enfocus SWITCH 12

Web Services Documentation

smart automation

enfocus

Contents

1. Copyrights

2. Switch Web Services

The Switch Web Services is a SOAP based development tool designed to create custom Switch client interfaces within existing applications or web-based interfaces. This can be useful for external users when they need access to jobs or workflows, or in cases where access to jobs needs to be built into existing application interfaces.

For more information on the Switch Web Services, please contact your local Enfocus reseller or contact Enfocus directly.

3. Introduction

3.1 Short summary

This section describes the Enfocus Web Service Module (SDK). The aim is to provide both a general introduction and a complete reference. After reading this document, the user should be able to use the Enfocus Web Service Module ("the SDK") to communicate with a Switch server.

3.2 Software requirement and licensing

To work with the SDK, you will need:

- Enfocus Switch Core Engine ("Switch Server"). A copy can be requested via the Enfocus website.
- The Enfocus Switch Web Services Module. This module does not require a separate installer and becomes available after installing Switch Core Engine.
- Either a trial or a permanent license for both Switch Core Engine and for Switch Web Services Module. The license for Switch Web Services also activates the Submit and Checkpoint flow elements.
- To run the Java sample: Java Platform, Standard Edition 6 Runtime Environment ("JRE").
- To build the Java sample: Java Platform, Standard Edition 6 Development Kit ("JDK").
- To run the PHP sample: a PHP 5 enabled web server.

3.3 Prerequisite knowledge

The letters SDK stand for Software Development Kit – meaning the SDK is used to create your own software, which communicates with a Switch Server. As such, the reader should be familiar with what is required to develop software. Specifically, it is assumed the reader is acquainted with the following concepts and technologies:

- Switch: the reader should have a basic understanding of Switch, and understand the concepts of flows, submit- and checkpoints, and jobs. Information about this can be found in the Switch documentation.
- SOAP: the SDK uses SOAP to communicate with the Switch Server. The reader should understand the basics of SOAP.
- WSDL: the Web Services Description Language is used to describe the SOAP operations.
- DIME: to send and receive jobs to and from the Switch Server, the Direct Internet Message Encapsulation protocol is used.
- SSL and HTTPS: Switch uses SOAP over HTTPS. The reader should have a basic understanding of the HTTPS protocol.

- Java and PHP: to understand the samples, the reader should have at least some understanding of the Java and/or PHP languages.
- Web servers: to run the PHP sample, a PHP 5 enabled web server is required. This document assumes that the reader knows how to set this up.

If a language is used other than Java or PHP, it is assumed the reader knows how to establish HTTPS connections and send SOAP messages using the chosen language.

4. Using the SDK

In this chapter we describe the structure of the Web Service SDK and how to use it.

4.1 Purpose of the SDK

The purpose of the Web Service SDK is to give developers a way to interact directly with a Switch Server. Possible use cases include:

- Providing a specialized, in-house substitute for SwitchClient.
- Creating a website where users can submit and manage jobs remotely.
- Creating a custom Web to Print web site where job information and files are directly submitted into Switch for production.

4.2 SDK contents

The Web Service SDK is a zip archive containing the two subfolders `Documentation` and `Samples`.

The `Documentation` folder contains this document, as well as the WSDL file (butterfly.wsdl) describing the SOAP operations, and the cacert.pem certificates file required for the SSL connection.

The `Samples` folder contains the Java and PHP samples. The Java sample is provided both as source and as a precompiled jar file. The PHP sample, since PHP is an interpreted language, contains only the source files.

Note: The WSDL file used in the samples may be outdated; the latest version of the WSDL file is located in the Documentation folder as described above.

The SDK does not include the Switch installer or the software required to build and run the samples. Switch installers can be downloaded from the Enfocus website. A trial key is included with the Switch installer.

The JRE and JDK required for the Java sample can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Most web servers have a PHP module available; check the documentation of your web server on how to get and install this module.

4.3 SDK licensing

Web Service SDK licensing is separate from Switch Server licensing. To be able to use the SDK, you will need to activate a special Web Service SDK license key, available from Enfocus sales

(info@enfocus.com). This key must be activated through the Switch Server UI: go to **Help** > **Manage licenses...** . Click **Activate New** button in the **Manage licenses** dialog that appears and follow the instructions.

For convenience, the Web Service SDK can be used without restriction on a Switch Server that is running in trial mode.

4.4 Using the SOAP API

In this section we discuss some typical use cases for the Web Service SDK. We do not discuss every SOAP call in detail. For more information about the calls and about the calls that are not used here, see this Reference chapter.

Establishing a connection

Note: Because the procedure to establish a HTTPS connection and send SOAP messages is completely dependent on the chosen technology, it is not explained in detail here. The reader should already know how to do this, or should refer to the documentation of their chosen platform. The samples show how to send SOAP messages over HTTPS in Java with Apache Axis, and PHP 5.

To connect to the Switch Server, we use SOAP over SSL. First, an SSL connection must be made. The `cacert.pem` file, located in the `Documentation > ssl subfolder` of the SDK, contains the root certificate for the SSL connection. The endpoint of the connection is the URL of the Switch Server, e.g. https://localhost:51008 for a Switch Server running on the local machine, using the default port. The port can be changed in the Switch Preferences UI.

Once the connection has been set up, we can test whether we can actually connect to the Switch Server with the CheckConnect SOAP operation:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
 <butterfly:CheckConnect/>
</soap:Body>
```

Note: For brevity, we will omit the HTTP header and SOAP envelope in the examples.

This call returns a reserved integer if the connection succeeded, e.g.:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
 <butterfly:CheckConnectResponse>
  <param-1>1</param-1>
 </butterfly:CheckConnectResponse>
</soap:Body>
```

Passing the user credentials

Once we know that a connection can be established, all following calls must supply user credentials. Switch uses HTTP basic authentication for this.

Note: Again, the procedure to set up basic authentication and set the username and password is very system-specific, so we do not go into detail here. Refer to the samples to see how this is done in Java/Axis and PHP.

The username must be the name of a user as defined in the Switch UI, followed by a semicolon and the two-letter code 'EN'. For example, when using the pre-defined "Administrator" user (which has an empty password), the username will be "Administrator;EN".

To check the username and password, we use the CheckUserNameAndPass call:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:CheckUserNameAndPass/>
</soap:Body>
```

This call will fail if the username or password are incorrect, and returns a reserved integer if the credentials are OK, e.g.:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:CheckUserNameAndPassResponse>
    <param-3>2</param-3>
  </butterfly:CheckUserNameAndPassResponse>
</soap:Body>
```

Requesting information

In a typical SDK usage, the first order of business after connecting to the Switch Server is getting information, either to present to the user or to use in later calls (or both). A lot of different information can be retrieved, e.g. about flows, jobs, or the Switch Server itself.

As an example, let's look at how to retrieve all the submit points present in our Switch Server. For this, we have the GetSubmitPoints SOAP method. GetSubmitPoints needs one parameter: the identifier (ID) of the flow for which we want to get the submit points. The flow IDs can in turn be retrieved using the GetFlowList SOAP method, which takes no parameters:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly: GetFlowList/>
</soap:Body>
```

This returns the IDs of all the flows (both active and inactive) on the Switch Server, e.g.:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:GetFlowListResponse>
    <flowList>10</flowList>
     <flowList>14</flowList>
  </butterfly:GetFlowListResponse>
</soap:Body>
```

Note: Flow IDs, submit point IDs, and checkpoint IDs can change when the Switch Server is restarted.

Now we can use these flow IDs to get the submit points for each flow. The GetSubmitPoints method takes one parameter: the flow ID.

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:GetSubmitPoints>
    <flowId>10</flowId>
      </butterfly:GetSubmitPoints>
</soap:Body>
```

The response is a list with 0 or more submit point IDs:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:GetSubmitPointsResponse>
    <submitPoints>1</submitPoints>
     <submitPoints>8</submitPoints>
      </butterfly:GetSubmitPointsResponse>
</soap:Body>
```

Note: Submit point IDs and Checkpoint IDs may have duplicates over different flows. Thus, to uniquely identify a Submit point or Checkpoint, the flow ID is always required.

The same scheme can now be used, for example, to get the names of the submit points to present to the user. Or the submit point IDs could be used to submit a job to the Switch Server.

Working with jobs

Jobs are a central concept in Switch. Using the Web Service SDK, we can submit, download, replace, lock and unlock, and push (i.e. move to a checkpoint's connection(s)) jobs.

To send jobs through SOAP, Switch uses DIME, and all attachments must be stored in a zip archive.

The SubmitFile10 SOAP method takes a SubmitEntry and the file to submit. In the SubmitEntry, we specify the ID of the flow and submit point, as well as the job name and job origin:

```
  <soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
    <butterfly:SubmitFile10>
       <inEntry href="#id0"/>
       <inSubmitData href="CE0B4222945833ED125480C7C1D7F25D"
 xsi:type="xsd:base64Binary"/>
    </butterfly:SubmitFile10>
    <multiRef id="id0" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xsi:type="butterfly:SubmitEntry" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/">
       <flowId>7</flowId>
       <submitPointId>1</submitPointId>
       <jobName>info.txt</jobName>
       <jobOrigin>/info.txt</jobOrigin>
       <metadataXmlStr/>
    </multiRef>
  </soap:Body>
```

Where CE0B4222945833ED125480C7C1D7F25D points to a DIME attachment of a zip file.

The response is a single, reserved string:

```
  <soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
    <butterfly:SubmitFile10Response>
       <outJobId/>
    </butterfly:SubmitFile10Response>
  </soap:Body>
```

Downloading or replacing a job is similar.

To push a job to a checkpoint's connections, the PushCheckpointJob method is used. This method requires the flow and checkpoint ID, the job ID, and the connection(s) to which to push the job. To get the connection IDs, we use the GetCheckpointJobInfo call:

```
  <soapenv:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
    <butterfly:GetCheckpointJobInfo>
       <flowId>7</flowId>
       <checkPointId>4</checkPointId>
       <jobId>0006J</jobId>
    </butterfly:GetCheckpointJobInfo>
  </soapenv:Body>
```

The resulting **JobStatus** element contains the possible outgoing connections, e.g.:

```
  <soapenv:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
   <butterfly:GetCheckpointJobInfoResponse>
       <jobInfo>
         <jobAllowMultipleOutputs>false</jobAllowMultipleOutputs>
         <jobEnableReportViewing>false</jobEnableReportViewing>
         <jobAllowReplacingJob>false</jobAllowReplacingJob>
         <jobName>info.txt</jobName>
```

```
        <jobReportName/>
        <fullUserName>bens</fullUserName>
        <jobSize>52</jobSize>
        <jobSubmitTime>29.06.2011 15:57:18.967</jobSubmitTime>
        <jobWaitingTime>30.06.2011 17:29:14.456</jobWaitingTime>
        <connections>
          <connectionId>7</connectionId>
          <connectionName>Folder 2</connectionName>
        </connections>
        <jobRequiresMetadata>false</jobRequiresMetadata>
      </jobInfo>
  </butterfly:GetCheckpointJobInfoResponse>
</soap:Body>
```

Now we can use these connection IDs to push the job:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:PushCheckpointJob>
     <flowId>7</flowId>
     <checkPointId>4</checkPointId>
     <jobId>0006J</jobId>
     <jobInfo href="#id0"/>
  </butterfly:PushCheckpointJob>
      <multiRef id="id0" soapenc:root="0"
     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
       <metadataXmlStr/>
       <connectionIds>7</connectionIds>
       <jobName>info.txt</jobName>
      </multiRef>
</soap:Body>
```

The response is a single, reserved integer:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:PushCheckpointJobResponse>
     <param-5>0</param-5>
  </butterfly:PushCheckpointJobResponse>
</soap:Body>
```

Working with metadata

Often, while submitting a job or pushing it to a checkpoint's connections, Switch will require metadata to be entered. There are basically two types of metadata: metadata that is to be displayed to the user (read-only metadata), and metadata that the user has to provide. The first is represented with MetadataToDisplayClass, the second with MetadataToFillClass. For example, when getting information about a job in a checkpoint (using the GetCheckpointJobInfo method), we may get both display metadata and required metadata:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:GetCheckpointJobInfoResponse>
     <jobInfo>
       ...
       <jobMetadata>
         <mtdLabel>Metadata Field 1</mtdLabel>
         <mtdDescription/>
         <mtdDataType>string</mtdDataType>
         <mtdValue>hello</mtdValue>
       </jobMetadata>
       <jobRequiresMetadata>true</jobRequiresMetadata>
       <jobRequiredMetadata>
         <mtfLabel>Metadata Field 2</mtfLabel>
         <mtfDescription/>
         <mtfDataType>string</mtfDataType>
         <mtfDataFormat/>
         <mtfValueIsRequired>false</mtfValueIsRequired>
         <mtfRememberLastValue>false</mtfRememberLastValue>
         <mtfValue/>
       </jobRequiredMetadata>
     </jobInfo>
  </butterfly:GetCheckpointJobInfoResponse>
</soap:Body>
```

Note: For brevity, part of the jobInfo element has been omitted.

Here we can see that there is a display metadata field "Metadata Field 1" with value "hello", and a second field "Metadata Field 2", for which we should provide the value. When pushing the job using PushCheckpointJob, we provide the value for this field:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:PushCheckpointJob>
     <flowId>7</flowId>
     <checkPointId>4</checkPointId>
     <jobId>0006M</jobId>
     <jobInfo href="#id0"/>
      </butterfly:PushCheckpointJob>
      <multiRef id="id0" soapenc:root="0"
     soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <jobRequiredMetadata href="#id1"/>
      <metadataXmlStr/>
      <connectionIds>7</connectionIds>
      <jobName>info.txt</jobName>
       </multiRef>
      <multiRef id="id1" soapenc:root="0"
     soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <mtfLabel>Metadata Field 2</mtfLabel>
      <mtfDescription/>
      <mtfDataType>string</mtfDataType>
      <mtfDataFormat/>
      <mtfValueIsRequired>false</mtfValueIsRequired>
      <mtfRememberLastValue>false</mtfRememberLastValue>
      <mtfValue>world</mtfValue>
     </multiRef>
</soap:Body>
```

Closing the connection

Although SOAP is inherently connectionless, Switch keeps a list of "open" connections. Before exiting, your product should inform the Switch Server that the connection information may be deleted:

```
<soap:Body xmlns:butterfly="http://tempuri.org/butterfly.xsd">
  <butterfly:CloseConnection/>
</soap:Body>
```

5. The samples

5.1 The Java sample

The Java sample uses Java SE 6 and Apache Axis. In the `Samples` > `java` subfolder of the SDK, you will find the sources (subfolder src) and a ready-to-run jar (subfolder build).

Running the sample

To run the Java sample, simply open a command window and navigate to the build folder. Then type:

`java -jar SwitchClientSDKSample.jar help` and press **Enter** key.

The Java sample will print a short usage message.

Note: To run the Java sample, it is important that the cacert.jks file is located in the current working directory. You must have the Java 6 JRE installed to be able to run the sample.

The Java sample can be used to get a list of Submit points, Checkpoints or jobs in a specific Checkpoint and can upload a job, send a Checkpoint job further through the flow and replace Checkpoint job.

For example, to submit a file to Submit point "16.3", the command could be:

```
java -jar SwitchClientSDKSample.jar --server=localhost
--user=Administrator --submitpoint=16.3 submit /Users/user/test.pdf
```

Note: Flow IDs, submit point IDs, and checkpoint IDs can change when the Switch Server is restarted.

Examining the sample code

The Java sample source code contains two packages:
`com.enfocus.switchclientsdk.sample` and
`com.enfocus.switchclientsdk.soapproxy`.

The soapproxy package contains code that was generated using the Axis WSDL2Java tool. Using the butterfly.wsdl file as input, this tool generates any required Java classes to be able to communicate with the Switch Server over SOAP.

The sample package contains a single Main class, which contains the code of the Java sample. What follows is a list of the most important members of the Main class.

Member	Description
Enfocus_ID	A static String containing the Enfocus ID, which will be appended to the username when authenticating.

Member	Description
main()	The program entry point. This function will parse the command-line parameters, create a new object of type Main, and start the application.
Main()	The class constructor, which will check that the command-line parameters (which were parsed in the main() function) are valid.
run()	The main application code: this function will establish and check the connection, check the username and password, and, after calling one of the functions below, close the connection. This uses the SOAP calls CheckConnect, CheckUserNameAndPass, and CloseConnection.
printSubmitPoints()	Retrieves the list of submit points. This uses the SOAP calls GetFlowList, GetSubmitPoints, and GetSubmitPointName. The result, a list of submit points per flow, is formatted and printed to the console.
submitFile()	Submits a job to the Switch Server. First, GetSubmitPointName is used to find out which metadata is required. After filling the metadata, the job is zipped and sent to the Switch Server using SubmitFile10.
printCheckpoints()	Retrieves the list of checkpoints. This uses the SOAP calls GetFlowList, GetFlowName, GetCheckPoints, and GetCheckPointName. The result, a list of checkpoints per flow, is formatted and printed to the console.
printJobs()	Retrieves the list of jobs present in a specific checkpoint. This uses GetCheckpointJobs10 to get the list of jobs, and GetCheckpointJobInfo to get information about each job (such as the connections to which it could be sent).
doJobAction()	Pushes a checkpoint job to a specific connection. GetCheckpointJobInfo is used to get the list of possible connections, which is presented to the user. After getting the required metadata, the job is sent to the selected connection using PushCheckpointJob.
replaceJob()	Replaces a checkpoint job with the file chosen by the user and pushes the result to a specified connection. GetCheckpointJobInfo is used to get the list of possible connections and available

Member	Description
	metadata. After getting the required metadata, the new file replaces the original job and is sent to the selected connection using ReplaceFile.

Building the sample

In the following, it is assumed that Java SE 6 JDK is installed and available on the PATH.

To build the Java sample, open a command window and navigate to the `src` subfolder of the Java sample. The sample uses Apache Axis, which means the Axis class files must be available on the classpath.

First, we build the soapproxy package:

```
javac -classpath .:../build/lib/axis.jar:../build/lib/jaxrpc.jar
 com/enfocus/switchclientsdk/soapproxy/*.java
```

And then we can build the sample code:

```
javac -classpath .:../build/lib/axis.jar:../build/lib/jaxrpc.jar
 com/enfocus/switchclientsdk/sample/Main.java
```

On Windows it is necessary to use ';' instead of ':' in command prompt:

```
javac -classpath .;../build/lib/axis.jar;../build/lib/jaxrpc.jar
 com/enfocus/switchclientsdk/soapproxy/*.java
javac -classpath .;../build/lib/axis.jar;../build/lib/jaxrpc.jar
 com/enfocus/switchclientsdk/sample/Main.java
```

Now pack just built '.class' files and manifest.txt file, located in src directory, into jar archive:

```
jar cvfm SwitchClientSDK.jar manifest.txt
 com\enfocus\switchclientsdk\sample\*.class
 com\enfocus\switchclientsdk\soapproxy\*.class
```

Now the sample is ready to run, however, to create a connection with the Switch Server, we need SSL. For this, a Java keystore (jks) file is needed.

This file can be generated from the cacert.pem root certificate (located in the `Documentation/ssl subfolder` of the SDK). The JDK includes a keytool application that will create such a jks file:

```
keytool -import -trustcacerts -file ../../../Documentation/ssl/
cacert.pem
-keystore cacert.jks -alias cacert
```

While generating cacert.jks the system asks the password. The correct password is 'password'.

After this, the sample is ready to run.

5.2 The PHP sample

The PHP sample shows how to use the Web Service SDK from PHP. This way, you can create your own website that interacts with a Switch Server.

Note: To use the PHP sample, you need a web server with PHP 5.3.1 or later, and PHP extensions php5-curl, php5-soap and php5-zip.

Note: The PHP sample uses HTML 5 features. If your web browser does not support HTML 5 the sample will still work in general, but it will not be as polished as with a HTML 5 compliant browser.

Examining the sample code

The PHP sample can conceptually be split into the parts Core, Application, Model-View-Controller (MVC), and Application Assets.

The Core part contains the main sample code, which handles the communication with the Switch Server.

Core component	Description
lib/butterfly.v2.0.wsdl	The service WSDL file. This contains the URL to the endpoint, http://127.0.0.1:51008 by default
lib/SwitchClientService.v2.0.php	The client classes, based on the WSDL
lib/soap-patterns/*	XML patterns for SOAP DIME requests

The Application part contains the sample PHP application, which uses the Core part for all Switch-related communication.

Application component	Description
index.php	The main entry point for the application
app/controllers/*	Application controller classes
app/views/*	The view classes for the controllers
app/views/header.php	The header template for the sample
app/views/footer.php	The footer template for the sample

The MVC component is a simple implementation of the Model-View-Controller pattern.

MVC component	Description
lib/mvc/*	A simple implementation of the MVC pattern to run the sample

The Application assets are assets used by the application, as well as a folder to cache binary data received from the Switch Server.

Application asset component	Description
public/images/*	Standard images and icons for the application
public/javascripts/*	Application javascripts
public/stylesheets/*	Application styles
public/thumbnails/*	In this folder, thumbnails will be stored. This folder is created automatically if it does not exist; the PHP user should have write access here.

Running the PHP sample

Note: The procedure to set up a web server and enable PHP depends very much on your choice of web server and Operating System. In what follows, we will give an example, but it is assumed the reader is familiar enough with web servers to make it work on their own system.

The PHP sample depends on some specific settings, which can be changed in the php.ini file. This table gives an overview:

Setting	Value	Remarks
allow_call_time_pass_reference	Off	
always_populate_raw_post_data	Off	
arg_separator.input	&	
arg_separator.output	&	
asp_tags	Off	
file_uploads	On	
upload_max_filesize	2M	Change this if you need to upload bigger files
post_max_size	8M	Change this if you need to upload bigger files
memory_limit	128M	
log_errors	On	
magic_quotes_gpc	Off	
magic_quotes_runtime	Off	
magic_quotes_sybase	Off	
max_file_uploads	20	
max_input_nesting_level	64	
max_input_time	60	
register_argc_argv	Off	
register_globals	Off	
register_long_arrays	Off	
request_order	GP	
variables_order	GPCS	
safe_mode	Off	
short_open_tag	Off	

Note: Before running the PHP sample, make sure the subfolder 'public' is writable for the user running the PHP server.

Example: running the PHP sample on Mac OS X

Mac OS X 10.6 (Snowleopard) comes with a built-in web server with PHP. However, PHP is by default not enabled.

To enable it, open the file `/etc/apache2/httpd.conf`. Locate the line,

```
#LoadModule php5_module        libexec/apache2/libphp5.so
```

Remove the leading hash (#), and save the file. A default php.ini file which matches our needs is already available, but it is disabled. To enable it, rename or copy the file `/etc/php.ini.default to /etc/php.ini`.

Now PHP is enabled, and we just need to copy the sample to the web server: copy the contents of the Samples/PHP subfolder of the SDK to `/Library/WebServer/Documents`.

As a final step, we need to enable the web server. In the Mac OS X System Preferences, go to Sharing, and enable the Web Sharing checkbox. Open a web browser, and go to http://localhost/index.php.

6. Reference

This chapter presents reference documentation for all the SOAP calls, their arguments, and their return values. See the file `butterfly.wsdl` in the `Documentation/wsdl` subfolder of the SDK for more information.

6.1 Operations

CheckConnect

Returns a reserved integer if a connection can be established, and the Switch Server is accepting SOAP calls. The HTTP username and password must not be set for this call.

SOAP method	CheckConnect
Parameters	None
Returns	Integer param-1: reserved

CheckUserNameAndPass

Returns a reserved integer if the username and password, set on the HTTP layer, correspond to an existing Switch user. The username must be set to the Switch username, a semi colon, and the code “EN”: "<username>;EN".

SOAP method	CheckUserNameAndPass
Parameters	None
Returns	Integer param-3: reserved

CloseConnection

Tells Switch that the information related to this client connection can be deleted.

SOAP method	CloseConnection
Parameters	None
Returns	Integer param-4: reserved

DownloadFile

Downloads a job (file or folder) from a checkpoint.

SOAP method	DownloadFile
Parameters	String flowId: the ID of the flow in which the job resides String checkPointId: the ID of the checkpoint in which the job resides String jobId: the ID of the job to download
Returns	Base64Binary inSubmitData: the zipped job

GetCheckPointName

Returns the name of the specified checkpoint.

SOAP method	GetCheckPointName
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint
Returns	String checkPointName: the name of the checkpoint

GetCheckPoints

Returns the list of checkpoint IDs for the specified flow.

SOAP method	GetCheckPoints
Parameters	String flowId: the ID of the flow from which to get the checkpoints
Returns	String list checkPoints: the list of checkpoints in the flow

GetCheckpointJobInfo

Returns information about a specific job. Only the information about jobs waiting in a checkpoint can be requested. See also GetCheckpointJobsInfo.

SOAP method	GetCheckpointJobInfo
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint String jobId: the ID of the job
Returns	JobStatus list jobInfo: a list with exactly one JobStatus element, containing the information for the job.

GetCheckpointJobs10

Returns the list of job IDs for the jobs waiting in the specified checkpoint.

SOAP method	GetCheckpointJobs10
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint
Returns	String list jobIds: the list of job IDs for jobs present in the checkpoint

GetCheckpointJobsCount

Returns the number of jobs waiting in the specified checkpoint.

SOAP method	GetCheckpointJobsCount
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint
Returns	Integer jobs: the number of jobs waiting in the checkpoint

GetCheckpointJobsInfo

Returns information about all the jobs waiting in a checkpoint. See also GetCheckpointJobInfo.

SOAP method	GetCheckpointJobsInfo
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint
Returns	JobIdStatus list jobInfos: a list with the information for each job in the checkpoint.

GetFlowList

Returns the flow IDs of all the flow on the Switch Server. This includes both active and inactive flows.

SOAP method	GetFlowList
Parameters	None
Returns	String list flowList: the IDs of the flows

GetFlowName

Returns the name of the flow with the given ID.

SOAP method	GetFlowName
Parameters	String flowId: the ID of the flow
Returns	String flowName: the name of the flow

GetFlowState

Returns the state of the specified flow.

SOAP method	GetFlowState
Parameters	String flowId: the ID of the flow
Returns	Integer flowState: the state of the flow. Possible values are: 1: the flow is stopped (inactive) 2: the flow is being started 3: the flow is running (active) 4: the flow is being stopped 5: the flow is invalid

GetJobLockStatus

Returns the name of the user who locked the job, or the empty string if the job is not locked.

SOAP method	GetJobLockStatus
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint String jobId: the ID of the job
Returns	String userName: the name of the user who locked the job, or an empty string

GetJobsAdditionalInfo

Returns the file type, size, modification date, and in the case of job folders the number of files for the specified job(s).

When using this function, you should always check the status member of the returned AdditionalInfo: if busy or failed, the data is not reliable.

SOAP method	GetJobsAdditionalInfo
Parameters	AdditionalInfoID list jobsId: the list of job IDs for which to get information
Returns	AdditionalInfo list jobsInfo: the information for each job

GetJobsHeavyInfo

Returns the thumbnail icon, number of pages and dimensions for the job.

When using this function, you should always check the status member of the returned HeavyInfo: if busy or failed, the data is not reliable.

The stamp member of the returned HeavyInfo is used to determine whether the heavy info should be refreshed: if calling this function more than once for the same job, you should set it to the value last returned by the Switch Server, to prevent unnecessary calculations.

SOAP method	GetJobsHeavyInfo
Parameters	HeavyInfoID list jobsId: a list of job IDs and information required to get the heavy info.
Returns	HeavyInfo list jobsInfo: the requested info per job.

GetJobsMetadataInfo

Returns the metadata for one or more specified jobs.

SOAP method	GetJobsMetadataInfo
Parameters	MetadataInfoID list jobsId: the list of job IDs for which to get the metadata
Returns	MetadataInfo list jobsInfo: the requested info per job

GetLogData10

Returns log messages from the Switch Server. Note that if there are a lot of log messages, this can take a long time. Because of that, lastTime should be chosen to minimize the number of messages to be returned.

SOAP method	GetLogData10
Parameters	String lastTime: the time of the earliest log that should be returned
Returns	LoggedRecord list logs: the list of log messages

GetProgressData

Returns progress data from the Switch Server.

SOAP method	GetProgressData

Parameters	Integer lastHours: progress data for this number of hours will be returned
Returns	ProgressRecord list logs: the progress data

GetStatus

Returns the status of the connection, and some information about the Switch Server.

SOAP method	GetStatus
Parameters	None
Returns	Status list res: a list with exactly one element of type Status

GetSubmitPointName

Returns information about the specified submit point.

SOAP method	GetSubmitPointName
Parameters	String flowId: the ID of the flow that contains the submit point String submitPointId: the ID of the submit point
Returns	SubmitPointData list submitPointData: a list with exactly one SubmitPointData item

GetSubmitPoints

Returns the list of submit points for a given flow.

SOAP method	GetSubmitPoints
Parameters	String flowId: the ID of the flow
Returns	String list submitPoints: the IDs of the submit points

GetUserJobs

Get information about jobs currently residing in the Switch Server.

SOAP method	GetUserJobs
Parameters	Integer hours: any jobs active within this last number of hours will be returned Integer flags: reserved
Returns	UserJob list userJobs: the list of jobs on the Switch Server

PushCheckpointJob

Pushes the specified job to one or more connections.

SOAP method	PushCheckpointJob
Parameters	String flowId: the ID of the flow String checkPointId: the ID of the checkpoint String jobId: the ID of the job

	PushJobStatus jobInfo: the necessary information to push the job to the connection(s)
Returns	Integer param-5: reserved

ReplaceFile

Replaces a job in a checkpoint, and pushes the result to one or more outgoing connection(s).

SOAP method	ReplaceFile
Parameters	ReplaceEntry inEntry: information about the job replacement
	Base64Binary inSubmitData: the job to submit; this must be a zip file containing either a single file or a single folder
	PushJobStatus jobInfo: the necessary information to push the (replaced) job to the connection(s)
Returns	Integer isReplaced: 1 if the job was replaced, 0 if not

SetJobLockStatus

Locks or unlocks a job.

SOAP method	SetJobLockStatus
Parameters	String flowId: the ID of the flow
	String checkPointId: the ID of the checkpoint
	String jobId: the ID of the job
	Integer lockStatus: one of the following values: 0: unlock 1: lock 2: solid unlock 3: solid lock
Returns	Integer isSet: 1 if set, 0 if not set

SubmitFile10

Submit a job to the specified submit point.

SOAP method	SubmitFile10
Parameters	SubmitEntry inEntry: information about the job to submit
	Base64Binary inSubmitData: the job to submit; this must be a zip file containing either a single file or a single folder
	MetadataToFillClass list metadata: the required metadata
Returns	String outJobId: reserved

DownloadReport

If a checkpoint has the Enable report viewing property set to Yes, this method can be used to download the dataset provided in the Report dataset name property. If Enable report viewing is off, or if the dataset doesn't exist, this method will fail.

SOAP method	DownloadReport
Parameters	String flowId: the ID of the flow in which the job resides String checkPointId: the ID of the checkpoint in which the job resides String jobId: the ID of the job whose report you wish to download
Returns	Base64Binary inSubmitData: the requested report

6.2 Types

JobStatus

Member	Type	Description
jobAllowMultipleOutputs	Boolean	True if the job may be sent to several output connections
jobName	String	The name of the job
fullUserName	String	The full name of the user who submitted the job
jobSize	Integer	The size of the job
jobSubmitTime	String	The timestamp of when the job was submitted
jobWaitingTime	String	The time the job has been waiting in the checkpoint
connections	List of ConnectionState	The list of output connections
jobMetadata	List of MetadataToDisplayClass	The list of read-only metadata
jobRequiresMetadata	Boolean	True if this job requires metadata
jobRequiredMetadata	List of MetadataToFillClass	The list of required metadata
jobEnableReportViewing	Boolean	True if report viewing is enabled
jobAllowReplacingJob	Boolean	True if the job may be replaced
jobReportName	String	The name of the report

AdditionalInfoID

Member	Type	Description
jobId	String	The ID of the job

AdditionalInfo

Member	Type	Description
status	Integer	The status of the request. Possible values: 0: ready 1: failed 2: busy
jobId	String	The ID of the job
fileType	String	The file type of the job
size	Integer	The size of the job, in bytes
files	Integer	The number of files in the job folder; undefined if the job is not a folder
modified	String	The modification timestamp for the job

HeavyInfoID

Member	Type	Description
stamp	String	The timestamp returned when the heavy info was last requested from the server, or empty for a first request.
jobId	String	The job ID
retreiveThumbnail	Boolean	If set to true, the thumbnail image will be generated and returned
thumbnailSize	Integer	The maximum thumbnail size, in pixels
retreivePages	Boolean	If true, the number of pages in the document will be returned
retreiveDimHor	Boolean	If true, the horizontal dimension of the job will be returned
retreiveDimVer	Boolean	If true, the vertical dimension of the job will be returned

HeavyInfo

Member	Type	Description
status	Integer	The status of the request. Possible values: 0: ready 1: failed 2: busy

Member	Type	Description
stamp	String	The modification timestamp for the job.
jobId	String	The job ID
thumbnailBase64	Base64Binary	The thumbnail, in png format (base-64 encoded)
pages	Integer	The number of pages in the document
dimensionsHor	Integer	The horizontal dimension of the document, in pixels
dimensionsVer	Integer	The vertical dimensions of the document, in pixels

MetadataInfoID

Member	Type	Description
jobId	String	The ID of the job

MetadataInfo

Member	Type	Description
status	Integer	The status of the request. Possible values: 0: ready 1: failed 2: busy
jobId	String	The ID of the job
dispMetadata	List of MetadataToDisplayClass	The list of read-only metadata
fillMetadata	List of MetadataToFillClass	The list of required metadata
metadataXmlStr	String	Metadata in Switch 10 format

LoggedRecord

Member	Type	Description
recId	Integer	The internal record ID
timeStamp	String	The time when the message was logged
module	String	The module that logged the message
serverName	String	The name of the server
flowName	String	The name of the flow
jobName	String	The name of the job
message	String	The logged message
type	String	The message type

ProgressRecord

Member	Type	Description
fullUserName	String	The full name of the user who submitted the job
jobName	String	The name of the job
serverName	String	The name of the server
flowName	String	The name of the flow
submitPoint	String	The name of the submit point where the job was submitted
submittedTime	String	The time when the job was submitted
state	String	The state of the job
enteredState	String	The time when this state was entered
onHold	String	Whether or not this job is on hold
checkpoint	String	The name of the checkpoint where this job is located, or empty if the job is not in a checkpoint

Status

Member	Type	Description
mServerName	String	The name of the Switch Server
mStatus	String	The status of the connection, e.g. “Connected”, “Number of client licenses exceeded” or “Wrong user or password”
mFlavourName	String	The flavor name, e.g. “Switch” for Switch 12 or “PowerSwitch” and “FullSwitch” for previous versions
mReleaseVersionNumber	Integer	The release version number, e.g. “12”
mReleaseType	String	The release type, e.g. “Beta”
mReleaseTypeNumber	Integer	The release type number, e.g. “1” for Beta 1
mUpdateVersionNumber	Integer	The update version number, e.g. “1” for 10, update 1

Member	Type	Description
mOperatingSystemName	String	The operating system name

SubmitPointData

Member	Type	Description
name	String	The name of the submit point
displayName	String	The display name of the submit point
displayThumbnailBase64	String	Reserved
metadataXmlStr	String	Metadata in Switch 10 format
metadata	List of MetadataToFillClass	The list of required metadata for the submit point

UserJob

Member	Type	Description
jobType	Integer	The current state of the job. Possible values: 0: unknown 1: submitted 2: submitting 3: replacing 4: processing 5: on hold 6: alert 7: complete
pointId	String	The ID of the submit point or checkpoint
pointName	String	The name of the submit point or checkpoint
flowId	String	The ID of the flow
isRequiresMetadata	Boolean	True if the job requires metadata
isAllowMultipleOutputs	Boolean	True if the job may be sent to multiple connections
pushConnections	List of ConnectionInfo	The list of connections
isEnableReportViewing	Boolean	True if report viewing is allowed
reportFileName	String	The name of the report file
isAllowReplacingJob	Boolean	True if the job can be replaced
prefix	String	The job prefix
name	String	The job name
fileType	String	The file type of the job
size	Integer	The size of the job, in bytes
files	Integer	The number of files in the job folder (recursive)

Member	Type	Description
pages	Integer	See HeavyInfo
dimensionHor	Integer	See HeavyInfo
dimensionVer	Integer	See HeavyInfo
modified	String	The modification timestamp for the job
flowName	String	The name of the flow in which this job resides
holdConnection	ConnectionInfo	If applicable, the connection info of the connection which is holding the job back
onHoldSince	String	The timestamp on which the job was first detected to be on hold
userName	String	The name of the user who submitted the job, or who has otherwise been associated with it
submittedTo	String	The name of the submit point to which this job was or is being submitted
initiated	String	The timestamp on which the job first arrived in a Switch flow (i.e. the time stamp of the first "Detected" or "Produced" occurrence in the jobs trail)
checkpointName	String	The name of the checkpoint in which this job is being held
onAlertSince	String	The timestamp since when the job resides in the checkpoint
state	String	The state of the job (corresponding to the "attach job state" property on Folder flow elements)
inStateSince	String	The timestamp on which the job entered the current state
clientLock	String	The client job lock id, if any
familyStarter	String	The job family starter id, if any
dispMetadata	List of MetadataToDisplayClass	A list of the display metadata fields (labels

Member	Type	Description
		and values) defined for the checkpoint in which the job resides
fillMetadata	List of MetadataToFillClass	A list of the required metadata fields (labels and values) defined for the checkpoint in which the job resides
metadataXmlStr	String	Metadata in Switch 10 format
backingItemAccessible	Boolean	Reserved
additionalDataComplete	Boolean	True if the data corresponding to the AdditionalInfo is complete
metadataComplete	Boolean	True if the data corresponding to the MetadataInfo is complete
internalError	Integer	Reserved

ConnectionInfo

Member	Type	Description
connId	String	The ID of the connection
connName	String	The name of the connection

PushJobStatus

Member	Type	Description
jobRequiredMetadata	List of MetadataToFillClass	The required metadata
metadataXmlStr	String	Metadata in Switch 10 format
connectionIds	List of Strings	The IDs of the connections where this job should be pushed
jobName	String	The name of the job; the job can be renamed by changing this

SubmitEntry

Member	Type	Description
flowId	String	The ID of the flow
submitPointId	String	The ID of the submit point
jobName	String	The name of the job
jobOrigin	String	The origin of the job

Member	Type	Description
metadataXmlStr	String	Metadata in Switch 10 format

ReplaceEntry

Member	Type	Description
flowId	String	The ID of the flow
checkPointId	String	The ID of the checkpoint
jobId	String	The ID of the (existing) job
filename	String	The name of the new job

6.3 Switch metadata format

Starting from Switch 10 new metadata feature was implemented: dependent metadata fields. Due to this the metadata format had been changed.

In order to keep compatibility with previous versions it was decided to extend existing SOAP calls with additional field containing metadata in new format.

The metadata is stored in the following format:

```
<ValueDescription Type="metadatafields">
<<Fields list>>
</ValueDescription>
```

Where <<Fields list>> is a list of xml tags in format:

<FieldID @Format @LocalizedTagName @Subtype @Tooltip @Type @ReadOnly @ValueIsRequired @Editor @FormatEditor @DisplayField @RememberLastValue @Dependency @Dependencyvalue>Default value</spMF_1>

When metadata came from Switch to Client tag contains default field value.

Attribute	Description
FieldID	Identification of the field, generated automatically by Switch
LocalizedTagName	Name of field displayed on UI
Tooltip	Tooltip of the field
Type	Type of field. Possible values are: - bool (Yes/No value) - enum (drop down list) - string (text edit) - date (datetime) - number (number edit) - password (password edit) - time (hours and minutes)
Subtype	Always “inline”
Editor	Always “inline”
Format	Indicates format of the field input. Actual only for “string” and “number” Types. It should be regular expression

FormatEditor	Specifies format of editor. Possible values are “inline” if format set as string and “regexp” if format is regular expression. Used only when modifying metadata fotmat
ReadOnly	Indicates if field is readonly. Possible values are “Yes”/”No”
ValueIsRequired	Indicates if field is mandatory or not. Possible values are “Yes”/”No”
DisplayField	Indicates if field should be shown on UI. Possible values are “Yes”/”No”
RememberLastValue	Indicates that Metadata editor should remember value entered last time, and provide it as default value. Possible values are “Yes”/”No”
Dependency	If field has this attribute it means that current field is a child of field specified in Dependency attribute
Dependencyvalue	This attribute specifies that current field will be displayed only when parent field has given value

Sample metadata:

```
<ValueDescription Type="metadatafields">
<spMF_1 Format="[0-9]{5}" LocalizedTagName="Customer Code" Subtype="inline"
 Tooltip="Please enter the customer code.&#xa;&#xa;e.g. 00234&#xa;&#xa;" Type="string"
 ReadOnly="No" ValueIsRequired="Yes" Editor="inline" FormatEditor="regexp"
 DisplayField="Yes" RememberLastValue="No"></spMF_1>
<spMF_2 Format="" Dependencyvalue="00123" LocalizedTagName="Publication"
 Subtype="inline" Tooltip="" Type="enum:TravelMagazine;NewsToday;" Dependency="spMF_1"
 ReadOnly="No" ValueIsRequired="No" Editor="inline" FormatEditor="inline"
 DisplayField="Yes" RememberLastValue="No"></spMF_2>
<spMF_3 Format="" Dependencyvalue="00123" LocalizedTagName="Color" Subtype="inline"
 Tooltip="Output color" Type="enum:B&amp;W;CMYK only;CMYK + Spot" Dependency="spMF_1"
 ReadOnly="No" ValueIsRequired="No" Editor="inline" FormatEditor="inline"
 DisplayField="Yes" RememberLastValue="No">CMYK only</spMF_3>
<spMF_4 LocalizedTagName="Express Print" Subtype="inline" Tooltip="" Type="bool"
 ReadOnly="No" ValueIsRequired="No" Editor="inline" DisplayField="Yes"
 RememberLastValue="No"></spMF_4>
<spMF_5 Format="" Dependencyvalue="Yes" LocalizedTagName="Timing" Subtype="inline"
 Tooltip="" Type="enum:Same day;Next day;3 day;" Dependency="spMF_4" ReadOnly="No"
 ValueIsRequired="No" Editor="inline" FormatEditor="inline" DisplayField="Yes"
 RememberLastValue="No"></spMF_5>
<spMF_10 Format="" Dependencyvalue="No" LocalizedTagName="Process date"
 Subtype="calendardate" Tooltip="" Type="date" Dependency="spMF_4" ReadOnly="No"
 ValueIsRequired="No" Editor="calendardate" FormatEditor="inline" DisplayField="Yes"
 RememberLastValue="No"></spMF_10>
<spMF_9 LocalizedTagName="Hard copy request" Subtype="inline" Tooltip=""
 Type="bool" ReadOnly="No" ValueIsRequired="No" Editor="inline" DisplayField="Yes"
 RememberLastValue="No"></spMF_9>
</ValueDescription>
```