



## **CLERK E-CERTIFY**

# **API INTERFACE SPECIFICATIONS**

Version 1.12.1

January 30, 2021

## REVISION HISTORY

---

Version	Date
1.0	9/24/2019
1.1	9/26/2019
1.2	10/7/2019
1.3	10/9/2019
1.4	01/28/2020
1.5	02/06/2019
1.6	03/6/2020
1.7	06/2/2020
1.8	6/20/2020
1.9	6/23/2020
1.10	8/13/2020
1.11	9/21/2020
1.12	10/14/2020
1.12.1	1/30/2021

<b>1</b>	<b>Overview .....</b>	<b>5</b>
<b>2</b>	<b>Architecture and process flow .....</b>	<b>5</b>
<b>3</b>	<b>Outbound connector interfaces .....</b>	<b>7</b>
<b>3.1</b>	<b>Court connectors .....</b>	<b>7</b>
3.1.1	E-Certify custom connector .....	7
3.1.2	GetImageByDocket() .....	8
3.1.3	Docket by Case Request .....	10
<b>4</b>	<b>Inbound connector interfaces .....</b>	<b>11</b>
<b>4.1</b>	<b>Requests requiring payment collection.....</b>	<b>12</b>
4.1.1	API End points .....	12
4.1.2	Integration of Clerk E-Certify for Online Payment Processing .....	13
4.1.3	ClerkecertifyConnector.....	14
4.1.4	AddCourtDocToShoppingCart().....	16
4.1.5	AddORDocToShoppingCart().....	17
4.1.6	Removedocfromcart().....	18
4.1.7	Browser redirection .....	19
4.1.7.1	GetShoppingCartUrl().....	19
4.1.8	Java script popup window .....	19
4.1.9	Embedded shopping cart view .....	22
4.1.10	Payment confirmation .....	27
<b>4.2</b>	<b>ClerkecertifyConnectorCom - Classic ASP Only .....</b>	<b>31</b>
<b>4.3</b>	<b>Requests without payment collection.....</b>	<b>36</b>
4.3.1	API End points .....	36
4.3.2	API name: SendCertifiedCourtImageByDocketID() .....	37
4.3.3	API name: GetCertifiedCourtImagesByDocketsAndSendEmail ().....	39
4.3.4	API name: SendCertifiedORImageByReferenceNo ().....	42
4.3.5	API name: SendCertifiedORImages ().....	43
4.3.6	API name: GetCertifiedImageByReferenceNoAndSendEmail () .....	45
4.3.7	X509 Certification for Authentication .....	47
4.3.7.1	Installing X509 certificate – Install triedata.com certificate on Client machine: .....	48
4.3.7.2	Generate self-signed certificate of the client: .....	50
4.3.7.3	Export public key from the self-signed certificate .....	52



- 4.3.7.4 Setting up CourtService.svc reference.....52
- 4.3.7.5 Consuming CourtService.svc reference .....55
- 5 E-Certify Email Delivery Service..... 57**
- 5.1 Custom email delivery service ..... 58**
- 5.1.1 SendCertifiedCourtDocEmail().....58
  - 5.1.1.1 SendEmailRequest .....58
  - 5.1.1.2 SendEmailResponse .....60
- 6 verification..... 60**
- 6.1 Verify by code..... 60**
- 6.1.1 Browser redirection .....61
- 6.1.2 Java script modal window .....61
- 6.1.3 Embedded view .....64
- 6.1.4 QR Code verification on your own web site .....66
- 6.2 Verify by file upload..... 68**
- 6.2.1 Browser redirection .....68
- 6.2.2 Java script modal window .....68
- 6.2.3 Embedded view .....71

## 1 OVERVIEW

Clerk E-Certify application is built with a number of external interfaces to connect with host CMS systems and vice versa. The system is built with interfaces to allow seam-less integration with Clerk web portal or Clerk applications. Whether the end user is an online user or Clerk's themselves or external government agencies, you are able to use the API specifications to integrate the system with Clerk E-Certify using this API set.

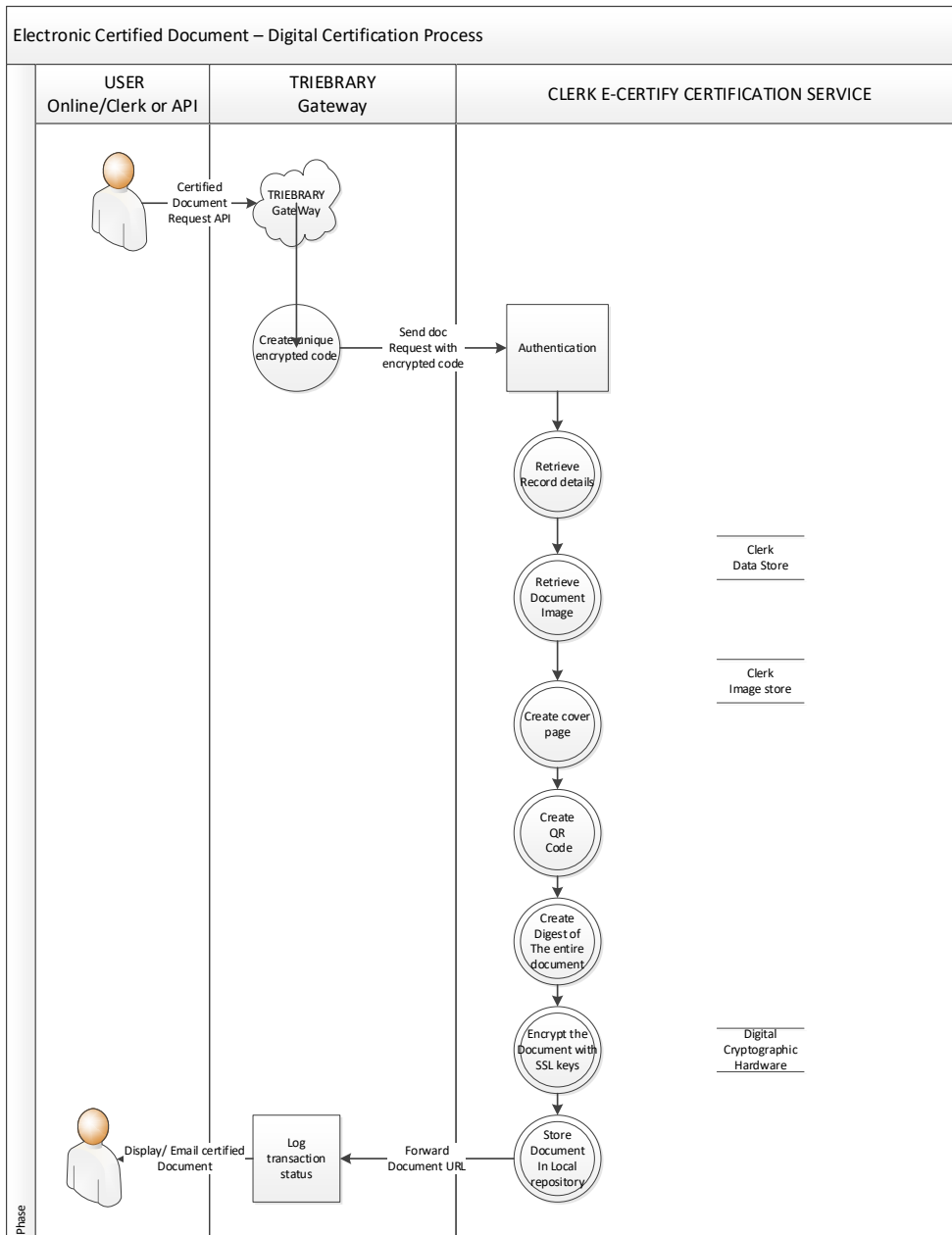
The interfaces are broadly classified into two categories namely outbound connector interfaces and inbound connector interfaces.

Outbound connector interface it utilized by Clerk E-Certify to interface with diverse court CMS/Official Records systems to retrieve and process case information, docket information, docket images, official record grantor/grantee information and official record images.

Inbound connector interface is utilized by external systems such as Clerk CMS applications to interface with Clerk E-Certify. This interface allows external applications to send documents for eCertification, utilize Clerk E-Certify shopping cart to hold and process customer requests, invoke Clerk E-Certify API set directly and so on.

Following sections detail both inbound and outbound connector interfaces.

## 2 ARCHITECTURE AND PROCESS FLOW



There are four distinct components for creating a certified document namely:

1. Search engine – This is the application used by the end user to search for records. This can be a case management system, official records system, Clerk E-Certify search system or any other system used by the end user to identify documents of interest to them. Clerk E-Certify offers its own search engine to search and locate case information as well as Official records ( Please visit <https://www.clerkecertify.com> ) to learn more.



2. Clerk E-Certify shopping cart – Shopping cart is a repository for storing user selected document information. Clerk E-Certify application and external applications can interface with the shopping cart and store user selected document information within the shopping cart. In the event the court applications are utilizing their own search engine, they can forward selected document information to Clerk E-Certify shopping cart using **inbound connector** API's
3. TRIEBRARY Gateway – This is a WCF based, enterprise scale, web services available to process certification requests. Access to the gateway requires multiple levels of security. At a minimum a subscriber code and a token are required to access the gateway. Your subscriber code and token are unique to you and will be different between the test and production environments. Authentication and authorization to access any data using the TRIEBRARY gateway is based on your subscriber code and location (IP Address). Access to court records require additional X509 certificate based mutual authentication.
4. Certification service – Clerk E-Certify certification service is hosted within Clerk's computing environment and uses **outbound connector** interface to connect with court CMS and Official Records systems.

A request to create a certified document is generated by the search engine by invoking one of Clerk E-Certify inbound API's. If the process requires collection of certification charges from the end requestor (public users, party to the case, attorney of record etc.), then shopping cart API's are invoked. If the request is from a non-paying customer such as government agencies and Clerk's, you may invoke direct API's to generate the certified documents.

Outbound API's are utilized by Clerk E-Certify certification service to retrieve dockets, images and grantor/grantee information from the CMS/Official Records repository.

## 3 OUTBOUND CONNECTOR INTERFACES

Outbound connector interfaces are used by Clerk E-Certify application to connect with Court CMS systems to retrieve dockets, images and other data entities. Since there are multiple vendors providing both Official Records and Court Management Systems, Clerk E-Certify offers connectors for most commonly available systems out of the box.

### 3.1 COURT CONNECTORS

Clerk E-Certify offers standard connectors to connect with various CMS systems

Depending upon your confirmation, you may choose one of the standard connectors or utilize a custom connector to retrieve information from the host system.

---

#### 3.1.1 E-CERTIFY CUSTOM CONNECTOR

TRIEDATA Will provide the source code and visual studio project for TRIEBRARYConnector WCF service. You must utilize this project and modify the methods to retrieve necessary information from your system.

Clerk E-Certify signature service will invoke a Clerk’s court service via the TriebraryConnector interface. Clerk’s IT is responsible for creating the service and provide TRIEDATA the URL of the service. The Clerk’s court service will perform one operation:

Operation	Description
GetImageByDocket	This operation returns the image of a given court document as memory stream

### 3.1.2 GETIMAGEBYDOCKET()

This service operation is invoked by Clerk E-Certify signature service to retrieve document images. Clerk’s IT is responsible for creating the service and this operation.

Method name: GetImageByDocket()

Source: Clerk E-Certify signature service

Target: Clerk’s court service

Method: Https Get

Request Format: JSON

Response Format: JSON

Body Style: Wrapped

Endpoints: Supports REST

Here is the data definition for input to the GetImageByDocket () operation:

Index	Field Name	Description	Definition/Values
1	SubscriberCode	A unique identifier associated with the caller	Not Null Max Size: 20 Contact TRIEDATA
2	SubscriberName	Caller name	Not Null Max Size: 30 i.e. “Online User”



3	CaseNo	Case No	Not Null Max Size: 30
4	CaseType	Indicating whether UCN or LOCALCASENUMBER	Not Null, integer UCN = 1, LOCALCASENUMBER = 2, UNKNOWN = 3
5	DocketNumber	Docket number associated with the document	Not Null Max Size: SQL integer (10 chars)
6	RedactionFlag	Indicator whether the docket is a redacted or un-redacted document	Char string, Not Null Max Size: 1 'Y'- redacted 'N' – un-redacted
7	Field1	Reserved field used for document version information	Numeric String, Not Null Max Size: SQL integer (10 chars)
7	Field2	Not used	Not used
8	Field3	Not used	Not used
9	Field4	Not used	Not used
10	Field5	Not used	Not used

#### GetImageByDocketResponse

Here is the data returned from the GetImageByDocketResponse () operation:

Index	Field Name	Description	Definition/Values
1	ResponseCode	Status code	Not Null, integer value Value = 1: success Value >1: exception
2	ResponseDetails	Information for exceptions	Nullable,

			Max Size: 100 If there is an exception, provide details on the error
<b>3</b>	ImageArr	Byte array of the document image	Nullable, if image is not available

### 3.1.3 DOCKET BY CASE REQUEST

This API is used by Clerk E-Certify certification service to retrieve docket information for a user provided case number. This API is applicable only if Clerk E-Certify search engine is used for retrieving case docket information.

API name: GetDocketsByCaseNumber ()

Source: TRIEDATA Clerk E-Certify certification service

Target: CMS API

Method: Http Post

Request Format: Json

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
<b>1</b>	SubscriberCode	A unique identifier associated with the caller	Alphanumeric String, Not Null
<b>2</b>	Subscriber name	Name of the subscriber.	Alphanumeric String, Not Null The subscriber name matches the username within Clerk E-Certify system.
<b>3</b>	Case number	UCN or Local case number	Alphanumeric String, Not Null
<b>4</b>	Case Type	Defines the case type	Alphanumeric String, Not Null Valid values are UCN – Uniform case number LOCALCASENUMBER – Local case number

Docket by Case Response

Response type: Synchronous

Response packet consists of one element namely a list of Docket response or null.

List<Docket> - If the subscriber is authorized to access the case and dockets are available

Null – if the subscriber is unauthorized or case is not available

### 3.1.3.1.1 DOCKET OBJECT

Index	Field Name	Description	Definition/Values
1	Docket ID	Document ID associated with the docket	Alphanumeric String, Not Null
2	Docket description	Description of the docket.	Alphanumeric String, Not Null Maximum length is 100.
3	Docket date	Date associated with the docket	Datetime String, Not Null
4	Pages	Number of pages within the image	Numeric, Not null
5	IsImageAvailable	Indicates whether there is an image associated with the docket	Numeric, Not null Valid values are: 1 – There is an image 0 – No image available
6	IsImagePublic	Indicates if the image is public and can be certified	Numeric, Not null Valid values are: 1 – There is an image 0 – No image available
7	Reserved1	Reserved fields	Unused
8	Reserved2	Reserved fields	Unused

## 4 INBOUND CONNECTOR INTERFACES

This chapter provides technical specifications for the interface between Clerk application and Clerk E-Certify for processing online certified document requests. There are two types of inbound requests.

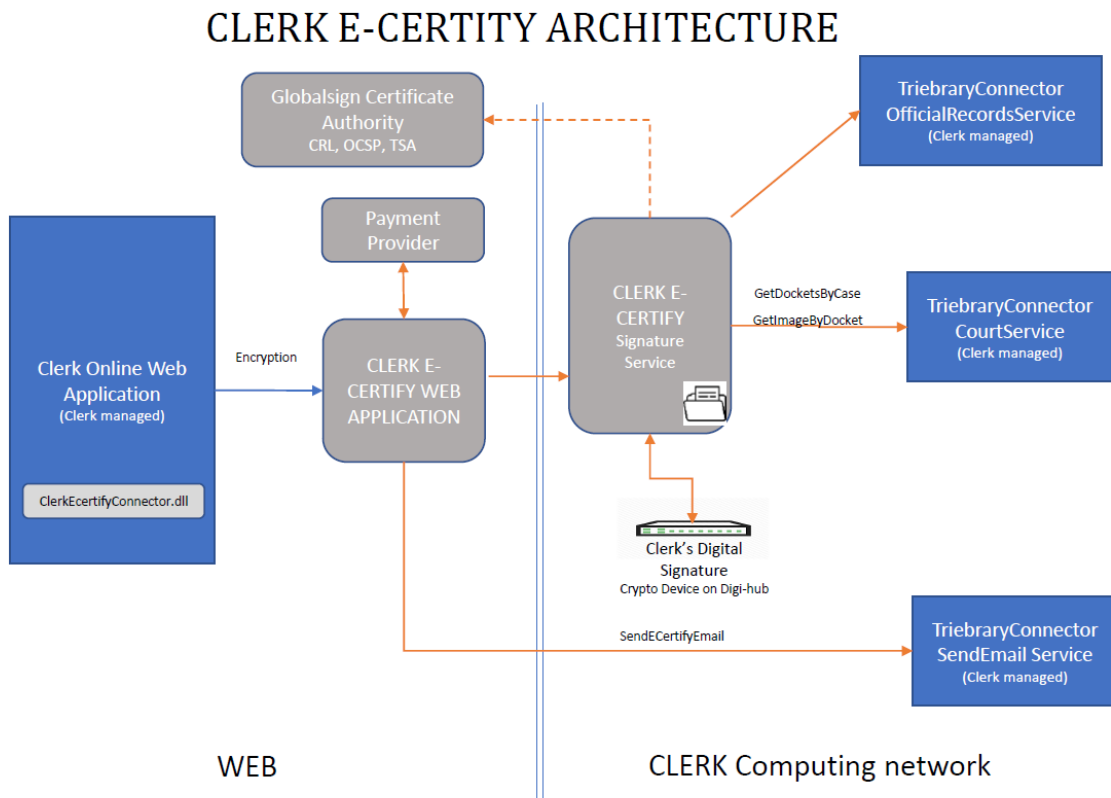
1. Requests requiring payment collection

## 2. Requests without payment processing

If the request requires collection of payment, then the items need to be added to the shopping cart. You can add as many items as needed. After adding the items, you can redirect the user to Clerkecertify.com web site to complete payment. Clerk E-Certify will add its service fee and credit card charges and collect the total amount from the requestor using a credit card payment method.

If the request does not require payment collection, then you may invoke API directly to generate certified documents. The documents will be generated within seconds and the call will return with the location where the files are stored (This will be an IIS based storage within your computing environment).

### 4.1 REQUESTS REQUIRING PAYMENT COLLECTION



Picture 1 – System Overview

#### 4.1.1 API END POINTS

Type	URL
------	-----

Test environment	<a href="https://test.clerkecertify.com">https://test.clerkecertify.com</a>
Production	<a href="https://www.clerkecertify.com">https://www.clerkecertify.com</a>

Subscriber code: Please contract TRIEDATA

Token: Please contract TRIEDATA

---

#### 4.1.2 INTEGRATION OF CLERK E-CERTIFY FOR ONLINE PAYMENT PROCESSING

Clerk E-Certify shopping cart is able to process external document requests. This method is utilized where the end user is searching court documents on Clerk's web site. The Clerk's web application is responsible for authentication of the end user and granting the user permission to access court cases and view dockets.

**SubscriberCode** and **token** must be obtained to interface with Clerk E-Certify. The SubscriberCode and Token is unique and used for encrypting information transmitted between Clerk's web application and Clerk E-Certify. SubscriberCode and token is not included within this documentation. Please contract TRIEDATA to obtain your SubscriberCode and token.

Note: Your SubscriberCode and token may differ between test environment and the production environment.

Online process integration with Clerk E-Certify is a two-step process. In step one, the Clerk's web application sends the docket payload information as an HTTP POST message to Clerk E-Certify web application with a unique identifier. Clerk E-Certify will save this information in its system. In the second step, Clerk's web application redirects the user to Clerk E-Certify shopping cart passing in the unique identifier for completing the payment processing.

After completing step one, there are three ways to launch the shopping cart.

1. Browser redirection - Redirect the user to Clerkecertify shopping cart window using browser redirection
2. Java script pop up - Use Java script-based shopping cart and display the shopping cart as a pop-up window within your application.
3. Embedded shopping cart – You can use Javascript to embed the shopping cart within your own HTML web page without the popup modal window.

Following data elements are transferred to Clerk E-Certify:

- Case number (UCN format)
- Case type (value = "UCN")
- Docket description
- Docket number ('Document ID' and 'Document Version')
- Page count

- Redaction Flag
- Quantity
- Version number (sequence number generated by SQL server)

Clerk’s web application can submit one or more documents to Clerk E-Certify shopping cart, by invoking our API’s multiple times. The Clerk E-Certify application offers following methods for invocation by the Clerk’s application:

Method	Description
AddCourtDocToShoppingCart	This method is used for sending document request and information to Clerk E-Certify
GetShoppingCartUrl	This method returns the URL of the Clerk E-Certify shopping cart
GetCorsShoppingCartReferenceId()	This method returns the CorrelationId of the shopping cart

#### 4.1.3 CLERKECERTIFYCONNECTOR

ClerkecertifyConnector is a windows DLL used for interfacing from Clerk’s web application to Clerk E-Certify. Please contact TRIEDATA to obtain this DLL. Clerk’s web application or download it from Visual Studio Nuget library. Your application can invoke methods within this DLL to allow online users to purchase electronic certified court documents with Clerk E-Certify.

CLERKECERTIFYCONNECTOR is available for download as a Visual studio Nuget package. You may download it from your visual studio-based application.

There are two ways to download it.

Method 1: Use Nuget package manager

1. From your VS project, go to tools -> Nuget package manager -> Manage Nuget packages for solution
2. Browse for “ClerkecertifyConnector” and add it to your project

Method 2: Use Install-package command

1. From your Nuget package manager console, type
2. Install-Package ClerkecertifyConnector

Use these instructions for installing the ClerkecertifyConnector within your visual studio-based application:

1. Contact TRIEDATA to obtain ClerkecertifyConnector.dll or download it from VS - Nuget
2. Add ClerkecertifyConnector.dll as a reference to your Visual Studio project
3. Configure your application.

Here is an example on how to invoke the ClerkecertifyConnector methods and send docket information to Clerk E-Certify

---

## ShoppingCart

---

```
/* Initialize the class */

/* calling application must pass in a unique identifier for each purchase transaction */

ShoppingCart myShoppingCart = new ShoppingCart(UniqueIdentifier, URL,
PublisherCode, SubscriberCode, Token);

OR

ShoppingCart myShoppingCart = new ShoppingCart(UniqueIdentifier, Url,
PublisherCode, SubscriberCode, Token, Firstname, Lastname, EmailId, PhoneNo);

//Firstname, Lastname, EmailID and Phone number of the end user can be passed onto
//clerkecertify application. This will avoid the need for the user to enter this
//information on Clerkecertify shopping cart checkout page

/* next send the docket information to Clerk E-Certify. You must call this method
for each docket ID

    CaseNO (UCN only): 372019CF0000001XXXXXX,
    CaseNoType: UCN,
    Description: "ARREST WARRANT",
    DocumentNo: 12345, (document ID = 12345)
    PageCount: 3,
    Qty: 1, (always 1 copy per document)
    Redactionflag: Y or N
    Field1: 444 (Document version),
    Field2: null (First name),
    Field3: null (Last name),
    Field4: null (Email Id),
    Field5: null (Phone Number)

*/

/* Repeat above step for additional court documents */

/* Finally get the URL for viewing the content of the shopping cart and making the payment */

string url = myShoppingCart.GetShoppingCartUrl();
```

---



A unique identifier to group user requests is always required. Typically, a browser session ID is used as the unique identifier.

In the event the Clerk’s web application is not able to utilize ClerkecertifyConnector DLL library, native methods are available to interface with Clerk E-Certify. Please contact TRIEDATA for details.

#### 4.1.4 ADDCOURTDOCTOSHOPPINGCART()

This service method is used by Clerk’s Web Application to send docket information to Clerk E-Certify application. Clerk application invokes this method, which results in a HTTP Post to Clerk eCertify.

Method name: AddCourtDocToShoppingCart ()

Source: Clerk’s Application

Target: Clerk E-Certify Web Application

Method: Https Post

Index	Field Name	Description	Definition/Values
1	CaseNo	Case number in UCN format	Not Null Max Size: 30
2	CaseType	UCN only	Not Null, Value = “UCN”; always
3	DocketDescription	Document description	Alphanumeric String, Not Null Max Size: 100
4	DocketNo	Docket number associated with the document.	Numeric String, Not Null Max Size: SQL integer (10 chars)
5	PageCount	Represents the total number of pages in each document	Numeric String, Not Null, greater than 0 Integer values
6	Quantity	Integer indicating the number of copies of e-certification	Numeric String, Not Null Always “1”, since Clerk E-Certify shopping cart native function provides ability to adjust qty



7	Redaction Flag	Indicator whether the docket is a redacted or un-redacted document	Char string, Not Null Max Size: 1 'Y'- redacted 'N' – un-redacted
8	Field1	Reserved field, used by Odyssey clients to send document version information.  Combined with Document ID, this field is used to retrieve images from Odyssey system	Numeric String, Not Null Max Size: SQL integer (10 chars)
9	Field2	Not used	Not used
10	Field3	Not used	Not used
11	Field4	Not used	Not used
12	Field5	Not used	Not used

Return value: This method returns the number of unique documents currently loaded in the shopping cart within Clerk E-Certify. Returns -1 if the correlation id refers to a shopping cart that is already processed.

#### 4.1.5 ADDORDOCTOSHOPPINGCART()

This service method is used by Clerk's Web Application to send official record information to Clerk E-Certify application. Clerk application invokes this method, which results in a HTTP Post to Clerk eCertify.

Method name: AddORDocToShoppingCart ()

Source: Clerk's Application

Target: Clerk E-Certify Web Application

Method: Https Post

Index	Field Name	Description	Definition/Values
1	ReferenceID	Instrument number or CFN number of the instrument	Not Null Max Size: 20

2	Description	Description of the instrument such as Marriage license, warranty deed etc.	Not Null, Max size: 100
3	PageCount	Represents the total number of pages in each document	Numeric String, Not Null, greater than 0 Integer values
4	Quantity	Integer indicating the number of copies of e-certification	Numeric String, Not Null Always "1", since Clerk E-Certify shopping cart native function provides ability to adjust qty

Return value: This method returns the number of unique documents currently available for purchase for the given identifier within Clerk E-Certify. Returns -1 if the correlation id refers to a shopping cart that is already processed.

The method authenticates request from Clerk website using the IP Address of the sender (Clerk’s web server back-end IP, not a front-end Client IP). Clerk’s web application can invoke the method multiple times, sending document information one document at a time. Upon successful data validation, Clerk E-Certify proceeds to save requests in its database.

#### 4.1.6 REMOVEDOCFROMCART()

This service method is used by Clerk’s Web Application to send official record information to Clerk E-Certify application. Clerk application invokes this method, which results in a HTTP Post to Clerk eCertify.

Method name: RemoveDocFromCart ()

Source: Clerk’s Application

Target: Clerk E-Certify Web Application

Method: Https Post

Index	Field Name	Description	Definition/Values
1	ReferenceID	Docket Number, Instrument number or CFN number of the instrument	Not Null Max Size: 20
2	DocType	This field indicates whether the document is an Official Record or	Possible values are: <ul style="list-style-type: none"> <li>DocumentTypeEnum.DocumentTypes.COURTDOCTYPE</li> <li>DocumentTypeEnum.DocumentTypes.ORDOCTYPE</li> </ul>

		Court Document. This is an enumerated data type	
--	--	--	--

Return value: This method returns the number of unique documents remaining in the shopping cart after removing the item.

The method authenticates request from Clerk website using the IP Address of the sender (Clerk’s web server back-end IP, not a front-end Client IP). Clerk’s web application can invoke the method multiple times, to remove each individual item. Please note that the user has options to remove items from shopping cart view as well.

---

#### 4.1.7 BROWSER REDIRECTION

After forwarding selected docket/instrument information using API method above, you can redirect the user to view the shopping cart. User can confirm the details and complete the checkout process using a valid US issued credit card. This is one of the ways to integrate shopping cart with your application.

---

##### 4.1.7.1 GETSHOPPINGCARTURL()

This method returns the URL of the Clerk E-Certify shopping cart. The URL query string will include the “UniquelIdentifier” as a parameter.

All you need to do is redirect the user using JavaScript or other means.

Example:

```
window.location.href = “URL”;
```

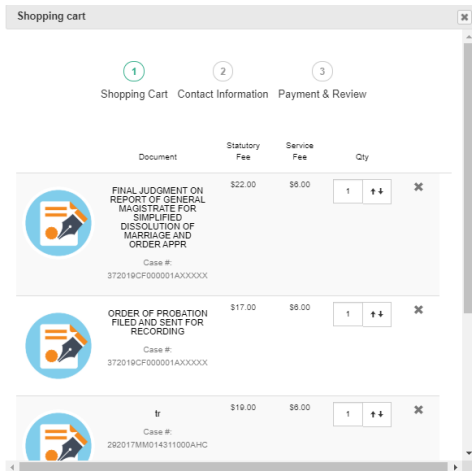
Clerk E-Certify performs following actions upon browser redirect.

1. Retrieves document information posted earlier using HTTP Post.
2. Groups all requests using the unique identifier passed in by the caller.
3. Verify images are available for all requested documents.
4. If images are available, they are automatically added to the shopping cart and user is redirected to the shopping cart view.
5. If one or more images are not available, user has the option to proceed to the cart or go back to search engine.
6. User can perform “checkout” to finalize the transaction and receive the documents in email.

---

#### 4.1.8 JAVA SCRIPT POPUP WINDOW

instead of redirecting the user to clerkecertify.com, you may use Java script to launch the shopping cart as a modal window within your application. Clerkecertify shopping cart will be displayed to the user as a popup window as shown below.



Here are the steps for integrating shopping cart as a popup window

1. Contact TRIEDATA to advise them about your test and production domain. TRIEDATA needs to enable requests coming from your specific domain (For example: browardclerk.org, osceolaclerk.com etc.)
2. Get shopping cart correlation ID
  - a. Initialize your shopping cart
  - b. Add items to the shopping cart as shown in example above
  - c. Call method GetCorsShoppingCartReferenceld to retrieve the correlation ID

```
ShoppingCart myShoppingCart = new ShoppingCart(SessionID, Url, PublisherCode,
SubscriberCode, Token, "First name", "Last name", "Email ID", "Phone number");
    foreach (var rec in model.eCertifyORDoc)
    {
        Var ret =
myShoppingCart.AddORDocToShoppingCart(rec.EncryptedInstrumentNo,
rec.DocumentType, rec.PageCount, rec.Quantity.ToString());
    }
var correlationId = myShoppingCart. GetCorsShoppingCartReferenceld ();
```

3. Load all pre-requisites such as bootstrap, jQuery etc. for the shopping cart. If your application has already loaded these components, then you do not need to load them.

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">

<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css" rel="stylesheet" />

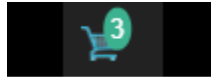
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.form/4.2.2/jquery.form.min.js" integrity="sha384-
FzT3vTVGXqf7wRfy8k4BiyzvbNfeYJK+frTVqZeNDFI8woCbF0CYG6g2fMEFFo/i" crossorigin="anonymous"></script>
```

4. Create a HTML Div element to hold the shopping cart modal window



```
<!--idCart is the placeholder to display number of items in the cart -->
<div id="idCart" style="height:50px; width:50px;padding-top:15px;"></div>
<div id="idCorrelationId" hidden></div>
<!--idCart is the placeholder to display shopping cart -->
<div id="idShoppingCartView"></div>
```

```
<div id="idCart" style="height:50px; width:50px;padding-top:15px;"></div>

<div id="idCorrelationId" hidden></div>
```

```
<script type="text/javascript">
    $(document).ready(function () {
        // Lets get the Cart Image with counts now
        var CorrelationId = $('#idCorrelationId').html();
        var url = ShoppingCartServer-URL +
"/Level2Request/GetCorsCart?CorrelationId="+ CorrelationId;
        $.ajax({
            url: url,
            type: 'get',
            success: function (data) {
                var text = data;
                $('#idCart').html(data);
            }
        });
    });
</script>

// ShoppingCartServer-URL - Please check your documentation for test and
production environments
```

5. Use JavaScript to populate the DIV upon completion of document load (You may use other events as appropriate)

That's all you need to do.







Once you have added new items to the shopping cart, please make sure to refresh the DIV –“ idCart”. This will update the quantities on the shopping cart automatically.

User may click on the shopping cart for checkout, and this will bring up the pop-up window.

---

#### 4.1.9 EMBEDDED SHOPPING CART VIEW

Instead of redirecting the user to clerkecertify.com or displaying the shopping cart as a pop-up modal window, you may use Java script to embed the shopping cart as an inline HTML within your own web portal. This will provide a seam-less look and feel with your application.

Document	Statutory Fee	Service Fee	
 <p>FINAL JUDGMENT ON REPORT OF GENERAL MAGISTRATE FOR SIMPLIFIED DISSOLUTION OF MARRIAGE AND ORDER APPR Case #: 372019CA00002XXXXXX</p>	\$6.00	\$6.00	
 <p>ORDER OF PROBATION FILED AND SENT FOR RECORDING Case #: 562004CF000011AXXXXX</p>	\$22.00	\$6.00	
 <p>CASE MANAGEMENT C Case #: 362019CT500194000ACH</p>	\$12.00	\$6.00	

Summary

Order Total \$58.00

*\* Credit card service fees will apply.*

Proceed to Checkout

Here are the steps for integrating shopping cart as a popup window

6. Contact TRIEDATA to advise them about your test and production domain. TRIEDATA needs to enable requests coming from your specific domain (For example: browardclerk.org, osceolaclerk.com etc.)
7. Get shopping cart correlation ID
  - a. Initialize your shopping cart
  - b. Add items to the shopping cart as shown in example above
  - c. Call method `GetCorsShoppingCartReferenceId` to retrieve the correlation ID

```
ShoppingCart myShoppingCart = new ShoppingCart(SessionID, Url, PublisherCode,
SubscriberCode, Token, "First name", "Last name", "Email ID", "Phone number");
    foreach (var rec in model.eCertifyORDoc)
    {
        Var ret =
myShoppingCart.AddORDocToShoppingCart(rec.EncryptedInstrumentNo,
rec.DocumentType, rec.PageCount, rec.Quantity.ToString());
    }
var correlationId = myShoppingCart.GetCorsShoppingCartReferenceId ();
```

8. Load all pre-requisites such as bootstrap, jQuery etc. for the shopping cart. If your application has already loaded these components, then you do not need to load them.

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">

<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css" rel="stylesheet" />

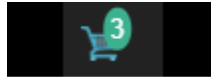
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.form/4.2.2/jquery.form.min.js" integrity="sha384-
FzT3vTVGXqf7wRfy8k4BiyzvbNfeYJK+frTVqZeNDFI8woCbF0CYG6g2fMEFFo/i" crossorigin="anonymous"></script>
```

9. Create a HTML Div element to hold the shopping cart modal window





```

<!--idCart is the placeholder to display number of items in the cart -->
<div id="idCart" style="height:50px; width:50px;padding-top:15px;"></div>
<div id="idCorrelationId" hidden></div>
<!--idCart is the placeholder to display shopping cart -->
<div id="idShoppingCartView"></div>

```

10. Use JavaScript to populate the shopping cart DIV upon completion of document load (You may use other events as appropriate). Please note that after loading the HTML, you need to override the click event and redirect it to your own shopping cart view. This will disable the pop window.

```

<script type="text/javascript">
    $(document).ready(function () {
        updateShoppingCartCounter();
        updateShoppingCartView();
    });

    function updateShoppingCartCounter() {
        // Lets get the Cart Image with counts now
        var CorrelationId = $('#idCorrelationId').html();
        var SubscriberCode = $('#idSubscriberCode').html();
        if ("@Model.Url" != null && CorrelationId != null) {
            var url = "@Model.Url" + "/Level2Request/GetCorsCart?CorrelationId=" +
                CorrelationId;
            $.ajax({
                url: url,
                type: 'get',
                success: function (data) {
                    var text = data;
                    $('#idCart').html(data).promise().done(function () {
                        $('#CECShoppingCartId').on("click", function (event) {
                            event.stopImmediatePropagation();
                            window.location.href = "/Home/Court";
                        });
                    });
                }
            });
        }
    }
}
</script>

// ShoppingCartServer-URL - Please check your documentation for test and
production environments

```

`$('#CECShoppingCartId')` is the DIV displaying current shopping cart count. This DIV is provided and populated by Clerk eCertify. Unless the click event is overridden, it will prompt a pop up modal window for the shopping cart. Above code

overrides the click event and redirects the user to your controller method. You can redirect it to your own controller/action here.

11. Use JavaScript to populate the shopping cart view. You can call below function to load the shopping cart partial view within the div element `idShoppingCartView`.

```

function updateShoppingCartView() {
    // Lets display shopping cart also here
    var CorrelationId = $('#idCorrelationId').html();
    var SubscriberCode = $('#idSubscriberCode').html();
    var url = "@Model.Url" +
"/Level2Request/corsexternalshoppingcartrequest?CorrelationId=" +
CorrelationId + "&CallbackFn=" + "updateShoppingCartCounter";
    $.ajax({
        type: "GET",
        url: url,
        xhrFields: {
            withCredentials : true
        },
        timeout: 10000,
        statusCode: {
            404: function () {
                // Simple not found page, but not
                CORS violation
            }
        }
    })
    $("#idShoppingCartView").html(this.url + " not found");
}

.fail(function (jqXHR, textStatus) {
    // Empty status is a sign that this may be a CORS violation
    // but also check if the request timed out, or that the domain exists
    if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
        $("#idShoppingCartView").html("Unable to launch shopping cart:
" + jqXHR.status + " " + jqXHR.statusText + " error");
        return;
    }
    else {
        $("#idShoppingCartView").html("Please refresh your browser page
again: Error code: " + jqXHR.status + "-" + jqXHR.statusText);
        return;
    }
})
.done(function (data) {
    // Successful ajax request
    $("#idShoppingCartView").html(data);
})

```

That’s all you need to do.

Each time you add a new item to the shopping cart, you need to refresh the div by reloading the contents again. It will reflect new items (or deleted items). User can proceed to checkout from your own web portal without being transferred to Clerkecertify.com.

Users are able to delete items within the shopping cart. Your application may not know if the user has removed items from the shopping cart. Once the user deletes an item, the shopping cart counter needs to be updated. This is accomplished by forwarding the callback function handle as a parameter ("**&CallbackFn=" + "updateShoppingCartCounter"**"). Upon deletion of the item, eCertify will invoke the callback function (**updateShoppingCartCounter**) and this will be able to display current counts.


Once the user checks out using credit card on MyFloridaCounty.com, you can setup hooks within MyFloridaCounty.com to redirect the user to your “Controller/Action”.

You can integrate “Return” button within MyFloridaCounty.com to redirect the user to your own portal. Please contact TRIEDATA to setup the return hooks.

MyFloridaCounty will pass the correlationID associated with your shoppingcart. Using the correlation ID, you can refresh the shopping cart view above, and this time the shopping cart will display payment confirmation view and users are able to download certified documents from your portal. Diagram below illustrates user view.

**Payment Confirmation:**

Payment Reference Order ID: 948319  
 Payment Total Amount: \$60.03  
 Electronically Certified Document Order Id: 7640  
 Email Address to Receive the Order: raghunath.menon@menchaninc.com  
 Order Submitted by: thinkodu, raghunath  
 Order Status Details: Created  
 Current status: System has processed 3 out of 3 requests  
 This window is set to auto update every 30 seconds

Document	Statutory Fee	Service Fee	Status	Certificate
 test	\$0.00	\$0.00		<a href="#">Click here to view certificate</a>

#### 4.1.10 PAYMENT CONFIRMATION

In the event your application is required to update your CMS or Official Records system with payment posting information for online transactions, you may follow steps below to set up your system. Whenever an online payment is received from credit card processing system, Clerk eCertify will be able to forward it to your system. The data packet is sent as a JSON string. Following table details the contents of the JSON object.



1. Create an OrderInfo class in your MVC application (If you are using Java, you can use this model to create your class as well)

```
public class OrderInfo
{
    [JsonProperty("CorrelationId")]
    public string CorrelationId { get; set; }
    [JsonProperty("OrderId")]
    public string OrderId { get; set; }
    [JsonProperty("FirstName")]
    public string FirstName { get; set; }
    [JsonProperty("LastName")]
    public string LastName { get; set; }
    [JsonProperty("Phone")]
    public string Phone { get; set; }
    [JsonProperty("Email")]
    public string Email { get; set; }
    [JsonProperty("TotalAmount")]
    public string TotalAmount { get; set; }
    [JsonProperty("TotalAdminFee")]
    public string TotalAdminFee { get; set; }
    [JsonProperty("TotalTechFee")]
    public string TotalTechFee { get; set; }
    [JsonProperty("CreditCardFees")]
    public string CreditCardFees { get; set; }
    [JsonProperty("OrderDate")]
    public System.DateTime OrderDate { get; set; }
    [JsonProperty("PaymentReference")]
    public string PaymentReference { get; set; }
    [JsonProperty("LineItems")]
    public List<LineItem> LineItems { get; set; }
    [JsonProperty("Field1")]
    public string Field1 { get; set; }
    [JsonProperty("Field2")]
    public string Field2 { get; set; }
    [JsonProperty("Field3")]
    public string Field3 { get; set; }
    [JsonProperty("Field4")]
    public string Field4 { get; set; }
    [JsonProperty("Field5")]
    public string Field5 { get; set; }
}
```

2. And create a LineItem class

```

public class LineItem
{
    [JsonProperty("OrderDetailId")]
    public int OrderDetailId { get; set; }
    [JsonProperty("DocumentType")]
    public string DocumentType { get; set; }
    [JsonProperty("DocIdentifier")]
    public string DocIdentifier { get; set; }
    [JsonProperty("Description")]
    public string Description { get; set; }
    [JsonProperty("CaseNo")]
    public string CaseNo { get; set; }
    [JsonProperty("AdminFee")]
    public decimal AdminFee { get; set; }
    [JsonProperty("TechFee")]
    public decimal TechFee { get; set; }
    [JsonProperty("Quantity")]
    public int Quantity { get; set; }
    [JsonProperty("PageCount")]
    public int PageCount { get; set; }
    [JsonProperty("StatusCode")]
    public Nullable<int> StatusCode { get; set; }
    [JsonProperty("FulfilledQty")]
    public Nullable<int> FulfilledQty { get; set; }
    [JsonProperty("RegisteredUserLoginName")]
    public string RegisteredUserLoginName { get; set; }
    [JsonProperty("ReactionStatus")]
    public string ReactionStatus { get; set; }
    [JsonProperty("CertifiedDocumentSet")]
    public List<CertifiedDocumentSet> CertifiedDocumentSet { get; set; }
    [JsonProperty("Field1")]
    public string Field1 { get; set; }
    [JsonProperty("Field2")]
    public string Field2 { get; set; }
    [JsonProperty("Field3")]
    public string Field3 { get; set; }
    [JsonProperty("Field4")]
    public string Field4 { get; set; }
    [JsonProperty("Field5")]
    public string Field5 { get; set; }
}
public class CertifiedDocumentSet
{
    [JsonProperty("UniqueCode")]
    public string UniqueCode { get; set; }
}
public class GenericResponse
{
    public string ResponseCode { get; set; }
    public string ResponseDetails { get; set; }
}

```

3. Create a new method to receive the Json packet as shown below

```
[HttpPost]
public string PaymentPost(OrderInfo data)
{
    // Do your processing here

    GenericResponse response = new GenericResponse();
    response.ResponseCode = "1";
    response.ResponseDetails = "SUCCESS";
    string myResponse = JsonConvert.SerializeObject(response);
    return myResponse;
}
```

4. Contact TRIEDATA and provide them your payment posting URL information. Please make sure to enable firewall rules to allow Clerk eCertify system to post information to your server.

## 4.2 CLERKECERTIFYCONNECTORCOM - CLASSIC ASP ONLY

ClerkecertifyConnectorCom is a windows DLL used for interfacing from Clerk's web application developed in Classic ASP. Clerk's web application can invoke methods within this DLL to send docket information to Clerk E-Certify shopping cart.

Use these instructions for installing the ClerkecertifyConnectorCom within your visual studio-based application:

1. Contact TRIEDATA to obtain ClerkecertifyConnectorCom.dll
2. Run Visual Studio Developer **32-bit** command-line tool Assembly Registration Tool (Regasm.exe) to register the assembly. Regasm.exe adds information about the class to system registry for COM clients to use the .NET Framework class
  - i) Open the command line prompt for VS developer as Administrator
  - ii) Execute command: RegAsm.exe -tlb "*the path to the ClerkecertifyConnectorCom.dll*"
  - iii) Execute command: RegAsm "*the path to the ClerkecertifyConnectorCom.dll*" /codebase
  - iv) Check registry and ensure the class being added to the system registry  
*"Computer\HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\ClerkecertifyConnector.ShoppingCart"*
3. Configure your application.

Here is an example on how to invoke the ClerkecertifyConnectorCom methods and send docket information to Clerk E-Certify

---

## Classic ASP code example for Official Records E-certify

---

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <%
    <!--example code for generate e-certified Official records for the clerks at the help center -->
    dim foo
      set foo = Server.CreateObject("ClerkecertifyConnector.DirectAccess")
      eCertifyResult =
foo.ECertifyORDocument("https://securetest.triedata.com/SmartWeb.svc/ssl/GetCertifiedImageByReferenceNoAndSendEmail",
PublisherCode, SubscriberCode,Token,InstrumentNo, "UserRef",EmailAddress)
      response.write(eCertifyResult)
        if eCertifyResult = 1 then
          ' We have a successful eCertify OR result now. Lets print the results
          Response.Write("<div/>")
          for each v in foo.UniqueCode
            Response.Write(v )
          Next
          for each v in foo.FileName
            Response.Write(v )
          Next
        end if
      Response.End
    <!--example code for purchase e-certified court documents by the anonymous users -->
    set foo1 = Server.CreateObject("ClerkecertifyConnector.ShoppingCart")
    result = foo1.AddORDocToShoppingCartCom(myUniqueID,"https://test.clerkecertify.com","12073", mySubscriberCode, myToken
, "20050009451", "Easement", "1", "1", "NULL", "NULL", "NULL", "NULL", "NULL")
    Response.Write(result)
    url = foo1.GetShoppingCartUrl("https://test.clerkecertify.com", myUniqueID, " mySubscriberCode ")
    Response.Write(url)
    Response.Redirect(url)
  %>
</head>
<body>
```

---

---



---

## Classic ASP code example for Court document e-certify

---

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="utf-8" />

  <%

    <!--example code for generate e-certified court documents for the clerks at the help center -->

    dim foo

    set foo = Server.CreateObject("ClerkecertifyConnector.DirectAccess")

    RecordCount = foo.AddCourtDocForEcertification("372019CA000006XXXXXX","UCN","description 1", "107832851","Y",
    "NULL","NULL","NULL", "NULL", "NULL")

    RecordCount = foo.AddCourtDocForEcertification("372019CA000006XXXXXX","UCN","description 2", "107805334","Y",
    "NULL","NULL","NULL", "NULL", "NULL")

    RecordCount = foo.AddCourtDocForEcertification("372019CA000006XXXXXX","UCN","description 3", " 107599289","Y",
    "NULL","NULL","NULL", "NULL", "NULL")

    eCertifyResult = foo.ECertifyCourtDocuments("http://localhost:93/Service1.svc", "12073", " mySubscriberCode ", "myToken",
    "userref","myEmailAddress")

    response.write(eCertifyResult)

    if eCertifyResult = 1 then

      ' We have successfully submitted e-Certify request. Lets print the results
      ' Lets find the total record count
      for each v in foo.ResponseCode
        TotalRecordCount = TotalRecordCount + 1
      Next

      ' Lets put resulting e-certified documents into an 2 dimension array
      reDim myResultSet(TotalRecordCount,5)

      ' First dimension is the total records, second dimensions are the following:
      ' 0 – DocketID ; 1 – UniqueCode; 2 - FileName (URL that calling program call used to retrieve the image, for example
      https://ecertify.clerk.leon.fl.us:13010/CertifiedCourtImages/CAA-FAF-BCAHD-BAHIAFDDE-BAJHJ-D.pdf
      ' 3 - ResponseCode: ResponseCode is "1" if successfull

      x = 0
      y = 0
      for each v in foo.DocketId
        myResultSet(x,y) = v
        x = x+1
      Next
      x = 0
      y = y+1
      for each v in foo.UniqueCode
        myResultSet(x,y) = v
```

```

        x = x+1
    Next
    x = 0
    y = y+1
    for each v in foo.FileName
        myResultSet(x,y) = v
        x = x+1
    Next
    x = 0
    y = y+1
    TotalRecordCount = 0
    for each v in foo.ResponseCode
        myResultSet(x,y) = v
        x = x+1
        TotalRecordCount = TotalRecordCount + 1
    Next
' Now lets print the results
    dim counter
    counter = 0
    Response.Write("<div/> Total record count = ")
    Response.Write(TotalRecordCount)
    Response.Write("<div/>")
Response.Write("Docket ID"+vbTab +"UniqueCode"+vbTab+"FileName"+vbTab+vbTab+vbTab+"ResponseCode")
Response.Write("<div/>")
        for counter = 0 to TotalRecordCount - 1
recordResult = myResultSet(counter,3)
    if recordResult = "1" then
            Response.Write(myResultSet(counter,0) + "-" + myResultSet(counter,1) + "-" +
myResultSet(counter,2) + "-" + myResultSet(counter,3) + "-" + myResultSet(counter,4) )
            Response.Write("<div/>")
        else
            Response.Write("Docket ID " + myResultSet(counter,0) +" Error code = " + myResultSet(counter,4))
        end if
        next
    else
        Response.Write("Error code : " + eCertifyResult)
    end if

<!--example code for purchase e-certified court documents by the anonymous users -->
set foo1 = Server.CreateObject("ClerkecertifyConnector.ShoppingCart")

```

```
result =  
foo1.AddCourtDocToShoppingCartCom(myUniqueID,"https://test.clerkecertify.com","12073","mySubscriberCode","myToken",  
"372019CA000002XXXXXX","UCN","My description","107918943","1","1","Y","","","","","")
```

```
Response.Write(result)
```

```
url = foo1.GetShoppingCartUrl("https://test.clerkecertify.com", myUniqueID, " mySubscriberCode ")
```

```
Response.Write(url)
```

```
Response.Redirect(url)
```

```
%>
```

```
</head>
```

```
<body>
```

---

---

## 4.3 REQUESTS WITHOUT PAYMENT COLLECTION

In the event CLERK E-CERTIFY is not required to collect payment charges for certified documents, then direct API methods are available to generate certified documents. Clerk web application or CMS can directly invoke this API to satisfy the request. This method is used by Clerk applications with their own credit card processing and/or point of sale counters.

### 4.3.1 API END POINTS

Type	URL
Test environment for Court Document e-certify	<a href="http://securetest.triedata.com/CourtService.svc">http://securetest.triedata.com/CourtService.svc</a>
Production for Court Document e-certify	<a href="http://secure.triedata.com/CourtService.svc">http://secure.triedata.com/CourtService.svc</a>
Test environment for Official Records e-certify	<a href="https://securetest.triedata.com/SmartWeb.svc">https://securetest.triedata.com/SmartWeb.svc</a>
Production for Official Records e-certify	<a href="https://secure.triedata.com/SmartWeb.svc">https://secure.triedata.com/SmartWeb.svc</a>

Subscriber code: Please contract TRIEDATA

Token: Please contract TRIEDATA

---

*CourtService.svc endpoints are protected with X509 Certification for Authentication. You must configure X509 certificates and supply the public keys of your certificate to TRIEDATA before testing the interface. You also need to configure TRIEDATA public certificate in your API service.*

---

There are two different types of API's available to satisfy your request. Please choose the API based on your usage.

### CERTIFIED COURT DOCUMENTS

- **SendCertifiedCourtImageByDocketID** – This API is used in scenarios where your system is responsible for collecting online payments. After processing the payment, you can invoke this service to request generation of certified documents. The request will be queued and processed within a minute.

- GetCertifiedCourtImagesByDocketsAndSendEmail – This API is used for generating certified document for point of sale Customer. This is a synchronous call and the call will wait until the certified document has been generated. This API is used for interfacing with point of sale terminals.

---

*SendCertifiedCourtImageByDocketID transaction queues the request and processes it upon availability of resources. In the event network resources are unavailable or system unavailability, the request is queued until the system is available. This is the preferred method for processing requests generated after collecting payment from an online source.*

*Certified copies of very large documents (more than 250 pages) will take a longer time than usually allocated time. Since SendCertifiedCourtImageByDocketId is an asynchronous process, it will generate the document even after XML API timeout.*

---

## CERTIFIED OFFICIAL DOCUMENTS

- SendCertifiedORImageByReferenceNo– This API is used in scenarios where your system is responsible for collecting online payments. After processing the payment, you can invoke this service to request generation of certified official documents. The request will be queued and processed within a minute.
- GetCertifiedImageByReferenceNoAndSendEmail – This API is used for generating certified document for point of sale Customer. This is a synchronous call and the call will wait until the certified document has been generated. This API is used for interfacing with point of sale terminals.

---

### 4.3.2 API NAME: SENDCERTIFIEDCOURTIMAGEBYDOCKETID()

Source: Clerk's website

Target: Clerk E-Certify Gateway Cloud Service (a.k.a. Gateway)

Method: Https Post

Request Format: Json

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
-------	------------	-------------	-------------------

1	SubscriberCode	A unique identifier associated with the caller	Not Null
2	Token	An encrypted token for the Subscriber	Not Null Token for the subscriber will be by provided by TRIEDATA)
3	PublisherId	A unique identifier associated with the county	Not Null
4	EmailID	Email provided by internet users to receive e-certificates	Alphanumeric String, less than 100 characters, Not Null
5	List<CertImagesRequest>	List of documents for electronic certification	Object
6	UserRef	Online order number	Alphanumeric String less than 50 characters, Not Null

Response type: Synchronous

Response packet consists of two elements namely **Websvcreponse** object and **CourtCertifiedImages** response.

Response details	Description	Definition/Values
Websvcreponse	Indicates the result of the web service request.	Object
List<CourtCertifiedImages>	Indicates the result of each requested docket	Object

#### COURTCERTIFIEDIMAGES OBJECT

Index	Field Name	Description	Definition/Values
1	DocketID	Document identifier. We will be using this identifier to retrieve the image	Alphanumeric String, Not Null
2	DocketDescription	Descriptive summary of the document. This is usually the	Alphanumeric String, Not Null Maximum length is 100.

		docket description in Clericus system.	
<b>3</b>	UniqueReferenceNo	Unique identifier for the certified document	Alphanumeric String, Not Null
<b>4</b>	URL	Full path to the file name	Alphanumeric String, Not Null example: <a href="https://clerkecertify.com/DownloadCertifiedCourtImage/GetECertDocketImage?UniqueReferenceCode=xxx">https://clerkecertify.com/DownloadCertifiedCourtImage/GetECertDocketImage?UniqueReferenceCode=xxx</a> (Encrypted unique code)

Note: The URL returned by the service points to a web page for the end user to download the certified documents. The web page will display current status of the certified document processing and allow end user to view/download certified document. Since the certification request is processed in a queue, there will be a time lag of 1-2 minutes before the document will be available for download.

---

#### 4.3.3 API NAME: GETCERTIFIEDCOURTIMAGESBYDOCKETSANDSENDEMAIL ()

Source: Clerk CMS or Clerk website

Target: Clerk E-Certify Gateway Cloud Service (a.k.a. Gateway)

Method: Https Post

Request Format: Json

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
<b>1</b>	SubscriberCode	Unique identifier associated with the caller	Not Null
<b>2</b>	Token	Encrypted token for the Subscriber	Not Null Token for the subscriber will be provided by TRIEDATA)
<b>3</b>	PublisherId	A unique identifier associated with the county	Not Null County FIPS code

4	EmailID	Email provided by internet users to receive e-certificates	Alphanumeric String, less than 100 characters, Not Null
5	List<CertImagesRequest>	List of documents for electronic certification	Object
6	UserRef	Online order number	Alphanumeric String less than 50 characters, Not Null

### CertImagesRequest Object

Index	Field Name	Description	Definition/Values
1	CaseNo	Can be either UCN or LOCALCASENUMBER	Alphanumeric String, Not Null
2	CaseType	Case number type	Alphanumeric String, Not Null Valid values are: "UCN" or "LOCALCASENUMBER"
3	DocketID	Document identifier. We will be using this identifier to retrieve the image;  If multiple fields are required to uniquely identify a document, use ";" as a delimiter. For example, docketId;VersionId	Alphanumeric String, Not Null  For example:  1) 12345 docketId only; 2) 28547270;28912781 docketId;VersionId
4	DocketDescription	Descriptive summary of the document.	Alphanumeric string, Not Null <b>Maximum length is 100.</b>  This is a database constraint, a string over 100 character's length will be truncated.
5	RegisteredUserName	Not applicable	Not applicable
6	RedAct	Indicates whether a redacted document is requested	Y - N

Clerk E-Certify Online Response



Response type: Synchronous

Response packet consists of two elements namely **Websvcresponse** object and **CourtCertifiedImages** response.

Response details	Description	Definition/Values
<b>Websvcresponse</b>	Indicates the result of the web service request.	Object
<b>List&lt;CourtCertifiedImages&gt;</b>	Indicates the result of each requested docket	Object

Websvcresponse Object

Index	Field Name	Description	Definition/Values
1	TransactionCode	Indicates the unique transaction code associated with the request	Alphanumeric String, Not Null Always "1056"- SendCertifiedCTDocket
2	RecordCount	Integer. Represents the total number of records returned within the data body	Numeric String, Not Null Integer values
3	ResponseCode	Indicates the status of the call	Numeric String, Not Null 1 – Success - Various return codes for all other errors
4	ResponseDetails	Description for the error	Alphanumeric String Ex. Success
5	RemainingBalance	Indicates remaining balance of funds available in your escrow account	Alphanumeric String Not used

CourtCertifiedImages Object

Index	Field Name	Description	Definition/Values
1	DocketID	Caller provided Docket id	Alphanumeric String, Not Null

2	DocketDescription	Caller provided description	Alphanumeric String, Not Null Maximum length is 100.
3	UniqueReferenceCode	Unique identifier for the certified document	Alphanumeric String, Not Null
4	FileName	Full path to the file name	Alphanumeric String, Not Null example: <a href="https://192.1.1.1/Images/CertifiedImages/AAA-BBB-CCC.pdf">https://192.1.1.1/Images/CertifiedImages/AAA-BBB-CCC.pdf</a>
5	CaseNo	Caller provided case number	
6	CaseType	Caller provided case type	
7	Websvcresponse	Result for individual requests	See websvcresponse object

Please note that in order to access any Court services, your system must be configured to support mutual authentication based on X509 certificates. Please see section 4.3.6 for detailed instructions for configuring X509 based mutual authentication certificate security.

#### 4.3.4 API NAME: SENDCERTIFIEDORIMAGEBYREFERENCENO ()

Source: Clerk’s website

Target: Clerk E-Certify Gateway Cloud Service (a.k.a. Gateway)

Method: Https GET

Request Format: XML

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
1	SubscriberCode	A unique identifier associated with the caller	Not Null
2	Token	An encrypted token for the Subscriber	Not Null Token for the subscriber will be provided by TRIEDATA)
3	PublisherId	A unique identifier associated with the county	Not Null

4	EmailID	Email provided by internet users to receive e-certificates	Alphanumeric String, less than 100 characters, Not Null
5	ReferenceNo	Instrument number of the document. This number must be unique within your application and should correspond to one Official Record document.	Numeric string – less than 20 characters, Not Null
6	UserRef	Online order number	Alphanumeric String less than 50 characters, Not Null

Response type: Synchronous

Response packet consists of two elements namely **Websvcresponse** object, UniqueReference and URL in response.

Response details	Description	Definition/Values
<b>Websvcresponse</b>	Indicates the result of the web service request.	Object
<b>UniqueReference No</b>	Unique identifier for the certified document	Alphanumeric string, not null
<b>URL</b>	Full path to the file name	Alphanumeric String, Not Null example: <a href="https://clerkecertify.com/DownloadCertifiedCourtImage/GetECertDocketImage?UniqueReferenceCode=xxx">https://clerkecertify.com/DownloadCertifiedCourtImage/GetECertDocketImage?UniqueReferenceCode=xxx</a> (Encrypted unique code)

Note: The URL returned by the service points to a web page for the end user to download the certified documents. The web page will display current status of the certified document processing and allow end user to view/download certified document. Since the certification request is processed in a queue, there will be a time lag of 1-2 minutes before the document will be available for download.

#### 4.3.5 API NAME: SENDCERTIFIEDORIMAGES ( )

This API is used for sending Official Record certification requests in bulk. The request is queued and processed by eCertify system. This call will receive a synchronous response as a response to the call with details on unique codes assigned to each individual requests within the call. However, the actual work of creating the certified document(s) will be queued for future processing and will be executed upon availability of network and other resources (Usually within a minute). An email message is sent out to the user upon successful completion of the process.

Source: Clerk's website

Target: Clerk E-Certify Gateway Cloud Service (a.k.a. Gateway)

Method: Https POST

Request Format: JSON

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
1	SubscriberCode	A unique identifier associated with the caller	Not Null
2	Token	An encrypted token for the Subscriber	Not Null Token for the subscriber will be provided by TRIEDATA)
3	PublisherId	A unique identifier associated with the county	Not Null
4	EmailID	Email provided by internet users to receive e-certificates	Alphanumeric String, less than 100 characters, Not Null
5	List<CertORImagesRequest>	This list includes Official Records document identifiers	List
6	UserRef	Online order number	Alphanumeric String less than 50 characters, Not Null

#### CertORImagesRequest

Item	Description	Definition/Values
<b>ReferenceNo</b>	Unique instrument number or reference number associated with the document	This field must be unique within your system. For example, typically the instrument numbers of Official Records are unique for each document within the Official Records repository. Cannot exceed 20 numeric characters
<b>DocDescription</b>	Description of the document	Example: MAR for marriage certificate, NOC for notice of commencement etc.  Max Size: 100

--	--	--

Response type: Synchronous

Response packet consists of two elements namely **Websvcresponse** object, UniqueReference and URL in response.

Response details	Description	Definition/Values
<b>Websvcresponse</b>	Indicates the result of the web service request.	Object
<b>List&lt;ORCertifiedImages&gt;</b>	List consisting of response to each request	List

#### ORCertifiedImages

Item	Description	Definition/Values
<b>UniqueReferenceNo</b>	This is the unique reference number of the certified document	Example :AAA-BBB-CCCC"
<b>ReferenceNo</b>	This is the reference number passed by your system to eCertify	Numeric Max Size: 20
<b>DocDescription</b>	This is the definition passed by your system to eCertify	Alphanumeric Max Size: 100
<b>URL</b>	This is the URL to access results of the certification process	This URL points to a web page to check status of the certification operation.

Note: The URL returned by the service points to a web page for the end user to download the certified documents. The web page will display current status of the certified document processing and allow end user to view/download certified document. Since the certification request is processed in a queue, there will be a time lag of 1-2 minutes before the document will be available for download.

---

#### 4.3.6 API NAME: GETCERTIFIEDIMAGEBYREFERENCENOANDSENDEMAIL ()

Source: Clerk CMS or Clerk website

Target: Clerk E-Certify Gateway Cloud Service (a.k.a. Gateway)

Method: Https GET

Request Format: XML

Response Format: XML

Body Style: Wrapped

Index	Field Name	Description	Definition/Values
1	SubscriberCode	Unique identifier associated with the caller	Not Null
2	Token	Encrypted token for the Subscriber	Not Null Token for the subscriber will be provided by TRIEDATA)
3	PublisherId	A unique identifier associated with the county	Not Null County FIPS code
4	EmailID	Email provided by internet users to receive e-certificates	Alphanumeric String, less than 100 characters, Not Null
5	ReferenceNo	Instrument number of the document.	Numeric string, 20 digit limit
6	UserRef	Online order number	Alphanumeric String less than 50 characters, Not Null

Clerk E-Certify Online Response

Response type: Synchronous

Response packet consists of two elements namely **Websvcreponse** object and additional certified document information.

Response details	Description	Definition/Values
<b>Websvcreponse</b>	Indicates the result of the web service request.	Object
<b>UniqueReferenceNo</b>	This is the unique code of the certified document	Alphanumeric string,
<b>FileName</b>	URL to retrieve the certified document.	This URL will point to the location where the file is stored within your computing platform. The URL server address will be pointing to external

		host address through which Clerk eCertify will be retrieving the image.
--	--	---

---

### 4.3.7 X509 CERTIFICATION FOR AUTHENTICATION

Purpose: Triedata XML-API gateway allows customer software applications to interface with Triedata utilizing either SOAP or RESTful endpoints. Triedata gateway offers two services namely

- SmartWeb.svc: This service is for Official Records Only
- CourtService.svc: This service is for Court Services only

SmartWeb.svc endpoint offers both SOAP as well as RESTful endpoints and does not require client authentication using certificates.

CourtService.svc offers SOAP endpoints, and require client authentication using X509 certificates. We are using message-based security to make sure that all messages are encrypted between the server and the Client and only authorized clients can consume the services.

See document <https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/message-security-in-wcf> for more details on message security.

How it works: In order to access CourtService.svc, the Client and the Server both must enable message security on their end. By default, message security is enabled on Triedata gateway server. Both the Client and the Server presents their own credentials as part of the "PeerTrust" connectivity. The theory is that both the Client and Server holds their own digital certificates with private keys within their own local certificate store. They both exchange the public certificates and these certificates are also stored within their "Personal" certificate store. Here is the configuration looks like

Triedata gateway:

- LocalMachine->Personal certificate store contains the X509 certificate of triedata.com with a private key
- LocalMachine->Personal certificate store contains the X509 certificate of the Client with its public key (No private keys)

Client system:

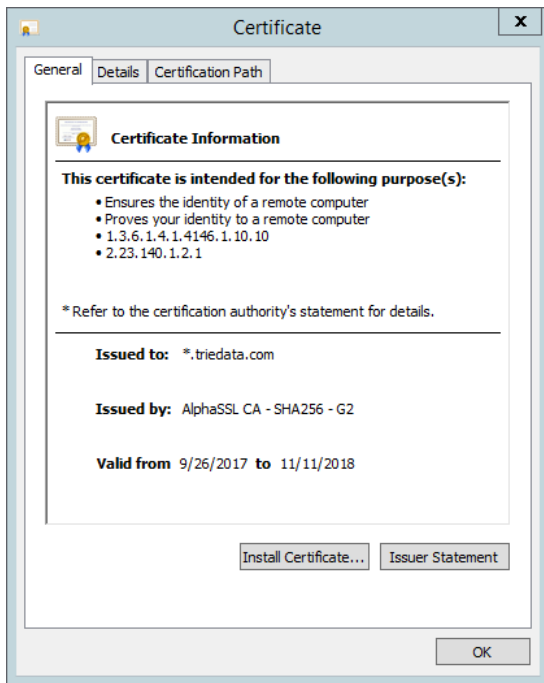
- LocalMachine->Personal certificate store contains the X509 certificate of the Client's digital certificate with the private key
- LocalMachine->Personal certificate store contains the X509 certificate of triedata.com with its public key (No private keys)

When the messages are sent from the Client to the server, they are encrypted using Clients private key. Triedata server decrypts them using the Clients public keys stores in its “Personal” folder. The same way, when the messages are sent from triedata.com, they are encrypted with Triedata.com’s private key and decrypted by the Client using Triedata.com’s public certificate.

---

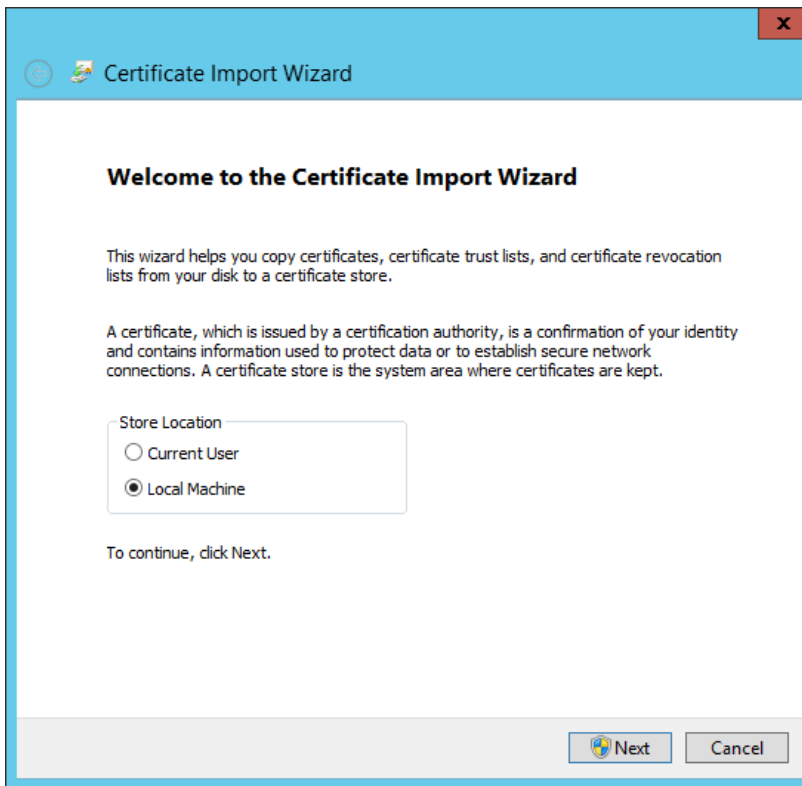
#### 4.3.7.1 INSTALLING X509 CERTIFICATE – INSTALL TRIEDATA.COM CERTIFICATE ON CLIENT MACHINE:

1. Triedata.com gateway CourtService.svc API is already configured to process X509 certificates. We will provide you a “.cer” file with triedata.com public key. Double click on this “.cer” file and click on “Install Certificate” as shown below

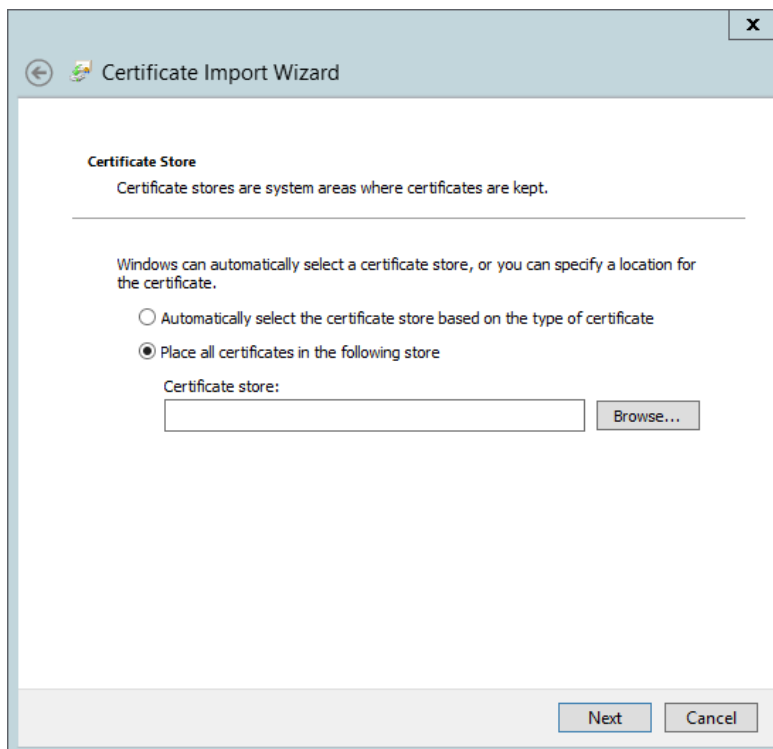


2. Select LocalMachine store

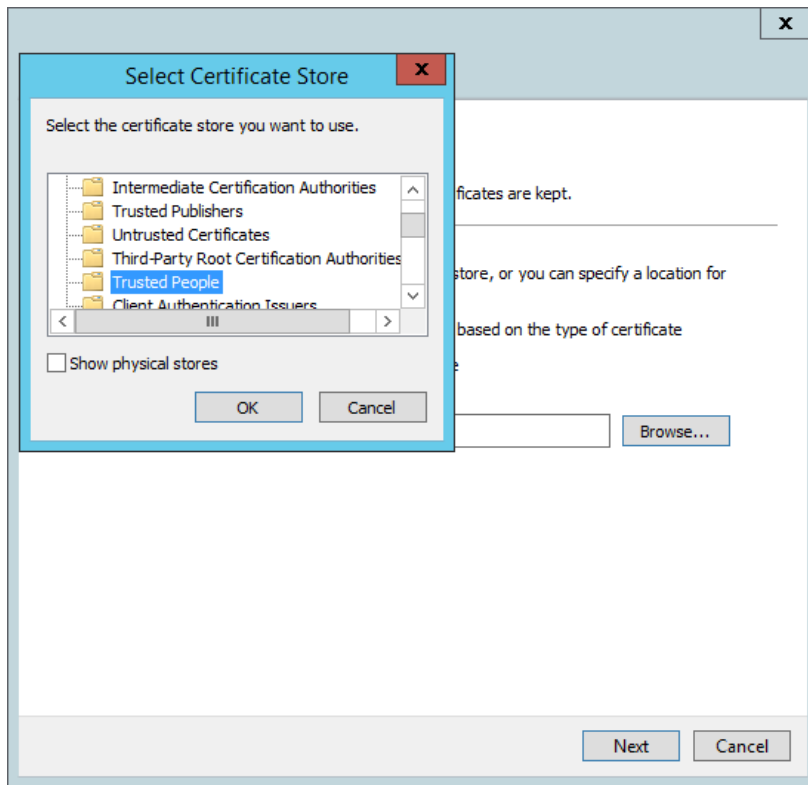




3. Select the location of the store by clicking on "Place all certificates in the following store"



4. Select “Personal” store



Now the first part of the configuration to access Triedata gateway CourtService.svc API's is complete.

---

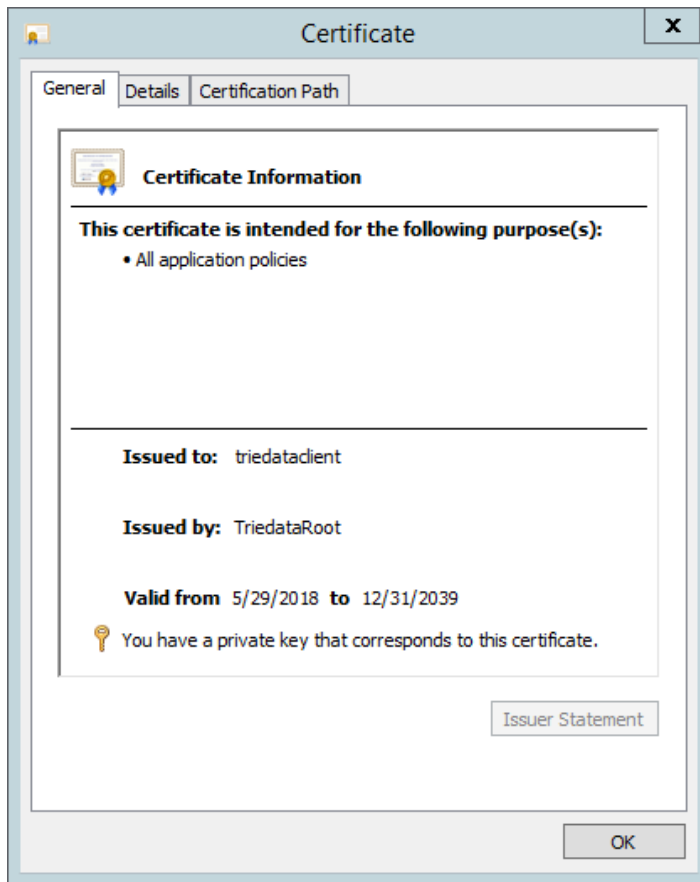
#### 4.3.7.2 GENERATE SELF-SIGNED CERTIFICATE OF THE CLIENT:

Next step is to create a self-signed certificate for the client. Follow the instructions provided by the link to create a self-signed certificate.

<https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate-in-iis-7.html>

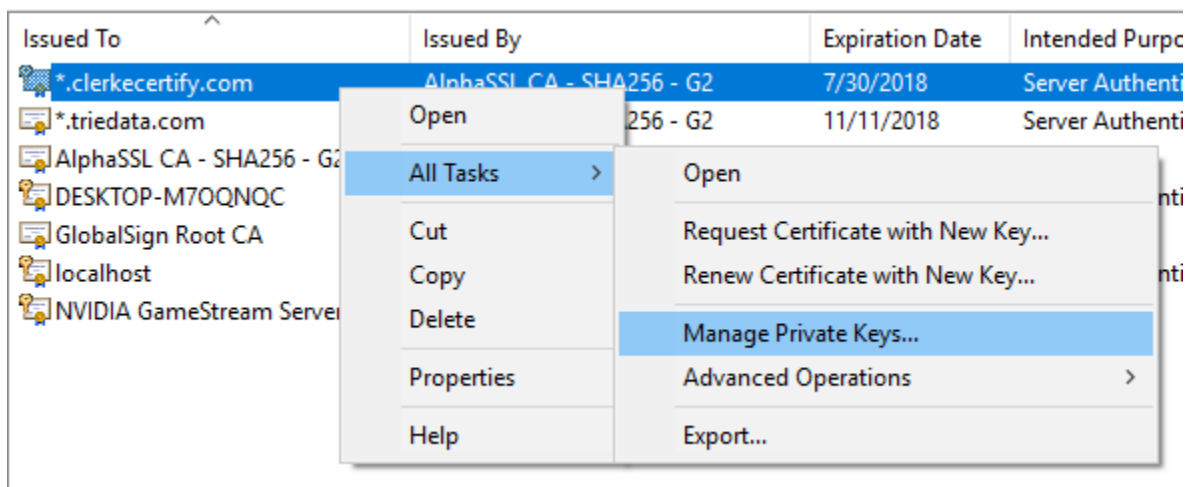
(Note: DO NOT MOVE THE CERTIFICATE TO “TRUSTED PEOPLE ROOT CERTIFICATION AUTHORITIES). You can ignore all steps after step 6.

- Name the client as “myclientCertificate”
- Now your self-signed certificate will be in the LocalMachine->Personal folder
- Go ahead and launch “mmc” and add Certificates snapshot for LocalMachine. Double click on your certificate and make sure that you have the private key corresponding to the certificate as shown below.



Alright, you have the certificate with private key in your LocalMachine->Personal folder. Next step is to provide access to the current user to be able to access this private key.

To do so select the certificate with private key and right click -> and select manage private keys...



Now add the application user to the approved list. If you are deploying it in IIS, add IIS\_USERS to this permission list.

---

#### 4.3.7.3 EXPORT PUBLIC KEY FROM THE SELF-SIGNED CERTIFICATE

Next step is to export the public key of the self-signed certificate. You need to forward this public key to Triedata for us to configure it in our system.

- In “mmc”, add snap-in for Certificates->ComputerAccount
- Select the certificate “myclientCertificate” from your “Personal” store.
- Retrieve the thumbprint of this certificate. We will need this for future reference.
  - Double click on your certificate and go to “Details” tab
  - Scroll down to the bottom and select the field named “thumbprint”. Copy the thumbprint to **notepad**. The thumbprint contains several spaces, please remove the spaces carefully. **ALSO, THERE IS AN INVISIBLE CHARACTER AT THE FRONT OF THE THUMBPRINT CHARACTER. MAKE SURE TO DELETE IT BY PRESSING BACKSPACE.**
  - Your thumbprint should look something like this “**eb5d607d414114e67afd758c164bfecf088aeed**”. We are going to use this thumbprint for future use.
- Right mouse click on your certificate and select “All tasks-> export”. Make sure to check “No, do not export the private key”.
- Store the certificate in your local folder as “myclientCertificate.cer”.
- Email this certificate to us at [Support@Triedata.com](mailto:Support@Triedata.com) with subject “x509 Client”

---

#### 4.3.7.4 SETTING UP COURTSERVICE.SVC REFERENCE

Here are the steps for consuming Triedata CourtService.svc:

1. Open visual studio project and add service reference to the following URL  
<http://securetest.triedata.com/CourtService.svc>
2. Name this as “TriedataReference” for easy reading. The service reference should have created all proxy classes needed to access the API’s.
3. Make the following changes in your application web.config file
  - a. Change endpoint as below

---

### *Endpoint configuration*

---

```
<endpoint address="http:// http://securetest.triedata.com/CourtService.svc
    /CourtService.svc/x509"
    binding="wsHttpBinding"
    bindingConfiguration="WSHttpBinding_ICourtService"
    contract="CourtService.ICourtService"
    name="WSHttpBinding_ICourtService"
    behaviorConfiguration="ClientCredentialsBehavior">
    <identity>
        <dns value="test.triedata.com"/>
    </identity>
</endpoint>
```

---

---

- b. Add clientCredentials behavior (Your certificate thumbprint will differ from example below)

---

## Service Model

---

```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="ClientCredentialsBehavior">
        <!-- DEV-SERVER CONFIGURATION-->
        <clientCredentials>
          <clientCertificate findValue="ENTER YOUR THUMBPRINT"
            x509FindType="FindByThumbprint"
            storeLocation="LocalMachine"
            storeName="My" />
          <serviceCertificate>
            <defaultCertificate findValue="Enter Server thumbprint" x509FindType="FindByThumbprint"
              storeLocation="LocalMachine" storeName="My"/>
          </serviceCertificate>
        </clientCredentials>
      </behavior>
    </endpointBehaviors>
  </behaviors>
```

- a. Verify the binding configuration as shown below

```
<wsHttpBinding>
  <binding name="WSHttpBinding_ICourtService" closeTimeout="00:01:00"
    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:10:00"
    maxBufferPoolSize="2147483647" maxReceivedMessageSize="2147483647">
    <security>
      <message clientCredentialType="Certificate" negotiateServiceCredential="false"
        establishSecurityContext="false" />
    </security>
  </binding>
</wsHttpBinding>
```

---

---

---

#### 4.3.7.5 CONSUMING COURTSERVICE.SVC REFERENCE

We are almost finished now. Let's consume the service.

Before we can accomplish this, we need to create a helper class as shown below. My service reference is called "CourtService". I am going to create a helper class so that I can access the service whether I am in production or test with the same application. Code is below

---

## Helper class

---

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Net;
using System.ServiceModel;
using System.Web;
namespace ClerkECertify.Helpers
{
    public class MyCourtServiceReference : TriedataReference.CourtServiceClient
    {
        public MyCourtServiceReference()
        {
            // Lets get Test and Prod server names
            string LocalServer =
ConfigurationManager.AppSettings["LocalCourtServiceAddress"];
            string ProdServer =
ConfigurationManager.AppSettings["ProdCourtServiceAddress"];
            EndpointIdentity identity =
EndpointIdentity.CreateDnsIdentity("triedata.com");
            this.Endpoint.Address = new System.ServiceModel.EndpointAddress(new
Uri(ProdServer),
                identity, this.Endpoint.Address.Headers);
        }
    }
}
```

---

---



Now we are ready to consume the service. All you need is one call as below

---

### GetCertifiedCourtImagesByDocketsAndSendEmail

---

```
using (MyCourtServiceReference myClient = new MyCourtServiceReference())
{
    var tempResponse =
    client.GetCertifiedCourtImagesByDocketsAndSendEmail(SubscriberId, Token, PublisherId,
    myRequests, UserRef, EmailId);
    var ResponseCode = tempResponse.websvcresponse.ResponseCode.ToString();
        if(ResponseCode == "1")
        {
            foreach(var tempRec in tempResponse.CertifiedImages)
            {
                // Do your processing
            }
        }
        else
        {
            // We need to log it somehow
        }
    }
}
```

---

---

## 5 E-CERTIFY EMAIL DELIVERY SERVICE

Clerk E-Certify web application residing within TRIEDATA data center is able to connect to multiple types of email services. We currently support following types of email services.

- SMTP

- MS-Exchange (OWA)
- Office 365

If your email service offers any of the above services, then there are no further changes needed. All you need to do is to provide TRIEDATA with following information.

- User name
- Password
- Server end point address

Upon completing the certification process, Clerk E-Certify will send an email, using your email server, to end user with links to download the certified document.

## 5.1 CUSTOM EMAIL DELIVERY SERVICE

In the event your office does not offer any of the services mentioned above, you can create a custom WCF service to send the emails. Clerk's IT is responsible for creating the service. Clerk E-Certify will send following information to Clerk's service.

Operation	Description
SendCertifiedCourtDocEmail	This service sends certified document links to Clerk service

In the event the Clerk email service is not available, Clerk eCertify will attempt to resend the information 5 times in succession. If the service is still not available, then Clerk eCertify will move the message for manual processing.

### 5.1.1 SENDCERTIFIEDCOURTDOCEMAIL()

This method sends both court records and official record emails. "Field 1" within the data packet denotes whether this is an Official Record or Court Record.

Method name: *SendCertifiedCourtDocEmail* ()

Source: Clerk E-Certify web application

Target: Clerk's WCF email service

Method: Https Post

Request Format: JSON

Response Format: JSON

Endpoints: Supports REST

#### 5.1.1.1 SENDEMAILREQUEST

Here is the data definition for input to the SendCertifiedCourtDocEmail () operation:

Index	Field Name	Description	Definition/Values
1	SenderEmailAddress	Email address of the recipient	String
2	CertifiedDocuments	list of certified documents	Object

CertifiedDocuments object is defined as follows:

Index	Field Name	Description	Definition/Values
1	CaseNO	Case No	Not Null Max Size: 30
2	CaseType	Indicating whether UCN or LOCALCASENUMBER	Not Null, integer UCN = 1, LOCALCASENUMBER = 2, UNKNOWN = 3
3	DocketNumber	Docket number associated with the document  If Field 1 is "100", the docket number;  If Field 1 is "200", official record instrument number	Not Null Max Size: SQL integer (10 chars)
4	DocketDescription	Document description  If Field 1 is "100", the docket description;  If Field 1 is "200", official record description	Alphanumeric String, Not Null Max Size: 100
5	UniqueCode	Unique identifier is created for each electronic certified document	Alphanumeric String, Not Null Max Size: 50
6	FileLink	URL location pointing to the certified document. It is encrypted.	Alphanumeric String, Not Null
7	Field1	This field indicates whether the document is an Official Record or Court Document.	Alphanumeric String, Not Null "100" – Official Record

			"200" – Court Record
8	Field2	Not used	Not used
9	Field3	Not used	Not used
10	Field4	Not used	Not used
11	Field5	Not used	Not used

### 5.1.1.2 SENDEMAILRESPONSE

Here is the data returned by the client operation:

Index	Field Name	Description	Definition/Values
1	ResponseCode	Status code	Not Null, integer value Value = 1: success Value >1: exception
2	ResponseDetails	Information for exceptions	Nullable, String Max Size: 100 If there is an exception, provide details on the error.

## 6 VERIFICATION

Clerk E-Certify offers two methods for verifying certified documents. The first method is to verify by unique code and the second method is to upload the certified document.

### 6.1 VERIFY BY CODE

Verify by code method is used by either entering the unique code on Clerk eCertify web site or by scanning the QR code embedded on the first page of the certified document. The system confirms the validity of the code and optionally allows the user to download watermarked original document for side-by-side comparison.

Please note that depending upon the document type, certain court documents such as Juvenile and mental health documents may not be available for side-by-side comparison. In this case, always advise the user to upload the original electronic certified copy for full verification.

Verify by code user interface can be launched by three methods

---

### 6.1.1 BROWSER REDIRECTION

You can redirect the user to <https://www.clerkecertify.com/VerifyImage> web site and let the user enter the code on Clerk E-Certify web site.

Verification by code URL Link : <https://test.clerkecertify.com/VerifyImage>

This option provides both verification by code and verification by file upload on the same user interface.

---

### 6.1.2 JAVA SCRIPT MODAL WINDOW

A java script-based interface is available for integrating with your web site. You can use following steps to embed the verification module within your web application.

1. Contact TRIEDATA and advise them about your domain name (Example: Browardclerk.org, Osceolaclerk.com etc.). Each domain name needs to be explicitly approved before your code can activate it. We will enable Cross Origin Resource Sharing permission (CORS) for your domain name.
2. Make sure that pre-requisite libraries such as bootstrap, jQuery, jQuery and jQuery forms are loaded. CDN path for all items are shown below

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9ifjh" crossorigin="anonymous">

<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css" rel="stylesheet" />

<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.form/4.2.2/jquery.form.min.js" integrity="sha384-FzT3vTVGXqf7wRfy8k4BiyzvbNfeYjK+frTVqZeNDFI8woCbF0CYG6g2fMEFFo/i" crossorigin="anonymous"></script>
```

3. Assign a DIV element to host the “Verify by Code textbox” in your view

```
<div class="row">  
  <div class="col-md-12" id="idCorsOrigin" style="width:auto;height:auto;margin:auto;background-color:white;padding:20px;"></div>  
</div>  
</div>
```

#### 4. Launch verification-by-code window on document ready event

```
<script type="text/javascript">
    $(document).ready(function () {
        var fileSource = 'https://www.clerkecertify.com/VerifyImage/CorsCodeGetIndex';
        $.ajax({
            url: fileSource,
            timeout: 4000,
            statusCode: {
                404: function () {
                    // Simple not found page, but not CORS violation
                    $("#idCorsOrigin").html(this.url + " not found");
                }
            }
        })
        .fail(function (jqXHR, textStatus) {
            // Empty status is a sign that this may be a CORS violation
            // but also check if the request timed out, or that the domain exists
            if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
                $("#idCorsOrigin").html("Unable to launch validation module Error
code: " + jqXHR.status + " " + jqXHR.statusText + " error");
                return;
            }
            else
            {
                $("#idCorsOrigin").html("Please refresh your browser page again: Error
code: " + jqXHR.status + "-" + jqXHR.statusText);
                return;
            }
        })
        .done(function (data) {
            // Successful ajax request
            $("#idCorsOrigin").html(data);
        });
    });
</script>
```

That's it.

You may have to tweak the timeout value to a higher value depending upon user bandwidth constraints. For most connections, default value specified above should be sufficient.

### 6.1.3 EMBEDDED VIEW

Instead of opening a modal pop-up window to display the results of code verification, you can embed the results in a DIV within your web page. This will allow you to display the results of document verification within your own portal without the pop up box.

All you need to do is to pass the name of the DIV as a parameter within the Javascript (Above).

Here is the full script



```

<!-- If this div is setup here and the id is passed to Clerk eCertify controller,
then Clerk eCertify will display the results in this DIV rather than as a pop up
box -->
    <div class="row">
        <div id="idVerifyByCodeResult" class="text-center">
    </div>

<script type="text/javascript">
    $(document).ready(function () {
        var fileSource =
'https://www.clerkecertify.com/VerifyImage/CorsCodeGetIndex?TargetDiv=idVerifyByCodeResult';
        $.ajax({
            url: fileSource,
            timeout: 4000,
            statusCode: {
                404: function () {
                    // Simple not found page, but not CORS violation
                    $("#idCorsOrigin").html(this.url + " not found");
                }
            }
        })
        .fail(function (jqXHR, textStatus) {
            // Empty status is a sign that this may be a CORS violation
            // but also check if the request timed out, or that the domain exists
            if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
                $("#idCorsOrigin").html("Unable to launch validation module Error
code: " + jqXHR.status + " " + jqXHR.statusText + " error");
                return;
            }
            else
            {
                $("#idCorsOrigin").html("Please refresh your browser page again: Error
code: " + jqXHR.status + "-" + jqXHR.statusText);
                return;
            }
        })
        .done(function (data) {
            // Successful ajax request
            $("#idCorsOrigin").html(data);
        });
    });
</script>
</script>

```

---

#### 6.1.4 QR CODE VERIFICATION ON YOUR OWN WEB SITE

Traditionally certified documents have a cover page containing a QR code for verifying the authenticity of the code and for retrieving the original image for side-by-side comparison. Users can use smartphones to scan the QR code, which will redirect them to launch clerkecertify.com web site to validate the code and optionally download the original file for side-by-side comparison.

QR code contains a hyperlink to launch default web browser. Here is an example for the hyperlink

<https://verify.clerkecertify.com/VerifyImage/VerifyCode?Code=CAA-CAFGEERTG-BFEGD-EEACFEIA-EAFGE-K>

As you can see, the QR code is pointing to clerkecertify.com web site

Since the Clerk web site has ability to integrate Clerk eCertify verification logic within their web portal, it makes sense to redirect the user to Clerk web portal for verifying the documents. To do this, you need to do the following

1. Create a class to hold the Code

```
public class EcertifyRequestViewModel
{
    public string CorrelationId { get; set; }
}
```

2. Create a controller action to receive the code. Let's say the controller is called "everify"

```
public ActionResult VerifyCode(string Code)
{
    EcertifyRequestViewModel myModel = new EcertifyRequestViewModel();
    myModel.CorrelationId = Code;
    return View(myModel);
}
```

- Use Javascript to load Clerk eCertify “Verify by code” widget as documented in above section and then update the loaded div with the correlation Id.

```

        <div class="row">
            <div class="col-md-12" id="idCorsOrigin" style="width:auto;
height:auto;margin:auto;background-color:white;padding:20px;"></div>
        </div>

<script type="text/javascript">
$(document).ready(function () {
var fileSource = https://test.clerkecertify.com/VerifyImage/CorsCodeGetIndex';
$.ajax({
    url: fileSource,
    timeout: 4000,
    statusCode: {
        404: function () {
            // Simple not found page, but not CORS violation
            $("#idCorsOrigin").html(this.url + " not found");
        }
    }
});
.fail(function (jqXHR, textStatus) {
    // Empty status is a sign that this may be a CORS
violation
    // but also check if the request timed out, or that the
domain exists
    if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
        $("#idCorsOrigin").html("Unable to launch
validation module Error code: " + jqXHR.status + " " + jqXHR.statusText + " error");
        return;
    }
    else {
        $("#idCorsOrigin").html("Please refresh your
browser page again: Error code: " + jqXHR.status + "-" + jqXHR.statusText);
        return;
    }
});
.done(function (data) {
    // Successful ajax request
    $("#idCorsOrigin").html(data);
    // Lets fill the correlation id
    var correlationVal = "@Model.CorrelationId";
    $('#idCode').val(correlationVal);
}); /* Drakes, 2015 */
});
</script>

```

Now you have a valid controller action and View to load the target DIV with the QR code.

Your cover page QR code can be updated to point to [https://your\\_web\\_site\\_url/everify/VerifyCode?Code=AAA-BBB-CCCC](https://your_web_site_url/everify/VerifyCode?Code=AAA-BBB-CCCC) and it should load Clerk eCertify widget and update Clerk eCertify DIV (idCode) with the parameter.

User will be prompted to click on google “Recaptcha” to continue verification process.

## 6.2 VERIFY BY FILE UPLOAD

Verify by file upload method is used for validating electronic copy of the certified document. User can upload the document and the system will perform an exhaustive set of checks to verify the authenticity of the document.

Verify by file upload is the most authentic and easiest way to validate a certified copy. This method checks following aspects of the document

- Does this document contain Clerk eCertify unique identifiers?
- Is it a signed document?
- Was this document signed with a valid digital signature?
- Was the signing certificate revoked at the time of signing the document?
- Was this document generated from the Clerk’s digital signature?

All certified documents can be verified by this method and does not require a side-by-side comparison. Since certain court documents are not allowed to be displayed to general public, this method is the preferred method for validating all document types.

Verify by file user interface can be launched by three methods

---

### 6.2.1 BROWSER REDIRECTION

You can redirect the user to <https://www.clerkecertify.com/VerifyImage> web site and let the user upload the file on Clerk E-Certify web site.

Verification by code URL Link: <https://test.clerkecertify.com/VerifyImage>

This option provides both verification by code and verification by file upload on the same user interface.

---

### 6.2.2 JAVA SCRIPT MODAL WINDOW

A java script based modal pop up window interface is available for integrating with your web site. You can use following steps to embed the verification module within your web application.

1. Contact TRIEDATA and advise them about your domain name (Example: Browardclerk.org, Osceolaclerk.com etc.). Each domain name needs to be explicitly approved before your code can activate it. We will enable Cross Origin Resource Sharing permission (CORS) for your domain name.

2. Make sure that pre-requisite libraries such as bootstrap, jQuery, jQuery and jQuery forms are loaded. CDN path for all items are shown below

```
<link href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">

<link href="https://code.jquery.com/ui/1.10.4/themes/ui-lightness/jquery-ui.css" rel="stylesheet" />

<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

<script src="https://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-
wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4lH7YwaYd1iqfktj0Uod8GCEl3Og8ifwB6"
crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.form/4.2.2/jquery.form.min.js"
integrity="sha384-FzT3vTVGXqf7wRfy8k4BiyzvbNfeYjK+frTVqZeNDFI8woCbF0CYG6g2fMEFFo/i"
crossorigin="anonymous"></script>
```

3. Assign a DIV element to host the file selection box in your view

```
<div class="row">

    <div class="row">

        <div class="col-md-12" id="idCorsFileOrigin" style="width:auto;
height:auto;margin:auto;background-color:white;padding:20px;"></div>

    </div>

</div>
```

4. Launch verification-by-code window on document ready event

```

<script type="text/javascript">
    $(document).ready(function () {
        var fileSource = 'https://www.clerkecertify.com/VerifyImage/CorsFileGetIndex';
        $.ajax({
            url: fileSource,
            timeout: 4000,
            xhrFields: {
                withCredentials: true
            },
            statusCode: {
                404: function () {
                    // Simple not found page, but not CORS violation
                    $("#idCorsFileOrigin").html(this.url + " not found");
                }
            }
        })
        .error(function (jqXHR, textStatus) {
            // Empty status is a sign that this may be a CORS violation
            // but also check if the request timed out, or that the domain exists

            $("#idCorsFileOrigin").html("We have encountered an error Status: " +
jqXHR.status + "-" + jqXHR.statusText);
            return;
        }
        )
        .fail(function (jqXHR, textStatus) {
            // Empty status is a sign that this may be a CORS violation
            // but also check if the request timed out, or that the domain exists
            if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
                $("#idCorsFileOrigin").html("Unable to launch
validation module Error code: " + jqXHR.status + " " + jqXHR.statusText + " error");
                return;
            }
            else
            {
                $("#idCorsFileOrigin").html("Please refresh your browser page again:
Error code: " + jqXHR.status + "-" + jqXHR.statusText);
                return;
            }
        })
        .done(function (data) {
            // Successful ajax request
            $("#idCorsFileOrigin").html(data);
        });
    });
</script>

```

That's it.

You may have to tweak the timeout value to a higher/lower value depending upon user bandwidth constraints.

---

### 6.2.3 EMBEDDED VIEW

Instead of opening a modal pop-up window to display the results of file verification, you can embed the results in a DIV within your web page. This will allow you to display the results of document verification within your own portal without the pop up box.

All you need to do is to pass the name of the DIV as a parameter within the Javascript (Above).

Here is the full script

```

<!-- If this div is setup here and the id is passed to Clerk eCertify controller,
then Clerk eCertify will display the results in this DIV rather than as a pop up
box -->
    <div class="row">
        <div id="idVerifyByFileResult" class="text-center">
    </div>

<script type="text/javascript">
    $(document).ready(function () {
        var fileSource =
'https://www.clerkecertify.com/VerifyImage/CorsCodeGetIndex?TargetDiv=idVerifyByFileResult';
        $.ajax({
            url: fileSource,
            timeout: 4000,
            statusCode: {
                404: function () {
                    // Simple not found page, but not CORS violation
                    $("#idCorsOrigin").html(this.url + " not found");
                }
            }
        })
        .fail(function (jqXHR, textStatus) {
            // Empty status is a sign that this may be a CORS violation
            // but also check if the request timed out, or that the domain exists
            if (jqXHR.status > 0 && jqXHR.statusText == "timeout") {
                $("#idCorsOrigin").html("Unable to launch validation module Error
code: " + jqXHR.status + " " + jqXHR.statusText + " error");
                return;
            }
            else
            {
                $("#idCorsOrigin").html("Please refresh your browser page again: Error
code: " + jqXHR.status + "-" + jqXHR.statusText);
                return;
            }
        })
        .done(function (data) {
            // Successful ajax request
            $("#idCorsOrigin").html(data);
        });
    });
</script>
</script>

```



