

# METIS PHP Webservice Client Quick Guide

**priorIT EDV-Dienstleistungen GmbH**

Kohldorferstraße 98,

A-9020 Klagenfurt, Austria, Europe

**First Edition (Jule 2010)**

This edition applies to METIS PHP Webservice Client version 1.0

© Copyright priorIT EDV-Dienstleistungen GmbH 2010. All rights reserved.

# Table of contents

- 1 METIS PHP WEBSERVICE CLIENT AT A GLANCE ..... 4**
- 1.1 GUIDELINES AND RECOMMENDATIONS..... 4
- 1.1.1 *OrderPixelService* ..... 4
- 1.1.2 *SendMessageService*..... 4
- 1.1.3 *CheckAuthService* ..... 4
- 1.1.4 *QualityControlService* ..... 5
- 2 PREREQUISITES..... 6**
- 3 DESIGN..... 7**
- 3.1 API..... 7
- 3.1.1 *lib.logic* ..... 7
- 3.2 EXCEPTIONHANDLING ..... 10
- 3.2.1 *Business Exceptions* ..... 10
- 3.2.2 *MetisClientExceptions* ..... 11
- 3.2.3 *Technical Exceptions* ..... 11
- 3.3 LOGGING..... 12
- 3.4 PIXELDEPOT ..... 12
- 3.4.1 *Overview* ..... 12
- 3.4.2 *Using the PixelDepot* ..... 12
- 3.4.3 *Create customized PixelDepot* ..... 13
- 3.4.4 *Behaviour*..... 13
- 3.5 EXAMPLE ..... 14
- 3.5.1 *Call OrderPixelService*..... 14
- 3.5.2 *Call NewMessageService*..... 15
- 3.5.3 *Call CheckAuthService*..... 16
- 3.5.4 *Call QualityControlService*..... 17

# 1 METIS PHP Webservice Client at a glance

METIS PHP Webservice Client is a tool designed to facilitate the communication between any PHP environment (e.g. a PHP CMS system) and the METIS system of VG WORT.

It provides functionality to

- order pixels (OrderPixelService)
- send messages for new announcements (SendMessageService)
- validate author information (CheckAuthorService)
- receive quality control information (QualityControlService)

## 1.1 Guidelines and recommendations

### 1.1.1 OrderPixelService

METIS PHP Webservice Client offers two ways to call the OrderPixelService.

- 1) Direct call of the webservice: The webservice call returns the response data including the pixels. The caller has to process the responded data himself.
- 2) Calling the webservice using a PixelDepot: This is the recommended way to use this webservice. A depot (in our case a FileDepot) is used to store received pixels temporary in a file. Before the webservice is called the client takes a look into the depot and returns the pixels from there if available. The advantage of this scenario is that in case of server problems there are still pixels available in the depot to use. The standard implementation of this depot (FileDepot) can easily be replaced by any other customized depot (e.g. DatabaseDepot, QueueDepot...). For further information about the PixelDepot see chapter 3.4. PixelDepot.

### 1.1.2 SendMessageService

- Only already stable messages should be sent (which won't be changed anymore). For this reason it is recommended to transmit messages with a delay of some days. This does not affect the royalties because the payout of the previous year happens at the end of September or at the beginning of October.
- The messages should be transmitted during the night (22.00 – 04.00)
- To avoid server overload messages should be transmitted sequentially (not parallel). Ideally there should be a delay of one second before sending another message. Each transaction is handled separately which means that each successful message is stored independent from a previous or following failed message.
- Business errors point to invalid message data. It is senseless to resubmit the same data again. Failed requests due to technical errors should be sent again at a later time.
- A successful servicecall without errors indicates that the message was OK from a technical viewpoint. In a later step the messages are reviewed by the responsible persons of VG WORT. If there are lacks the message is rejected and the responsible person for the account is informed per mail. These messages can be reviewed, deleted, reworked and resent again in the portal.

### 1.1.3 CheckAuthorService

- This service provides functionality to validate an author's information based on cardnumber and surname.
- Before a new message is sent to METIS this check should be performed.

#### 1.1.4 QualityControlService

- The QualityControlService returns quality information about ordered pixels. The statistics are available from the time of activation of the portal. If the minimal access for the current year is not set, the value of the previous year will be taken.
- Each 'qualityControlValues'-element contains the month and year for which the statistics are calculated. Further the count of ordered pixels, the count of started pixels, the count of pixels which reached the minimum access limit and the count of pixels which reached the minimum access limit without a message already created.
- The value "orderedPixelTillToday" is calculated on a daily base, the value "startedPixelTillToday" is based on the value 2 days before.

## 2 Prerequisites

- Running PHP environment
- Minimum PHP Version 5.x
- Installing the METIS PHP Webservice Client needs an agreement on the 'priorIT services GmbH EULA License' located in the license folder of the application.

### 3 Design

Quick Guide shows a short overview of the API, the complete API is documented in “METIS PHP Webservice Client – Documentation and Installation Guide”.

#### 3.1 API

##### 3.1.1 lib.logic

<b>Class OrderPixelService</b> lib.logic	
<b>Function Summary</b>	
OrderPixelResponse	orderPixel(int \$count)
OrderPixelResponse	orderPixelUsingDepot(int \$count, PixelDepot \$pixelDepot)
<b>Function Detail</b>	
<b>orderPixel</b>  public function orderPixel(int \$count)  Input:           - \$count: count of ordered pixels Output:           - OrderPixelResponse: response data of the webservicecall  <b>ReturnCodes:</b>  "ERROR_ORDER_PIXEL_VALIDATION" = 99 Input parameters were not set correctly  "ERROR_ORDER_PIXEL_TECHNICAL" = 100 A technical problem occurred  "ERROR_ORDER_PIXEL_MAX_COUNT_ORDER" = 1 MaxCount of pixels for this order was reached  "ERROR_ORDER_PIXEL_MAX_COUNT_YEAR" = 2 MaxCount of pixels for current year was reached	
<b>orderPixelUsingDepot</b>  public function orderPixelUsingDepot(int \$count, PixelDepot \$pixelDepot)	

Input:           - \$count: count of ordered pixels  
                   - \$pixelDepot: the implementation of a PixelDepot

Output:           - OrderPixelResponse: response data of the webservicecall

**ReturnCodes:**

"ERROR\_ORDER\_PIXEL\_VALIDATION" = 99  
 Input parameters were not set correctly

"ERROR\_ORDER\_PIXEL\_TECHNICAL" = 100  
 A technical problem occurred

"ERROR\_ORDER\_PIXEL\_MAX\_COUNT\_ORDER" = 1  
 MaxCount of pixels for this order was reached

"ERROR\_ORDER\_PIXEL\_MAX\_COUNT\_YEAR" = 2  
 MaxCount of pixels for current year was reached

"ERROR\_ORDER\_PIXEL\_STORE\_TO\_DEPOT" = 3  
 Error while storing pixel to depot

**Class NewMessageService**

lib.logic

**Function Summary**

NewMessageResponse	newMessage (NewMessageRequest \$newMessageRequest)
--------------------	--

**Function Detail**

**newMessage**

public function newMessage (NewMessageRequest \$newMessageRequest)

Input:           - \$newMessageRequest: NewMessageRequest data (text, pixelID, authors, translators, webranges...)

Output:           - NewMessageResponse: OK Flag

**ReturnCodes:**

"NEW\_MESSAGE\_SERVICE\_SUCCESS" = 0  
 Service success

"ERROR\_NEW\_MESSAGE\_TECHNICAL" = 100

A technical problem occurred

"ERROR\_NEW\_MESSAGE\_NO\_PIXEL" = 1

No existing pixel for requested pixelID

"ERROR\_NEW\_MESSAGE\_USER\_PIXEL" = 2

The pixel is assigned to an different user

"ERROR\_NEW\_MESSAGE\_HAS\_ALREADY\_METISMESSAGE" = 3

Message for pixel already exists

"ERROR\_NEW\_MESSAGE\_CARDNUMBER\_NOT\_SURNAME" = 4

The cardNumber does not fit to the name of the user

"ERROR\_NEW\_MESSAGE\_MIN\_TEXT\_LENGTH" = 5

The reported text has to have a minLength of 1800

"ERROR\_NEW\_MESSAGE\_FILE\_IS\_NO\_PDF" = 8

The reported file is no valid PDF file

"ERROR\_NEW\_MESSAGE\_DUPLICANT\_PARTICIPANT" = 9

A participant has been set more than once for this message

## Class CheckAuthService

lib.logic

### Function Summary

CheckAuthorResponse	checkAuthor(\$cardNumber, \$surname)
---------------------	--------------------------------------

### Function Detail

#### checkAuthor

```
public function checkAuthor($cardNumber, $surname)
```

Input:

- \$cardNumber: cardnumber of the author
- \$surname: surname of the author

Output:

- CheckAuthorResponse: response data of the webservicecall

CheckAuthorResponse->valid = {AUTHOR\_VALID, AUTHOR\_INVALID}

**ReturnCodes:**

```
"ERROR_CHECK_AUTHOR_TECHNICAL" = 100
```

A technical problem occurred.

**Class QualityControlService**

```
lib.logic
```

**Function Summary**

QualityControlResponse	qualityControl()
------------------------	------------------

**Function Detail****qualityControl**

```
public function qualityControl()
```

Output: - QualityControlResponse: response data of the webservicecall

**ReturnCodes:**

```
"ERROR_QUALITY_CONTROL_TECHNICAL" = 100
```

A technical problem occurred.

## 3.2 ExceptionHandling

METIS PHP Webservice Client differs between three exception types:

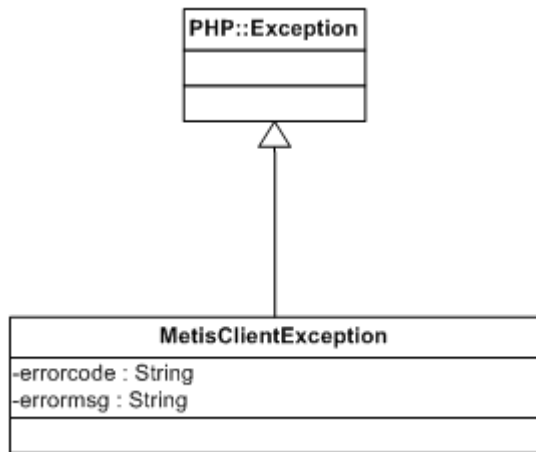
### 3.2.1 Business Exceptions

BusinessExceptions returned from the webservice are mapped to the "errmsg" and "errorcode" fields of the response. These values have to be evaluated before going through the response data. Errorcodes are defined in `lib.common.MetisNamingConstants`

```
$errmsg = $serviceResponse->errmsg;
$errorcode = $serviceResponse->errorcode;
//check if error occurred
if(!isset($errorcode)) {
    // go on ...
} else {
    //HANDLE ERROR HERE
}
```

### 3.2.2 MetisClientExceptions

MetisClientExceptions are predefined technical exceptions occurring in the MetisClient. MetisClientException is a subclass of PHP::Exception and is thrown at runtime. Variables „errorcode“ and „errormsg“ contain additional information about the problem. This exception is thrown e.g. when configuration parameters are not set up correctly.



```

try{

    //call webservice
    $orderPixelResponse = $orderPixelServiceRef->orderPixel($count);

}catch(MetisClientException $exception){

    //handle MetisClientException → get detail-information
    $exception->errorcode;
    $exception->errormsg;
}catch(Exception $exception){

    //handle Exception

}
  
```

### 3.2.3 Technical Exceptions

Any other Exception can optionally be caught in a separate catch block.

```

try{

    //call webservice
    $orderPixelResponse = $orderPixelServiceRef->orderPixel($count);

}catch(MetisClientException $exception){

    //handle MetisClientException → get detail-information
    $exception->errorcode;
    $exception->errormsg;

}catch(Exception $exception){

    //HANDLE EXCEPTION HERE

}
  
```

### 3.3 Logging

METIS PHP Webservice Client provides displaying and logging functionality for errors and further application information. The output will be written to the file configured in the `error_log` config in the `php.ini`

```

; This directive informs PHP of which errors, warnings and notices you would like
; it to take action for.
error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED

; This directive controls whether or not and where PHP will output errors,
; notices and warnings too.
display_errors = On

; Besides displaying errors, PHP can also log errors to locations such as a
; server-specific log, STDERR, or a location specified by the error_log
log_errors = On

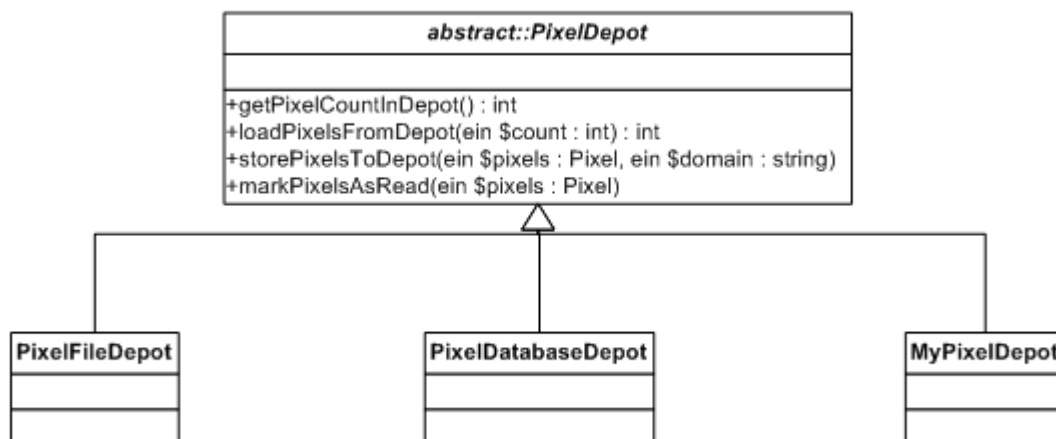
; Log errors to specified file. PHP's default behavior is to leave this value empty
error_log = "C:\xampp\apache\logs\php_error.log"

```

### 3.4 PixelDepot

#### 3.4.1 Overview

PixelDepot is a built-in functionality of the METIS PHP Webservice Client to provide sophisticated client side pixel handling. If a server error occurs there is no possibility to order pixels at this time. The idea of PixelDepot is to store an amount of pixels on the client side to be independent from any server or connection problems. 'PixelFileDepot' is an implementation of the PixelDepot and integrated into the WebserviceClient. An own implementation can be created by extending the abstract class 'PixelDepot' and implementing all abstract Methods.



#### 3.4.2 Using the PixelDepot

To use the `orderPixelService` with a `PixelDepot` following function has to be called:

```
OrderPixelService::orderPixelUsingDepot($count, new PixelFileDepot())
```

For your own implementation:

```
OrderPixelService::orderPixelUsingDepot($count, new MyFileDepot())
```

### 3.4.3 Create customized PixelDepot

To create a customized PixelDepot a class has to be created which extends the abstract class PixelDepot. There are four methods which have to be implemented (for details see 3.1.2 – Class PixelDepot).

For the configuration of the depot the section `additional_depot_properties` in the `orderPixelDepot.ini` file should be used to store all depot configuration properties at one point.

### 3.4.4 Behaviour

The PixelDepot implementation works as follows:

- 1) The method `orderPixelUsingDepot($count, new XXXFileDepot())` of the Class `OrderPixelService` is called
- 2) `getPixelCountInDepot()` is called and the count of pixels in the depot is returned.
- 3) If the ordered count of pixels is greater than the pixel count from the depot the pixels are taken from the depot (by calling `loadPixelsFromDepot()`) and the missing ones are loaded via the webservice. If there are enough pixels available in the depot all pixels are taken from the depot.
- 4) Refill the depot: Using the parameters from the configuration file: `orderPixelDepot.ini` the amount of pixels is calculated which get stored in the depot again. Storing the pixels happens calling the Method `storePixelsToDepot()`. So if the depot contains 40 pixels and the `depotMINSize` is 50 and the `depotMAXSize` is 100 the webservice is called to order 60 pixels. The recommended sizing of both parameters is 100 because of the order count restrictions of the webservice (only 100 pixels per day can be ordered).
- 5) To avoid transactional problems in the end the method `markPixelsAsRead()` is called which sets the state of all read pixels to 'used'. This step could also be implemented when loading the pixels from the depot but if an error occurs after loading them they might get lost. So they are marked as read in the end of the function.
- 6) The `OrderPixelResponse` is returned.

NOTICE: In comparison to the standard `orderPixel()` call the returned pixel objects by this call also include the domain!

NOTICE: If an error occurs in step 3 of the webservice call no pixels are returned but the errorcode in the `OrderPixelResponse` is set to the errorcode responded from the webservice. So the needed count of pixels could not be provided.

If an error occurs in step 4 of the webservice call all pixels read in the steps before are returned AND the errorcode in the `OrderPixelResponse` is set to the errorcode responded from the webservice or to `ERROR_ORDER_PIXEL_STORE_TO_DEPOT` if an technical exception occurred (e.g. server not available).

## 3.5 Example

### 3.5.1 Call OrderPixelService

```
include_once 'lib\logic\OrderPixelService.php';
//include_once 'lib\common\PixelFileDepot.php';

$orderPixelServiceRef = new OrderPixelService();
$count = 2;

try{

    //call webservice
    $orderPixelResponse = $orderPixelServiceRef->orderPixel($count);

    //call webservice with PixelDepot
    //$orderPixelResponse = $orderPixelServiceRef
    //    ->orderPixelUsingDepot($count, new PixelFileDepot());

    //evaluate webservice response
    $errorMsg = $orderPixelResponse->errorMsg;
    $errorCode = $orderPixelResponse->errorCode;
    $domain = $orderPixelResponse->domain;
    $orderDateTime = $orderPixelResponse->orderDateTime;
    $pixels = $orderPixelResponse->pixels;

    //check if error occurred
    if(!isset($errorCode)){

        //achieve pixels
        $pixel = $pixels->pixel;
        foreach ($pixel as $currentPixel){

            //get private and public identification ID
            $currentPublicIdentificationId =
                $currentPixel->publicIdentificationId;
            $currentPrivateIdentificationId =
                $currentPixel->privateIdentificationId;

        }
    }else{

        // handle error

    }

} catch (MetisClientException $exception) {

    //handle MetisClientException
    $exception->errorCode;
    $exception->errorMsg;
} catch (Exception $exception) {

    //handle Exception

}
```

### 3.5.2 Call NewMessageService

NOTICE: There are 4 ways to set Involved (authors and translators). For each of the possibilities one example is shown in the following code. Depending on the existing data of the Involved the PHP-object has to be chosen (see the class diagram in 3.4.2.1 *Involved class hierarchy*).

```
include_once 'lib\model\NewMessageModel.php';
include_once 'lib\logic\NewMessageService.php';

//set privateIdentificationID
$privateIdentificationId = 'abc123abc123abc123abc123abc123abc123';

//set parties (Four ways to set an author or a translator)
$parties = new Parties();
$authors = new Authors();
$translators = new Translators();

//1) set involved by name and code
$involved1 = new InvolvedByNameAndCode();
$involved1->code = 'GMU';
$involved1->firstName = 'Gerhard';
$involved1->surName = 'Muster';

//2) set involved by name and cardnumber
$involved2 = new InvolvedByNameAndCardNumber();
$involved2->cardNumber = '999000';
$involved2->firstName = 'Markus';
$involved2->surName = 'Test';

//3) set involved by name
$involved3 = new InvolvedByName();
$involved3->firstName = 'Test';
$involved3->surName = 'Dummy';

//4) set involved by code
$involved4 = new InvolvedByCode();
$involved4->code = 'XYZ';

$authors->author = array($involved1, $involved2);
$translators->translator = array($involved3, $involved4);
$parties->authors = $authors;
$parties->translators = $translators;

//set messageText
$messageText = new MessageText();
$messageText->lyric = true;
$messageText->shorttext = 'Test Title';
$text = new Text();
$text->plainText = 'article test';
//alternative set a PDF text instead of the plaintext.
//$text->pdf = $pdfText;
$messageText->text = $text;

//set webranges
```

```
$webranges = new Webranges();
$webrange = new Webrange();
$webrange1->url = array('http://dummy.com', 'https://www.dummy2.com');
$webrange2->url = array('http://dummy3.com', 'https://dummy4.com');
$webranges->webrange = array($webrange1, $webrange2);

//fill request
$newMessageRequest = new NewMessageRequest();
$newMessageRequest->parties = $parties;
$newMessageRequest->privateidentificationid = $privateIdentificationId;
$newMessageRequest->messagetext = $messageText;
$newMessageRequest->webranges = $webranges;

try{
    //call the webservice
    $messageServiceRef = new NewMessageService();
    $response = $messageServiceRef->newMessage($newMessageRequest);
    $errorcode = $response->errorcode;
    $errormsg = $response->errormsg;
} catch (MetisClientException $exception){

    //handle MetisClientException
    $exception->errorcode;
    $exception->errormsg;
} catch (Exception $exception){
    //handle Exception
}
}
```

### 3.5.3 Call CheckAuthService

```
include_once 'lib\logic\CheckAuthService.php';

$cardNumber = '0123467';
$surname = 'Muster';

$checkAuthServiceRef = new CheckAuthService();
try{

    //call webservice
    $checkAuthorResponse = $checkAuthServiceRef
        ->checkAuthor($cardNumber, $surname);

    //evaluate webservice response
    $errormsg = $checkAuthorResponse->errormsg;
    $errorcode = $checkAuthorResponse->errorcode;

    if(!isset($errorcode))
    {
        $isValid = $checkAuthorResponse->valid;
    }else{
        // handle error
    }
}
```

```
} catch (MetisClientException $exception) {  
  
    //handle MetisClientException  
    $exception->errorcode;  
    $exception->errmsg;  
  
} catch (Exception $exception) {  
    //handle Exception  
  
}
```

### 3.5.4 Call QualityControlService

```
include_once 'lib\logic\QualityControlService.php';  
  
$qualityControlServiceRef = new QualityControlService();  
  
try{  
    //call webservice  
    $qualityControlResponse = $qualityControlServiceRef  
        ->qualityControl();  
  
    //evaluate webservice response  
    $errmsg = $qualityControlResponse->errmsg;  
    $errorcode = $qualityControlResponse->errorcode;  
  
    if(!isset($errorcode)){  
        $orderedPixelTillToday = $qualityControlResponse  
            ->orderedPixelTillToday;  
        $startedPixelTillToday = $qualityControlResponse  
            ->startedPixelTillToday;  
        $qualityControlValues = $qualityControlResponse  
            ->qualityControlValues;  
  
        foreach ($qualityControlValues as  
            $currentQualityControlValue){  
            $month = $currentQualityControlValue->month;  
            $year = $currentQualityControlValue->year;  
            $orderedPixel = $currentQualityControlValue  
                ->orderedPixel;  
            $startedPixel = $currentQualityControlValue  
                ->startedPixel;  
            $minAccess = $currentQualityControlValue->minAccess;  
            $minAccessNoMessage = $currentQualityControlValue  
                ->minAccessNoMessage;  
            // process statistics here  
        }  
    }else{  
        // handle error  
    }  
  
} catch (MetisClientException $exception) {  
    //handle MetisClientException  
    $exception->errorcode;
```

```
        $exception->errmsg;  
    } catch (Exception $exception) {  
        //handle Exception  
    }  
}
```