

**United States Army
Records Management and Declassification Agency (USARMDA)**

Army Records Information Management System

ARIMS Interface Control Document (ICD)



Prepared by



Nov 18, 2011

Table of Contents

1	ARIMS Interface Control	2
2	Uploading and Retrieving with the ARIMS Interface Control.....	2
2.1	Upload Process	2
2.2	Retrieval Process.....	3
3	Code Sample	4
4	Methods.....	7
5	Web Service Interface Tool Testing Findings.....	8

1 ARIMS Interface Control

The ARIMS Interface Control is a streamlined web service used to provide efficient conveyance, upload capabilities and retrieval capabilities, of documents to the ARIMS AEA. There are two main methods to this interface: one method to upload a document; and the other method is to retrieve a document. The user will have to provide a user name and a password to be authenticated, in order to gain access (authorization) to upload a document to a particular folder, or to retrieve a particular document.

2 Uploading and Retrieving with the ARIMS Interface Control

A sample code is provided below for uploading and retrieving a document from the client side. A web reference (or proxy class) to the web service will be needed.

The URL to reference the web service is: https://stage.arims.army.mil/ARIMS_ICD/icdservice.aspx
If further assistance is needed please contact RMDA.

2.1 Upload Process

Process to upload a document:

1. Get an instance of **IcdService**
2. Get an instance of **ICD_Token**
3. Call the Login method and provide the parameters against the instance created on Step 1., and store the return value in the instance created on Step 2.
4. Check the **ICD_Token.token** value if it is null user does not have access (contact RMDA if you should have access). If it is not null you have successfully authenticated and you are authorized to upload
5. Call **GetRecordPath** against the **IcdService** instance to get the location where the file will be stored on the file server.
6. Call the **AppendToUpload** against the instance of the **IcdService** created in Step 1. to upload the file in chunks.
7. When all the chunks are uploaded call the **EndUpload** method in the **IcdService** object created in Step 1 to get the ID of the document just submitted to the AEA.

2.2 Retrieval Process

Process to retrieve a document:

1. Get an instance of **IcdService**
2. Call the **AccessDocument** method of the **IcdService** object created in Step 1.
3. Get back the **indexDestinationPath** and **fileName**
4. Call the **RetrieveDocument** method of the **IcdService** object created in Step 1. using the values obtained in Step 3. The file is downloaded in chunks.

3 Code Sample

```
*****
*****
                        CODE SAMPLE
*****
*****

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestICDWebService.ICDWebService;
using System.IO;

namespace TestICDWebService
{
    class Program
    {
        static void Main(string[] args)
        {
            Program P = new Program();
            //P.UploadToAEA();
            //P.Retrieve();
            P.Retrieve();
            Console.WriteLine("We are finished");
            Console.ReadLine();
        }
        public void UploadToAEA()
        {
            ICD_Token icd_Token = new ICD_Token();
            //UPLOAD record to AEA

            string userName = "username";
            string password = "password";
            string UIC = "W313AA";
            string officeSymbol = "AAHS-TES-TNG";
            string folderName = "DBATestFolder";
            string officeRecordList = "ContractorTest_Robert";

            try
            {

                IcdService icdService = new IcdService();
                icd_Token = icdService.Login(userName, password, UIC,
officeSymbol, folderName, officeRecordList);

                //will be repeated for each file to be uploaded
                //Get the path where the file will be uploaded on the file server
                string index_Path_Destination = icdService.GetRecordPath();

                string filePath = @"c:\";
                string fileName = "ICD_test2.txt";
                Stream fs = File.OpenRead(filePath + fileName);
                byte[] buffer = new byte[8192];
                int totalRead = 0;
                int read = 0;
            }
        }
    }
}
```

```

while ((read = fs.Read(buffer, 0, buffer.Length)) > 0)
{
    fileName = fileName.Replace("\\", "_");
    byte[] data2 = null;
    if (buffer.Length > read)
    {
        data2 = new byte[read];
        Array.Copy(buffer, data2, read);
    }
    //Upload the file to the file server
    icdService.AppendToUpload(iCD_Token.ticket, fileName,
index_Path_Destination, 8192, buffer);
    totalRead += read;
}

//When upload done create meta data in the database for that file
and get the record_detail_id back.
Console.WriteLine(icdService.EndUpload(fileName,
iCD_Token.record_id, totalRead, iCD_Token.recordTitle, iCD_Token.officeSymbol_id,
index_Path_Destination, iCD_Token.ticket, userName));

//Console.WriteLine(t.ticket);
Console.ReadLine();

}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
public void Retrieve()
{
    //Retrieve a record from AEA
    try
    {
        IcdService icdService = new IcdService();
        byte[] readBuffer;
        int chunkSize = 1;
        int i = 1;
        string userName = "userName";
        int record_details_id = 1346306;
        string password = "password";

        Program P = new Program();

        string indexDestinationPath = "";
        string fileName = "";
        icdService.AccessDocument(userName, password, record_details_id,
out indexDestinationPath, out fileName);
        while (chunkSize > 0)
        {
            chunkSize = icdService.RetrieveDocument(fileName,
indexDestinationPath, i, out readBuffer);
            P.AppendToUpload(fileName, "", chunkSize, readBuffer);
            i += 1;
        }
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine();
        }
    }

    //The following method reside and is executed locally on the
    client when you retrieving document.
    public long AppendToUpload(string fileName, string Path, int chunkSize,
    byte[] Data)
    {

        //AeaRoot and Path should both have trailing slashes
        string fullPath=@"C:\" + fileName;

        FileStream fileStream = null;
        long fileLength = 0;

        try
        {
            //open or create the file
            fileStream = new FileStream(fullPath, FileMode.OpenOrCreate,
            FileAccess.Write);

            //go to the end and append the chunk
            fileStream.Seek(fileStream.Length, 0);
            fileStream.Write(Data, 0, chunkSize);
            fileStream.Flush();
            fileLength = fileStream.Length;
        }
        finally
        {
            if (fileStream != null)
                fileStream.Close();
        }

        return fileLength;
    }
}
}

```

4 Methods

Method Name	Description
IcdService	Runs the ICD
ICD_Token	Creates a Token for the ICD
ICD_Token.token	Example of Token.token
GetRecordPath	Retrieve the path for where the record resides
AppendToUpload	Append the file to perform the Upload process
EndUpload	End the Upload Process
AccessDocument	Access the Document
indexDestinationPath	Find the Index destination path
RetrieveDocument	Retrieve the document

5 Web Service Interface Tool Testing Findings

The client interface went through several iterations of unit testing to ensure functionality of the tool itself. After successful unit testing, integration testing began. Based upon integration testing, some adjustments were made. The most significant adjustment was changing the Byte size for the **AppendUpload** method from 8192 to 65536 in order to boost throughput. Once satisfied with the way the tool performed we proceeded with larger scale testing.

NOTE:

Throughput: Byte size for **AppendUpload** operation is now increased to 65536 to boost throughput.

A stress test of 42,793 documents was sent over the course of several days; it processed through the system successfully. This test was to ensure that a long running data load could take place with a large number of documents. Performance tests were run with three (3) separate folders; the results are in the table below.

Table 1 – Performance Test Results

Folder Name	Docs	Rate (Docs/hr)	Time (Hr)	Size (MB)	Avg Size (MB)	Rate (MB/hr)
DAAMS TUBE RESULTS-REPORTS 1991	1089	3843.52	0.2833	247	0.226	871.76
DAAMS TUBE RESULTS-REPORTS 2002	2676	2919.27	0.916	772	0.288	842.18
DAAMS TUBE RESULTS-REPORTS 2007	3253	3253	1	1070	0.328	1070
Total/Average	7018	3253	2.2	2089	0.297	949.54

At the above rates, it is expected that it would take no fewer than forty-four (44) twenty-four-hour (24 hours) days of non-stop uploading to transmit one terabyte of data.