Department of Veterans Affairs

API Developer Playbook

API Developer Playbook Blueprint Documentation

Maps to Framework Set #1



Document Version Draft 01.00.01

Revision History

| Date | Version | Description | Author |
| --- | --- | --- | --- |
|  |  |  |  |

Table of Contents

[1. Baseline Logging 1](#_Toc519773818)

[1.1. Logging Overview 1](#_Toc519773819)

[1.2. Logging Framework Used within Each API 1](#_Toc519773820)

[1.3. Logging with Java supported API implementations 1](#_Toc519773821)

[1.3.1. Simple Logging Facade for Java (SLF4J) 1](#_Toc519773822)

[1.3.1.1. Subset of Features 1](#_Toc519773823)

[1.3.2. Apache Log4j 2 1](#_Toc519773824)

[1.3.2.1. Subset of Features 2](#_Toc519773825)

[1.3.3. Logback 2](#_Toc519773826)

[1.3.3.1. Subset of Features 2](#_Toc519773827)

[1.3.4. java.util.logging (jul) 2](#_Toc519773828)

[1.3.4.1. Subset of Features 2](#_Toc519773829)

[1.4. Logging with .Net supported API implementations 3](#_Toc519773830)

[1.4.1. Overview of .NET Framework and .NET Core 3](#_Toc519773831)

[1.4.1.1. .NET Core Logging Capabilities 3](#_Toc519773832)

[1.5. Logged API Messages 5](#_Toc519773833)

[1.5.1. Enterprise Shared Services Logging Model 5](#_Toc519773834)

[1.6. Logging Policy Assertions 9](#_Toc519773835)

[1.7. API Audit Level Logging 11](#_Toc519773836)

[1.8. Integration with Third-Party Log Aggregators 12](#_Toc519773837)

[1.8.1. Splunk 12](#_Toc519773838)

[1.8.2. Logentries 12](#_Toc519773839)

[1.8.3. GoAccess 13](#_Toc519773840)

Table of Tables

[Table 1: ESS Logging Description Table 5](#_Toc519773841)

[Table 2: Logging Policy Assertion Table 9](#_Toc519773842)

**Table of Figures**

[Figure 1 ESS Logging Model 5](#_Toc519773843)

[Figure 2 Logging Example 8](#_Toc519773844)

# Baseline Logging

## Logging Overview

To manage highly dynamic application environments, organizations turn to application logging infrastructures for assistance, but reliance on these systems can have an undesired impact on the applications themselves. Organizations typically use some type of logging system or systems to log errors, traces, information messages or debugging information etc. Applications can write logs to a file, database, or big data store and there are many logging frameworks and practices being used.

Logging provides meaningful insights concerning application behavior, especially dealing with failures. However, by being part of the application, logging also participates in the execution chain, which can result in disadvantages such as a negative impact on performance.

Typically, the overhead of logging is negligible. This scenario changes when the application is subjected to significant loads, these are the circumstances when a well thought out logging implementation such as the deployment of asynchronous implementation where needed pays dividends.

## Logging Framework Used within Each API

This section will discuss logging framework capabilities for two of the more prevalent application development environment at the VA namely Java (Enterprise as well Standard Editions) and well as Microsoft .NET

## Logging with Java supported API implementations

This article discusses the most popular java logging frameworks: Simple Logging Facade for Java (SLF4J) which is a logging facade that provides a common interface for different logging frameworks, Log4j 2 and its predecessor Log4J and Logback.

### Simple Logging Facade for Java (SLF4J)

The Simple Logging Facade for Java (SLF4J) serves as a simple facade for various logging frameworks (e.g. , log4j 2, java.util.logging (jul), Logback) allowing the end user to select the desired logging framework at deployment time. In practice, 4 out of 5 Java projects choose to go with slf4j based on data from the top Java libraries in 2016 on Github.

#### Subset of Features

* The separation of the client API from the logging backend reduces the coupling between an application and any logging framework. This can make it easier to integrate with existing logging frameworks or to integrate functionality into applications projects that have already made a choice of logging backend.

### Apache Log4j 2

Apache Log4j 2 is an upgrade to Log4j and provides significant improvements over its predecessor plus it provides many of the improvements available in Logback while fixing some inherent problems in Logback’s architecture.

#### Subset of Features

* API Separation

The API for Log4j is separate from the implementation making it clear for application developers which classes and methods they can use while ensuring forward compatibility. This allows the Log4j team to improve the implementation safely and in a compatible manner.

* Improved Performance

Log4j 2 contains next-generation Asynchronous Loggers based on the LMAX Disruptor library. In multi-threaded scenarios Asynchronous Loggers have higher throughput and orders of magnitude lower latency than Log4j and Logback.

### Logback

Logback is intended as a successor to the popular log4j project, picking up where log4j leaves off and is conceptually very similar to log4j as both projects were created by the same developer.

#### Subset of Features

* Faster implementation

Based on the previous work on log4j, Logback internals have been re-written to perform faster on certain critical execution paths. Not only are Logback components faster, they have a smaller memory footprint as well.

* Logback-classic “speaks” SLF4J natively

Since the ***Logger*** class in Logback-classic implements the SLF4J API natively, there is zero overhead when invoking a SLF4J logger with Logback classic as the underlying implementation. Additionally, since Logback-classic strongly encourages the use of SLF4J as its client API, if the need arises the switch to log4j or to java.util.logging.

### java.util.logging (jul)

The Java Runtime Java provides a default framework in the java.util.logging package although This most developers would agree that java.util.logging is being replaced with 3rd party logging frameworks.

#### Subset of Features

java.util.logging in included in the Java Runtime so third-party integration (i.e. jar files) is not required.

## Logging with .Net supported API implementations

### Overview of .NET Framework and .NET Core

.NET Core is essentially a fork of the NET Framework whose implementation is also optimized around factoring concerns raised by the .NET framework siloed architecture. This section will focus on internal and external logging functionality supplied by .NET Core.

The following is a very high level over view of the two logging platforms:

* .NET Core

.NET Core is a cross-platform version of .NET for building websites, services, and console apps.

* .NET Framework

.NET Framework is a Windows-only version of .NET for building any type of app that runs on Windows.

#### .NET Core Logging Capabilities

ASP.NET Core supports a logging API that works with a variety of logging providers. Built-in providers facilitate the sending logs to one or more destinations plus plug in capability to a third-party logging framework.

.NET Core ships with the following built-in logging providers:

* Console

The ***Microsoft.Extensions.Logging.Console*** provider package sends log output to the console.

* Debug

The ***Microsoft.Extensions.Logging.Debug*** provider package writes log output by using the ***System.Diagnostics.Debug*** class (Debug.WriteLine method calls).

* EventSource

For apps that target ASP.NET Core 1.1.0 or higher, the ***Microsoft.Extensions.Logging.EventSource*** provider package can implement event tracing.

* EventLog

The Microsoft.Extensions.Logging.EventLog provider package sends log output to the Windows Event Log.

* TraceSource

The ***Microsoft.Extensions.Logging.TraceSource*** provider package uses the ***System.Diagnostics.TraceSource*** libraries and providers.

* Azure App Service

The ***Microsoft.Extensions.Logging.AzureAppServices*** provider package writes logs to text files in an Azure App Service app's file system and to blob storage in an Azure Storage account. The provider is available only for apps that target ASP.NET Core 1.1 or later.

## Logged API Messages

This section of the document is consistent with the Exception Logging Guidelines section of the Exception Handling Document.

The Enterprise Shared Services (ESS) Exception Handling Guidelines document, is pointed to by this imbedded document:



This document contains XML based logging guidelines that were defined for SOAP services. From a consistency perspective, continuing these guidelines for RESTful based services would be beneficial.

### Enterprise Shared Services Logging Model

Figure 1 ESS Logging Model



Table 1: ESS Logging Description Table

| **Element** | **Type** | **Required** | | **Example** | **Description** |
| --- | --- | --- | --- | --- | --- |
| transaction ID | String | Yes | | a20014d3e17ff4fcb578-b7a6-41f9-b974-cef684d5ec62 | A unique identifier passed to the service in the message header. |
| Code | int | Yes | 32154 | | A descriptor of the fault condition, and generated by the service. It may be unique to the service, but is typically a coded entry defined by the protocol used on the service interface. |
| Text | String | Yes | gov.va.ess.service.AccessServiceLegacySystemError | | A textual description of the fault condition, and generated by the service. It may be unique to the service, but is typically defined by the protocol used on the service interface. It should be human readable. |
| essCode | String | Yes | gov.va.ess.fault.io.FileCannotBeFound | | A code representing the mapping of the fault condition to the standard ESS taxonomy. |
| essText | String | No | Requested action is denied due to insufficient authorization | | A user interface friendly text description of the fault condition. The service generating the fault may populate this field. |
| nestedFault |  | No |  | | Nested structure of faults bubbled up from lower in the chain of service calls.  Only populated if fault is triggered by another fault. |
| severity | String | Yes | Error | | Enumeration:  **Fatal** – Severe error that causes premature termination of service execution. The service would not able to continue processing the request, and a response message would not be sent. An example would be an unrecoverable system level error.  **Error** – Significant error that causes the service to return a fault. An exception occurred that prevented the service from completing the request normally and a fault was returned. Examples might include a database constraint error, or disk full.  **Warning** – Significant error that does not result in the return of a fault, but is otherwise important to understanding service behavior. A recoverable exception that does not result in a fault message. Abnormal retries on a database connection for example. May also include interesting events not generated by exceptions, but useful in forensic analysis  **Information** – Not an error, but information that is otherwise important to understanding service behavior. Example might be shutting the service down for maintenance. |
| timestamp | Timestamp | Yes | 2013-05-05T08:15:30-05:00 | | Timestamp of the time the error occurred. |
| serviceName | String | Yes | gov.va.ess.util.FileAccessService | | Unique name of the service. This is the service namespace. |
| serviceInstance | String | Yes | <http://101.12.34.97:443 /eligibility/claimsService/Benefits/eligibilityStatus>/v1 | | This is the service endpoint |
| hostName | String | No | vaaacmhvapp12 | | Name of the individual host within the cluster hosting the service instance. |
| hostIP | String | No | 101.34.28.134 | | IP address of the individual host. |
| userId | String | No | CN=Test User, OU=ESS,DC=OIT,DC=VA,DC=GOV | | Fully qualified identifier of the user passed on the request. |
| subjectId | String | No | <Id>1234567890V654321<Id> <IdType>NI<IdType>  <AssigningLocation>123VA</AssigningLocation>  <IdSource>200M</IdSource> | | Fully qualified identifier of the subject of the request. This would typically be the veteran or beneficiary but may be any entity that is the primary subject of the request and/or important to understanding the error. |
| codePackage | String | No | gov.va.ess.resource.fileAccess | | Identifier of the code package or deployment unit originating the exception. This should be unique. |
| serviceDomain | String | No | gov.va.ess.resources.v2.2.3 | | Name of the service domain |
| businessDomain | String | No | Education Eligibility | | Name of the business domain |
| correlationId | String | No | c1d2398c3-1449-7654-9793-29373a9b9c53c1d2398c3-1449-7654-9793-29373a9b9c53 | | Correlation ID of the current request |
| sessionId | String | No | 8839272636511237 | | Session ID of the user session |
| Fault Trace  or equivalent | String | No | java.io.IOException: Unable to find file 'topicalIndex.csv' ... | | Detailed information on the state of the execution of service code package when the error occurred. |

Figure 2 Logging Example

Some elements in this figure are truncated for readability.

{

"ESSError": {

"transaction ID": "a20014d3e17ff4fcb578-b7a6-41f9-b974-cef684d5ec62",

"code": "32154",

"text": "gov.va.ess.service.AccessServiceLegacySystemError",

"essCode": "gov.va.ess.fault.io.ResourceNotFound",

"essText": "UnabletolocateresourceXXX",

"nestedFault": {

"code": "400",

"text": "NotFound"

"essCode": "gov.va.ess.fault.io.ResourceNotFound",

"essText": "UnabletolocateresourceXXX",

}

"severity": "Information",

"timestamp": "2013-05-05T08:15:30-05:00",

"serviceName": "gov.va.ess.util.FileAccessService",

"serviceInstance": "http://101.12.34.97:443...,

"hostName": "vaaacmhvapp12",

"hostIP": "101.34.28.134",

"userID": "CN=Test User, OU=ESS,DC=OIT,DC=VA,DC=GOV",

"subjectID": "Id>1234567890V654321<Id> <IdType>NI<IdType>…>",

"codePackage": "gov.va.ess.resource.fileAccess",

"serviceDomain":"gov.va.ess.resources.v2.2.3",

"businessDomain": "Education Eligibility",

"Correlation ID": "c1d2398c3-1449-7654-9793-29373a9b9c53c1d2398c3…”, "Session ID": "8839272636511237",

"faultTrace": "java.io.IOException: Unable to find file 'topicalIndex.csv' ..."

}

}

## Logging Policy Assertions

The following table is a modified version of the Appendix C Extracted Policy Assertions table contained in the Enterprise Shared Services (ESS) Exception Handling Guidelines document, which pointed to by this imbedded document:



The table was modified to include references to Logging Assertions only. Refer to **Figure 1 ESS Logging Model** which defines the response logging model.

References in the table regarding central logging are dependent on the creation of a centralized logging service. There will be situations where exceptions should be logged centrally, but until this centralized error logging service is available, local logging will have to suffice.

Table 2: Logging Policy Assertion Table

| **Logging Policy Assertions** | | |
| --- | --- | --- |
| **Specification / Standard / Guideline** | **Notes** | **Required / Recommended** |
| Any exception that results in a fault message must log information according to the ESS logical data model for errors. |  | Required |
| Exceptions that do not result in a fault message, but do affect the behavior of service orchestrations or service consumers must log information. | Examples might include partial data, and exceptions that trigger manual data quality workflow. | Required |
| Access services and Partner services must all log errors returned by the components called. | Legacy system components and partner system components do not comply with ESS error logging conventions, therefore the services that wrap these components must log such errors. | Required |
| Faults received by an ESS service must be either wrapped as a nested fault, or handled and logged to the error log. | Refer to **Figure 1 ESS Logging Model** | Required |
| Errors in service orchestrations that result in incomplete result data sets must be indicated in the ESS Response Code according to the ESS response information model. | When a service or operation returns partial data, the status returned to the calling consumer shall flag the partial nature of the returned data. | Required |
| Exceptions that do not return a fault and do not influence the behavior of service orchestrations and service consumers should not be logged centrally. | Such exceptions should be logged locally according to local system conventions. | Recommended |
| Information such shutdowns due to scheduled maintenance should be logged centrally. |  | Recommended |
| Services should log errors locally. | All exceptions should be logged locally according to local system conventions. | Recommended |

## API Audit Level Logging

Logging is simply the abstract task of recording data about events that take place in an application. As such it focuses on things that are of interest at the development environment level.

Auditing Logging focuses on recording domain-level events: a transaction is created, a user is performing an action, etc. In certain types of application such as banking, there is a legal obligation to record such events. The audit log retains events that are logged to an audit logging infrastructure. Auditing is used to typically answer questions line who did what, why and when.

Audit implies active review of the logging. A system cannot audit without logging, but logging can exist without audit.

The elements that require an audit log to be created will vary on the business rules of the application. A stated above certain application has legal constraints that dictate what had to be included in an audit log. Because an audit log is a specialization of a log, the attribute requirements defined for logging should also be applied to audit logging.

The analysis that goes into the determination of business process that flow from business requirements will have to take into consideration and identify what are the auditing requirements.

## Integration with Third-Party Log Aggregators

Application logs can be difficult to deal with, but regardless logging is an important aspect of any production system. When faced with investigating a system issue, it’s much easier to use a log aggregation management solution then it is to parse a jumble of text-files spread throughout the system environment.

A log aggregation management tools will prove its worth by facilitating the pinpointing of the root cause of any application or software error, within a single query. Another factor is having a visual overview of how software is being used globally by the system’s user base, getting all this crucial data in one single component such as a dashboard is going to make the job of monitoring a system’s health easier.

There are many log management features in the market today and the following is but a small subset of these products although Splunk is being integrated in the VA’s cloud infrastructures.

### Splunk

Splunk focuses its log management services around enterprise customers who need concise tools for searching, diagnosing and reporting any events surrounding data logs. Splunk’s software is built to support the process of indexing and deciphering logs of any type, whether structured, unstructured, or sophisticated application logs, based on a multi-line approach.

The VA has two Splunk products in the TRM; [Splunk Enterprise](https://www.oit.va.gov/Services/TRM/ToolPage.aspx?tid=6615) and [Splunk Enterprise Security](https://www.oit.va.gov/Services/TRM/ToolPage.aspx?tid=8352). Splunk Enterprise is used to manage logs and provide feedback on function and performance to developers and system owners. The Security product looks for vulnerabilities and is used in risk mitigation by VA Security. Both read the same log files from the same forwarders and indexers. While the ATO for a system may already have Splunk installed and in use, permissions for Splunk Enterprise for use by developers may be required separately. Licensing for Splunk is ‘log sized’ based. Thus developers use of Splunk may trigger larger log storage. It is important to contact the system owner for licensing information.

Key Features:

* Splunk understands machine-data of any type; servers, web servers, networks, exchanges, mainframes, security devices, etc.
* Flexible UI for searching and analyzing data in real-time.
* Drilling algorithm for finding anomalies and familiar patterns across log files.
* Monitoring and alert system for keeping an eye on important events and actions.
* Visual reporting using an automated dashboard output.

### Logentries

Logentries is a cloud-based log management platform that makes any type of computer-generated type of log data accessible to developers, IT engineers, and business analysis groups of any size. Logentries’ easy onboarding process ensures that any business team can quickly and effectively start understanding their log data from day one.

Logentries is currently not in the VA TRM. While application for a project to use Logentries and add it to the TRM is more work for the initial user, the benefit to the VA in the long term may be worth the effort.

Key Features:

* Real-time search and monitoring; contextual view, custom tags, and live-tail search.
* Dynamic scaling for different types and sizes of infrastructure.
* In-depth visual analysis of data trends.
* Custom alerts and reporting of pre-defined queries.
* Modern security features to protect your data.
* Flawless integration with leading chat and performance management tools.

### GoAccess

GoAccess is a real-time log analyzer software intended to be run through the terminal of Unix systems, or through the browser. It provides a rapid logging environment where data can be displayed within milliseconds of it being stored on the server.

GoAccess is currently not in the VA TRM. While application for a project to use GoAccess and add it to the TRM is more work for the initial user, the benefit to the VA in the long term may be worth the effort.

Key Features:

* Truly real-time; updates log data within milliseconds within the terminal environment.
* Custom log strings.
* Monitor pages for their response time; ideal for apps.
* Effortless configuration; select your log file and run GoAccess.
* Understand your website visitor data in real-time.