



DuoShare HTTP Distributed Batch Integration Getting Started Guide

Document Date: June 25, 2007

DuoShare HTTP Distributed Batch Integration Getting Started Guide

Copyright Notice

Copyright © 2007 DuoShare, Inc. All rights reserved.

Documentation version 1.7.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of DuoShare, Inc.

Trademarks

W3C[®] is a trademark (registered in numerous countries) of the World Wide Web Consortium; marks of W3C are registered and held by its host institutions MIT, ERCIM, and Keio.

Contact

Registration: <http://duoshare.com/registration.html>

Sales: sales@duoshare.com

Support: support@duoshare.com

Phone: 214-691-4884

Table of Contents

Introduction.....	1
Purpose.....	1
Scope.....	1
Typographical Conventions.....	1
Definitions, Acronyms, and Abbreviations.....	1
What You Need to Know.....	2
References.....	2
Getting Started.....	2
Registering with DuoShare.....	2
Overview.....	3
Creating an Import/Export Mapping.....	3
Connecting to DuoShare.....	4
Communicating with DuoShare (the basics).....	5
Sending a Request.....	5
Receiving a Response.....	5
Checking for Errors.....	6
Uploading a File.....	7
Downloading a File.....	7
Specific calls to DuoShare.....	8
Logging In.....	8
Login.....	8
Login User.....	9
Switch Store.....	9
Switch Customer.....	9
Creating a List.....	10
Start New Import Upload.....	10
Submit Import Upload.....	10
Import Zip File Chooser.....	11
Import Mapping Info.....	11
Create Jobs.....	11
Merging a List.....	12
Epoch.....	12
Import Merge Upload.....	12
Import Merge Upload Results.....	12
Import Merge Zip File Chooser.....	13
Import Merge Example.....	13
Running Mail Preparation Jobs.....	14
Sequence.....	14
Presort.....	14
Presort Results.....	14
Downloading a List.....	15
Export.....	15

Export Results.....	15
Download Job.....	15
Other Calls.....	16
View List.....	16
Appendix A: Samples.....	18
C# .Net 2.0.....	18
Performing a GET request.....	18
Performing a POST request.....	19

Introduction

Purpose

This document serves as a guide for DuoShare distributed batch integration via the *Hypertext Transfer Protocol (HTTP)*. After reading this document you will know how to import, merge, and download address lists using HTTP.

Scope

The integration calls outlined here are for batch operations on a list where an entire list will be manipulated as a result of each call. This is a concrete implementation of DuoShare Address Quality Integrator (DAQI) Distributed Batch Validation. For a higher level view of integration possibilities see the DAQI Abstract Design Document in references.

Typographical Conventions

Different fonts and styles are used throughout this guide to allow certain content to be easily identified.

Examples are indented with numbered lines:

1. A code/protocol example (CR) (LF)
2. A [variable] in the example (CR) (LF)

Other conventions include:

- A `variable` or `Header` referenced from the example above
- A literal *Carriage Return* (CR) and/or *Line Feed* (LF)
- *Italics* are used to denote emphasis on terms that can be found in the index and to reference user-interface elements for some examples.

Definitions, Acronyms, and Abbreviations

Carriage Return (CR): Represents a carriage return character used in HTTP examples.

GET: Method for HTTP requests where query parameters are appended to the request's URI.

HTTPS: HTTP over an encrypted SSL/TLS connection. HTTPS is not a separate protocol from HTTP.

Hypertext Transfer Protocol (HTTP): HTTP is a stateless protocol which defines how messages are formatted, transmitted, and handled for the World Wide Web. This document assumes the use of HTTP version 1.1. The HTTP/1.1 protocol is detailed in RFC 2616. See References.

Integrated Development Environment (IDE): Tools software developers use to increase programming productivity.

Line Feed (LF): Represents a line feed character used in HTTP examples.

Primary Key (PK): Unless otherwise specified, a primary key is a 64 bit two's complement number with a value greater than 0. In simpler terms, the minimum value is 1 and the maximum value is $2^{63}-1$.

POST: Method for HTTP requests where request contents are placed in an entity header portion of the HTTP message.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS): SSL and TLS provide secure communications on the Internet.

Uniform Resource Identifier (URI): RFC 2616 defines them as “Formatted strings which identify a resource.” This guide considers URIs to be either fully qualified web addresses or relative web addresses. For example “<http://duoshare.com/ds/App/Main>” is fully qualified while “/ds/App/Main” is relative.

Extensible Hypertext Markup Language (XHTML): XHTML is a reformulation of HTML 4 in XML 1.0. See references.

Hypertext Markup Language (HTML): HTML is a language for publishing content on the world wide web. See references.

What You Need to Know

You should have an intermediate understanding of HTTP. Debugging is easier if you know how to get a trace of HTTP traffic between DuoShare and your application.

You should know how to:

- Send GET and POST requests using your integration's programming language
- Accept the DuoShare SSL certificate
- Send and receive messages using HTTPS
- Parse an XHTML response for field contents and/or attributes

References

DAQI Abstract Design Document: <http://duoshare.com/DuoSharePostalIntegration.pdf>

SSL: “The SSL Protocol”, Alan O. Freier, Philip Karlton, Paul C. Kocher, 18 November 1996. The SSL 3.0 specification can be found at <http://wp.netscape.com/eng/ssl3/draft302.txt>.

HTTP / RFC 2616: “Hypertext Transfer Protocol – HTTP/1.1”, R. Fielding et al, June 1999. This RFC is available at <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.

XHTML: "XHTML 1.0: The Extensible HyperText Markup Language", S. Pemberton et al. The latest version of this specification is available at <http://www.w3.org/TR/xhtml1>. As of the publication of the current document, XHTML 1.0 is a W3C Proposed Recommendation.

HTML: "HTML 4.01 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999. Latest version available at: <http://www.w3.org/TR/html401>.

Getting Started

Registering with DuoShare

The first step in integrating with DuoShare is to create an account with us. Your account is not charged during

development. For production pricing contact our sales department.

You can register online or by phone. You will be contacted when your account is set up.

Please be sure to let your representative know that you are interested in an integrating with DuoShare. You will receive your initial customer PK, store PK, and company PK which will be necessary for DuoShare integration.

Overview

You will integrate with DuoShare via HTTP by emulating a web browser. Therefore, you may find it easier to integrate if you familiarize yourself with the DuoShare web site by running a few address lists first.

Before you can upload any lists programatically, you will need to define an import/export mapping for your address lists. An import/export mapping is the combination of the file type, number of fields, and field order for importing and exporting your address list. You will need to create one mapping for each list format you have. Details about how to create an import/export mapping are in the section titled “Creating an Import/Export Mapping.”

After you are done creating a mapping, your application will be able to start integrating with DuoShare programatically by:

1. Logging In
2. Creating/Merging a List
3. Running Mail Preparation Jobs
4. Downloading a List

Each of these steps is detailed under “Specific Calls to DuoShare”.

Creating an Import/Export Mapping

Log on to DuoShare and create a new list. Just after you upload your list, you will see the *Enter Mapping Information* page. On this page you should pay careful attention to the *Scope Type* and *Assign To* fields. Your decision here will affect the scope of your integration's users.

Create List:
Enter Mapping Information

Help

Mapping Information

Address List Name:

New Mapping

Mapping Title:

Scope Type:

Assign To:

Predefined Mapping

Predefined Mappings:

Scope Type determines the range of access granted for viewing mappings. Access to mappings can be given to

certificate authority you should ensure that the certificate's subject is `CN=duoshare.com, OU=Corporate, O="DuoShare, Inc", L=Dallas, S=Texas, C=US`.

If you have any errors from an untrusted certificate or the certificate's subject does not match what is provided here, you should call us immediately.

You should see your implementation language's API documentation on implementing SSL support for your requests. All of the requests and response examples shown here are in plain text.

Communicating with DuoShare (the basics)

Sending a Request

All requests should specify the `User-Agent` header to help us identify your integration. This can be the name of your integration and its version number.

Below is an example POST request attempting to perform a login to DuoShare.

```
1. POST /ds/j_security_check HTTP/1.1 (CR) (LF)
2. User-Agent: [useragent] (CR) (LF)
3. Host: duoshare.com (CR) (LF)
4. Connection: Keep-Alive (CR) (LF)
5. Content-Length: [length] (CR) (LF)
6. Content-Type: application/x-www-form-urlencoded (CR) (LF)
7. (CR) (LF)
8. j_username=[user] & j_password=[pass] (CR) (LF)
```

For security reasons, you should always supply credentials in a POST rather than a GET. Here is a valid example of a GET request.

```
1. GET /ds/App/PostalJob?task=Presort&listPK=[listpk] HTTP/1.1 (CR) (LF)
2. User-Agent: [useragent] (CR) (LF)
3. Host: duoshare.com (CR) (LF)
4. Cookie: LtpaToken2=[ltpatoken2]; LtpaToken=[ltpatoken] (CR) (LF)
```

Receiving a Response

After sending your request, you will receive a response from the server. Because covering every possible HTTP response code is beyond the scope of this document, you should familiarize yourself with *RFC 2616*.

Here is a response example for the login request above:

```
1. HTTP/1.1 302 Found (CR) (LF)
2. Content-Language: en-US (CR) (LF)
3. Content-Length: 0 (CR) (LF)
4. Location: https://duoshare.com/ds/ (CR) (LF)
5. Set-Cookie: LtpaToken2=[token2] (CR) (LF)
```

Which instructs you to redirect to `Location`. You should note the `Set-Cookie` header, you will need to store the cookies set by the server so you can supply them in subsequent requests.

After following the redirection, the final response from DuoShare is:

```
1.HTTP/1.1 200 OK (CR) (LF)
2.Content-Language: en-US (CR) (LF)
3.Content-Length: 1696 (CR) (LF)
4.Content-Type: text/html;charset=ISO-8859-1 (CR) (LF)
5.Date: Thu, 08 Feb 2007 20:48:39 GMT (CR) (LF)
6.Set-Cookie: JSESSIONID=[jsessionid]; Path=/ (CR) (LF)
7.Server: WebSphere Application Server/6.0 (CR) (LF)
8. (CR) (LF)
9.[1696 bytes of text/html]
```

Checking for Errors

You should check all responses for errors, even when you don't need to store anything from the response. Some of the calls that need to be made will set session variables, if any of them fail, these variables may not be set and undefined behavior will result.

Aside from handling any HTTP or networking errors you will also need to parse the XHTML responses from DuoShare.

Some responses will contain an XHTML table with status information. If this table is present, and the `message` field does not contain an expected success message, you should treat the message as an error.

```
1.<tr>
2.  <td class="cellh5">Message</td>
3.  <td id="message">All fields are required.&nbsp;</td>
4.</tr>
5.<tr>
6.  <td class="cellh5">Detail</td>
7.  <td id="detail">&nbsp;</td>
8.</tr>
9.<tr>
10. <td class="cellh5">Source</td>
11. <td id="source">class com.ds.servlet.PostalJobHandler&nbsp;</td>
12.</tr>
13.<tr>
14. <td class="cellh5">Time</td>
15. <td id="date">Fri Feb 09 11:44:22 CST 2007&nbsp;</td>
16.</tr>
```

You can parse this response to display an error message for your end-user. Messages containing *"has been created"* or *"have been created"* are common in success messages. You should do a case-insensitive check when checking for these two success messages.

Unexpected errors on the server are shown on a page with the `title` element set to “*error*”. You should check for elements with the `errorClass` and `errorStackTrace` identifiers to extract information about the error that has occurred. An example of this kind of error is shown below.

1. `<p>Class that reported the error and description (in attribute request.error:) <strong id="errorClass">class javax.ejb.ObjectNotFoundException</p>`
2. `<p>Complete message:</p>`
3. `<pre id="errorStackTrace">`
4. `javax.ejb.ObjectNotFoundException: 59395654`
5. `at com.ibm.ejs.container.EJSHome....`
6. `</pre>`

Uploading a File

Uploading a file to DuoShare is very similar to any other POST request. You will need to change the `Content-Type` to match the file you are uploading. For batch operations, you will need to use *zip* files.

Here is an example of a request to upload an address list inside a zip archive named `list.zip`.

1. `POST /ds/App/PostalJob?task=ImportUploadFileResults&Submit=Upload&callURL=StartNew HTTP/1.1 (CR) (LF)`
2. `Content-Type: multipart/form-data; boundary=-----8c919a75d264a5a (CR) (LF)`
3. `User-Agent: [useragent] (CR) (LF)`
4. `Host: duoshare.com (CR) (LF)`
5. `Cookie: [cookies] (CR) (LF)`
6. `-----8c919a75d264a5a (CR) (LF)`
7. `Content-Disposition: form-data; name="file"; filename="list.zip" (CR) (LF)`
8. `Content-Type: application/x-zip-compressed (CR) (LF)`
9. `(CR) (LF)`
10. `[content] (CR) (LF)`
11. `-----8c919a75d264a5a (CR) (LF)`

Downloading a File

Downloading a file is the reverse of uploading a file. However, you will not need to concern yourself with the boundaries when downloading a file from DuoShare.

When you receive a response you will read the content as bytes instead of a string. The `Content-Type` header will help you determine what kind of content you are downloading, when downloading from DuoShare this will usually be set to `zip`. You can use the `Content-Disposition` header to retrieve a suggested filename.

Here is an example of downloading a file where `content` is the file's bytes:

- 1.HTTP/1.1 200 OK (CR) (LF)
- 2.Content-Disposition: filename=list.zip (CR) (LF)
- 3.Content-Language: en-US (CR) (LF)
- 4.Content-Length: 524588 (CR) (LF)
- 5.Content-Type: zip (CR) (LF)
- 6.Date: Tue, 13 Feb 2007 17:09:43 GMT (CR) (LF)
- 7.Server: WebSphere Application Server/6.0 (CR) (LF)
8. (CR) (LF)
9. [content] (CR) (LF)

Specific calls to DuoShare

This section describes individual calls you will need to make to DuoShare. Each call is formatted like so:

Method: Specifies if the call is a POST or GET.

URI: URI relative to the host.

Query Parameters: Parameters that should be supplied in the request. In a GET these are appended to the end of the URI. In a POST these are supplied as the content.

Expected Response Type/Title: The value you should expect to appear in the `title` tag of the response. You should do case insensitive comparisons when checking for a correct title. If the call does have an XHTML response type, this will be the value of the response type to expect.

Prerequisites: Calls that must be successfully made prior to making this call. The entire *Logging In* section must be completed prior to making any other calls.

Remarks: Additional notes about the call.

Logging In

Logging in is a four step process.

1. *Login*
2. *Login User*
3. *Switch Store*
4. *Switch Customer*

Login

Method: POST

URI: /ds/j_security_check

Query Parameters:

- `j_username`: The user's username
- `j_password`: The user's password

Expected Response Title: Varies depending on the user's session state. If the user attempted to access a page without having been logged in, he/she is redirected to the login page. After logging in the user will be taken back to the page he/she was trying to access. It is recommended that you always login from a clean session state, that is don't supply any cookies from a previous session.

Prerequisites: None.

Remarks: If the response contains “*Login Error*” or `title` is “*login.html*” there was an error logging in. After checking for general errors, call *LoginUser* to finish logging in.

Login User

Method: GET

URI: /ds/App/Main?task=LoginUser

Query Parameters: None. Already included in *URI*.

Expected Response Title: Like *Login*, this call also varies depending on the user's session state. If the user attempted to access a page without having been logged in, he/she is redirected to the login page. After logging in the user will be taken back to the page he/she was trying to access. It is recommended that you always login from a clean session state, that is don't supply any cookies from a previous session.

Prerequisites: *Login*

Remarks: After a successful login there will be a tag with an element containing the user's `userPK`. You can find it in by parsing the following a element where `x` is the `userPK`.

```
<a href="/ds/App/User?task=ViewUser&userPK=x">
```

Switch Store

Method: GET

URI: /ds/App/Main?task=LoginUserResults&page=MenuMain

Query Parameters: Some parameters are already included in the *URI*.

- `login`: A string formatted like so “`S:companyPK-storePK`” where `companyPK` and `storePK` are your company primary key and store primary key. *e.g.* `Login=S:1-1`

Expected Response Title: See *Login User* for more information about redirection during login. When called directly, this should take you a page titled “*Main Menu*”. If the `title` of the response page is “*login.html*” your cookie/session most likely timed out and you should login again with a new session state.

Prerequisites: *Login User*

Remarks: None.

Switch Customer

Method: GET

URI: /ds/App/Main?task=LoginUserResults&page=MenuMain

Query Parameters: Some parameters are already included in the *URI*.

- **login:** A string formatted like so "C:customerPK-" where `customerPK` is the customer primary key. Note the trailing hyphen

Expected Response Title: See *Login User* for more information about redirection during login. When called directly, this should take you a page titled "Main Menu". If the `title` of this page is "login.html" your cookie/session most likely timed out and you should login again with a new session state.

Prerequisites: *Switch Store*

Remarks: None.

Creating a List

1. Start New Import Upload
2. Submit Import Upload
 - 2.1 Import Zip File Chooser
3. Import Mapping Info
4. Create Jobs

Start New Import Upload

Method: GET

URI: /ds/App/PostalJob?task=ImportUploadFile&callURL=StartNew

Query Parameters: None. Already included in *URI*.

Expected Response Title: "Upload File"

Prerequisites: None.

Remarks: None.

Submit Import Upload

Method: POST

URI: /ds/App/PostalJob?task=ImportUploadFileResults&Submit=Upload
&callURL=StartNew

Query Parameters: Some are already included in *URI*. The following headers are required after the boundary. Set `filename` to the name of your file.

- **Content-Disposition:** form-data; name="file";
filename="[filename]"
- **Content-Type:** application/x-zip-compressed

Expected Response Title: “Enter Mapping Information” or “Choose File To Import”

Prerequisites: *Start New Import Upload*

Remarks: This is where you will upload the file. Supply the query parameters in the URI above and upload the file as shown in *Uploading a File*.

There are two possible response titles. When `title` is “Enter Mapping Information” you should move on to *Import Mapping Info*. If `title` is “Choose File To Import” go to *Import Zip File Chooser*.

Import Zip File Chooser

Method: GET

URI: /ds/App/PostalJob?task=ImportZipFileChooserResults

Query Parameters: Some are already included in *URI*.

- `fileName`: The name of the file *inside* of the zip archive uploaded in *Submit Import Upload*. For example, if your zip file contains multiple files and a file named “*addresslist.csv*” is the name of the address list you are uploading, this parameter would be “*addresslist.csv*”.

Expected Response Title: “Enter Mapping Information”

Prerequisites: *Submit Import Upload* and the response `title` was “Choose File To Import”

Remarks: None.

Import Mapping Info

Method: GET

URI: /ds/App/PostalJob?task=ImportMappingInfoResults&callURL=ReviewMappingSummary&mappingSave=predefined

Query Parameters: Some are already included in *URI*.

- `selection`: This should be the document id for your mapping

Expected Response Title: “Review Mappings” even if the `title` is correct, the response may contain error messages about an invalid mapping.

Prerequisites: *Submit Import Upload*

Remarks: None.

Create Jobs

Method: GET

URI: /ds/App/PostalJob?task=CreateJobsResults&Submit=Finish

Query Parameters: Some are already included in *URI*.

- `serverPK`: If you want the import to schedule jobs to initially run on a specific server,

provide the server's primary key here. This parameter is optional, if you do not need to use a specific server, simply exclude this parameter from the query.

Expected Response Title: *“Congratulations!”*

Prerequisites: *Import Mapping Info*

Remarks: This call should be parsed for the new `listPK`. You can find it by parsing the response for `listPK="x"` where `x` is the list's primary key.

Merging a List

1. *Epoch*

2. *Import Merge Upload*

3. *Import Merge Upload Results*

 3.1 *Import Merge Zip File Chooser*

4. *Import Merge Example*

Epoch

Method: GET

URI: /ds/App/PostalJob?task=StartEpoch

Query Parameters:

- `listPK`: Primary key for a list you have previously created

Expected Response Title: *“View List”* with message *“Epoch Has Been Created”*

Prerequisites: List exists.

Remarks: None.

Import Merge Upload

Method: GET

URI: /ds/App/PostalJob?task=ImportMergeUpload

Query Parameters:

- `listPK`: Primary key for a list you have previously created

Expected Response Title: *“Upload File”*

Prerequisites: *Epoch*

Remarks: None.

Import Merge Upload Results

Method: POST

URI: /ds/App/PostalJob?task=ImportMergeUploadResults

Query Parameters: Some are already included in *URI*.

The following headers are required after the boundary. Set `filename` to the name of your file.

- `Content-Disposition`: form-data; name="file"; filename="[filename]"
- `Content-Type`: application/x-zip-compressed

Expected Response Title: “*Example of Import*” or “*Choose File To Import*”

Prerequisites: *Import Merge Upload*

Remarks: This is where you will upload the file. Supply the query parameters in the URI above and upload the file as shown in *Uploading a File*.

There are two possible response titles. When `title` is “*Example of Import*” you should call *Import Merge Example*. If `title` is “*Choose File To Import*” call *Import Merge Zip File Chooser*.

Import Merge Zip File Chooser

Method: GET

URI: /ds/App/PostalJob?task=ImportMergeZipFileChooserResults

Query Parameters: Some are already included in *URI*.

- `fileName`: The name of the file *inside* of the zip archive uploaded in *Submit Import Upload*. For example, if your zip file contains multiple files and a file named “addresslist.csv” is the name of the list you are uploading, this parameter would be “addresslist.csv”.

Expected Response Title: “*Example of Import*”

Prerequisites: *Import Merge Upload Results* and the response `title` was “*Choose File To Import*”

Remarks: None.

Import Merge Example

Method: GET

URI: /ds/App/PostalJob?task=ImportMergeExampleResults&Submit=Merge

Query Parameters: Some are already included in *URI*.

- `purge`: If you would like to remove the addresses that are in the DuoShare list, but not in your csv file, you should set this to “purge”. If you would like to keep all addresses in DuoShare, even those that are not in the list you are uploading, this should be an empty string.

Example:

```
/ds/App/...&Submit=Merge&purge=purge
```

or

/ds/App/...&Submit=Merge&purge=

Expected Response Title: N/A (If no errors are present, the call was successful)

Prerequisites: *Import Merge Upload Results*

Remarks: None.

Running Mail Preparation Jobs

Sequence

Method: GET

URI: /ds/App/PostalJob?task=SequenceResults

Query Parameters: Some are already included in *URI*.

- **listPK:** Primary key for a list you have previously created

Expected Response Title: None.

Prerequisites: List exists.

Remarks: The result page will have the message, “Sequence Job Has Been Created”, when successful.

Presort

Method: GET

URI: /ds/App/PostalJob?task=Presort

Query Parameters: Some are already included in *URI*.

- **listPK:** Primary key for a list you have previously created

Expected Response Title: “*Submit Jobs*”

Prerequisites: *Sequence*.

Remarks: This method preps the next required call, *Presort Results*. For a *Presort* job to run *Presort Results* must be the next call.

Presort Results

Method: GET

URI: /ds/App/PostalJob?task=PresortResults&submit=Finish

Query Parameters: Some are already included in *URI*.

- **containerType:** containerType
- **rateClassID:** rateClassID
- **maxPerContainer:** maxPerContainer

- `originZipCode`: Origin Zip Code
- `goodAddressesOnly`: Set to “true” to virtually eliminate return mail by only mailing to 'good' addresses. Otherwise, set to “false”

Expected Response Title: “View List” with message “Presort Job Has Been Created”

Prerequisites: *Presort*.

Remarks: None.

Downloading a List

1. *Export*
2. *Export Results*
3. *Download Job*

Export

Method: GET

URI: /ds/App/PostalJob?task=export

Query Parameters: Some are already included in *URI*.

- `listPK`: Primary key for a list you have previously created

Expected Response Title: “Export Options”

Prerequisites: List exists and has an import/export mapping.

Remarks: None.

Export Results

Method: GET

URI: /ds/App/PostalJob?task=exportResults&Submit=Submit

Query Parameters: None. Already included in *URI*.

Expected Response Title: None.

Prerequisites: *Export*.

Remarks: The result page will have the message, “Recalculate Postal Stats and Export Jobs have been created”, when successful.

Download Job

Method: GET

URI: /ds/App/PostalJob?task=DownloadJob

Query Parameters: Some are already included in *URI*.

- `listPK`: Primary key for a list you have previously created

Expected Response Type: zip

Prerequisites: The list with the specified `listPK` must meet the following requirements:

- The list exists
- The list has been exported
- The list's `status` is “Export Finished”

If other jobs were run on this list after the last export job finished, calling *Download Job* may return an error page. The results are undefined when an export job is currently running on the list. If the list's status is not “Export Finished” and this call does succeed, you are most likely downloading an outdated version of the list.

Remarks: See **Downloading a File** on page 7 for more information about reading the HTTP response.

Other Calls

View List

Method: GET

URI: /ds/App/List?task=ViewList

Query Parameters: Some are already included in *URI*.

- `listPK`: Primary key for a list you have previously created

Expected Response Title: “View List”

Prerequisites: The list exists.

Remarks: This call will allow you to obtain the `status` of a list. To obtain the list's status, extract the `alt` attribute from the `img` tag where `id` is “status” . In the example below, `status` is “Presort Finished.”

```
1.<tr>
2.  <td>Status</td>
3.  <td>
4.      <strong>
5.          
9.          
12.      </strong>
13. </td>
14. <td class="errordisplay"></td>
15.</tr>
```

You can also use web services to obtain the list's status. See the ListEF service in the *DuoShare Web Services Integration: Getting Started Guide* for information.

Appendix A: Samples

C# .Net 2.0

Performing a GET request

```
1.String DoGet(Uri uri, System.Net.CookieContainer cookies)
2.{
3.    if (uri == null)
4.        throw new ArgumentNullException("uri");
5.
6.    System.Net.HttpWebRequest httpRequest =
7.        (System.Net.HttpWebRequest)System.Net.HttpWebRequest.Create(uri);
8.
9.    httpRequest.Method = "GET";
10.   httpRequest.UserAgent = "DuoShare Getting Started Sample";
11.   httpRequest.CookieContainer = cookies;
12.
13.   using (System.Net.HttpWebResponse httpResponse =
14.       (System.Net.HttpWebResponse)httpRequest.GetResponse())
15.   {
16.       using (System.IO.StreamReader sr =
17.           new System.IO.StreamReader(httpResponse.GetResponseStream()))
18.       {
19.           String result = sr.ReadToEnd();
20.           return result;
21.       }
22.   }
23.}
```

Performing a POST request

```
1. String DoPost(Uri uri, System.Net.CookieContainer cookies,
2.   byte[] contentBytes, String contentType)
3. {
4.   if (uri == null)
5.     throw new ArgumentNullException("uri");
6.
7.   if (contentBytes == null)
8.     throw new ArgumentNullException("contentBytes");
9.
10.  System.Net.HttpWebRequest httpRequest =
11.    (System.Net.HttpWebRequest) System.Net.HttpWebRequest.Create(uri);
12.
13.  httpRequest.Method = "POST";
14.  httpRequest.UserAgent = "DuoShare Getting Started Sample";
15.  httpRequest.ContentType = contentType;
16.  httpRequest.ContentLength = contentBytes.Length;
17.  httpRequest.CookieContainer = cookies;
18.
19.  using (System.IO.Stream requestStream = httpRequest.GetRequestStream())
20.  {
21.    requestStream.Write(contentBytes, 0, contentBytes.Length);
22.    requestStream.Flush();
23.  }
24.
25.  using (System.Net.HttpWebResponse httpResponse =
26.    (System.Net.HttpWebResponse) httpRequest.GetResponse())
27.  {
28.    //Check the response for valid login
29.    using (System.IO.StreamReader sr =
30.      new System.IO.StreamReader(httpResponse.GetResponseStream()))
31.    {
32.      String result = sr.ReadToEnd();
33.      return result;
34.    }
35.  }
36. }
```


Alphabetical Index

C

Carriage Return
 CR 1
Content-Type
 zip 16
cookie 6
cookie
 Set-Cookie 6
Creating a List
 Create Jobs 11
 Import Mapping Info 11
 Import Zip File Chooser 11
 Start New Import Upload 10
 Submit Import Upload 10

D

DAQI Abstract Design Document 2
Downloading a File 7
Downloading a List
 Download Job 15
 Export 15
 Export Results 15
 View List 16

E

Error Checking
 message
 Epoch Has Been Created 12
 Presort Job Has Been Created 15
 Recalculate Postal Stats and Export Jobs have been created 15
errorClass 7
errorStackTrace 7

G

GET 8
 definition 1
 example, C# 18
 example, HTTP 5

H

HTML 2
HTTPS 1
Hypertext Transfer Protocol
 Headers
 Content-Disposition 7, 10, 13
 Content-Type 7, 10, 13
 Location 6
 Set-Cookie 6
 User-Agent 5

HTTP 1, 2
 Integrated Development Environment
 IDE 1
 Line Feed
 LF 1
 Logging In
 Login 8
 Login User 9
 Switch Customer 9
 Switch Store 9
 Merging a List
 Epoch 12
 Import Merge Example 13
 Import Merge Upload 12
 Import Merge Upload Results 12
 Import Merge Zip File Chooser 13
 POST 7p.
 definition 2
 example, C# 19
 example, HTTP 5
 Prerequisites 8
 Primary Key
 PK 1
 RFC 2616 2
 Running Mail Preparation Jobs
 Presort 14
 Presort Results 14
 Sequence 14
 Samples
 C#
 GET 18
 POST 19
 Secure Sockets Layer
 SSL 2
 Title
 Choose File To Import 11, 13
 Congratulations! 12
 definition 8
 Enter Mapping Information 11
 error 7

I

L

M

P

R

S

T

Example of Import 13
Export Options 15
login.html 9p.
Main Menu 9p.
Review Mappings 11
Submit Jobs 14
Upload File 10, 12
View List 12, 15p.
Transport Layer Security
TLS 2

U

Uniform Resource Identifier
URI 2, 8
Uploading a File 7

X

XHTML 2
