



BIC Download Automation

Interface Specification

These technical specifications provide detailed information about the BIC Download Automation. It is intended for developers, IT operations, and architects.

Version, July 2010

Table of contents

1	Introduction	3
2	References	3
3	Interface description.....	3
4	Errors and warnings	8
5	Authentication and authorisation.....	8
6	Writing a Java client	9
	Revision record	Error! Bookmark not defined.
	End of document	Error! Bookmark not defined.

1 Introduction

SWIFT has made a new set of web resources available. These resources allow customers, which subscribed to the BIC download service, to programmatically and automatically download BIC publication files.

This document describes how SWIFT customers can use this new service from the SWIFT website, and how programmers can write code to invoke the service.

The main targeted audience is developers, IT operations, and architects.

2 References

To ensure an adequate use of the service, it is mandatory to read the user manual before using the information contained in this document. The user manual provides valuable information about the password policy, subscription to the services, account management, and other similar information. This manual is available on swift.com.

3 Interface description

Automatic download of BIC files is available through a service accessible only through HTTPS. The service base URL is:

<https://www2.swift.com/bicdownload/>

The service mapped name is:

`/bicdownloader`

Currently only one action is defined for this service. It is the default action when the action is omitted from the service call.

The `getFile` action defines the following parameters:

Parameter	Description	Data type	Mandatory	Default value
<code>year</code>	The activation year (for example, 2007)	Integer (4)	No	Latest
<code>month</code>	The activation month (for example, 01 for January)	Integer (2)	No	Latest
<code>content</code>	<code>full</code> or <code>delta</code>	String	Yes	
<code>productline</code>	Product category (for example, <i>SWIFTAlliance Bank File</i>)	String	Yes	

Parameter	Description	Data type	Mandatory	Default value
product	Product under product category (for example, Bankfile)	String	Yes	
format	File content format, currently limited to <code>txt</code> , <code>dat</code> , <code>ebcdic</code> , <code>dos</code>	String	Yes	
platform	Windows ZIP or UNIX compressed tar file (<code>tar.Z</code>)	String	Yes	

The following table provides the description of the parameters and associated constraints.

year	When omitted, it defaults to the year of the latest available file of the selected type. The year is always 4 digits long, for example 2007.
month	When omitted, it defaults to the month of the latest available file of the selected type. The month must be two digits long representing the number of the month in the year, for example, 01 for January and 12 for December.
content	<p>“full” or “delta” are the only valid types. Invalid values will return an HTTP error.</p> <p><i>Note:</i> Country/Currency/Holidays (CCH) product does not have delta content type.</p>
productline	<p>There are six product families, these are: <i>SWIFT Alliance Bank File</i>, <i>BIC Directory</i>, <i>BIC Database Plus</i>, <i>BIC Plus IBAN Directory</i>, <i>SEPA Routing Directory</i>, and <i>Country/Currency/Holidays (CCH)</i>. Each has one or more products.</p> <p>Valid parameter values for <code>productline</code> are respectively:</p> <pre> swbankfile bicdir bicplusiban separouting els1 wr ssi cch </pre>

	<p>Invalid values will return an HTTP error.</p>																
product	<p>The following products are possible for each combination:</p> <p><i>SWIFT Alliance Bank File:</i> Bankfile <i>BIC Directory:</i> BICDB, Bankfile, BICDU <i>BIC Plusiban Directory:</i> BICPLUSIBAN, MOREBIC <i>Sepa Routing Directory:</i> SEPAROUTING <i>Euro1/Step1 Directory:</i> E1S1 <i>Worker's Remittances Directory:</i> LIVE, TEST <i>SSI Directory:</i> SSI <i>for CCH:</i> CCH, Other data</p> <p>Valid possible parameter values for each product line are:</p> <table> <tr> <td>swbankfile</td> <td>bankfile</td> </tr> <tr> <td>bicdir</td> <td>bicdb, bankfile, bicdu</td> </tr> <tr> <td>bicplusiban</td> <td>bicplusiban, morebic</td> </tr> <tr> <td>separouting</td> <td>separouting</td> </tr> <tr> <td>els1</td> <td>els1</td> </tr> <tr> <td>wr</td> <td>live, test</td> </tr> <tr> <td>ssi</td> <td>ssi</td> </tr> <tr> <td>cch</td> <td>cch, other</td> </tr> </table> <p>Invalid values will return an HTTP error.</p>	swbankfile	bankfile	bicdir	bicdb, bankfile, bicdu	bicplusiban	bicplusiban, morebic	separouting	separouting	els1	els1	wr	live, test	ssi	ssi	cch	cch, other
swbankfile	bankfile																
bicdir	bicdb, bankfile, bicdu																
bicplusiban	bicplusiban, morebic																
separouting	separouting																
els1	els1																
wr	live, test																
ssi	ssi																
cch	cch, other																
format	<p>Valid possible parameter values are:</p> <p>txt dat ebcdic dos</p> <p>Invalid values will return an HTTP error.</p> <p>Some formats are not available for some products. Valid combinations are described hereafter.</p>																
platform	<p>File request file is either a Windows ZIP file or a UNIX compressed tar file (*.tar.Z). Valid parameter values are:</p> <p>win unix</p> <p>Invalid values will return an HTTP error.</p> <p>This parameter is mandatory when producline=swbankfile and is ignored otherwise.</p>																

--	--

IMPORTANT NOTES:

- Header field `content-length` is set to its correct value, that is, the actual file size. This allows developers to proactively manage disk space availability if required. Alternatively, the file size can be an indication of download success if an HTTP error is returned. In that case developers may decide to ignore the error.
- The service name defaults to the action `getFile` when the action is omitted. However, it is highly recommended to explicitly enter `action=getfile` for clarity of code and maintainability. Also, it is not guaranteed that this action will continue to be the default one in the future.
- Wildcards are not allowed.
- When a parameter is not relevant in the context of a product it will be ignored, if provided.

Valid parameter combinations are described in the table below.

productline=swbankfile

year	month	content	product	format	platform
4digits	{1 to 12}	{full, delta}	bankfile	{txt, dat}	{win, unix}

productline=bicdir

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{bicdb, bankfile, bicdu}	{txt, dat, ebcdic, dos}

productline= bicplusiban

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{bicplusi ban, morebic}	{txt }

productline= separouting

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{separouting}	{txt }

productline= els1

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{els1}	{txt }

productline= wr

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{live, test}	{txt }

productline= ssi

year	month	content	product	format
4digits	{1 to 12}	{full, delta}	{ss1}	{txt }

productline= cch

year	month	content	product	format
4digits	{1 to 12}	full	{cch, other}	{txt, dat, dos}

There are exceptions to the combinations described here above. For instance, `bicdirdbplus` and `bicdbp` exist in `ebcdic` format only for full content. These cases will return HTTP 404 with a comprehensive message in the body.

4 Errors and warnings

Developers must manage HTTP errors if they want to avoid abnormal program termination. This is achieved by evaluating the HTTP return code. Note, HTTP 200 OK always corresponds to successful responses.

Managed HTTP errors

- Error 404 is returned if the file does not exist or if the parameter combination is not valid. This error can be detected by the service and contains a comprehensive message in the body in plain text.
- Error 500 results from internal erratic conditions in the server environment. It also contains a comprehensive message in the body.

Runtime HTTP errors

All other HTTP error codes go unnoticed to the service and as such are not managed in that there is no explicit message added to the response body. The client program must consider these as runtime errors.

5 Authentication and authorisation

This service imposes the **non pre-emptive basic authentication method**, which mandates that client applications set the HTTP header with a username and password pair for every request.

This service imposes also the **cookie based method and follow redirects method**, which mandates that client applications must keep the cookies returned to the requestor, because the cookies are used during the HTTP redirection (302 HTTP code).

How the SWIFT authentication and authorization works:

- The client requests the URL
- The server gives an unauthorized response and requests a basic authentication challenge.
- The client sends the basic authentication challenge.
- The server authorizes the access and gives a redirection response (302 HTTP code) with cookies.
- The client accepts the redirection and request the new URL with the cookies received by the server.
- The server serves the initial URL request.

Please refer to RFC-2617 for a full description of Basic HTTP Authentication framework. Specifically, if HTTP client libraries other than Apache are selected, particular attention must be paid to the way the HTTP challenge is managed.

6 Writing a Java client

The following Java code snippet illustrates a sample client application. It assumes the use of the Apache HTTP Client library (for more information, see <http://commons.apache.org/httpclient>).

It must be understood that this code sample is provided for illustration purposes, and is subject to [the terms and conditions published on swift.com](#).

Because the request goes over HTTPS, it is required to load the relevant root certificate in the JVM, if you use Java to code your client. In the following example, we used SWIFT CA as root certificate.

For reminder, the command is:

```
keytool -import -keystore $JAVA_HOME/jre/lib/security/cacerts" -storepass $PASS -  
alias "SWIFT CA" -file $CERTFILE
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.util.StringTokenizer;
import java.util.Properties;
import org.apache.commons.httpclient.DefaultHttpMethodRetryHandler;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.Header;
import org.apache.commons.httpclient.HttpException;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.params.HttpMethodParams;
import org.apache.commons.httpclient.UsernamePasswordCredentials;
import org.apache.commons.httpclient.auth.AuthScope;

/*
 * Created on Mar 2, 2007
 * S.W.I.F.T. s.c.r.l.
 */

public class BICDownloader {

    public static void main(String[] args) {
        int exitcode = 0;
        int statusCode;
        String url = "https://www2.swift.com/bicdownload/bicdownloader?"
            + "action=getfile&productline=bicdir&product=bicdb&content=delta&format=txt";
        HttpClient client = new HttpClient();
        GetMethod method = new GetMethod(url);

        // Provide custom retry handler if required
        method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
            new DefaultHttpMethodRetryHandler(3, false));

        try {
            // We strongly recommend obfuscation of password and restricted access to its storage
```

```
Properties prop = new Properties();
InputStream fis = (InputStream)new FileInputStream("E:/KK/bicdownload.prop");
prop.load(fis);

// Set Credentials
UsernamePasswordCredentials credentials;
credentials = new UsernamePasswordCredentials(prop.getProperty("username"),
                                              prop.getProperty("password"));
client.getState().setCredentials(new AuthScope("www2.swift.com", 443), credentials);

// Executing the method.
statusCode = client.executeMethod(method);

if (statusCode != HttpStatus.SC_OK) {
    // Handling HTTP error 404 and 500 not covered in this example
    // All http error cause in this example exit with status 1.
    System.err.println("Method failed: " + method.getStatusLine() + "\n" +
                      method.getResponseBodyAsString());
    System.out.println(method.getRequestCharset() + "\n" + method.getRequestHeader("").toString());
    exitcode = 1;
}
else {

    //Get the file size from the response body and do something with it
    Header[] contlen = method.getResponseHeaders("Content-Length");
    if (contlen.length != 0) {
        StringTokenizer stone = new StringTokenizer(contlen[0].getValue(), "=");
        int size = new Integer(stone.nextToken()).intValue();
        // Do something with the file size
        System.out.println("File size is: " + size);
    }

    // Get the filename from the response body.
    InputStream is = method.getResponseBodyAsStream();
    Header[] contdisp = method.getResponseHeaders("Content-Disposition");

    String filename = null;
    StringTokenizer sttwo = new StringTokenizer(contdisp[0].getValue(), "=");
```

```
        while (sttwo.hasMoreTokens()) filename = sttwo.nextToken();

        // Hardcoded disk and directory path are indicative
        FileOutputStream fos = new FileOutputStream("E:/ZZ/" + filename);
        byte[] buffer = new byte[4096];

        int count = is.read(buffer);
        while (count != -1) {
            fos.write(buffer, 0, count);
            count = is.read(buffer);
        }
        fos.flush();
        fos.close();
        is.close();
    }
} catch (HttpException e) {
    exitcode = 2;
    System.err.println("Fatal HTTP Error: " + e.getMessage());
    e.printStackTrace();
} catch (IOException e) {
    exitcode = 3;
    System.err.println("Fatal I/O error: " + e.getMessage());
    e.printStackTrace();
} finally {
    // Release the connection.
    method.releaseConnection();
    System.exit(exitcode);
}
System.out.println("Download done");
}
```

