OXID eSales Documentation

Extending the OXID ERP Interface

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

1 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.

2 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.

2.1 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'
    ]
```

2.2 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.

2.3 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.

2.4 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;
    }
}
```

3 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.

3.1 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");
```

3.2 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]);
    }
```

3.3 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.

3.4 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;
    }
}
```

4 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
<?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;
    }
}
```