# BCBSRI Outbound Claims Processing – Flow 3

Software Design Document (SDD) — Detailed Design

## Table of Contents

1. 1. General Information
2. 1.1 Executive Summary
3. 1.2 Scope
4. 1.3 Document Usage
5. 1.4 Diagrams (Placeholders)
6. 2. Technical Design
7. 2.1 Azure Function One — Copyer (Queue Trigger)
8. 2.1.1 Input Message Structure
9. 2.1.2 Processing Steps
10. 2.1.3 Outputs & Audit
11. 2.1.4 Error Handling (Function One)
12. 2.2 Azure Function Two — Processor (Queue Trigger)
13. 2.2.1 Input Message Structure
14. 2.2.2 Processing Steps
15. 2.2.3 Outputs & Audit
16. 2.2.4 Error Handling (Function Two)
17. 2.3 End-to-End Sequence
18. 3. Error Handling and Recovery
19. 3.1 Error Taxonomy and Codes
20. 3.2 Retry, DLQ, and Quarantine
21. 3.3 ERROR\_LOG and PROCESS\_LOG usage
22. 4. Business Rules
23. 5. Developer Implementation Plan (Phase A–E)
24. 5.1 Developer Checklist & Local Setup
25. 6. Monitoring & Operations
26. 7. References
27. 8. Amendment History
28. Appendix A: Snowflake DDL (PENDED\_CLAIMS\_OUTBOUND + supporting tables)
29. Appendix B: Sample JSON Payloads (Queue, Claim, API)
30. Appendix C: Error Code Reference Table

## 1. General Information

### 1.1 Executive Summary

Flow 3 automates copying and processing claim files from Azure Blob Storage into Snowflake. The flow consists of two Azure Functions: Function One (Copyer) copies incoming claim files from a source container to a staging container; Function Two (Processor) reads the staged files, filters Prime-pended claims, transforms the payloads, and ingests them into Snowflake via the Member API. The design focuses on reliability, idempotency, auditability, and operational visibility.

### 1.2 Scope

In-Scope:

- Function One (Queue-triggered): download & copy blob to staging container; create audit entry; enqueue staging message.

- Function Two (Queue-triggered): parse staged blob; filter & transform claims; call Member API to load into Snowflake; record errors and audit.

Out-of-Scope:

- Kafka ingestion and the upstream Facets system (already existing).

- Downstream Matillion transformation and SFTP transfer (handled by other flows).

### 1.3 Document Usage

This SDD is intended for developers, architects, DevOps engineers, QA, and operations responsible for implementing Flow 3.

### 1.4 Diagrams (Placeholders)

[Placeholder: Figure 1 — High Level Data Flow Diagram]

[Placeholder: Figure 2 — Sequence Diagram: Queue -> Copy -> Process -> Snowflake]

[Placeholder: Figure 3 — Use Case Diagram]

## 2. Technical Design

### 2.1 Azure Function One — Copyer (Queue Trigger)

Function One is triggered by a Storage Queue message published by Event Grid when a new blob is created in the 'claims-raw' container. It is responsible for validating the message, downloading the blob, copying it to the 'claims-staging' container, creating an audit record, and pushing a message to the staging queue for Function Two.

#### 2.1.1 Input Message Structure (Event Grid shape)

{  
 "topic": "/subscriptions/{sub}/resourceGroups/{rg}/providers/Microsoft.Storage/storageAccounts/{sa}",  
 "subject": "/blobServices/default/containers/claims-raw/blobs/2025/09/25/claimsfile.json",  
 "eventType": "Microsoft.Storage.BlobCreated",  
 "id": "event-id-123",  
 "data": {  
 "api": "PutBlob",  
 "clientRequestId": "abcd-1234",  
 "requestId": "req-5678",  
 "eTag": "0x8DDFC7792",  
 "contentType": "application/json",  
 "contentLength": 38774,  
 "url": "https://{sa}.blob.core.windows.net/claims-raw/2025/09/25/claimsfile.json"  
 }  
}

#### 2.1.2 Processing Steps

1. Deserialize the queue message and validate 'data.url'.

2. Validate blob extension (.json/.ndjson/.csv) and size thresholds (configurable).

3. Download blob content using Azure Blob SDK (BlobClient).

4. Optionally validate top-level JSON structure (array/object).

5. Copy blob stream to destination container 'claims-staging' using a deterministic path (e.g., /{yyyy}/{mm}/{dd}/{batchId}-{filename}).

6. Insert AUDIT\_CLAIM\_FILE record with TransactionStatus = 'N' and store metadata (BATCH\_ID, PROCESS\_ID).

7. Enqueue message to 'claims-staging-queue' with destination blob metadata for Function Two.

#### 2.1.3 Outputs & Audit

- Writes AUDIT\_CLAIM\_FILE row (initial TransactionStatus = 'N').

- Sends message to 'claims-staging-queue'.

- Logs operation and metrics to Application Insights and PROCESS\_LOG.

#### 2.1.4 Error Handling (Function One)

Validation errors: If the queue message is malformed or missing URL, log ERR-QMSG-001, insert ERROR\_LOG entry, and move to quarantine if appropriate.  
Blob not found: ERR-BLOB-404 — write audit entry, optionally requeue (transient) or create ticket if persistent.  
Copy failures: ERR-COPY-3001 — retry with exponential backoff; if exhausted, write to DLQ and ERROR\_LOG.  
All exceptions: capture stack trace, correlation IDs (BATCH\_ID/PROCESS\_ID), and write PROCESS\_LOG and ERROR\_LOG entries.

### 2.2 Azure Function Two — Processor (Queue Trigger)

Function Two processes staging blob messages. It downloads the staged file, iterates claims, filters 'Prime-pended' claims, transforms messages for Snowflake ingestion, calls the Member API endpoint and records outcomes.

#### 2.2.1 Input Message Structure (Staging Queue)

{  
 "topic": "...",  
 "subject": "/blobServices/default/containers/claims-staging/blobs/2025/09/25/batch-123-claimsfile.json",  
 "data": {  
 "url": "https://{sa}.blob.core.windows.net/claims-staging/2025/09/25/batch-123-claimsfile.json",  
 "contentType": "application/json"  
 }  
}

#### 2.2.2 Processing Steps

1. Deserialize the staging queue message and extract blob URL.

2. Download blob and parse JSON (array of claim objects).

3. For each claim:

a. Validate mandatory fields (claim\_id, member\_id).

b. Check business filter: status == 'PENDING' and pended\_reason contains 'Prime' (case-insensitive) or primePended flag == true.

c. If valid: generate CLAIM\_GUID (UUID), add ingestion\_timestamp, normalize diagnosis & procedure codes arrays; build claim object for Member API.

d. If invalid: insert record into ERROR\_LOG with appropriate error code and continue.

4. Group claims into batch payload and call Member API (HTTP POST) with authentication via Managed Identity or API Key from Key Vault.

5. On API success (200 OK): update AUDIT\_CLAIM\_FILE TransactionStatus='Y' and insert records into PENDED\_CLAIMS\_OUTBOUND with LOAD\_STATUS='SUCCESS'.

6. On API 4xx: treat as permanent failure for those payloads — insert into ERROR\_LOG; do not retry automatically unless requeued by operator.

7. On API 5xx / network errors: retry with exponential backoff; if exhausted, mark batch as failed, write ERROR\_LOG, push to DLQ, and create ServiceNow ticket if configured.

#### 2.2.3 Outputs & Audit

- Inserts rows into PENDED\_CLAIMS\_OUTBOUND for successful claims.

- Updates AUDIT\_CLAIM\_FILE with TransactionStatus and timestamps.

- Writes claim-level errors into ERROR\_LOG for manual remediation.

#### 2.2.4 Error Handling (Function Two)

Validation errors: ERR-CLAIM-VAL-001 — insert claim-level ERROR\_LOG, continue processing others.  
Member API 5xx: ERR-API-4003 — retry with backoff; if still failing, mark batch-level failure, write to ERROR\_LOG, and DLQ.  
Snowflake insert issues: ERR-SNOW-500 — log error detail, mark LOAD\_STATUS='FAILED' for affected rows.

### 2.3 End-to-End Sequence

1. Blob uploaded to claims-raw -> Event Grid publishes message to 'claims-notifications-queue'.  
2. Function One picks up message -> copies blob to claims-staging -> writes AUDIT\_CLAIM\_FILE (N) -> enqueues staging message.  
3. Function Two picks up staging message -> processes claims -> calls Member API -> writes PENDED\_CLAIMS\_OUTBOUND -> updates audit (Y/N).  
4. Operator investigates ERROR\_LOG entries and re-enqueues failed claims if applicable.

## 3. Error Handling and Recovery

### 3.1 Error Taxonomy and Codes

Error codes below include both a string and a numeric mapping for clarity and operational automation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Numeric Code | String Code | Category | Description | Suggested Action |
| 2001 | ERR-QMSG-001 | Queue Message | Invalid or malformed queue message | See remediation steps in 3.3 |
| 2002 | ERR-BLOB-404 | Blob | Source blob not found | See remediation steps in 3.3 |
| 2003 | ERR-BLOB-403 | Blob | Blob access denied (permissions) | See remediation steps in 3.3 |
| 3001 | ERR-COPY-3001 | Copy | Failed to copy blob to staging | See remediation steps in 3.3 |
| 1001 | ERR-CLAIM-VAL-001 | Validation | Claim missing mandatory fields or malformed | See remediation steps in 3.3 |
| 4003 | ERR-API-4003 | API | Member API returned 5xx/server error | See remediation steps in 3.3 |
| 5001 | ERR-SNOW-500 | Snowflake | Insert/load failure in Snowflake | See remediation steps in 3.3 |
| 6001 | ERR-DLQ-6001 | DLQ | Message moved to DLQ after exceeding retries | See remediation steps in 3.3 |
| 7001 | ERR-AUDIT-100 | Audit | Failed to write audit record | See remediation steps in 3.3 |

### 3.2 Retry, DLQ, and Quarantine

Retry policies: Transient errors are retried with exponential backoff (e.g., 2s, 4s, 8s). Configure host.json maxDequeueCount=5. After max retries, messages go to DLQ where a DLQ handler processes them.

Quarantine: Files failing schema-wide validation are moved to 'claims-quarantine' container and flagged in AUDIT\_CLAIM\_FILE for manual review.

### 3.3 ERROR\_LOG and PROCESS\_LOG usage

PROCESS\_LOG: function-level operational logs (INFO/WARN/ERROR). ERROR\_LOG: claim-level or batch-level failure records with ERROR\_CODE, ERROR\_DETAILS and retry metadata. Both tables are used for alerting and ServiceNow ticket creation.

## 4. Business Rules

- Only claims with status == 'PENDING' are processed by Function Two.  
- PendedReason should contain 'Prime' or primePended flag true to include claim.  
- Duplicate claims are detected by CLAIM\_GUID/CLAIM\_ID and skipped (logged as INFO).

## 5. Developer Implementation Plan (Phase A–E)

Phase A — Scaffolding:

- Create solution with two Azure Function projects (FunctionOne.Copyer, FunctionTwo.Processor).

- Add shared library: Models (Claim, AuditEntry), Services (BlobService, QueueService, MemberApiClient), Repositories (SnowflakeRepository).

Phase B — Function One development:

- Implement QueueTrigger, message parsing, blob download & copy, audit insert, enqueue staging message.

Phase C — Function Two development:

- Implement QueueTrigger, blob parsing, filtering logic, Member API integration, insert to PENDED\_CLAIMS\_OUTBOUND, and ERROR\_LOG handling.

Phase D — Infra & Deployment:

- Terraform for storage accounts, queues, function apps, managed identities, key vault.  
- Azure DevOps YAML for CI/CD with unit tests, code scanning, and deployment stages.

Phase E — QA & Integration:

- Integration tests with Azurite or sandbox storage account. Load testing and observability verification.

### 5.1 Developer Checklist & Local Setup

- Install .NET 6 LTS SDK, Azure Functions Core Tools, Visual Studio/VSCode.  
- Configure local.settings.json with placeholders for StorageConnectionString, KeyVault URI, Member API URL, Snowflake placeholders.  
- Run unit tests with xUnit & Moq.

## 6. Monitoring & Operations

- Application Insights: function traces, exceptions, custom metrics (processed\_count, error\_count).

- Azure Monitor Alerts: high error rate, DLQ depth, Snowflake ingestion failures.

- ServiceNow integration: auto-create ticket for critical failures (DLQ items, Snowflake outage).

## 7. References

- Azure Functions docs, Azure Blob & Queue Storage docs, Snowflake docs, Apigee docs, ServiceNow API docs.

## 8. Amendment History

Version 1.0 — 2025-10-05 — Initial detailed SDD for Flow 3

# Appendix A – Snowflake DDL and Sample Insert

Use the following DDLs to create the required tables in the BCBSRI\_CLAIMS.OUTBOUND schema. Update placeholders for your environment.

CREATE OR REPLACE DATABASE IF NOT EXISTS BCBSRI\_CLAIMS;  
CREATE OR REPLACE SCHEMA IF NOT EXISTS BCBSRI\_CLAIMS.OUTBOUND;  
  
CREATE OR REPLACE TABLE BCBSRI\_CLAIMS.OUTBOUND.PENDED\_CLAIMS\_OUTBOUND (  
 CLAIM\_GUID VARCHAR(36) NOT NULL,  
 CLAIM\_ID VARCHAR(64) NOT NULL,  
 MEMBER\_ID VARCHAR(64) NOT NULL,  
 POLICY\_ID VARCHAR(64),  
 CLAIM\_TYPE VARCHAR(50),  
 STATUS VARCHAR(20),  
 PENDED\_REASON VARCHAR(255),  
 SERVICE\_START\_DATE DATE,  
 SERVICE\_END\_DATE DATE,  
 PROVIDER\_ID VARCHAR(64),  
 DIAGNOSIS\_CODES ARRAY,  
 PROCEDURE\_CODES ARRAY,  
 CLAIM\_LINES VARIANT,  
 SOURCE\_FILE\_NAME VARCHAR(512),  
 BATCH\_ID VARCHAR(64),  
 RAW\_PAYLOAD VARIANT,  
 PENDED\_TIMESTAMP TIMESTAMP\_LTZ(9),  
 INGESTION\_TIMESTAMP TIMESTAMP\_LTZ(9) DEFAULT CURRENT\_TIMESTAMP(),  
 LOAD\_STATUS VARCHAR(20) DEFAULT 'PENDING',  
 ERROR\_CODE VARCHAR(10),  
 ERROR\_MESSAGE VARCHAR(512),  
 PROCESSED\_BY VARCHAR(128),  
 LAST\_UPDATED\_BY VARCHAR(128),  
 LAST\_UPDATED\_TIMESTAMP TIMESTAMP\_LTZ(9),  
 CONSTRAINT PK\_PENDED\_CLAIMS\_OUTBOUND PRIMARY KEY (CLAIM\_GUID)  
);

CREATE OR REPLACE TABLE BCBSRI\_CLAIMS.OUTBOUND.AUDIT\_CLAIM\_FILE (  
 AUDIT\_ID VARCHAR(36) PRIMARY KEY,  
 CREATED\_DATETIME TIMESTAMP\_LTZ(9) DEFAULT CURRENT\_TIMESTAMP(),  
 UPDATED\_DATETIME TIMESTAMP\_LTZ(9),  
 CLAIM\_FILE\_NAME VARCHAR(512),  
 MESSAGE\_PATH VARCHAR(1024),  
 TRANSACTION\_STATUS CHAR(1),  
 ERROR\_DETAILS VARIANT,  
 ORIGIN\_TRANSACTION VARCHAR(128),  
 PROCESS\_ID VARCHAR(64),  
 BATCH\_ID VARCHAR(64)  
);

CREATE OR REPLACE TABLE BCBSRI\_CLAIMS.OUTBOUND.ERROR\_LOG (  
 ERROR\_ID VARCHAR(36) PRIMARY KEY,  
 CLAIM\_GUID VARCHAR(36),  
 CLAIM\_ID VARCHAR(64),  
 BATCH\_ID VARCHAR(64),  
 ERROR\_CODE VARCHAR(10),  
 ERROR\_MESSAGE VARCHAR(512),  
 ERROR\_DETAILS VARIANT,  
 STATUS VARCHAR(20) DEFAULT 'FAILED',  
 FUNCTION\_NAME VARCHAR(128),  
 CREATED\_TIMESTAMP TIMESTAMP\_LTZ(9) DEFAULT CURRENT\_TIMESTAMP(),  
 UPDATED\_TIMESTAMP TIMESTAMP\_LTZ(9)  
);

CREATE OR REPLACE TABLE BCBSRI\_CLAIMS.OUTBOUND.PROCESS\_LOG (  
 LOG\_ID VARCHAR(36) PRIMARY KEY,  
 PROCESS\_ID VARCHAR(64),  
 FUNCTION\_NAME VARCHAR(128),  
 TIMESTAMP TIMESTAMP\_LTZ(9) DEFAULT CURRENT\_TIMESTAMP(),  
 SEVERITY VARCHAR(10),  
 MESSAGE VARCHAR(1024),  
 ERROR\_CODE VARCHAR(10),  
 ERROR\_DETAILS VARIANT  
);

-- Sample insert using PARSE\_JSON for a single claim  
INSERT INTO BCBSRI\_CLAIMS.OUTBOUND.PENDED\_CLAIMS\_OUTBOUND (  
 CLAIM\_GUID, CLAIM\_ID, MEMBER\_ID, POLICY\_ID, CLAIM\_TYPE, STATUS, PENDED\_REASON,  
 SERVICE\_START\_DATE, SERVICE\_END\_DATE, PROVIDER\_ID,  
 DIAGNOSIS\_CODES, PROCEDURE\_CODES, CLAIM\_LINES, RAW\_PAYLOAD, BATCH\_ID, SOURCE\_FILE\_NAME,  
 PROCESSED\_BY, LOAD\_STATUS  
)  
SELECT  
 UUID\_STRING(),  
 claim:"claim\_id"::string,  
 claim:"member\_id"::string,  
 claim:"policy\_id"::string,  
 claim:"claim\_type"::string,  
 claim:"status"::string,  
 claim:"pended\_reason"::string,  
 claim:"service\_start\_date"::date,  
 claim:"service\_end\_date"::date,  
 claim:"provider\_id"::string,  
 claim:"diagnosis\_codes"::array,  
 claim:"procedure\_codes"::array,  
 claim:"claim\_lines"::variant,  
 TO\_VARIANT(claim),  
 '<BATCH123>',  
 '<claimsfile.json>',  
 'FunctionTwo\_Process',  
 'SUCCESS'  
FROM VALUES  
 (PARSE\_JSON('{  
 "claim\_id": "CLM0001",  
 "member\_id": "M1001",  
 "policy\_id": "P1001",  
 "claim\_type": "Medical",  
 "status": "PENDING",  
 "pended\_reason": "PrimeReview",  
 "service\_start\_date": "2025-09-01",  
 "service\_end\_date": "2025-09-01",  
 "provider\_id": "PRV001",  
 "diagnosis\_codes": ["D1","D2"],  
 "procedure\_codes": ["P1"],  
 "claim\_lines": [  
 {"line\_id":"L1","service\_code":"SVC001","service\_date":"2025-09-01","billed\_amount":100.0,"allowed\_amount":80.0}  
 ]  
 }')) AS t(claim);

# Appendix B – Sample JSON Payloads

1) Source Queue Message (Event Grid shape):

{  
 "topic": "/subscriptions/{sub}/resourceGroups/{rg}/providers/Microsoft.Storage/storageAccounts/{sa}",  
 "subject": "/blobServices/default/containers/claims-raw/blobs/2025/09/25/claimsfile.json",  
 "eventType": "Microsoft.Storage.BlobCreated",  
 "data": {  
 "url": "https://{sa}.blob.core.windows.net/claims-raw/2025/09/25/claimsfile.json",  
 "contentType": "application/json"  
 }  
}

2) Staging Queue Message (for Function Two):

{  
 "topic": "...",  
 "subject": "/blobServices/default/containers/claims-staging/blobs/2025/09/25/batch-123-claimsfile.json",  
 "data": {  
 "url": "https://{sa}.blob.core.windows.net/claims-staging/2025/09/25/batch-123-claimsfile.json",  
 "contentType": "application/json"  
 }  
}

3) Claim File JSON (array of claims):

[  
 {  
 "claim\_id": "CLM0001",  
 "member\_id": "M1001",  
 "policy\_id": "P1001",  
 "claim\_type": "Medical",  
 "status": "PENDING",  
 "pended\_reason": "PrimeReview",  
 "service\_start\_date": "2025-09-01",  
 "service\_end\_date": "2025-09-01",  
 "provider\_id": "PRV001",  
 "diagnosis\_codes": ["D1","D2"],  
 "procedure\_codes": ["P1"],  
 "claim\_lines": [  
 {  
 "line\_id": "L1",  
 "service\_code": "SVC001",  
 "service\_date": "2025-09-01",  
 "billed\_amount": 100.0,  
 "allowed\_amount": 80.0  
 }  
 ]  
 }  
]

# Appendix C – Error Code Reference Table

|  |  |  |  |
| --- | --- | --- | --- |
| Error Code | Category | Description | Remediation |
| ERR-QMSG-001 | Queue Message | Invalid/malformed queue message | Inspect original Event Grid message; re-enqueue after fix |
| ERR-BLOB-404 | Blob | Source blob not found | Verify blob path; check upstream producer |
| ERR-COPY-3001 | Copy | Failed to copy to staging | Check storage permissions; retry |
| ERR-CLAIM-VAL-001 | Validation | Claim missing fields or invalid types | Quarantine claim; notify business |
| ERR-API-4003 | API | Member API 5xx/server error | Retry with backoff; escalate if persistent |
| ERR-SNOW-500 | Snowflake | Insert/load failure | Review ERROR\_LOG details; correct data and reprocess |
| ERR-DLQ-6001 | DLQ | Message moved to DLQ | Manual inspect and re-enqueue when fixed |
| ERR-AUDIT-100 | Audit | Failed to write audit entry | Investigate DB connectivity; re-attempt write |