“BARCODE” WEB SERVICE
MANUAL CLIENT API

Version 2.4
July 2018

SWISS POST

Table of contents

References

[1] "Barcode" web service, manual, http://www.swisspost.ch/post-barcode-handbuch.pdf

[2] Documentation about "Barcode" web service, http://www.swisspost.ch/post-barcode-cug.htm

1 Introduction

This documentation explains the connection procedure for the "Barcode" web service Version 2.4 using the client API libraries for Java and C# / .NET. The use of the libraries to access the web service is explained by means of a number of typical examples. Each example is presented in C# and Java.

This documentation should not be regarded either as complete client API documentation or as a full description of the web service.

1.1 "Barcode" web service client API documentation

For full documentation of the entire client library API, please refer to the source code documentation for the libraries, which is attached in the distribution as HTML.

– **Java:** The Javadoc API documentation can be found in the **doc** directory, and the sources are located in the source ZIP.
– **C#/.NET:** The C# API documentation can be found in the **doc** directory, the sources are located in the source ZIP archive.

We have marked all sections with changes to content with a line at the margin.

1.2 Web service documentation

For full documentation of the "Barcode" web service, please refer to the Web Service User Manual [1], which also explains all the terms, attributes, service codes etc. Additional resources and tools for the "Barcode" web service can be found on the "Barcode" web service homepage [2].

1.3 Source examples

The examples on the use of the client library are also attached to the Library Distribution as sources, in the **examples** directory.

2 Configuration

The following example illustrates the simple initialization procedure for the "Barcode" web service.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("your username", "your password");

// Now the web service operations can be called on the service instance.
```

```
== C# Code: ==

BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "your username";
service.Password = "your password";

// Now the web service operations can be called on the service instance.
```

The desired language for any error and warnings messages is selected in the "Barcode" web service constructor. The following settings are also mandatory:

– **setLoginCredentials – Basic Authentication:** Username and password for authentication with the web service (Note: in C# these values are set using the corresponding **UserName** and **Password** properties)
– **setFrankingLicense – Franking Licence:** A valid franking licence for generating barcode labels. However, this setting is only essential for calling up the **generateLabel** operations (Note: in C# the franking licence is set using the **Franking-License** property)

Once a BarcodeWebService object has been instantiated, various other settings for the web service can be easily configured using the appropriate setter methods. The following settings are available and have already been pre-initialized with appropriate default settings. If a setting is not explicitly configured, the default standard values will be used.

Setting property	Meaning	Default standard value
WebserviceEndpointURL	The URL at which the web service will be used. This setting can, for example, be useful for test purposes with a special test server	Productive URL of the "Barcode" web service: https://wsbc.post.ch/wsbc/barcode/v2_4
LabelLayout	Layout of the address labels to be generated: A5, A6, A7 or FE	A6
ImageFileType	Image or file format of the address labels: PNG, GIF, PDF, SPDF, EPS, ZPL2 or JPG	PNG
ImageResolution	Resolution of the address labels in DPI: 200, 300 or 600.	300
PrintAddresses	Setting for which addresses are to be printed on the address labels: RECIPIENT_AND_CUSTOMER, ONLY_RECIPIENT, NONE	RECIPIENT_AND_CUSTOMER
PPFranking	Specifies whether the "PP Franking" note should be printed on the label.	False
CustomerSystem	Serves to identify WSBC Services' customer system. Makes it easier to understand problems that arise.	Zero

2.1 Proxy configuration

If a proxy is being used to access the Internet, this must be properly configured.

In the sections below, this configuration is described for Java and C#.

2.1.1 Proxy configuration in Java

The proxy can be globally configured in Java under System Properties. The following system properties can be easily set using the appropriate VM arguments:

System-Property	Description	Example for VM argument
proxyHost	The URL of the web proxy host	– DproxyHost="proxy.mycompany.com"
proxyPort	The port of the web proxies	– DproxyPort=8080
proxyUser	Username for authentication with the web proxy	– DproxyUser="myusername"
proxyPassword	Password for authentication with the web proxy	– DproxyPassword="mypassword"

These system proxy settings can also be set programmatically, as per the following example:

```
== Java Code: ==

// Example: Setting proxy "proxy.mycompany.com" on port 8080
System.setProperty("proxyHost", "proxy.mycompany.com");
System.setProperty("proxyPort", "8080");

// Optional: username and password for web proxy that requires authentication
System.setProperty("proxyUser", "your proxy username");
System.setProperty("proxyPassword", "your proxy password");
```

Further information about proxy configuration in Java can be found at http://java.sun.com/javase/6/docs/technotes/guides/net/proxies.html.

2.1.2 Proxy configuration in C#/.NET

In C#, the proxy can be configured directly on the WebRequest class.

```
== C# Code: ==

// Example: Setting proxy "proxy.mycompany.com" on port 8080
WebProxy wProxy = new WebProxy("http://proxy.mycompany.com:8080");
WebRequest.DefaultWebProxy = wProxy;

// Optional: username and password for web proxy that requires authentication
NetworkCredential credentials =
    new NetworkCredential("myProxyUsername", "myProxyPassword");
WebRequest.DefaultWebProxy.Credentials = credentials;
```

The constructor of the BarcodeWebService class can also be given a parameter as to whether the operating system default proxy (can be configured in Internet Explorer, is stored in the registry) is used:

```
== C# Code: ==

// Use Proxy (configured by OS or manually configured)
BarcodeWebService service = new BarcodeWebService(Language.de, true);

// Don't use any proxy (default)
BarcodeWebService service = new BarcodeWebService(Language.de, false);
```

The parameter is optional, the default is false. In the case of productive use of a proxy, we recommend configuring it manually. Using the operating system default proxy can result in longer response times.

3 Read operations

The read operations can be used to call up all the service codes and label layouts of the web service that are currently supported. All read operations display a **BarcodeErrorException** if the server responded to a "Barcode" web service query with error messages (in the chosen language).

Each of the following chapters contains a brief example of calling up each read operation.

3.1 Operation "Read service groups"

The **readServiceGroups** operation can be used to call up the service groups currently available, with their respective **serviceGroupID.**

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

try {
    List<ServiceGroup> groups = service.readServiceGroups();
    System.out.println("Available Service-Groups:");
    for (ServiceGroup group : groups) {
        System.out.print(group.getServiceGroupID() + ": ");
        System.out.println(group.getDescription());
    }
} catch (BarcodeErrorException e) {
    System.out.println("Error in call to readServiceGroups:");
    System.out.println(e.getErrorMessages());
}
```

```
== C# Code: ==

BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

try
{
    List<PostLogistics.WebServiceBarcode.Client.ServiceGroup> groups =
service.ReadServiceGroups();
    Console.WriteLine("Available Service-Groups:");
    foreach (PostLogistics.WebServiceBarcode.Client.ServiceGroup group in groups)
    {
        Console.WriteLine("{0}: {1}", group.ServiceGroupID,
            group.Description);
    }
}
catch (BarcodeErrorException ex)
{
    Console.WriteLine("Error in call to ReadServiceGroups():");
    Console.WriteLine(ex.GetErrorMessages());
}
```

3.2 Operation “Read basic services”

The **readBasicServices** operation can be used to call up the basic services currently available, with the respective service codes of each service group.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials(“username”, “passwort”);

// First read all service groups with ServiceGroupID
List<ServiceGroup> groups;
try {
    groups = service.readServiceGroups();
} catch (BarcodeErrorException e) {
    System.out.println(“Error in readServiceGroups:”);
    System.out.println(e.getErrorMessages());
    return;
}

// basic services per service group
System.out.println(“Available basic service codes:”);
System.out.println(“==============================”);
for (ServiceGroup group : groups) {
   System.out.println(“Basic Services for ‘” + group.getDescription() +
                                                        “’:”);
   try {
        List<BasicService> basicServices =
                        service.readBasicServices(group.getServiceGroupID());
        for (BasicService basicService : basicServices) {
            System.out.println(“  - ” + concatStrings(basicService.getPRZL())
                                    + “: ” + basicService.getDescription());
        }
    } catch (BarcodeErrorException e) {
        // usually means that this service group is not available yet
        System.out.println(e.getErrorMessages());
    }
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// First read Service-Groups with ServiceGroupID
List<PostLogistics.WebServiceBarcode.Client.ServiceGroup> groups;
try
{
    groups = service.ReadServiceGroups();
}
catch (BarcodeErrorException ex)
{
    Console.WriteLine("Error in ReadServiceGroups():");
    Console.WriteLine(ex.GetErrorMessages());
}
Console.WriteLine("Available basic service codes:");
Console.WriteLine("------------------------------");
foreach (PostLogistics.WebServiceBarcode.Client.ServiceGroup group in groups)
{
    // basic services per service group
    Console.WriteLine("Basic Services for {0}:", group.Description);
    try
    {
        List<PostLogistics.WebServiceBarcode.Client.BasicService> basicServices =
            service.ReadBasicServices(group.ServiceGroupID);
        foreach (PostLogistics.WebServiceBarcode.Client.BasicService basicService in
basicServices)
        {
            Console.WriteLine(" - {0}", basicService.PRZL);
        }
    }
    catch (BarcodeErrorException ex)
    {
        // usually means that this service group is not available yet
        Console.WriteLine(ex.GetErrorMessages());
    }
}
```

3.3 Operation “Read basic services of a franking licence”

With the operation **readAllowedServicesByFrankingLicense** information on availability of service groups and basic services for a specific franking licence can be requested.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials(“username”, “passwort”);

// Read the allowed services for a given franking license
String frankingLicense = “franking license”;
try {
    List<ReadAllowedServicesByFrankingLicenseResponse.ServiceGroups> allowedServices =
service.readAllowedServicesByFrankingLicense(frankingLicense, Language.DE);
    System.out.println(“Allowed services for the franking license 60022220:”);
    for (ReadAllowedServicesByFrankingLicenseResponse.ServiceGroups sgs :
allowedServices) {
        ServiceGroup sg = sgs.getServiceGroup();
        System.out.println(“ServiceGroupDescription: “ + sg.getDescription());
        System.out.println(“PostID: “ + sg.getServiceGroupID());
        System.out.println(“Associated BasicServices with Description and PRZLs”);
        for (BasicService bs : sgs.getBasicService()) {
            System.out.println(“BasicServiceDescription: “+bs.getDescription());
            System.out.println(“Associated PRZLs: “);
            int counter = 1;
            for (String s : bs.getPRZL()) {
                System.out.println(“PRZL “ + counter + “: “ + s);
                counter++;
            }
        }
    }
} catch (BarcodeErrorException e) {
    // this occurs if the basic service does not exist.
    System.out.println(“Errors: “);
    System.out.println(e.getErrorMessages());
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// Read allowedServices with a test franking license
List<AllowedService> allowedServices;
int listCounter = 1;
try
{
    allowedServices = service.ReadAllowedServicesByFrankingLicense(FrankingLicense);
    foreach (AllowedService allowedService in allowedServices)
    {
        PostLogistics.WebServiceBarcode.Client.ServiceGroup sg =
allowedService.ServiceGroup;
        Console.WriteLine("********** ServiceGroups Element Nr. " + listCounter +
" **********");
        Console.WriteLine("ServiceGroupDescription: " + sg.Description);
        Console.WriteLine("PostID: " + sg.ServiceGroupID);
        Console.WriteLine("********** List of BasicServices **********");
        foreach (PostLogistics.WebServiceBarcode.Client.BasicService bs in
allowedService.BasicServices)
        {
            Console.WriteLine("********** BasicService Element **********");
            Console.WriteLine("BasicServiceDescription: " + bs.Description);
            int counter = 1;
            if (bs.PRZL != null)
            {
                foreach (String przl in bs.PRZL)
                {
                    Console.WriteLine("PRZL " + counter + ": " + przl);
                    counter++;
                }
            }
        }
        Console.WriteLine("");
        Console.WriteLine("********** (END ServiceGroups Element Nr. " + listCounter +
") **********");
        Console.WriteLine("");
        listCounter++;
    }
}
catch (BarcodeErrorException ex)
{
    Console.WriteLine(Fehler in WebService Aufruf
ReadAllowedServicesByFrankingLicense():");
    Console.WriteLine(ex.GetErrorMessages());
}
```

3.4 Operation “Read additional services”

The **readAdditionalServices** operation can be used to call up the additional services currently available, with the respective service codes for each basic service.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials(“username”, “passwort”);

// Read available additional service codes for “Sperrgut Priority”.
try {
    List<AdditionalService> additionalServices = service.readAdditionalServices
(BasicServiceCode.PRI, BasicServiceCode.SP);
    System.out.println(“Available additonal services for PRI, SP:”);
    for (AdditionalService additionalService : additionalServices) {
        System.out.println(additionalService.getPRZL() + “: “ +
        additionalService.getDescription());
    }
} catch (BarcodeErrorException e) {
    // this occurs if the basic service does not exist
System.out.println(“Error in readAdditionalServices(): “ + e.getErrorMessages());
}
```

```
== C# Code: ==

BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = “username”;
service.Password = “password”;

// Read available additional service codes for “Sperrgut Priority”.
try
{
    List<AdditionalService> additionalServices =
        service.ReadAdditionalServices(BasicServiceCode.PRI,
        BasicServiceCode.SP);
    Console.WriteLine(“Available additional services for PRI, SP:”);
    foreach (AdditionalService additionalService in additionalServices)
    {
        Console.WriteLine(“{0} : {1}”, additionalService.PRZL,
           additionalService.Description);
    }
}
catch (BarcodeErrorException ex)
{
    // this occurs if the basic service does not exist.
    Console.WriteLine(“Error in ReadAdditionalServices():”);
    Console.WriteLine(ex.GetErrorMessages());
}
```

3.5 Operation “Read delivery instructions”

The **readDeliveryInstructions** operation can be used to call up the delivery instructions currently available, with the respective service codes of each basic service.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// Read available delivery instruction codes for basic service PRI, SP:
try {
    List<DeliveryInstructions> deliveryInstructions =
        service.readDeliveryInstructions(BasicServiceCode.PRI,
            BasicServiceCode.SP);
    System.out.println("Available delivery instructions for
        basic service PRI, SP:");
    for (DeliveryInstructions zaw : deliveryInstructions) {
        System.out.println(zaw.getPRZL() + ": "  + zaw.getDescription());
    }
} catch (BarcodeErrorException e) {
    // this occurs if the basic service does not exist.
    System.out.println("Errors: ");
    System.out.println(e.getErrorMessages());
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// First get all available service groups.
List<PostLogistics.WebServiceBarcode.Client.ServiceGroup> groups =
service.ReadServiceGroups();
// Get available label layouts for each service group and display them
Console.WriteLine ("Supported delivery instructions per basic service ...");
Console.WriteLine ("=====================================================");
foreach (PostLogistics.WebServiceBarcode.Client.ServiceGroup group in groups)
{
    Console.WriteLine("for ServiceGroup '" + group.Description + "':");
    try {
        List<PostLogistics.WebServiceBarcode.Client.BasicService> basicServices =
service.ReadBasicServices(group.ServiceGroupID);
        foreach (PostLogistics.WebServiceBarcode.Client.BasicService basicService in
basicServices)
        {
             Console.WriteLine("  - " + basicService.PRZL + ": ");
             String[] basicServiceCodes = basicService.PRZL;
             List<DeliveryInstruction> deliveryInstructions =
service.ReadDeliveryInstructions(basicServiceCodes);
            foreach (DeliveryInstruction deliveryInstruction in deliveryInstructions)
            {
                Console.WriteLine(deliveryInstruction.PRZL);
                Console.WriteLine(", ");
            }
            Console.WriteLine ("");
            }
        }
        catch (BarcodeErrorException ex)
        {
        // This usually means, that this service group is currently not supported at all.
        Console.WriteLine("Error in WebService call to ReadDeliveryInstructions:");
        Console.WriteLine(ex.GetErrorMessages());
    }
}
```

3.6 Operation "Read notations"

With the operation **readLabelLayouts** information on momentarily availability of notations for a valid combination PRZLs can be requested.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// First get all available service groups with serviceGroupID.
List<ServiceGroup> groups;
try {
    groups = service.readServiceGroups();
} catch (BarcodeErrorException e) {
    System.out.println("Error in readServiceGroups(): ");
    System.out.println(e.getErrorMessages());
    return;
}

// Get available label layouts for each service group and display them
System.out.println("Available Label Layouts:");
System.out.println("========================");
for (ServiceGroup group : groups) {
    System.out.println("LabelLayouts for '" + group.getDescription() + "':");
    List<BasicService> basicServices;
   try {
      basicServices = service.readBasicServices(group.getServiceGroupID());
   } catch (BarcodeErrorException e) {
        System.out.println("Error in readBasicServices(): ");
        System.out.println(e.getErrorMessages());
        return;
}
for (BasicService basicService : basicServices) {
   try {
       System.out.println("for BasicService " + basicService.getDescription());
       // read label layouts with a valid combination of PRZLs (here extracted from a
given BasicService)
       List<LabelLayoutResponse> layouts = service.readLabelLayouts
(basicService.getPRZL());
       for (LabelLayoutResponse layout : layouts) {
            System.out.print(" - " + layout.getLabelLayout() + ": ");
            System.out.print(" max. " + layout.getMaxServices() + " services ");
            System.out.print(" and " + layout.getMaxDeliveryInstructions()
             + " delivery instructions per label, ");
            if (layout.isFreeTextAllowed()) {
               System.out.println("freetext allowed.");
            } else {
                System.out.println("freetext not allowed.");
            }
        }
   } catch (BarcodeErrorException e) {
       // This usually means, that this service group is currently not supported at all.
       System.out.println(e.getErrorMessages());
   }
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// First get all available service groups with serviceGroupID.
List<PostLogistics.WebServicebarcode.Client.ServiceGroup> groups;
try
{
    groups = service.ReadServiceGroups();
}
catch (BarcodeErrorException ex)
{
    Console.WriteLine("Error in ReadServiceGroups():");
    Console.WriteLine(ex.GetErrorMessages());
}
// Get available label layouts for each service group and display them
Console.WriteLine("Available Label Layouts:");
Console.WriteLine("========================");
foreach (PostLogistics.WebServicebarcode.Client.ServiceGroup group in groups)
{
    Console.WriteLine(LabelLayouts for '{0}':", group.Description);
    List<PostLogistics.WebServiceBarcode.Client.BasicService> basicServices;
    try
    {
        basicServices = service.ReadBasicServices(group.ServiceGroupID);
        foreach (PostLogistics.WebServiceBarcode.Client.BasicService basicService in
basicServices)
        {
            try
            {
            Console.WriteLine("for BasicService " + basicService.Description);
            // read label layouts with a valid combination of PRZLs (here extracted from
a given BasicService)
            List<PostLogistics.WebServiceBarcode.Client.LabelLayout> layouts =
service.ReadLabelLayouts(basicService.PRZL);
            Foreach (PostLogistics.WebServiceBarcode.Client.LabelLayout layout in
layouts)
            {
                Console.WriteLine(" - " + layout.Layout + ": ");
                Console.WriteLine(" max. " + layout.MaxServices + " services ");
                Console.WriteLine(" and " + layout.MaxDeliveryInstructions + " delivery
instructions per label, ");
                if (layout.FreeTextAllowed)
                {
                    Console.WriteLine("freetext allowed.");
                }
                else
                {
```

```
                    Console.WriteLine("freetext not allowed.");
                }
            }
        }
        catch (BarcodeErrorException ex)
        {
            Console.WriteLine(ex.GetErrorMessages());
        }
    }
}
catch (BarcodeErrorException ex)
{
    Console.WriteLine("Error in ReadBasicServices(): ");
    Console.WriteLine(ex.GetErrorMessages());
}
```

4 Validation operation

The web service makes an operation available in order to validate a combination of service codes for a label request. The system checks that the service codes contained in the request (basic service, additional services and delivery instructions) can be combined. The system also checks whether the codes used are permitted in the chosen label format (A5, A6, A7 or FE). Only a certain number of service codes and delivery instructions are permitted, depending on the format.

The interface of the client API libraries uses the same BarcodeLabelRequest classes as the **generateLabel** methods as the input for validation. The advantage of this is that the requests generated can easily be checked in advance for the correct service code combinations. Naturally, the web service performs the same validation when the **generateLabel** methods are called up.

The service codes currently available are available in special constant classes as string constants: **BasicServiceCode, AdditionalServiceCode** and **DeliveryInstructionCode**. These classes only contain the codes that were available at the time the "Barcode" web service client API was developed. Occasionally, additional codes may be subsequently added and passed on as normal strings. The read operations must be used in order to receive all the current codes.

The following example demonstrates a check on a combination of basic service codes, additional service codes and delivery instruction codes.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// Set label layout to validate allowed service codes for
// (otherwise the default is used)
service.setLabelLayout(LabelFormat.A5);

// Label request with basic service code(s) (PRI = Priority)
BarcodeLabelRequest request = service.createLabelRequest();
request.addServiceCode(BasicServiceCode.PRI);

// additional service codes
request.addServiceCode(AdditionalServiceCode.FRA);
request.addServiceCode("SI");

// delivery instruction codes
request.addServiceCode(DeliveryInstructionCode.ZAW3213);
request.addServiceCode("ZAW3215");

// set the country required to validate the combination
request.setCountryIsoCode("CH");

// validation
ValidationResponse result = service.validateCombination(request);
if (!result.hasErrors()) {
    System.out.println("Validation was successful.");
} else {
    System.out.println("Validation has ERRORS: ");
    for (BarcodeError error : result.getErrors()) {
        System.out.println(error.getCode() + " - " + error.getMessage());
    }
}
if (result.hasWarnings()) {
    System.out.println("WARNINGS: ");
    for (BarcodeWarning warning : result.getWarnings()) {
        System.out.println(warning.getCode() + " - " + warning.getMessage());
    }
}
```

```
== C# Code: ==

BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// Set label layout to validate allowed service codes for
// (otherwise the default is used)
service.LabelLayout = LabelFormat.A5;

// Label request with basic service code(s) (PRI = Priority)
BarcodeLabelRequest request = service.CreateLabelRequest();
request.AddServiceCode(BasicServiceCode.PRI);

// additional service codes
request.AddServiceCode(AdditionalServiceCode.FRA);
request.AddServiceCode("SI");

// delivery instruction codes
request.AddServiceCode(DeliveryInstructionCode.ZAW3213);
request.AddServiceCode("ZAW3215");

// set the country required to validate the combination
request.AddCountryIsoCode("CH");

// validation
ValidationResponse result = service.ValidateCombination(request);
if (!result.HasErrors())
{
    Console.WriteLine("Validation was successful.");
}
else
{
    Console.WriteLine("Validation has ERRORS: ");
    foreach (ErrorsTypeError error in result.GetErrors())
    {
        Console.WriteLine(error.Code + " - " + error.Message);
    }
}
if (result.HasWarnings())
{
    Console.WriteLine("Validation has WARNINGS: ");
    foreach (WarningsTypeWarning warning in result.GetWarnings())
    {
        Console.WriteLine(warning.Code + " - " + warning.Message);
    }
}
```

This example validates without errors.

If, however, in the examples "ZAW3215" is replaced with "ZAW3222" the web service request replies with a corresponding validation error message.

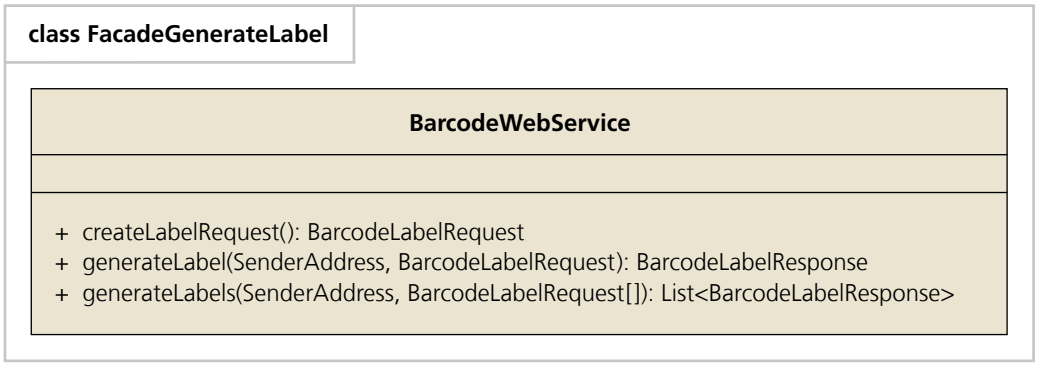
There is another **validateCombinations** method that can be used to validate several **BarcodeLabelRequests** at once in a single Query web service.

5 Label/Barcode Generation

5.1 Label/Barcode generation

The operation to generate address labels expects two parameters to be input:
– **SenderAddress:** self-explanatory
– **LabelRequest:** the required parameters, including recipient address for an address label to be generated

There are two possible methods for calling up this operation: one for generating a single address label and one for generating several address labels with barcode in one request.



The factory method **createLabelRequest** should be used to create a **BarcodeLabelRequest** instance. This method ensures that the requests are initialized with a sequential **itemID.** If necessary, the **itemID** can also be set independently. However, in this case the uniqueness of the **itemID** within a **generateLabel** request must be ensured.

The generated labels are returned as BarcodeLabelResponse objects. A BarcodeLabelResponse can in turn contain error messages (in the chosen language), if the corresponding address label could not be generated with the desired attributes. Otherwise, the generated address label is contained in the corresponding file format in the **LabelBinaryData** property. These binary data can be saved as, for example, an image file.

The following example demonstrates how a simple, individual address label can be generated and the response read.

== Java Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// Franking licence (mandatory!):
// this has to be a valid license for your barcode web service user account
service.setFrankingLicense("12345678");


// enable/disable print preview
service.setPrintPreview(false);

// sender address
SenderAddress sender = new SenderAddress();
sender.setName1("Meier AG");
sender.setStreet("Viktoriaplatz 10");
sender.setZIP("8048");
sender.setCity("Zürich");

// BarcodeLabelRequest with recipient address
BarcodeLabelRequest labelRequest = service.createLabelRequest();
RecipientAddress address = labelRequest.getAddress();
address.setFirstName("Melanie");
address.setName1("Steiner");
address.setName2 ("Müller AG");
address.setName3 ("Abteilung Marketing");
address.setAddressSuffix ("Gebäude 6-Ost");
address.setStreet("Viktoriastrasse");
address.setHouseNo("21a");
address.setZIP("3030");
address.setCity("Bern");

// Add notification services with Builder-pattern and immediate validation
NotificationService.Builder notificationServiceBuilder =
   new NotificationService.Builder(
      NotificationService.NotificationType.CODE_1, Language.DE);
NotificationService notificationService =
   notificationServiceBuilder.email("test@test.ch")
                             .freeText1("Test 1")
                             .freeText2("Test 2").build();
labelRequest.addNotificationService(notificationService);

// set service-code
labelRequest.addServiceCode(BasicServiceCode.ECO);

// Service call generateLabel:
BarcodeLabelResponse response = service.generateLabel(sender, labelRequest);

// display response information
try {

    // read binary label image data
    // (throws an Exception if error in the response)
    byte[] binaryData = response.getLabelBinaryData();
```

```
      // dsplay barcode information
    System.out.println("Barcode successfully generated: " +
          response.getIdentCode());
    System.out.print(" Binary-Data: " + response.getImageFileType());
    System.out.print(", " + response.getImageResolution() + " DPI, ");
    System.out.print(response.isColorPrintRequested() ? "color, " : "black, ");
    System.out.println(binaryData.length + " bytes");

} catch (BarcodeErrorException e) {

// label not generated, display errors:
// (error messages are in the choosen language)
System.out.println("Barcode was not generated due to errors: ");
    System.out.println(e.getErrorMessages());
}

// Also display warning messages
if (response.hasWarnings()) {
    System.out.print(", WARNINGS:");
    System.out.println(concatMessages(response.getWarnings()));
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";

// Franking licence (mandatory!):
// this has to be a valid license for your barcode web service user account
service.FrankingLicense = "12345678";
service.PrintPreview = false;

BarcodeLabelRequest labelRequest = service.CreateLabelRequest();

// Sender address
SenderAddress sender = new SenderAddress();
sender.Name1 = "Meier AG";
sender.Street = "Viktoriaplatz 10";
sender.Zip = "8048";
sender.City = "Zürich";

// BarcodeLabelRequest with recipient address
RecipientAddress address = labelRequest.Address;
address.Title = "Frau";
address.FirstName = "Melanie";
address.Name1 = "Steiner";
address.Name2 = "Müller AG";
address.Name3 = "Abteilung Marketing";
address.AddressSuffix = "Gebäude 6-Ost";
address.Street = "Viktoriastrasse";
address.HouseNo = "21a";
address.Zip = "3030";
address.City = "Bern";
```

```
// Add notification services
NotificationService.Builder notificationServiceBuilder = new NotificationService.Builder(
    NotificationService.NotificationType.CODE_1, Language.de);
NotificationService notificationService = notificationServiceBuilder.email
("test@test.ch").freeText1("Test 1").
    freeText2("Test 2").build();
labelRequest.AddNotificationService(notificationService);

// set service-code
labelRequest.AddServiceCode(BasicServiceCode.ECO);

// Service call generateLabel:
BarcodeLabelResponse response = service.GenerateLabel(sender, labelRequest);

// display response information
try
{
    // Read binary label image data
    // (throws an Exception if error in the response)
    byte[] binaryData = response.LabelBinaryData;

    // display barcode information
    Console.Write("Barcode successfully generated: " + response.IdentCode);
    Console.Write(" Binary-Data: " + response.ImageFileType);
    Console.Write(", " + response.ImageResolution + " DPI, ");
    Console.Write(response.ColorPrintRequested ? "color, " : "black, ");
    Console.WriteLine(binaryData.Length + " bytes");
}
catch (BarcodeErrorException ex)
{
    // label not generated, display errors:
     // (error messages are in the choosen language)
    Console.WriteLine("Barcode was not generated due to errors: ");
    Console.WriteLine(ex.GetErrorMessages());
}

// Also display warning messages
if (response.HasWarnings())
{
    Console.Write(", WARNINGS:");
    Console.WriteLine(response.GetWarnings());
}
```

5.2 Individual barcode generation

With the operation **generateSingleBarcodes** the barcodes of an address label can be accessed individually. This operation is very similar to the operation **generateLabel** and is different only in that in the initiation certain parameters (PrintAdresses, Printpreview, labelLayout) cannot be set. The reply likewise is only slightly different. Instead of an individual label, a list of barcodes is generated.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// Franking licence (mandatory!):
// this has to be a valid license for your barcode web service user account
service.setFrankingLicense("12345678");

// enable/disable print preview
service.setPrintPreview(false);
// sender address
SenderAddress sender = new SenderAddress();
sender.setName1("Meier AG");
sender.setStreet("Viktoriaplatz 10");
sender.setZIP("8048");
sender.setCity("Zürich");
sender.setDomicilePostOffice("8048 Zürich");

// BarcodeLabelRequest with recipient address
SingleBarcodesRequest labelRequest = service.createSingleBarcodesRequest();
RecipientAddress address = labelRequest.getAddress();
address.setTitle("Frau");
address.setFirstName("Melanie");
address.setName1("Steiner");
address.setStreet("Viktoriastrasse");
address.setHouseNo("21a");
address.setZIP("3030");
address.setCity("Bern");

// set service-code
labelRequest.addServiceCode("RINL");
labelRequest.addServiceCode("eAR");
labelRequest.addServiceCode("ZAW2511");

// Service call generateLabel:
SingleBarcodesResponse response = service.generateSingleBarcodes(sender, labelRequest);

// display response information
try {
    // Read binary barcodes data
    // (throws an Exception if error in the response)
    List<byte[]> binaryData = response.getBarcodesBinaryData();
```

```
    // Informationen zum Bar-Code ausgeben
    System.out.println("Barcode successfully generated: " + response.getIdentCode());
    System.out.print(" Binary-Data: " + response.getImageFileType());
    System.out.print(", " + response.getImageResolution() + " DPI, ");
    System.out.print(response.isColorPrintRequested() ? "color, " : "black, ");
    System.out.println("Number of generated single barcodes: " + binaryData.size());
    for (int i = 0; i < binaryData.size(); i++) {
        System.out.println("Barcode " + i + "  " + binaryData.get(i).length + " bytes");
    }
} catch (BarcodeErrorException e) {
    // label not generated, display errors:
    // (error messages are in the chosen language)
    System.out.println("Barcode was not generated due to errors: ");
    System.out.println(e.getErrorMessages());
}

// also display warning messages
if (response.hasWarnings()) {
    System.out.println(" with WARNINGS:");
    for (BarcodeWarning warning : response.getWarnings()) {
        System.out.println("  " + warning.getCode() + " - " + warning.getMessage() + ", ");
}
```

== C# Code: ==

```
BarcodeWebService service = new BarcodeWebService(Language.de);
service.UserName = "username";
service.Password = "password";
service.FrankingLicense = FrankingLicense;
service.PrintPreview = false;

// Sender address
SenderAddress sender = new SenderAddress();
sender.Name1 = "Meier AG";
sender.Street = "Viktoriaplatz 10";
sender.Zip = 8048L;
sender.City = "Zürich";
sender.DomicilePostOffice = "Zürich";

// BarcodeLabelRequest inklusive recipient address
SingleBarcodesRequest request = service.CreateSingleBarcodesRequest();
RecipientAddress address = request.Address;
address.setTitle("Frau");
address.setFirstName("Melanie");
address.setName1("Steiner");
address.setStreet("Viktoriastrasse");
address.setHouseNo("21a");
address.setZIP("3030");
address.setCity("Bern");

request.AddServiceCode("RINL");
request.AddServiceCode("eAR");
request.AddServiceCode("ZAW2511");

// call the service to create the single barcodes
SingleBarcodesResponse response = service.GenerateSingleBarcodes(sender, request);

try
{
    // Read binary label image data
    // (throws an Exception if error in the response)
    byte[] binaryData = response.LabelBinaryData;
}
catch (BarcodeErrorException ex)
{
    // label not generated, display errors:
    // (error messages are in the choosen language)
    Console.WriteLine("Barcode was not generated due to errors: ");
    Console.WriteLine(ex.GetErrorMessages());
}

// Also display warning messages
if (response.HasWarnings())
{
    Console.Write(", WARNINGS:");
    Console.WriteLine(response.GetWarnings());
}
```

5.3 Operation “Generate Barcode”

With the operation **generateBarcode** a customer can generate and print predefined barcodes.

```
== Java Code: ==

BarcodeWebService service = new BarcodeWebService(Language.DE);
service.setLoginCredentials("username", "passwort");

// Generate an LSO-1 barcode

// create first the request object for the generateBarcode request
IndividualBarcodeRequest request = service.createIndividualBarcodeRequest();
// choose which barcode to generate (here LSO-1)
BarcodeDefinition barcodeDefinition = new BarcodeDefinition();
barcodeDefinition.setImageFileType(ImageFileType.PDF);
barcodeDefinition.setImageResolution(300);
barcodeDefinition.setBarcodeType(BarcodeType.LSO_1);
request.setBarcodeDefinition(barcodeDefinition);

// service call generateBarcode:
IndividualBarcodeResponse response = service.generateBarcode(request);

try {
    // read binary barcode data
    // (throws an Exception if there are errors in the response)
    byte[] binaryData = response.getBarcodeData();

    // display barcode information
    System.out.println("Barcode successfully generated!");
    System.out.print(" Binary-Data: " + response.getBarcodeData());
    System.out.print(", " + response.getBarcodeDefinition().getImageResolution() +
" DPI, ");
    System.out.print(response.isColorPrintRequired() ? "color, " : "black, ");
    System.out.println(binaryData.length + " bytes");

} catch (BarcodeErrorException e) {
    // barcode not generated, display errors:
    // (error messages are in the chosen language)
    System.out.println("Barcode was not generated due to errors: ");
    System.out.println(e.getErrorMessages());
}
```

```
== C# Code: ==

BarcodeWebService service = new BarcodeWebService(Lang);

service.UserName = UserName;
service.Password = Password;
service.EndpointAddress = new EndpointAddress(Url);

IndividualBarcodeRequest request = service.CreateIndividualBarcodeRequest();
BarcodeDefinition barcodeDefinition = new BarcodeDefinition();
barcodeDefinition.BarcodeType = BarcodeType.LSO_1;
barcodeDefinition.ImageFileType = ImageFileType.JPG;
barcodeDefinition.ImageResolution = 300;
request.Definition = barcodeDefinition;

IndividualBarcodeResponse response = service.GenerateBarcode(request);
try
{
    // Read binary label image data
    // (throws an Exception if error in the response)
    byte[] binaryData = response.BarcodeData;
}
catch (BarcodeErrorException ex)
{
    // barcode not generated, display errors:
    // (error messages are in the choosen language)
    Console.WriteLine("Barcode was not generated due to errors: ");
    Console.WriteLine(ex.GetErrorMessages());
}
if (response.HasWarnings())
{
    Console.Write(", WARNINGS:");
    Console.WriteLine(response.GetWarnings());
}
```

5.4 Examples of electronic COD

For electronic COD, various fields need to be recorded besides the electronic COD ServiceCode. These differ in the case of electronic COD with ISR and electronic COD with IBAN.

N.B.: Electronic COD with IBAN may not yet currently be used.

Electronic COD with ISR:

```
== Java Code: ==

labelRequest.addServiceCode(AdditionalServiceCode.BLN);

labelRequest.setCashOnDeliveryAmount(10f);
labelRequest.setCashOnDeliveryRefNrESR(„965993000000000000001237460“);
labelRequest.setCashOnDeliveryCustomerNrESR(„010003757“);
labelRequest.setCashOnDeliveryCustomerEmail(„hans.muster@mail.ch“);
labelRequest.setCashOnDeliveryCustomerMobile(„0791234567“);
```

```
== C# Code: ==

labelRequest.AddServiceCode(AdditionalServiceCode.BLN);

labelRequest.CashOnDeliveryAmount = 10;
labelRequest.CashOnDeliveryRefNrESR = „965993000000000000001237460“;
labelRequest.CashOnDeliveryCustomerNrESR = „010003757“;
labelRequest.CashOnDeliveryCustomerEmail = „hans.muster@mail.ch“;
labelRequest.CashOnDeliveryCustomerMobile = „0791234567“;
```

Electronic COD with IBAN:

```
== Java Code: ==

labelRequest.addServiceCode(AdditionalServiceCode.BLN);

labelRequest.setCashOnDeliveryAmount(10f);
labelRequest.setCashOnDeliveryIbanNumber(„CH10002300A1023502601“);
labelRequest.setCashOnDeliveryIbanName(„Hans Muster“);
labelRequest.setCashOnDeliveryIbanStreet(„Musterstrasse 11“);
labelRequest.setCashOnDeliveryIbanZip(„3011“);
labelRequest.setCashOnDeliveryIbanCity(„Bern“);
labelRequest.setCashOnDeliveryCustomerEmail(„hans.muster@mail.ch“);
labelRequest.setCashOnDeliveryCustomerMobile(„0791234567“);
```

```
== C# Code: ==

labelRequest.AddServiceCode(AdditionalServiceCode.BLN);

labelRequest.CashOnDeliveryAmount = 10;
labelRequest.CashOnDeliveryIbanNumber = „CH10002300A1023502601“;
labelRequest.CashOnDeliveryIbanName = „Hans Muster“;
labelRequest.CashOnDeliveryIbanStreet = „Musterstrasse 11“;
labelRequest.CashOnDeliveryIbanZip = „3011“;
labelRequest.CashOnDeliveryIbanCity = „Bern“;
labelRequest.CashOnDeliveryCustomerEmail = „hans.muster@mail.ch“;
labelRequest.CashOnDeliveryCustomerMobile = „0791234567“;
```

4080 en 07.2018

Post CH Ltd
Support Webservices
Wankdorfallee 4
P.O. Box
3030 Berne
Switzerland

webservice@post.ch
www.swisspost.ch/webservice

SWISS POST