category is for types not covered above.  **Logging**: The original exception should be logged and a unique identifier generated to be included in the error metadata.  **Exception Hiding**: In a public interface no details as to the particular systems and reason for the failure should be returned to the caller.  **Retry**: Not retryable |

Table 5: Exception Categories

### Error Formatting

All exceptions which occur in source systems or in the middleware will be transformed into an error with common metadata attributes:

| Attribute | | Definition and Handling |
| --- | --- | --- |
| Error Id | unique identifier for error instance (GUID) | |
| Error Timestamp | UTC timestamp when error occurred | |
| Error Class | “BusinessException” or “SystemException” | |
| Error Category | one of agreed-upon list of error categories from Table 5 | |
| Error Code | A code that consistently matches the error type. Suggested prefix + 4-digit code. (prefix=”SF”, “NS”, “ESB”,…) | |
| Error Message | descriptive error message | |
| Error Retryable | true or false – if known. This is to support automatic resubmission. | |
| Error Module | module or subsystem in which error occurred | |
| Error Sub Module | detailed sub module name, if applicable, in which error occurred | |
| Error Method | method name in which error occurred | |
| Error Logged in User | current username | |
| Related Record Id | context related id from application | |

Table 6 : Error Metadata

### Exception Handling Principles

The exception management tasks and activities are classified into four different categories:

* **Detection** - The process of identifying exceptions when they occur.
* **Notification** - The process of notifying the appropriate system and/or person of an exception.
* **Logging** - The process of maintaining an exception repository for analysis and reporting.
* **Recovery** - The process of accurately recovering the system state and restarting processing. Includes message resubmission and automatic retry.

The table below identifies the exception management principles for each category:

| Exception Activity | Principles |
| --- | --- |
| Detection | Processes shall follow a consistent exception trapping approach, including checks for validation, business logic, and system exceptions. |
| Processes shall capture and store identical exception information such as time of exception, source system, event location, exception type, and exception description. |
| Trapped exceptions requiring the process to discontinue should be propagated to the parent process, if applicable, for exception handling. |
| Exception handling logic shall be performed by each application. |
| Notification | Application-specific error notifications should be handled through the application, while other communication errors between source and destination are done through the middleware. |
| Exception notification types should correspond to the exception types. For example, system administrators should be notified of technical exceptions, whereas business users should be notified of business logic exceptions.  The ESB will provide a mechanism to submit a ticket to ServiceNow to be used to notify the appropriate team of system and business exceptions that require investigation.  Monitoring will detect system failures and notify support personnel. |
| Exception Logging | Exceptions shall be logged using a standard format defined in section 1.4.4 Error Formatting |
| Exception logs shall be accessible by the exception recovery managers. |
| Exception Recovery | Exceptions shall not be automatically recovered unless there is a specific business rule that requires automatic recovery. |
| The ErrorRetryable attribute will indicate, where possible, whether the operation which failed should be retried.  Auto-retry will be configured on a process-by-process basis. |
| Rejection of an entire batch file containing validation exceptions will be determined by the business requirement for that particular integration process. |
| Rollback of an entire batch file if it cannot be loaded into CI Solution applications (All or None) in its entirety will be determined by the business requirement for that particular integration. |

Table 7 : Exception Handling Principles

The exception handling principles must be considered when applying integration exceptions validation / verification method across source, destination, and third-party systems.

## Assumptions

The following assumptions are made as it pertains to integration exception and error handling:

* The purpose of this document is to provide guidance. Individual integration exception and error handling may require deviation from this guidance for specific Web Services. These deviations should be done with approval on an individual case-by-case basis.
* The native integration error and exception handling mechanisms in applications are used to the greatest extent practicable.
  + For Web Services, refer to updated TCPT service detailed design templates.
  + For Salesforce, refer to Salesforce Integration Patterns and Practices document.
  + For NetSuite, refer to NetSuite Help Center sections: Error Status Code, SuiteScript Errors, SOAP Faults, SOAP Fault Status Code.
  + For SDL, refer to SDL WorldServer exceptions and errors files.
* Legacy applications use existing Integration and error handling capabilities, which are outside the scope of this document.
* The Neuron ESB is the ESB middleware for CI Solution.

## Nomenclature

TCPT-related definitions and acronyms can be found on the TCPT SharePoint site at:

<http://technology.ci.org/docs/kb/glossary/default.aspx>.

# Errors and Exceptions

The CI Solution contains integration modules that will handle the task of transferring in-bound and out-bound data and communications. Error and exception notifications should contain specific descriptive information. Integration team members must follow integration exceptions principles, classifications, and considerations, while applying specific validation / verification methods. There are three important components of integration: source system, target system, and middleware.

Cornerstone Connect is the middleware solution and external API management component for the CI Solution. It is a centrally managed set of technology services used by CI to exchange and manage core-shared data and non-core data. The Cornerstone Connect components are described in more detail below.

All distributed systems, whether source, target or intermediary that implement REST Services for integration purposes must also support a mapping of errors to the appropriate HTTP Status Codes as defined in the Compassion REST Standard v1.2.

**Neuron ESB**

Cornerstone Connect has an ESB service, Neuron ESB. When using Neuron ESB, applications and/or services simply publish and exchange messages onto the bus, without regard to the type or number of consumers; similarly, they may subscribe to specific messages or groups of messages, without regard to the source of the messages.

Neuron has the concept of an audit database. Endpoint (flow) can be audited and called anytime for on demand audit. Exceptions are automatically audited into a standard SQL database with a status of Exception. The SQL database can be monitored by monitoring solutions such as Nagios, OpenView, or others. Neuron does not perform automated notification, but the monitoring solutions can act as the source of notification. In case a more sophisticated capability is needed, email adapter endpoints can be integrated with Neuron.

Neuron hosts Windows Communication Foundation (WCF) services framework for service-oriented applications. Neuron is also implementing Katana - a subset of Open Web Interfaces for .NET (OWIN) released by Microsoft that includes both infrastructure and functional components such as authentication components and bindings to frameworks. Katana focuses on three goals: portable, modular / flexible, and lightweight / scalable.

The illustration below depicts the general system exceptions flow from source system to target system via Neuron ESB.



Figure 1 : System Exceptions Flow from Source to Target

**Mashery**

Cornerstone Connect also uses Mashery API management for external consumption and communication. This includes in-bound communication with International Partner (IP) applications.

The CI Solution integration validation and verification methods are summarized in a matrix on the next page, categorized by connectivity type (point to point, SOA / middleware, or API calls).

**Salesforce**

Custom APEX REST Services used for integration purposes will be written to follow this guidance for exception and error handling and will support the appropriate use of HTTP Status Codes defined in the Compassion REST Standard.

In the event that “out of the box” Salesforce APIs are used and do not conform to the Compassion REST Standard, the ESB will have the responsibility to transform exception and error information into this standard format.

**Netsuite**

Custom Restlet Services used for integration purposes will be written to follow this guidance for exception and error handling and will support the appropriate use of HTTP Status Codes defined in the Compassion REST Standard.

In the event that “out of the box” Netsuite APIs are used and do not conform to the Compassion REST Standard, the ESB will have the responsibility to transform exception and error information into this standard format.

The table below specifies the different integration and exception error handling mechanisms that are implemented for the different integration types. Exception handling principles must be followed per Table 7 : Exception Handling Principles. Error considerations listed on Table 3: Error Handling Considerations must be done in parallel when reviewing the integration matrix guidelines. Error and exception details should be logged and archived for troubleshooting.

|  | Point to Point | SOA / Middleware | | API Calls | |
| --- | --- | --- | --- | --- | --- |
| Publish / Subscription Broadcast | Request / Reply | REST (As part of Request/Reply) | SOAP (As part of Request/Reply) |
| Header Metadata Control | Follow ESB Services and Messaging Standards for headers where needed | The attributes in the header should contain the topic for publication  Follow ESB Messaging Standard for headers where needed. | The attributes in the header should contain the message id and the destination end point. | Follow ESB Services and Messaging Standard for headers where needed | Follow ESB Services and Messaging Standard for headers where needed |
| Input / Output Control | Message schema should contain validation against interface definition to ensure quality | Message schema should contain validation against interface definition to ensure quality | Message schema should contain validation against interface definition to ensure quality | Validation by input parameters and schema.  Message schema should contain validation against interface definition to ensure quality | Validation by input parameters and schema.  Message schema should contain validation against interface definition to ensure quality |
| Record Count Control | NA | NA | NA | NA | NA |
| Message Content Verification | NA | NA | NA | NA | NA |
| Message Delivery Verification | Optional, need to setup destination point to verify message delivery | NA | Receiving system should verify receipt and confirmation of delivery on records to inform sending system that all data was processed. | NA | NA |
| Application Error | Connection  Timeout  Security | There could be a number of application error related to connectivity, message/schema validation, timeout issues, and security related exceptions.  All these application exceptions should be logged in the persistent storage at server side rather than showing stack trace to end users.  Any exception should be handled gracefully.  The exception can happen on the Publisher or Subscriber end | There could be a number of application error related to connectivity, message/schema validation, timeout issues, and security related exceptions.  All these application exceptions should be logged in the persistent storage at server side rather than showing stack trace to end users.  Any exception should be handled gracefully | RESTfull service uses the error codes from HTTP's status code registry,  1xx: Meta  2xx: Success  3xx: Redirection  4xx: Client-Side Error  5xx: Server-Side Error | SOAP typically delivers errors as SOAP faults.  There could be a number of application errors related to connectivity, message /schema validation, timeout issues, and security related exceptions.  All these application exceptions should be logged in the persistent storage at server side rather than showing stack trace to end users.  Any exception should be handled gracefully |
| Transport Protocol | MQ, JMS, File Transfer, WS, API, TCP, MSMQ, Peer, Named Pipe, Rabbit MQ | MQ, JMS, File Transfer, WS, API, TCP, MSMQ, Peer, Named Pipe, Rabbit MQ | All these application exceptions should be logged in the persistent storage at server side rather than showing stack trace to end users. | Only HTTP protocol is supported by RESTfull services | All these application exceptions should be logged in the persistent storage at server side rather than showing stack trace to end users. |
| Payload Format | Supported payload format by both systems | Supported payload format by publishing and subscribing systems | Supported payload format by source and target systems | Payload format supported are XML, JSON | XML |
| Schema Validation | Refer to input/output control above | Refer to input/output control above | Refer to input/output control above | Refer to input/output control above | Refer to input/output control above |
| Service Identification | NA | NA | Supported service by source and target systems | UriTemplate is the unique URI for each specific service operations. Service consumer only knows about unique URI defined in UriTemplate | URI and WS-Addressing are supported in web services |
| Methods | NA | Supported methods by publishing and subscribing systems | Optional | RESTfull supports HTTP verbs as defined in the Compassion REST Standard:  **GET** - Getting a specific representation of a resource.  **PUT** - Update a resource.  **POST** - Create a resource.  **DELETE** - Delete a resource. | Actions as defined by the developer creating the service or by definition specs; at least one action has to exist. |
| Parameters | Optional | Optional | Name, type and range of input parameters should match the published contract. | Name, type and range of input parameters should match the published contract. | Name, type and range of input parameters should match the published contract. |
| Security | Verifying the identity of the source system either via an IP address or the SSL certificate's (CN) common name or distinguished name (DN). | Optional header authentication, Microsoft AD, ESB Identity Store | TCP/IP, HTTP, WS, Microsoft AD, ESB Identity Store | REST supports HTTPS, OAuth, and Open ID Connect | WS\*, HTTPS, SAML, Microsoft AD, ESB Identity Store |

**Table 8: Integration Exception and Error Handling Guidelines**

## Source System Expected Message Handling

Source system expected message handling refers to management of error and exceptions by the source system. This includes: errors driven by application logic, data-specific errors, or anything else triggered by the source system.

### Source System Errors Driven by Application Logic

The source system notifies end users of unexpected results (i.e. timeout) driven by application logic. Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type.

### Source System Data Specific Errors

The source system has an out-of-the-box exception handling of standard features (i.e. required field, data type, etc.). Applications should be programmed to reject or truncate inputs that violate allowable input lengths / formats. Please refer to Application Error on Table 8: Integration Exception and **Error Handling Guidelines**.

### Other Source System Exceptions and Handling

The source system should have a timeout limit when no positive acknowledgement is received from the call, except for broadcast messages. Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type.

## Target System (Destination) Expected Message Handling

Target system message handling refers to management of errors and exceptions by the target system. This includes: errors driven by application logic, data specific errors, or anything else detected by the target system.

### Target System Errors Driven by Application Logic

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type.

### Target System Data Specific Errors

Please refer to Application Error on Table 8: Integration Exception and **Error Handling Guidelines**.

### Other Target System Exceptions and Handling

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type.

### Other Exceptions

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type.

## Third Party Interaction Error Handling

Third party applications are defined as systems, external to the CI solution, that integrate with the CI solution via Mashery. Some native error handling is performed by Mashery. Web-service error handling capabilities should generate safe error messages (i.e. do not contain information that can be exploited by attacker).

### Third Party Application Error Handling

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type for Third Party applications.

### Third Party Data Error

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type for Third Party applications.

### Other Third Party Exceptions and Handling

Please refer to Table 8: Integration Exception and **Error Handling Guidelines** for a complete enforcement type for Third Party applications.

# Roles and Responsibilities

Integration error troubleshooting and resolution is managed by system administrators. The table below defines the roles involved in troubleshooting and resolving integration errors.

| **Role** | **Responsibility** |
| --- | --- |
| Source system admin | Troubleshoots and leads error resolution of source system specific errors |
| ESB admin | Troubleshoots errors happening between the source and target systems |
| Target system admin | Troubleshoots and leads error resolution of target system errors |

Table 9: Roles and Responsibilities in Integration Error Handling

Note: There are exceptions where source, target, and ESB admins must collaborate to resolve errors. In certain exception scenarios, it may be required to design the error resolution process differently than what is proposed below. These guidelines can be adapted as the business and technology require it.

## Source and Destination Systems

The table below describes high-level source system error type, error resolution, and responsibility.

| **Error Type** | **Error Resolution** | **Responsibility** | **Comments** |
| --- | --- | --- | --- |
| Header metadata error | Validate that content corresponds to the metadata | Source system admin |  |
| Input / Output error | Validate expected input and output of the application flow | ESB admin |  |
| Record count error | Validate expected record against actual record count | Source system admin  ESB admin  Target system admin |  |
| Invalid content | Validate that outbound content is the correct type and is not null | Target system admin |  |
| Failed message delivery | Validate the connectivity between the source, ESB, and target systems. | ESB admin  Target system admin |  |
| Application error | Verify that user are authorized and required input parameters are supplied | Target system admin |  |
| Transport protocol error | Verify that message delivery adheres to supported protocols | Source system admin  ESB admin |  |
| Payload error | Validate that payload is in the correct format | Source system admin  ESB admin  Target system admin | Responsible party is dependent upon system performing validation of payload |
| Schema error | Validate that expected schema adheres to supported format | Source system admin  ESB admin  Target system admin | Responsible party is dependent upon system performing validation of schema |
| Service identification error | Verify URIs are known entities | ESB admin |  |
| Methods error | Verify that the HTTP call is authorized for that system | Target system admin |  |
| Parameter error | Verify that parameters conform to the published contract. | Source system admin  ESB admin  Target system admin |  |
| Security error | Verify authentication method of source and target system | Source system admin  Target system admin |  |

Table 10: Examples of Source and Destination System Errors

## Third-party System

Third party systems are defined as Web Services and applications that connect to the CI Solution through Mashery, including International Partner (IP) applications. Third party system can be source system or target systems and will be treated the same way as described in Table 10: Examples of Source and Destination System Errors

# Appendix

The Appendix contains tactical considerations for the Implementation Team to consider during the Blueprint and Realization-Build phases.

* Compassion International is finalizing the selection of solution components for the CI Solution at the time of developing this Integration Exception and Error Handling Mechanism document. The Integration Team must consider the impact that selection of a particular solution component will have on the implementation of this Integration Exception and Error Handling Mechanisms defined in this document.