Table of Contents

Document Information

This section contains information about this document itself. It does not refer to the API described by this document.

Current Version

- Version: 0.1
- Date: Jul. 24, 2018

Revision History

The following changelog contains the revision history of this document:

Revision	Date	Primary Authors	Overview of Changes
0.1	Jul. 24, 2018	Philip Choi	Initial document creation: defined technical specifications.

Glossary

Term	Definition
API	See "Application Programming Interface"
API Endpoint	The combination of a URL path and an HTTP method used to access a single function of the LOS API. For example, to create and update a draft service request, two separate API endpoints must be accessed.
Application Programming Interface	A well-defined set of software interfaces designed to allow a program to interoperate with other programs.
C360	See "Collateral 360"
Collateral 360	EDR's workflow management software for managing real estate due diligence tasks.
Curl	A command-line application (invoked as `curl`) designed to perform requests using URLs.
Endpoint	See "API Endpoint"
JSON	An abbreviation for JavaScript Object Notation. This is a standard textual data format for representing complex data structures. It is defined at https://www.json.org/
Loan Origination System	The software and computer system through which borrowers' requests for loans are processed by a lender. This system handles steps in this process up to (and sometimes including) disbursement of funds upon approval.
LOS	See "Loan Origination System"
Service Request Form	A collection of data that identifies services to perform on one or more physical locations used as collateral for a loan.
SRF	See "Service Request Form"
URI	An abbreviation for Uniform Resource Identifier. Each URI uniquely identifies a resource using a standardized format.
URL	An abbreviation for Uniform Resource Locator. Each URL is a URI that specifies a resource's location according to the syntax defined in RFC 1738. Website addresses like `http://www.example.com/` are the most common form of URL, but other types exist (*e.g.* FTP addresses).
User Agent	A client application designed to connect to a Web server. Web browsers like Internet Explorer or Chrome are the most common types of user agents, but a client application that interfaces with the LOS API is also a user agent.

Glossary for Multi-Language Support

Term	Definition
ASCII	An abbreviation for American Standard Code for Information Interchange. This is a character encoding where every character is represented in seven bits. Although designed in 1960 to support only the English language, for historical reasons ASCII was the most important character encoding until widespread adoption of the Unicode standard.
Basic Multilingual Plane	Unicode code points from `U+0000` to `U+FFFF` comprise the basic multilingual plane. Code points for most major world languages in current use reside here, though only the most common code points for CJK languages are in this range.

BOM	See "Byte Order Mark"
BMP	See "Basic Multilingual Plane."
Byte Order Mark	An optional character sequence whose presence at the beginning of a string of Unicode text identifies the order in which the bytes of any multi-byte character should be read (this will vary between big-endian and little-endian systems). The byte order mark can be used to distinguish between UTF-8, UTF-16, and UTF-32 encodings.
Case Folding	An operation that converts one alphabetic case to another (for example, uppercase to lowercase in the Latin alphabet used by the English language). Not all languages have the concept of alphabetic case, and in some languages case folding is not always reversible to obtain the original text.
Character	A single letter or symbol in a language. In Unicode, diacritics (such as accents diæreses, circumflexes, *etc.*) are usually considered individual characters, but this varies by character set.
Character Encoding	A scheme for representing the code points of a character set digitally (*i.e.* how to store each code point as one or more bytes).
Character Set	A mapping of characters to their corresponding integer code points.
CJK	An abbreviation for "Chinese, Japanese, and Korean." This list of languages contains the major ideographic scripts in current use.
Code Point	An integer used to identify a single character in a character encoding.
Collation	An algorithm for comparing or sorting characters in a character set.
Emoji	A set of pictorial symbols defined as part of Unicode. These are most commonly used on smart phones.
i18n	An abbreviation for "internationalization" (indicating 18 characters excised from the middle of the word).
l10n	An abbreviation for "localization" (indicating 10 characters excised from the middle of the word).
SMP	See "Supplementary Multilingual Plane."
Supplementary Multilingual Plane	Unicode code points higher than `U+FFFF` comprise the supplementary multilingual plane. This includes less commonly used characters, including historical scripts, various mathematical symbols, and less frequently used characters in the CJK ideographic scripts.
Unicode	A standard character set containing most human languages and symbols in use, as well as historical scripts and emoji. Nearly all modern software that supports the input and display of multiple languages does so by supporting Unicode.
UCS	An abbreviation for Universal Coded Character Set. This is one of two character encoding schemes for Unicode (the other being UTF). The term "UCS" is usually suffixed by a number that indicates how many bytes are used to encode each code point.
UCS-2	A character encoding scheme for Unicode where each code point is encoded in exactly two 8-bit bytes. This encoding cannot represent code points higher than `U+FFFF` (and therefore cannot represent any characters outside the basic multilingual plane).
UCS-4	A character encoding scheme for Unicode where each code point is encoded in exactly four 8-bit bytes.
UTF	An abbreviation for Unicode Transformation Format. This is one of two character encoding schemes for Unicode (the other being UCS). The term "UTF" is usually suffixed by a number that indicates how many bits are used to encode each code point.
UTF-8	A character encoding scheme for Unicode where each character can be encoded in a variable number of 8-bit bytes. Code points 0 to 127 are compatible with ASCII.
UTF-16	A character encoding scheme for Unicode where many code points are encoded in exactly two 8-bit bytes. Higher Unicode code points, however, are encoded in multiple adjacent two-byte UTF-16 units (called "surrogate pairs").

UTF-32	A character encoding scheme for Unicode where each code point is encoded in exactly four 8-bit bytes.

API Introduction

The APIs enable you to use your own custom software to create and manage service requests within Collateral 360, so you can seamlessly integrate Collateral 360's industry-leading features into your own familiar due diligence and workflow control software. Using EDR's APIs can help you achieve time and cost savings, increase efficiency, and reduce troublesome manual data entry errors between different systems.

Handling Errors

Unless a catastrophic error occurs, LOS API responses should always contain an HTTP response status code indicating either success or redirection. Any errors encountered by the API server will be identified by the value of the API response's `responseCode` datum in its top-level `meta` element, which will be equal to an HTTP response status code that describes the error that occurred.

Additional Error Metadata

When an error occurs, the API response's top-level `meta` element will usually contain the following additional data:

- The `errors` element will be an array consisting of all relevant error messages. Each error message will be an associative structure with the following two elements:
  - A `description` attribute (as a string) that contains the text of the error message,
  - A `field` attribute (as a string) that identifies the path to the field in the request that caused the error. If no specific field was the cause of the error, then this will instead be the empty string).

  EDR may add more attributes to each error message over time. Your application should be written in such a way that the addition of these new attributes will not cause an error.

  An example of the `errors` datum follows:

  ```
  "errors" : [
    {
      "description": "The \"First Name\" field is required.",
      "field": "data.firstName"
    },
    {
      "description": "The \"Surname\" field is required.",
      "field": "data.surname"
    }
  ]
  ```

- The `reason` datum will contain a brief textual description of the `responseCode` element's value (for example, if the `responseCode` is 404, then the `reason` may be equal to "Not Found").

  The specific text is subject to change over time, and should be considered informational only. While applications may choose to display or record this text as part of an error message or debugging capability, the application's logic should only rely on the `responseCode` field's corresponding integer value.

Field References in Error Attributes

As described earlier, each element in the `errors` attribute will contain a `field` attribute that identifies the path to the field in the request that caused the error.

The format of this attribute follows the following rules:

1. If no specific field was responsible for the error, then the path will be an empty string.
2. Fields are identified from the root element to the element that caused the error in the string, read from left to right.
3. If an attribute is an array, and the field that caused the error is within one of its array elements, then its zero-indexed position in the array will be added to the path.

4. Field names will be delimited by periods.
5. If a field name contains a literal period or backslash, then each such character will be escaped by a preceding backslash.
6. A field that is not present in the supplied request data may be referred to (*e.g.* if a required field is missing from the request).

An example can be used to illustrate these rules. Given the following request:

```
{
  "meta": {},
  "data": {
    "collection": [
      {
        "goodField": "Good Value"
      },
      {
        "badField": "Bad Value"
      }
    ]
  }
}
```

. . . if we assume that the `badField` element is the cause of an error, then the corresponding error might look like so:

```
{
  "description": "The field value is just plain bad.",
  "field": "data.collection.1.badField"
}
```

If the invalid field was instead named `full.name`, then the `field` attribute would look like this (due to the escaping rules for periods):

```
"data.collection.1.full\\.name"
```

Note the presence of two backslashes, since the single backslash that escapes the period is also escaped by an additional backslash when in a JSON string. When rendered outside the context of JSON, this would appear as `data.collection.1.full\.name` instead.

If the field were instead `full\name`, then the escaping would turn out like this (since the original backslash and the additional backslash the LOS API uses to escape it would both be escaped in the JSON string by their own backslashes):

```
"data.collection.1.full\\\\name"
```

Similarly, when rendered outside the context of JSON, this would appear as `data.collection.1.full\\name` instead.

Warnings

Every API response will contain an array of warnings in its top-level `meta` element. This array will be empty if no conditions that merit a warning occurred while processing the API request.

Each warning will have the same structure as an error, but the presence of one or more warnings will not cause the API response to indicate failure.

Depending on the nature of the condition that caused a warning, the request may be only partially fulfilled, or may be fulfilled completely but with user input that the API server considers to be indicative of a possible mistake.

As a contrived example, if an API client submits data to create a service request with a loan amount of 10,000,000,000 USD, the system might generate a warning indicating that this amount seems unreasonable, but would fulfill the request regardless (since loans are occasionally underwritten for such large amounts).

Common Error Codes

The following list represents the error codes you are most likely to encounter as a user of the LOS API, and what they are likely to represent within the specific context of the LOS API system.

This list is not exhaustive. Error codes are mapped to HTTP response status codes, so refer to the HTTP specification for the full list of possible values.

Name	Code	Description
Bad Request	400	The API request's body content was malformed, or its format was correct but its contents failed to pass validation checks. This may also occur if a dynamic value in the URL was missing or invalid.
Unauthorized	401	The API request either lacked authentication credentials, or the authentication credentials expired. Obtain a new access token and try again.
Forbidden	403	The API request attempts to acces or modify a resource, but the client does not have permission to do so.
Not Found	404	A requested resource was not found. This is most likely to occur when attempting to download a resource via the download API.
Method Not Allowed	405	The specified HTTP method is not allowed for the API endpoint. For example, an HTTP `GET` request was made to an endpoint that only accepts `POST` requests.
Too Many Requests	429	Too many API requests have been issued in a period of time. Wait for a while and then try to resubmit the request. If your application submits multiple requests in a large batch, consider introducing a delay or velocity limitation.
Internal Server Error	500	A technical error occured in the API server. This indicates a problem within the API server itself.
Not Implemented	501	The API request attempted to perform an action that is not yet implemented on EDR's servers. This is only likely to occur during a closed beta test of new features.