Service Postal

API Description

Version 11.3 – 23/10/2015

Table of Contents

Introduction

The purpose of this document is to describe the API feature that allow a client to submit letters to ours servers from its own application.

Services Overview

Service Postal Web Service is divided into three main services. Each of them provides you with functions that are specific to the type of action you want to perform:

Session Service: The Session Service is used to open a session context that will be used by the other services later. This is the first service you need to instantiate, as it is an entry point to all other services.

Submission Service: The Submission Service is used to use Service Postal services, launch printing.

Query Service: The Query Service is used to query information from our servers about previously submitted jobs.

Typical API call sequence

Step 1 - Initialization and authentication

This will allow you to log in the proper server and create a new session:

- Instantiate a `SP_SessionService` object (client).
- Retrieve the Web service bindings (which contains the proper URLs to associate with each instantiated service), using the `sp_get_bindings` function from instantiated **`SP_SessionService`** object.
- Set URL of instantiated `SP_SessionService` object to the appropriate URL retrieved by `sp_get_bindings` in `SP_BindingList` object using `sp_set_url` function.
- Log on the server using the `sp_login` function, which retrieves the `SP_LoginResult` object which contains field `SessionID`.
- Instantiate `SP_SubmissionService` and `SP_QueryService` objects. Set URL's of instantiated objects using `sp_set_url` function with appropriate urls from `SP_BindingList` retrieved earlier with `sp_get_bindings` function. Create `SP_SessionHeader` object (for each `SP_SubmissionService` and `SP_QueryService`) and set its `SP_SessionID` field to `SessionID` value retrieved in `SP_LoginResult` earlier.

Step 2 - Submit a document

All the functions below are available for use after the first step from `SP_SubmissionService` instantiated object:

- Upload the attachment file(s) on the server, using the `sp_upload_file` function to initiate the upload or the `sp_append_file` (for bigger files) function.

- If a validation is necessary, retrieve the preview of the submitted job with a call to `sp_preview_letter` or `sp_preview_mailing` to display the results to the users. Then use the `sp_validate_letter` or `sp_validate_mailing` to start the task.
- If a validation is not needed, submit the job with the `sp_submit_letter` or `sp_submit_mailing` functions to start the printing and sending process.

Step 3 - Document tracking

All the functions below are available for use after the first step from `SP_QueryService` instantiated object. After you have submitted a document to the server, you want to know the state in which you submission is in real time, to make sure no error occurred and that your documents have been effectively sent. To do so:

- Call `sp_query_status` to get the status of the document and the tracking ID if available
- Call `sp_query_document` to retrieve a document like a proof of delivery or the content of the letter which has been sent
- Call `sp_query_cost` to get the price of the submitted task

Step 4 - Release the session and its allocated resources.

Use the `sp_logout` function of the `SP_SessionService` object to release the session on the server.

Uploading attachments to the server

When you need to upload big attachments to the server, it is strongly recommended that you upload it by chunks of data to avoid losing the connection and needing to re-upload the file. To do so, use the `sp_append_file` function of the submission service.

Session Service overview

User authentication

The authentication of a user must be done in three steps, first by retrieving the correct URL that will be used by the service, then by binding the url to the service and last by logging in. This is done by calling two functions provided by the `SP_SessionService`

1. First, `sp_get_bindings` : returns a list of service entry points (URLs) to use in your application.
2. Then, bind the retrieved url to the service.
3. And last, call `sp_login`: returns a unique session identifier that must be set in the session headers so that the API calls are authenticated.

Transfer protocols

In order to guarantee an optimal protection of the information, all connections are made through secure HTTP. Secure HTTP is the standard encrypted communication mechanism on the World Wide Web, which is actually HTTP over SSL.

Submission Service overview

The `SP_SubmissionService` object allows the client to submit documents, generate previews, upload files on the server.

Before any use, initialize a `SP_SubmissionService` object with an URL and the `SP_SessionID` provided by a `SP_SessionService` object:

1. Set the service URL with the `spSubmissionServiceLocation` parameter returned by `sp_get_bindings` using `sp_set_url` function from `SP_SubmissionService` object.
2. Initialize the `SP_SessionHeader` object, field: `SP_SessionID` with `SessionID` value returned by `sp_login` function in `SP_LoginResult` object.

The client sends us document using either two of the following functions: `sp_upload_file` or `sp_append_file`

The `sp_upload_file` function must be used with file smaller than 64 kilobytes. This function returns a `SP_File` object

The `sp_append_file` function can be used to upload large files in several parts on the server to avoid network timeouts. We recommend using 64 kilobytes packets. To use the `sp_append_file` function, a first call to the `sp_upload_file` function must be made to get a `SP_File` object.

When calling a submit function (such as `sp_submit_letter` or `sp_submit_mailing`), the task will be processed by Service Postal teams who will be print and send the letter. It is the caller's responsibility to check with the user that the letter has been validated (using `sp_preview_letter` for example). The preview step is not mandatory in the printing/sending process.

So if a validation step is needed, the function `sp_preview_letter` (or `sp_preview_mailing`) should be used, followed by a call to `sp_validate_letter` (or `sp_validate_mailing`).

If no validation step is required, then the function `sp_submit_letter` or the function `sp_submit_mailing` should be called.

Query Service overview

The `SP_QueryService` object allows the client retrieve data about previously submitted jobs via `sp_query_status, sp_query_document` and `sp_query_letter_cost` functions.

Before any use, initialize a `SP_QueryService` object with an URL and the `SP_SessionID` provided by a `SP_QueryService` object:

1. Set the service URL with the `spQueryServiceLocation` parameter returned by `sp_get_bindings` using `sp_set_url` function from `SP_QueryService` object.
2. Initialize the `SP_SessionHeader` object, field: `SP_SessionID` with `SessionID` value returned by `sp_login` function in `SP_LoginResult` object.

API Quick Reference

List of available API functions

Function name	Description
Session Service	
sp_get_bindings	Requests the list of service entry points (URLs) to use in the client application.
sp_get_session_information	Retrieves the logged user's data. (can be called only after sp_login)
sp_login	Opens an authenticated session on the server.
sp_logout	Closes an authenticated session on the server.
sp_set_url	Set the service URL (must be called before just after the sp_get_bindings, and before sp_login).
Submission Service	
sp_preview_letter	Previews a document with the given printing parameters. Returns a link to a PDF file or the PDF file in byte array depending on the parameter sent.
sp_submit_letter	Submits a document with the given printing parameters.
sp_upload_file	Uploads some data to the server for later availability.
sp_append_file	Uploads some large data on the server. (sp_upload_file must be called first to initiate uploading)
sp_submit_mailing	Submits a document and a csv file with the given printing parameters for starting a mailing printing job
sp_preview_mailing	Previews a document from a mailing job with the given printing parameters. Returns a link to a PDF file or the PDF file in byte array depending on the parameter sent.
sp_validate_letter	Submits a previously previewed letter
sp_validate_mailing	Submits a previously previewed mailing
sp_set_url	Set the service URL (must be called before any other function).
sp_cancel_job	Cancels a previously submitted or previewed job. If the letter(s) have already been printed, the cancellation is not possible. A failure status is returned
sp_set_session_header	Sets the unique session id value. Must be called before any other function of this service to instantiate session on the service, as well as sp_set_url. The order of calling sp_set_session_header and sp_set_url isn't important, but it is important that both should be called prior instantiation of service object.
sp_estimate_price	Returns the price of a letter from an SP_JobParameters object and the expected number

Function name	Description
	of pages of the document. The function returns also the price of the stamp.
sp_preview_mailing_pdf	Previews a document from a mailing job with the given printing parameters. The caller is responsible for building the mailing. The whole letters made by the caller will be transmitted through a single pdf file. All the letters must have the same number of pages. The function will cut the pdf file into letters. Returns a link to a PDF file or the PDF file in byte array depending on the parameter sent.
Query Service	
sp_query_status	Returns the status of a specified letter (using the spServicePostalID of the task): printed, given to the Postal Office, received, distributed. Returns also the tracking ID from La Poste when available
sp_query_document	Retrieves a document from a specified task (using the spServicePostalID).
sp_query_letter_cost	Returns the cost of a specified letter (using the spServicePostalID of the task).
sp_query_letter_rate	Returns the rate of a specified letter (using the spServicePostalID of the task). The function returns also the price of the stamp.
sp_set_url	Set the service URL. (must be called before any other function).
sp_set_session_header	Sets the unique session id value. Must be called before any other function of this service to instantiate session on the service, as well as sp_set_url. The order of calling sp_set_session_header and sp_set_url isn't important, but it is important that both should be called prior instantiation of service object.

List of available API Objects

Object name	Description
`SP_SessionService`	The SP_SessionService object allows the client to authenticate itself and open a session context that will be used with other services. This is the very first service you need to instantiate, as it is an entry point to all other services
`SP_SubmissionService`	The SP_SubmissionService object allows the client to submit documents, generate previews, upload files on the server.
`SP_SessionHeader`	SessionHeader which includes SessionID retrieved by SP_LoginResult
`SP_LoginResult`	Retrieves the unique SessionID
`SP_QueryService`	The SP_QueryService object allows the client to retrieve data about previously submitted jobs.
`SP_BindingList`	This structure describes the bindings (also known as locations, or Web Service's URLs) to use with your Web services objects. This object is returned by a call to sp_get_bindings
`SP_File`	This structure specifies a file reference, who can be stored inside the object or who has been previously uploaded to the server

Object name	Description
SP_JobParameters	This structure contains all the parameters needed to print a letter and send it to its recipient
SP_PostalAddress	This structure stores a postal address with the names of the person (last name, first name, company name)
SP_JobResult	This structure contains the result of a printing task submission
SP_PreviewResult	This structure contains the result of a preview task.
SP_QueryFileResult	This structure contains the result of a file query
SP_MailingResult	This structure contains the result of a mailing task
SP_PreviewMailingResult	This structure contains the result of a preview mailing task
SP_DocumentStatusResult	This structure contains the result of document's status query
SP_ServiceCode	This structure contains a service code and the corresponding price for a unitary service provided by Service Postal
SP_QueryRateResult	This structure contains the result of a query about rate and stamp price of a letter

List of available API Enumeration

Enumeration name	Description
SP_LetterTypeEnum	List of possible type of letters (registered, registered with proof of delivery, economic letter, green letter, priority letter)
SP_ColorEnum	List of possible color printing options
SP_RectoEnum	List of recto or recto/verso options
SP_EnveloppeEnum	List of enveloppe options
SP_PaperFormatEnum	List of paper format options
SP_YesNoEnum	Yes or no values
SP_FIleStorageModeEnum	List of possible storage mode in a SP_File object
SP_DocumentTypeEnum	List of all possible document type that could be queried by the API
SP_DocumentStatusEnum	List of all possible document status that could be queried by the API
SP_SuccessFailureEnum	Success or Failure values

Objects description

SP_LoginResult object

Object of this type is retrieved prior to call sp_login method from SP_SessionService object.

It returns SessionID.

Properties name	Type	Description
spSessionID	String	Unique sessionID for the current user retrieved from server
spErrorMessage	String	Error message

SP_BindingList object

This structure describes the bindings (also known as locations, or Web Service's URLs) to use with your Web services objects. This object is returned by a call to sp_get_bindings.

Properties name	Type	Description
spSessionServiceLocation	String	URL for SP_SessionService object
spSubmissionServiceLocation	String	URL for SP_SubmissionService object
spQueryServiceLocation	String	URL for SP_QueryService object
spErrorMessage	String	Error message

SP_SessionInformation object

This structure describes returned the data about the logged in user.

Properties name	Type	Description
spEmail	String	Client’s email
spLogin	String	Client’s username
spCompany	String	Client’s company
spErrorMessage	String	Error message

SP_PostalAddress object

This structure stores a postal address with the names of the person (last name, first name, company name)

Properties name	Type	Description
spFirstName	String	First name of the person
spLastName	String	Last name of the person
spCompanyName	String	Name of the company
spFirstLine	String	First line of the postal address
spSecondLine	String	Second line of the postal address
spPostalCode	Num	Postal code
spCityName	String	City name
spCountry	String	Country

SP_JobParameters object

This structure contains all the parameters needed to print a letter and send it to its recipient. This parameters could be stored inside the object or linked to a previously uploaded file who contains the parameters

Properties name	Type	Description
spLetterType	SP_LetterTypeEnum	Type of the letter to send. Mandatory
spRecipient	SP_PostalAddress	Address and name of the recipient. Mandatory
spSender	SP_PostalAddress	Address and name of the sender. Mandatory for registered letters
spColorParameter	SP_ColorEnum	Color or B&W printing. Mandatory
spRectoParameter	SP_RectoEnum	Recto or Recto-Verso printing. Mandatory
spEnveloppeParameter	SP_EnveloppeEnum	Size of the envelope to use. Mandatory
spPaperFormat	SP_PaperFormatEnum	Paper format to use (only A4 allowed for the moment). Mandatory
spHeaderPage	SP_YesNoEnum	Indicates whether or not a header page with name and address of the recipient and of the sender must be added. Mandatory

`spDepositSlipHandling`	SP_YesNoEnum	Indicates whether or not the deposit slip must be returned to Service Postal address. This option must be activated on the client account. If the deposit slip handling is asked and is not allowed for the demanding client, an error will occur
`spExternalReference`	String	Client reference for this letter for binding the letter to its own system

SP_JobResult object

This structure contains the result of a printing task submission.

Properties name	Type	Description
`spServicePostalID`	String	Internal reference of the submitted job. Used for querying information about this job
`spStatus`	Integer	0 means no error, below value means error occurs
`spTotalCost`	Numeric	Total cost of the submitted printing job
`spErrorMessage`	String	Error message
`spExpectedDispatchNotice`	Integer	Expected notice (in working days) for the letter to be dispatched. 0 means that the letter will be sent today. 1 means that the letter will be sent tomorrow

SP_PreviewResult object

This structure contains the result of a preview task.

Properties name	Type	Description
`spOutputFile`	SP_File	PDF file produced by the preview function
`spStatus`	Integer	0 means no error, below value means error occurs
`spTotalCost`	Numeric	Total cost of the printing job if submitted
`spServicePostalID`	String	Internal reference of the previewed job. To be used for starting the job
`spErrorMessage`	String	Error message

spEstimatedDispatchNotice	Integer	Estimated notice (in working days) for the letter to be dispatched. 0 means that the letter will be sent today. 1 means that the letter will be sent tomorrow. This notice is informational, because it will depends on the date and hour of the real submission of the job.

SP_QueryFileResult object

This structure contains the result of a file query.

Properties name	Type	Description
spOutputFile	SP_File	File requested
spStatus	Integer	0 means no error, below value means error occurs
spErrorMessage	String	Error message

SP_MailingResult object

This structure contains the result of a mailing task.

Properties name	Type	Description
spServicePostalID	String	Internal reference of the submitted job. Used for querying information about this job
spStatus	Integer	0 means no error, below value means error occurs
spLettersCount	Numeric	Number of letters produced by the mailing task (even if only one document is asked to be printed)
spTotalCost	Numeric	Total cost of the submitted printing job
spErrorMessage	String	Error message
spExpectedDispatchNotice	Integer	Expected notice (in working days) for the letter to be dispatched. 0 means that the letter will be sent today. 1 means that the letter will be sent tomorrow

SP_PreviewMailingResult object

This structure contains the result of a preview mailing task.

Properties name	Type	Description

`spOutputFile`	SP_File	PDF file produced by the preview function
`spStatus`	Integer	0 means no error, below value means error occurs
`spLettersCount`	Numeric	Number of letters produced by the mailing task (even if only one document at a time is returned by the preview call)
`spTotalCost`	Numeric	Total cost of the printing job if submitted
`spServicePostalID`	String	Internal reference of the previewed mailing. To be used for starting the mailing
`spErrorMessage`	String	Error message
`spEstimatedDispatchNotice`	Integer	Estimated notice (in working days) for the letter to be dispatched. 0 means that the letter will be sent today. 1 means that the letter will be sent tomorrow. This notice is informational, because it will depends on the date and hour of the real submission of the job.

SP_File object

This structure specifies a file reference, who can be stored inside the object or who has been previously uploaded to the server.

Properties name	Type	Description
`spName`	String	The logical name of the file
`spStorageMode`	SP_FIleStorageModeEnum	Specifies where the file content is located
`spContent`	Array of byte values	The file content when the storage mode parameter is SP_INLINED
`spURL`	String	The file content when the storage mode parameter is SP_UPLOADED
`spFileID`	String	Unique file id. This is the key to access the file within the API

SP_DocumentStatusResult object

This structure contains the result of querying the status of a document.

Properties name	Type	Description
`spStatus`	SP_DocumentStatusEnum	The status of the document (submitted, printed, …)

spTrackingID	String	Tracking ID used to track the letter on “La Poste” website. If the tracking ID is not available, the string is empty

SP_ServiceCode object

This structure contains a service code and the corresponding price for a unitary service provided by Service Postal. The list of the unitary services and their corresponding code is provided further in this document.

Properties name	Type	Description
spCode	String	Code of the unitary service
spPrice	Numeric	Price (in euros) of one unitary service (billed by Service Postal). To get the total price, you need to multiply spPrice by spQuantity
spQuantity	Numeric	Number of unitary services used
spVAT	Boolean	True if the VAT apply to the unitary service

SP_QueryRateResult object

This structure contains the result of a query about rate code and price of a letter.

Properties name	Type	Description
spServiceCodeList	List of SP_ServiceCode object	List of all the service codes used to build and send a letter or a mailing
spWeight	Numeric	Expected weight of the letter in gram
spServicePrice	Numeric	Price of the service to build the letter without the VAT (in euros).
spStampPrice	Numeric	Price of the stamp to send the letter (in euros). There is no VAT on stamp prices

SP_LetterTypeEnum object

List of possible type of letters (registered, registered with proof of delivery, economic letter, green letter, priority letter)

Enum values	Type	Description

SP_REGISTERED_LETTER	Enum	Registered letter without proof of delivery
SP_REGISTERED_WITH_PROOF	Enum	Registered letter with proof of delivery
SP_PRIORITY_LETTER	Enum	Priority letter
SP_ECONOMIC_LETTER	Enum	Economic letter (ecopli)
SP_GREEN_LETTER	Enum	Green letter

SP_ColorEnum object

List of possible color printing options

Enum values	Type	Description
SP_BLACK_AND_WHITE	Enum	Black and white printing
SP_COLOR	Enum	Color printing

SP_RectoEnum object

List of recto or recto/verso options

Enum values	Type	Description
SP_RECTO_VERSO	Enum	Recto/verso printing
SP_RECTO	Enum	Recto printing

SP_EnveloppeEnum object

List of envelope options

Enum values	Type	Description
SP_DL_ENVELOPPE_THIRD_A4	Enum	Use a DL envelope. Paper creased in three
SP_C5_ENVELOPPE_HALF_A4	Enum	Use a C5 envelope. Paper creased in two
SP_C4_ENVELOPPE_A4	Enum	Use a C4 envelope. Paper not creased

SP_PaperFormatEnum object

List of paper format options

Enum values	Type	Description
`SP_A4_FORMAT`	Enum	A4 paper format
`SP_A3_FORMAT`	Enum	For future use, not allowed for the moment

SP_YesNoEnum object

Yes or no values

Enum values	Type	Description
`SP_YES`	Enum	Yes
`SP_NO`	Enum	No

SP_FIleStorageModeEnum object

List of possible storage mode in a SP_File object.

Enum values	Type	Description
`SP_INLINED`	Enum	File is stored inside the SP_File object
`SP_UPLOADED`	Enum	File has been uploaded to our servers. An URL must be provided

SP_DocumentTypeEnum object

List of all possible document type that could be queried by the API.

Enum values	Type	Description
`SP_DOCUMENT`	Enum	Letter content sent by the client
`SP_PROOF_OF_DEPOSIT`	Enum	Proof of deposit to the postal office, generated by Service Postal server
`SP_PROOF_OF_DELIVERY`	Enum	Proof of delivery (in the case the deposit slip is sent back to our office with the appropriate option)

SP_DocumentStatusEnum object

List of all possible document status that could be queried by the API

Enum values	Type	Description
`SP_SUBMITTED`	Enum	The letter has been sent to Service Postal server and the job of printing and sending has been submitted
`SP_PRINTED`	Enum	The letter has been printed
`SP_SCANNED`	Enum	The letter has been scanned and is about to be carried to the post office
`SP_DELIVERED`	Enum	The letter has been delivered. For now I cannot see how this status could be returned as we don't know when the letter is delivered.
`SP_DEPOSIT_SLIP_RECEIVED`	Enum	The deposit slip has been received by Service Postal
`SP_UNKNOWN`	Enum	The letter is not found in our database
`SP_PREVIEWED`	Enum	The letter has been previewed but not validated

SP_SuccessFailureEnum object

Success or Failure values

Enum values	Type	Description
`SP_SUCCESS`	Enum	Success
`SP_FAILURE`	Enum	Failure

Functions description

Session service

sp_get_bindings

Requests the list of service entry points (URLs) to use in the client application.

Parameters name	Type	Description
`spUsername`	string	Client’s username
Returns	**Type**	**Description**
`SP_BindingList`	`SP_BindingList`	Contains URLs for each of the services
Exceptions	**Type**	**Description**
	`FaultException`	`Username not submited`
	`FaultException`	`Wrong username`
	`FaultException`	`Database problem`

C# sample

Before starting with examples, if using .NET client for calling web service, web services should be added to the service reference with provided WSDL files. Also make sure that the app.config or web.config of the application has this part:

```
     <bindings>
            <basicHttpBinding>
                <binding name="BasicHttpBinding_IQuery" maxReceivedMessageSize="2147483647" />
                <binding name="BasicHttpBinding_ISubmission" maxReceivedMessageSize="2147483647" />
                <binding name="BasicHttpBinding_ISession" maxReceivedMessageSize="2147483647" />
            </basicHttpBinding>

        </bindings>
```

`maxReceivedMessageSize="2147483647"` is attribute that user should add himself – it won’t be added automatically. The value `2147483647` is defined by client, and it presents the number of bytes that its application can receive from Web Services. (for example when receiving file in bytes).

The following C# sample code shows how to retrieve the entry points to use with the Web services objects.

```
SessionEnt.SessionClient sessionclient = new SessionEnt.SessionClient();
SessionEnt.SP_BindingObject bindings = sessionclient.sp_get_bindings("username");
sessionclient.sp_set_url(bindings.sessionServiceLocation);

SubmissionEnt.SubmissionClient submissionclient = new SubmissionEnt.SubmissionClient();
submissionclient.sp_set_url(bindings.submissionServiceLocation);
```

```
QueryEnt.QueryClient queryclient = new QueryEnt.QueryClient();
queryclient.sp_set_url(bindings.queryServiceLocation);
```

Java sample

The following Java/AXIS sample code shows how to retrieve the entry points to use with the Web services objects.

```
ISessionProxy sessionproxy = new ISessionProxy();
SP_BindingObject bindings = sessionproxy.sp_get_bindings("username");
sessionproxy.sp_set_url(bindings.getSessionServiceLocation());


ISubmissionProxy submissionproxy = new ISubmissionProxy();
submissionproxy.sp_set_url(bindings.getSubmissionServiceLocation());


IQueryProxy queryproxy = new IQueryProxy();
queryproxy.sp_set_url(bindings.getQueryServiceLocation());
```

sp_get_session_information

Retrieves the logged user's data.

Parameters name	Type	Description
/	/	/
Returns	**Type**	**Description**
SP_SessionInformation	SP_SessionInformation	Contains the data about the logged user

sp_login

Opens an authenticated session on the server.

Parameters name	Type	Description
spUsername	string	Client's username
spPassword	string	Client's password
Returns	**Type**	**Description**
SP_LoginResult	SP_LoginResult	Contains SessionID
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	An error has occured while calling remote server

C# sample

The following C# sample code shows how to create a session to use with the Web services objects.

```
SessionEnt.SessionClient sessionclient = new SessionEnt.SessionClient();
SessionEnt.SP_BindingObject bindings = sessionclient.sp_get_bindings("username");
sessionclient.sp_set_url(bindings.sessionServiceLocation);

SessionEnt.SP_LoginResult loginresult = sessionclient.sp_login("username", "password");

SubmissionEnt.SubmissionClient submissionclient = new SubmissionEnt.SubmissionClient();
submissionclient.sp_set_url(bindings.submissionServiceLocation);

SubmissionEnt.SP_SessionHeader ssheader = new SubmissionEnt.SP_SessionHeader();

ssheader.spSessionID = loginresult.SessionID;

submissionclient.sp_set_session_header(ssheader);

QueryEnt.QueryClient queryclient = new QueryEnt.QueryClient();
queryclient.sp_set_url(bindings.queryServiceLocation);

QueryEnt.SP_SessionHeader qsheader = new QueryEnt.SP_SessionHeader();

qsheader.spSessionID = loginresult.SessionID;

queryclient.sp_set_session_header(qsheader);

// Start to code here
// ...

sessionclient.sp_logout();
```

Java sample

The following Java/AXIS sample code shows how to create a session to use with the Web services objects.

```
ISessionProxy sessionproxy = new ISessionProxy();
SP_BindingObject bindings = sessionproxy.sp_get_bindings("username");
sessionproxy.sp_set_url(bindings.getSessionServiceLocation());
SP_LoginResult loginresult = sessionproxy.sp_login("username", "password");


ISubmissionProxy submissionproxy = new ISubmissionProxy();
submissionproxy.sp_set_url(bindings.getSubmissionServiceLocation());
SP_SessionHeader spSessionHeader = new SP_SessionHeader();// SP_SessionHeader from submission package
spSessionHeader.setSpSessionID(loginresult.getSessionID());
submissionproxy.sp_set_session_header(spSessionHeader);


IQueryProxy queryproxy = new IQueryProxy();
queryproxy.sp_set_url(bindings.getQueryServiceLocation());


SP_SessionHeader spSessionHeader = new SP_SessionHeader();// SP_SessionHeader from query package
spSessionHeader.setSpSessionID(loginresult.getSessionID());
queryproxy.sp_set_session_header(spSessionHeader);
// Start to code here
// ...
sessionproxy.sp_logout();
```

sp_logout

Closes an authenticated session on the server.

Parameters name	Type	Description
/	/	/
Returns	**Type**	**Description**
/	/	/
Exceptions	**Type**	**Description**
	`FaultException`	`An error has occured while calling remote server`

sp_set_url

Set the service URL for the SP_SessionService.

Parameters name	Type	Description
`spURL`	String	URL returned by the call to sp_get_bindings
Returns	**Type**	**Description**
/	/	/
Exceptions	**Type**	**Description**
	`FaultException`	`An error has occured while calling remote server`

Submission service

sp_upload_file

Uploads some data to the server for later availability.

Parameters name	Type	Description
`spFileContent`	Array of bytes	File to be uploaded
`spFileName`	String	The logical file name (without the path)
Returns	**Type**	**Description**

	SP_File	Link to the uploaded file. This object must be stored for future use of the file

Exceptions	Type	Description
	FaultException	Byte array to large, max is 64KB
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set
	FaultException	An error has occured while calling remote server
	FaultException	Destination file is missing

C# sample

The following C# sample code shows how to upload a text file which size is less then 64KB on the server. If the filesize is bigger then 64KB, sp_append_file should be used.

```
SP_JobParameters jobparam = new SP_JobParameters();
jobparam.spLetterType = SP_LetterTypeEnum.SP_GREEN_LETTER;
jobparam.spColorParameter = SP_ColorEnum.SP_BLACK_AND_WHITE;
jobparam.spRectoParameter = SP_RectoEnum.SP_RECTO;
jobparam.spEnveloppeParameter = SP_EnveloppeEnum.SP_DL_ENVELOPPE_THIRD_A4;
SP_PostalAddress adr = new SP_PostalAddress();

adr.spFirstName = "destinationFirstName";
adr.spLastName = "destinationLastName";
adr.spCompanyName = "destinationCompany";
adr.spFirstLine = "destinationAddress1";
adr.spSecondLine = "destinationAddress2";
adr.spPostalCode = 12345;
adr.spCityName = "destinationCity";

jobparam.spRecipient = adr;

SP_File file = new SP_File();
file.spName = "file.txt";
string filepath = @"D:\data\file.txt";
file.spContent = System.IO.File.ReadAllBytes(filepath);
SP_File filereturn = submissionclient.sp_upload_file(buffer, file.spName);

SP_PreviewResult pr = submissionclient.sp_preview_letter(filereturn, jobparam,
SubmissionReference.SP_FileStorageModeEnum.SP_UPLOADED);
```

Java sample

The following Java/AXIS sample code shows how to upload a text file on the server.

```
SP_JobParameters jobparam = new SP_JobParameters();
jobparam.setSpLetterType(SP_LetterTypeEnum.SP_GREEN_LETTER);
jobparam.setSpColorParameter(SP_ColorEnum.SP_BLACK_AND_WHITE);
jobparam.setSpRectoParameter(SP_RectoEnum.SP_RECTO);
jobparam.setSpEnveloppeParameter(SP_EnveloppeEnum.SP_DL_ENVELOPPE_THIRD_A4);
SP_PostalAddress adrdestination = new SP_PostalAddress();
adrdestination.setSpFirstLine("destinationFirstName");
adrdestination.setSpLastName("destinationLastName");
adrdestination.setSpCompanyName("destinationCompany");
adrdestination.setSpFirstLine("destinationAdsress1");
adrdestination.setSpSecondLine("destinationAdsress2");
adrdestination.setSpPostalCode(12345);
adrdestination.setSpCityName("destinationCity");
jobparam.setSpRecipient(adr);
SP_File file = new SP_File();
file.setSpName("file.txt");
File filetxt = new File("D:\\data\\file.txt");
byte[] bFile = new byte[(int) filetxt.length()];
java.io.FileInputStream fileOuputStream = new FileInputStream(filetxt);
fileOuputStream.read(bFile);
fileOuputStream.close();
file.setSpContent(bFile);
SP_File filereturn = submissionproxy.sp_upload_file(file.getSpContent(), file.getSpName());
SP_PreviewResult pr =  submissionproxy.sp_preview_letter(filereturn, jobparam,
SP_FileStorageModeEnum.SP_UPLOADED);
```

sp_append_file

Uploads some large data on the server.

Parameters name	Type	Description
`spFileContent`	Array of bytes	File to be uploaded
`spDestinationFile`	SP_File	Destination file where the file will be appended
Returns	**Type**	**Description**
	SP_File	Link to the uploaded file. This object must be stored for future use (or append) of the file
Exceptions	**Type**	**Description**
	`FaultException`	`Please set URL first`
	`FaultException`	`Wrong binding id`
	`FaultException`	`Session header is not set!`
	`FaultException`	`Destination file is missing`
	`FaultException`	`Wrong session header!`
	`FaultException`	`Wrong spFile id`

C# sample

The following C# sample code shows how to upload a text file which size is less then 64KB on the server. If the filesize is bigger then 64KB, sp_append_file should be used.

```
SP_File filetxt = new SP_File();
file.spName = "file.txt";
//path of a file (use your own path)
string filepath = @"D\data\file.txt";
filetxt.spContent = System.IO.File.ReadAllBytes(filepath);
SP_File filereturn = new SP_File();

//example of file bigger then 64KB (using append function)

if (filetxt.spContent.Length > 65536)
{
    Stream stream = new FileStream(filepath, FileMode.Open);
    byte[] buffer = new byte[64 * 1024];
    while (true)
    {
        int space = 64 * 1024, read, offset = 0;
        while (space > 0 && (read = stream.Read(buffer, offset, space)) > 0)
        {
            space -= read;
            offset += read;
        }
        if (space != 0)
        { // EOF - final
            if (offset != 0)
            { // something to send
                Array.Resize(ref buffer, offset);
                filereturn = submissionclient.sp_append_file(buffer, newfile);
            }
            break;
        }
        else
        { // full buffer
          filereturn = submissionclient.sp_upload_file(buffer, filetxt.spName);
        }
    }
}
//else - file smaller than 64KB
else
{
   filereturn = submissionclient.sp_upload_file(filetxt.spContent, filetxt.spName);
}


SP_PreviewResult pr = submissionclient.sp_preview_letter(filereturn, jobparam,
SubmissionReference.SP_FileStorageModeEnum.SP_UPLOADED);
```

Java sample

The following Java/AXIS sample code shows how to upload a text file which size is less then 64KB on the server. If the filesize is bigger then 64KB, sp_append_file should be used.

```
SP_File uploadedFile = null;
FileInputStream stream = new FileInputStream("D:\\data\\file.txt");
         if( stream.available() > 64*1024)
         {
         while( stream.available() > 0 )
         {
             byte[] data = new byte[(int)java.lang.Math.min(stream.available(),64*1024)];
             stream.read(data, 0, data.length);
```

```
            if( uploadedFile == null )
                uploadedFile = submissionproxy.sp_upload_file(data, file.getSpName());
            else
                uploadedFile = submissionproxy.sp_append_file(data, uploadedFile);
        }
        }
        else
        {
            byte[] data = new byte[(int)java.lang.Math.min(stream.available(),64*1024)];
            uploadedFile = submissionproxy.sp_upload_file(data, file.getSpName());
        }
SP_PreviewResult previewResult = submissionproxy.sp_preview_letter(uploadedFile, jobparam,
SP_FileStorageModeEnum.SP_UPLOADED);
        stream.close();
```

sp_submit_letter

Submits a document with the given printing parameters.

Parameters name	Type	Description
spDocumentToPrint	SP_File	File containing the letter to print
spPrintingParameters	SP_JobParameters	Printing parameters (including recipient address).
Returns	**Type**	**Description**
	SP_JobResult	Status and ServicePostalID
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Wrong session header
	FaultException	Session header is not set!
	FaultException	Not supported file extension.
	FaultException	Cannot be converted
	FaultException	Sender's address is not set
	FaultException	Destination address is not set

C# sample

The following C# sample code shows how to upload a text file which size is less then 64KB on the server. If the filesize is bigger then 64KB, sp_append_file should be used.

```
SP_File filetxt = new SP_File();
```

```
file.spName = "file.txt";
//path of a file (use your own path)
string filepath = @"D:\data\file.txt";
filetxt.spContent = System.IO.File.ReadAllBytes(filepath);
SP_File filereturn = new SP_File();
SP_JobResult res = submissionclient.sp_submit_letter(fileDocx, jobparam);


SP_PreviewResult pr = submissionclient.sp_preview_letter(filereturn, jobparam,
SubmissionReference.SP_FileStorageModeEnum.SP_UPLOADED);
```

Java sample

The following Java/AXIS sample code shows how to upload a text file which size is less then 64KB on the server. If the filesize is bigger then 64KB, sp_append_file should be used.

```
SP_File file = new SP_File();
file.setSpName("file.txt");
File filetxt = new File("D:\\data\\file.txt");
byte[] bFile = new byte[(int) filetxt.length()];
java.io.FileInputStream fileOuputStream = new FileInputStream(filetxt);
fileOuputStream.read(bFile);
fileOuputStream.close();
file.setSpContent(bFile);
SP_JobResult jobresult =  submissionproxy.sp_submit_letter(file, jobparam);
```

sp_preview_letter

Previews a document with the given printing parameters. Returns a link to a PDF file or the PDF file itself (depending on the spReturnMode parameter).

Parameters name	Type	Description
spDocumentToPrint	SP_File	File containing the letter to print
spPrintingParameters	SP_JobParameters	Printing parameters (including recipient address).
spReturnMode	SP_FIleStorageMode Enum	Indicates how to return the PDF file (inside the SP_File or as an URL)
Returns	**Type**	**Description**
	SP_PreviewResult	Function returns : Preview status. Total cost of the printing job A ServicePostalID is provided to allow the submission of the previewed job
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Wrong session header
	FaultException	Session header is not set!

	FaultException	Not supported file extension.
	FaultException	Cannot be converted
	FaultException	Sender's address is not set
	FaultException	Destination address is not set

sp_preview_mailing

Previews a document from a mailing job with the given printing parameters. Returns a link to a PDF file or the PDF file itself (depending on the spReturnMode parameter).. The preview will concern only the nth letter whom index is provided (1 for the first letter, and so on).

The function will return the number of letters that the mailing will produce. Even if only one document is returned by the call to this function.

Parameters name	Type	Description
spDocumentToPrint	SP_File	File containing the letter to print
spFieldValuesFile	SP_File	File containing the fields value in a CSV format to be merged with the letter to print.
spPrintingParameters	SP_JobParameters	Printing parameters. No recipient address required.
spReturnMode	SP_FIleStorageMode Enum	Indicates how to return the PDF file (inside the SP_File or as an URL)
spIndex	Numeric	Index of the letter to be returned
Returns	**Type**	**Description**
	SP_PreviewMailingR esult	Function returns : Execution status. Number of documents to be produce in the mailing. PDF File for the specific given index. Total cost of the printing job. A ServicePostalID is provided (which is the same for all the letters within the mailing job) to allow the submission of the previewed job
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Filename is missing
	FaultException	File content is missing

	FaultException	Wrong file extension, should be docx!
	FaultException	Wrong file extension, should be csv!
	FaultException	Destination: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Destination: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!
	FaultException	Sender: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Sender: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!
	FaultException	Wrong File ID of template file!
	FaultException	Please provide file content!
	FaultException	Wrong File ID of content file!
	FaultException	Please select bigger enveloppe
	FaultException	Index out of range!

C# sample

The following C# sample code shows how to use mailing functionality.

```
SP_File fileDocx = new SP_File();
fileDocx.spName = "file.docx";
fileDocx.spContent = File.ReadAllBytes(@"D:\data\file.docx");
SP_File fileCSV = new SP_File();
fileCSV.spName = "file.csv";
fileCSV.spContent = File.ReadAllBytes(@"D:\data\file.csv");
SubmissionReference.SP_PreviewMailingResult pmr = submissionclient.sp_preview_mailing(fileDocx, fileCSV,
jobparam, SubmissionReference.SP_FileStorageModeEnum.SP_UPLOADED, 1);
SP_MailingResult resMail = submissionclient.sp_validate_mailing(pmr.spServicePostalID);
```

Java sample

The following Java/AXIS sample code shows how to use mailing functionality.

```
SP_File filedocx = new SP_File();
```

```
filedocx.setSpName("file.docx");
File filesend = new File("C:\\data\\file.docx");
byte[] bFile = new byte[(int) filesend.length()];
java.io.FileInputStream fileOuputStream = new FileInputStream(filesend);
fileOuputStream.read(bFile);
fileOuputStream.close();
file.setSpContent(bFile);
SP_File csvfile = new SP_File();
csvfile.setSpName("file.csv");
File csvfilesend = new File("D:\\data\\file.csv");
byte[] bFileCSV = new byte[(int) csvfilesend.length()];
java.io.FileInputStream fileInputStream = new FileInputStream(filesend);
fileInputStream.read(bFileCSV);
fileInputStream.close();
csvfile.setSpContent(bFileCSV);
SP_PreviewMailingResult previewmailigresult =  submissionproxy.sp_preview_mailing(filedocx, csvfile,
jobparam, SP_FileStorageModeEnum.SP_UPLOADED, 1);
SP_MailingResult mailingresult =
submissionproxy.sp_validate_mailing(previewmailigresult.getSpServicePostalID());
```

sp_preview_mailing_pdf

Previews a document from a mailing job with the given printing parameters. The caller is responsible for building the mailing. The whole letters made by the caller will be transmitted through a single pdf file. All the letters must have the same number of pages. The function will cut the pdf file into letters.

The function returns a link to a PDF file or the PDF file itself (depending on the spReturnMode parameter).. The preview will concern only the nth letter whom index is provided (1 for the first letter, and so on).

The function will return the number of letters that the mailing will produce. Even if only one document is returned by the call to this function.

Parameters name	Type	Description
spDocumentToPrint	SP_File	File containing all the letters to print
spFieldValuesFile	SP_File	File containing the fields value in a CSV format to build the header pages
spPrintingParameters	SP_JobParameters	Printing parameters. No recipient address required.
spReturnMode	SP_FIleStorageModeEnum	Indicates how to return the PDF file (inside the SP_File or as an URL)
spIndex	Numeric	Index of the letter to be returned
spNumberOfPages	Numeric	Number of pages of one single letter (all the letters must have the same length)
Returns	**Type**	**Description**
	SP_PreviewMailingResult	Function returns : Execution status. Number of documents to be produce in the mailing. PDF File for the specific given index. Total cost of the printing job. A ServicePostalID is provided (which is the same for all the letters within the mailing job) to allow the submission of the previewed job

Exceptions	Type	Description
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Filename is missing
	FaultException	File content is missing
	FaultException	Wrong file extension, should be pdf!
	FaultException	Wrong file extension, should be csv!
	FaultException	Destination: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Destination: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!
	FaultException	Sender: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Sender: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!
	FaultException	Wrong File ID of template file!
	FaultException	Please provide file content!
	FaultException	Wrong File ID of content file!
	FaultException	Number of pages in the document doesn't match the number of addresses!
	FaultException	Please select bigger enveloppe
	FaultException	Index out of range!

C# sample

The following C# sample code shows how to use pdf mailing functionality.

```
//Create job parameters
SP_JobParameters jobparam = new SP_JobParameters();
jobparam.spLetterType = SP_LetterTypeEnum.SP_ECONOMIC_LETTER;
jobparam.spColorParameter = SP_ColorEnum.SP_COLOR;
jobparam.spRectoParameter = SP_RectoEnum.SP_RECTO_VERSO;
jobparam.spEnveloppeParameter = SP_EnveloppeEnum.SP_DL_ENVELOPPE_THIRD_A4;
jobparam.spHeaderPage = SP_YesNoEnum.SP_YES;

SP_File fileDocx = new SP_File();
fileDocx.spName = "Courriers.pdf";

byte[] tempBuffer = new byte[1024 * 1024];
string filepath = @"D:\data\Courriers.pdf";
//tempBuffer = System.IO.File.ReadAllBytes(filepath);
fileDocx.spContent = System.IO.File.ReadAllBytes(filepath);
//fileDocx = submissionclient.sp_upload_file(tempBuffer, fileDocx.spName);

SP_File newfile = new SP_File();

//files for publipostage are usually bigger then 64KB, so we provide example with
appending content

if (fileDocx.spContent.Length > 65536)
{
    Stream stream = new FileStream(filepath, FileMode.Open);
    byte[] buffer = new byte[64 * 1024];
    while (true)
    {
        int space = 64 * 1024, read, offset = 0;
        while (space > 0 && (read = stream.Read(buffer, offset, space)) > 0)
        {
            space -= read;
            offset += read;
        }
        if (space != 0)
        { // EOF - final
            if (offset != 0)
            { // something to send
                Array.Resize(ref buffer, offset);
                newfile = submissionclient.sp_append_file(buffer, newfile);
            }
            break;
        }
        else
        { // full buffer

            newfile = submissionclient.sp_upload_file(buffer, fileDocx.spName);
        }
    }
}
//else - file smaller than 64KB
else
{
    newfile = submissionclient.sp_upload_file(fileDocx.spContent, fileDocx.spName);
}

SP_File fileCSV = new SP_File();
fileCSV.spName = "liste-destinataires.csv ";
filepath = @"C:\data\liste-destinataires.csv";
tempBuffer = System.IO.File.ReadAllBytes(filepath);
fileCSV = submissionclient.sp_upload_file(tempBuffer, fileCSV.spName);
```

```
            Submission.SP_PreviewMailingResult pmr =
            submissionclient.sp_preview_mailing_pdf(fileDocx, fileCSV, jobparam,
            Submission.SP_FileStorageModeEnum.SP_UPLOADED, 1,4);
            SP_MailingResult spmailing =
            submissionclient.sp_validate_mailing(pmr.spServicePostalID);
```

Java sample

The following Java/AXIS sample code shows how to use mailing functionality.

```
SP_File filePDF = new SP_File();
filePDF.setSpName("file.docx");
File filesend = new File("D:\\data\\Courriers.pdf");
byte[] bFile = new byte[(int) filesend.length()];
java.io.FileInputStream fileOuputStream = new FileInputStream(filesend);
fileOuputStream.read(bFile);
fileOuputStream.close();
file.setSpContent(bFile);
SP_File csvfile = new SP_File();
csvfile.setSpName("file.csv");
File csvfilesend = new File("D:\\data\\liste-destinataire.csv");
byte[] bFileCSV = new byte[(int) csvfilesend.length()];
java.io.FileInputStream fileInputStream = new FileInputStream(filesend);
fileInputStream.read(bFileCSV);
fileInputStream.close();
csvfile.setSpContent(bFileCSV);
SP_PreviewMailingResult previewmailigresult =  submissionproxy.sp_preview_mailing_pdf(filePDF, csvfile,
jobparam, SP_FileStorageModeEnum.SP_UPLOADED, 1, 4);
            SP_MailingResult mailingresult =
            submissionproxy.sp_validate_mailing(previewmailigresult.getSpServicePostalID());
```

~~C# sample~~

~~The following C# sample code shows how to use mailing functionality.~~

```
~~SP_File fileDocx = new SP_File();~~
~~fileDocx.spName = "file.docx";~~
~~fileDocx.spContent = File.ReadAllBytes(@"D:\data\file.docx");~~
~~SP_File fileCSV = new SP_File();~~
~~fileCSV.spName = "file.csv";~~
~~fileCSV.spContent = File.ReadAllBytes(@"D:\data\file.csv");~~
~~SubmissionReference.SP_PreviewMailingResult pmr = submissionclient.sp_preview_mailing(fileDocx, fileCSV,~~
~~jobparam, SubmissionReference.SP_FileStorageModeEnum.SP_UPLOADED, 1);~~
~~SP_MailingResult resMail = submissionclient.sp_validate_mailing(pmr.spServicePostalID);~~
```

~~Java sample~~

~~The following Java/AXIS sample code shows how to use mailing functionality.~~

```
~~SP_File filedocx = new SP_File();~~
~~filedocx.setSpName("file.docx");~~
~~File filesend = new File("C:\\data\\file.docx");~~
~~byte[] bFile = new byte[(int) filesend.length()];~~
```

```
java.io.FileInputStream fileOuputStream = new FileInputStream(filesend);
fileOuputStream.read(bFile);
fileOuputStream.close();
file.setSpContent(bFile);
SP_File csvfile = new SP_File();
csvfile.setSpName("file.csv");
File csvfilesend = new File("D:\\data\\file.csv");
byte[] bFileCSV = new byte[(int) csvfilesend.length()];
java.io.FileInputStream fileInputStream = new FileInputStream(filesend);
fileInputStream.read(bFileCSV);
fileInputStream.close();
csvfile.setSpContent(bFileCSV);
SP_PreviewMailingResult previewmailigresult =  submissionproxy.sp_preview_mailing(filedocx, csvfile,
jobparam, SP_FileStorageModeEnum.SP_UPLOADED, 1);
SP_MailingResult mailingresult =
submissionproxy.sp_validate_mailing(previewmailigresult.getSpServicePostalID());
```

sp_submit_mailing

Submits a mailing job with the given printing parameters. Optionally one single letter could be produced.

The function will return the number of letters that the mailing will produce and the total cost for the mailing. If a single letter is asked to print, the total cost will give the amount for this letter only.

Parameters name	Type	Description
spDocumentToPrint	SP_File	File containing the letter to print
spFieldValuesFile	SP_File	File containing the fields value in a CSV format to be merged with the letter to print.
spPrintingParameters	SP_JobParameters	Printing parameters. No recipient address required.
spIndex	Numeric	Index of the letter to be produced. 0 means all the letters will be printed
Returns	**Type**	**Description**
	SP_MailingResult	Function returns : Execution status. Number of documents to be produced in the mailing. ServicePostalID. Total cost of the printing job
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	File content is missing
	FaultException	Filename is missing

	FaultException	Wrong file extension, should be docx!
	FaultException	Wrong file extension, should be csv!
	FaultException	Destination: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Destination: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!
	FaultException	Sender: mandatory fields missing.Please make sure that RecAddressLine1,RecZipPostalCode,RecCity have values !
	FaultException	Sender: mandatory fields missing.Please make sure that at least one of the fields is filled: RecLastName, RecFirstName,RecCompany!

sp_validate_letter

Submits a previously previewed letter.

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job.
Returns	**Type**	**Description**
	SP_JobResult	Function returns : Execution status. Total cost of the printing job
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header

sp_validate_mailing

Submits a previously previewed mailing. The call to this function starts the whole mailing task.

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job.
Returns	**Type**	**Description**
	SP_MailingResult	Function returns : Execution status. Number of documents to be produce in the mailing. Total cost of the printing job
Exceptions	**Type**	**Description**
	`FaultException`	`Please set URL first`
	`FaultException`	`Wrong binding id`
	`FaultException`	`Session header is not set!`
	`FaultException`	`Wrong session header`

sp_set_url

Set the service URL for the SP_SubmissionService.

Parameters name	Type	Description
spURL	String	URL returned by the call to sp_get_bindings
Returns	**Type**	**Description**
Exceptions	**Type**	**Description**

sp_cancel_job

Cancels a previously submitted or previewed job. If the letter(s) have already been printed, the cancellation is not possible. A failure status is returned.

In case of a mailing task, an index must be provided to cancel one single letter. If an index of 0 is provided, the whole mailing task is cancelled. If at least one letter from the mailing have been printed, it is not possible to cancel the whole mailing.

In case of a mailing task, if at least one letter has been printed, the whole process cannot be cancelled using 0 value for spIndex. If this happens, the `sp_cancel_job` function will return a failure value and the user will have to cancel letter by letter in a loop to remove the letters who have not be printed yet from the job queue. The returned status will tell if the cancellation has been done or not for each letter.

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the job who must be cancelled
spIndex	Numeric	Index of the letter to be cancelled. (in case of a mailing job). Parameter is optional
Returns	**Type**	**Description**
	SP_SuccessFailureEnum	Success or failure
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Not your file!
	Exception	Wrong spServicePostalID
	IndexOutOfRange	Index out of range

sp_estimate_price

Returns the rate of a letter from an SP_JobParameters object and the expected number of pages of the document. The function returns also the price of the stamp.

Parameters name	Type	Description
spLetterOptions	SP_JobParameters	Printing options of the requested letter
spNbOfPages	Numeric	Number of the page of the document. The header page with address shouldn't be taken into account. It will be automatically added if the option is chosen.
Returns	**Type**	**Description**
	SP_QueryRateResult	Rate and price stamp of the letter
Exceptions	**Type**	**Description**
	FaultException	Please set URL first

	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Number of pages out of range

C# sample

The following C# sample code shows how to use this functionality.

```
SP_JobParameters jobparam = new SP_JobParameters();
jobparam.spLetterType = SP_LetterTypeEnum.SP_REGISTERED_WITH_PROOF;
jobparam.spColorParameter = SP_ColorEnum.SP_BLACK_AND_WHITE;
jobparam.spRectoParameter = SP_RectoEnum.SP_RECTO;
jobparam.spEnveloppeParameter = SP_EnveloppeEnum.SP_DL_ENVELOPPE_THIRD_A4;


//the same address object will be set for both sender and recipient just for example

SP_PostalAddress adr = new SP_PostalAddress();

adr.spFirstName = "Test";
adr.spLastName = "User";
adr.spCompanyName = "Test Company";
adr.spFirstLine = "Test Address Line 1";
adr.spSecondLine = "Test Address Line 2";
adr.spPostalCode = 12345;
adr.spCityName = "Test City";

jobparam.spSender = adr;
jobparam.spRecipient = adr;


SP_QueryRateResult result = submissionclient.sp_estimate_price(jobparam, 4);

decimal amoutPoste = result.spStampPrice;
decimal amountService = result.spServicePrice;
decimal totalWeight = result.spWeight;

decimal totalPrice = amountService + amoutPoste;

foreach (var item in result.spServiceCodeList)
{
       Console.WriteLine(item.spCode);
       Console.WriteLine(item.spPrice);
       Console.WriteLine(item.spQuantity);
       Console.WriteLine(item.spVAT);
}
```

Java sample

The following Java/AXIS sample code shows how to use this functionality.

```
SP_QueryRateResult qr = sub.sp_estimate_price(jobparam, 4);
       BigDecimal amountService = qr.getSpServicePrice();
       BigDecimal amountPost = qr.getSpStampPrice();
       BigDecimal weight = qr.getSpWeight();
       BigDecimal totalPriceVatFree = new BigDecimal(0);
       totalPriceVatFree.add(amountPost);
       totalPriceVatFree.add(amountService);
       SP_ServiceCode [] spServiceCode =  qr.getSpServiceCodeList();
       for(int i = 0;i<spServiceCode.length;i++)
       {
         System.out.println(spServiceCode[i].getSpCode());
         System.out.println(spServiceCode[i].getSpPrice().toString());
         System.out.println(spServiceCode[i].getSpQuantity().toString());
         System.out.println(spServiceCode[i].getSpVAT().toString());

       }

       System.out.println(weight);
       System.out.println(totalPriceVatFree.toString());
```

Query service

sp_query_status

Ask the status of a specified letter (using the ServicePostalID of the task): printed, given to the Postal Office, received, distributed.

In case of a mailing task, an index must be provided to query the status of one single letter

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job. Used for querying information about this job
spIndex	Numeric	Index of the letter to be queried. (in case of a mailing job)
Returns	**Type**	**Description**
	SP_DocumentStatus Result	Status of the queried document
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Not your file!

	FaultException	Seems like the letter is part of mailing job, please provide valid spIndex parameter
	FaultException	Index out of range!
	FaultException	spIndex cannot be 0!

sp_query_document

Retrieve a document from a specified task (using the ServicePostalID).

In case of a mailing task, an index must be provided to query the document of one single letter

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job.
spIndex	Numeric	Index of the letter to be queried. (in case of a mailing job)
spDocumentType	SP_DocumentTypeEnum	Type of the document to be retrieved (proof of depository, proof of delivery, content of the letter as PDF)
spReturnMode	SP_FIleStorageModeEnum	Indicates how to return the document (inside the SP_File or as an URL)
Returns	**Type**	**Description**
	SP_QueryFileResult	Requested file
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Index out of range

sp_query_letter_cost

Ask the cost of a specified letter (using the spServicePostalID of the task).

In case of a mailing task, an index must be provided to query the cost of one single letter. If an index of 0 is provided, the total cost of the mailing is returned.

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job.
spIndex	Numeric	Index of the letter to be queried. (in case of a mailing job)
Returns	**Type**	**Description**
		Cost of the letter in euros
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!
	FaultException	Wrong session header
	FaultException	Index out of range
	FaultException	Wrong spServicePostalID

sp_query_letter_rate

Returns the rate of a letter from its ServicePostalID. The function returns also the price of the stamp. To call this function, the letter must have been sent to ServicePostal server through a preview or submit functions.

Parameters name	Type	Description
spServicePostalID	String	Internal reference of the submitted job.
spIndex	Numeric	Index of the letter to be queried. (in case of a mailing job)
Returns	**Type**	**Description**
	SP_QueryRateResult	Rate and price stamp of the letter
Exceptions	**Type**	**Description**
	FaultException	Please set URL first
	FaultException	Wrong binding id
	FaultException	Session header is not set!

	FaultException	Wrong session header
	FaultException	Index out of range
	FaultException	Wrong spServicePostalID

C# sample

The following C# sample code shows how to use this functionality.

```
string spServicePostalID = "123456";
int index = 1;

SP_QueryRateResult result = queryclient.sp_query_letter_rate(spServicePostalID, index);

decimal amoutPoste = result.spStampPrice;
decimal amountService = result.spServicePrice;
decimal totalWeight = result.spWeight;

decimal totalPrice = amountService + amoutPoste;

foreach (var item in result.spServiceCodeList)
{
        Console.WriteLine(item.spCode);
        Console.WriteLine(item.spPrice);
        Console.WriteLine(item.spQuantity);
        Console.WriteLine(item.spVAT);
}
```

Java sample

The following Java/AXIS sample code shows how to use this functionality.

```
string spServicePostalID = "123456";
int index = 1;
IQueryProxy query = new IQueryProxy();
               query.sp_set_url(bindings.getQueryServiceLocation());
               //SP_SessionHeader spSessionHeader = new SP_SessionHeader();
               com.servicepostal.doc.Query.SP_SessionHeader spSessionHeaderQuery = new
com.servicepostal.doc.Query.SP_SessionHeader();
               spSessionHeaderQuery.setSpSessionID(loginresult.getSessionID());
               query.sp_set_session_header(spSessionHeaderQuery);
               com.servicepostal.doc.Query.SP_QueryRateResult queryresult =
query.sp_query_letter_rate spServicePostalID, index );

               BigDecimal amountServiceQuery = queryresult.getSpServicePrice();
              BigDecimal amountPostQuery = queryresult.getSpStampPrice();
              BigDecimal weightQuery = queryresult.getSpWeight();
              BigDecimal totalPriceVatFreeQuery = new BigDecimal(0);
              totalPriceVatFree.add(amountServiceQuery);
              totalPriceVatFree.add(amountServiceQuery);
```

```
           com.servicepostal.doc.Query.SP_ServiceCode [] spServiceCodeQuery = 
queryresult.getSpServiceCodeList();
           for(int i = 0;i<spServiceCodeQuery.length;i++)
           {
             System.out.println(spServiceCodeQuery[i].getSpCode());
             System.out.println(spServiceCodeQuery[i].getSpPrice().toString());
             System.out.println(spServiceCodeQuery[i].getSpQuantity().toString());
             System.out.println(spServiceCodeQuery[i].getSpVAT().toString());

           }

           System.out.println(weightQuery);

           System.out.println(totalPriceVatFreeQuery.toString());
```

sp_set_url

Set the service URL for the SP_QueryService.

Parameters name	Type	Description
spURL	String	URL returned by the call to sp_get_bindings
Returns	**Type**	**Description**
Exceptions	**Type**	**Description**

Mandatory Fields for mailing

Here is a list of fields needed for a mailing and that must be given in the data file. If the column of a mandatory field is missing, the mailing process will fail.

The order of the fields in the data file is not important. The fields's names are case sensitive. sensitive and should be uppercase. In the mailing document, the field names should be surrounded by two '#' (for example : ##NOM_DESTINATAIRE##).

Recipient's fields	Type	Maxlength (characters)	Mandatory	Description
NOM_DESTINATAIRE	String	32	Yes	Last name
PRENOM_DESTINATAIRE	String	32	Yes	First name
NOM_SOCIETE_DESTINATAIRE	String	32	No	Company name Default is empty
ADRESSE_LIGNE1_DESTINATAIRE	String	32	Yes	Line 1 of the address
ADRESSE_LIGNE2_DESTINATAIRE	String	32	No	Line 2 of the address Default is empty
CODE_POSTAL_DESTINATAIRE	Numeric	5	Yes	Zip postal code
VILLE_DESTINATAIRE	String	26	Yes	City name
PAYS_DESTINATAIRE	String	32	No	Country Default is France
Sender's fields	**Type**		**Mandatory**	**Description**
NOM_EXPEDITEUR	String	32	Yes if registered letter	Last name
PRENOM_EXPEDITEUR	String	32	Yes if registered letter	Last name
NOM_SOCIETE_EXPEDITEUR	String	32	No	Company name Default is empty
ADRESSE_LIGNE1_EXPEDITEUR	String	32	Yes if registered letter	Line 1 of the address
ADRESSE_LIGNE2_EXPEDITEUR	String	32	No	Line 2 of the address Default is empty
CODE_POSTAL_EXPEDITEUR	Numeric	5	Yes if registered letter	Zip postal code
VILLE_EXPEDITEUR	String	26	Yes if registered letter	City name
PAYS_EXPEDITEUR	String	32	No	Country Default is France
REFERENCE_CLIENT	String	32	No	A string to be printed on the deposit slip. The meaning is for the client only

Service code list

The object SP_ServiceCode gives the service code of a unitary service provided by Service Postal.

The final price billed by Service Postal will be the sum of the all these unitary services depending on the printing options and the type of letters (registered, ecopli, …) :

Unitary service description	Rate Code	VAT
Timbre Ecopli	TIM-ECO	No
Timbre Lettre prioritaire	TIM-LP	No
Timbre Lettre verte	TIM-LV	No
Timbre Lettre recommandée avec accusé de réception	TIM-LRAR	No
Timbre Lettre recommandée sans accusé de réception	TIM-LR	No
Enveloppe au format C5 (feuilles A4 pliées en deux)	ENV-C5-1F	Yes
Enveloppe au format C4 (feuilles A4 non pliées)	ENV-C4-1F	Yes
Impression première page en noir et blanc	PAGE-1-NB	Yes
Impression première page en couleur	PAGE-1-COUL	Yes
Impression page suivante en noir et blanc recto	PAGE-S-NB	Yes
Impression page suivante en noir et blanc recto/verso	PAGE-S-NB-RV	Yes
Impression page suivante en couleur recto	PAGE-S-COUL	Yes
Impression page suivante en couleur recto/verso	PAGE-S-COUL-RV	Yes
Fabrication d'un recommandé	LR-AR	Yes
Gestion du retour des AR	LR-AR-RETOUR	Yes
Option lettre recommandé internationale	LR-AR-INTER	Yes
Option stockage AR 3 ans	LR-AR-STOCK1	Yes
Option stockage AR 6 ans	LR-AR-STOCK6	Yes
Option stockage AR 10 ans	LR-AR-STOCK10	Yes
Gestion des plis non distribués	PND	Yes

Examples :

- To build and send a registered letter of 4 recto color pages, we will give the following rate codes:

Quantity	Code	VAT
1	TIM-LRAR	No
1	PAGE-1-COUL	Yes
3	PAGE-S-COUL	Yes
1	LR-AR	Yes

- To build and send an ecopli letter of 10 recto/verso B&W pages, we will give the following rate codes:

Quantity	Code	VAT
1	TIM-ECO	No
1	PAGE-1-NB	Yes
9	PAGE-S-NB	Yes
1	ENV-C5-1F	Yes

Remarque : There is no price code for envelope DL (pages folded in three) because its price is included in the first page price.