OXID eSales  
Documentation

Extending the OXID ERP Interface

Copyright

Copyright © 2018 OXID eSales AG, Germany

Copying of this document or its contents, in particular, using texts or parts of text is subject to the explicit prior permission by OXID eSales AG.

The information provided in this document was prepared according to the current state of the art. OXID eSales AG, however, will assume no liability or warranty for the timeliness, correctness and completeness of the information provided. Since errors – despite all efforts – cannot be ruled out entirely, we always appreciate suggestions.

License

Licensing of the software product depends on the shop edition used.

The software for OXID eShop Community Edition is published under the GNU General Public License v3. You may distribute and/or modify this software according to the licensing terms published by the Free Software Foundation. Legal licensing terms regarding the distribution of software being subject to GNU GPL can be found under http://www.gnu.org/licenses/gpl.html.

The software for OXID eShop Professional Edition and Enterprise Edition is released under commercial license. OXID eSales AG has the sole rights to the software. Decompiling the source code, unauthorized copying as well as distribution to third parties is not permitted. Infringement will be reported to the authorities and prosecuted without exception.

Conventions

The following typographic conventions are used in this document:

Monospace font with grey background

for user inputs, source code and URLs

Italic, grey font

for file names and paths

Bold font

for input fields and navigation steps

Bold, dark red font

for warnings and important notes

Legal Notice

OXID eSales AG

Bertoldstraße 48

79098 Freiburg

Germany

Phone: +49 (761) 36889 0

Fax: +49 (761) 36889 29

Executive board: Roland Fesenmayr (CEO), Dr. Oliver Ciupke

Supervisory board: Michael Schlenk (chairman)

Headquarters: Freiburg

Country court Freiburg i. Brg.

Commercial register number: HRB 701648

Table of contents

[Copyright 2](#_Toc518307476)

[License 2](#_Toc518307477)

[Conventions 3](#_Toc518307478)

[Legal Notice 3](#_Toc518307479)

[Table of contents 4](#_Toc518307480)

[1 Extending the OXID ERP Interface 5](#_Toc518307481)

[2 Extending the OXID ERP Interface (CSV) 5](#_Toc518307482)

[2.1 Register additional export methods 5](#_Toc518307483)

[2.2 Add additional export methods to oxERPCsvGateway 5](#_Toc518307484)

[2.3 Add new methods to oxErpCsv 6](#_Toc518307485)

[2.4 Calling CSV export endpoint 8](#_Toc518307486)

[3 Extending the OXID ERP Interface (SOAP) 9](#_Toc518307487)

[3.1 Add your own import/export methods to ERP SOAP 9](#_Toc518307488)

[3.2 Extend ErpSoapGateway with a module 10](#_Toc518307489)

[3.3 Add handler methods 11](#_Toc518307490)

[3.4 Example for own class extending oxErpType 12](#_Toc518307491)

[4 Extending existing ERP types 12](#_Toc518307492)

# Extending the OXID ERP Interface

With the OXID ERP Interface 3.1 we introduced the possibility to extend the ERP Interface (SOAP and CSV) with own modules.

# Extending the OXID ERP Interface (CSV)

In OXID ERP Interface version 3.1 it is possible to chain extend some of the ERP classes in order to add own functionality to the CSV interface.

## Register additional export methods

To register own export methods for the OXID ERP Interface (CSV) service endpoint (oxerpcsvexport.php) you need to chain extend class oxErpCsvExportMethods with a module.

//module metadata.php  
'extend' => [  
'oxErpCsvExportMethods' => \MyModule\MyErpInterface\Core\ErpCsvExportMethods::class,

<?php

namespace MyModule\MyErpInterface\Core;

class ErpCsvExportMethods extends ErpCsvExportMethods\_parent  
{

/\*\*

\* Custom map

\*

\* @var array

\*/

protected $customMethods = [

'OXERPGetMyErpType' => 'identifier',

'OXERPGetAllMyErpTypes' => 'OXERPGetAllMyErpTypes'

]

## Add additional export methods to oxERPCsvGateway

The custom export methods registered in \MyModule\MyErpInterface\Core\ErpCsvExportMethods need to be introduced into oxERPCsvGateway by chain extending with a module.

//module metadata.php

'extend' => [

'oxERPCsvGateway' => \MyModule\MyErpInterface\Core\ErpCsvGateway::class,

<?php

namespace MyModule\MyErpInterface\Core;

class ErpCsvGateway extends ErpCsvGateway\_parent

{

public function OXERPGetMyErpType($parameters)

{

$where = ''; //enter condition here

return $this->\_export(\MyModule\MyErpInterface\Core\ErpType::class, $where);

}

public function OXERPGetAllMyErpTypes($parameters)

{

$where = ''; //enter condition here

return $this->\_export(\MyModule\MyErpInterface\Core\ErpType::class, $where);

}

For further examples how the internals of those methods might look, check ERP interface code.

## Add new methods to oxErpCsv

New methods can be added to oxErpCsv by chain extending it with a module. The first column in the CSV file is reserved for the so called indicator chars, which are indicating the ERPType of each line during CSV import. When extending with a module keep in mind that other modules might extend the same place. Pick your indicator chars in a way that they do not interfere with others.

* Good example: Use a prefix 'MYM' to mark it for MyModule plus one additional char 'O' to tell ERP that target type is \MyModule\MyErpInterface\Core\ErpType\MyErpType.
* Bad example: Use indicator chars like 'A', 'L', 'K' (meaning any from oxErpCsv::\_aObjects) that are already used by OXID ERP Interface.

//module metadata.php

'extend' => [

'oxErpCsv' => \MyModule\MyErpInterface\Core\ErpCsv::class,

<?php

namespace MyModule\MyErpInterface\Core;

class ErpCsv extends ErpCsv\_parent

{

//Indicator char

protected $myObjects = [

'MYMT' => \MyModule\MyErpInterface\Core\ErpType\MyErpType::class,

];

/\*\*

\* IMPORTANT: merge your indicator char array into oxErpCsv::customObjects

\*/

public function \_\_construct()

{

$this->customObjects = array\_merge($this->customObjects, $this->myObjects);

parent::\_\_construct();

}

/\*\*

\* Import handler

\*

\* @param oxErpType $type type object

\* @param array $row import data

\*

\* @return bool

\*/

protected function \_ImportMyErpType(\oxERPType $type, $row)

{

$result = $this->\_save($type, $row);

return (boolean) $result;

}

/\*\*

\* Delete handler

\*

\* @param oxErpType $type type object

\* @param string $id delete data id

\*

\* @return bool

\*/

protected function \_DeleteMyErpType(\oxERPType $type, $id)

{

$myErpType = $type->getObjectForDeletion($id);

return $type->deleteObject($myErpType, $id);

}

/\*\*

\* Export handler

\*

\* @param array $row data for export

\*

\* @return bool

\*/

protected function \_ExportMyErpType($row)

{

$this->\_writeDsToBuffer("MYMT", $row);

return true;

}

**Note**: Class oxErpCsv already comes with default methods for import (oxErpCsv::\_Import()) and deletion (oxErpCsv::\_Delete()) so you only need to implement specific methods for your ERP types in case special functionality is needed. The export method (\_ExportMyErpType()) needs to be implemented.

## Calling CSV export endpoint

To handle special GET parameters when calling CSV export endpoint <your shop url>/modules/erp/oxerpcsvexport.php you can extend class oxErpCsvGatewayParameters with a module.

//module metadata.php

'extend' => [

'oxErpCsvGatewayParameters' => \MyModule\MyErpInterface\Core\ErpCsvGatewayParameters::class,

<?php

namespace MyModule\MyErpInterface\Core;

class ErpCsvGatewayParameters extends ErpCsvGatewayParameters\_parent

{

/\*\*

\* Extract GET parameters from request.

\*

\* @param string $method requested method

\* @param string $sid session id

\* @param array $aFunctions2Id func<>id map

\*

\* @return stdClass

\*/

public function getParameters($method, $sid, $aFunctions2Id)

{

$parameters = parent::getParameters($method, $sid, $aFunctions2Id);

//Add your own special parameters here

$parameters->myFirstParameter = (isset($\_GET['myFirstParameter])) ? $\_GET['myFirstParameter'] : null;

$parameters->mySecondParameter = (isset($\_GET['mySecondParameter'])) ? $\_GET['mySecondParameter'] : null;

return $parameters;

}

}

# Extending the OXID ERP Interface (SOAP)

From OXID ERP Interface version 3.1 (interface version 2.14.0), the wsdl file is generated on the fly from a given configuration. Own classes extending oxErpType class can come with a module namespace. Class oxERPSoapGatewayConfiguration can easily be extended by a module making it possible to add own SOAP methods. The following documentation will explain how to extend the OXID ERP interface with a module.

## Add your own import/export methods to ERP SOAP

Configuration example that adds the three new SOAP methods OXERPSetMyErpType, OXERPGetMyErpType and OXERPDeleteMyErpType.

<?php

namespace MyModule\MyErpInterface\Core\Configuration;

class ErpSoap extends ErpSoap\_parent

{

protected $myConfiguration = [

'OXERPSetMyErpType' => [

'request' => [

'sSessionID' => ['minOccurs' => '1', 'type' => 's:string'],

'myErpType' => ['minOccurs' => '1', 'type' => 'tns:ArrayOfOXERPType'],

],

'response' => [

'OXERPSetMyErpTypeResult' => ['type' => 'tns:ArrayOfOXERPType'],

]

],

'OXERPGetMyErpType' => [

'request' => [

'sSessionID' => ['minOccurs' => '1', 'type' => 's:string'],

'identifier' => ['minOccurs' => '1', 'type' => 's:string']

],

'response' => [

'OXERPGetMyErpTypeResult' => ['type' => 'tns:OXERPType'],

]

],

'OXERPDeleteMyErpType' => [

'request' => [

'sSessionID' => ['minOccurs' => '1', 'type' => 's:string'],

'identifier' => ['minOccurs' => '1', 'type' => 's:string'],

],

'response' => [

'OXERPDeleteMyErpTypeResult' => ['type' => 'tns:OXERPType'],

]

],

Register this class (\MyModule\MyErpInterface\Core\Configuration\ErpSoap) in the module's ***metadata.php*** so that it chain extends the ERP class named oxerpsoapgatewayconfiguration.

//module metadata.php

'extend' => [

'oxerpsoapgatewayconfiguration' => \MyModule\MyErpInterface\Core\Configuration\ErpSoap::class,

The newly introduced oxERPWsdlGenerator will generate the wsdl from the chain extended configuration, introducing your SOAP methods.

**Important while developing**: Always disable the wsdl cache. Best add the following line into ***bootstrap.php***:

//shop bootstrap.php

ini\_set("soap.wsdl\_cache\_enabled", "0");

## Extend ErpSoapGateway with a module

Now those new methods needs to be introduced in the ErpSoapGateway class by chain extending the original ERP class with a module class.

//module metadata.php

'extend' => [

'oxERPSoapGateway' => \MyModule\MyErpInterface\Core\ErpSoapGateway::class,

With class \MyModule\MyErpInterface\Core\ErpSoapGateway like follows.

<?php

namespace MyModule\MyErpInterface\Core;

class ErpSoapGateway extends ErpSoapGateway\_parent

{

public function OXERPSetMyErpType($parameters)

{

$object = $this->\_formatInput($parameters->myErpType);

return $this->\_import(\MyModule\MyErpInterface\Core\ErpType::class, $object);

}

public function OXERPGetMyErpType($parameters)

{

$where = ''; //enter condition here

return $this->\_export(\MyModule\MyErpInterface\Core\ErpType::class, $where);

}

public function OXERPDeleteMyErpType($parameters)

{

return $this->\_delete(\MyModule\MyErpInterface\Core\ErpType::class, [$parameters->identifier]);

}

## Add handler methods

ERP methods \_import, \_delete, \_export by default call oxErpSoap::\_Import(), oxErpSoap::\_Delete(), oxErpSoap::\_Export() unless your module chain extends oxErpSoap and implements own import, delete and export handling functionality.

//module metadata.php

'extend' => [

'oxErpSoap' => \MyModule\MyErpInterface\Core\ErpSoap::class,

With class \MyModule\MyErpInterface\Core\ErpSoap like follows.

<?php

namespace MyModule\MyErpInterface\Core;

class ErpSoap extends ErpSoap\_parent

{

/\*\*

\* export handler

\*

\* @param array $row data

\*

\* @return boolean

\*/

protected function \_ExportMyErpType($row)

{

//implementation

}

/\*\*

\* delete handler

\*

\* @param oxErpType $type type object

\* @param string $id object id

\*

\* @return boolean

\*/

protected function \_DeleteMyErpType(\oxERPType $type, $id)

{

//implementation

}

For further examples how the internals of those methods might look, check ERP interface code.

## Example for own class extending oxErpType

Here's an example how class MyErpType might look.

**Note:** your type/related database table MUST come with a unique OXID to be able to use the underlying shop objects (\OxidEsales\Eshop\Core\Model\BaseModel).

<?php

namespace MyModule\MyErpInterface\Core\ErpType;

class MyErpType extends \oxERPType

{

const FUNCTION\_SUFFIX = 'MyErpType';

protected $\_sTableName = 'myerptypetable';

protected $\_aFieldList = [

'OXID' => 'OXID',

//field description

];

protected $\_aKeyFieldList = [

'OXUSERID' => 'OXUSERID'

];

/\*\*

\* Getter for \_sFunctionSuffix

\*

\* @return string

\*/

public function getFunctionSuffix()

{

return self::FUNCTION\_SUFFIX;

}

}

# Extending existing ERP types

In case you have custom columns added to some of the standard shop tables, you need to extend the existing ErpTypes.

Short example how to do this in case you added custom columns to the oxuser table will be shown here.

//module metadata.php

'extend' => [

'oxerptype\_user' => \MyModule\MyErpInterface\Core\ErpType\User::class,

Let's assume the additional columns are named MYMCOLUMNA, MYMCOLUMNB and MYMCOLUMNC.

<?php

namespace MyModule\MyErpInterface\Core\ErpType;

class User extends User\_parent

{

protected $additionalFields = [

'MYMCOLUMNA' => 'MYMCOLUMNA',

'MYMCOLUMNB' => 'MYMCOLUMNB',

'MYMCOLUMNC' => 'MYMCOLUMNC',

];

public function \_\_construct()

{

parent::\_\_construct();

$this->\_sFunctionSuffix = 'User';

$this->\_aFieldList = $this->\_aFieldList + $this->additionalFields;

}

}