PROGRAMMER GUIDE

VisionLink® Raw Telematics Data API

VLRT_API_01_0420
April 2020

VISIONLINK.

Corporate Office

VirtualSite Solutions LLC
10368 Westmoor Drive
Westminster, CO 80021
USA

Legal Notices

Copyright and Trademarks

Release Notice

This is the April 2020 release (VLRT_API_01_0420) of the **VisionLink API Programmer Guide**. This **VisionLink API Programmer Guide** replaces the **VisionLink API Programmer Guide** published July 2019 and earlier.

Contents

Introduction

This manual describes the VisionLink® Raw Telematics Data API (known as VisionLink API) and the services it provides.

Even if you have used the VisionLink API before, VirtualSite Solutions recommends that you spend some time reading this manual to learn about the special features of this product.

About the VisionLink API

The VisionLink API provides the user access to some of the essential API topics that can be used in the user-specified application.

What's new

Be sure to use the updated URLs as follows to ensure successful requests for data to our database:

Updated URLs:
https://legacy.myvisionlink.com/APIService/ CATDataTopics/v4/feed/YourOrganization/SMULOC/0
https://legacy.myvisionlink.com/APIService/ CATDataTopics/v5/feed/YourOrganization/Event/0
https://legacy.myvisionlink.com/APIService/ CATDataTopics/v4/feed/YourOrganization/Diagnostic/0
https://legacy.myvisionlink.com/APIService/ CATDataTopics/v6/feed/YourOrganization/Engine/0

Related information

For access to current API documentation and to see sample code, go to:

https://legacy-apidocumentation.myvisionlink.com

This page provides links that enable you to quickly find the information you need to get started with integrating VisionLink API.

Accessing VisionLink API services

Follow this procedure to gain access to the VisionLink API services for the first time:

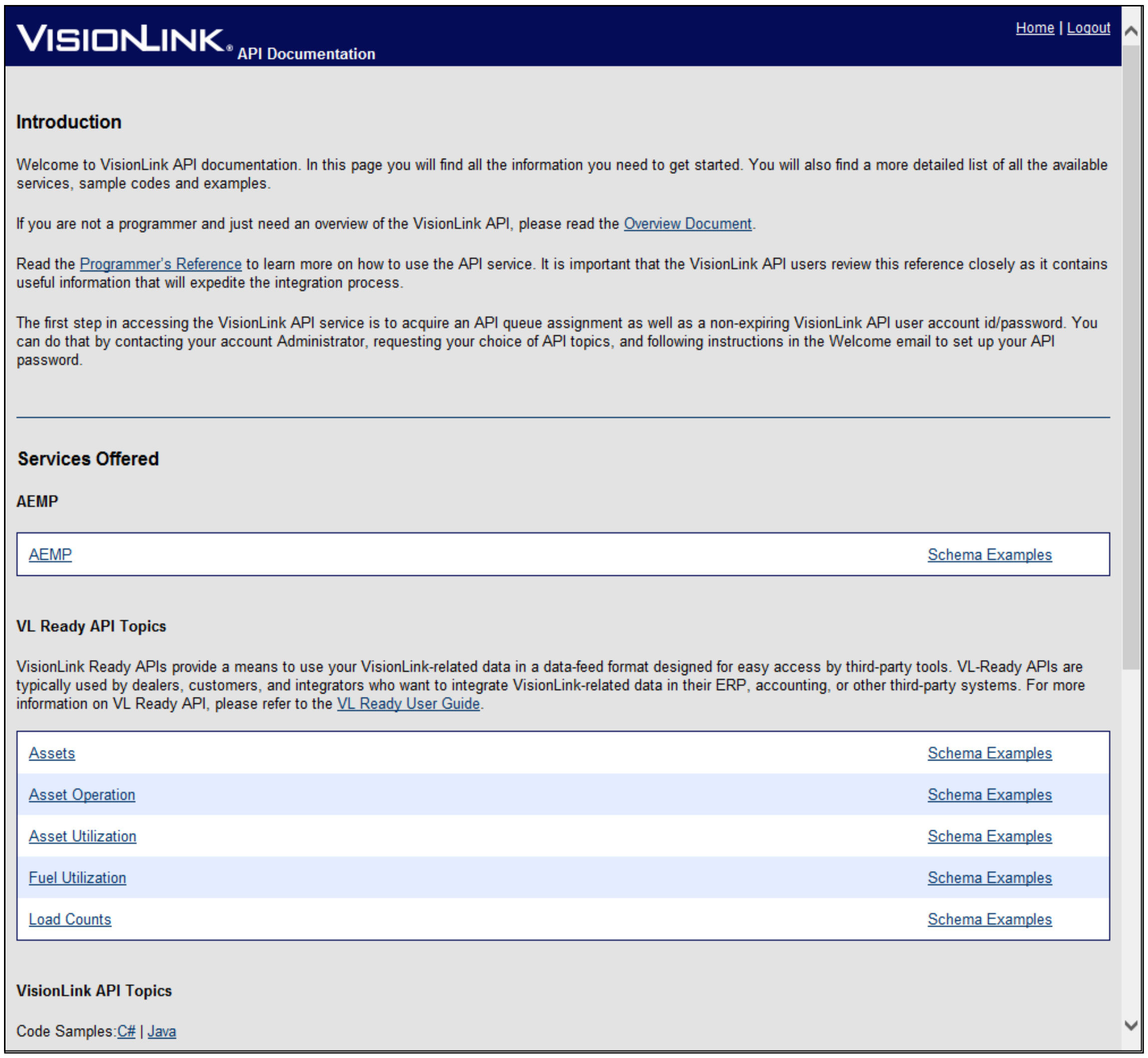
1. Acquire an API queue assignment and request a non-expiring VisionLink API user account id/password from your VisionLink account administrator.
2. Request your choice of API topics from the VisionLink account administrator.
3. Follow the instructions in the Welcome email to create your API password.

An automatic email notifies you once you are granted access to the API. In the future, request access to additional API topics by contacting your VisionLink account administrator.

Accessing and managing a queue

To access and manage queues, go to https://legacy-apidocumentation.myvisionlink.com

After you log in, the following screen shows the API programmer resource site:



Introduction

Welcome to VisionLink API documentation. In this page you will find all the information you need to get started. You will also find a more detailed list of all the available services, sample codes and examples.

If you are not a programmer and just need an overview of the VisionLink API, please read the Overview Document.

Read the Programmer's Reference to learn more on how to use the API service. It is important that the VisionLink API users review this reference closely as it contains useful information that will expedite the integration process.

The first step in accessing the VisionLink API service is to acquire an API queue assignment as well as a non-expiring VisionLink API user account id/password. You can do that by contacting your account Administrator, requesting your choice of API topics, and following instructions in the Welcome email to set up your API password.

Services Offered

AEMP

AEMP	Schema Examples

VL Ready API Topics

VisionLink Ready APIs provide a means to use your VisionLink-related data in a data-feed format designed for easy access by third-party tools. VL-Ready APIs are typically used by dealers, customers, and integrators who want to integrate VisionLink-related data in their ERP, accounting, or other third-party systems. For more information on VL Ready API, please refer to the VL Ready User Guide.

Assets	Schema Examples
Asset Operation	Schema Examples
Asset Utilization	Schema Examples
Fuel Utilization	Schema Examples
Load Counts	Schema Examples

VisionLink API Topics

Code Samples: C# | Java

This site provides several resources:

- Documentation detailing the content of each APItopic.
- Programmer resources to help develop application code to interface with theAPI.
- Links to XML schema (XSD) that provide access to the schemas describing the XML content for each topic. The schemas can be used with a validating XML parser to validate the content of the API XML.
- Facilities for managing the topics to which a queue issubscribed.

Potential API users can begin experimenting with the API using the code examples and the test queue before a dedicated queue is set up for the user's organization.

TESTQUEUE

As a convenience testing mechanism, a test queue named TESTQUEUE is available for access facilitating ramp up on API usage. Any authentic API login credentials can be provided as the security credential required to access the service. The TESTQUEUE is populated with at least one message for each supported API topic every one hour. As the TESTQUEUE is updated hourly, users can test API access and also the scheduling of API access, which will produce new message data every hour.

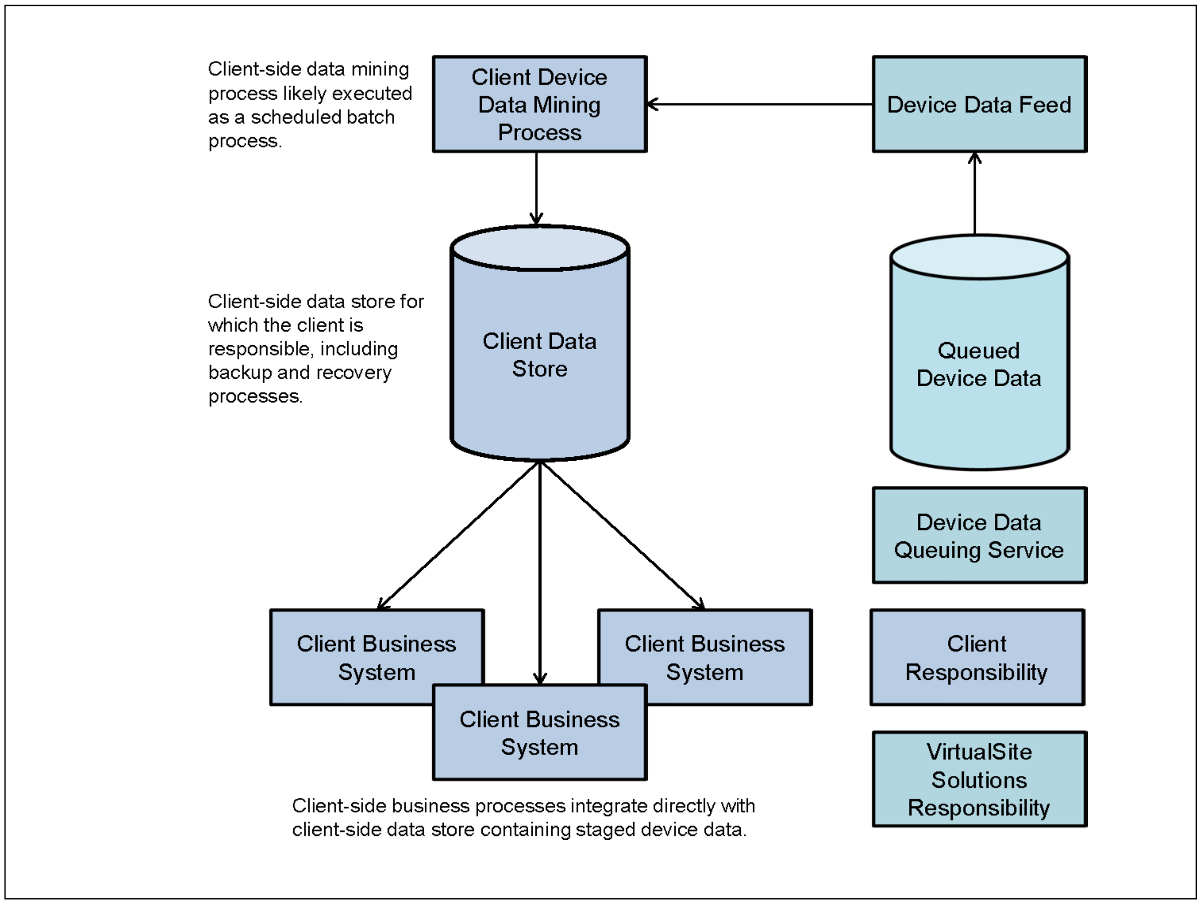
Authorization to queues

To access queue data (other than the TESTQUEUE), the user's API login credentials must be granted user access to the queue. If the queue has already been defined, please contact your VisionLink account administrator to get access to additional login credentials.

High level architecture

The API provides a simple REST feed over which device data can be retrieved. The VisionLink API is a "pull" service. That is, a client invokes the service as needed and pulls the required data to their computing platform for processing.

The following diagram illustrates the division of responsibility between VirtualSite Solutions IT infrastructure and the client IT infrastructure. It shows a typical client integrating with the feed and highlights relative responsibilities owned by VirtualSite Solutions and those owned by the client.


Client-side data mining process likely executed as a scheduled batch process.
Client Device Data Mining Process
Device Data Feed
Client-side data store for which the client is responsible, including backup and recovery processes.
Client Data Store
Queued Device Data
Device Data Queuing Service
Client Business System
Client Business System
Client Business System
Client Responsibility
VirtualSite Solutions Responsibility
Client-side business processes integrate directly with client-side data store containing staged device data.

API design

The API is ***not*** designed as an ad hoc query API (for example, existing VisionLink web services). For example, you will not find interfaces such as “getLocationForSerialNo” or “getDiagnosticsForDateAndSerialNo”.

The API is designed to be a queue-oriented replication mechanism. A client of the VisionLink API reads all VisionLink (VL) messages sent by all equipment to which they have visibility from a single queue. The intent is that the client “mines” every single message sent by their PL-enabled fleet for which they have expressed interest and then secures that message within their IT infrastructure as they see appropriate. The API bookmarking “next buffer” mechanism enables a client to efficiently retrieve delta changes to their queue data set.

VisionLink data visibility

A VisionLink API is created for each client (dealer or dealer customer) who uses the API. For a dealer, the dealer’s unique ID (internal database identifier for the dealer in the dealer storefront schema) is attached to this publisher. For a dealer customer, the customer’s UCID (internal database identifier for customer) is attached to the publisher.

The API implements a rolling 14-day window for providing all data as reported into VisionLink. The client system retrieving this data is expected to store it within a period of 14 calendar days. If there is a desire to retrieve data beyond the standard 14-day window, a separate request to VisionLink support can be submitted, which will then be handled on a case-by-case basis.

Following is a description of how the API queuing system determines what data is placed into which queue:

1. As the API queuing process receives VisionLink data, an attempt is made to discover the equipment on which the VisionLink module is deployed.
2. Once the equipment is located, the system attempts to identify active VisionLink subscription ownership for that equipment. Subscription level and ownership determines whether or not data from a specific message is queued to a specific client queue. A client can access all API topics, but not receive data for all topics in their queue. Accessing an API is an expression of what VisionLink data a client is interested in. However, the VisionLink subscription level determines whether or not feed access is honored.
3. The VisionLink Asset Visibility rules also apply for the VisionLink API.

REST service

The VisionLink API service is a REST (REpresentational State Transfer) -ful service. REST is not a technology but an architectural style based on HTTP. Most simply put, REST is a web interface that uses XML over HTTP. A client using a REST service sends and receives XML through HTTP GET, PUT, POST, and DELETE operations. REST services can also offer operations such as options, and head, but are less common.

REST differs from web services based on SOAP (Simple Object Access Protocol) in a variety of ways. SOAP services are based on a series of WS-* standards such as WS-Reliable Messaging, WS-Security, WS-Addressing, and WS-Transaction. These standards prescribe to the format of XML messages required to invoke service operations as well as accomplish various features such as security and transactionality.

REST services are sometimes referred to as resource-based as opposed to operation-based as SOAP-based services are classified by some. A REST resource can be addressed using the URI (Uniform Resource Identifier). In the case of the VisionLink API the URI is:

URI type	URI
(production URI)	https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/SMULoc/0

The user is requesting version 4 of the VisionLink API service resource smuloc (service meter and location topic) from the top (0) of the queue named testqueue. For more information, see A closer look at the API service URI, page 12.

Why REST for the VisionLink API?

One of the advantages for using the VisionLink API is that new device data can be added to the API with minimal IT investment. Through one generic service in the VirtualSite Solutions IT infrastructure, new data can flow to a client with no changes to server-side code.

Another advantage is to make the API easy to use, providing as low a barrier to entry as possible. A user from any standard web browser can experiment with the VisionLink API service, as long as they have valid API login credentials. As a user becomes familiar with the service, they can move toward more advanced uses of the service.

A closer look at the API service URI

An example API URI is:

https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/SMULoc/0

This URI can be deconstructed as follows:

protocol://address/context root/version/service/queue name/topic name/queue position:

Component Name	Description
Protocol - https	Protocol over which this request flows. API service resources are served through secured HTTP to protect the user ID and password that flow across the wire for each request.
Address	Address or DNS name of the API service
Context root - API Service	APIService This is referred to as the context root of the service and can simply be thought of as a further qualification of the service address.
Version – v4	The version of the resource being requested. Over time, it is likely that changes will occur to API interfaces. This version allows for the coexistence of legacy interfaces beside newer versions of the same interface.
Service - feed	Service from which data is being requested.
Queue name - TESTQUEUE	Device data is collected into queues. For example, a dealer named abc123 would be assigned a queue named abc123, which would be used in the URI requesting API service.
Topic name - SMULOC	Device data is organized into logical groupings called topics. The smuloc topic contains service meter and location data. Other topics include event, diagnostic, and fuel. Topic data can be extracted from one or many device message types.
Queue position - 0	Each message that flows over the VisionLink API is assigned a unique ID. By specifying a queue position, the client is directing the service to return data beyond the ID provided in the URL. Zero can be used to start retrieving data from the top of the queue. VisionLink API are FIFO (first in, first out) queues.

This address can be entered directly into a standard browser address dialog. The resulting XML appears in the browser window if appropriate user privileges have been granted to the login credentials issuing the request. The queues present in the **Queues** drop-down list are those to which the user's login credentials have been granted access.

Current API

Here is a list of all current API and a brief description of the kind of data each exposes.

***Note –** The URL shown for each API contains the API version that was current at the time of publication. Your API version may be different.*

For additional details about individual fields, refer to the actual schema documents pertaining to each API (outlined in Accessing VisionLink API services, page 6).

Diagnostics Data

One of the three faults generated on-board the ECM. A diagnostic refers to a process within an electronic control module (ECM) of detection, display, and/or storage of abnormal asset/engine information.

Primarily provides the following information:

- The severity level of the diagnostic (0=Unknown, 1=Low, 2=Medium, 3=High)
- The MID (machine component identifier) reporting the event
- The FMI (failure mode indicator) identifies the diagnostic
- The CID (component identifier) describes the on-board sensor that reported the diagnostic

  (Both FMI and CID are internal details of how the Caterpillar Data Link polls the diagnostic and can be cross-referenced to the Service Information System (SIS) to locate troubleshooting steps)
- The timestamp (in UTC) when the diagnosticwas generated

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/Diagnostic/0

Digital Switch Status

Delivers all relevant status information about the power mode and digital switch turned on/off on the PLM module.

Communicates the particular status information of the ECM.

The switch active flags correlate to one of the four digital switches that are allowed on specific devices (or one of up to seven digital switches for specific devices). If any of the bytes that it defines is set to “Yes”, this indicates that a particular condition has occurred since the last report of this status byte in either the status message or SMU report. The state change is relative to the digital switch configuration that has been configured on the device.

The pending flag indicates whether an event or diagnostic is queued when this message was sent.

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v3/feed/testqueue/DigStatus3/0

Engine Parameters

Delivers all the information gathered by the Engine ECM.

Primarily provides the following information:

- Maximum theoretical Fuel Burned by the asset when under full load
- Idle Fuel consumed when the asset is in neutral and none of the implements is working
- Actual fuel consumption by the asset
- Total number of engine starts
- Total number of engine revolutions

The uom indicates the unit of measure for each value field found in the response from the API. The data feed provides the value and unit of measure in case any conversions (for example, gallons to liters) must occur. Count indicates that a value is a numeric count of occurrences.

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v6/feed/testqueue/Engine/0

Engine Start/Stop Time

Delivers a list of engine start and engine stop times for the asset.

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v3/feed/testqueue/StartStop/0

Event Data

Delivers ECM event data to VisionLink.

Primarily provides the following information:

- The event severity level (0=Unknown, 1=Low, 2=Medium, 3=High)
- The MID (machine component identifier) reporting the event
- The event identifier (also known as EID)
- The event occurrence count

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v5/feed/testqueue/Event/0

Fleet Summary

Delivers a high-level look at the current status of your fleet.

Primarily provides the following information:

- Current latitude/longitude of the asset
- Address (reverse-geocoded address based on current latitude/longitude; blank if no address lookup can be made)
- Last location (timestamp of the last time the asset's location was reported, in UTC)
- Last reported (timestamp of the last time the device reported, in UTC)
- Fuel percent remaining
- Runtime hours
- Lifetime fuel consumed
- Last known status

URL: https://legacy.myvisionlink.com/APIService/AEMP/v1/Fleet

Fuel Information

Delivers all the information gathered by the Fuel ECM.

Primarily provides the following information:

- Actual fuel consumption by the asset
- Percentage level of remaining fuel for an asset

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/Fuel/0

Geo Fence Alert

Delivers geofence entry and exit, using geofences dispatched to the device.

PL321/PL121 devices support one inclusive, up to five exclusive geofences, and a one-time fence at any time. Each type of fence (inclusive, exclusive, time) can be active (enabled) or alarming (currently being violated) at any time. The remaining flags indicate whether the device was currently experiencing a satellite blockage (the device has no view to a satellite) or if the master disconnect switch was used.

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v6/feed/testqueue/FenceAlert/0

SMU Location

Delivers the equipment usage by hours since the PLM module was turned on or was last reset in addition to the latitude and longitude of the asset on which the telematics device is mounted.

URL: https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/SMULOC/0

Example code

To view example Java and .NET C# code, go to:

https://legacy-apidocumentation.myvisionlink.com

You can download two zip files from the site. Examples range from simple connectivity and topic access to more advanced examples illustrating techniques for parsing XML response.

Example code

Java

The Java code example zip file contains four examples:

- One validating (Castor) XML parsing example.
- Two non-validating (Castor and XStream) XML parsing examples.
- A quick start example that illustrates simple service connectivity and navigation:
  - Quick start
  - XML serialization

Refer to the Readme.txt in the zip file and carefully read the code comments.

C#

The C# zip file contains two sample solutions:

- DataFeedQuickStart – QuickStart.cs illustrates simple service connectivity and navigation.
- DataFeedSerialization – FeedSerializerMain.cs illustrates service connectivity, navigation, and XML parsing through using Microsoft® XSD serialization tooling.

Invoking the service

Since the service is accessible through an HTTPS GET operation, a user can access the service through a standard web browser. This can be valuable in order to understand data content and navigation.

However, the API service is designed to be a program-to-program integration technology. Programmatic access to the service can be broken into the following steps:

1. Opening a connection.
2. Providing authentication.
3. Executing the "GET" operation.
4. Parsing the result.
5. Getting the next message buffer.

This section shows .NET C# and Java examples.

For more information, see Quick start examples, page 27. The sections below show a sample of the complete examples and describe their function.

The queue name TESTQUEUE is available in the Production API environment and serves as a test queue from which clients can access data as they experiment with the VisionLink API.

Opening a connection

Most, if not all, modern programming languages provide functionality that allow for opening an HTTP connection. To open the HTTPS connection, the client program must have access to the public SSL certificate available at the appropriate API address.

Java

Note – *Address (https://legacy.myvisionlink.com/APIservice/CATdataTopics)*

```
URL url = new URL(address);
HttpURLConnection urlConnection = (HttpURLConnection)
url.openConnection();
```

C#

Note – *url (https://legacy.myvisionlink.com/APIservice/CATdataTopics)*

```
HttpWebRequest request = WebRequest.Create(url) as
HttpWebRequest;
```

Providing authentication

The API service is secured through HTTP Basic Authentication. Access to the TESTQUEUE is through using any VisionLink logon credentials.

Java

```
BASE64Encoder aEncoder = new BASE64Encoder();
String encodedString =
(aEncoder.encodeBuffer(new
String(getUsername().concat(":").concat(getPassword())).getBytes
()));
encodedString = encodedString.substring(0,encodedString.length()
- 2);
urlConnection.setRequestProperty("Authorization", "Basic "+
encodedString);
```

C#

```
request.Credentials = new NetworkCredential(login, password);
```

Executing the GET operation

The API service supports only the HTTP GET operation. The GET is executed as follows.

Java

```
urlConnection.setRequestMethod("GET");
urlConnection.setDoOutput(true);
urlConnection.connect();
BufferedReader reader = new BufferedReader(new InputStreamReader
(urlConnection
.getInputStream()));
```

C#

```
response = (HttpWebResponse) request.GetResponse();
```

Parsing the result

Validating versus non-validating parsers

An XML document can be parsed in one of two modes: validating or non-validating.

Validating

In validating mode, another document provides the XML parser with rules that define the content of a properly formed XML document. Some of these rules might include:

- The number of occurrences of a specific field that should be present.
- The data type of a specific field: integer, string.
- The length of a stringfield.

- The appropriate values that a field should contain.
- The appropriate tag names the document should contain.

The validation document can take the form of a DTD (Document Type Definition) or an XSD (XML Schema Definition). Many more recent XML interfaces are accompanied with an XSD as XSD technology provides a number of benefits over the older DTD validation document style.

XSD documents will be provided for the VisionLink API.

A benefit of an XSD is that you can use off-the-shelf tools to parse and validate the content of an XML schema using an XSD. In addition, tools exist that can generate language specific (for example, Java, C#) objects containing the appropriately typed fields (for example, long, int, string) to receive XML content.

Non-validating

The user of an XML document can choose to trust that the provider of a XML document provides a well-formed document (where well-formed could be defined as conforming to example XML or interface documentation) and not use the XSD at all. In this mode of operation, you either manually code your own validations or skip the validation altogether. You would also need to create codes that can extract content from an XML response. There are off-the-shelf frameworks that make non-validating parsing much easier (for example, Stream in the Java space).

The quick start examples provided in this document do not show how to parse the VisionLink API response.

VisionLink API schema

An XSD is provided for each VisionLink API interface, for example, service meter, location, fault, and event.

For sample schemas, go to: https://legacy-apidocumentation.myvisionlink.com

Using XSD with Java

The code examples illustrate the use of two popular open source frameworks for parsing XML.

Both Castor and XStream examples can be executed from within the example class:

```
example.FeedObjectSerializationExampleV2
```

Comments within the class contain instructions that show the user how to activate examples illustrating the use of each framework.

Castor

Castor is a framework that can parse and validate XML using XML schema.

Castor features utilities that can be used to generate Java serialization codes. The code examples do not use this feature but rather illustrate a simple configuration scheme easily modified to support new API topics. Castor configuration can be found in the example project comfit/castor folder. The following configuration files can be found:

- castor.properties – This properties file provides configuration that enables Castor integration with SAX parsers.
- castor-common-mappingV2.xml – Contains mappings for XML common to multiple API topics, for example, location, equipment.
- castor-mappingV2.xml – Primary Castor config file that includes common config and topic specific config sufficient for a non-validating parser implementation.
- castor-smuloc-validatingV2.xml – Primary Castor configuration file that includes common configuration and topic specific configuration sufficient for a validating parser implementation.
- castor-smuloc-mappingV2.xml – Mapping configuration for service meter location API topic.
- mapping.xsd – XML schema defining castor XML config syntax.

XStream

XStream is a high-performing easy-to-use XML framework. However, XStream does not offer XML validation via XML schema.

To help in the use of XStream for processing VisionLink API responses, tshe Java zip file contains a configuration driven XStream implementation.

XStream configuration can be found in the example project config/xstream folder. The following configuration files can be found:

- feed-xstream-mappingV2.xml – This configuration breaks up API mapping into:
  - Root node (top level API response XML node)
  - Header node (mapping for the XML node containing information common between all API topics)
  - Message node (mapping for message node)
  - Body nodes (mapping for message body content)

The feed-xstream-mappingV2.xml configuration file provides configuration for the classes that can be found in cat.em.feed.xstream. If you do not want to use these XStream wrapping/configurable classes, the classes can be used as examples of how to use the XStream parser API.

Using XSD with Microsoft C#

The Microsoft XSD utility can considerably ease the task of XML processing.

Note – *These instructions apply to Microsoft Visual Studio 2005. Depending on the version you are using, your procedure may differ slightly.*

To execute the Microsoft XSD utility from the command line:

1. Go to the Visual Studio Command prompt (select **Windows Start** / **All Programs** / **Microsoft Visual Studio 2005** / **Visual Studio Tools** / **Visual Studio 2005 Command Prompt**.)
2. At the command line, enter the following example:
   xsd /c /l:cs CommonSchema.xsd ServiceMeterLocation.xsd
3. Access the VisionLink API schema from the Internet and then place them in a directory that is visible to the utility:
   https://legacy-apidocumentation.myvisionlink.com

XSD utility from within Visual Studio

To configure Visual Studio to generate C# codes for processing the VisionLink API:

1. Add the XSD utility to the **External Tools** menu (select **Tools** / **External Tools** / **Add**).
2. Set the title to something suitable; for example, **GenerateAPISMULoc**.
3. Set command to **c:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\ xsd.exe** (assumes you are at Visual Studio 2005).
4. To display tool output in Visual Studio, select the **Use Output Window** check box.
5. Set arguments to **/c /l:cs CommonSchema.xsd ServiceMeterLocation.xsd**. Use the /n:Feed.V2 option to control the namespace in which the generated class is placed.
6. Set Initial Directory to the directory you want the CS file to be deposited; for example, **$(ItemDir)**.
7. Put copies of the VisionLink API schema into the chosen directory. For more information about acquiring the API schema, see Invoking the service, page 17. If you selected **$(ItemDir)** as your Initial Directory, you can place XSD files directly in the solution root directory.
8. To run it, select **Tools** / **GenerateAPISMULoc**.
9. The generated serialization .cs class is not immediately added to your project. The first time you run it, you need to add CommonSchema_ServiceMeterLocation.cs, for example, to the project. To do this, from the shortcut menu select **Add** / **Existing Item**.

You must still create custom code to use the generated mapping class. Refer to the EMFeedSerializer.cs example provided in the C# examples zip file.

Get the next message buffer

Every API topic response contains a <nextBuffer> tag. This tag contains a URL that, if invoked, returns the next buffer of API messages for a specific queue and topic. The blue highlighted tag value below could be copied and pasted into a browser address entry to retrieve the next buffer.

When accessing the API programmatically, correct use of this value is essential to client performance as well as to the preservation of service performance.

After each successful service access, the client application should save the next buffer URL value to a database table, for example, and be used in the next API access. Managing the nextBuffer this way guarantees that only the incremental additions to the queue since the last access are retrieved, which makes the service more available and responsive by improving client performance and reducing the load on the service.

***Note –** The version 2 interface added the moreData tag. The moreData tag was added to prevent clients from "spinning" or retrieving a single message on every request. More data will return false if there are not enough messages to fill the message buffer size configured for a specific queue. The default buffer size is 500. If moreData indicates false, the client should not invoke the URL indicated in the nextBuffer URL tag until the next scheduled client invocation interval. For example, if a client application is scheduled to invoke the API every 15 minutes and a moreData false is returned, the client should wait 15 minutes until invoking the URL returned in the last successful invocation. If moreData returns true, the client can immediately invoke the URL indicated in the nextBuffer tag (a one-second wait would be appropriate).*

```
    <?xml version="1.0" encoding="UTF-8"?>
    <smuLoc:topic
xmlns:smuLoc="http://www.cat.com/em/feed/v4/ServiceMeterLocation"

     xmlns:feed="http://www.cat.com/em/feed/v4/Common"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.cat.com/em/feed/v4/ServiceMeterLocatio
n
https://emfeed.rd.cat.com/EMFeedFiles/schema/v4/ServiceMeterLocation.
xsd
    <nextBuffer>
    <url>
    https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/
    testqueue/SMULoc/0
     </url>
    <moreData>true</moreData>
    </nextBuffer>
    <message>
      <header>
        <messageId>4967366</messageId>
        <masterMsgId>4184753</masterMsgId>
        <equipment>
          <serialNumber>1CF08874</serialNumber>
          <make>ING</make>
        </equipment>
        <moduleCode>DATA_FEED_TEST08874</moduleCode>
        <moduleTime>2008-08-08  17:38:08</moduleTime>
        <receivedTime>2008-08-08 17:38:16</receivedTime>
      </header>
      <location>
        <latitude>34.21137</latitude>
        <longitude>-84.13144</longitude>
        <validity>V</validity>
      </location>
    </message>
</smuLoc:topic>
```

Synchronize access to a specific topic (for example, smuloc) between multiple threads. For example, if a client wants to execute API access in multiple threads, access to the nextBuffer bookmark should be synchronized between the threads to ensure that duplicate messages are not delivered to the client. That is, for a specific topic (for example, smuloc), only one client thread should execute requests at any occasion (that is, single-thread topic requests).

Processing topic responses on the client-side can be multi-threaded. A single topic reading thread could dispatch responses for processing to multiple client-side processing threads. In addition, the client could create multiple topic reading threads provided that each topic reading thread reads only one topic at a specific time. For example, individual topic reading threads for smuloc, fuel, and fault could be executing at the same time.

***Note –** The data API is stateless. Accesses to your queue do not modify the state of your queue. You can access the same queue bookmark repeatedly. If the previous invocation returned moreData=false, the next buffer URL returned in that request response should still be used for the next invocation. If the previous invocation resulted in a 404 (No Data Found) exception, the URL that produced the 404 error should be used to invoke the service again on your next scheduled invocation. Your persisted bookmark should only be advanced using the value returned in the URL field of nextBuffer resulting from a successful API invocation.*

Java

```
      FeedReader reader = new FeedReader(APPLICATION_ID, PASSWORD);
      String response = null;
      BufferedReader breader = reader.getFeedReader(bookMark);
      NextBuffer buffer = new NextBuffer();
      buffer.setMoreData(true);
      while (breader != null && buffer.isMoreData()) {
      response = readResponse(breader);
      System.out.println(response);
        buffer = findNextBuffer(response);
        bookMark = buffer.getUrl();
        System.out.println(bookMark);
        saveBookMark(buffer.getUrl());
        if (buffer.isMoreData()) {
        breader = reader.getFeedReader(bookMark);
      }
    }
    privatestatic NextBuffer findNextBuffer(String response) {
    NextBuffer buffer = new NextBuffer();
      String nextBufferTag = extractTagValue(response,NEXT_BUFFER,NEXT_BUFFER_END);
      buffer.setUrl(extractTagValue(nextBufferTag,URL,URL_END));
      String boolString = extractTagValue(nextBufferTag,MORE_DATA,MORE_DATA_END);
      buffer.setMoreData(Boolean.parseBoolean(boolString));
      return buffer;
    }
```

C#

```
HttpWebResponse response = MakeRequestWithBasicAuth(address, APPLICATION_ID,
PASSWORD);

NextBuffer buffer = null;
string xml = null;
Boolean moreData = true;
while (response != null && moreData)
{
xml = newStreamReader(response.GetResponseStream()).ReadToEnd();
Console.Out.WriteLine(xml);
buffer = FindNextBuffer(xml);
address = buffer.Url;
moreData = buffer.MoreData;
SaveBookMark(address);
response = MakeRequestWithBasicAuth(address, APPLICATION_ID,
PASSWORD);
}
privateNextBuffer FindNextBuffer(string xml)
  {
    int start = xml.IndexOf(NEXT_BUFFER);
    int end = xml.IndexOf(NEXT_BUFFER_END);
    if (start > -1 && end > -1)
  {
    string nbuff = xml.Substring(start + NEXT_BUFFER.Length, end -
    start -
     NEXT_BUFFER_END.Length + 1);
      start = nbuff.IndexOf(URL)+URL.Length;
      end = nbuff.IndexOf(URL_END);
      string url = nbuff.Substring(start, end-start);
      start = nbuff.IndexOf(MORE_DATA) + MORE_DATA.Length;
      end = nbuff.IndexOf(MORE_DATA_END);
      string moreDataStr = nbuff.Substring(start, end - start);
      Boolean moreData = Boolean.Parse(moreDataStr);
      returnnewNextBuffer(moreData, url);
    }
    else
    {
      thrownewException("No next buffer tag found.");
    }
}
```

Detecting duplicate messages

Each API message contains a <messageId> tag. The client to detect duplicate message delivery can utilize this tag. Depending on how the client manages maintenance of the nextBuffer value, the client could receive duplicate messages.

To determine if a specific message was previously processed, use the combination <messageID>, <masterMsgID>, <timeStamp>, and <receivedTime>. These items together uniquely identify a record.

Tying messages together

Each API message also contains a <masterMsgId> tag. This value allows for the association of messages across multiple API topics. For example, a message received in the smuloc topic with masterMsgId 12345 can be associated with a message received in the event API topic with the same masterMsgId 12345. That is, an event occurred at a specific location. The event topic message can be tied to a located provided in the smuloc topic.

Quick start examples

Java quick start example, version 2.0

cat.em.feed.example.QuickStartExample2

```
package cat.em.feed.example;
  /*
  * Copyright (c) 2010 VirtualSite Solution LLC. All Rights
  Reserved.
  *
  * This work contains VirtualSite Solution LLC.  Unpublished
  * proprietary information which may constitute a trade  secret
  * and/or be confidential. This work may be used only for  the
  * purposes for which it was provided, and may not be  copied
  * or disclosed to others. Copyright notice is  precautionary
  * only, and does not imply publication.
  */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.StringWriter;

import cat.em.feed.commonv2.NextBuffer;

public class QuickStartExample2 {

  /*
  * Insert your VirtualSite Solution LLC non-expiring application
  id here.
  */
 private static String APPLICATION_ID = "XXXXX";


  /*
  * Insert your VirtualSite Solution LLC non-expiring application
  id password
  * here.
  * For experimentation with the "TESTQUEUE" you can provide your
  own personal
  * cws user id and password.
  *
  * By default this example hits our production TESTQUEUE. If you
  change the
  * example to utilize QA and are experimenting with your personal
  CWS account
  * be sure to provide your QA CWS  password.
  */
 private static String PASSWORD = "XXXXX";

 private static String NEXT_BUFFER = "<nextBuffer>";
 private static String NEXT_BUFFER_END = "</nextBuffer>";
 private static String URL = "<url>";
 private static String URL_END = "</url>";
 private static String MORE_DATA = "<moreData>";
 private static String MORE_DATA_END = "</moreData>";

   /**
  * @param args
  */
 public static void main(String[] args) {
  /*
  * QA address.
  */
  //String bookMark =
  "https://legacy.myvisionlink.com/APIService/CATDataFeedTopics/v
  2/feed/
   TESTQUEUE/SMULOC/0";
  /*
  * Prod address.
  */
String bookMark =
"https://legacy.myvisionlink.com/APIService/CATDataTopics/v2/feed
/testqueue/
 SMULoc/0";


FeedReader reader = new FeedReader(APPLICATION_ID, PASSWORD);

String response = null;
BufferedReader breader = reader.getFeedReader(bookMark);
NextBuffer buffer = new NextBuffer();
buffer.setMoreData(true);
while (breader != null && buffer.isMoreData()) {
    response = readResponse(breader);
    System.out.println(response);
    buffer = findNextBuffer(response);
    bookMark = buffer.getUrl();
    System.out.println(bookMark);
    saveBookMark(buffer.getUrl());
    if (buffer.isMoreData()) {
    breader = reader.getFeedReader(bookMark);
    }
  }
}


   /**
  * @param breader
  * @return
  */
 private static String readResponse(BufferedReader breader)  {
try {
StringWriter swriter = new StringWriter();
BufferedWriter writer = new  BufferedWriter(swriter);
String line = breader.readLine();
while (line != null) {
  if (line != null) {
  writer.write(line);
  writer.newLine();
  }
  line = breader.readLine();
  }
  writer.close();
  breader.close();
  swriter.close();
  return swriter.toString();
  } catch (IOException e) {
  throw new RuntimeException(e);
  }
}


   /**
  * @param response
  * @return
  */
 private static NextBuffer findNextBuffer(String response) {
NextBuffer buffer = new NextBuffer();
String nextBufferTag = extractTagValue(response,NEXT_BUFFER,NEXT_
BUFFER_END);
  buffer.setUrl(extractTagValue(nextBufferTag,URL,URL_END));
String boolString = extractTagValue(nextBufferTag,MORE_DATA,MORE_
DATA_END);
  buffer.setMoreData(Boolean.parseBoolean(boolString));
  return buffer;
}


   /**
  * @param response
  * @return
  */
 private static String extractTagValue(String tag, String
tagBegin, String tagEnd) {

  int start = tag.indexOf(tagBegin);
  int end = tag.indexOf(tagEnd);
  if (start > -1 && end > -1) {
  return tag.substring(start + tagBegin.length(), end);
  } else {
  throw new RuntimeException("No tag value found.");
  }
}


   /**
  * @param bookMark
  */
 private static void saveBookMark(String bookMark) {
  /*
  * Each feed response contains a nextBuffer attribute that
  represents
  * the url to be invoked to retrieve the next buffer of  messages
  * immediately following the messages represented in the
  response. The
  * last parameter of the bookmark is a number that represents the
  unique
  * message identifier of the last message in the retrieved
  result. This
  * unique id is also contained within each message body  as
  <messageId>.
  *
  * By saving this book mark the app developer can insure  that...
  *
  * A)  If a client side application crash or restart occurs the
  next
  * invocation of the feed will produce new data (i.e. data not
  yet
  * received by the client.
  *
  * B) The response is received as quickly as possible given that
  only
  * new data will be queried.
  *
  * C)  If all clients track their book mark and retrieve the
  minimum
  * number of records required, the stability and responsiveness
  of the
  * service for all clients will be insured.
  *
  * It is very important that the app developer make an attempt  at
  * managing their client side state/bookmark.
   */

  }
}
```

cat.em.feed.commonv2.NextBuffer

```
package  cat.em.feed.commonv2;


public class NextBuffer {
   private String url;
   private boolean moreData;

    public NextBuffer() {
  super();
  }
   public String getUrl() {
  return url;
```

```
        }
         public void setUrl(String url) {
        this.url = url;
        }
         public boolean isMoreData() {
        return moreData;
        }
         public void setMoreData(boolean moreData) {
          this.moreData = moreData;
        }
    }
```

C# quick start example, version 2.0

QuickStart.QuickStartExample

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.IO;
    using System.Net;
    //using EMFeed.v1;
    using EMFeed.v2;

    namespace QuickStart
    {


        /// <summary>
        /// Copyright (c) 2010 VirtualSite Solution LLC. All Rights
        Reserved.
        /// This work contains VirtualSite Solution  LLC.Unpublished
        /// proprietary information which may constitute a trade
        secret
        /// and/or be confidential. This work may be used only for  the
        /// purposes for which it was provided, and may not be  copied
        /// or disclosed to others. Copyright notice is  precautionary
        /// only, and does not imply publication.
        ///
        /// This class illustrates a very simple approach for
        connecting to the EM feed service and reading  XML
         response. The EM feed service is a /// RESTful service
        serving XML over https.
        /// </summary>
        classQuickStartExample
        {



          staticvoid Main(string[] args)
          {
          newFeedReader().Execute();
        }



      }
  }
```

EMFeed.v2.FeedReader

```
    using System;
    using System.Collections.Generic;
    using System.Text;
    using System.IO;
    using System.Net;


    namespace EMFeed.v2
    {
      /// <summary>
      /// Copyright (c) 2010 VirtualSite Solution LLC. All Rights
      Reserved.
      /// This work contains VirtualSite Solution LLC.'s  unpublished
      /// proprietary information which may constitute a trade  secret
      /// and/or be confidential. This work may be used only for  the
      /// purposes for which it was provided, and may not be  copied
      /// or disclosed to others. Copyright notice is  precautionary
      /// only, and does not imply publication.
      /// </summary>
      classFeedReader
      {
        privateconststring NEXT_BUFFER = "<nextBuffer>";
        privateconststring NEXT_BUFFER_END = "</nextBuffer>";
        privateconststring URL = "<url>";
        privateconststring URL_END = "</url>";
        privateconststring MORE_DATA = "<moreData>";
        privateconststring MORE_DATA_END =  "</moreData>";

         /*
        * Insert your VirtualSite Solution LLC non-expiring
        application id here.
        * For experimentation with the "TESTQUEUE" you can provide
        your own personal cws user id
        * and password.
        *
        */
        privateconststring APPLICATION_ID = "XXXXX";

           /*
          * Insert your VirtualSite Solution LLC provided application
          password.
          *
          * By default this example hits our production TESTQUEUE. If
          you change the example to utilize
          * QA and are experimenting with your personal CWS account be
          sure to provide your QA CWS
          * password.
          */
          privateconststring PASSWORD = "XXXXX";

          publicvoid Execute()
          {
           /*
          * QA service url.
          */

          /*
          * Production service url.
          */
          string address =

          "https://legacy.myvisionlink.com/APIService/CATDataTopics/v2
          /feed/testqueue/SMULoc/0";


          /*
          * Make a request to the Feed Service, providing login and
          password for basic authentication.
           */
          HttpWebResponse response = MakeRequestWithBasicAuth(address,
          APPLICATION_ID,
           PASSWORD);

          NextBuffer buffer = null;
          string xml = null;
          Boolean moreData = true;
          while (response != null && moreData)
          {
          xml = newStreamReader(response.GetResponseStream
          ()).ReadToEnd();
            Console.Out.WriteLine(xml);
            buffer = FindNextBuffer(xml);
            address = buffer.Url;
            moreData = buffer.MoreData;
            SaveBookMark(address);
            response = MakeRequestWithBasicAuth(address, APPLICATION_
            ID, PASSWORD);
            }
      }


       /// <summary>
      /// Locates the next buffer url in the xml  response.
      /// </summary>
      /// <param name="xml"></param>
      /// <returns></returns>
      privateNextBuffer FindNextBuffer(string xml)
      {
        int start = xml.IndexOf(NEXT_BUFFER);
        int end = xml.IndexOf(NEXT_BUFFER_END);
        if (start > -1 && end > -1)
        {
        string nbuff = xml.Substring(start + NEXT_BUFFER.Length, end -
        start -
         NEXT_BUFFER_END.Length + 1);
          start = nbuff.IndexOf(URL)+URL.Length;
          end = nbuff.IndexOf(URL_END);
          string url = nbuff.Substring(start, end-start);
          start = nbuff.IndexOf(MORE_DATA) + MORE_DATA.Length;
          end = nbuff.IndexOf(MORE_DATA_END);
          string moreDataStr = nbuff.Substring(start, end - start);
          Boolean moreData = Boolean.Parse(moreDataStr);
          returnnewNextBuffer(moreData, url);
        }
        else
        {
          thrownewException("No next buffer tag found.");
        }
      }

      /// <summary>
      /// Creates the appropriate basic auth credential and invokes
      the provided url.
      /// </summary>
      /// <param name="url"></param>
      /// <param name="login"></param>
      /// <param name="password"></param>
      /// <returns></returns>
      privateHttpWebResponse MakeRequestWithBasicAuth(string url,
      string login, string password)
      {
        HttpWebRequest request = WebRequest.Create(url)
        asHttpWebRequest;
        request.Credentials = newNetworkCredential(login, password);
        HttpWebResponse response = null;
        try
        {
          response = (HttpWebResponse)request.GetResponse();
        }
        catch (WebException e)
        {
          if (((HttpWebResponse)e.Response).StatusCode ==
          HttpStatusCode.NotFound)
          {
             // return null response
            }
            else
            {
            throw e;
            }
        }
        return response;
```

```
    }

     /// <summary>
    /// Saves the current buffer book mark.
    /// </summary>
    /// <param name="xml"></param>
    /// <returns></returns>
    privatevoid SaveBookMark(string address)
    {
       /*
    * Each feed response contains a nextBuffer attribute that
    represents
    * the url to be invoked to retrieve the next buffer of
    messages
    * immediately following the messages represented in the
    response. The
    * last parameter of the bookmark is a number that represents
    the unique
    * message identifier of the last message in the retrieved
    result. This
    * unique id is also contained within each message body  as
    <messageId>.
    *
    * By saving this book mark the app developer can insure
    that...
    *
    * A) If a client side application crash or restart occurs the
    next
    * invocation of the feed will produce new data (i.e. data not
    yet
    * received by the client.
    *
    * B) The response is received as quickly as possible given
    that only
    * new data will be queried.
    *
    * C) If all clients track their book mark and retrieve the
    minimum
    * number of records required, the stability and responsiveness
    of the
    * service for all clients will be  insured.
    *
    * It is very important that the app developer make an attempt
    at
    * managing their client side state/bookmark.
    */
    }
  }
}
```

EMFeed.v2.NextBuffer

```
using System;
using System.Collections.Generic;
using System.Text;

namespace EMFeed.v2
{
  classNextBuffer
  {
    privatestring url;
    privateBoolean moreData;

    public NextBuffer(Boolean moreData, String url)
    {
      this.moreData = moreData;
      this.url = url;
    }

    publicstring Url
    {
      get
      {
        returnthis.url;
      }
      set
      {
        this.url = value;
      }
    }


    publicBoolean MoreData
    {
      get
      {
        returnthis.moreData;
      }
      set
      {
        this.moreData = value;
      }
    }
  }
}
```

Frequently asked questions

How can data already read from a queue be re-retrieved?

The number at the end of the url represents the position in the queue to be read. A typical API URL might look like the following:

https://legacy.myvisionlink.com/APIService/CATDataTopics/v4/feed/testqueue/SMULoc/0

By resetting the API "bookmark" to 0, data can be retrieved from the beginning or head of the queue. As noted above in the VisionLink data visibility section, the 'beginning of the queue' is defined as data that was received by VisionLink 'before 14 calendar days'.

Can the API be queried for a specific equipment serial number(s) or date range?

No, the API is designed as a queue-oriented "data pump" interface. A queue is essentially a bucket into which all subscribed to VisionLink data is placed. The design intent is to provide a very simple interface through which a client could mine all VisionLink data with the intent of securing that data within their own IT infrastructure.

How long will data remain in a queue?

For two weeks (14 calendar days) to guarantee the API responsiveness/performance.

Is all data flowing from the VisionLink-enabled equipment placed in an API user's queue?

Not necessarily. Only data that is normally visible to the owner through the VisionLink user interface will be accessible through the VisionLink API. See VisionLink data visibility, page 10 for more detail.

Does reading messages from a queue remove them from the queue?

No. The API is stateless, which enables a user to repeatedly read the API queue messages without changing the state of the queue.

How frequently can an API be accessed?

The API can be accessed once every second per user login. If there are multiple user logins, they can make parallel requests.

I’m receiving an error code. What does it mean?

Some of the common error codes that can be encountered during execution of an API:

- HTTP 400 - Bad Gateway. This means that the API request did not complete in the specified amount of time (typically 2 minutes).
- HTTP 401 - Unauthorized. This means that the username/password used to make the API request is not valid. The calling system will need to double-check the credentials being used. If necessary, contact VisionLink support for password update requests.
- HTTP 404 - Resource not found. This means that the bookmark that is being used in the API request does not exist.
- HTTP 500 - Internal Server Error. This means that the API request could not be satisfied due to an unexpected error during processing.

How do I get support for API?

If an API request is returning an error code that needs explanation, the best way to get support is to contact your dealer or dealer representative. Send an email outlining the nature of your problem to allow VisionLink experts to evaluate your problem and provide a solution.