Discover Payment Services

CPP Logging
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
16/AUG/2016	1.0	Initial version	Mike Knauff

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

The CPP Logging Software Architecture Document provides the logical view of system as well as the major technical requirements, both functional and non-functional, for CPP application logging domain.

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 KPI

Key Performance Indicator

1.3.2 SOP

Standard Operating Procedure

1.4 References

- Common Payments Platform Software Architecture Document

1.5 Overview

The CPP Logging Software Architecture Document is organized into the following major sections:

- *Architectural Representation* that provides a logical view of the CPP logging domain or ecosystem
- *Architecture Quality Requirements* that provides the major non-functional requirements (NFRs) that system architecture and any subsequent designs must satisfy
- *Key Features and Functions* defines the major functional requirements that must be supported by the architecture and the physical designs that implement the architecture
- *Data View* that provides an overview of the major CPP log event types including their taxonomy and purpose

Formatted: Body Text, Bulleted + Level: 1 + Aligned at: 0.75" + Indent at: 1"

Formatted: Body Text

Formatted: Font: Italic

Formatted: Body Text, Bulleted + Level: 1 + Aligned at: 0.75" + Indent at: 1"

Formatted: Font: Italic

Formatted: Font: Italic

Formatted: Font: Italic

2. Architectural Representation

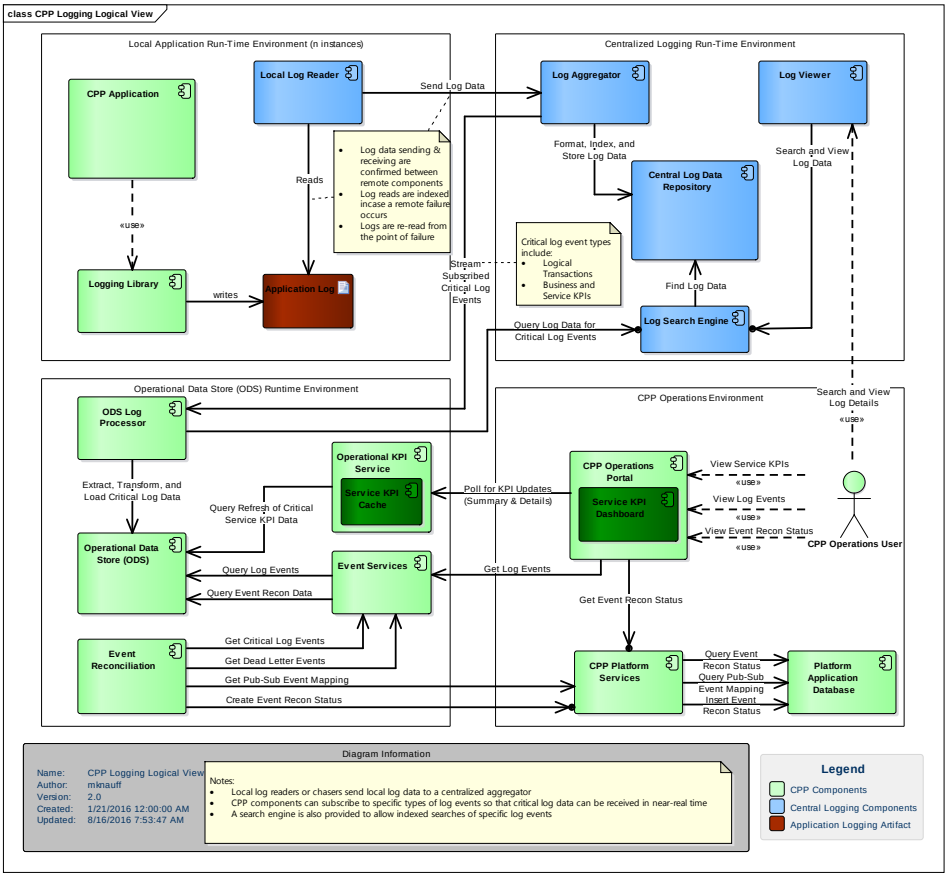

Figure 1 - CPP Logging Logical View

2.1 Description

The CPP utilizes a centralized logging architecture to support three (3) key use-cases required for the management, support, and compliance of the platform:

- **Operational Management** of the many distributed instances of platform applications and (micro) services for state/status monitoring and alerts of system and business KPIs, as well as exceptional conditions (errors and warnings)
- **Reconciliation of Critical Platform Events** that require transaction-like integrity ("guaranteed" delivery of asynchronous messages) such as events that trigger monetary assessments, fees, adjustments, and settlement with platform tenants and tenant customers
- **Auditing of Security-Related Events** that require monitoring for violations of security policies or detection of security threats

Formatted: Body Text

The CPP relies on a combination and cooperation of several component categories to support the logging requirements:

2.1.1 Application Logging Components

2.1.1.1 CPP Application

The *CPP Application* captures the required log information utilizing standard labels and formats so that these can be identified uniformly by the clients of the application logs. The application also needs to provide sufficient information so that the individual application and instance can be identified in large and intermixed pool of log data from many applications and application instances.

2.1.1.2 Logging Library

The *Logging Library* is a third-party component logging library and framework utilized by CPP Applications to enable the logging of messages according to message type and level, and to control at runtime how these messages are formatted and where they are reported.

2.1.1.3 Application Log

The *Application Log* is the formatted output of the application and logging library that contains the events captured by the application for reporting in the *Application Log*. The output of the *Application Log* is configured at runtime and can be a file on disk, a database (memory or disk), or a stream.

2.1.2 Centralized Logging Components

2.1.2.1 Local Log Reader

The *Local Log Reader* is part of the central logging infrastructure that is deployed within the same environment as the applications and is responsible for reading the application log files locally and then sending the logging output to centralized log aggregator. It is the component that moves the log data from the many application instances to the central log collection point.

2.1.2.2 Log Aggregator

The *Log Aggregator* is a component of the central logging framework that collects input from all of the local log readers, transforms it if necessary and/or enriches with any logging metadata, indexes the content, and the stores it into the Central Log Data Repository. The *Log Aggregator* provides the ability for central logging clients to subscribe to specific log event topics in order to receive these log events in near-real time and before they have been placed into the Central Log Data Repository.

2.1.2.3 Central Log Data Repository

The *Central Log Data Repository* is a component of the central logging framework that is responsible for the central storage of all log data. Log data is stored in a common format with standard tags, these tags are indexed for rapid search and retrieval by log reader clients such as the Log Viewer and the CPP ODS Log Batch ETL Processor. The log data held in this repository is transitory and held for a limited amount of time.

2.1.2.4 Log Search Engine

The *Log Search Engine* is a component of the central logging framework that is responsible for providing a query/search API that allows the Log Viewer and CPP components to dynamically search the Central Log Data Repository for CPP-specific log events by log event type, logical transaction ID, system/service ID, date, time range, etc.

2.1.2.5 Log Viewer

The *Log Viewer* is part of the central logging framework that is responsible for providing access to the log data stored in the Central Log Data Repository. The *Log View* is a user interface that allows users such as CPP Operations Users the ability to search and view log data. Typical search criteria are related to application IDs and instance IDs, as well output levels (WARN, ERROR, FATAL, etc.) and event GUIDs and correlation IDs.

2.1.3 CPP Operational Data Store (ODS) Components

2.1.3.1 ODS Log Processor

The *ODS Log Processor* is responsible for receiving subscribed to log events (real-time stream) and querying the Central Log Data Repository for specific log events throughout the day, extracting these events, transforming them if required into the canonical event format of the ODS, and then loading this data into the ODS for use by various CPP processes including reconciliation of critical events and notification and display of CPP Business and Service KPIs.

2.1.3.2 Operational Data Store (ODS)

The *Operational Data Store (ODS)* provides for the storage of critical log events for the various CPP processes that are dependent on these events types and includes critical event reconciliation and logical transaction integrity, business and service KPI monitoring and alerts, critical log event queries and research, historical critical log event trends and operational-analysis of these events.

2.1.3.3 Operational KPI Service

The *Operational KPI Service* provides near real-time access to the critical log events that have been stored in the ODS. The *Operational KPI Service* caches the service KPI data both at a summary and detail level for quick access by the Service KPI Dashboard located within the CPP Operations Portal.

2.1.3.4 Event Service

The *Event Service* provides search access to clients for critical and other event types. Clients such as the CPP Operations Portal can invoke search APIs that retrieve details about critical log events that match the search criteria as well as business and system KPI data. The Event Service also provides status of the event reconciliation process.

2.1.3.5 Event Reconciliation

The *Event Reconciliation* retrieves critical log data from the Event Service and ensures that all published events were successfully received by the registered publishers and processed. The *Event Reconciliation* updates the ODS with the current status of the event reconciliation processing.

2.1.4 CPP Operations

2.1.4.1 CPP Operations Portal

The *CPP Operations Portal* provides a user interface that allows CPP Operations Users the ability to query and view critical log events such as errors, KPIs, and logical transactions that are stored in the CPP ODS.

2.1.4.2 Service KPI Dashboard

The *Service KPI Dashboard* allows CPP Operations Users to view in near-real time critical platform business KPIs such as service availability, service volume, and other important service metrics as defined by the business.

2.1.4.3 CPP Operations User

A *CPP Operations User* is a business or technology user that is responsible for configuring and monitoring the state of the platform and platform services, as well as researching and resolving platform errors and issues.

2.1.4.4 CPP Platform Services

The *CPP Platform Services* is responsible for storing the publisher and subscriber event mapping used in the event reconciliation process, as well as the status of the event reconciliation process for use by the CPP Operations User.

2.1.4.5 Platform Application Database

The *Platform Application Database* provides the durable storage of the publisher-subscriber event mapping and the event reconciliation status.

Formatted: Heading 4

Formatted: Font: Not Italic

Formatted: Heading 4

Formatted: Heading 4

2.1.5 Other User Types

2.1.5.1 DFS Command Center User

A *DFS Command Center User* is a DFS resource responsible for 24x7 monitoring of DFS applications and systems for critical errors and exceptions. The *DFS Command Center Users* may be dependent on the display of specific errors and exceptions that occur in the CPP logs for being alerted to conditions that they need to respond to as part of an SOP.

2.1.5.2 Security Operations User

A *Security Operations User* is a DFS resource that is responsible for monitoring and auditing application log information for security violations and predictive indicators of nefarious activity.

3. Architectural Quality Requirements

3.1 Availability

3.1.1 Application Logging

- Applications shall always be able to send their log output to the logging component
- Applications shall not block on logging operations
    - All logging operations will be asynchronous

3.1.2 Centralized Logging

- The centralized logging system shall be highly available (99.999%)
    - This is inclusive of system maintenance
    - This is required since application and security issues can occur at any time and the logging system must be available to capture these events and allow for searching and view of the application logs

3.2 Integrability and Interoperability

3.2.1 Application Types

- The logging solution shall capture log output for all application types including Java, C++, and vendor packages – CPP applications, middleware, and databases

3.2.2 Log Capture Methods

- The logging solution must be able to capture log output via a variety of output channels including standard input/output, standard error, and other stream types, databases, as well as log files, etc.

3.2.3 Environment Types

- The logging solution must be able to run minimally in Linux and Windows environments, or emulated versions of these
    - PaaS, IaaS, and bare metal environments

3.3 Modifiability

3.3.1 Adaptability

- The logging solution shall use an industry standard API that provides for different implementation libraries or components

Formatted: Bulleted + Level: 1 + Aligned at: 0.75" + Indent at: 1"

3.4 Reliability

3.4.1 Application Logging

- The ability of the applications to capture their log data locally is determined by the environmental factors and beyond the control of the local logging component, however it is expected that 99.8% of log contents will be accurately captured within the local environment

3.4.2 Centralized Logging

- The ability of the centralized logging system to reliably capture log contents and store them in a central repository is also dependent on environmental factors such as WAN and LAN connections, data packet integrity, multiple component availability, etc, however it is expected that the centralized logging system will accurately capture, transmit, and store 99.8% of log events that are transmitted

3.5 Size, Scalability, and Performance

To be determined

4. Key Features and Functions

4.1 Required Log Output

4.1.1 KPIs

- Depends on context/purpose of Log Event
- How long significant external service calls are taking (round-trip)
- How long significant internal processing is taking

4.1.2 Security Info

- Depends on context/purpose of Log Event

4.1.3 Transaction Event Info

- Depends on Log Event type

4.1.4 Error/Exception Information

- Depends on context/purpose of Log Event

4.1.5 Application/Service Execution Journal

- Discretion of the developer

4.2 Standard Log Event Format

- Follows the canonical model for CPP events
- New CPP event sub-type for logging
    - CPP Log Event
        - Journal Log Event
            - Information and Debug
        - Exception Log Event
            - Warnings and Errors
        - Transaction Log Event
            - Events that require logical transaction integrity

4.3 Content of log events

4.3.1 Required Security Content (minimum content)

- Log event timestamps
- Credential identifier (e.g., user ID, application ID, etc.)
- Event type
- Event date and time (at least to the millisecond and microsecond if possible)
- Success or failure indication
- Event source or origin
- Identity or name of the affected data, resources, or system components
- Event description
- Severity, priority, or alert indicator(e.g., fatal, error, warn, info, debug)

4.3.2 Identity of the Service

4.3.3 Identity of the Service Instance

4.3.4 Location and Environment of the Service Instance

4.3.5 Linkage to other Components and Events in a Logical Event Handling or Transaction Chain

- *origEventCorrelationID*
  - Located in the Transaction Event Header
  - Provides the original transaction event's eventCorrelationID
  - Can serve as a search key to get all log information related to the original transaction
- *origEventAncestorID*
  - Located in the Transaction Event Header
  - Provides the original transaction event's individual and unique eventMessageGUID
  - Can serve as a search key to get all log information related to the specific transaction event
- *origEventOutcome*
  - Located in the Transaction Event Header
  - Provides the processing outcome (Success or Failure) of the original transaction by the current CPP component of service
  - Can serve as a search key to get only those transaction event service or processing points that failed in the logical transaction
- *origEventOccuranceTime*
  - Located in the Transaction Event Header
  - Provides the time occurrence of the original transaction
  - Can serve as a search key to get only those transaction events that occurred during a specific time period

4.4 Interfaces to the Centralized Logging Service

4.4.1 Non-PCF Applications

4.4.2 PCF Applications

4.4.3 Database Interface

4.4.4 Legacy Applications and Vendor Products (Non-PaaS)

4.5 The Standard CPP Client logging API

- slf4j
  - The Simple Logging Framework for Java is the standard logging API for CPP applications and services
- LOGBack
  - Implements all slf4j features

4.5.1.1 What are the points in the code that are required to have logging points?

- Results or completion of transaction event processing transaction
- Application or Service Audit Events
    - CRUD operations to all data/CRUD operations on highly sensitive data
    - All actions taken by administrative accounts
    - All actions taken by users with shared, generic, or group accounts
    - Actions taken to obtain access to privileged, administrative, or generic (shared) accounts
    - Initialization, stopping, or pausing of audit logs
    - Service configuration changes
    - Deletion, creation, or modification of any user/system accounts or access privileges
    - Successful and failed login attempts
    - Authorization failures
    - Access to cryptographic keys or associated cryptographic containers
    - Assignment or modification of unique system identifiers
    - All cross-zone user or application access to network or system resources
    - Access to the Internet by users or applications
    - Attempts to connect to remote systems, services, or applications
    - Application input validation errors such as unexpected characters, data field overflows, etc.
    - Material system or application events such as errors, aborts, restarts, etc.

4.6 Points in the Code that require Logging Events

- Service calls
    - name
    - result
    - elapsed time
    - security information
- Auditable security points
    - per security logging requirements
- All event publishing, routing, subscription, and processing
    - event details
    - steps
    - results

4.7 Division of Labor between CPP Apps and the DFS Central Logging Service

- CPP Apps will provide the logging output and formatting
- The Central Logging product could do formatting but the CPP apps will want control of this so that they can ensure getting the log info out at the central repository

- The Central Logging product will be responsible for aggregating all of the CPP instance logs into a central repository, providing a search engine to search and extract log data, and a UI for users to search and view the log data
    - Log event generation (publishing) based upon pre-defined events types
        - E.g., critical events
    - Log event dash boarding

4.8 Critical Functional Requirements

- Must be able to segment log contents by tenant and tenant users
    - ACL controls access to log data
- Searching log contents based upon tenant (Discover, DCI, PULSE, etc.)
- Searching log contents based upon service instance (component depth search)
- Searching log contents based upon a logical transaction or by associated events (component breadth search)
- Extract service and component KPI's

5. Data View

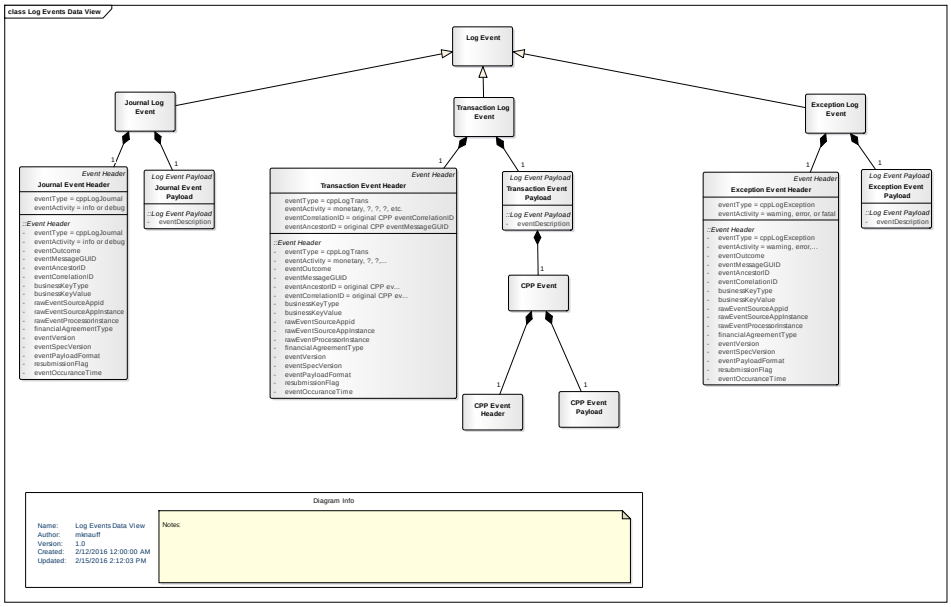

Figure 2 - Log Event Logical View

5.1 Log Events (Canonical Model)

5.1.1 Log Event Header

The *Log Event Header* provides the meta-data about the log event that allows the event to be indexed and catalog for retrieval out of the central logging system, and logically separated from other types of events from other tenants and systems.

Formatted: Body Text

- from *Event Header*
  - *eventType*
    - "logEvent"
  - *rawEventSourceAppid*
    - The application or service ID of the application/service generating the log event
  - *rawEventSourceAppInstance*
    - The unique instance ID of the application/service instance generating the log event
  - *eventVersion*
    - The version of the log event
  - *eventSpecVersion*
    - The version of the specification that defines the log event

- *eventOccuranceTime*
  - The date-time that the log event was generated

5.1.2 Log Event Payload

The *Log Event Payload* is a "marker" interface only. All log payloads are specific to the log event type.

5.1.3 Journal Log Events

Formatted: Heading 3, No bullets or numbering

Journal Events are a type of Log Event that captures either typical operational activity of a component, specialized information that is captured for debugging of component behavior, or alerting of exceptional conditions including warnings, errors, and fatal conditions.

Journal Events can also be used for journaling operational milestones within the code such as method or service call response times that can be used for either business or system KPI metrics.

This type of log event has the least amount of restrictions on its format and content.

5.1.4 Transaction Log Events

Formatted: Heading 3, No bullets or numbering

Transaction Events are a type of Log Event that captures critical information for the logging of event types that require transactional integrity. Format is critical for these types of log events since the CPP platform uses an eventual consistency model for enforcing transactional integrity.

The component that provides the event reconciliation function uses elements of the header to retrieve these events from the central logging service and perform the reconciliation processing. The payload contains the original business or system event that requires the transactional integrity. The event embedded in the payload can be used for "event replay" purposes in cases where the original event was not received or not successfully processed.

5.1.5 Exception Log Events

Formatted: Heading 3, No bullets or numbering

Exception Events are a type of log event that indicates an unexpected or unusual event that should be placed in the logging stream for potential action or analysis by other systems. These systems can make the determination of what action to take based upon the type of error or warning contained within the payload.

The event payload contains the type of error or warning as well as the throwable object complete with the stack trace of the error or warning if applicable.

5.1.6 Security Log Events

Formatted: Heading 3, No bullets or numbering

Security Events are any activities that are defined within the DFS Cybersecurity Guidelines for Enterprise Logging (version 01/12/2016) that require the event to be recorded within an application log.

The payload for these events contain critical attributes that allow DFS mechanisms to actively monitor for security risks or to reconstruct events from a previously undetected security incident.